МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ПОЛТАВСЬКИЙ ПОЛІТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ
НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ЗАТВЕРДЖУЮ
Заступник директора з НР
_____ **Олена БАБИЧ**
«____» _____2020 р.

НАВЧАЛЬНО-МЕТОДИЧНИЙ КОМПЛЕКС НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Об'єктно-орієнтоване програмування

назва навчальної дисципліни

спеціальність 12 Інформаційні технології

шифр і назва спеціальності

спеціалізація 121 Інженерія програмного забезпечення

назва спеціалізації

відділення Комп'ютерних технологій

назва відділення

розробник Олександр БАБИЧ

ім'я та прізвище викладача

Програма розглянута й затверджена на засіданні циклової комісії
дисциплін Програмної інженерії

назва циклової комісії

Протокол від «____» _____20___ року № _____
Голова циклової комісії _____ Олександр БАБИЧ

підпис *ім'я та прізвище*

ПОЛТАВА – 2021

АНОТАЦІЯ ЗМІСТУ НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Об’єктно-орієнтоване програмування

назва навчальної дисципліни

нормативна

нормативна/вибіркова

цикл професійної та практичної підготовки

цикл дисциплін за навчальним планом

Об’єктно-орієнтоване програмування є теоретичною та практичною основою професійних знань та вмінь, що формують профіль фахівця в предметній області «Інженерія програмного забезпечення». Тож НМК навчальної дисципліни посідає одне з чільних місць в підготовці фахівців з спеціальності.

Предмет навчальної дисципліни

Предметом вивчення навчальної дисципліни є методи та засоби проектування та розробки складних об’єктно-орієнтованих програмних систем з використанням платформи JAVA.

Мета навчальної дисципліни

Метою викладання навчальної дисципліни «ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ» є дати здобувачами освіти уявлення, початкові знання та практичні навички, які відповідають сучасним тенденціям в галузі проектування та розробки складних програмних систем на прикладі платформи JAVA.

Основні завдання

Основними завданнями вивчення дисципліни «ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ» є:

- ознайомити з принципами, методами і засобами об’єктно-орієнтованого програмування;
- дати уявлення про платформу та мову програмування JAVA;
- ознайомити з шаблонами проектування та типовими архітектурними рішеннями (патернами), які добре себе зарекомендували під час проектування програмних систем;
- ознайомити з основами документування проектних рішень;
- сформувати практичні навички створення програмних систем за допомогою мови JAVA та IDE NetBeans.

Місце навчальної дисципліни в структурно-логічній схемі

Дисципліна «ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ» тісно взаємопов'язана з наступними навчальними дисциплінами:

- Основи програмної інженерії
- Конструювання програмного забезпечення
- Алгоритми та структури даних
- Основи програмування та алгоритмічні мови

Вимоги до знань і вмінь

Згідно з вимогами освітньо-професійної програми навчальної дисципліни здобувачі освіти повинні **знати**:

- основні поняття та методи об'єктно-орієнтованого аналізу та проектування;
- особливості платформи JAVA та синтаксис мови програмування JAVA;
- основні підходи до проектування сучасних програмних систем;
- основні методи забезпечення якості програмних продуктів, види тестування;

вміти:

- застосовувати ключові концепції об'єктно-орієнтованого програмування, такі як інкапсуляція, успадкування та поліморфізм;
- виявляти об'єкти предметної області, розробляти діаграми класів для проблемної області,
- створювати ієрархію класів, засновану на діаграмах класів;
- проектувати додатки за допомогою UML-діаграм, виконувати кодогенерацію та зворотне проектування;
- розробляти і тестувати JAVA-застосунки, компілювати в байт-код та запускати JAVA-програми на виконання;
- виконувати рефакторинг існуючого коду, налагоджувати, тестувати і профілювати свої проекти;
- документувати свої додатки;
- створювати графічні JAVA-застосунки з використанням відповідних компонентів Swing API;
- реалізувати введення/виведення для роботи з файловою системою;
- створювати багатопоточні JAVA-застосунки;
- використовувати сокети для організації взаємодії в реальному часі за протоколом TCP/IP;
- розробляти додатки для платформи Netbeans.

ЗМІСТ

ПРОГРАМА НАВЧАЛЬНОЇ ДИСЦИПЛІН

Об’єктно-орієнтоване програмування

назва навчальної дисципліни

нормативна

нормативна/вибіркова

цикл професійної та практичної підготовки

цикл дисциплін за навчальним планом

РОЗРОБНИК(-И) ПРОГРАМИ: викладач-методист, викладач вищої кваліфікаційної категорії, к.т.н., Бабич О.В.

Програма розглянута на засіданні циклової комісії дисциплін програмної інженерії

Протокол від « 26 » травня 2020 року № 8

Голова циклової комісії _____ Олександр БАБИЧ

Програма схвалена на засіданні циклової комісії дисциплін програмної інженерії

Протокол від « 26 » травня 2020 року № 8

Голова циклової комісії _____ Олександр БАБИЧ

ВСТУП

Програма вивчення нормативної навчальної дисципліни «Об’єктно-орієнтоване програмування» складена відповідно до освітньо-професійної програми підготовки молодшого спеціаліста спеціалізації: Розробка програмного забезпечення.

Предметом вивчення навчальної дисципліни є методи та засоби проектування та розробки складних об’єктно-орієнтованих програмних систем з використанням платформи JAVA.

Міждисциплінарні зв'язки:

- основи програмної інженерії
- конструювання програмного забезпечення
- алгоритми та структури даних
- основи програмування та алгоритмічні мови

Програма навчальної дисципліни складається з таких змістових модулів:

Змістовий модуль 1. Основні поняття ООП. Об’єктно-орієнтований аналіз та проектування. Огляд технології JAVA.

Змістовий модуль 2. Складні типи даних. Основні алгоритмічні конструкції. Методи.

Змістовий модуль 3. Інкапсуляція. Успадкування та повторне використання коду.

Змістовий модуль 4. Розробка об’єктно-орієнтованих програм мовою JAVA та їх документування.

Змістовий модуль 5. Основні можливості платформи JAVA.

Змістовий модуль 6. Проектування та реалізація архітектури програм. Забезпечення якості програмних продуктів.

Змістовий модуль 7. Консоль. Файлова система. Графічний Інтерфейс користувача.

Змістовий модуль 8. Багатопоточність. Мережеві можливості. Платформа Netbeans.

1. МЕТА ТА ЗАВДАННЯ НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Метою викладання навчальної дисципліни «ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ» є дати здобувачами освіти уявлення, початкові знання та практичні навички, які відповідають сучасним тенденціям в галузі проектування та розробки складних програмних систем на прикладі платформи JAVA.

Основними завданнями вивчення дисципліни «ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ» є:

- ознайомити з принципами, методами і засобами об'єктно-орієнтованого програмування;
- дати уявлення про платформу та мову програмування JAVA;
- ознайомити з шаблонами проектування та типовими архітектурними рішеннями (паттернами), які добре себе зарекомендували під час проектування програмних систем;
- ознайомити з основами документування проектних рішень;
- сформувати практичні навички створення програмних систем за допомогою мови JAVA та IDE NetBeans.

Згідно з вимогами освітньо-професійної програми здобувачі освіти повинні:

знати:

- основні поняття та методи об'єктно-орієнтованого аналізу та проектування;
- особливості платформи JAVA та синтаксис мови програмування JAVA;
- основні підходи до проектування сучасних програмних систем;
- основні методи забезпечення якості програмних продуктів, види тестування.

вміти:

- застосовувати ключові концепції об'єктно-орієнтованого програмування, такі як інкапсуляція, успадкування та поліморфізм;
- виявляти об'єкти предметної області, розробляти діаграми класів для проблемної області, створювати ієрархію класів, засновану на діаграмах класів;
- проектувати додатки за допомогою UML-діаграм, виконувати кодогенерацію та зворотне проектування;
- розробляти і тестувати JAVA-застосунки, компілювати в байт-код та запускати JAVA-програми на виконання;

- виконувати рефакторинг існуючого коду, налагоджувати, тестувати і профілювати свої проекти
- документувати свої додатки;
- створювати графічні JAVA-застосунки з використанням відповідних компонентів Swing API;
- реалізувати введення/виведення для роботи з файловою системою;
- створювати багатопоточні JAVA-застосунки;
- використовувати сокети для організації взаємодії в реальному часі за протоколом TCP/IP;
- розробляти додатки для платформи Netbeans.

Сформовані компетенції

- ціннісно-змістовна (свідома організація власної діяльності у складі команди проектувальників ПЗ)
- інформаційна (вміння знаходити потрібну інформацію користуючись усіма доступними джерелами і обмінюватись нею з іншими членами команди)
- комунікативна (здатність до вдосконалення навичок командної роботи, вміння працювати на результат, доводити власну думку, вести діалог)

Форма підсумкового контролю успішності навчання:

☐ залік
☑ екзамен

На вивчення навчальної дисципліни відводиться 150 годин 5 кредитів ECTS.

2. ІНФОРМАЦІЙНИЙ ОБСЯГ НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Змістовий модуль 1. Основні поняття ООП. Об'єктно-орієнтований аналіз та проектування. Огляд технології JAVA (5 семестр)

Тема 1.1. Поняття про об'єкти та класи

Тема 1.2. Взаємодія об'єктів

Тема 1.3. Об'єктно-орієнтований аналіз та проектування з використанням UML. Аналіз проблеми та розробка алгоритму її вирішення

Тема 1.4. Огляд технології JAVA. Розробка та тестування JAVA-програми. Декларування, ініціалізація та використання змінних

Тема 1.5. Підготовка середовища виконання та розробки JAVA-застосунків. JRE, SDK та JDK

Тема 1.6. Інші технології подвійного компілювання. Microsoft .NET

Тема 1.7. Підведення підсумків. Підсумкове тестування

Змістовий модуль 2. Складні типи даних. Основні алгоритмічні конструкції. Методи (6 семестр)

Тема 2.1. Створення та використання об'єктів

Тема 2.2. Використання операторів та алгоритмічних конструкцій. Використання циклів

Тема 2.3. Розробка та використання методів

Змістовий модуль 3. Інкапсуляція. Успадкування та повторне використання коду

Тема 3.1. Інкапсуляція та конструктори

Тема 3.2. Створення та використання масивів

Тема 3.3. Реалізація успадкування

Змістовий модуль 4. Розробка об'єктно-орієнтованих програм мовою JAVA та їх документування

Тема 4.1. JAVA як платформа. IDE NetBeans

Тема 4.2. Інструменти продуктивності розробника в NetBeans

Тема 4.3. Об’єктно-орієнтоване програмування. Документування програм

Змістовий модуль 5. Основні можливості платформи JAVA

Тема 5.1. Нові ідентифікатори, ключові слова та типи даних

Тема 5.2. Поглиблене використання виразів, керування виконанням програми

Тема 5.3. Поглиблене використання масивів

Змістовий модуль 6. Проектування та реалізація архітектури програм. Забезпечення якості програмних продуктів

Тема 6.1. Проектування ієрархії класів. Використання UML. Особливості створення класів

Тема 6.2. Рефакторинг. Типові архітектурні рішення та антипатерни

Тема 6.3. Обробка помилок та виключень. Відлагодження, тестування та профілювання

Тема 6.4. Колекції та дженерики

Змістовий модуль 7. Консоль. Файлова система. Графічний Інтерфейс користувача

Тема 7.1. Основи вводу-виводу. Робота с консоллю та файловою системою

Тема 7.2. Робота з базами даних. JDBC. Entity Classes.

Тема 7.3. Створення графічного інтерфейсу користувача

Тема 7.4. Обробка подів від інтерфейсних елементів

Тема 7.5. Тонке налагодження інтерфейсу користувача

Змістовий модуль 8. Багатопоточність. Мережеві можливості. Платформа Netbeans

Тема 8.1. Багатопоточність в JAVA

Тема 8.2. Робота з мережею

Тема 8.3. Контрольна робота

Тема 8.4. Створення та використання веб-сервісів. REST

Тема 8.5. JAVA-аплети та JSP

Тема 8.6. Розробка додатків для платформи NetBeans

Тема 8.7. Розповсюдження додатків для платформи NetBeans. Інсталятори. Bootstrap’ери. Портативні додатки

Тема 8.8. Підведення підсумків. Підсумкове тестування

3. РОЗПОДІЛ ГОДИН ЗА ВИДАМИ ЗАНЯТЬ ВІДПОВІДНО ДО НАВЧАЛЬНОГО ПЛАНУ

(тематичний план)

№ заняття	Назва теми	Кількість годин				
		Лекції	Семінарські та	Лабораторні роботи	Самостійне вивчення	Всього
Змістовий модуль 1. Основні поняття ООП. Об'єктно-орієнтований аналіз та проектування. Огляд технології JAVA (5 семестр)						
1/2	Тема 1.1. Поняття про об'єкти та класи	2	2			4
3/4	Тема 1.2. Взаємодія об'єктів	2	2			4
5	Тема 1.3. Об'єктно-орієнтований аналіз та проектування з використанням UML. Аналіз проблеми та розробка алгоритму її вирішення		2			2
6/7	Тема 1.4. Огляд технології JAVA. Розробка та тестування JAVA-програми. Декларування, ініціалізація та використання змінних	2	2			4
	Тема 1.5. Підготовка середовища виконання та розробки JAVA-застосунків. JRE, SDK та JDK				6	6
	Тема 1.6. Інші технології подвійного компілювання. Microsoft .NET				6	6
8	Тема 1.7. Підведення підсумків. Підсумкове тестування	2				2
	Разом за змістовим модулем 1	**8**	**8**		**12**	**28**
Змістовий модуль 2. Складні типи даних. основні алгоритмічні конструкції. методи (6-семестр)						
9	Тема 2.1. Створення та використання об'єктів	2				2
10/11/12	Тема 2.2. Використання операторів та алгоритмічних конструкцій. Використання циклів	4	2			6
13/14	Тема 2.3. Розробка та використання методів	2	2			4
	Разом за змістовим модулем 2	**8**	**4**			**12**

№ заняття	Назва теми	Кількість годин				
		Лекції	Семінарські та	Лабораторні роботи	Самостійне вивчення	Всього
Змістовий модуль 3. Інкапсуляція. Успадкування та повторне використання коду						
15/16	Тема 3.1. Інкапсуляція та конструктори	2	2			4
17/18	Тема 3.2. Створення та використання масивів	2	2			4
19/20	Тема 3.3. Реалізація успадкування	2	2		4	8
	Разом за змістовим модулем 3	**6**	**6**		**4**	**16**
Змістовий модуль 4. Розробка об'єктно-орієнтованих програм мовою JAVA та їх документування						
21	Тема 4.1. JAVA як платформа. IDE NetBeans	2				2
	Тема 4.2. Інструменти продуктивності розробника в NetBeans				4	4
22/23/24	Тема 4.3. Об'єктно-орієнтоване програмування. Документування програм	4	2			6
	Разом за змістовим модулем 4	**6**	**2**		**4**	**12**
Змістовий модуль 5. Основні можливості платформи JAVA						
25/26	Тема 5.1. Нові ідентифікатори, ключові слова та типи даних	2	2			4
27/28/29	Тема 5.2. Поглиблене використання виразів, керування виконанням програми	4	2			6
30/31	Тема 5.3. Поглиблене використання масивів	2	2			4
	Разом за змістовим модулем 5	**8**	**6**			**14**
Змістовий модуль 6. Проектування та реалізація архітектури програм. Забезпечення якості програмних продуктів						
32/33	Тема 6.1. Проектування ієрархії класів. Використання UML. Особливості створення класів	4				4

№ заняття	Назва теми	Кількість годин				
		Лекції	Семінарські та	Лабораторні роботи	Самостійне вивчення	Всього
34/35/36	Тема 6.2. Рефакторинг. Типові архітектурні рішення та антипатерни	4	2			6
37/38	Тема 6.3. Обробка помилок та виключень. Відлагодження, тестування та профілювання	2	2		4	8
39/40	Тема 6.4. Колекції та дженерики	2	2			4
	Разом за змістовим модулем 6	**12**	**6**		**4**	**22**
Змістовий модуль 7. Консоль. Файлова система. Графічний Інтерфейс користувача						
41/42/43	Тема 7.1. Основи вводу-виводу. Робота с консоллю та файловою системою	4	2			6
44	Тема 7.2. Робота з базами даних. JDBC. Entity Classes.	2				2
45	Тема 7.3. Створення графічного інтерфейсу користувача	2				2
46/47	Тема 7.4. Обробка подій від інтерфейсних елементів	2	2			4
48/49	Тема 7.5. Тонке налагодження інтерфейсу користувача	4				4
	Разом за змістовим модулем 7	**14**	**4**			**18**
Змістовий модуль 8. Багатопоточність. Мережеві можливості. Платформа Netbeans						
50	Тема 8.1. Багатопоточність в JAVA	2				2
51/52	Тема 8.2. Робота з мережею	2	2			4
53	Тема 8.3. Контрольна робота	2				2
	Тема 8.4. Створення та використання веб-сервісів. REST				6	6
	Тема 8.5. JAVA-аплети та JSP				6	6
54	Тема 8.6. Розробка додатків для платформи NetBeans	2				2

№ заняття	Назва теми	Кількість годин				
		Лекції	Семінарські та	Лабораторні роботи	Самостійне вивчення	Всього
55	Тема 8.7. Розповсюдження додатків для платформи NetBeans. Інсталятори. Bootstrap’ери. Портативні додатки	2			2	4
56	Тема 8.8. Підведення підсумків. Підсумкове тестування	2				2
	Разом за змістовим модулем 8	**12**	**2**		**14**	**28**
	Разом за семестр	**66**	**30**		**26**	**122**
	Всього	**74**	**38**		**38**	**150**

4. РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Основна

1. Яків Файн. Програмування мовою Java для дітей, батьків, дідусів та бабусь (2015) – http://myflex.org/books/java4kids/JavaKid_ua.zip
2. Копитко М.Ф., Іванків К.С. Основи програмування мовою Java: Тексти лекцій. – Львів: Видавничий центр ЛНУ ім. Івана Франка, 2002. – 83 с.
3. Брнакевич І.Є., Вагін П.П. Програмування мовою Java: використання фундаментальних класів: Тексти лекцій. – Львів: Видавничий центр ЛНУ імені Івана Франка, 2002. – 75 с.
4. Вступ до алгоритмів Томас Г. Кормен, Чарлз Е. Лейзерсон, Роналд Л. Рівест і Кліфорд Стайн. Переклад з англійської (третього видання). К.І.С.,2019 - 1288 стор.
5. Кунгурцев О.Б. Основи програмування на мові Java. Середовище Net Beans. Навчальний посібник-Одеса: ВМВ, 2006. -183с.
6. Будай А. Дизайн-патерни - просто, як двері (2012) [PDF] - https://toloka.to/t34059

Додаткова

1. Арнольд К., Гослинг Д.- Язык программирования Java СПб.: Питер, 2002.– 250 с.
2. Дейтел Х.М., Дейтел П.Дж., Сантри С.И. - Технологии программирования на Java. Том 1. Графика. Москва.: Бином-пресс, 2003. – 560с.
3. Дейтел Х.М., Дейтел П.Дж., Сантри С.И. - Технологии программирования на Java. Том 2. Распределенные приложения. Москва.: Бином-пресс, 2003. – 464с.
4. Дейтел Х.М., Дейтел П.Дж., Сантри С.И. - Технологии программирования на Java. Том 3. Корпоративные системы, сервлеты, JSP, Web-сервисы. Москва.: Бином-пресс, 2003. – 672с.
5. Хабибулин И.Ш. Создание распределенных приложений на Java 2. – СПб.: БХВ-Петербург, 2002.– 701 с.
6. Дэвид Флэнэген. Java in a Nutshell.- O'Reilly & Associates, Inc., 1997, Издательская группа BHV, Киев, 1998
7. Чен М.С. и др.Программирование на JAVA:1001 совет: Наиболее полное руководство по Java и Visual J++:Пер.с англ./Чен М.С.,Грифис С.В.,Изи Э.Ф..- Минск:Попурри,1997.-640с.ил.+ Прил.(1диск).
8. Майкл Эферган. Java: справочник.- QUE Corporation, 1997, Издательство "Питер Ком", 1998

9. Ноутон П., Шилдт Г.- Java2. Наиболее полное руководство СПб.: БХВ-Петербург, 2006.– 1034 с.
10. Скотт К. Java для студента – СПб.: БХВ-Петербург, 2007.– 448 с.
11. Флэнаген Д. Java в примерах: Справочник.-2е издание. Спб.: Символ-Плюс, 2003. – 664 с.
12. Нотон П. JAVA: Справ.руководство: Пер.с англ./Под ред.А.Тихонова. –М.: БИНОМ: Восточ. Кн. Компания, 1996: Восточ. Кн. Компания. – 447с.
13. Патрик Нотон, Герберт Шилдт Полный справочник по Java.- McGraw-Hill,1997, Издательство "Диалектика",1997

Інформаційні ресурси

1. Віртуальна академія. Програмування на Java: https://college.page.link/6FWN
2. Освоюємо Java: Вікіпідручник: https://uk.wikibooks.org/wiki/Освоюємо_Java
3. Курс "Основы программирования на языке Java (уровень II)": https://sites.google.com/site/javafund0/
4. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
5. Язык программирования Java и среда NetBeans: http://www.intuit.ru/studies/courses/569/425/info
6. Программирование на Java: http://www.intuit.ru/studies/courses/16/16/info
7. Построение распределенных систем на Java: http://www.intuit.ru/studies/courses/633/489/info
8. Углубленное программирование на Java: http://www.intuit.ru/studies/courses/3711/953/info

РОБОЧА ПРОГРАМА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Робоча програма дисципліни «ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ» для здобувачів освіти 3 курсу
спеціальності: 121 Інженерія програмного забезпечення
спеціалізації: Розробка програмного забезпечення

«___» серпня 2020 року

Розробник (-и): викладач-методист, викладач вищої кваліфікаційної категорії, к.т.н., Ол-др.В. Бабич
Робоча програма розглянута на засіданні циклової комісії «Дисциплін програмної інженерії »

Протокол від «____» серпня 2020 року № 1

Голова циклової комісії _____ Ол-др.В. Бабич

Робоча програма схвалена на засіданні циклової комісії «Дисциплін програмної інженерії »

Протокол від «____» серпня 2020 року № 1

Голова циклової комісії _____ Ол-др.В. Бабич

1. ОПИС НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

<table>
<tr><th colspan="6">Найменування показників</th><th>Галузь знань, спеціальність,спеціалізація, освітньо-кваліфікаційний рівень</th><th>Характеристика навчальної дисципліни</th></tr>
<tr><td colspan="6">Кількість кредитів
ECTS – 5</td><td rowspan="2">галузь знань:
12 Інформаційні технології</td><td rowspan="2">Форма навчання:
☑ Денна
☐ Заочна</td></tr>
<tr><td colspan="6">Модулів – 2</td></tr>
<tr><td colspan="6">Змістових модулів – 8</td><td rowspan="2">спеціальність:
121 Інженерія програмного забезпечення</td><td>Вид дисципліни
☑ Нормативна
☐ Вибіркова</td></tr>
<tr><td colspan="6" rowspan="2">Загальна кількість – 150 год.
аудиторних – 112 год.
самостійної роботи – 38 год.</td><td>Вид контролю:
☑ Іспит
☐ Диференційований залік</td></tr>
<tr><td rowspan="11">спеціалізація:
Розробка програмного забезпечення</td><td rowspan="11"></td></tr>
<tr><td>Семестр</td><td>Лекції</td><td>ПР/Семінарські</td><td>ЛР</td><td>Самостійна робота</td></tr>
<tr><td>I</td><td></td><td></td><td></td><td></td></tr>
<tr><td>II</td><td></td><td></td><td></td><td></td></tr>
<tr><td>III</td><td></td><td></td><td></td><td></td><td rowspan="7">Освітньо-кваліфікаційний рівень:молодший спеціаліст</td></tr>
<tr><td>IV</td><td></td><td></td><td></td><td></td></tr>
<tr><td>V</td><td>8</td><td>8</td><td></td><td>12</td></tr>
<tr><td>VI</td><td>66</td><td>30</td><td></td><td>26</td></tr>
<tr><td>VII</td><td></td><td></td><td></td><td></td></tr>
<tr><td>VIII</td><td></td><td></td><td></td><td></td></tr>
</table>

2. МЕТА ТА ЗАВДАННЯ НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Метою викладання навчальної дисципліни «ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ» є дати здобувачами освіти уявлення, початкові знання та практичні навички, які відповідають сучасним тенденціям в галузі проектування та розробки складних програмних систем на прикладі платформи JAVA.

Основними завданнями вивчення дисципліни «ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ» є:

- ознайомити з принципами, методами і засобами об'єктно-орієнтованого програмування;
- дати уявлення про платформу та мову програмування JAVA;
- ознайомити з шаблонами проектування та типовими архітектурними рішеннями (паттернами), які добре себе зарекомендували під час проектування програмних систем;
- ознайомити з основами документування проектних рішень;
- сформувати практичні навички створення програмних систем за допомогою мови JAVA та IDE NetBeans.

Згідно з вимогами освітньо-професійної програми здобувачі освіти повинні:
знати:

- основні поняття та методи об'єктно-орієнтованого аналізу та проектування;
- особливості платформи JAVA та синтаксис мови програмування JAVA;
- основні підходи до проектування сучасних програмних систем;
- основні методи забезпечення якості програмних продуктів, види тестування.

вміти:

- застосовувати ключові концепції об'єктно-орієнтованого програмування, такі як інкапсуляція, успадкування та поліморфізм;
- виявляти об'єкти предметної області, розробляти діаграми класів для проблемної області, створювати ієрархію класів, засновану на діаграмах класів;
- проектувати додатки за допомогою UML-діаграм, виконувати кодогенерацію та зворотне проектування;

- розробляти і тестувати JAVA-застосунки, компілювати в байт-код та запускати JAVA-програми на виконання;
- виконувати рефакторинг існуючого коду, налагоджувати, тестувати і профілювати свої проекти
- документувати свої додатки;
- створювати графічні JAVA-застосунки з використанням відповідних компонентів Swing API;
- реалізувати введення/виведення для роботи з файловою системою;
- створювати багатопоточні JAVA-застосунки;
- використовувати сокети для організації взаємодії в реальному часі за протоколом TCP/IP;
- розробляти додатки для платформи Netbeans.

3. КРИТЕРІЇ ОЦІНЮВАННЯ

Оцінювання знань, вмінь та навичок здобувачів освіти здійснюється за:

☑ Чотирьохбальною шкалою
☐ Дванадцятибальною шкалою

Оцінювання знань, вмінь та навичок здобувачів освіти за чотирьохбальною шкалою:

5 «Відмінно» – здобувач освіти володіє матеріалом та навичками цілісно комплексного аналізу теми. Самостійно оцінює, знаходить, виправляє допущені помилки, працює з різними джерелами інформації, систематизує та творчо використовує дібраний матеріал. Тісно пов'язує теорію з практикою і правильно демонструє знання та виконання практичних навичок. Вирішує задачі та завдання підвищеної складності, вміє узагальнювати матеріал, володіє методами дослідницької роботи. Виконує заплановану індивідуальну роботу (підготовка презентацій, дослідницьких робіт, рефератів, складання таблиць, схем відповідно до теми, тощо). Здобувач освіти володіє матеріалом, справляється із завданнями на 90-100%.

4 «Добре» – здобувач освіти добирає переконливі аргументи на підтвердження висловленого судження або рішення, правильно і по суті відповідає на стандартизовані питання поточної теми, включно із питаннями лекційного курсу та самостійної роботи. Демонструє знання та виконання практичних навичок. Правильно використовує теоретичні знання для вирішення практичних задач. Вміє вирішувати задачі та завдання середньої складності. Володіє необхідними практичними навиками і прийомами їх виконання в обсязі, який перевищує необхідний мінімум. Здобувач освіти володіє матеріалом, справляється із завданнями на 68-89%.

3 «Задовільно» – здобувач освіти відтворює матеріал. Має неглибокі знання, але добре аналізує основні елементи теми. Наводить окремі власні приклади на підтвердження певних суджень, відповідає на стандартизовані питання поточної теми, лекційного курсу та самостійної роботи. Не може самостійно побудувати чітку, логічну відповідь. Під час відповіді і демонстрації практичних навичок допускає помилки. Вирішує лише найлегші ситуаційні задачі, не в повному обсязі володіє обов'язковим мінімумом методів дослідження. Здобувач освіти володіє матеріалом, справляється із завданнями на 50-67%.

2 «Незадовільно» – здобувач освіти не відповідає на елементарні запитання, не може побудувати логічну відповідь, не відповідає на додаткові питання, не розуміє змісту матеріалу. Під час відповіді і демонстрації практичних навичок робить значні, грубі помилки. Здобувач освіти досить поверхнево володіє матеріалом, справляється із завданнями менше, ніж на 50%.

4. ІНФОРМАЦІЙНИЙ ОБСЯГ НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Змістовий модуль 1. Основні поняття ООП. Об'єктно-орієнтований аналіз та проектування. Огляд технології JAVA (5 семестр)

Тема 1.1. Поняття про об'єкти та класи

Тема 1.2. Взаємодія об'єктів

Тема 1.3. Об'єктно-орієнтований аналіз та проектування з використанням UML. Аналіз проблеми та розробка алгоритму її вирішення

Тема 1.4. Огляд технології JAVA. Розробка та тестування JAVA-програми. Декларування, ініціалізація та використання змінних

Тема 1.5. Підготовка середовища виконання та розробки JAVA-застосунків. JRE, SDK та JDK

Тема 1.6. Інші технології подвійного компілювання. Microsoft .NET

Тема 1.7. Підведення підсумків. Підсумкове тестування

Змістовий модуль 2. Складні типи даних. Основні алгоритмічні конструкції. Методи (6 семестр)

Тема 2.1. Створення та використання об'єктів

Тема 2.2. Використання операторів та алгоритмічних конструкцій. Використання циклів

Тема 2.3. Розробка та використання методів

Змістовий модуль 3. Інкапсуляція. Успадкування та повторне використання коду

Тема 3.1. Інкапсуляція та конструктори

Тема 3.2. Створення та використання масивів

Тема 3.3. Реалізація успадкування

Змістовий модуль 4. Розробка об'єктно-орієнтованих програм мовою JAVA та їх документування

Тема 4.1. JAVA як платформа. IDE NetBeans

Тема 4.2. Інструменти продуктивності розробника в NetBeans

Тема 4.3. Об’єктно-орієнтоване програмування. Документування програм

Змістовий модуль 5. Основні можливості платформи JAVA

Тема 5.1. Нові ідентифікатори, ключові слова та типи даних

Тема 5.2. Поглиблене використання виразів, керування виконанням програми

Тема 5.3. Поглиблене використання масивів

Змістовий модуль 6. Проектування та реалізація архітектури програм. Забезпечення якості програмних продуктів

Тема 6.1. Проектування ієрархії класів. Використання UML. Особливості створення класів

Тема 6.2. Рефакторинг. Типові архітектурні рішення та антипатерни

Тема 6.3. Обробка помилок та виключень. Відлагодження, тестування та профілювання

Тема 6.4. Колекції та дженерики

Змістовий модуль 7. Консоль. Файлова система. Графічний Інтерфейс користувача

Тема 7.1. Основи вводу-виводу. Робота с консоллю та файловою системою

Тема 7.2. Робота з базами даних. JDBC. Entity Classes.

Тема 7.3. Створення графічного інтерфейсу користувача

Тема 7.4. Обробка подів від інтерфейсних елементів

Тема 7.5. Тонке налагодження інтерфейсу користувача

Змістовий модуль 8. Багатопоточність. Мережеві можливості. Платформа Netbeans

Тема 8.1. Багатопоточність в JAVA

Тема 8.2. Робота з мережею

Тема 8.3. Контрольна робота

Тема 8.4. Створення та використання веб-сервісів. REST

Тема 8.5. JAVA-аплети та JSP

Тема 8.6. Розробка додатків для платформи NetBeans

Тема 8.7. Розповсюдження додатків для платформи NetBeans. Інсталятори. Bootstrap’ери. Портативні додатки

Тема 8.8. Підведення підсумків. Підсумкове тестування.

5. РОЗПОДІЛ ГОДИН ЗА ВИДАМИ ЗАНЯТЬ ВІДПОВІДНО ДО РОБОЧОГО НАВЧАЛЬНОГО ПЛАНУ

№ заняття	Назва теми	Кількість годин				
		Лекції	Семінарські та практичні	Лабораторні роботи	Самостійне вивчення	Всього
Змістовий модуль 1. Основні поняття ООП. Об'єктно-орієнтований аналіз та проектування. Огляд технології JAVA (5 семестр)						
1/2	Тема 1.1. Поняття про об'єкти та класи	2	2			4
3/4	Тема 1.2. Взаємодія об'єктів	2	2			4
5	Тема 1.3. Об'єктно-орієнтований аналіз та проектування з використанням UML. Аналіз проблеми та розробка алгоритму її вирішення		2			2
6/7	Тема 1.4. Огляд технології JAVA. Розробка та тестування JAVA-програми. Декларування, ініціалізація та використання змінних	2	2			4
	Тема 1.5. Підготовка середовища виконання та розробки JAVA-застосунків. JRE, SDK та JDK				6	6
	Тема 1.6. Інші технології подвійного компілювання. Microsoft .NET				6	6
8	Тема 1.7. Підведення підсумків. Підсумкове тестування	2				2
	Разом за змістовим модулем 1	**8**	**8**		**12**	**28**
Змістовий модуль 2. Складні типи даних. основні алгоритмічні конструкції. методи (6-семестр)						
9	Тема 2.1. Створення та використання об'єктів	2				2
10/11/12	Тема 2.2. Використання операторів та алгоритмічних конструкцій. Використання циклів	4	2			6
13/14	Тема 2.3. Розробка та використання методів	2	2			4
	Разом за змістовим модулем 2	**8**	**4**			**12**
Змістовий модуль 3. Інкапсуляція. Успадкування та повторне використання коду						

№ заняття	Назва теми	Кількість годин				
		Лекції	Семінарські та практичні	Лабораторні роботи	Самостійне вивчення	Всього
15/16	Тема 3.1. Інкапсуляція та конструктори	2	2			4
17/18	Тема 3.2. Створення та використання масивів	2	2			4
19/20	Тема 3.3. Реалізація успадкування	2	2		4	8
	Разом за змістовим модулем 3	**6**	**6**		**4**	**16**
Змістовий модуль 4. Розробка об'єктно-орієнтованих програм мовою JAVA та їх документування						
21	Тема 4.1. JAVA як платформа. IDE NetBeans	2				2
	Тема 4.2. Інструменти продуктивності розробника в NetBeans				4	4
22/23/24	Тема 4.3. Об'єктно-орієнтоване програмування. Документування програм	4	2			6
	Разом за змістовим модулем 4	**6**	**2**		**4**	**12**
Змістовий модуль 5. Основні можливості платформи JAVA						
25/26	Тема 5.1. Нові ідентифікатори, ключові слова та типи даних	2	2			4
27/28/29	Тема 5.2. Поглиблене використання виразів, керування виконанням програми	4	2			6
30/31	Тема 5.3. Поглиблене використання масивів	2	2			4
	Разом за змістовим модулем 5	**8**	**6**			**14**
Змістовий модуль 6. Проектування та реалізація архітектури програм. Забезпечення якості програмних продуктів						
32/33	Тема 6.1. Проектування ієрархії класів. Використання UML. Особливості створення класів	4				4
34/35/36	Тема 6.2. Рефакторинг. Типові архітектурні рішення та антипатерни	4	2			6
37/38	Тема 6.3. Обробка помилок та виключень. Відлагодження, тестування та профілювання	2	2		4	8
39/40	Тема 6.4. Колекції та дженерики	2	2			4

№ заняття	Назва теми	Кількість годин				
		Лекції	Семінарські та практичні	Лабораторні роботи	Самостійне вивчення	Всього
	Разом за змістовим модулем 6	**12**	**6**		**4**	**22**
Змістовий модуль 7. Консоль. Файлова система. Графічний Інтерфейс користувача						
41/42/43	Тема 7.1. Основи вводу-виводу. Робота с консоллю та файловою системою	4	2			6
44	Тема 7.2. Робота з базами даних. JDBC. Entity Classes.	2				2
45	Тема 7.3. Створення графічного інтерфейсу користувача	2				2
46/47	Тема 7.4. Обробка подій від інтерфейсних елементів	2	2			4
48/49	Тема 7.5. Тонке налагодження інтерфейсу користувача	4				4
	Разом за змістовим модулем 7	**14**	**4**			**18**
Змістовий модуль 8. Багатопоточність. Мережеві можливості. Платформа Netbeans						
50	Тема 8.1. Багатопоточність в JAVA	2				2
51/52	Тема 8.2. Робота з мережею	2	2			4
53	Тема 8.3. Контрольна робота	2				2
	Тема 8.4. Створення та використання веб-сервісів. REST				6	6
	Тема 8.5. JAVA-аплети та JSP				6	6
54	Тема 8.6. Розробка додатків для платформи NetBeans	2				2
55	Тема 8.7. Розповсюдження додатків для платформи NetBeans. Інсталятори. Bootstrap’ери. Портативні додатки	2			2	4
56	Тема 8.8. Підведення підсумків. Підсумкове тестування	2				2
	Разом за змістовим модулем 8	**12**	**2**		**14**	**28**
	Разом за семестр	**66**	**30**		**26**	**122**
	Всього	**74**	**38**		**38**	**150**

6. МЕТОДИ НАВЧАННЯ

Словесні методи навчання

☑ Розповіді
☑ Пояснення
☑ Дискусія
☑ Лекції
☑ Лекції з використанням мультимедійних презентацій
☑ Бесіди

Методи навчання за джерелами знань

☑ Метод роботи з книгою
☑ Питання
☐ Переказ
☐ Виписування
☐ Складання плану
☑ Складання таблиць, схем та ін.

Наочні методі навчання

☑ Демонстрації
☑ Ілюстрації
☐ Спостереження

Практичні методи навчання

☐ Лабораторна робота
☑ Практична робота
☑ Виконання вправ
☑ Самостійна робота
☑ Проблемно-пошукові роботи
☑ Розгляд творчо-аналітичних завдань
☑ Робота в групах
☐ Розв’язування ситуаційних завдань

7. ЗАСОБИ ОЦІНЮВАННЯ (МЕТОДИ КОНТРОЛЮ)

Засоби оцінювання:

☑ Екзамен
☐ Диференційований залік
☑ Підсумковий тест
☐ Курсова робота
☐ Курсовий проект
☑ Командний проект
☑ Комплект звітів про виконання лабораторних /практичних робіт
☑ Реферат, есе
☐ Розрахункова / розрахунково-графічна робота
☑ Презентація результатів виконаних завдань та досліджень
☐ Виконання завдання на лабораторному обладнанні
☐ _____

Методи контролю:

Поточний:

☑ Усне опитування
☑ Фронтальне
☑ Групове
☑ Індивідуальне
☑ Комбіноване
☐ _____

Письмовий контроль:

☑ Самостійна робота
☑ Тести
☑ Вправи
☐ Диктанти
☑ Реферати
☑ Презентації
☑ Звіти

☐ _____

Періодичний контроль:

☑ Підсумкові письмові контрольні роботи за змістовими модулями
☑ Тематичні роботи
☐ _____

Підсумковий контроль:

☑ Екзамен
☐ Диференційований залік
☐ Захист курсового проекту
☐ Захист курсової роботи
☐ Захист розрахунково-графічної роботи
☐ _____

8. МЕТОДИЧНЕ ЗАБЕЗПЕЧЕННЯ

☑ Конспект лекцій
☑ Презентації лекцій
☑ Роздатковий матеріал
☑ Таблиці, схеми, ментальні карти
☑ Електронний підручник з дисципліни
☑ Програмне забезпечення навчального призначення
☑ Пакет завдань для контрольної роботи
☑ Пакет завдань для самостійної роботи
☑ Комплект тестових завдань
☑ Комплексна контрольна робота
☑ Екзаменаційні білети
Методичні вказівки до
- ☐ Семінарських занять
- ☐ Практичних занять
- ☐ Лабораторних занять
- ☐ Самостійної та індивідуальної роботи

☐ _____

9. РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Основна

1. Яків Файн. Програмування мовою Java для дітей, батьків, дідусів та бабусь (2015) – http://myflex.org/books/java4kids/JavaKid_ua.zip
2. Копитко М.Ф., Іванків К.С. Основи програмування мовою Java: Тексти лекцій. – Львів: Видавничий центр ЛНУ ім. Івана Франка, 2002. – 83 с.
3. Брнакевич І.Є., Вагін П.П. Програмування мовою Java: використання фундаментальних класів: Тексти лекцій. – Львів: Видавничий центр ЛНУ імені Івана Франка, 2002. – 75 с.
4. Вступ до алгоритмів Томас Г. Кормен, Чарлз Е. Лейзерсон, Роналд Л. Рівест і Кліфорд Стайн
5. Переклад з англійської (третього видання). К.І.С.,2019 - 1288 стор.
6. Кунгурцев О.Б. Основи програмування на мові Java. Середовище Net Beans. Навчальний посібник-Одеса: ВМВ, 2006. -183с.
7. Будай А. Дизайн-патерни - просто, як двері (2012) [PDF] - https://toloka.to/t34059
8. Електронний підручник з дисципліни. Бабич О.В. © 2018.

Додаткова

1. Арнольд К., Гослинг Д.- Язык программирования Java СПб.: Питер, 2002.– 250 с.
2. Дейтел Х.М., Дейтел П.Дж., Сантри С.И. - Технологии программирования на Java. Том 1. Графика. Москва.: Бином-пресс, 2003. – 560с.
3. Дейтел Х.М., Дейтел П.Дж., Сантри С.И. - Технологии программирования на Java. Том 2. Распределенные приложения. Москва.: Бином-пресс, 2003. – 464с.
4. Дейтел Х.М., Дейтел П.Дж., Сантри С.И. - Технологии программирования на Java. Том 3. Корпоративные системы, сервлеты, JSP, Web-сервисы. Москва.: Бином-пресс, 2003. – 672с.
5. Хабибулин И.Ш. Создание распределенных приложений на Java 2. – СПб.: БХВ-Петербург, 2002.– 701 с.
6. Дэвид Флэнэген. Java in a Nutshell.- O'Reilly & Associates, Inc., 1997, Издательская группа BHV, Киев, 1998
7. Чен М.С. и др.Программирование на JAVA:1001 совет: Наиболее полное руководство по Java и Visual J++:Пер.с англ./Чен М.С.,Грифис С.В.,Изи Э.Ф..-Минск:Попурри,1997.-640с.ил.+ Прил.(1диск).

8. Майкл Эферган. Java: справочник.- QUE Corporation, 1997, Издательство "Питер Ком", 1998
9. Ноутон П., Шилдт Г.- Java2. Наиболее полное руководство СПб.: БХВ-Петербург, 2006.– 1034 с.
10. Скотт К. Java для студента – СПб.: БХВ-Петербург, 2007.– 448 с.
11. Флэнаген Д. Java в примерах: Справочник.-2е издание. Спб.: Символ-Плюс, 2003. – 664 с.
12. Нотон П. JAVA: Справ.руководство: Пер.с англ./Под ред.А.Тихонова. –М.: БИНОМ: Восточ. Кн. Компания, 1996: Восточ. Кн. Компания. – 447с.
13. Патрик Нотон, Герберт Шилдт Полный справочник по Java.- McGraw-Hill,1997, Издательство "Диалектика",1997

Інформаційні ресурси

1. Віртуальна академія. Програмування на Java: https://college.page.link/6FWN
2. Освоюємо Java: Вікіпідручник: https://uk.wikibooks.org/wiki/Освоюємо_Java
3. Курс "Основы программирования на языке Java (уровень II)": https://sites.google.com/site/javafund0/
4. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
5. Язык программирования Java и среда NetBeans: http://www.intuit.ru/studies/courses/569/425/info
6. Программирование на Java: http://www.intuit.ru/studies/courses/16/16/info
7. Построение распределенных систем на Java: http://www.intuit.ru/studies/courses/633/489/info
8. Углубленное программирование на Java: http://www.intuit.ru/studies/courses/3711/953/info

10. КОРЕКЦІЯ

Навчальний рік 2020-2021

Аудиторні год. (за семестр)	Зняття	До виконання	Корекція заняття (об'єднання тем, винесення на самостійне опрацювання)	Підпис

Навчальний рік 2021-2022

Аудиторні год. (за семестр)	Зняття	До виконання	Корекція заняття (об'єднання тем, винесення на самостійне опрацювання)	Підпис

Навчальний рік 2022-2023

Аудиторні год. (за семестр)	Зняття	До виконання	Корекція заняття (об'єднання тем, винесення на самостійне опрацювання)	Підпис

ПЛАНИ-КОНСПЕКТИ ЛЕКЦІЙНИХ ЗАНЯТЬ

Змістовий модуль 1. Основні поняття ООП. Об'єктно-орієнтований аналіз та проектування. Огляд технології JAVA (5 семестр)

Тема 1.1. **Поняття про об'єкти та класи**
Номер заняття: 1

Мета: ознайомити студентів з ключовими поняттями об'єктно-орієнтованого програмування; продемонструвати переваги об'єктно-орієнтованого підходу як найадекватнішого способу відображення оточуючої реальності.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/SKXq

частина 2: https://college.page.link/dxud

частина 3: https://college.page.link/VK9X

Короткі теоретичні відомості:

Об'єктно-орієнтований дизайн являє собою процес планування системи взаємодіючих об'єктів для вирішення програмних проблем. Це один з підходів до розробки програмного забезпечення. Вхідними даними для ООД є вихідні дані об'єктно-орієнтованого аналізу. Виходом є послідовність діаграм чи діаграма класів.

Об'єкт являє собою єдине ціле даних та методів. *Інтерфейс об'єкта* являє собою набір визначень про те, яким чином об'єкт взаємодіє з зовнішніми об'єктами, сутностями тощо. Об'єкти та їхні інтерфейси визначаються під час проведення аналізу. З точки зору бізнесу, об'єктно-орієнтований дизайн розробляє систему об'єктів, кожний з яких виконує свою справу. Наприклад, для певної компанії, бізнес-об'єкти можуть містити персонал, вироби, устаткування, транспортні засоби тощо.

Об'єктно-орієнтоване програмування – це метод програмування, заснований на поданні програми у вигляді сукупності взаємодіючих об'єктів, кожен з яких є екземпляром певного класу, а класи є членами певної ієрархії наслідування.

Програмісти спочатку пишуть клас, а на його основі при виконанні програми створюються конкретні об'єкти (екземпляри класів). На основі класів можна створювати нові, які розширюють базовий клас і таким чином створюється *ієрархія класів*.

На думку Алана Кея, розробника мови Smalltalk, якого вважають одним з «батьків-засновників» ООП, об'єктно-орієнтований підхід полягає в наступному наборі *основних принципів*:

- Все є *об'єктами*.
- Всі дії та розрахунки виконуються шляхом взаємодії (обміну даними) між об'єктами, при якій один об'єкт потребує, щоб інший об'єкт виконав деяку дію. Об'єкти взаємодіють, надсилаючи і отримуючи повідомлення.
- *Повідомлення* – це запит на виконання дії, доповнений набором аргументів, які можуть знадобитися при виконанні дії.
- Кожен об'єкт має незалежну пам'ять, яка складається з інших об'єктів.
- Кожен об'єкт є представником (*екземпляром*, примірником) класу, який виражає загальні властивості об'єктів.
- У класі задається *поведінка* (функціональність) об'єкта. Таким чином усі об'єкти, які є екземплярами одного класу, можуть виконувати одні й ті ж самі дії.
- Класи організовані у єдину деревоподібну структуру з загальним корінням, яка називається *ієрархією успадкування*. Пам'ять та поведінка, зв'язані з екземплярами деякого класу, автоматично доступні будь-якому класу, розташованому нижче в ієрархічному дереві.

Таким чином, програма являє собою набір об'єктів, що мають *стан та поведінку*. Об'єкти взаємодіють використовуючи повідомлення. Будується *ієрархія об'єктів*: програма в цілому – це об'єкт, для виконання своїх функцій вона звертається до об'єктів що містяться у ньому, які у свою чергу виконують запит шляхом звернення до інших об'єктів програми. Звісно, щоб уникнути безкінечної рекурсії у зверненнях, на якомусь

етапі об’єкт трансформує запит у повідомлення до стандартних системних об’єктів, що даються мовою та середовищем програмування. Стійкість та керованість системи забезпечуються за рахунок чіткого розподілення відповідальності об’єктів (за кожну дію відповідає певний об’єкт), однозначного означення інтерфейсів міжоб’єктної взаємодії та повної ізольованості внутрішньої структури об’єкта від зовнішнього середовища (*інкапсуляції*).

В результаті дослідження Дебори Дж. Армстронг комп’ютерної літератури, що була видана протягом останніх 40 років, вдалось відокремити фундаментальні поняття (принципи), використані у переважній більшості визначень об’єктно-орієнтованого програмування. До них належить:

Клас. Клас визначає абстрактні характеристики деякої сутності, включаючи характеристики самої сутності (її атрибути або властивості) та дії, які вона здатна виконувати (її поведінки, методи або можливості). Наприклад, клас Собака може характеризуватись рисами, притаманними всім собакам, зокрема: порода, колір хутра, здатність гавкати. Класи вносять *модульність та структурованість* в об’єктно-орієнтовану програму. Як правило, клас має бути зрозумілим для не-програмістів, що знаються на предметній області, що, у свою чергу, значить, що клас повинен мати значення в контексті. Також, код реалізації класу має бути досить самодостатнім. Властивості та методи класу, разом називаються його членами.

Об’єкт. Окремий екземпляр класу (створюється після запуску програми і ініціалізації полів класу). Клас Собака відповідає всім собакам шляхом опису їхніх спільних рис; об’єкт Сірко є одним окремим собакою, окремим варіантом значень характеристик. Собака має хутро; Сірко має коричнево-біле хутро. Характеристика об’єкта може бути іншим об’єктом. Об’єкт Сірко є екземпляром (примірником) класу Собака. Сукупність значень атрибутів окремого об’єкта називається станом. На основі класу Собака можна, також, створити інший об’єкт Дружок, який відрізнятиметься від об’єкта Сірко своїм станом (наприклад кольором хутра). Обидва об’єкта (Сірко і Дружок) є екземплярами класу Собака.

Метод. Можливості об’єкта. Оскільки Сірко – Собака, він може гавкати. Тому гавкати() є одним із методів об’єкта Сірко. Він може мати й інші методи, зокрема: місце(), або їсти(). В межах програми, використання методу має впливати лише на один об’єкт; всі Собаки можуть гавкати, але треба щоб гавкав лише один окремий собака.

Обмін повідомленнями. «Передача даних від одного процесу іншому, або надсилання викликів методів.»

Успадкування (наслідування). Клас може мати «підкласи», спеціалізовані, розширені версії надкласу. Можуть навіть утворюватись цілі дерева успадкування. Наприклад, клас Собака може мати підкласи Коллі, Пекінес, Вівчарка і т.п. Так, Сірко може бути екземпляром класу Вівчарка. Підкласи успадковують атрибути та поведінку своїх батьківських класів, і можуть вводити свої власні. Успадкування може бути одиничне (один безпосередній батьківський клас) та множинне (кілька батьківських класів). Це залежить від вибору програміста, який реалізовує клас та мови програмування. Так, наприклад, в Java дозволене лише одинарне успадкування, а в C++ і те і інше.

Приховування інформації (*інкапсуляція*). Приховування деталей про роботу класів від об'єктів, що їх використовують або надсилають їм повідомлення. Так, наприклад, клас Собака має метод гавкати(). Реалізація цього методу описує як саме повинно відбуватись гавкання (приміром, спочатку вдихнути() а потім видихнути() на обраній частоті та гучності). Петро, хазяїн пса Сірка, не повинен знати як він гавкає. Інкапсуляція досягається шляхом вказування, які класи можуть звертатися до членів об'єкта. Як наслідок, кожен об'єкт представляє кожному іншому класу певний інтерфейс – члени, доступні іншим класам. Інкапсуляція потрібна для того, аби запобігти використанню користувачами інтерфейсу тих частин реалізації, які, швидше за все, будуть змінюватись. Це дозволить полегшити внесення змін, тобто, без потреби змінювати і користувачів інтерфейсу. Наприклад, інтерфейс може гарантувати, що щенята можуть додаватись лише до об'єктів класу Собака кодом самого класу. Часто, члени класу позначаються як публічні (англ. public), захищені (англ. protected) та приватні (англ. private), визначаючи, чи доступні вони всім класам, підкласам, або лише до класу в якому їх визначено. Деякі мови програмування йдуть ще далі: Java використовує ключове слово private для обмеження доступу, що буде дозволений лише з методів того самого класу, protected – лише з методів того самого класу і його нащадків та з класів із того ж самого пакету, C# та VB.NET відкривають деякі члени лише для класів із тієї ж збірки шляхом використання ключового слова internal (C#) або Friend (VB.NET), а Eiffel дозволяє вказувати які класи мають доступ до будь-яких членів.

Абстрагування. Спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми. Наприклад Собака Сірко більшу частину часу може розглядатись як Собака, а коли потрібно отримати доступ до інформації специфічної для собак породи коллі – як Коллі і як Тварина (можливо, батьківський клас Собака) при підрахунку тварин.

Поліморфізм. Поліморфізм означає залежність поведінки від класу, в якому ця поведінка викликається, тобто, два або більше класів можуть реагувати по-різному на однакові повідомлення. Наприклад, якщо Собака отримує команду голос(), то у відповідь можна отримати Гав; якщо Свиня отримує команду голос (), то у відповідь можна отримати Рох-рох. На практиці це реалізовується шляхом реалізації ряду підпрограм (функцій, процедур, методи тощо) з однаковими іменами, але з різними параметрами. В залежності від того, що передається і вибирається відповідна підпрограма.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 1.2. **Взаємодія об’єктів**
Номер заняття: 3

Мета: ознайомити студентів з поняттям об’єкту; сформувати уявлення про програму як про сукупність об’єктів різних класів, які взаємодіють один з одним, продемонструвати переваги такого підходу як найадекватнішого способу відображення оточуючої реальності.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв’язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/vPBJ

частина 2: https://college.page.link/sLQA

Короткі теоретичні відомості:

Об'єкт – конкретний екземпляр класу. Наприклад, зелена "Тойота" вашого сусіда є екземпляром класу "Автомобіль". По суті, це клас, поля якого ініціалізовані і він завантажений у пам'ять комп'ютера. На основі одного класу, можна створити безліч об'єктів.

Дані в об’єкті (тобто глобальні змінні даного об’єкту) називаються *полями* (атрибутами) екземпляру класу. Процедури, які оперують даними називаються *методами* (операціями). Специфічний об’єкт, який є екземпляром певного класу матиме специфічні значення екземплярних полів. Сукупність цих значень називається *станом об’єкта*. Коли викликається певний метод стан об’єкту може змінюватись. Згідно принципу інкапсуляції намагаються, щоб доступ до приватних полів екземпляру здійснювався лише за допомогою відповідних публічних методів.

Майже весь код, написаний мовою Java – це класи та об'єкти, тому що Java – об'єктно-орієнтована мова. Java-об'єкти зберігають свій стан у змінних, ці змінні ще називають *полями* або членами екземпляру класу.

Об'єкт - це деяка сутність у віртуальному просторі, що має певний стан і поведінку, задані значення властивостей (атрибутів) та операцій над ними (методів).

Об’єкт – це *конкретна реалізація* певного класу. На основі одного класу може бути створено безліч об’єктів. При цьому в об’єктах виділяють:

- *Поведінку* об'єкту – що можна з робити з даним об'єктом, або які методи можна застосовувати до нього
- *Стан* об'єкту – те як об'єкт змінюється, коли Ви застосовуєте його методи
- *Ідентичність* об'єкту – відмінність об'єкту від інших об'єктів. Об'єкти можуть мати однаковий стан, проте все рівно вони ідентифікуються як різні об'єкти.

Об'єкт може існувати самостійно. До цього об'єкту можна ставити запитання за допомогою методів і ці самі методи можу давити відповідь, повертаючи значення.

Щоб створити об’єкт певного класу програміст викликає один з *конструкторів* даного класу. Це спеціальні методи, які покликані задати об'єкту початковий стан і носять його ім'я.

new Date(); //створення об'єкту, який містить поточний час

Проте такий підхід застосовується якщо об’єкт потрібен нам в програмі лише раз. Якщо ж його потрібно буде використати повторно, то для цього використовуються *об'єктні змінні* –змінні, які посилаються на певний об’єкт.

Date curDate = new Date();

Отже, процес створення об'єктів складається з трьох частин:

- *Декларація* – процес «зв'язування» змінної з типом об'єкта - *Date curDate*
- *Створення примірника* – ключове слово *new* створює об'єкт - *new*
- *Ініціалізація* – після ключового слова *new* йде виклик *конструктора* який ініціалізує об'єкт – *Date();*

Посилання - це вказівник на щось. Існує три види типів посилань:

- *класовий* тип *SomeClassseRef*
- *інтерфейсний* тип *SomeInterface someInterfaceRef*

– тип *масив int[] someArrayRef*

Декларація типу посилання не створює об'єкт!

Оператор *new* створює екземпляр об'єкта, *виділяючи під нього пам'ять*. Цей оператор має лише один постфіксний аргумент – виклик конструктора, і повертає посилання на створений об'єкт. Необов'язково присвоювати змінній посилання на створений об'єкт:

```
int height = (new Rectangle ()).getHeight();
```

Посилання на об'єкти, також передаються в методи за значенням. Це означає, що коли метод завершить свою роботу, передане посилання як і раніше буде вказувати на той же об'єкт, що і вказувало до виклику методу. Проте, *значення полів об'єкта можуть бути змінені* в методі, якщо вони мають відповідний рівень доступу.

Наприклад, маємо клас виду

```
class Demo{
    public int a;
}
```

та метод, що використовує посилання на нього

```
class Test{
    void someVoid(Demo d){
        d.a = 20;
        d = new Demo();
        d.a =40;
    }
}
```

Після виконання наступного коду:

```
Demo demo = new Demo();
Test test = new Test();
test.someVoid(demo);
System.out.println(demo.a)
```

Матимемо на екрані **20**;

При цьому в пам'яті JVM буде *два* об'єкти типу *Demo*, на один з яких відсутнє посилання!

В Java розробник класів може гарантувати ініціалізацію кожного об'єкта, забезпечивши спеціальний метод, що називають *конструктором*. Якщо клас має конструктор, Java автоматично викликає конструктор, коли створюється об'єкт.

Якщо конструктор явно не створити, то все одно він буде створений *за замовчуванням*.

Перевантажений конструктор викликається в залежності від параметрів, зазначених при виконанні операції *new*.

Операції з посиланнями на об'єкти:

- *Доступ до поля*, з використанням імені або виразу
- Виклик *методу*
- Операція *приведення типу*
- *Конкатенація посилання з рядком* (буде використано метод *toString()*)
- Операція *instanceof*
- Операції *порівняння* == і *! =*
- *Тернарний* (умовний) операто *? :*

Присвоювання посилань:

Box b1 = new Box(); Box b2 = b1;

b1 = null ;

b2 = null;

b1 = new Box();

null – це спеціальний літерал, який може бути будь-якого типу, змінну такого типу створити не можна.

Використання об'єктів:

- *Доступ до полів*
  - всередині класу доступ до полів відбувається *за іменем* поля
  - за межами класу доступ до полів відбувається з використанням *посилання на об'єкт* і оператора крапка (.) Наприклад, *objectReference.fieldName*
- *Виклик методів*
  - всередині класу – *за іменем* методу

– за межами класу доступ до полів відбувається з використанням *посилання на об'єкт* і оператора крапка (.) Наприклад, *objectReference. methodName(args)*

Об’єкти взаємодіють шляхом *виклику методів* один одного.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t
5. Java. Алгоритмізація та програмування / Об’єкти – SchoolboyProgrammer: https://schoolboyprog10.blogspot.com/p/blog-page_89.html

Тема 1.4. **Огляд технології JAVA. Розробка та тестування JAVA-програми. Декларування, ініціалізація та використання змінних**
Номер заняття: 6

Мета: ознайомити студентів з платформою JAVA; сформувати уявлення про процес розробки ПЗ; дати поняття про структуру JAVA-програм та використання змінних.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/SKXq

частина 2: https://college.page.link/dxud

частина 3: https://college.page.link/VK9X

Короткі теоретичні відомості:

Що таке Java? Створюючи програми більшістю мов програмування, треба визначити, *в якій операційній системі і на якому процесорі* вони працюватимуть. Тільки визначивши це, можна долучити до програми виклик функцій з бібліотеки, призначеної для відповідної операційної системи. Наприклад: якщо розробляють програму для Windows, то можна використати бібліотеку Microsoft Foundation Classes; для роботи на платформі Macintosh − функції з Mac OS Toolbox. Після компіляції вихідних текстів отримуємо код, готовий до виконання *на певному процесорі*.

Створюючи програми на Java, можна не замислюватися над тим, в якій операційній системі вони працюватимуть. Java має *власний набір машинно-незалежних бібліотек*, які називають пакетами. Щодо процесорів ситуація аналогічна. Компілятор Java *не генерує безпосередньо інструкції процесору*. Він створює проміжний код − *байткод* для

віртуальної машини Java (Java Virtual Machine − JVM). Виходячи з того, що ядро віртуальної машини Java реалізовано практично для всіх типів комп'ютерів, вважатимемо файли байткодів незалежними від платформи.

Започатковано мову Java у проекті фірми Sun Microsystems під назвою Green (1991). Головними розробниками першої робочої версії були *Джеймс Гослінг* (James Gosling), *Патрік Ноутон* (Patric Naughton), *Кріс Ворс* (Chris Warth), *Ед Френк* (Ed Frank) і *Майкл Шерідан* (Mike Sheridan). До 1995 року мову називали *Оак*, однак не пройшовши перевірку на допустимість торгової марки, було перейменовано на Java. Ідеєю створення нової мови програмування були не потреби Internet, а необхідність створення програмного забезпечення, яке не залежить від платформи (тобто архітектури) і використовується в *побутових електронних приладах*. Під час відпрацювання деталей Java виник вагомий чинник, який відіграв важливу роль щодо цієї мови. Таким чинником була всесвітня інформаційна служба World Wide Web (WWW), розквіт якої припадає на 1994−1995 р.

Ідеї створення ефективних і незалежних (працюючих на різноманітних процесорах під керівництвом різних операційних систем) програм такі ж давні, як і саме програмування, і завжди витіснялися нагальнішими проблемами. Якщо зважити на те, що більшість програмістів належить до одного з трьох кланів (Windows, Mac OS, UNIX/Linux), які безупинно змагаються між собою, то зрозумілим стає те, що *нагальної потреби в переміщенні коду довший час не виникало*. Проте з виникненням *Internet і WWW* проблема переміщення програм стала втричі гострішою. Різко змінилися акценти: від створення коду для вбудованих контролерів побутової техніки до *програмування для Internet*. Це і спричинило великий успіх мови Java.

Після виникнення в 1979 р. мови C++, до якої було додано порівняно з мовою C об'єктно-орієнтовані засоби і збережено усі сильні сторони C, складається враження, що програмісти знайшли нарешті досконалу мову. Проте навіть могутня і популярна мова програмування має свої *недоліки*. Це важкі щодо розуміння і використання аспекти C++, пов'язані з керуванням пам'яттю і покажчиками. У C++ також легко зберігається процедурний стиль програмування. Згідно з думкою П. Нортона, *наступним кроком у логічному розвитку мов програмування є Java*. Ця мова опирається на найпопулярнішу мову C++ (розробники Java зробили це свідомо), проте цілковито *відкидає поняття процедурного програмування і змушує підкорятися принципам об'єктно-орієнтованого програмування* (ООП). У мові Java *відсутні покажчики, а керування пам'яттю відбувається автоматично*.

Зважаючи на подібність мов Java та C++, дехто вважає, що Java − це "Internet-версія C++". Насправді це не так. Незважаючи на значний вплив C++ на створення Java, щодо них не існує ніякої сумісності (ні зверху вниз, ні знизу вверх). Крім того, цю мову не розроблено з метою заміни C++. Мову C++ застосовують для вирішення одних проблем, мову Java − для інших. Найважливіша з них − *створення програм, які не залежать від платформ виконання*.

Розробники Java створювали мову з метою втілення наступних *базових принципів*:

- простота;
- безпека;
- перенесення;
- об'єктно-орієнтована направленість;
- стійкість щодо помилок;
- багатопоточність;
- незалежність від архітектури;
- інтерпретація;
- висока продуктивність;
- розподіленість;
- динамічність.

Простота, інтерпретація, перенесення, незалежність від архітектури. До простих мов програмування належать ті, які працюють з *інтерпретатором* (наприклад, Бейсік). Перші персональні комп'ютери поставлялися з інтерпретатором Бейсіка. Сьогодні їхнє місце займають HTML і мови сценаріїв для Web. Вивчати і використовувати мови програмування, які компілюються, набагато складніше, ніж ті, які інтерпретуються. Тобто є мови програмування для професіоналів. Наприклад, C++, де використання покажчиків і керування пам'яттю є складними не тільки для початківців, але й для досвідчених програмістів. Один рядок програми, який звертається до недозволеного місця в пам'яті, може спричинити до збоїв не тільки програми, але й комп'ютера загалом.

Java − це мова, програми якою *компілюються та інтерпретуються*, і водночас вона має просту структуру мови високого рівня. Написана програма компілюється в проміжну форму − *байткод*. Пізніше ця програма виконується, тобто інтерпретується виконавчим (runtime) середовищем Java. Байткод дуже відрізняється від машинного коду, який є послідовністю нулів та одиничок. *Байткод − це набір інструкцій, які подібні до команд Асемблера*. Машинний код комп'ютер виконує безпосередньо, а байткод потрібно інтерпретувати. Тому машинний код можна використати тільки на конкретній

платформі, для якої його скомпільовано. *Байткод можна виконувати на будь-якій платформі, на якій встановлено виконавче середовище Java*. Саме ця можливість і робить програми на Java незалежними від архітектури. Оскільки байткод є проміжною формою програми, то його інтерпретація вимагає незначних витрат. Байткод створено для машини, яка реально не існує. Цю машину називають *віртуальною Java-машиною* (JVM), вона існує тільки в пам'яті комп'ютера. Створення компілятором Java байткоду для неіснуючої машини − це тільки половина процесу, який забезпечує незалежність від архітектури. Другу частину виконує *інтерпретатор Java*, який виконує роль посередника між віртуальною Java-машиною та реальним комп'ютером.

Архітектура мови для розподіленого мережевого середовища. Головною вимогою щодо мови для роботи в розподіленому просторі комп'ютерів (наприклад, в Internet) – це можливість працювати на різнорідних і розподілених платформах. Мова Java є пристосованою до перенесення завдяки підтримці стандартів IEEE для структур даних, наприклад, цілих чисел, чисел з плаваючою комою і рядків. До мови Java зачислено безпосередньо підтримку таких розповсюджених протоколів як FTP, HTTP, що забезпечує сумісність під час роботи в мережі. Java забезпечує розподілену роботу за допомогою механізму *виклику віддалених методів* (RMI), тобто дає змогу використовувати об'єкти, розташовані на локальних і віддалених машинах.

Багатопоточність. У багатопоточних операційних системах для кожного застосування (процесу) надається *окрема захищена область пам'яті*, в якій зберігаються коди програми і дані. А час одного процесора *квантується* між цими процесами. З метою запуску процесу або переключення з одного на інший на рівні операційної системи необхідно виконати значний об'єм роботи. Тому для розробників прикладних програм спеціально створили "полегшену" версію системного процесу − *потік*. Найбільшою проблемою, пов'язаною з процесами і потоками, є їхнє функціонування під керуванням конкретної операційної системи. Спеціалісти компанії Sun зробили потоки частиною мови програмування. Тому багатопоточне застосування, написане мовою Java, працюватиме і в операційних середовищах Windows, Unix, Mac OS.

Висока продуктивність. Інтерпретатор Java може виконувати байткоди зі швидкістю, яка наближається до швидкості виконання коду, відкомпільованого до машинного формату, що досягається завдяки використання інтерпретатором *багатьох потоків виконання*. Наприклад, доки комп'ютер чекає на введення даних, фонові потоки можуть зайнятися очищенням пам'яті.

Стійкість до помилок. Java − це мова строгого використання типів, що зумовлює зменшення числа помилок під час написання програми. У мові Java відбувається

автоматична перевірка виконання граничних умов під час роботи з масивами і рядками, які в Java є класами. В Java *немає арифметики покажчиків*, а *керування пам'яттю здійснюється автоматично*. Програмний код, написаний мовою Java не може зіслатися на пам'ять поза простором програми або зробити помилку внаслідок вивільнення пам'яті і тим самим вичерпати всю пам'ять. Зазначимо, що у Java організовано процес *автоматичного збирання сміття*, тобто об'єктів, на які більше ніхто не вказує.

Безпека. Функції забезпечення безпеки дуже важливі для розподілених мереж з безліччю вірусів, "троянських коней" і т. п. Для реалізації цієї мети розробники мови Java створили механізм, який отримав назву *пісочниці* (sandbox):

- перевірку на рівні JVM;
- захист на рівні мови;
- інтерфейс Java Security (цифрового підпису).

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. *Усунуто можливість появи деяких конфліктних ситуацій*, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім, Java розроблялась як *платформо-незалежна мова*, тому вона має *менше низькорівневих можливостей* для роботи з апаратним забезпеченням. За необхідності таких дій java дозволяє викликати підпрограми, написані іншими мовами програмування.

Java вплинула на розвиток *J++*, що розроблялась компанією Microsoft. Роботу над J++ було зупинено через судовий позов компанії Sun Microsystems, оскільки ця мова програмування була модифікацією Java. Пізніше в новій платформі Microsoft .NET випустило J#, щоб полегшити міграцію програмістів J++ або Java на нову платформу. З часом нова мова програмування *C#*, стала основною мовою платформи, перейнявши багато чого з Java. J# востаннє включався в версію Microsoft Visual Studio 2005. Мова сценаріїв JavaScript має схожу із Java назву і синтаксис, але не пов'язана із Java.

Java дозволяє створювати *самодостатні програми для різних операційних систем* як то Windows, Linux тощо. Крім того, в даний час Java широко застосовується для *програмування різних пристроїв*, наприклад, мобільних телефонів, на ній також пишуться комп'ютерні ігри для них, створюють також програми для інтернету - Аплети (вважаються небезпечними тож більше не підтримуються веб-оглядачами) і програми для серверів - Сервлети та JSP (Java Server Pages).

Дещо про випуски Java та їх найменування. *19 березня 2020 року вийшла вже 14 версія Java*. Назви випусків подавалися розробниками Java дещо по-різному, що створило в літературі деяку плутанину. Спочатку писали JDK 1.0, JDK 1.1 (JDK - Java Development Kit). Після появи версії 1.2 у випусках java почали в назві писали *Java2SE*, наприклад, J2SE 1.4 (SE - це Standart Editon, крім того додатково йдуть випуски EE – Enterprise Edition, ME – Mobile Edition). П'яту ж версію почали писати J2SE 5.0, а далі викинули 2 з назви і почали писати Java SE 6. Різкі зміни в найменуванні передусім пояснювалися ґрунтовними переробками в Java. За історію Java найбільш серйозні зміни були внесені у версію 1.2 відповідно це пояснює появу 2-ки в назвах випусків. Наступний перегляд був у версії 1.5. В Java SE 8 одним з основних нововведень стали лямда-вирази.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t
5. Копитко М.Ф., Іванків К.С. Основи програмування мовою Java: Тексти лекцій. − Львів: Видавничий центр ЛНУ ім. Івана Франка, 2002.− 83 с.

Змістовий модуль 2. Складні типи даних. Основні алгоритмічні конструкції. Методи (6 семестр)

Тема 2.1. **Створення та використання об'єктів**
Номер заняття: 9

Мета: ознайомити студентів з поняттям об'єкту; пояснити як створюються об'єкти та як вони використовуються для реалізації поставлених завдань.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/vPBJ

частина 2: https://college.page.link/sLQA

Короткі теоретичні відомості:

В Java для роботи з об'єктами використовується єдиний синтаксис. Об'єкт класу оголошується з допомогою посилання (reference). Загальна форма оголошення об'єкту класу без виділення пам'яті для нього має наступний вигляд:

ClassName objName;

Де *ClassName* – ім'я класу для якого створюється об'єкт з іменем objName, *objName* – ім'я посилання на об'єкт класу ClassName.

Вищенаведене оголошення говорить про те, що ідентифікатор *objName* є посиланням на об'єкт класу *ClassName*. Для посилання потрібно виділити пам'ять (ініціалізувати посилання) з допомогою оператора *new* як показано нижче:

objName = new ClassName();

Якщо не виділити пам’ять для посилання і звернутись до нього як до об’єкту класу, то виникне помилка.

Існує й інша загальна форма оголошення об’єкту класу. У цьому випадку пам’ять виділяється при його оголошенні:

ClassName objName = new ClassName();

Виділення пам’яті для посилання на об’єкт класу ще називається “приєднання” об’єкту до посилання.

Приклади створення об’єктів різних класів:

Приклад 1. Створення об’єкту класу *CLine*, що реалізує лінію на координатній площині. Програмний код оголошення класу наступний:

```
// клас, що реалізує лінію на координатній площині
public class CLine
{
   // внутрішні змінні класу
   private double x1, y1, x2, y2;
   // конструктори класу
   // конструктор без параметрів
   CLine()
   {
      x1 = y1 = 0;
      x2 = y2 = 1;
   }
   // конструктор з 4 параметрами
   CLine(double x1, double y1, double x2, double y2)
   {
      this.x1 = x1; this.y1 = y1;
```

```
        this.x2 = x2; this.y2 = y2;
    }
    // методи доступу
    // читання даних
    public double GetX1() { return x1; }
    public double GetY1() { return y1; }
    public double GetX2() { return x2; }
    public double GetY2() { return y2; }
    // запис даних
    void SetXY(double nx1, double ny1, double nx2, double ny2)
    {
        x1 = nx1; y1 = ny1;
        x2 = nx2; y2 = ny2;
    }
    // метод, що обчислює довжину лінії
    double Length()
    {
        double len;
        len = Math.sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
        return len;
    }
}
```

Нижченаведений програмний код демонструє створення та використання об’єктів класу *CLine*.

```
public class CTestLine
    {
```

```
    public static void main(String[] args)
    {
        double x, y; // допоміжні змінні
        // створення об'єкту класу CLine з допомогою конструктора без параметрів
        CLine line1 = new CLine();
       // створення об'єкту класу CLine з допомогою конструктора з 4 параметрами
        CLine line2 = new CLine(2.0, 3.0, 4.0, 5.0);
        // використання об'єктів класу line1
        x = line1.GetX1(); // x = 0.0
        y = line1.GetY2(); // y = 1.0
        // використання об'єкту класу line2
        x = line2.GetX1(); // x = 2.0
        x = line2.GetX2(); // x = 4.0
        y = line2.GetY2(); // y = 5.0
          // перевизначення об'єкту line1, попередній об'єкт буде знищено при "зборі"
          сміття
        line1 = new CLine(-4.0, -2.0, 33.4, -20.5);
        x = line1.GetX1(); // x = -4.0
        y = line1.GetY2(); // y = -20.5
    }
}
```

Як видно з коду, у функції *main()* оголошуються два посилання з іменами *line1*, *line2* на об’єкти класу *CLine*. Для цих посилань виділяється пам’ять з допомогою оператора *new*. Потім, для об’єкту *line2* пам’ять заново перевизначається. Стара пам’ять буде звільнена при черговому “зборі” сміття.

Приклад 2. Створення об’єкту класу *CName*, що реалізує ім’я. Функція *main()* демонструє створення об’єкту класу *CName* різними способами з допомогою різних конструкторів.

```
// клас, що реалізує ім'я
public class CName
{
    private String name;
    private String surname;
    private String patronymic;
    // конструктори класу
    // конструктор без параметрів
    CName()
    {
        name = surname = patronymic = "";
    }
    // конструктор з трьома параметрами
    CName(String _name, String _surname, String _patronymic)
    {
        name = _name;
        surname = _surname;
        patronymic = _patronymic;
    }
    // методи доступу
    String GetName() { return name; }
    String GetSurname() { return surname; }
    String GetPatronymic() { return patronymic; }

    // функція, що демонструє застосування класу CName
    public static void main(String[] args)
```

```
    {
        CName nm1 = new CName(); // викликається конструктор без параметрів
        CName nm2; // просто оголошення посилання на об'єкт класу CName, пам'ять ще не виділена
        nm2 = new CName("Happy", "New", "Year!"); // створення об'єкту - виділення пам'яті
        // перевірка
        String str;
        str = nm1.GetName(); // str = ""
        str = nm1.GetSurname(); // str = ""
        str = nm2.GetName(); // str = "Happy"
        str = nm2.GetSurname(); // str = "New"
        str = nm2.GetPatronymic(); // str = "Year!"
    }
}
```

В Java для зберігання даних (об’єктів) існує 5 різних сховищ:

- *регістри*. У цьому випадку дані зберігаються всередині процесора. У регістрах дані обробляються швидше за все. Однак, кількість регістрів є строго обмежена. Компілятор використовує регістри в міру необхідності. У Java немає безпосередніх команд, щоб зберігати усі дані тільки в регістрах. Навіть у потужних мовах C/C++ команди-вказівки для розміщення даних у регістрах носять тільки рекомендаційний характер;
- *стек*. Для програми стек розміщується в загальній оперативній пам’яті (RAM). Стек організовується з використанням покажчиків стеку. Стек працює за принципом LIFO (Last-In-First-Out) – останній прийшов, перший вийшов. Така організація є зручною, коли потрібно виділяти пам’ять для локальних функцій (методів) різних рівнів вкладень. У стеку розміщуються тільки покажчики на об’єкти. Самі об’єкти розміщуються в “купі” (heap або “куча”). Покажчик стеку рухається вниз якщо потрібно виділити пам’ять, і вверх, якщо пам’ять звільняється. Таким чином, за швидкодією, стек поступається тільки регістрам.

Однак, стек не володіє такою гнучкістю як “купа”, тому що компілятору потрібно знати життєвий цикл даних, розміщених в стеку;

– *“купа” або “куча” (heap).* Це є сховище загального призначення яке розміщується в оперативній пам’яті (RAM). Тут зберігаються усі об’єкти (екземпляри об’єктів) Java. “Купа” є більш гнучкою порівняно зі стеком. Тому що компілятор не витрачає додаткових зусиль на визначення тривалості існування об’єктів, що знаходяться в “купі”. У програмі створення об’єкту відбувається з використанням оператора new. У результаті виділяється пам’ять з “купи”. Однак, виділення пам’яті пам’яті з “купи” займає більше часу ніж в стеку. Слід зауважити, що потужні мови C/C++ підтримують явне створення об’єктів як в стеку, так і в “купі”;
– *постійне сховище.* У програмах часто використовуються дані, які є незмінними. До таких даних відносяться константи (наприклад, рядкові константи). Ці дані доцільно вбудовувати прямо в код програми. Інколи константи розміщуються в постійній статичній пам’яті (ROM);
– *зовнішнє сховище.* Зовнішнім сховищем можуть бути довготривалі (persistent) носії інформації, наприклад, жорсткий диск даного комп’ютера або носії інформації віддалених комп’ютерів мережі. Цей вид зберігання даних дозволяє зберігати об’єкти на носіях інформації, а потім відновлювати їх для зберігання в оперативній пам’яті.

Об’єкти зберігаються в “купі” (heap). Посилання на об’єкти зберігаються в стеку.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t
5. Java. Алгоритмізація та програмування / Об’єкти – SchoolboyProgrammer: https://schoolboyprog10.blogspot.com/p/blog-page_89.html
6. Об’єкти. Створення та збереження об’єктів класів. Оператор new. Області зберігання даних в пам’яті. Використання масивів посилань на об’єкти – https://www.bestprog.net/uk/2018/08/19/objects-creating-and-saving-the-objects-of-classes-areas-of-data-storage-in-memory-using-an-arrays-of-references-to-objects_ua/

Тема 2.2. **Використання операторів та алгоритмічних конструкцій. Використання циклів**
Номер заняття: 10-11

Мета: нагадати студентам основні алгоритмічні конструкції, продемонструвати їх реалізацію в Java; розглянути основні види циклів та продемонструвати їх використання.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://youtu.be/xPo1qeLDFXc

частина 2: https://youtu.be/E0ABoPd6lvU

частина 3: https://youtu.be/T-01-ax4FTI

частина 4: https://youtu.be/KdlP8mLarMg

частина 5: https://youtu.be/bL_KIlKrEV8

Короткі теоретичні відомості:

Java, як і інші мови програмування, підтримує умовні інструкції та цикли, що визначають порядок виконання інструкцій у програмі. В англійській мові для цього поняття застосовують термін control flow — керування потоком.

Перед тим як знайомитись з керуючими структурами, спочатку необхідно ознайомитися з блоками. *Блок або складений оператор* – це будь-яка кількість простих інструкцій, які оточені парою фігурних дужок. Блок визначає область видимості ваших змінних. Блоки можуть бути вкладені в середину інших блоків. Ви вже зустрічалися з блоками при

створенні найпростіших програм у методі *main()*. Наступний приклад демонструє вкладення блоку у блок методу *main*:

```
public static void main(String[] args)
{
  int n;
  . . .
  {
    int k;
    . . .
  } // змінна k визначена лише до цього місця
}
```

Проте *не можна визначати однакові змінні в двох вкладених блоках* (на відміну від C++, де це можливо):

```
public static void main(String[] args)
{
  int n;
  . . .
    {
    int k;
    int n; // помилка! – не можна перевизначити n у внутрішньому блоці
    . . .
    }
}
```

Умовний оператор в Java має форму:

if (умова) інструкція;

Умова має бути оточена дужками і, якщо, умова задовольняється (true) буде виконана інструкція за умовою, інакше вона не буде виконана, а буде виконана наступна інструкція після умовної інструкції.

Приклад:

```
int a = 5;
if (a < 100) System.out.println("Число менше ста");
```

Зазвичай, необхідно виконати не одну інструкцію, в такому разі інструкції розміщають у блоці:

```
if (умова){
  інструкція 1;
  .....
  інструкція n;
}
```

В такому разі при істинності умови, виконуються всі інструкції у блоці, якщо умова невірна, то виконується наступна інструкція після закриваючої дужки блоку. Якщо ж необхідно здійснити певну дію в разі не виконання умови, то в такому разі застосовують умовну інструкцію наступного виду:

```
if (умова) інструкція1; else інструкція2;
```

Наприклад:

```
if (yourSales >= target)
{
  performance = "Satisfactory";
  bonus = 100 + 0.01 * (yourSales - target);
}
else
{
  performance = "Unsatisfactory";
  bonus = 0;
```

```
}
```

Інструкції if можуть йти одна за одною без використання else:

```
if (x <= 0) if (x == 0) sign = 0; else sign = -1;
```

Для того, щоб програма була читабельнішою бажано застосовувати фігурні дужки:

```
if (x <= 0) { if (x == 0) sign = 0; else sign = -1; }
```

Вони нічого не змінюють, але вираз стає зрозумілішим. Інструкція чи блок інструкцій виконується лише в разі виконання усіх умов.

Щоправда дану інструкцію можна також переписати ускладнивши умову використавши булевий оператор і (&&):

```
If (x <= 0 && x==0) sign = 0; else sign=-1;
```

Можна також використовувати повторюваність інструкцій *if….else*.

```
if (yourSales >= 2 * target)
{
  performance = "Excellent";
  bonus = 1000;
}
else if (yourSales >= 1.5 * target)
{
   performance = "Fine";
   bonus = 500;
}
else if (yourSales >= target)
{
   performance = "Satisfactory";
   bonus = 100;
}
```

```
else
{
  System.out.println("You're fired");
}
```

Це дає можливість перевірити ряд умов, якщо попередні умови не виконуються.

Цикли – це послідовність інструкцій, які можуть повторно виконуватись певну кількість раз в залежності від заданої в програмі умови. Розрізняють цикли з передумовою, з післяумовою та з лічильником.

Цикл while (перекладається як «доки») – це цикл з передумовою, тіло якого (тобто інструкція або блок інструкцій) виконується, якщо умова істинна. Якщо умова з самого початку хибна, то цикл не виконається жодного разу.

Загальний вигляд:

```
while (умова) інструкція;
```

Якщо немає фігурних дужок, то перша інструкція, яка йде після оголошення циклу, вважається *тілом циклу*. Всі інші інструкції знаходяться поза циклом. Якщо в циклі повинні виконуватись кілька інструкцій, то необхідно використати фігурні дужки. Вони також можуть використовуватись при одній інструкції заради кращого візуального розуміння коду.

```
while (умова){
  інструкція 1;
  ...
  інструкція N;
}
```

В наступному прикладі демонструється мінігра із вгадуванням числа від 0 до 10, яка створена з використанням циклу while:

```
import java.util.*;
public class Game {
   public static void main(String[] args) {
```

```
        Scanner in = new Scanner(System.in); // створюємо сканер для введення даних з консолі
        Random generator = new Random();      //створюємо генератор випадкових чисел
        System.out.println("Спробуйте відгадати число від 0 до 10");
        int gn;
        String more = "Y";
        while (more.equals("Y") || more.equals("y")) {  // поки змінна more рівна “Y” або “y”
          gn = generator.nextInt(10); // генерація випадкового числа від 0 до 10;
          System.out.print("Введіть число від 0 до 10: ");
          int number = in.nextInt();  // зчитуємо число з клавіатури
          if (gn == number)
             System.out.print("Вгадали!!!  Спробуєте ще раз? (Y/N)");
          else
             System.out.print("Не вгадали. Спробуєте ще раз? (Y/N)");
          more = in.next();    // отримати відповідь
        }
    }
}
```

Результат виконання:

```
Спробуйте відгадати число від 0 до 10
Введіть число від 0 до 10: 6
Не вгадали. Спробуєте ще раз? (Y/N)y
Введіть число від 0 до 10: 7
Вгадали!!!  Спробуєте ще раз? (Y/N) n
```

Програма генерує випадкове число при кожному повторі циклу і пропонує вгадати його. Після вводу користувачем числа – виводить відповідне повідомлення вгадано чи ні. Після цього пропонується здійснити нову спробу. Якщо користувач вводить з клавіатури “Y" або “y”, то гра продовжується, якщо введе щось інше, то завершується.

Для генерації випадкових чисел використано клас *Random*, що містить методи для генерації випадкових чисел. Зокрема, у нашій програмі використано метод *nextInt()*, який дозволяє генерувати випадкові числа.

Для зчитування з клавіатури використано клас *Scanner*, який був доданий в java 5.0, для зручного вводу з клавіатури. Метод *nextInt()* – читає ціле число, *next()* – читає цілий рядок з клавіатури.

Для того, щоб переконатися, що користувач хоче продовжити гру використано метод *equals()* з класу *String – Strint1.equals (String2)*, що перевіряє чи один рядок (*String1*) тексту рівний іншому (*String2*).

У вищенаведеній програмі пояснення потребують три рядка. Для початку Вам необхідно лише в загальному зрозуміти їхнє призначення.

```
import java.util.*;

Random generator = new Random();

gn=generator.nextInt(10);
```

Нам необхідний генератор цілих чисел, який реалізує класс *Random* з пакету *java.util*. Для цього ми імпортуємо відповідний клас (в C/C++ аналогом є підключення бібліотеки). Далі в програмі створюємо змінну generator, яка вказуватиме на екземпляр класу *random* і дозволятиме звернення до методів даного класу. *new Random()* – створює відповідний об'єкт. Для генерації випадкових чисел ми використовуємо метод *nextInt()* класу *Random*.

Якщо необхідно, щоб умова виконувалася хоча б один раз можна скористатися *циклом з післяумовою do/while*:

```
do інструкція while (умова);
```

Зокрема, в програмі з вгадуванням чисел, логічніше було б застосувати саме даний цикл, оскільки необхідне хоча б одне виконання тіла циклу.

```
import java.util.*;
```

```
public class Tmp {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in); // створюємо Сканер для введення даних з консолі
        Random generator = new Random();      // створюємо генератор випадкових чисел
        System.out.println("Спробуйте відгадати число від 0 до 10");
        int gn;
        String more;
        do {
            gn = generator.nextInt(10); //генерація випадкового числа від 0 до 10;
            System.out.print("Введіть число від 0 до 10: ");
            int number = in.nextInt();
            if (gn == number)
                System.out.print("Вгадали!!!  Спробуєте ще раз? (Y/N)");
            else
                System.out.print("Не вгадали. Спробуєте ще раз? (Y/N)");
            more = in.next();
        } while (more.equals("Y") || more.equals("y"));
    }
}
```

Цикл for (цикл з лічільником) – доволі часто вживаний цикл. Він застосовується при необхідності виконати інструкції певну кількість раз з одночасним збільшенням або зменшенням певної змінної. Часто використовується для здійснення перебору певних масивів даних, зокрема, також для сортування масивів. Приклад використання:

```
for (int i = 1; i <= 10; i++) {
        System.out.println(i);
```

```
}
```

Наведений вище приклад виведе на консолі в стовпчик числа від 1 до 10. Як бачимо в умові циклу перший слот відводиться для ініціалізації змінної, причому оголосити змінну можна і в іншому місці. Другий слот – для умови, яка перевіряється перед виконанням ітерації, третій слот – вказує як модифікувати змінну-лічильник. Тобто в наведеному прикладі при кожному виконанні ітерації, лічильник "i" буде збільшуватися на одиницю поки не стане рівним десяти.

Найчастіше даний цикл використовується для перебору елементів масиву. *Масив* – це впорядкований набір даних одного типу. Найпростіший масив можна оголосити та ініціалізувати таким чином: *int a[]={1, 5, 6, 1, 3};*. Для того, щоб звернутися до певного елементу масиву використовуються квадратні дужки з відповідним індексом елементу. Наприклад *a[3]* – звернення до четвертого елементу масиву (номери елементів відраховуються з нуля). В наступному прикладі створюється масив і послідовно виводяться його елементи:

```
public class MyArray {
   public static void main(String[] args) {
      int a[] = {1, 5, 6, 1, 3}; // створюємо масив і заповнюємо його числами
      int size = a.length;
      System.out.println("Елементи масиву:");
      for (int j = 0; j < size; j++) {
         System.out.println("a[" + j + "]=" + a[j]);
      }
   }
}
```

Результат виконання:

```
Елементи масиву:
a[0]=1
a[1]=5
a[2]=6
```

a[3]=1

a[4]=3

Починаючи з java SE 5.0 в мові з'явився новий цикл – *покращений for*, призначення якого є перебір елементів масиву або подібних до масиву типів даних (колекції).

Загальний вигляд циклу наступний:

```
for (type var : arr) {
   //тіло циклу
}
```

Наприклад, вивести елементи масиву, можна таким чином:

```
for (int element : a)
   System.out.println(element);
```

Використання цього циклу, дозволяє уникнути проблем пов'язаних з помилками при заданні умови в класичному циклі for. В інших мовах програмування цикл такого виду так і називається *foreach*, проте, щоб уникнути необхідності значних змін в пакетах, в java пішли простішим шляхом і *перевантажили цикл for*.

В java відсутня інструкція *goto*, яка дозволяла переходити в будь-яке місце в програмі. Її використання давно вважається *поганим стилем програмування*, оскільки робить текст програми заплутаним. Про це, зокрема, писав ще *Дональд Кнут* і він же зазначав, що інколи все таки корисно її застосовувати, щоб припинити виконання певного методу, циклу чи блоку і вийти за їхні межі. Для цієї мети в java існують спеціальні інструкції. Зокрема, інструкцію *break* можна використати для *передчасного виходу з циклу*. Якщо ми використовуємо вкладені цикли (один цикл в іншому), то інструкція *break* припинить цикл, в якому вона знаходиться. Якщо вона знаходиться у внутрішньому циклі, то він припинить своє виконання, а зовнішній же цикл буде виконуватись і далі. Якщо потрібно повністю припинити виконання і внутрішнього і зовнішнього, то використовується інструкція *break* з міткою.

Якщо ж нам не потрібно виходити з циклу, а лише *припинити певну його ітерацію*, то для цієї використовується інструкція *continue*, яка переносить порядок виконання до заголовку інструкції.

Щоправда *завжди можна обійтися без цих інструкцій змінивши логіку програми*. Деякі з програмістів уникають використовувати інструкції break та continue.

Інструкція if/else може бути доволі громіздкою, якщо необхідно здійснити *множинний вибір* з багатьох альтернатив. Тож як і в C/C++ в java існує інструкція *switch*, яка здійснити вибір з багатьох варіантів. Щоправда вона дещо незграбна і деякі програмісти вважають за краще уникати її використання.

Наприклад, якщо Ви організовуєте певне меню і пропонуєте користувачу вибрати, номер конкретного пункту, то можна використати наступний код:

```
Scanner in = new Scanner(System.in);
System.out.print("Select an option (1, 2, 3, 4) ");
int choice = in.nextInt();
switch (choice) {
   case 1:
      . . .
      break;
   case 2:
      . . .
      break;
   case 3:
      . . .
      break;
   case 4:
      . . .
      break;
   default:
      // bad input
      . . .
      break;
```

Якщо пропустити інструкцію *break*, то всі інші інструкції будуть також виконані. Тобто якщо справдиться умова першого варіанту, то будуть здійснені ще й дії вказані для виконання у всіх інших варіантах.

До версії Java 7, яка вийшла у 2011 році, case мітка мала бути лише цілим числом або нумерованою константою. *Починаючи із Java 7 можна перевіряти таким чином на рівність також рядки*:

```
String input = . . .;
switch (input)
{
   case "A":
      . . .
      break;
   . . .
}
```

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 2.3. **Розробка та використання методів**
Номер заняття: 13

Мета: ознайомити студентів з поняттям методу; продемонструвати переваги об'єктно-орієнтованого підходу як найадекватнішого способу відображення оточуючої реальності; розглянути різні аспекти створення та використання методів.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://youtu.be/-khOv3pyZ_8

частини 2-6: https://college.page.link/99kR

Короткі теоретичні відомості:

Методи в JAVA – це аналог підпрограм, функцій, процедур в інших мовах програмування. За допомогою методів ми виносимо текст повторюваного коду програми окремо в тіло методу, після чого можна викликати даний метод з будь-якого місця програми, безліч разів.

Спрощене оголошення та визначення методу, який ми зараз будемо використовувати, має вигляд:

```
тип_повернення  назва_методу(параметри){
//тіло методу;
інструкція1;
інструкція2;
```

.....

інструкціяN;

}

Тип_повернення – результат виконання методу, наприклад він може повертати об’єм сейфу, тоді тип_повернення буде *double*. Якщо метод нічого не повертає, то вказується слово ключове слово *void*.

Виклик методу здійснює наступна інструкція: *назва_методу(параметри)*. Взагалі в термінології мов програмування для виклику методу використовуються *аргументи*, а в самому методі – це вже *параметри*, оскільки передаються лише значення змінних, а не самі змінні. Аргументи та параметри мають бути одного і того ж типу. Якщо ми передаємо цілочисельне значення, то і параметр повинен бути цілочисельним і т.п. Тобто в методі створюються нові змінні. В деяких мовах, наприклад в C++, як аргумент можна передати посилання на певну змінну, таким чином її можна буде модифікувати в функції через вказівник на дану змінну. *В Java в метод передаються лише значення змінних*, тому розрізнення аргументів і параметрів не настільки суттєве, тож ми і там і там використовуватимемо термін «параметр». Об'єкти ж в методи передаються за посиланням. Тобто, при передачі в якості аргументу об'єкта, буде передане посилання на об'єкт, а не створений новий об'єкт.

Повернемось до прикладом з сейфом. Повторюваними є присвоювання значень змінним, обчислення та виведення об’ємів. Тому корисно буде створити два методи *safeValue (double width, double height, double depth)* та *safeVolume ()*. Перший для присвоєння змінним значень, а другий для обчислення об'єму.

Відповідна програма матиме вигляд:

```
class Safe {
    double width = 10; // поля класу можуть бути ініціалізовані з самого початку
    double height = 10;
    double depth = 10;
    double safeVolume = 0;
    // метод присвоєння значень змінним
    void safeValue(double pWidth, double pHeight, double pDepth) {
        width = pWidth;
```

```
        height = pHeight;
        depth = pDepth;
    }
    // метод для обчислення об'єму сейфа
    double safeVolume() {
        return width * height * depth;
    }
}
public class CoinVolume {
    public static void main(String[] args) {
        double width1 = 10, height1 = 20, depth1 = 40;
        Safe mySafe1 = new Safe(); //створюємо перший сейф
        Safe mySafe2 = new Safe(); //створюємо другий сейф
        //задаємо розміри сейфу
        //викликаємо метод safeValue() класу Safe, що ініціалізує поля об'єкту
        mySafe1.safeValue(width1, height1, depth1);
        mySafe2.safeValue(10.0, 15.0, 15.5); //можна і так
        //виводимо на екран об'єми сейфів
          //для чого викликаємо метод safeVolume(), який повертає обчислений об'єм
          кожного сейфу
        System.out.println("Об'єм 1-го сейфу=" + mySafe1.safeVolume());
        System.out.println("Об'єм 2-го сейфу=" + mySafe2.safeVolume());
    }
}
```

Як бачимо код програми досить простий. Тепер набагато легше модифікувати програму, додаючи нові сейфи. Крім того можна, наприклад, вивести повідомлення з інформацією

про об’єм, розмістивши його в метод *safeVolume()* або зовсім в окремий метод, наприклад, *printVolume()*.

Слід зауважити, щодо назв змінних-параметрів. Так для висоти, ширини та глибини в методі *safeValue()* вибрані назви *pWidth, pHeight, pDepth*. Вони могли б мати назви і просто *width, height, depth*, але тоді б вони перекрили доступ до однойменних змінних класу. В такому випадку, щоб звернутися до змінних класу з методу необхідно вживати ключове слово *this*. Наприклад: *this.height=height* – тут ми присвоюємо змінній класу *this.height* одержаний методом параметр *height*.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Змістовий модуль 3. Інкапсуляція. Успадкування та повторне використання коду

Тема 3.1. **Інкапсуляція та конструктори**

Номер заняття: 15

Мета: нагадати студентам основні принципи ООП; продемонструвати переваги об'єктно-орієнтованого підходу як найадекватнішого способу відображення оточуючої реальності; познайомити їх з поняттям конструктора та особливостями створення та використання конструкторів для створення об'єктів класу.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://youtu.be/ybQk1G1lsBo

Короткі теоретичні відомості:

Одною із переваг ООП є те, що користувачі класів можуть використовувати їх, майже не знаючи як вони реалізовані, використовуючи лише кілька методів для роботи з конкретним об'єктом. Таке *приховування реалізації класу* слугує також і певним захистом від неправильної роботи з даними. Даний принцип реалізації називається «інкапсуляцією». Якщо візьмемо реальний світ, то наприклад ви не знаєте як влаштований телевізор, проте ви можете його увімкнути та за допомогою кнопок переключати канали. Сама ж схемо-технічна реалізація прикрита корпусом телевізора і переважно невідома глядачам ТВ. Аналогічно і класи розробляються за схожим принципом. Створюються певні методи (*інтерфейс класу*), через які відбувається доступ до полів класу(змінних) та його методів. Всі інші класи, методи, поля, які слугують лише для обслуговування внутрішнього функціонування класу намагаються захистити, щоб

до них не було доступу без крайньої на те потреби. Це в свою чергу зменшує кількість помилок в програмі через невмілі дії користувача цього класу.

Інкапсуляція – один з основних принципів об'єктно-орієнтованого програмування. Йдеться про те, що об’єкт вміщує не тільки дані, але і правила їх обробки, оформлені в вигляді виконуваних фрагментів(методів). Доступ до стану об'єкта напряму заборонено, і ззовні з ним можна взаємодіяти виключно через заданий інтерфейс, що дозволяє контролювати правильність звернення до полів та їхню ініціалізацію. Також інкапсулюються методи, які виконують допоміжну роль і не бажано, щоб користувач-програміст мав доступ до них. Оскільки користувачі-програмісти працюють лише через відкриті елементи класів, то розробники класу можуть як-завгодно змінювати всі закриті елементи і навіть перейменовувати та видаляти їх, не турбуючись, що десь хтось їх використовує у своїх програмах.

Наприклад, вам потрібно зробити певну програму по роботі із списком автомобілів. Для цього потрібний відповідний клас Car:

```
package org.wikibooks.uk.osvjava;
public class Car {
    public static int count;
    public int id;
    public String _maker;
    public double _price;
    public String _year;
    public String _color;
    //конструктор без параметрів
    public Car() {
        count++;
        id = count;
    }
    //конструктор з параметрами, який ініціалізує всі поля класу
    public Car(String maker, String color, double price, String year) {
```

```
    _maker = maker;
    _price = price;
    _year = year;
    _color = color;
    count++;
    id = count;
}
//заміщення (перевизначення) методу toString() класу Object
//замість дескриптора об'єкту, він виводитиме інформацію по автомобілю
@Override
public String toString() {
      return "Авто " + id + " " + _maker + " " + _color + " " + _price + " " + _year
      + " ";
}
//тестовий метод main
public static void main(String[] args) {
   //створюємо об'єкт car1 конструктором без параметрів
    Car car1 = new Car();
    car1._maker = "Audi";
    car1._price = 10000;
    car1._year = "2000";
    car1._color = "red";
   //створюємо об'єкт car2 конструктором з параметрами
    Car car2 = new Car("BMW", "black", 12000, "2001");
    //виведення інформації про автомобілі
    //при цьому застосовуватиметься заміщений в цьому класі метод toString
```

```
        System.out.println(car1);
        System.out.println(car2);
    }
}
```

Результат виконання:

```
Авто 1: Audi red 10000.0 2000
Авто 2: BMW black 12000.0 2001
```

У вищенаведеному прикладі для того, щоб рахувати кількість об'єктів ми створюємо статичну змінну *count*, яка буде спільною для всіх об'єктів *car*. В методі *main* ми створюємо екземпляри класу *Car* двома способами. Спочатку вручну ініціалізуємо кожне поле *car1*, а поля *car2* ініціалізуємо через конструктор. Крім того, що другий спосіб є більш простий, перший спосіб може слугувати ще й джерелом ряду помилок. Наприклад, ми можемо забути ініціалізувати певне поле. Крім того ініціалізуючи через конструктор ми можемо здійснити попередню перевірку на правильність введення даних. Наприклад на правильність введення назви виробника тощо. Також, маємо два службових поля: *count* та *id*, які ініціалізуються в конструкторах, проте ми можемо задати значення і напряму, що може зашкодити логіці функціонування об'єкту класу. Ми запросто можемо вказати, що є 100 автомобілів, хоча насправді їх 66. І якщо будемо десь використовувати цикл з перебору автомобілів з врахування змінної *count*, це викличе помилку. Тому в ООП і придумано інкапсуляцію – приховування внутрішньої реалізації класу. Рекомендується оголошувати поля та методи з самого початку закритими і лише в разі необхідності надавати до них більший доступ.

Модифікуємо дещо нашу програму:

```
package org.wikibooks.uk.osvjava;
public class Car {
    private static int count=0;
    private int id;
    private String _maker;
    private double _price;
    private String _year;
```

```
private String _color;
//конструктор з параметрами, який ініціалізує всі поля класу
public Car(String maker, String color, double price, String year) {
    _maker = maker;
    _price = price;
    _year = year;
    _color = color;
    count++;
    id = count;
}
//заміщення (перевизначення) методу toString() класу Object
//замість дескриптора об'єкта, він виводитеме інформацію по автомобілю
@Override
public String toString() {
    return id+". "+ _maker + " " + _color + " " + _price + " " + _year + " ";
}
//метод для отримання значення поля id
public int getId() {
    return id;
}
//метод для отримання кількості автомобілів
public static int getCount() {
    return count;
}
//тестовий метод main
public static void main(String[] args) {
```

```
        Car car[]=new Car[5];
        car[0] = new Car("Audi","red",10000,"2000" );
        car[1] = new Car("BMW", "black", 12000, "2001");
        car[2] = new Car("Daewoo", "white", 8000, "2001");
        car[3] = new Car("Reno", "black", 12000, "2001");
        for (int i = 0; i < Car.getCount(); i++) {
            System.out.println(car[i]);
        }
    }
}
```

Результат виконання:

1. Audi red 10000.0 2000

2. BMW black 12000.0 2001

3. Daewoo white 8000.0 2001

4. Reno black 12000.0 2001

Як бачимо усі поля у нас тепер приватні. Робота з закритими полями можлива лише через відповідні методи. Для доступу до полів *count* та *id* створено методи *getCount* та *getId*. В разі необхідності можна створити методи для доступу до інших полів. Також можна створити певні методи модифікації окремих полів (*setId* і т.п.) та передбачити в них попередню перевірку значень, що вводяться. Тож ми усунули можливість появи помилок при програмуванні пов'язаних з неправильним використанням полів та методів класу. Іншим програмістам буде значно легше використовувати клас *Car*, їм не потрібно вникати в особливості реалізації даного класу.

Необхідно зазначити, що методи, які читають значення полів прийнято називати з використанням префіксу *get* з наступним вказанням назви змінної, а для тих які модифікують значення використовують префікс *set*. В англомовній термінології можна зустріти терміни *getter* та *setter* методи або метод *accesor* та метод *mutator*.

Зверніть увагу як здійснюється звернення до змінної *count*. Насправді до методу можна звернутися з використанням об'єкту *car[i].getCount()*, але оскільки даний метод та змінна

є статичними, тобто вона та метод спільно використовуються усіма об'єктами, то більш логічним є зверненням до неї через назву клас *Car.getCount()*. Даний метод можна викликати до створення будь-яких об'єктів. В такому разі ми просто отримаємо 0. Якщо б змінна не була приватною, то доступ до неї можна було б робити без застосування методу, безпосередньо: *Car.count*.

Згадаємо тепер приклад з сейфами з попередньої лекції. Замість методу *safeValue()*, який в нас заповнює змінні об’єкту значеннями ми можемо створити метод, який буде мати назву таку ж як і клас, тобто *Safe(pWidth, pHeight, pDepth)*. Це дасть нам можливість ще скоротити програму, оскільки при створенні об’єкту ми зможемо зразу ж задавати розміри сейфу.

```
Safe mySafe1 = new Safe(10.0, 15.0, 20.0)
```

Такі методи носять назву *конструктор класу*. Коли ми писали просто *new Safe();* то віртуальна машина використовувала конструктор по замовчуванню без параметрів, який практично нічого корисного для нас не робив. Тепер же ми можемо використовувати новий створений нами конструктор.

Таким чином, новий варіант програми:

```
class Safe {
   double width;
   double height;
   double depth;
   // конструктор
   Safe(double pWidth, double pHeight, double pDepth) {
      width = pWidth;
      height = pHeight;
      depth = pDepth;
   }
   // обчислюємо об'єм сейфу
   double getVolume() {
      return width * height * depth;
```

```
    }
}
public class CoinsVolume {
    public static void main(String[] args) {
        double width1 = 10, height1 = 20, depth1 = 40;
        Safe safe1 = new Safe(width1, height1, depth1); // створюємо 1-й сейф
        Safe safe2 = new Safe(10.0, 15.0, 20.0); // створюємо 2-й сейф
        Safe safe3 = new Safe(10.3, 15.4, 20.5); // створюємо 3-й сейф
        Safe safe4 = new Safe(20.0, 30.0, 20.0); // створюємо 4-й сейф
        printSafeVolume(safe1, 1); // виводимо об'єм 1-го сейфу
        printSafeVolume(safe2, 2); // виводимо об'єм 2-го сейфу
        printSafeVolume(safe3, 3); // виводимо об'єм 3-го сейфу
        printSafeVolume(safe4, 4); // виводимо об'єм 4-го сейфу
    }
    // виведення об'єму сейфу
    // safe - сейф
    // number - номер сейфу
    static void printSafeVolume(Safe safe, int number) {
        // викликаємо метод getVolume(), що обчислює об’єм сейфу і результат виводимо на екран
        System.out.println("Об'єм " + number + "-го сейфу = " + safe.getVolume());
    }
}
```

Результат виконання:

```
Об'єм 1-го сейфу = 8000.0
Об'єм 2-го сейфу = 3000.0
```

Об'єм 3-го сейфу = 3251.71

Об'єм 4-го сейфу = 12000.0

В наведеному прикладі, щоб обчислити об’єм нового сейфу, нам потрібно додати лише два рядки тексту (дві інструкції). Щоправда навіть це можна автоматизувати за допомогою використання іншого типу даних – масивів, які будуть розглядатися пізніше. Можна також додати клас *Coin*, в якому був би метод для обчислення сукупного об’єму різноманітних монет. Ви можете спробувати зробити таку програму зараз. Це буде корисним для засвоєння викладеного матеріалу.

Тема класів та методів значно комплексніша, тому основне, що Ви повинні винести зцього заняття – це розуміння того, як використовуються методи (в тому числі – конструктори) класів. Java надає великий набір уже готових класів та методів, які значно спрощують роботу програміста.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 3.2. **Створення та використання масивів**
Номер заняття: 17

Мета: нагадати студентам поняття масиву; розповісти про особливості створення та використання масивів в JAVA; масиви різних видів.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://youtu.be/_CIHMRxUzak

частини 2-6: https://college.page.link/4vua

Короткі теоретичні відомості:

Масив – це впорядкований набір однотипних елементів, на які посилаються по спільному імені. Це доволі зручний засіб групування інформації. Масиви можна створювати з елементів будь-якого типу. До конкретного елементу в масиві звертаються за індексом (номером). Вони можуть бути як *одновимірні* так і *багатовимірні*.

Одновимірні масиви – це список однотипних елементів. Загальний формат оголошення такого масиву:

тип-елементів назва-масиву[];

Наприклад

int month_days[]; // масив цілих чисел

Існує також інша форма оголошення масиву:

```
int[] month_days;
```

Проте для того, щоб масив почав існувати необхідно виділити під нього пам'ять, за допомогою операції *new*.

```
назва-масиву = new тип-елементів [розмір];
```

де розмір - планована кількість елементів у масиві.

```
month_days = new int[12];
```

або зразу ж:

```
int month_days[] = new int[12];
```

Таким чином відбувається виділення пам'яті під масив і ініціалізації елементів масиву нулями. В подальшому можна напряму звертатися до елементів масиву вказуючи індекс у квадратних дужках. Нумерація елементів в масиві в java відбувається з нуля. Тобто в наведеному прикладі звернення до першого(нульового) елемента – *month_days[0]*, а до останнього – *month_days[11]*. Java не дозволить програмі звернутися поза межі масиву, щоправда помилка буде вказана лише на етапі виконання програми через викидання винятку (виключення, exception).

```
month_days[5] = 30;

System.out.println(month_days[5]);
```

Масиви також можна ініціалізувати зразу ж при їхньому оголошенні, не використовуючи операції *new*, аналогічно як це відбувається при роботі з простими типами даних.

Наступний приклад зразу ж при оголошенні ініціалізує масив *month_days[]* кількістю днів в місяцях.

```
public class DaysOfMonth {
   public static void main(String[] args) {
   int month_days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}; //оголошуємо та ініціалізуємо масив
   System.out.println("Травень має " + month_days[4] + " день"); // вивід на консоль
   }
}
```

Результат виконання на екрані:

Травень має 31 день

Наступний приклад демонструє знаходження максимального числа в одновимірному масиві.

```
public class ArrayMax {
   public static void main(String[] args) {
      double array[] = {1.1, 2.2, 1.1, 3.2, 1.2, 2.1};
      double max = array[0];
      for (int i = 0; i < 6; i++) {
         if (max < array[i])
            max = array[i];
      }
      System.out.println("Максимальне число в масиві: " + max);
   }
}
```

Як бачимо спочатку змінній *max* присвоюється значення нульового елементу масиву, після чого в циклі іде послідовне порівняння з кожним наступним числом до останнього. Якщо при порівнянні чергове значення в масиві більше за максимальне в змінній *max*, то змінній *max* присвоюється дане значення. Як Ви уже зрозуміли, по закінченню циклу у змінній *max* міститиметься максимальне значення, яке і буде виведене на консоль:

Максимальне число в масиві: 3.2

В Java масиви є *об'єктами* (про об'єкти ми вже говорили раніше), що забезпечує деяку додаткову функціональність масивам. Зокрема, можна дізнатися довжину масиву наступним чином *array.length*. Для вищенаведеного прикладу можна замінити рядок з циклом таким чином:

```
for (int i =0; i < array.length; i++){
```

Багатовимірні масиви по суті – це *масив масивів*. Робота з багатовимірними масивами подібна до роботи з одновимірними. Відмінність лише в тому, що використовуються додаткові квадратні дужки. Переважно використовуються двовимірні масиви, які

служать для роботи з табличними даними та трьохвимірні масиви. Двовимірний масив та трьохвимірний, можна оголосити наступним чином:

```
int twoD[][] = new int [4][5]; //створення масиву 4x5
int threeD[][][] = new int[5][5][5]; //створення масиву 5x5x5
```

Для двовимірного лівий індекс означає номер рядка, а правий номер стовпця. Це можна уявити наступним чином:

```
[0,0][0,1][0,2][0,3][0,4]
[1,0][1,1][1,2][1,3][1,4]
[2,0][2,1][2,2][2,3][2,4]
[3,0][3,1][3,2][3,3][3,4]
```

Трьохвимірний масив можна уявити у вигляді куба. Крім номера рядка і номера стовпця, додається ще індекс елемента вглибину.

Наступна програма створює масив 5 на 4, заповнює його випадковими числами і виводить на екран.

```
import java.util.Random;                    // імпортуємо клас Random
public class RandomArray {
   public static void main(String[] args) {
      int m = 5, n = 4; //оголошуємо і ініцілізуємо змінні з розмірами масиву
      int Array[][] = new int[m][n];    //оголошуємо і ініціалізуємо масив
      Random generator = new Random();  // створюємо генератор випадкович чисел
      int gn; //змінна в яку буде записуватися згенероване генератором число
      /* заповнюємо масив випадковими числами */
      for (int i = 0; i < m; i++)            //проходимось по стовпцях
         for (int j = 0; j < n; j++) {       //проходимось по рядках
            gn = generator.nextInt(100);  //генерація випадкового числа від 0 до 100;
            Array[i][j] = gn;           //записуємо згенероване випадкове число
         }
```

```
        /* Виводимо результат */
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++)       // зверніть увагу на відсутність фігурної дужки
                System.out.print(Array[i][j] + "    ");// рядок відноситься до масиву по j
            System.out.println();       //виводимо символи переводу каретки і нового рядка
            //після кожного проходження стовпцевих елементів рядка
        }
    }
}
```

В результаті на екрані одержимо:

```
94   47   65   0
99   20   60   69
80   33   63   73
35   50   48   81
39   19   4    85
```

В наведеному прикладі в кожному рядку однакова кількість елементів(стовбців). В Java можна створити двовимірні масиви з різною кількістю елементів в рядках (*зазубрені або зубчасті масиви, jagged arrays*).

```
int twoD[][] = new int[5][]; //створюємо двовимірний масив з 5-ма рядками
twoD[0] = new int[5]; // виділяємо пам'ять для 5-ти елементів нульового рядка
twoD[1] = new int[4]; // перший рядок матиме 4-ри елементи
twoD[2] = new int[3]; // другий - 3
twoD[3] = new int[2]; // третій - 2
twoD[4] = new int[1]; // четвертий – 1
```

Використання таких *нерівних (нерегулярних) масивів* не рекомендується, оскільки з ними важче працювати і можна припуститися ряд помилок, але в деяких ситуаціях можуть бути доволі корисними.

Як і з одновимірними масивами, ми можемо зразу ж ініціалізувати масив необхідними значеннями при його оголошенні.

```
public class Array2 {
   public static void main(String[] args) {
      int[][] Array= {
         {5, 6, 1, 3},
         {3, 4, 2, 1},
         {1, 2, 2, 2}
       };
       for (int i = 0; i < 3; i++){
         for (int j = 0; j < 4; j++)
             System.out.print (Array[i][j]+"   ");
         System.out.println();
      }
   }
}
```

Результат виконання:

```
5   6   1   3
3   4   2   1
1   2   2   2
```

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 3.3. **Реалізація успадкування**
Номер заняття: 19

Мета: нагадати студентам основні поняття ООП; розповісти про особливості створення ієрархії класів, використання UML, кодогенерацію та зворотний інжинірінг.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://youtu.be/ybQk1G1lsBo 

Короткі теоретичні відомості:

Успадкування або *наслідування* (англ. inheritance) – це ще один важливий механізм об'єктно-орієнтованого програмування, який дозволяє створювати нові класи на основі вже існуючих. Клас на базі якого створюється підклас називають надкласом, *суперкласом* або ж батьківським класом. Новий клас, що розширює (extends) батьківський клас називають *підкласом* або дочірнім класом. На відміну від C++ де клас може мати кілька батьківських класів, *мова програмування Java підтримує лише одинарне успадкування*, тобто може бути лише один безпосередній надклас. У надкласу звичайно може бути свій надклас, проте також лише один безпосередній. Множинне успадкування доволі складне у застосуванні і вимагає обережного підходу, тому творці Java вирішили відмовитися від нього.

Припустимо у вас є клас *SimpleRoom*, який містить поля *width* (ширина) та *length* (довжина) кімнати та методи виведення інформації про кімнату.

```
package ua.wikibooks.oj;

public class SimpleRoom {
```

```
    protected double width=0.0;
    protected double length=0.0;
    public SimpleRoom(double width, double length) {
        this.width=width;
        this.length=length;
        System.out.println("SimpleRoom створено");
    }
    public void info (){
        System.out.println("Кімната: ширина = "+width+", довжина = "+length);
        System.out.println("Площа кімнати: "+width*length);
    }
    public static void main(String[] args) {
        SimpleRoom s=new SimpleRoom(5, 5);
        s.info();
    }
}
```

Результат виконання:

```
SimpleRoom створено
Кімната: ширина = 5.0, довжина = 5.0
Площа кімнати: 25.0
```

Тепер нехай вам необхідний клас, який би містив ще й інформацію про висоту кімнати й обчислював ще й об'єм кімнати. Можна створити повністю новий клас, можна модифікувати вже існуючий клас, а можна створити клас *розширивши базовий клас SimpleRoom*. Якщо клас *SimpleRoom* вже використовується в інших програмах, то змінювати його прийдеться обережно. Для даного прикладу прийдеться створити ще один конструктор, а не модифікувати існуючий. В складніших випадках може знадобитися набагато більше дій. Крім того може бути, що клас *SimpleRoom* вже стандартизований, задокументований, його використовує чимало інших розробників і

змінювати його просто так ви не маєте права. Тож виходом є створення нового класу під ваші потреби. Завдяки ж можливості успадкування, нам не потрібно повністю переписувати клас. На основі класу *SimpleRoom* можна створити новий клас *SimpleRoom2*. Для цього достатньо вказати ключове слово *extends* (що означає "розширює") і вказати назву батьківського класу. Новий, *дочірній клас отримує доступ до публічних і захищених полів та методів батьківського класу*.

```
package ua.wikibooks.oj;
public class SimpleRoom2 extends SimpleRoom {
   protected double height;
   public SimpleRoom2(double w, double l, double h) {
      super(w, l);
      height=h;
      System.out.println("SimpleRoom2 створено");
   }
   public void info2(){
        System.out.println("Кімната: ширина = "+super.width+", довжина = "+super.length+", висота= "+this.height);
      System.out.println("Площа кімнати: "+width*length); // якщо немає конфлікту з іменами, то можна і пропустити super
      System.out.println("Об'єм кімнати: "+width*length*height);
   }
   public static void main(String[] args) {
      SimpleRoom2 s2 = new SimpleRoom2(5, 5, 3);
      System.out.println("Метод info SimpleRoom");
      s2.info();
      System.out.println();
      System.out.println("Метод info2 SimpleRoom2");
      s2.info2();
```

```
    }
}
```

Розберемо вищенаведений приклад. При створенні класу ми зазначили, який клас ми розширюємо. В класі введене нове поле *height*. Далі ми створили конструктор, в якому іде звернення до батьківського конструктора. Якщо б не було конструкторів з параметрами, то неявні виклики конструкторів викликалися б у такій же послідовності. Спочатку викликається конструктор дочірнього класу, з нього викликається батьківський конструктор, створюється батьківський об'єкт, далі іде завершення конструктора дочірнього класу і створюється дочірній об'єкт. Для виклику конструктора суперкласу ми скористалися методом *super* з відповідними аргументами для батьківського конструктора:

```
super(w, l);
```

Також зверніть увагу як відбувається звернення до полів батьківського класу. Якщо поля не приватні, то вони доступні з дочірнього класу. Тож до них можна звертатися безпосередньо по імені, або ж скористатися для доступу ключовим словом *super* (замість об'єктної змінної). Якщо б у нашому дочірньому класі існували б однойменні поля з батьківським класом (наприклад, і там і там *width*), то поля батьківського класу були б доступні лише з допомогою *super*.

Результат виконання SimpleRoom2:

```
SimpleRoom створено
SimpleRoom2 створено
Метод info SimpleRoom
Кімната: ширина = 5.0, довжина = 5.0
Площа кімнати: 25.0

Метод info2 SimpleRoom2
Кімната: ширина = 5.0, довжина = 5.0, висота= 3.0
Площа кімнати: 25.0
Об'єм кімнати: 75.0
```

Зверніть увагу, що ми без проблем через об'єктну змінну класу *SimpleRoom2* викликаємо метод *info* класу *SimpleRoom*:

```
s2.info();
```

Тож крім полів дочірньому класу також доступні неприватні методи батьківського класу.

Насправді, у дочірньому класі (*SimpleRoom2*) можна було б створити однойменний клас *info* і він би замістив відповідний метод батьківського класу. Даний механізм так і називається *заміщення або перевизначенням методів* (англ. method overriding).

Інколи може виникнути необхідність *заборонити можливість здійснення успадкування*. Тобто можливість створення нових класів, на базі певного класу. Тоді клас можна оголосити як *final*. Також можна *заборонити заміщення методу у класах нащадках*, використавши при оголошенні методу той же модифікатор *final*. Як уже мабуть ви знаєте, цей же модифікатор *final* також застосовується для оголошення констант — змінних, які ініціалізуються лише раз.

Наприклад, в Java фінальним оголошено клас *String*.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Змістовий модуль 4. Розробка об'єктно-орієнтованих програм мовою JAVA та їх документування

Тема 4.1. **JAVA як платформа. IDE NetBeans**

Номер заняття: 21

Мета: познайомити студентів з інтелектуальними можливостями Netbeans та інструментами продуктивності розробника, які надає це середовище розробки.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1-11: https://college.page.link/BFSM

Короткі теоретичні відомості:

NetBeans IDE – вільне *інтегроване середовище розробки* (IDE) для мов програмування Java, JavaFX, C/C++, PHP, JavaScript, HTML5, Python, Groovy. Середовище може бути встановлене і для підтримки окремих мов, і у повній конфігурації. Середовище розробки NetBeans за замовчуванням підтримує розробку для платформ J2SE і J2EE.

Поширюється у вихідних кодах під ліцензією *Apache License*. Проект NetBeans IDE підтримувався і спонсорувався фірмою Sun Microsystems і після придбання Sun — Oracle. У жовтні 2016 року Oracle передав NetBeans у власність Apache Software Foundation, яка займається розробкою і підтримкою проекту.

NetBeans IDE доступна для платформ Microsoft Windows, GNU/Linux, FreeBSD, і Solaris (як SPARC, так x86). Для інших платформ доступна можливість зібрати NetBeans самостійно з вихідних кодів.

За якістю і можливостям останні версії NetBeans IDE змагається з найкращими інтегрованими середовищами розробки для мови Java, підтримуючи рефакторинг, профілювання, виділення синтаксичних конструкцій кольором, автодоповнення мовних конструкцій на льоту, шаблони коду та інше.

Розробку середовища NetBeans розпочато в 1996 під назвою Xelfi (гра букв на основі Delphi), як проект студентів зі створення Java IDE під керівництвом факультету математики і фізики Карлова Університету в Празі. У 1997 році Роман Станек сформував компанію навколо проєкту і став випускати комерційні версії середовища NetBeans до передачі всіх прав на IDE корпорації Sun Microsystems в 1999 році. Sun відкрила сирцеві коди середовища розробки NetBeans IDE в червні наступного року. Відтоді спільнота NetBeans постійно розвивається і росте завдяки людям і компаніям, що використовують і підтримує проект.

Версія 12, опублікована 09 червня 2020, має підтримку найновіших функцій Java, підтримку PHP 7.4, виправлені помилки та покращене відображення на моніторах hiDPI Windows. Версія 12.02, опублікована 07 грудня 2020. В цій версії додана підтримка PHP 8.0, була вилучена підтримка Nashorn.

NetBeans IDE підтримує *плагіни*, дозволяючи розробникам розширювати можливості середовища. У версії NetBeans IDE 6.0 підтримуються засоби застосунків на J2ME, UML, SOA, мова програмування Ruby (включаючи підтримку Ruby on Rails). Проте через невелику популярність NetBeans серед розробників Ruby у версії 7.0 підтримка Ruby та Ruby on Rails скасована, а наявні ресурси залучені для розвитку Java SE 7 та JDK 7.

У версії IDE 7.0 забезпечена підтримка Oracle Database, підтримуються наступні *зовнішні компоненти*:

- Java EE та J2EE
- Java Card SDK
- Struts
- Spring
- Hibernate
- Java API for RESTful Web Services (JAX-RS)
- Java Wireless Toolkit для CLDC
- Системи відстеження помилок Bugzilla і Jira
- C/C++/Fortran
- PHP
- Groovy
- Grails

- Apache Ant
- Apache Maven
- Системи управління версіями: CVS; Subversion; Mercurial; ClearCase; Git;
- GlassFish Server Open Source Edition
- Oracle WebLogic Server
- Сервери застосунків: GlassFish Enterprise Server, Tomcat, JBoss

Додаткові матеріали:

1. NetBeans – Вікіпедія: https://uk.wikipedia.org/wiki/NetBeans
2. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
3. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
4. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/

Тема 4.3. **Об’єктно-орієнтоване програмування. Документування програм**
Номер заняття: 22-23

Мета: познайомити студентів з можливостями документування програм за допомогою Javadoc-коментарів та сторонніми інструментами генерації проектної документації.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв’язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/2MKa

частина 2: https://youtu.be/NRhsIuPs8BI

Короткі теоретичні відомості:

Коментарі Java – це примітки у файлі коду Java, які ігноруються компілятором та механізмом виконання. Вони використовуються *для анотації коду з метою уточнення його дизайну та призначення*. Ви можете додати необмежену кількість коментарів до файлу Java, але є кілька "найкращих практик", яких слід дотримуватися при використанні коментарів.

Як правило, коментарі коду – це коментарі «реалізації», які *пояснюють вихідний код*, такі як описи класів, інтерфейсів, методів та полів. Зазвичай це кілька рядків, написаних над кодом Java або поруч із ним, щоб пояснити, що він робить.

Інший тип коментарів Java – це *коментар Javadoc*. Коментарі Javadoc трохи відрізняються за синтаксисом від коментарів до реалізації і використовуються програмою javadoc для генерації документації Java HTML.

Це хороша практика, коли у вас є звичка додавати коментарі Java до свого вихідного коду, щоб покращити його *читабельність та зрозумілість* для себе та інших програмістів. Не завжди миттєво ясно, що виконує розділ коду Java. Кілька пояснювальних рядків можуть суттєво скоротити час, необхідний для розуміння коду.

Коментарі в коді Java доступні лише людям. Компілятори Java не дбають про них, і при складанні програми вони просто пропускають їх. *Кількість коментарів у вихідному коді не вплине на розмір та ефективність вашої скомпільованої програми*.

В Java існує *три типи коментарів*:

– *однорядкові* з застосуванням двох похилих рисок (//). Все що йде в рядку після них вважається коментарем.

// це коментар

System.out.println("Вивчаємо коментарі"); // виводимо на екран

– *багаторядкові* – текст можна вказувати в кілька рядків, також зручно, якщо потрібно усунути кілька рядків тексту під час зневадження програми. Для відкриття коментаря використовується /*, а для закриття */. Все що між ними вважається коментарем. Вкладення багаторядкового коментаря в інший багаторядковий коментар не дозволяється.

/

Це багаторядковий коментар

System.out.println("Це не буде виведено на екрані");

System.out.println("І це також не буде виведено на екрані");

**/*

System.out.println("А це буде виведено");

– *для автоматичної генерації документації* – розпочинаються символами /** і закінчуються */. Використовуються для додавання додаткової інформації про класи та методи, їх призначення, параметри. На основі цієї інформації можна згенерувати документацію за розробленими вами класами та їх методами. До того ж, ця документація буде подібною до офіційної документації Java.

Javadoc – генератор документації в HTML-форматі з коментарів вихідного коду на Java від Sun Microsystems. Цей формат був обраний для забезпечення можливості зв'язати воєдино пов'язані документи за допомогою посилань. Коментарі javadoc стали «де

факто» стандартом для документування створених Java-класів. Більшість середовищ розробки, таких як Eclipse та Netbeans автоматично генерують документацію за допомогою javadoc.

Javadoc також забезпечує інтерфейс для створення доклетів та теглетів, що надають можливість аналізувати структуру Java-програми.

Javadoc-коментар – це так званий коментар документації. В коді він виділяється такою конструкцією:

```
/**
 * Так можна коментувати змінні класу.
 */
```

Ці коментарі дають можливість додавати в програму інформацію про неї, яка пізніше може бути використана утилітою javadoc (входить до складу Java Development Kit) для створення HTML-файлів. Коментарі документації можна використовувати при коментуванні:

- полів (змінних);
- методів;
- конструкторів:
- класів;
- інтерфейсів;
- пакетів.

Варто відмітити, що в будь-якому разі коментарі повинні знаходитися *перед документованим об'єктом*.

В коментарях також можна використовувати і стандартні HTML теги, наприклад . Проте на використання деяких з них накладається заборона, наприклад заголовки порушують зовнішній вигляд HTML-файлу, сформованого за допомогою утиліти javadoc.

В документації можна також використовувати спеціальні *дескриптори*, призначені для вказування утиліті javadoc певної інформації. Їх поділяють на:

- автономні (починаються з символу @, наприклад *@see*);
- вбудовані (починаються з символу {, наприклад *{@code}*).

Відмінність цих дескрипторів полягає у тому, що автономні мають використовуватись у власному рядку, в той час як вбудовані можуть бути використані всередині великого опису.

Приклад документування пакету:

```
/**
 * Пакет включає класи для використання XML потоків в програмах.
 *
 * @author Mir4ik
 * @version 1.0 10/06/12
 */
package XMLTools;
```

Приклад документування методу:

```
/**
 * Метод формує дерево XML елементів.
 * <p>
 * <strong>Увага:</strong> Дані поза кореневим тегом XML буде втрачено!
 * Використовуйте спеціальне поле для коментаря в <code>XMLElement</code>,
 * що можна задати методом {@link XMLElement#setComment(String)}.
 *
 * @return   найвищий <code>XMLElement</code> в сформованому дереві
 * @throws XMLStreamException якщо під час парсингу виникла помилка
 * @throws IOException якщо виникла виняткова ситуація при читанні даних
 */
public XMLElement readRootTag() throws XMLStreamException, IOException {
        // код методу
}
```

Doxygen – кросплатформна система документування початкового коду програм, яка підтримує C++, Ci, Objective-C, Python, Java, IDL, PHP, Perl, C#, Фортран, VHDL і, частково, D.

Doxygen генерує документацію на основі набору вихідних текстів і також може бути налаштований для вилучення структури програми з недокументованих вихідних текстів. Можливе складання графів залежностей програмних об'єктів, діаграм класів та вихідних кодів з гіперпосиланнями.

Doxygen має вбудовану підтримку генерації документації в форматі *HTML, LaTeX, man, RTF і XML*. Також результати його роботи можуть бути легко конвертовані в CHM, PostScript, PDF.

Для HTML-представлення документації, що розміщується на web-серверах, існує зручний спосіб організації *пошуку* (за допомогою створюваного Doxygen'ом PHP-модуля) і посилань на зовнішню документацію.

Doxygen використовується в багатьох проектах, в тому числі KDE, Pidgin, Torque Game Engine, AbiWord, Mozilla, FOX toolkit, Crystal Space, Drupal. Є вбудована підтримка в KDevelop.

Doxygen – консольна програма в стилі класичної Unix. Вона працює подібно компілятору, аналізуючи вихідні тексти і створюючи документацію. Додаткові параметри для створення документації можуть читатись із конфігураційного файлу, що має простий текстовий формат.

Для спрощення маніпуляцій з конфігураційним файлом (який містить досить багато налаштувань), існує кілька програм з графічним інтерфейсом: програма *doxywizard* (реалізована з використанням Qt-3) поставляється разом з Doxygen; програма Doxygat заснована на Qt версії 4. Пізніше doxywizard був переписаний на Qt-4 і проект Doxygate був закритий.

Додаткові матеріали:

1. Doxygen – Вікіпедія: https://uk.wikipedia.org/wiki/Doxygen
2. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
3. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
4. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
5. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Змістовий модуль 5. Основні можливості платформи JAVA

Тема 5.1. **Нові ідентифікатори, ключові слова та типи даних**

Номер заняття: 25

Мета: нагадати студентам ключові слова та типи даних, типові для С-подібних мов програмування; сформувати уявлення про стандарти в галузі написання коду, зокрема стосовно ідентифікаторів, та важливість їх дотримання; розповісти про особливості синтаксису Java.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/mmUH

Короткі теоретичні відомості:

Ідентифікатори використовуються для *іменування змінних, методів, класів*. В мові Java ідентифікатор складається з будь-якої послідовності:

– рядкових і прописних букв латинського алфавіту;
– цифр від '0' до '9';
– символу підкреслення '_';
– символу грошової одиниці (в особливих випадках).

Ідентифікатор обов'язково має *починатися з літери*. Мова Java розрізняє рядкові та прописні літери як різні. Це означає, що ідентифікатор з іменем MAX відрізняється від ідентифікатора з іменем Max – це два різних імені.

Приклади імен:

x

weight

a1

array1

MyClass

Max

a1b2c3

Ключові слова – це зарезервовані мовою Java слова. Ключовими словами можуть бути оператори, інструкції, твердження, що складають основу мови Java. *Ключові слова не можна використовувати в якості імен ідентифікаторів, змінних, класів, методів тощо*.

У мові Java визначено наступні ключові слова:

abstract	*continue*	*for*	*new*	*switch*
assert	*default*	*goto*	*package*	*synchronized*
boolean	*do*	*if*	*private*	*this*
break	*double*	*implements*	*protected*	*throw*
byte	*else*	*import*	*public*	*throws*
case	*enum*	*instanceof*	*return*	*transient*
catch	*extends*	*int*	*short*	*try*
char	*final*	*interface*	*static*	*void*
class	*finally*	*long*	*strictfp*	*volatile*

Примітивні типи даних можна використовувати безпосередньо або для створення власних типів класів. Примітивні типи служать основою для всіх інших типів даних.

В Java визначені 8 примітивних (простих) типів даних:

- *byte*
- *short*
- *int*
- *long*

– *char*
– *float*
– *double*
– *boolean*

Примітивні типи являють собою *одинарні значення*, вони не є складними об'єктами. Примітивні типи є *простими типами*. На основі примітивних типів можна створювати власні складні типи.

Примітивні типи доцільно використовувати в програмах, тому що це суттєво збільшує *її продуктивність*. Якщо реалізувати примітивні типи у вигляді об'єктів, то це призвело би до суттєвого зниження продуктивності. Примітивні типи мають строго визначений діапазон допустимих значень.

Цілочисельні типи даних являють собою цілі значення, які можуть бути як додатними так і від'ємними. До цілочисельних типів даних належать такі типи:

– *byte*
– *short*
– *int*
– *long*

В Java не існує беззнакових цілочисельних типів даних.

Довжина цілочисельних типів даних наступна:

– *byte – 8 розрядів (біт);*
– *short – 16 розрядів;*
– *int – 32 розряди;*
– *long – 64 розряди.*

В Java до *типів з плаваючою точкою* (дійсних типів) належать типи:

– float
– double

До типу *float* належать числові значення одинарної точності. До типу *double* належать числові значення подвійної точності. Змінна типу *float* займає 32 біти в пам'яті. Змінна типу *double* займає 64 біти.

В Java для представлення *символів* введено тип даних *char*.

Змінні типу *char* займають в пам'яті комп'ютера 16 біт. Це пов'язано з тим, що для представлення символів *char* в Java використовується кодування Юнікод (Unicode).

Діапазон допустимих значень цього типу складає від 0 до 65535. Використання кодування Unicode пов'язане з тим, що програми на Java використовуються у всьому світі.

Тип *boolean* призначений для зберігання *логічних значень*. Змінні типу *boolean* можуть приймати тільки два значення: *true* або *false*.

Між примітивними типами і типами-посиланнями існують наступні відмінності:

- змінна примітивного типу не є змінною-посиланням (об'єктом);
- змінна примітивного типу є "автоматичною" і зберігається в стеку. Об'єкт (змінна) зберігається в "купі". Тому продуктивність роботи зі змінними примітивного типу вища;
- при оголошенні змінної примітивного типу не потрібно виділяти пам'ять оператором *new*;
- змінна примітивного типу напряму зберігає значення на відміну від змінної-посилання.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 5.2. **Поглиблене використання виразів, керування виконанням програми**
Номер заняття: 27-28

Мета: нагадати студентам основні оператори та математичні функції, пріоритети операцій, конструкції для керування виконання програми, продемонструвати їх реалізацію в Java.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://youtu.be/NsnisqveUdg

частина 2: https://youtu.be/B-xdu83f8mY

частина 3: https://youtu.be/Q6z2MO_zsmg

частина 4: https://youtu.be/ikLPx4mFo7M

частина 5: https://youtu.be/xHyshvY_lTo

частина 6: https://youtu.be/xPo1qeLDFXc

частина 7: https://youtu.be/E0ABoPd6lvU

частина 8: https://youtu.be/T-01-ax4FTI

частина 9: https://youtu.be/KdlP8mLarMg

частина 10: https://youtu.be/bL_KIlKrEV8

Короткі теоретичні відомості:

Вирази є важливими будівельними блоками будь-якої програми Java, яка зазвичай створюється для створення нового значення, хоча іноді вираз присвоює значення змінній. Вирази будуються з використанням значень, змінних , операторів та викликів методів.

З точки зору синтаксису мови Java, *вираз є подібним до речення в англійській мові*, яке відображає конкретне значення. При правильній пунктуації воно іноді може стояти самостійно, хоча це може бути і частиною речення. Деякі вирази прирівнюються до висловлювань самі по собі (додаючи крапку з комою в кінці), але частіше вони складають *частину висловлювання*.

Наприклад,

*(a * 2)* – це вираз.

*b + (a * 2);* є висловлюванням.

Можна сказати, що вираз є реченням, а висловлювання – повним реченням, оскільки воно утворює повну одиницю виконання Однак вираз не повинен містити кілька виразів. Ви можете перетворити простий вираз на твердження, додавши крапку з комою:

*(a * 2);*

Хоча вираз часто дає результат, але не завжди. У Java існує *три типи виразів*:

- Ті, що *мають вартість*, тобто результат: *(1 + 1)*
- Ті, що *присвоюють змінну*: (v = 10)
- Ті, *які не мають результату, але можуть мати "побічний ефект"*, оскільки вираз може включати широкий спектр елементів, таких як виклики методів або оператори приросту, які змінюють стан (тобто пам'ять) програми.

У виразах часто присутні *математичні функції*. Для використання математичних функцій слід скористатися бібліотекою *Math*.

Основні функції і константи класу Math:

Константа або функція	Пояснення
E	Константа e
PI	Константа
abs( )	Модуль числа або виразу
sqrt( )	Квадратний корінь
sin( )	Синус
cos( )	Косинус
tan( )	Тангенс
log( )	Натуральний логарифм (ln x)
log10( )	Десятковий логарифм (lg x)
pow( )	Піднесення до степеня
exp( )	Експонента (e f(x))
asin( )	Арксинус
acos( )	Арккосинус
atan( )	Арктангенс
random( )	Функція випадкового дробового числа з проміжку [0; 1)

Арифметичні оператори використовуються в математичних виразах так само як і в алгебрі і представлені в таблиці:

Оператор	Операція	Оператор	Операція
+	Додавання	+=	Додавання з присвоєнням
-	Віднімання (а також унарний мінус)	-=	Віднімання з присвоєнням
*	Множення	* =	Множення з присвоєнням
/	Ділення	/=	Ділення з присвоєнням
%	Залишок ділення по модулю	%=	Залишок ділення по модулю з присвоєнням
++	Інкремент (збільшення на 1)	--	Декремент (зменшення на 1)

Java визначає ряд *порозрядних (побітових) операторів*, що можуть застосовуватися до цілочисельних типів, *long*, *int*, *short*, *char* та *byte*. Ці оператори виконують дії над окремими розрядами(бітами) їхніх операндів. Наступна таблиця наводить список даних операторів:

Оператор	Опис	Оператор	Опис
~	Порозрядне одномістне НІ (NOT)	&	Порозрядне І (AND)
|	Порозрядне АБО (OR)	^	Порозрядне виняткове АБО (XOR)
>>	Зсув вправо	>>>	Зсув вправо із заповненням нулями
<<	Зсув вліво	&=	Порозрядне І (AND) з присвоєнням
|=	Порозрядне АБО (OR) з присвоєнням	^=	Порозрядне виняткове АБО (XOR) з присвоєнням
>>=	Зсув вправо з присвоєнням	>>>=	Зсув вправо з заповненням нулями і присвоєнням

Булеві логічні оператори застосовуються для операндів типу boolean, результатом є булева величина.

Оператор	Опис
&	Логічне І (AND)
|	Логічне АБО (OR)
^	Логічне виняткове АБО (XOR)
||	Коротке АБО
&&	Коротке І
!	Логічне одиничне НІ (NOT)
&=	І з присвоєнням
|=	АБО з присвоєнням
^=	Виняткове АБО з присвоєнням
==	Рівність
!=	Не рівність
?:	Тернарний if-then-else (якщо-тоді-інакше)

В операторів програмування як і в математичних операторів є своя *першочерговість (пріоритет).* Наприклад, операція множення виконується раніше за додавання. І як і в математиці для зміни першочерговості операцій використовуються дужки. Якщо ви невпевнені в якій послідовності будуть виконуватися дії, то використайте їх для прямого вказання порядку виконання операцій у виразі. Крім того використання дужок не впливає на ефективність виконання трансльованої програми і покращує читабельність програми.

Наступна таблиця вказує на першочерговість операцій – зверху донизу.

Першочерговість операторів від найвищого (зверху) до найнижчого (знизу)
( ) [ ] .
++ – – ~ !
* / %
+ –
>> >>> <<
> >= < <=
== !=
&
^
|
&&
||
?:
= op=

Java, як і інші мови програмування, підтримує умовні інструкції та цикли, що визначають порядок виконання інструкцій у програмі. В англійській мові для цього поняття застосовують термін control flow — керування потоком.

Перед тим як знайомитись з керуючими структурами, спочатку необхідно ознайомитися з блоками. *Блок або складений оператор* – це будь-яка кількість простих інструкцій, які оточені парою фігурних дужок. Блок визначає область видимості ваших змінних. Блоки можуть бути вкладені в середину інших блоків. Ви вже зустрічалися з блоками при створенні найпростіших програм у методі *main()*.

Проте *не можна визначати однакові змінні в двох вкладених блоках* (на відміну від C++, де це можливо).

Умовний оператор в Java має форму:

if (умова) інструкція;

Умова має бути оточена дужками і, якщо, умова задовольняється (true) буде виконана інструкція за умовою, інакше вона не буде виконана, а буде виконана наступна інструкція після умовної інструкції.

Зазвичай, необхідно виконати не одну інструкцію, в такому разі інструкції розміщають у блоці:

```
if (умова){
  інструкція 1;
  .....
  інструкція n;
}
```

В такому разі при істинності умови, виконуються всі інструкції у блоці, якщо умова невірна, то виконується наступна інструкція після закриваючої дужки блоку. Якщо ж необхідно здійснити певну дію в разі не виконання умови, то в такому разі застосовують умовну інструкцію наступного виду:

```
if (умова) інструкція1; else інструкція2;
```

Інструкції if можуть йти одна за одною без використання else:

```
if (x <= 0) if (x == 0) sign = 0; else sign = -1;
```

Для того, щоб програма була читабельнішою бажано застосовувати фігурні дужки:

```
if (x <= 0) { if (x == 0) sign = 0; else sign = -1; }
```

Вони нічого не змінюють, але вираз стає зрозумілішим. Інструкція чи блок інструкцій виконується лише в разі виконання усіх умов.

Щоправда дану інструкцію можна також переписати ускладнивши умову використавши булевий оператор і (&&):

```
If (x <= 0 && x==0) sign = 0; else sign=-1;
```

Можна також використовувати повторюваність інструкцій *if….else*.

Це дає можливість перевірити ряд умов, якщо попередні умови не виконуються.

Цикли – це послідовність інструкцій, які можуть повторно виконуватись певну кількість раз в залежності від заданої в програмі умови. Розрізняють цикли з передумовою, з післяумовою та з лічильником.

Цикл while (перекладається як «доки») – це цикл з передумовою, тіло якого (тобто інструкція або блок інструкцій) виконується, якщо умова істинна. Якщо умова з самого початку хибна, то цикл не виконається жодного разу.

Загальний вигляд:

while (умова) інструкція;

Якщо немає фігурних дужок, то перша інструкція, яка йде після оголошення циклу, вважається *тілом циклу*. Всі інші інструкції знаходяться поза циклом. Якщо в циклі повинні виконуватись кілька інструкцій, то необхідно використати фігурні дужки. Вони також можуть використовуватись при одній інструкції заради кращого візуального розуміння коду.

```
while (умова){
  інструкція 1;
  ...
  інструкція N;
}
```

Якщо необхідно, щоб умова виконувалася хоча б один раз можна скористатися *циклом з післяумовою do/while*:

do інструкція while (умова);

Цикл for (цикл з лічільником) – доволі часто вживаний цикл. Він застосовується при необхідності виконати інструкції певну кількість раз з одночасним збільшенням або зменшенням певної змінної. Часто використовується для здійснення перебору певних масивів даних, зокрема, також для сортування масивів. Приклад використання:

```
for (int i = 1; i <= 10; i++) {
      System.out.println(i);
}
```

Наведений вище приклад виведе на консолі в стовпчик числа від 1 до 10. Як бачимо в умові циклу перший слот відводиться для ініціалізації змінної, причому оголосити змінну можна і в іншому місці. Другий слот – для умови, яка перевіряється перед виконанням ітерації, третій слот – вказує як модифікувати змінну-лічильник. Тобто в наведеному прикладі при кожному виконанні ітерації, лічильник "i" буде збільшуватися на одиницю поки не стане рівним десяти.

Найчастіше даний цикл використовується для перебору елементів масиву. *Масив* – це впорядкований набір даних одного типу. Найпростіший масив можна оголосити та

ініціалізувати таким чином: *int a[]={1, 5, 6, 1, 3};*. Для того, щоб звернутися до певного елементу масиву використовуються квадратні дужки з відповідним індексом елементу. Наприклад *a[3]* – звернення до четвертого елементу масиву (номери елементів відраховуються з нуля).

Починаючи з java SE 5.0 в мові з'явився новий цикл – *покращений for*, призначення якого є перебір елементів масиву або подібних до масиву типів даних (колекції).

Загальний вигляд циклу наступний:

```
for (type var : arr) {
   //тіло циклу
}
```

Наприклад, вивести елементи масиву, можна таким чином:

```
for (int element : a)
   System.out.println(element);
```

Використання цього циклу, дозволяє уникнути проблем пов'язаних з помилками при заданні умови в класичному циклі for. В інших мовах програмування цикл такого виду так і називається *foreach*, проте, щоб уникнути необхідності значних змін в пакетах, в java пішли простішим шляхом і *перевантажили цикл for*.

В java відсутня інструкція *goto*, яка дозволяла переходити в будь-яке місце в програмі. Її використання давно вважається *поганим стилем програмування*, оскільки робить текст програми заплутаним. Про це, зокрема, писав ще *Дональд Кнут* і він же зазначав, що інколи все таки корисно її застосовувати, щоб припинити виконання певного методу, циклу чи блоку і вийти за їхні межі. Для цієї мети в java існують спеціальні інструкції. Зокрема, інструкцію *break* можна використати для *передчасного виходу з циклу*. Якщо ми використовуємо вкладені цикли (один цикл в іншому), то інструкція *break* припинить цикл, в якому вона знаходиться. Якщо вона знаходиться у внутрішньому циклі, то він припинить своє виконання, а зовнішній же цикл буде виконуватись і далі. Якщо потрібно повністю припинити виконання і внутрішнього і зовнішнього, то використовується інструкція *break* з міткою.

Якщо ж нам не потрібно виходити з циклу, а лише *припинити певну його ітерацію*, то для цієї використовується інструкція *continue*, яка переносить порядок виконання до заголовку інструкції.

Щоправда *завжди можна обійтися без цих інструкцій змінивши логіку програми*. Деякі з програмістів уникають використовувати інструкції break та continue.

Інструкція if/else може бути доволі громіздкою, якщо необхідно здійснити *множинний вибір* з багатьох альтернатив. Тож як і в C/C++ в java існує інструкція *switch*, яка здійснити вибір з багатьох варіантів. Щоправда вона дещо незграбна і деякі програмісти вважають за краще уникати її використання.

До версії Java 7, яка вийшла у 2011 році, case мітка мала бути лише цілим числом або нумерованою константою. *Починаючи із Java 7 можна перевіряти таким чином на рівність також рядки*.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 5.3. Поглиблене використання масивів
Номер заняття: 30

Мета: нагадати студентам основні прийоми роботи з масивами та операції, які вони можуть використовувати; познайомити їх з класом Arrays та його методами.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/xqzo

Короткі теоретичні відомості:

Масив – це впорядкований набір однотипних елементів, на які посилаються по спільному імені. Це доволі зручний засіб групування інформації. Масиви можна створювати з елементів будь-якого типу. До конкретного елементу в масиві звертаються за індексом (номером). Вони можуть бути як *одновимірні* так і *багатовимірні*.

Одновимірні масиви – це список однотипних елементів. Загальний формат оголошення такого масиву:

тип-елементів назва-масиву[];

Наприклад

int month_days[]; // масив цілих чисел

Існує також інша форма оголошення масиву:

int[] month_days;

Проте для того, щоб масив почав існувати необхідно виділити під нього пам'ять, за допомогою операції *new*.

назва-масиву = new тип-елементів [розмір];

де розмір - планована кількість елементів у масиві.

month_days = new int[12];

або зразу ж:

int month_days[] = new int[12];

Таким чином відбувається виділення пам'яті під масив і ініціалізації елементів масиву нулями. В подальшому можна напряму звертатися до елементів масиву вказуючи індекс у квадратних дужках. Нумерація елементів в масиві в java відбувається з нуля. Тобто в наведеному прикладі звернення до першого(нульового) елемента – *month_days[0]*, а до останнього – *month_days[11]*. Java не дозволить програмі звернутися поза межі масиву, щоправда помилка буде вказана лише на етапі виконання програми через викидання винятку (виключення, exception).

month_days[5] = 30;

System.out.println(month_days[5]);

Масиви також можна ініціалізувати зразу ж при їхньому оголошенні, не використовуючи операції *new*, аналогічно як це відбувається при роботі з простими типами даних.

Як бачимо спочатку змінній *max* присвоюється значення нульового елементу масиву, після чого в циклі іде послідовне порівняння з кожним наступним числом до останнього. Якщо при порівнянні чергове значення в масиві більше за максимальне в змінній *max*, то змінній *max* присвоюється дане значення. Як Ви уже зрозуміли, по закінченню циклу у змінній *max* міститиметься максимальне значення, яке і буде виведене на консоль:

Максимальне число в масиві: 3.2

В Java масиви є *об'єктами* (про об'єкти ми вже говорили раніше), що забезпечує деяку додаткову функціональність масивам. Зокрема, можна дізнатися довжину масиву наступним чином *array.length*. Для вищенаведеного прикладу можна замінити рядок з циклом таким чином:

for (int i =0; i < array.length; i++){

Багатовимірні масиви по суті – це *масив масивів*. Робота з багатовимірними масивами подібна до роботи з одновимірними. Відмінність лише в тому, що використовуються додаткові квадратні дужки. Переважно використовуються двовимірні масиви, які служать для роботи з табличними даними та трьохвимірні масиви. Двовимірний масив та трьохвимірний, можна оголосити наступним чином:

int twoD[][] = new int [4][5]; //створення масиву 4x5

int threeD[][][] = new int[5][5][5]; //створення масиву 5x5x5

Для двовимірного лівий індекс означає номер рядка, а правий номер стовпця. Це можна уявити наступним чином:

[0,0][0,1][0,2][0,3][0,4]

[1,0][1,1][1,2][1,3][1,4]

[2,0][2,1][2,2][2,3][2,4]

[3,0][3,1][3,2][3,3][3,4]

Трьохвимірний масив можна уявити у вигляді куба. Крім номера рядка і номера стовпця, додається ще індекс елемента вглибину.

В Java можна створити двовимірні масиви з різною кількістю елементів в рядках (*зазубрені або зубчасті масиви, jagged arrays*).

int twoD[][] = new int[5][]; //створюємо двовимірний масив з 5-ма рядками

twoD[0] = new int[5]; // виділяємо пам'ять для 5-ти елементів нульового рядка

twoD[1] = new int[4]; // перший рядок матиме 4-ри елементи

twoD[2] = new int[3]; // другий - 3

twoD[3] = new int[2]; // третій - 2

twoD[4] = new int[1]; // четвертий – 1

Використання таких *нерівних (нерегулярних) масивів* не рекомендується, оскільки з ними важче працювати і можна припуститися ряд помилок, але в деяких ситуаціях можуть бути доволі корисними.

Також в Java існує спеціальний клас для обробки масивів – *java.util.Arrays*.

Утилітний клас *Arrays* містить статичні методи, що реалізують популярні алгоритми обробки масивів. Нижче наведена група методів цього класу для масивів типу *double*.

Такі самі методи є і для інших примітивних типів і є лише один метод для сортування будь яких об’єктів. Цей клас наочно демонструє переваги об’єктно-орієнтованого програмування.

public static double[] copyOf(double[] original, int newLength)

public static int binarySearch(double[] a, double key)

public static void fill(double[] a, double val)

public static void sort(double[] a)

public static String toString(double[] a)

public static void setAll(double[] array, IntToDoubleFunction generator)

Методи *setAll* та *sort* мають аналоги для паралельної обробки масивів *setParallelAll* та *parallelSort*. Але використання цих методів доцільно тільки для дуже і дуже великих масивів.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Змістовий модуль 6. Проектування та реалізація архітектури програм. Забезпечення якості програмних продуктів

Тема 6.1. **Проектування ієрархії класів. Використання UML. Особливості створення класів**

Номер заняття: 32-33

Мета: нагадати студентам основні принципи ООП зокрема успадкування; познайомити студентів з засобами побудови діаграм, кодогенерації та зворотного інжинірингу для Java.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

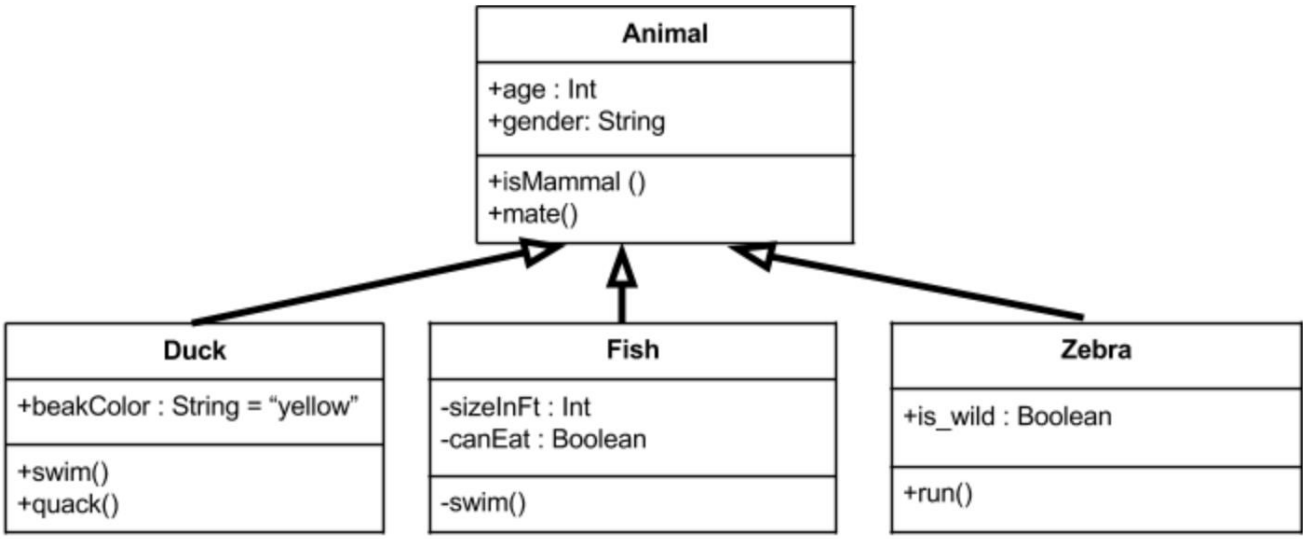
частина 1: https://college.page.link/FWNs

Короткі теоретичні відомості:

Успадкування або *наслідування* (англ. inheritance) – це ще один важливий механізм об'єктно-орієнтованого програмування, який дозволяє створювати нові класи на основі вже існуючих. Клас на базі якого створюється підклас називають надкласом, *суперкласом* або ж батьківським класом. Новий клас, що розширює (extends) батьківський клас називають *підкласом* або дочірнім класом. На відміну від C++ де клас може мати кілька батьківських класів, *мова програмування Java підтримує лише одинарне успадкування*, тобто може бути лише один безпосередній надклас. У надкласу звичайно може бути свій надклас, проте також лише один безпосередній. Множинне успадкування доволі складне у застосуванні і вимагає обережного підходу, тому творці Java вирішили відмовитися від нього.

Для графічного представлення ієрархії успадкування зазвичай використовують *діаграму класів UML* – статичне представлення структури моделі. Вона відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення.

Діаграма класів може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм. Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні стосунки. Приклад діаграми класів:



Для створення діаграм класів прямо в середовищі NetBeans можна скористатись плагіном *easyUML*, який підтримує також кодогенерацію та зворотний інжиніринг.

Генерація коду або *кодогенерація* – частина процесу компіляції, коли спеціальна частина компілятора, кодогенератор, конвертує синтаксично коректну програму або діаграму класів UML в послідовність інструкцій, які можуть виконуватися на машині. Слід зазначити, що на основі діаграми класів можна отримати лише *каркасний код*, який вимагає доопрацювання.

Зворотна розробка або *зворотний інжиніринг* – дослідження деякого пристрою чи програми з метою розуміння принципів роботи досліджуваного об'єкта. Найчастіше використовується з метою створення об'єкта, за функціональністю аналогічного досліджуваному але без точного копіювання його функцій.

Зазвичай використовується у випадках, коли розробник оригінального пристрою чи програми не надає точних відомостей про алгоритми функціонування виробу, або будь-яким чином намагається завадити його використанню.

В нашому випадку зворотний інжиніринг полягає в *побудові діаграми класів за існуючим кодом*. Такі діаграми можуть слугувати засобом контролю – відслідковування прогресу проекту.

Ще один доступний інструмент для UML-проектування, кодогенерації та зворотного інжинірингу – *StarUML* –інструмент, який було ліцензовано під модифікованою версією GNU GPL до 2014 року, коли переписана версія 2.0.0 була випущена для бета-тестування під власною ліцензією.

У 2014 році була випущена переписана версія як запатентоване програмне забезпечення. Заявленою метою проекту було замінити комерційні програми, такі як Rational Rose та Borland Together. StarUML підтримує більшість типів діаграм, зазначених у UML 2.0 . Починаючи з версії 4.0.0 (29 жовтня 2020 р.), вона включає оглядові діаграми та діаграми взаємодії. За допомогою плагінів StarUML підтримує кодогенерацію та зворотний інжиніринг для таких мов як C/C++, C# та Java.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 6.2. **Рефакторинг. Типові архітектурні рішення та антипатерни**
Номер заняття: 34-35

Мета: познайомити студентів з теоретичними засадами рефакторингу; розповісти про патерни та антипатерни; продемонструвати вбудовані в NetBeans інструменти рефакторингу.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/Kdhk

Короткі теоретичні відомості:

Рефакторинг – перетворення програмного коду, зміна внутрішньої структури програмного забезпечення для полегшення розуміння коду і легшого внесення подальших змін без зміни зовнішньої поведінки самої системи. Слово «рефакторинг» пішло від терміну «факторинг» в структурному програмуванні, який означав декомпозицію програми на максимально автономні та елементарні частини.

Існує міф про те, що правильно організований процес розробки продукту методично дотримується поставлених вимог, визначає однозначний, стабільний список обов'язків програми, і при цьому програмний код може бути написаний майже лінійно – від початку до закінчення, кожна ділянка – один раз написана, відтестована й забута. Згідно з цим міфом, єдиний випадок, коли наявний код може змінюватись, – це в процесі підтримки і адміністрування програми, коли початкова версія продукту уже здана замовнику.

Однак реальність є дещо інакшою. Насправді *код еволюціонує в процесі розробки продукту*. Як правило, кодування, відлагодження та модульне тестування займають в середньому 30–65% зусиль від загального часу існування проекту (залежно від величини проекту). Навіть в ході добре організованих проектах вимоги змінюються в середньому на 1–4% за місяць, що неминуче спричиняє зміни до програмного коду – як дрібні, так і досить серйозні.

Також, на відміну від старіших методик розробки програмних продуктів, де основний акцент ставився на мінімізації змін до коду, сучасна методика вбачає *великий потенціал у внесенні змін*. Вона є сфокусованою на коді (code-centered) і під час розробки можна очікувати, що код буде вдосконалюватися більше, ніж зазвичай.

Підстави для проведення рефакторингу:

- *Код дублюється* («Copy And Paste is a Design Error»). Це нерідко призводить до необхідності вносити однакові зміни до кількох скопійованих ділянок коду, що не відповідає принципам «DRY» (Don't Repeat Yourself).
- *Підпрограма занадто довга*. Хоча питання, яку максимальну довжину може мати підпрограма, є досить суперечливим, однак загальноприйнятим неофіційним стандартом є написання підпрограм *довжиною не більше, ніж один екран коду (20-25 рядків)*.
- *Тіло циклу занадто довге* або рівень вкладеності циклів занадто великий.
- *Клас має багато обов'язків*, слабко пов'язаних між собою, що порушує принцип єдиного обов'язку (Single responsibility principle). У такому разі краще розділити клас на кілька атомарних.
- Інтерфейс класу *не забезпечує достатній рівень абстракції*.
- Назва класу чи методу *недостатньо точно відповідає його змісту*.
- Пов'язані дані, які використовуються разом, *не організовані в клас*.
- Функція має *занадто багато параметрів*.
- У ланцюжку виклику методів передається *багато зайвих даних*.
- Потрібно одночасно змінювати кілька *паралельних ієрархій класів*. Для вирішення цієї проблеми можна, наприклад, скористатися шаблоном «Міст».
- *Клас не виконує ніякої роботи самостійно*, а тільки передоручає її іншим класам.
- Клас має *занадто багато відкритих (public) членів*.
- Нестатичний *клас складається тільки з даних або тільки з методів*.
- *Занадто широке застосування глобальних змінних*.

Існує чимало *прийомів (методів) рефакторингу* – Відокремлення методу (Extract Method), відокремлення базового класу (Extract Superclass), інкапсуляція поля (Encapsulate Field) тощо.

Багато інтегрованих середовищ розробки, зокрема Netbeans, містять *вбудовані механізми рефакторингу коду*. Крім інтегрованої функціональності, існує також багато продуктів сторонніх виробників, які, як правило, реалізовані у вигляді додатків (plugins) до відповідного IDE.

Патерни (або шаблони) проектування описують типові способи вирішення поширених проблем при проектуванні програм. Хоча ви можете цілком успішно працювати, не знаючи жодного патерна, опанувавши їх ви отримаєте ще один потужний інструмент в свій набір професіонала.

Патерни відрізняються за рівнем складності, охоплення і деталізації проектованої системи. Проводячи аналогію з будівництвом, ви можете підвищити безпеку на перехресті, встановивши світлофор, а можете замінити перехрестя цілою автомобільною розв'язкою з підземними переходами.

Низькорівневі та найпростіші патерни – *ідіоми*. Вони не дуже універсальні, позаяк мають сенс лише в рамках однієї мови програмування.

Максимально універсальними є *архітектурні* патерни, які можна реалізувати практично будь-якою мовою. Вони потрібні для проектування всієї програми, а не окремих її елементів. Крім цього, патерни відрізняються і за призначенням. Існує *три основні групи патернів*:

- *Породжуючі* патерни піклуються про гнучке створення об'єктів без внесення в програму зайвих залежностей.
- *Структурні* патерни показують різні способи побудови зв'язків між об'єктами.
- *Поведінкові* патерни піклуються про ефективну комунікацію між об'єктами.

Взагалі існують *32 класичних патерни*, які відрізняються за призначенням.

Поряд з патернами існують і *антипатерни*. *Антипатерн*, або антишаблон – загальний спосіб вирішення проблеми, що часто виникає під час проектування програмного забезпечення, який, як правило, неефективний та зменшує продуктивність комп'ютерної програми. Інакше кажучи, *антипатерн – шкідливий і неефективний патерн*.

Концепція антипатерну є універсальною і придатна не лише для програмної інженерії, але й для практично будь-якої сфери людської діяльності; втім, термін не набув поширення поза межами IT-індустрії.

Поняття антипатерну з'явилось тоді, коли програмісти зрозуміли, що описувати і документувати необхідно не лише гарні ідеї, але й погані. Правильно сформульований антипатерн складається не лише з опису типової помилки, але й пояснює, чому таке рішення виглядає привабливим (наприклад, економить час розробки, чи дійсно працює в деякому обмеженому контексті), до яких негативних наслідків воно призводить, і яким патерном бажано його замінити.

Найважливіша *мета документування антипатернів* – полегшити можливість розпізнавання шкідливого рішення і успішного виправлення помилок ще на ранніх етапах роботи. Антипатерн не просто застерігає «Не роби цього!», а вказує інженеру на те, що він, можливо, не усвідомлює, що рішення такого типу принесе набагато більше шкоди, ніж користі.

Існує 3 ключових правила, коли рішення вважається антипатерном:

- Загальноприйнятий процес, структура чи план дій, який на перший погляд дає ефективне рішення, найчастіше дає *більше негативних наслідків, ніж позитивних*.
- Цей процес (структура, план), незважаючи на свою шкідливість, *достатньо поширений на практиці*.
- *Правильне рішення вже існує*, воно задокументовано та перевірено на ефективність.

У більшості випадків рефакторинг має на меті саме усунення антипатернів.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 6.3. **Обробка помилок та виключень. Відлагодження, тестування та профілювання**
Номер заняття: 37

Мета: познайомити студентів з поняттями виключень; розповісти про інструменти відлагодження, тестування та профілювання, вбудовані в середовище розробки NetBeans.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://youtu.be/D3HMJw8ZmJ8

частина 2: https://youtu.be/9DBQyLXXvYI

частина 3: https://youtu.be/DI4EFkzqCCg

Короткі теоретичні відомості:

Виняткова ситуація або просто *виняток, виключення чи exception* – це аварійний стан, який відбувається саме під час виконання програми. Прикладом є ділення на нуль, помилки читання з файлу та мережі тощо. Іншими словами – це *помилки які можуть виникнути при виконанні програми*. В ряді мов програмування необхідно заздалегідь передбачити можливість тієї чи іншої помилки і передбачити шлях її обробки. В java для цього передбачений спеціальний механізм винятків.

Виняток в java – це *об'єкт, який описує виняткову* (тобто, помилкову) ситуацію, що відбулась в певному місці коду. Коли така ситуація виникає створюється об'єкт, який передається («вкидається») в метод, в якому виникла помилка. Далі в методі даний виняток може оброблятися, або бути переданий ще кудись для обробки.

Розглянемо для прикладу наступну програму DivZero.java

```
public class DivZero {
    public static void main(String[] args) {
        int my = 0;
        int medium = 44 / my;
        System.out.println("medium=" + medium);
    }
}
```

Як бачимо в програмі присутнє ділення на нуль. При компіляції ми не отримаємо помилок. Проте, після запуску програми, отримаємо наступне:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at DivZero.main(DivZero.java:4)
```

Це так звана траса стеку викликів. Перший рядок означає тип-винятку. Як бачимо тут маємо *ArithmeticException* з діленням на нуль. *java.lang* – це пакет класів, який завжди доступний в програмі і який містить найбільш використовувані класи. Тобто його не потрібно імпортувати. Другий рядок вказує де саме відбулась виняткова ситуація: 4-й рядок у файлі *DivZero.java* в методі *main()* класу *DivZero*. При цьому як бачимо програма завершила своє виконання аварійно. Для уникнення цього існує відповідний механізм обробки винятків, який дозволяє перехопити виняток, одержати інформацію про нього, обробити його здійснивши певні дії для нормального закінчення або продовження виконання програми.

Усі типи винятків є підкласами класу *Throwable*, який входить в базовий пакет класів *Java – java.lang*. Тобто він є вершиною ієрархії класів винятків. Його два підкласи *Error* та *Exception* утворюють дві основні гілки винятків.

Клас *Error* з його підкласами – це помилки виконавчого середовища java. І які зазвичай не виникають при нормальній роботі середовища java. Такі винятки зазвичай не можуть бути оброблені в програмі.

Гілка класу *Exception* – це винятки, які програма повинна вловлювати(*catch*). Від даного класу та його підкласів можна утворювати власні підкласи. Важливим його підкласом є клас *RuntimeException*. Винятки даного типу включають такі винятки як ділення на нуль та помилкова індексація масивів.

Актуальну ієрархію класів винятків можна подивитися і уточнити в офіційній документації до JDK.

Для обробки виняткових ситуацій використовується п’ять ключових слів: *try, catch, throw, throws* та *finally*. Інструкції програми, в яких може виникнути помилка, контролюються за допомогою конструкції *try*.

Загальна форма наступна:

```
try{
//блок коду для контролю над помилками
} catch (тип-винятку1 об’єкт-винятку) {
  //дії при виникненні типу винятку1 (обробник винятку)
} catch (тип-винятку2 об’єкт-винятку) {
  //дії при виникненні типу винятку2 (обробник винятку)
}
 //….
[finally{
   //дії при виході з конструкції try.
 }]
```

Після інструкції *try* ми розміщуємо «небезпечний» код, у блоці *catch* відбувається обробка винятку, причому може бути кілька інструкцій *catch*. Завершувати конструкцію може інструкція *finally*, в ній розміщується код, який буде виконаний після обробки винятку в інструкції *catch*.

Таким чином можемо переписати програму з діленням на нуль:

```
public class DivZero {
   public static void main(String args[]){
      int my=0;
      try{
      int medium=44/my;
      System.out.println("medium="+medium);
```

```
        }catch(ArithmeticException e){
            System.out.println("Ділення на нуль!");
        }
        System.out.println("Продовження виконання...");
    }
}
```

Результат виконання:

Ділення на нуль!

Продовження виконання...

Як бачимо, для перехоплення винятку, код, через який виникала помилка, знаходиться у середині конструкції *try*. Також, зверніть увагу, що після виникнення виняткової ситуації наступний рядок *System.out.println("medium="+medium);* не було виведено, оскільки виняток був переданий для обробки в *catch*. Після інструкції програми в якій відбулась виняткова ситуація, всі наступні рядки до інструкції *catch* пропускаються і не будуть виконуватись. І, як бачимо з результату виконання, наша програма продовжила виконання по закінченню блоку *try*, а не завершила своє виконання аварійно.

При використанні всередині *try* певних ресурсів, наприклад, файлів, при виникненні винятку необхідно було передбачити закриття відкритих ресурсів. Для цієї мети раніше приходилося використовувати блок *finally*. У Java 7 з'явилася конструкція *try* з ресурсами (англ. try-with-resources). Тепер просто можна створити ресурс у дужках зразу ж після ключового слова *try* і java сама потурбується про закриття ресурсу. Приклад:

```
static String readFirstLineFromFile(String path) throws IOException {
    try (BufferedReader br =
                 new BufferedReader(new FileReader(path))) {
        return br.readLine();
    }
}
```

В наведених вище прикладах ми здійснювали лише обробку винятків викинутих виконавчим середовищем *java*. Проте існує можливість викидання власних винятків. Для цього існує інструкція *Throw*. Загальна форма її наступна:

throw ThrowableInstance;

Тут *ThrowableInstance* – це тип винятку, який повинен бути або типом *Throwable*, або мати тип його підкласів. Щоб програма викинула ваш виняток необхідно скористатися оператором *new*. Для того, щоб одержати(перехопити) тип винятку можна скористатися інструкцією *catch*, як це ми робили вище.

Після інструкції *throw* відбувається аналогічне породження винятку як у вищенаведених прикладах. Тобто усі інструкції пропускаються до найближчого блоку інструкції *catch*, де необхідно здійснити обробку винятку. Якщо *catch* не буде знайдено, то обробник винятків, що використовується по замовчуванню, призупиняє виконання програми і друкує відбиток стеку *(stack trace)*.

Приклад:

```
public class ThrowNullException {
   public static void main(String args[]){
      try{
         throw new NullPointerException("Пробний виняток");
      }catch(Exception e){
         System.out.println("Наш витяток: "+e);
      }
   }
}
```

Результат:

Наш витяток: java.lang.NullPointerException: Пробний виняток

Програма демонструє як створювати один із стандартних винятків. Більшість вбудованих runtime-винятків Java мають щонайменше два конструктори. Один за замовчуванням без параметрів і один із параметром *String*, який дозволяє задати додатковий опис. Опис можна вивести на консоль за допомогою методів *print(), println()*. Також його можна отримати використавши метод *getMessage()* класу *Throwable*.

Можна створити власний тип винятку як підклас уже існуючого типу.

Якщо метод породжує виняток і не обробляє його, то він повинен вказати про це, щоб обробка винятку була здійснена у місці виклику даного методу. Це здійснюється за допомогою застереження *throws* в оголошенні методу. Після нього вказується підряд через кому усі винятки, які можуть бути викинуті методом, окрім винятків класів *Error* та *RuntimeException* і їхніх підкласів. Нагадаємо, що клас *Error* – це необроблювані винятки, *RuntimeException* – винятки, які виникають в результаті помилки програміста (вихід за межі масиву, нульове посилання, невірне перетворення типів). Інші винятки – це помилки доступу, які доволі часто вимагають відповідної обробки.

Загальна форма оголошення методу наступна:

```
тип ім'я_методу(список_параметрів) throws список_винятків
{
// тіло методу
}
```

Наступна програма демонструє використання *throws* у методі де виникає виняток *IllegalAccessException*.

```
public class ThrowsException {
   public static void exceptionMethod () throws IllegalAccessException{
      System.out.println("Всередині exceptionMethod().");
      throw new IllegalAccessException("Помилка доступу");

   }
    public static void main(String args[]){
      try{
         exceptionMethod();
         System.out.println("Кінець програми"); //даний рядок не буде виведений
      }catch(IllegalAccessException e){
         System.out.println("Наш витяток: "+e);
```

```
        }
    }
}
```

Результат:

Всередині ExceptionMethod().

Наш витяток: java.lang.IllegalAccessException: Помилка доступу

Як бачимо тепер обробка винятку відбувається у методі *main()*. Без інструкції *try-catch* програма призупинятиметься з друком відбитку стеку. Слід зауважити, що у методах інструкція *throw* поводить себе подібно до інструкції *return*. Тобто виконання методу припиняється і відбувається повернення в місце виклику методу.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 6.4. **Колекції та дженерики**
Номер заняття: 39

Мета: познайомити студентів з поняттям колекції; продемонструвати переваги гомогенних колекцій (дженериків); зробити огляд класів та інтерфейсів з Java Collection API.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://youtu.be/8nlQb5hCEYk

Короткі теоретичні відомості:

При об'єктно орієнтованому програмуванні доводиться працювати з великою кількістю об'єктів. Зручно мати засоби групування об'єктів. З цією метою в Java розроблено набір інтерфейсів і класів на їх основі під назвою *колекції*. В основі ієрархії колекцій знаходиться інтерфейс *Collection*.

Згадаймо, що *інтерфейс* не містить реалізації методів, а лише їхні оголошення. Можна реалізувати безліч реалізацій інтерфейсу. Програмісту, який використовуватиме ці реалізації, достатньо знати базовий інтерфейс для роботи з його реалізаціями, тобто знати методи, які передбачає даний інтерфейс.

Collection – базовий інтерфейс, крім нього на його основі в структурі колекцій є ще декілька інтерфейсів, які розширюють базовий інтерфейс *Collection*. Зокрема, *List, Set* та *SortedSet*. Окремо виділяють ще інтерфейс *Map*. Він не походить на пряму від інтерфейсу *Collection*, проте його також відносять до колекцій. На їх основі створено набір класів,

які згодяться програмістам для більшості випадків роботи з набором об’єктів. Тож вам не прийдеться самим їх реалізовувати.

Якщо вам цікаво, для чого стільки різних класів в колекціях? Суть в тому, що різні класи по різному реалізовують роботу з даними. Одні класи швидше здійснюють читання даних, інші вставлення і видалення, одні перевіряють, щоб не було дублювань, інші дозволяють вставляти дані за певним ключем і т.п. Доволі важливо підібрати клас, який найкраще підходить для вашого завдання і забезпечить найбільшу швидкодію. Особливо це актуально, коли кількість об’єктів величезна.

Клас *ArrayList* призначений для читання об'єктів по індексу. Тож не дарма у назві є слово *Array* (масив). Після створення колекції на основі *ArrayList*, прочитати дані можна кількома способами. Наступний приклад демонструє створення *ArrayList*, його наповнення об'єктами типу *String* та їх читання за допомогою методу *get (int index)* та за допомогою ітератора.

```
import java.util.ArrayList;
import java.util.ListIterator;

public class TestArrayList {

   private ArrayList<String> a1;

   public static void main(String[] args) {
      TestArrayList test = new TestArrayList();
      test.create();
      test.getData();
      test.iterateData();
   }

   void create() {
      //створюємо і наповнюємо ArrayList
```

```
        a1 = new ArrayList<String>();
        a1.add("Привіт");
        a1.add("тобі");
        a1.add("божевільний");
        a1.add("світе!");
    }

    //читаємо дані по індексу
    void getData() {
        for (int i = 0; i < a1.size(); i++) {
            System.out.print(a1.get(i) + " ");
        }
    }

    //Читаємо вміст ArrayList з допомогою ітератора
    void iterateData() {
        ListIterator<String> it = a1.listIterator();
        while (it.hasNext()) {
            System.out.print(it.next() + " ");
        }
    }
}
```

Результат:

Привіт тобі божевільний світе! Привіт тобі божевільний світе!

Крім вищенаведених способів можна передати вміст *ArrayList* у звичайний масив за допомогою методу *toArray().* Якщо ви хочете детально розібратися з *ArrayList* і його

методами, то для цього також дивіться інформацію про інтерфейси *Collection, List* та *Iterator*.

Окремо розглянемо перегляд даних з допомогою ітератора.

ListIterator<String> it=a1.listIterator();

Таким чином створюється об’єкт ітератора, посилання на який передається об'єктній змінній *it* типу *ListIterator*. *ListIterator* – це інтерфейс, який розширює інтерфейс *Iterator* декількома новими методами. Базовими ж методами інтерфейсу *Iterator* є ті, що використані у нас в програмі, а саме:

- *boolean hasNext()* – повертає *true*, якщо ітерація має наступний елемент
- *E next()* – повертає наступний елемент ітерації (буква *E* вказує, що це може бути елемент будь-якого типу)
- *void remove()* – знищує останній елемент, що повертався ітератором.

LinkedList – це структура даних, що являє собою пов’язаний список елементів (об’єктів).

Різниця між *ArrayList* та *LinkedList* полягає в тому, що *ArrayList* реалізований у вигляді масиву, а *LinkedList* у вигляді пов’язаних між собою об’єктів. *ArrayList* швидко виконує читання і заміну елементів (посилань на об’єкти), проте, щоб вставити новий елемент в середину *ArrayList* або видалити існуючий в середині *ArrayList* здійснюється послідовний зсув цілого ряду елементів масиву. В *LinkedList* доволі швидко відбувається вставлення нового елементу або видалення існуючого. Це відбувається тому, що в середині реалізації *LinkedList* змінюються лише посилання на попередній і наступний об’єкти (елементи). Проте доступ до об’єктів по індексу в *LinkedList* відбувається повільніше ніж в *ArrayList*. Тож загалом, *LinkedList* корисний, коли необхідно часто вставляти та видаляти елементи зі списку, а в інших випадках краще використовувати *ArrayList*.

Існує два конструктури *LinkedList*:

LinkedList()

LinkedList (Collection c)

Перший конструктор створює пустий список, а другий – створює пов’язаний список із іншої колекції. Клас *LinkedList* розширює клас *AbstractSequentalList* та реалізує інтерфейси *List, Dequeue* та *Queue*. Реалізація останніх двох інтерфейсів (черг) означає, що ми можемо працювати із пов’язаним списком як із стеком з використанням методів

pop(), push(), poll(), pollFirst(), pollLast() і т.п. Детальніше дивіться документацію по заданим інтерфейсам.

HashSet – це клас призначений для зберігання даних у вигляді множини невпорядкованих елементів. Якщо потрібна впорядкована множина, то використовуйте *TreeSet*. *HashSet* також не гарантує стабільного порядку збереження об’єктів. Тобто при додаванні об’єктів порядок зберігання елементів змінюється. Вони можуть бути збережені як в кінці множити так і в середині. Якщо потрібен один порядок зберігання об’єктів використовуйте *LinkedHashSet*.

Сам термін «множина» означає, що елементи не будуть повторюватися. Для зберігання і пошуку елементів використовується хеш-код об’єкта. *HashSet* також може містити значення *null*. Власне всередині самої реалізації *HashSet* використовується клас *HashMap*, який дозволяє зберігати елементи у вигляді двох складових ключа та хеш-коду. У класі *HashSet* хеш-код недоступний і використовується неявно для користувача.

Клас *HashSet* розширює клас *AbstractSet* та реалізує інтерфейс *Set*. Також реалізовує інтерфейси *Serializable* та *Clonable*.

Клас *LinkedHashSet* розширює клас *HashSet* не додаючи ніяких нових методів. Працює він дещо довше за *HashSet* проте зберігає порядок в якому елементи додаються до нього. Відповідно це дозволяє організувати послідовну ітерацію вставлення та витягнення елементів. Всі конструктори та методи роботи з *LinkedHashSet* аналогічні методам класу *HashSet*.

Клас *TreeSet* дозволяє створювати відсортовану множину. Тобто елементи не повторюються та зберігаються у відсортованому порядку. Для зберігання елементів застосовується бінарна деревоподібна структура. Об'єкти зберігаються в відсортованому порядку по зростанню. Час доступу та одержання елементів доволі малий, тому клас *TreeSet* підходить для зберігання великих об’ємів відсортованих даних, які повинні бути швидко знайдені.

Клас *TreeSet* розширює клас *AbstractSet* та реалізує інтерфейс *NavigableSet*. *NavigableSet* реалізується на базі *TreeMap*.

Ранні версії Java не включали структуру *Collections*. Там було визначено декілька класів та інтерфейсів, що надавали методи для зберігання об'єктів. Структура *Collections* була додана в Java 2 (j2se 1.2). Тоді початкові класи були перероблені для підтримки інтерфейсів колекцій. Ці ранні класи також знані як *успадковані класи* (Legacy classes).

В Java 5 успадковані класи та інтерфейси були перероблені для підтримки узагальнень. Їх підтримують, тому що й досі існує код, який їх використовує. Успадковані класи – синхронізовані. Класи входять в пакет *java.util*.

Успадкованими є наступні класи:

- *Dictionary*
- *HashTable*
- *Properties*
- *Stack*
- *Vector*

Успадкованим є інтерфейс *Enumeration*, на заміну якому прийшов інтерфейс *Iterator*. *Enumeration* інтерфейс й досі використовується в кількох методах класів *Vector* та *Properties*.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Змістовий модуль 7. Консоль. Файлова система. Графічний Інтерфейс користувача

Тема 7.1. **Основи вводу-виводу. Робота с консоллю та файловою системою**

Номер заняття: 41-42

Мета: познайомити студентів з поняттям консолі та способами консольного введення-виведення; зробити огляд бібліотек для побудови текстового інтерфейсу користувача; познайомити з класами для роботи з файлами.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/anrJ

Короткі теоретичні відомості:

Потік *вводу даних з консолі* можна організувати одним з двох способів:

– з допомогою *потоку вводу байтів*;
– з допомогою *потоку вводу символів*. У цьому випадку дані, що вводяться з консолі, зчитуються з стандартного потоку вводу, якому відповідає змінна *System.in* з пакету *java.lang*. Іншими словами, змінна *System.in* посилається на стандартний потік вводу, яким, за замовчуванням, вважається клавіатура.

У мові Java для реалізації операцій вводу з консолі використовується клас *InputStreamReader*. Цей клас є підкласом абстрактного базового класу *Reader* і призначений для конвертування байт в символи.

Щоб отримати об'єкт типу *InputStreamReader* потрібно створити екземляр цього типу за наступною загальною формою:

InputStreamReader objStream = new InputStreamReader(input_stream);

тут

- *objStream* – екземпляр, що зв’язаний з потоком вводу символів;
- *inputStream* – потік вводу символів. У випадку консолі, цим потоком виступає *System.in.*

Отже, для потоку *System.in*, щоб створити екземпляр (об’єкт) типу *InputStreamReader*, потрібно викликати наступний конструктор

InputStreamReader objStream = new InputStreamReader(System.in);

В пакеті *java.lang* реалізовано дві змінні, що зв’язані з потоком запису даних на консоль:

- *System.out*. Ця змінна відповідає потоку виведення на консоль;
- *System.err*. Ця змінна зв’язана з потоком виведення помилок.

Для роботи з виведенням на консоль розроблено спеціальний клас *PrintStream*, який є похідним від класу *OutputStream*. У цьому класі основними є методи *print()* та *println()*.

Щоб створити потік виводу, зв’язаний з консоллю потрібно оголосити екземпляр класу *PrintStream*. При створенні екземпляру потрібно зв’язати його з відповідним потоком як показано в наступному прикладі:

```
public static void main(String[] args) throws IOException
{
  // Створення потоку виводу. Клас PrintStream.
  // 1. Зв'язати посилання outStream з потоком виводу System.out
  PrintStream outStream = new PrintStream(System.out);

  // 2. Зв'язати посилання errStream з потоком виводу System.err
  PrintStream errStream = new PrintStream(System.err);

  // 3. Використати обидва посилання
  outStream.println("Hello world!"); // Метод println()
  errStream.print("Some error..."); // Метод print()
```

```
}
```

Результат виконання програми:

```
Hello world!

Some error...
```

Як відомо, з консоллю зв’язана стандартна змінна *System.in*. Щоб прочитати символи з консолі без буферизації можна використати можливості класу *InputStreamReader*. Клас *InputStreamReader* є підкласом абстрактного класу *Reader*. Клас *InputStreamReader* призначений для конвертування байт у символи.

Основним методом класу *InputStreamReader* є метод *read()*, який має дві перевантажені реалізації. Перша реалізація має наступну загальну форму

```
int read();
```

У цьому випадку метод повертає код введеного символу з консолі.

Загальна форма другої реалізації методу наступна

```
int read(char[] ac, int index, int count);
```

Тут метод заповнює масив *ac* типу *char[]* введеними значеннями символів з консолі. Параметр count задає кількість символів, які потрібно вставити у масив *ac*. Для масиву ac попередньо має бути виділена пам’ять. Другий варіант методу повертає кількість символів, що можуть бути прочитані.

Щоб реалізувати буферизацію при читанні символів з консолі, потрібно екземпляр класу *InputStreamReader* помістити в оболонку об’єкту класу *BufferedReader*. У цьому випадку, загальна форма конструктора класу *BufferedReader* наступна

```
BufferedReader(Reader потік_читання_введених_даних)
```

Тут *потік_читання_введених_даних* – деякий потік даних (файл, консоль тощо). У випадку консолі тут задається екземпляр класу *InputStreamReader* (дивіться попередній пункт).

Такий підхід реалізує *паттерн Декоратор (Decorator)*. У паттерні Декоратор екземпляр одного класу служить оболонкою для екземпляру іншого класу. Таким чином відбувається нашаровування об’єктів.

Нехай задано символьний (текстовий) файл, який потрібно вивести на консоль. Читання рядків з файлу можна реалізувати з допомогою методу *read()* класу *FileReader*. Рядок з файлу читається у наперед визначену ділянку пам'яті – буфер.

Щоб визначити кінець файлу потрібно використати метод *ready()* класу *FileReader*. Якщо кінець файлу, то метод повертає *false*.

```
import java.io.*;
...
public static void main(String[] args) throws IOException
{
  // Зчитати рядки з файлу та вивести їх на консоль
  FileReader fr = new FileReader("strings.txt");
  char buffer[] = new char[1000]; // буфер для читання рядків

  // Цикл читання рядків з файлу
  do {
    fr.read(buffer); // Зчитати рядок в буфер buffer
    System.out.println(buffer); // Вивести рядок на консоль
  } while (fr.ready()); // перевірка, чи не кінець файлу
 fr.close();
}
```

Нехай задано деякий рядок типу *String*. Потрібно записати цей рядок у файл шляхом використання перенаправлення потоку виводу у файл.

Щоб реалізувати запис рядка у файл потрібно виконати наступні дії:

- створити екземпляр класу *FileOutputStream* з іменем файлу, в який буде здійснюватись запис рядка;
- створити екземпляр класу *PrintStream*, в конструктор якого помістити створений екземпляр класу *FileOutputStream*;
- використати один з методів *print()* або *println()* для запису рядка у файл.

Фрагмент програмного коду, що здійснює перенаправлення потоку з рядка в файл, наведено нижче

```
import java.io.*;
...
public static void main(String[] args) throws IOException
{
  // Направити вивід з рядка String у файл з іменем "output.txt"
  // 1. Вхідні дані
  String s = "www.bestprog.net"; // рядок, що записується у файл
  String filename = "output.txt"; // ім'я файлу, в який записується рядок
  // 2. Створити екземпляр класу FileOutputStream
  FileOutputStream fs = new FileOutputStream(filename);
  // 3. Створити екземпляр класу PrintStream, в який помістити екземпляр FileOutputStream
  PrintStream ps = new PrintStream(fs);
  // 4. Записати рядок "www.bestprog.net" у файл "output.txt"
  ps.println(s);
  // 5. Закрити файл
  ps.close();
}
```

За даним прикладом можна реалізувати функцію, яка записує масив рядків у файл.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 7.2. **Робота з базами даних. JDBC. Entity Classes.**
Номер заняття: 44

Мета: познайомити студентів з JDBC та класами сутностей; продемонструвати ефективність інструментів для роботи з базами даних, вбудованих в NetBeans.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/RKNy

Короткі теоретичні відомості:

Java DataBase Connectivity (з'єднання з базами даних на Java), скорочено JDBC) – прикладний програмний інтерфейс Java, який визначає методи, з допомогою яких програмне забезпечення на Java здійснює доступ до бази даних. JDBC – це платформо-незалежний промисловий стандарт взаємодії Java-застосунків з різноманітними СУБД, реалізований у вигляді пакета java.sql, що входить до складу Java SE.

В основі JDBC лежить концепція так званих *драйверів*, що дозволяють отримувати з'єднання з базою даних по спеціально описаному URL. Драйвери можуть завантажуватись динамічно (під час роботи програми). Завантажившись, драйвер сам реєструє себе й викликається автоматично, коли програма вимагає URL, що містить протокол, за який драйвер «відповідає».

JDBC API містить два основні типи інтерфейсів: перший – для розробників застосунків і другий (нижчого рівня) – для розробників драйверів.

З'єднання з базою даних описується класом, що реалізує інтерфейс *java.sql.Connection.* Маючи з'єднання з базою даних, можна створювати об'єкти типу *Statement*, використовувані для здійснення запитів до бази даних мовою SQL.

Існують такі види типів *Statement*, що відрізняються своїм призначенням:

- *java.sql.Statement* – загального призначення;
- *java.sql.PreparedStatement* – служить для здійснення запитів, котрі містять підставні параметри (позначаються символом '?' у тілі запиту);
- *java.sql.CallableStatement* – призначений для виклику збережених процедур.

Клас *java.sql.ResultSet* дозволяє легко обробляти результати запитів.

Перевагами JDBC вважаються:

- *простота розробки*: розробник може не знати специфіки бази даних, з якою працює;
- *код не змінюється*, якщо компанія переходить на іншу базу даних;
- *не треба встановлювати* громіздку клієнтську програму;
- *до будь-якої бази можна під'єднатись* через легко зрозумілий URL.

Цей приклад використовує вільний драйвер JDBC для MySQL, який легко встановлюється в більшості дистрибутивів Linux через стандартні репозиторії:

```
package javaapplication1;

import java.sql.*;

public class Main {

   public static void main(String[] args) throws SQLException

   {

      /**

      * цей рядок вказує драйвер DB.

      * розкоментуйте, якщо прописуєте драйвер вручну

      */

      //Class.forName("com.mysql.jdbc.Driver").newInstance();

      Connection conn = DriverManager.getConnection(

         "jdbc:mysql://localhost:3306/db_name",
```

```
            "user", "password");
        if (conn==null)
        {
            System.out.println("Немає з'єднання з БД!");
            System.exit(0);
        }
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM users");
        while(rs.next())
        {
            System.out.println(rs.getRow() + ". " + rs.getString("firstname") + "\t" +
rs.getString("lastname"));
        }
        // if(rs!=null)rs.close();
        /**
         * stmt.close();
         * При закритті Statement автоматично закриваються
         * всі пов'язані з ним відкриті об'єкти ResultSet
         */
        // if(stmt!=null)stmt.close();
        stmt.close();
    }
}
```

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 7.3. **Створення графічного інтерфейсу користувача**
Номер заняття: 45

Мета: познайомити студентів зі Swing – інфраструктурою для побудови інтерфейсів користувача; продемонструвати можливості дизайнера інтерфейсів Matisse, вбудованого в NetBeans.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/vWYd

Короткі теоретичні відомості:

Для реалізації графічного інтерфейсу (GUI) в Java існують два основні пакети класів:

– *Abstract Window Toolkit (AWT)*
– *Swing*

Перевагами першого є простота використання, інтерфейс подібний до інтерфейсу операційної системи та дещо краща швидкодія, оскільки базується на засобах ОС, щоправда має обмежений набір графічних елементів. Другий пакет Swing реалізує власний Java-інтерфейс. Цей пакет створювався на основі AWT, і має набагато більше можливостей та більшу кількість графічних елементів. Swing-компоненти ще називають полегшеними (англ. lightweight), оскільки вони написані повністю на Java і, через це, *платформонезалежні*.

Принципи роботи із обома пакетами схожі, тож опанувавши роботу з одним, робота з іншим не матиме труднощів. *Swing* містить великий набір компонентів для розробки графічного інтерфейсу користувача (GUI). Оскільки Swing – це повніший

інструментарій ніж AWT, тому основну увагу ми приділимо саме йому. Робота з AWT компонентами майже аналогічна, щоправда їх дещо менше. Для розрізнення компонентів інструментаріїв, компоненти Swing містять літеру «J» спереду назви (*JButton, JPanel* тощо). Назви AWT-компонентів не мають даної літери спереду.

Swing та AWT є частиною *JFC (Java Foundation Classes)*, що в свою чергу становить левову частку стандартної платформи Java. Класи Swing розміщуються в пакеті *javax.swing* та його підпакетах. Суфікс *–x* прийнято ставити для нестандартних розширень, проте конкретно ці пакети є стандартними починаючи з JDK 1.2.

Вікно верхнього рівня (таке що не містить в середині іншого вікна) в мові Java називається фреймом (frame - каркас). В бібліотеці AWT для цього вікна призначений клас *Frame*, а в бібліотеці Swing – *JFrame*.

Наступний код демонструє, як можна створити фрейм розміром 300x200 пікселів:

```
import javax.swing.JFrame;
public class SimpleFrame
{
  public static void main(String[] args)
  {
    JFrame frame = new JFrame();     // створити фрейм
    frame.setSize(300, 200);         // задаємо ширину і висоту фрейму
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // дії при закритті
   //фрейму
    frame.setVisible(true); // показати фрейм на екрані (зробити видимим)
  }
}
```

Проте розміщувати увесь код в *main()* поганий стиль, оскільки, наприклад, коли Вам доведеться додавати нові фрейми та інші елементи, код може стати доволі заплутаним. Крім того метод *main* статичний і потрібно враховувати деякі особливості робити у статичному контексті. Тому в *main()* часто намагаються залишити лише саме необхідне. І взагалі, якщо якийсь графічний елемент потребує значного коду, то бажано роботу з ним розмістити окремо. В нашому випадку всю роботу з фреймом можна доручити класу, який розширюватиме клас *JFrame*.

Наступний переписаний код робить теж саме, що і попередній. На перший погляд може здатися, що такий спосіб є доволі незручним, насправді ж у великих проектах це виправдано.

```
import javax.swing.*;
public class SimpleFrame
{
  public static void main(String[] args)
  {
    MyFrame frame = new MyFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
class MyFrame extends JFrame
{
  public MyFrame()
  {
    setSize(FRAME_WIDTH, FRAME_HEIGHT);
  }
  public static final int FRAME_WIDTH = 350;
  public static final int FRAME_HEIGHT = 200;
}
```

Програма складається з двох класів, хоча метод *MyFrame()* можна б було розмістити і в класі *SimpleFrame*. Проте, на думку ряду програмістів, краще відділяти клас який запускає програму на виконання від класу, в якому описується інтерфейс користувача. Одною з причин такого є більш краща читабельність коду програми.

Фрейм – це контейнер в який поміщають всі інші компоненти (меню, кнопки, прапорці та інші елементи графічного елементу). Сам клас *JFrame*, який реалізує фрейм складається з чотирьох областей (pane), що накладаються одна на одну: коренева область

(root pane), область шару (layered pane), прозора область (glass pane) та область вмісту (content pane). Перші три застосовують для створення та обслуговування меню. Для роботи з графічними елементами застосовується область вмісту, в яку і додають компоненти. Додати компонент можна наступним чином:

```
Container contentPane = frame.getContentPane();
Component c=....;
contentPane.add (c);
// з версії JDK 1.5 реалізований метод JFrame.add()
// що переадресовує виклик методу області вмісту contentPane.add()
// і тепер можна просто писати: add (c)
```

Щоправда напряму у фреймі додавати компоненти не прийнято. Для цього використовується спеціальний компонент-контейнер – панель (panel), що додається до фрейму. Після цього в панель можна додавати різні графічні компоненти.

Для початку спробуємо намалювати прямокутник. Щоб додати відповідну панель, в якій буде здійснюватись малювання, необхідно:

- Визначити клас, що розширює клас *JPanel*;
- Замістити (перевизначити) в цьому класі метод *paintComponent()*.

```
class MyPanel extends JPanel{
  public void paintComponent(Graphics g){
     .....// код, що здійснює малювання
  }
}
```

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 7.4. **Обробка подій від інтерфейсних елементів**
Номер заняття: 46

Мета: познайомити студентів поняттями події, обробників та адаптерів подій; продемонструвати можливості дизайнера інтерфейсів Matisse, вбудованого в NetBeans.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/spST

Короткі теоретичні відомості:

Натиснення кнопки, закриття вікна, клацання мишкою – все це є прикладами *подій*, які отримує операційна система і передає відповідній програмі на обробку. Програміст повинен передбачити як потрібно обробляти дані події. Розглянемо як обробка подій реалізовується в Java.

В Java запропонована, так звана, модель делегування подій (event delegation model). *Джерело події (event source)* породжує подію, після чого вона передається в *обробник подій* (event listener – дослівно слухач події). При цьому будь-який об'єкт може бути призначеним як обробник деякої події. Така модель доволі гнучка, оскільки кожен програміст може вибрати зручний для нього спосіб обробки події (де саме її обробляти), проте інколи текст програми може бути дещо заплутаним для тих хто не звик до такої моделі.

Інформація про подію інкапсулюється в *об'єкті події* (event object). Всі події описуються підкласами *java.util.EventObject*. Як приклади, можна навести підкласи *ActionEvent* та *WindowEvent*. Перші об'єкти породжують кнопки, а другі вікна.

Джерела подій містять методи, які дозволяють зв’язати його з обробниками подій. Коли виникає подія, джерело повідомляє про неї усіх зареєстрованих обробників. Обробники подій на основі інформації у об’єкті події визначає як реагувати на ту чи іншу подію.

При розробці графічного інтерфейсу розробнику необхідного здійснити наступне:

- Створити клас, який *оброблятиме* подію чи ряд подій і, який реалізовуватиме відповідний інтерфейс
- Створити *джерела подій* (вікна, кнопки, смуги прокрутки тощо)
- *Зв’язати джерела подій з обробниками подій*

Сказане демонструє наступний фрагмент програми:

```
ActionListener listener = . . .;    // створити обробник подій

JButton button = new JButton("Ok"); // створюємо кнопку

button.addActionListener(listener);  // зв'язуємо кнопку з обробником подій
```

Клас, який реалізовуватиме інтерфейс *ActionListener* повинен мати метод *actionPerformed()* який в якості параметру отримуватиме об’єкт *ActionEvent.*

```
class MyListener implements ActionListener

{

. . .

public void actionPerformed(ActionEvent event)

{

// тут відбувається реакція на натиснення кнопки

. . .

}

}
```

Реалізувати такий механізм можна кількома способами. Часто також замість створення окремого класу обробника використовують *внутрішні класи* і навіть *внутрішні анонімні класи*. Також в якості обробника події може виступати і *сам клас*, в якому описується графічний інтерфейс користувача.

Класи-адаптери являють собою *пусту реалізацію інтерфейсів-слухачів*, що мають більше одного методу. Їх імена складаються із імені події і слова *Adapter*. Наприкладр, для дії з мишею є два класи-адаптери. Виглядають вони дуже просто:

```
public abstract class MouseAdapter implements MouseListener{
public void mouseClicked(MouseEvent e){}
public void mousePressed(MouseEvent e){}
public void mouseReleased(MouseEvent e){}
public void mouseEntered(MouseEvent e){}
public void mouseExited(MouseEvent e){}
}

public abstract class MouseMotionAdapter implements MouseMotionListener{
public void mouseDragged(MouseEvent e){}
public void mouseMoved(MouseEvent e){}
}
```

Замість того щоб реалізувати інтерфейс, можна розширятb ці класи. Крім вже згадуваних трьох класів, існують ще класи *ComponentAdapter*, *ContainerAdapter*, *FocusAdapter* і *KeyAdapter*.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 7.5. **Тонке налагодження інтерфейсу користувача**
Номер заняття: 48-49

Мета: познайомити студентів з JavFX та супутніми технологіями; продемонструвати можливості дизайнера інтерфейсів Monet, вбудованого в NetBeans.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

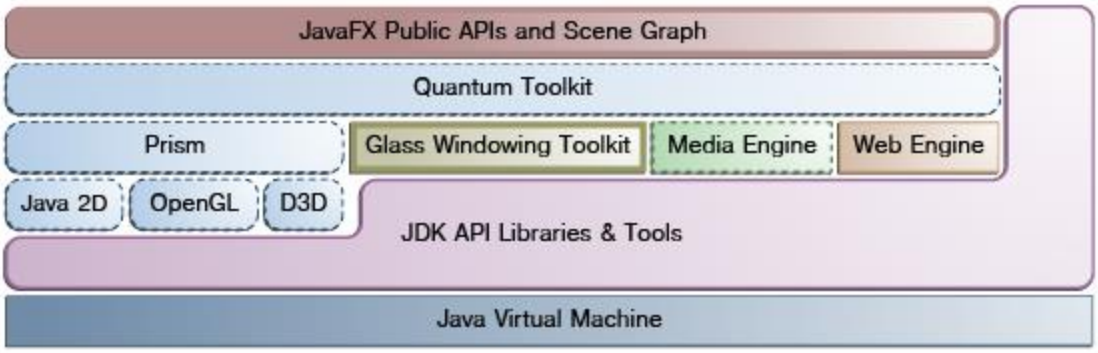
Відеозапис лекції:

частина 1: https://youtu.be/XToB42ngkYM

Короткі теоретичні відомості:

JavaFX – платформа та набір інструментів для створення *насичених інтернет-застосунків (англ. Rich Internet Applications, RIA)* з можливістю підвантаження медіа та змісту. Вперше продемонстровано Sun Microsystems на Міжнародній конференції Java-розробників JavaOne у травні 2007. JavaFX містить у собі набір утиліт, за допомогою яких веб-розробники та дизайнери можуть швидко створювати та надавати розвинуті інтернет-застосунки для десктопів, мобільних пристроїв, телебачення та інших платформ.

JavaFX складається з JavaFX Script і JavaFX Mobile. Починаючи з випуску JavaFX 2.0 забезпечено можливість створення JavaFX-застосунків, написаних цілком мовою Java. Для розробки застосунків доступний багатий графічний і мультимедійний API, що спрощує створення візуальних програм.



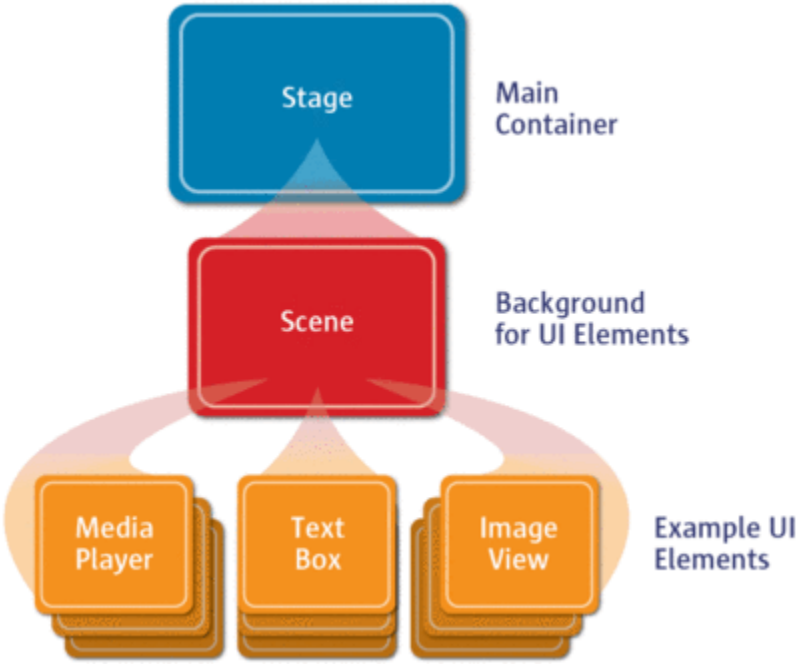
Цікавою особливістю є те, що вигляд та поведінку програм, написаних на JavaFX, можно налаштовувати за допомогою каскадних таблиць стилів (CSS – Cascading Style Sheets). Цей підхід, який історично був першим застосований для веб-сайтів, дозволяє відокремити зовнішній вигляд програм (інтерфейс користувача) від реалізації, що дозволяє програмістам концентруватись на кодуванні. Турбота про графічний інтерфейс тепер лежить на плечах графічних дизайнерів, які налаштовують зовнішній вигляд за допомогою скриптової мови FXML та технології CSS, а програмісти зосереджені на розробці бізнес-логіки додатка. Нижче наведено вигляд додатку із різними налаштуваннями CSS.

Існує два шляхи створення інтерфейсу користувача з JavaFX: використовувати *файл розмітки FXML* чи *програмувати все на Java*. Для більшості випадків ми будемо використовувати XML (.fxml). Цей спосіб більше підходить для збереження роздільності контролера та представлення один від іншого. В подальшому, ми зможемо використовувати графічний Scene Builder для візуального редагування нашого XML. А це означає, що нам не потрібно працювати з XML напряму.

Також потрібно створити *основний Java клас*, який запускає наш додаток з RootLayout.fxml.

Створений клас MainApp.java наслідує клас *Application* та вміщує два методи. Це базова структура, що потрібна для для запуску JavaFX додатку. Для нас важливий метод *start(Stage primaryStage).* Він автоматично викликається коли додаток запускається з методу *main*.

Як бачите, метод *start(...)* приймає екземпляр класу *Stage* в ролі параметра. На рисунку знизу представлена структура будь-якого JavaFX додатку:



Це наче театральна п'єса: *Stage* є основним контейнером, який, як правило, представляє вікно з рамками та стандартними кнопками закрити, мінімізувати та максимізувати. Всередину *Stage* додається *Scene*, яка, звичайно, може бути замінена іншою *Scene*. Всередину *Scene* вже додаються стандартні компоненти типу *AnchorPane*, *TextBox* та інші.

Якщо додаток не може знайти вказаного fxml файлу, ви отримаєте наступне повідомлення про помилку:

java.lang.IllegalStateException: Location is not set.

Для вирішення проблеми перевірте правильність шляхів до файлу та правильність написання його назви.

Детально створення і функціонування JavaFX-додатків описані в офіційній документації Java – https://college.page.link/hCHs.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Змістовий модуль 8. Багатопоточність. Мережеві можливості. Платформа Netbeans

Тема 8.1. **Багатопоточність в JAVA**

Номер заняття: 50

Мета: познайомити студентів з поняттями фонової обробки та потоку; продемонструвати можливості побудови багатопоточних додатків на практичному прикладі.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/jmFB

Короткі теоретичні відомості:

Паралелізм в програмуванні (Concurrency) – це створення програм у вигляді кількох виконуваних, паралельно-взаємодійних між собою гілок (частин) програми. Розрізняють *процеси* (*processes*) та *потоки* або нитки (*threads*). Частина перекладачів перекладають thread, як "*потоки виконання*", проте в ряді випадків цей переклад призводить до плутанини через існування потоків вводу/виводу (I/O Streams). Нитки є паралельно-виконуваними частинами процесу. Процеси – незалежні від інших процесів, ніж нитки від інших ниток. Зокрема, кожен процес має власний адресний простір у пам'яті комп'ютера, нитки ж одного процесу використовують спільний адресний простір пам'яті.

У програмуванні на Java, програмістам зазвичай, доводиться мати справу з окремими нитками. *Багатонитковість* (multithreading) є важливою можливістю Java. Кожна

програма Java має хоча б одну нитку виконання. Виконання програми розпочинається зі створенням головної нитки (main thread).

При роботі з нитками, програміст може або вручну керувати ними, або ж використовувати спеціальний менеджер – *виконувач (executor)*. Спочатку, необхідно розглянути перший спосіб керування нитками виконання.

Кожна нитка – це екземпляр класу *Thread*. Щоб мати змогу створити нову нитку та оперувати нею, зрозуміло, що необхідний код для виконання у цій нитці. Це можна зробити двома шляхами:

– через *реалізацію інтерфейсу Runnable*, що робиться через визначення методу run:

```
public class HelloRunnable implements Runnable {
    public void run() {
        System.out.println("Привіт - це нитка!");
    }
    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }
}
```

– другий спосіб полягає у *створенні підкласу для Thread*. Клас *Thread* уже містить реалізацію інтерфейсу *Runnable*. Проте його реалізація методу *run* – порожня. Тож програміст у своєму коді просто заміщає метод *run*:

```
public class HelloThread extends Thread {
    public void run() {
        System.out.println("Привіт - це нитка!");
    }
    public static void main(String args[]) {
        (new HelloThread()).start();
    }
}
```

Як бачимо в обох випадках для створення нової нитки викликається метод *Thread.start*.

Рекомендованим є перший спосіб, оскільки він дозволяє нам розширити ще якийсь клас у нашому класі (В Java відсутнє множинне успадкування, тож неможливо розширити декілька класів).

Виконання нитки можна призупинити за допомогою методу *sleep* класу *Thread*. Можливо, також, перервати виконання нитки з іншої нитки за допомогою методу *interrupt*. Скориставшись методом *join*, ми можемо задати час очікування на завершення виконання іншої нитки (нескінченно або вказаний час). Наступний приклад демонструє роботу двох ниток. Основна (*main*) нитка запускає виконання іншої нитки, яка просто виводить важливе повідомлення по одному рядку через кожні 4 секунди. Основна нитка виконання кожної секунди виводить повідомлення, що вона все ще очікує на завершення породженої нитки. Якщо виконання породженої нитки триває більше періоду часу, що заданий у змінній *patience* (терпіння), то нитка перериває виконання породженої нитки (закінчується терпіння). При перериванні породженої нитки, у ній виникає виняток типу *InterruptedException*, який обробляється нею за допомогою конструкції *try-catch*.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 8.2. **Робота з мережею**
Номер заняття: 51

Мета: познайомити студентів з поняттями сокетів; продемонструвати можливості побудови мережевих додатків на прикладі розробки месенджера.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/7XaF 

Короткі теоретичні відомості:

На сьогоднішній день клієнти служб миттєвого обміну повідомленнями (instant messenger) стали незамінним інструментом для всіх користувачів Інтернету. Існує безліч різних протоколів і клієнтів про які кожен чув і які ми всі щодня використовуємо. Але не всі знають, що покладено в основу їх роботи. Припустимо, ви встановили один з клієнтів служб миттєвого обміну повідомленнями на ваш комп'ютер. Після його запуску і введення імені і пароля вашого користувача, програма намагається підключитися до сервера. Що саме означає слово "підключитися"?

Кожен комп'ютер в мережі має *IP-адресу*. Ця електронна адреса схожа на вашу домашню адресу – вона однозначно ідентифікує ваш комп'ютер і дозволяє іншим спілкуватися з вашим комп'ютером. Хоча існує й інший спосіб ідентифікації комп'ютерів в мережі – *доменне ім'я*, яке зручніше й наочніше ідентифікує комп'ютер, ніж простий набір чисел (наприклад, www.quizful.net). В Інтернеті існують спеціальні комп'ютери, які здійснюють перетворення доменного імені в IP-адресу і навпаки.

На одному комп'ютері може паралельно виконуватись кілька програм. Припустимо, що ви запустили 10 програм на своєму комп'ютері, і всі вони очікують, щоб інші комп'ютери зв'язалися з ними. Можете уявити собі це так: вас 10 чоловік у великому офісі з 1 телефоном і всі чекають дзвінків від їх власних клієнтів. Як вирішити цю проблему? Можна звичайно призначити відповідального працівника, і він буде приносити телефон людині, якій телефонують, але тоді інші клієнти не зможуть додзвонитися до інших людей. Крім того, це дуже важко і безглуздо мати відповідального працівника для маршрутизації дзвінків. Ви мабуть, уже здогадалися, до чого я веду – якщо всі ці програми, що виконуються на одному комп'ютері, просять своїх клієнтів зв'язатися з ними за певною IP-адресою, то клієнти не будуть задоволені. Ідея полягає в наступному – мати окреме IP-адресу для кожної програми, правильно? Неправильно! Ну, тоді може окремих телефонних номерів буде достатньо? Так! На мережевому жаргоні «окремі телефонні номери» мають назву *порти*. *Порт* – це просто число і кожна з програм, яка виконується на певному комп'ютері, може вибрати унікальний порт, щоб відкрити себе для зовнішнього світу. Запам'ятайте – ці порти ви не зможете знайти серед апаратних засобів комп'ютера (навіть не намагайтеся їх шукати). Ці числа – логічні.

Тобто існує IP-адреса, за допомогою якого інші комп'ютери можуть розпізнавати певний комп'ютер в мережі, і порт – число, яке визначає якусь програму, що працює на комп'ютері. Також стає зрозумілим й те, що дві програми на різних комп'ютерах можуть використовувати один і той же порт (два будинки на різних вулицях теж можуть мати один і той самий номер, чи ні?).

Якщо скласти докупи все сказане, то отримаємо:

IP-адреса + порт-число = _____

Іншими словами:

_____ = унікально визначає програму в мережі.

_____ – це і є ... СОКЕТ!

Підіб'ємо підсумки, *сокет – це комбінація IP-адреси і порту*. Сокет-адреса надає можливість іншим комп'ютерам в мережі знаходити певну програму, яка виконується на певному комп'ютері. Ви можете відображати сокет-адресу наступним чином: 64.104.137.58:80, де 64.104.137.58 – IP-адреса і 80 – порт.

Досить про теорію, давайте перейдемо до дій. Розробимо дуже простий і наочний код на Java, який продемонструє можливості використання сокетів:

Серверний код:

```
import java.net.*;
import java.io.*;
public class Server {
  public static void main(String[] ar)    {
    int port = 6666; // випадковий порт (від 1025 до 65535)
      try {
        ServerSocket ss = new ServerSocket(port); // створюємо серверний сокет
        System.out.println("Waiting for a client...");
        Socket socket = ss.accept(); // чекаємо на під'єднання
        System.out.println("Got a client :) ... Finally, someone saw me!");
        System.out.println();
          // отримуємо вхідний та вихідний потоки для спілкування
        InputStream sin = socket.getInputStream();
        OutputStream sout = socket.getOutputStream();
          // полегшуємо обробку текстових повідомлень
        DataInputStream in = new DataInputStream(sin);
        DataOutputStream out = new DataOutputStream(sout);
        String line = null;
        while(true) {
          line = in.readUTF(); // чекаємо на повідомлення
          System.out.println("The dumb client just sent me this line : " + line);
          System.out.println("I'm sending it back...");
          out.writeUTF(line); // відповідаємо тим же повідомленням
          out.flush(); // завершуємо передачу
          System.out.println("Waiting for the next line...");
          System.out.println();
```

```
            }
          } catch(Exception x) { x.printStackTrace(); }
        }
      }
```

Клієнтський код:

```
    import java.net.*;
    import java.io.*;
    public class Client {
       public static void main(String[] ar) {
          int serverPort = 6666; // порт сервера
          String address = "127.0.0.1"; // IP-адреса сервера
          try {
              InetAddress ipAddress = InetAddress.getByName(address);
            // створюємо об'єкт IP-адреси
             System.out.println("Any of you heard of a socket with IP address " + address +
             " and port " + serverPort + "?");
              Socket socket = new Socket(ipAddress, serverPort); // створюємо сокет
              System.out.println("Yes! I just got hold of the program.");
              // отримуємо потоки
              InputStream sin = socket.getInputStream();
              OutputStream sout = socket.getOutputStream();
              // полегшуємо обробку текстових повідомлень
              DataInputStream in = new DataInputStream(sin);
              DataOutputStream out = new DataOutputStream(sout);
              // створюємо потік для читання з клавіатури
```

```
            BufferedReader keyboard = new BufferedReader(new
            InputStreamReader(System.in));
             String line = null;
             System.out.println("Type in something and press enter");
             System.out.println();
             while (true) {
                line = keyboard.readLine(); // чекаємо вводу користувача
                System.out.println("Sending this line to the server...");
                out.writeUTF(line); // надсилаємо на сервер
                out.flush(); // завершуємо передачу
                line = in.readUTF(); // чекаємо сервера
                System.out.println("The server was very polite. It sent me this : " + line);
                System.out.println("Looks like the server is pleased with us.");
                System.out.println();
             }
          } catch (Exception x) {
             x.printStackTrace();
          }
       }
    }
```

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 8.6. **Розробка додатків для платформи NetBeans**
Номер заняття: 54

Мета: ознайомити студентів з платформою для розробки десктопних застосунків NetBeans Platform; продемонструвати побудову застосунків зі стандартним інтерфейсом користувача, який надається «стандартним» застосунком NetBeans на реальних прикладах.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/8wcp

Короткі теоретичні відомості:

Платформа NetBeans – це платформа Java, на якій можуть базуватись масштабні додатки для робочого середовища. Саме середовище IDE NetBeans є одним з сотень додатків на основі платформи NetBeans. Платформа NetBeans містить інтерфейси API, які спрощують обробку вікон, дій, файлів і багатьох інших об'єктів, а також функцій, типових для додатків.

Кожна можливість в додатку на платформі NetBeans може забезпечуватись окремим *модулем NetBeans*, який можна порівняти з модулем, що підключається. Модуль NetBeans – це *група класів* Java, яка надає додаток з певними можливостями.

Користувачі також можуть створювати *власні модулі* для середовища IDE NetBeans. Наприклад, користувачі можуть створювати модулі, що забезпечують доступ до популярних технологій та сервісів для користувачів IDE NetBeans. Також можна створити, наприклад, модуль, який буде надавати додаткові функції редагування.

В ході відеолекції було продемонстровано створення простих застосунків на платформі NetBeans з використанням стандартних шаблонів, які входять до складу IDE.

Детальні кроки процесу можна переглянути в офіційній документації платформи – https://college.page.link/o2Qt.

Також на офіційному сайті IDE доступні навчальна карта з платформи NetBeans (https://college.page.link/uo7u), і галерея додатків, створених на цій платформі (https://college.page.link/kmtB).

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

Тема 8.7. **Розповсюдження додатків для платформи NetBeans. Інсталятори. Bootstrap'ери. Портативні додатки**
Номер заняття: 55

Мета: ознайомити студентів з платформою для розробки десктопних застосунків NetBeans Platform; продемонструвати створення інсталяційних пакетів для застосунків, створених з використанням платформи.

Тип: заняття повідомлення нових знань

Структура заняття:

1. Організаційний момент (перевірка відвідування та готовності до заняття)
2. Повідомлення теми, мети і завдань заняття
3. Мотивація навчальної діяльності студентів (практичне застосування знань з теми, міжпредметні зв'язки)
4. Виклад нового матеріалу
5. Підведення підсумків (узагальнення, систематизація)
6. Контрольні питання
7. Пояснення домашнього завдання, рекомендація додаткових матеріалів

Відеозапис лекції:

частина 1: https://college.page.link/KzFR

Короткі теоретичні відомості:

Платформа NetBeans – це платформа Java, на якій можуть базуватись масштабні додатки для робочого середовища. Саме середовище IDE NetBeans є одним з сотень додатків на основі платформи NetBeans. Платформа NetBeans містить інтерфейси API, які спрощують обробку вікон, дій, файлів і багатьох інших об'єктів, а також функцій, типових для додатків.

Одна з чудових можливостей NetBeans – можливість *створення інсталяційних пакетів* для програм, створених з допомогою цього середовища (не обов'язково з використанням однойменної платформи).

Вбудовані засоби створення інсталяторів були вперше представлені як частина пакета SDK JavaFX 2.2, що дозволяє упакувати програму у вигляді інсталятора, а потім встановити та запустити програму без будь-яких зовнішніх залежностей на системному JRE або JavaFX SDK. Згодом ця можливість стала доступною і для проектів Java SE.

NetBeans не змінює модель розгортання вашої програми: він сприймає програму такою, якою вона є, упаковує її разом з середовищем виконання Java і створює інсталятор, стандартний для операційної системи, яку ви використовуєте. Суть полягає в тому, щоб зробити все це незалежним від того, чи мають користувачі Java на цільовій машині. Ви можете взяти такий інсталятор і запустити його на машині, де немає Java, й він встановить і додаток, і все необхідне для виконання Java. Розмір таких інсталяторів досить великий, тому що навіть "Привіт світ" додаток потягне за собою значну частину артефактів виконання Java.

В ході відеолекції було продемонстровано створення інсталяційних пакетів з допомогою NetBeans, а також – за допомогою популярних відкритих інсталяторів Inno Setup та Free Extractor. Ще один варіант – використання WiX Toolset.

Детальні кроки процесу можна переглянути на сайті IDE – https://college.page.link/f4Ud.

Додаткові матеріали:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t

ІНСТРУКТИВНО-МЕТОДИЧНІ МАТЕРІАЛИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ ТА ПРАКТИЧНИХ ЗАНЯТЬ

Змістовий модуль 1. Основні поняття ООП. Об'єктно-орієнтований аналіз та проектування. Огляд технології JAVA

Тема 1.1. **Поняття про об'єкти та класи**
Номер заняття: 2

Інструкція до виконання практичної роботи:

1. Відкрити тестове завдання за посиланням https://forms.gle/yLr4qptXXX75GjHJ8
2. Дати відповіді на питання
3. Здати завдання через віртуальний клас Google Classroom

Результат виконання роботи:

Виконане тестове завдання з теми.

Тема 1.2. **Взаємодія об'єктів**
Номер заняття: 4

Інструкція до виконання практичної роботи:

1. Отримати завдання в середовищі GitHub Classroom (стартовий код доступний за посиланням https://github.com/ppc-ntu-khpi/First-Starter, завдання 1)
2. клонувати створений для вас репозиторій в Netbeans, відкрити в Repl.It або просто завантажити його вміст у ZIP-форматі і відкрити в IDE за вибором
3. створити в теці *src* новий файл *Shirt.java*, в якому набрати текст з Readme репозиторію, зберегти файл
4. відкрити файл *ShirtTest.java* з цього репозиторію, запустити його
5. замінити у файлі *Shirt.java* номер сорочки, її опис та вартість довільними значеннями, запустити прроект повторно
6. зробити та зберегти в теці *Solution* скріншоти результатів роботи програми
7. створити (теж у теці *Solution*) файл *task1.md* і додати туди скріншоти (УВАГА! Має бути видно код програми!)
8. Закомітити зміни до репозиторію
9. Здати завдання через віртуальний клас Google Classroom, вказавши посилання на репозиторій

УВАГА! Readme репозиторію містить диференційовані завдання на «три», «чотири» та «п’ять».

Результат виконання роботи:

Репозиторій з вихідним кодом, Markdown-файл з описом виконаного завдання.

Тема 1.3. **Об’єктно-орієнтований аналіз та проектування з використанням UML. Аналіз проблеми та розробка алгоритму її вирішення**
Номер заняття: 5

Інструкція до виконання практичної роботи:

1. Отримати завдання в середовищі GitHub Classroom (стартовий код доступний за посиланням https://github.com/ppc-ntu-khpi/First-Starter, завдання 2)
2. клонувати створений для вас репозиторій в Netbeans, відкрити в Repl.It або просто завантажити його вміст у ZIP-форматі і відкрити в IDE за вибором
3. створити в теці *src* новий файл *Quotation.java*, в якому набрати текст з Readme репозиторію, зберегти файл
4. відкрити файл *QuotationTest.java* з цього репозиторію, запустити його
5. замінити текст у файлі *Quotation.java* довільною цитатою, знайденою в Мережі, запустити проект повторно
6. зробити та зберегти в теці *Solution* скріншоти результатів роботи програми
7. створити (теж у теці *Solution*) файл *task2.md* і додати туди скріншоти (УВАГА! Має бути видно код програми!)
8. закомітити зміни до репозиторію
9. здати завдання через віртуальний клас Google Classroom, вказавши посилання на репозиторій

УВАГА! Readme репозиторію містить диференційовані завдання на «три», «чотири» та «п’ять».

Результат виконання роботи:

Репозиторій з вихідним кодом, Markdown-файл з описом виконаного завдання.

Тема 1.4. **Огляд технології JAVA. Розробка та тестування JAVA-програми. Декларування, ініціалізація та використання змінних**
Номер заняття: 7

Інструкція до виконання практичної роботи:

1. Отримати завдання в середовищі GitHub Classroom (стартовий код доступний за посиланням https://github.com/ppc-ntu-khpi/Class-Starter, завдання 1-2)
2. клонувати створений для вас репозиторій в Netbeans, відкрити в Repl.It або просто завантажити його вміст у ZIP-форматі і відкрити в IDE за вибором
3. створити в пакеті *domain* новий файл *Customer.java,* з вказаними приватними атрибутами (зі значеннями за замовчуванням), зберегти файл
4. творити в пакеті *test* клас *CustomerTest*, в методі *main* якого створити об'єкт класу *Customer* та вивести на екран його властивості з допомогою методу *displayCustomerInfo*
5. зробити та зберегти в теці *Solution* скріншоти результатів роботи програми
6. змінити Readme репозиторію, додати туди скріншоти (УВАГА! Має бути видно код програми!)
7. закомітити зміни до репозиторію
8. здати завдання через віртуальний клас Google Classroom, вказавши посилання на репозиторій

УВАГА! Readme репозиторію містить диференційовані завдання на «три», «чотири» та «п’ять».

Результат виконання роботи:

Репозиторій з вихідним кодом, Markdown-файл з описом виконаного завдання.

Змістовий модуль 2. Складні типи даних. основні алгоритмічні конструкції. методи

Тема 2.2. **Використання операторів та алгоритмічних конструкцій. Використання циклів**

Номер заняття: 12

Інструкція до виконання практичної роботи:

1. Відкрити тестове завдання за посиланням https://forms.gle/t1cAENvzL4EahXpHA
2. Дати відповіді на питання
3. Здати завдання через віртуальний клас Google Classroom

Результат виконання роботи:

Виконане тестове завдання з теми.

Тема 2.3. **Розробка та використання методів**
Номер заняття: 14

Інструкція до виконання практичної роботи:

1. Отримати завдання в середовищі GitHub Classroom (стартовий код доступний за посиланням https://github.com/ppc-ntu-khpi/identifiers-types-starter)
2. модифікувати стартовий код таким чином, щоб метод *Calculate* класу *Exercise* містив код обчислення значення у відповідності до вашого варіанту (у разі необхідності можна додавати до класу нові приватні методи)
3. змінити Readme репозиторію, додати туди скріншоти (УВАГА! Має бути видно код програми!)
4. закомітити зміни до репозиторію
5. здати завдання через віртуальний клас Google Classroom, вказавши посилання на репозиторій

УВАГА! Readme репозиторію містить 24 варіанти завдань.

Результат виконання роботи:

Репозиторій з вихідним кодом, Markdown-файл з описом виконаного завдання.

Змістовий модуль 3. Інкапсуляція. Успадкування та повторне використання коду
Тема 3.1. **Інкапсуляція та конструктори**
Номер заняття: 16

Інструкція до виконання практичної роботи:

1. Відкрити тестове завдання за посиланням https://forms.gle/uWg5LfYy3PsuhkWf9
2. Дати відповіді на питання
3. Здати завдання через віртуальний клас Google Classroom

Результат виконання роботи:

Виконане тестове завдання з теми.

Тема 3.2. **Створення та використання масивів**
Номер заняття: 18

Інструкція до виконання практичної роботи:

1. Отримати завдання в середовищі GitHub Classroom (стартовий код доступний за посиланням https://github.com/ppc-ntu-khpi/control-arrays-Starter)
2. модифікувати стартовий код таким чином, щоб метод *Calculate* класу *Exercise* містив код обчислення значення у відповідності до вашого варіанту (у разі необхідності можна додавати до класу нові приватні методи)
3. змінити Readme репозиторію, додати туди скріншоти (УВАГА! Має бути видно код програми!)
4. закомітити зміни до репозиторію
5. здати завдання через віртуальний клас Google Classroom, вказавши посилання на репозиторій

УВАГА! Readme репозиторію містить 17 варіантів завдань.

Результат виконання роботи:

Репозиторій з вихідним кодом, Markdown-файл з описом виконаного завдання.

Тема 3.3. **Реалізація успадкування**
Номер заняття: 20

Інструкція до виконання практичної роботи:

1. Отримати завдання в середовищі GitHub Classroom (стартовий код доступний за посиланням https://github.com/ppc-ntu-khpi/Inheritance-Starter)
2. обрати тварину, з допомогою *easyUML* для *Netbeans* або *StarUML* створити діаграму класів для обраної тварини
3. згенерувати на основі діаграми каркасний код, допрацювати його у відповідності до завдання
4. змінити Readme репозиторію, додати туди діаграму та зображення тварини
5. закомітити зміни до репозиторію
6. здати завдання через віртуальний клас Google Classroom, вказавши посилання на репозиторій

УВАГА! Readme репозиторію містить детальні відеоінструкції, доступні також за посиланням https://youtu.be/SFSC1omkE8Q.

Результат виконання роботи:

Репозиторій з вихідним кодом, Markdown-файл з описом виконаного завдання.

Змістовий модуль 4. Розробка об'єктно-орієнтованих програм мовою JAVA та їх документування

Тема 4.3. **Об'єктно-орієнтоване програмування. Документування програм**
Номер заняття: 24

Інструкція до виконання практичної роботи:

1. Використати код, створений в результаті виконання попереднього (стартовий код доступний за посиланням https://github.com/ppc-ntu-khpi/Inheritance-Starter)
2. з попомогою *EasyCodeForAll* додати до коду *javadoc*-коментарі
3. завантажити й встановити *Doxygen*
4. згенерувати з його допомогою *html*-документацію для проекту
5. завантажити отриману документацію до теки */docs* того ж репозиторію
6. увімкнути в налаштуваннях репозиторію *GitHub Pages* (для теки */docs*)
7. здати завдання через віртуальний клас Google Classroom, вказавши посилання на репозиторій та на опублікований сайт документації

УВАГА! Детальні відеоінструкції до виконання практичної доступні за посиланням https://youtu.be/NRhsIuPs8BI.

Результат виконання роботи:

Репозиторій з вихідним кодом, Markdown-файл з описом виконаного завдання.

Змістовий модуль 5. Основні можливості платформи JAVA

Тема 5.1. **Нові ідентифікатори, ключові слова та типи даних**
Номер заняття: 26

Інструкція до виконання практичної роботи:

1. Відкрити тестове завдання за посиланням https://forms.gle/vVR56Ho7ZRZn4vPr5
2. Дати відповіді на питання
3. Здати завдання через віртуальний клас Google Classroom

Результат виконання роботи:

Виконане тестове завдання з теми.

Тема 5.2. **Поглиблене використання виразів, керування виконанням програми**
Номер заняття: 29

Інструкція до виконання практичної роботи:

1. Відкрити тестове завдання за посиланням https://forms.gle/ycCCLQB7zBQPnC3a7
2. Дати відповіді на питання
3. Здати завдання через віртуальний клас Google Classroom

Результат виконання роботи:

Виконане тестове завдання з теми.

Тема 5.3. **Поглиблене використання масивів**
Номер заняття: 31

Інструкція до виконання практичної роботи:

1. Отримати завдання в середовищі GitHub Classroom (стартовий код доступний за посиланням https://github.com/ppc-ntu-khpi/advanced-arrays-Starter)
2. написати клас, який містить методи для розв'язання обраного вами завдання та тестовий клас, який дозволяє перевірити його роботу
3. змінити Readme репозиторію, додати туди скріншоти (УВАГА! Має бути видно код програми!)
4. закомітити зміни до репозиторію
5. здати завдання через віртуальний клас Google Classroom, вказавши посилання на репозиторій

УВАГА! Readme репозиторію містить 16 варіантів завдань.

Результат виконання роботи:

Репозиторій з вихідним кодом, Markdown-файл з описом виконаного завдання.

Змістовий модуль 6. Проектування та реалізація архітектури програм. Забезпечення якості програмних продуктів

Тема 6.2. **Рефакторинг. Типові архітектурні рішення та антипатерни**
Номер заняття: 36

Інструкція до виконання практичної роботи:

1. Отримати завдання в середовищі GitHub Classroom (стартовий код доступний за посиланням https://github.com/ppc-ntu-khpi/UML-Starter)
2. проаналізувати предметну область завдання, з допомогою будь-якого з рекомендованих інструментів створити діаграму класів
3. згенерувати на основі діаграми каркасний код, допрацювати його у відповідності до завдання (скористатись вбудованими в NetBeans засобами рефакторингу). УВАГА! Подбати про застосування патернів проектування!
4. змінити Readme репозиторію, додати туди діаграму та опис завдання
5. закомітити зміни до репозиторію
6. здати завдання через віртуальний клас Google Classroom, вказавши посилання на репозиторій

УВАГА! Readme репозиторію містить 5 варіантів завдань (є можливість виконання завдання групою студентів).

Результат виконання роботи:

Репозиторій з вихідним кодом, Markdown-файл з описом виконаного завдання.

Тема 6.3. **Обробка помилок та виключень. Відлагодження, тестування та профілювання**
Номер заняття: 38

Інструкція до виконання практичної роботи:

1. Відкрити тестове завдання за посиланням https://forms.gle/iXMbzWnyAyJt3VRF8
2. Дати відповіді на питання
3. Здати завдання через віртуальний клас Google Classroom

Результат виконання роботи:

Виконане тестове завдання з теми.

Тема 6.4. **Колекції та дженерики**
Номер заняття: 40

Інструкція до виконання практичної роботи:

1. Відкрити тестове завдання за посиланням https://forms.gle/dbdAoKsYVMVRqrGK9
2. Дати відповіді на питання

3. Здати завдання через віртуальний клас Google Classroom

Результат виконання роботи:

Виконане тестове завдання з теми.

Змістовий модуль 7. Консоль. Файлова система. Графічний Інтерфейс користувача

Тема 7.1. **Основи вводу-виводу. Робота с консоллю та файловою системою**
Номер заняття: 43

Інструкція до виконання практичної роботи:

1. Отримати завдання в середовищі GitHub Classroom (стартовий код доступний за посиланням https://github.com/ppc-ntu-khpi/TUI-Lab1-Starter, https://github.com/ppc-ntu-khpi/TUI-Lab2-Starter)
2. виконати завдання у відповідності до Readme репозиторію
3. змінити Readme репозиторію, додати туди діаграму та опис завдання
4. закомітити зміни до репозиторію
5. здати завдання через віртуальний клас Google Classroom, вказавши посилання на репозиторій

УВАГА! Readme репозиторію містить диференційовані завдання на «три», «чотири» та «п’ять», а також завдання двох рівнів складності на вибір студента.

Результат виконання роботи:

Репозиторій з вихідним кодом, Markdown-файл з описом виконаного завдання.

Тема 7.4. **Обробка подій від інтерфейсних елементів**
Номер заняття: 47

Інструкція до виконання практичної роботи:

1. Отримати завдання в середовищі GitHub Classroom (стартовий код доступний за посиланням https://github.com/ppc-ntu-khpi/GUI-Lab1-Starter, https://github.com/ppc-ntu-khpi/GUI-Lab2-Starter, https://github.com/ppc-ntu-khpi/GUI-Lab3-Starter)
2. виконати завдання у відповідності до Readme репозиторію
3. змінити Readme репозиторію, додати туди діаграму та опис завдання
4. закомітити зміни до репозиторію

5. здати завдання через віртуальний клас Google Classroom, вказавши посилання на репозиторій

УВАГА! Readme репозиторію містить диференційовані завдання на «три», «чотири» та «п’ять», а також завдання двох рівнів складності на вибір студента.

Результат виконання роботи:

Репозиторій з вихідним кодом, Markdown-файл з описом виконаного завдання.

Змістовий модуль 8. Багатопоточність. Мережеві можливості. Платформа Netbeans

Тема 8.2. **Робота з мережею**

Номер заняття: 52

Інструкція до виконання практичної роботи:

1. Отримати завдання в середовищі GitHub Classroom (стартовий код доступний за посиланням https://github.com/ppc-ntu-khpi/Sockets-Starter)
2. виконати завдання у відповідності до Readme репозиторію
3. змінити Readme репозиторію, додати туди діаграму та опис завдання
4. закомітити зміни до репозиторію
5. здати завдання через віртуальний клас Google Classroom, вказавши посилання на репозиторій

УВАГА! Readme репозиторію містить диференційовані завдання на «три», «чотири» та «п’ять».

Результат виконання роботи:

Репозиторій з вихідним кодом, Markdown-файл з описом виконаного завдання.

МЕТОДИЧНІ МАТЕРІАЛИ, ЩО ЗАБЕЗПЕЧУЮТЬ САМОСТІЙНУ РОБОТУ СТУДЕНТІВ

З метою забезпечення самостійної роботи студентів пропонується ряд безкоштовних та вільно доступних у Мережі тренінгів, книг та сертифікаційних курсів. Особливу увагу слід звернути на курс від Oracle Academy – офіційний тренінг від вендора-володаря торгівельної марки Java, доступний студентам коледжу в рамках партнерської угоди між компанією Oracle та ВСП «ППФК НТУ «ХПІ».

1. JFo основи Java / Oracle Academy – https://myacademy.oracle.com/
2. Java-курс від Принстонського університету:
   вміст курсу – https://introcs.cs.princeton.edu/java/home/
   шпаргалка з Java – https://introcs.cs.princeton.edu/java/11cheatsheet/
3. Вікіпідручник «Освоюємо Java» – https://college.page.link/GK8t
4. Програмування мовою Java для дітей, батьків, дідусів та бабусь:
   http://myflex.org/books/java4kids/java4kids.htm
5. Курс «Java Professional» – https://college.page.link/Upp4
6. Andrei Dmitriev's Java SE Course – https://edu.netbeans.org/contrib/slides/dmitriev/
7. Learn Java / Sololearn – https://www.sololearn.com/Course/Java/
8. Програмування на Java / Віртуальна академія – https://college.page.link/2TyR
9. Java Tutorial / w3schools – https://www.w3schools.com/java/
10. Основи програмування на Java / Prometheus – https://college.page.link/A6vH

НАВЧАЛЬНО-МЕТОДИЧНІ МАТЕРІАЛИ ДЛЯ ПРОМІЖНОГО КОНТРОЛЮ ТА ПЕРЕЛІК ПИТАНЬ ПІДСУМКОВОГО КОНТРОЛЮ

Проміжний та підсумковий (у деяких випадках, див. наступний розділ НМК) контроль здійснюється з допомогою автоматизованого тестування. Проміжне тестування – на платформі Google Classroom з використанням Google Forms. Посилання на тести доступні у віртуальному Класі в день тестування, деякі з них наведені в розділі практичних робіт НМК.

Підсумковий контроль (за умови виконання всіх практичних робіт) здійснюється на платформі ClassMarker (посилання доступне у віртуальному Класі в день іспиту). Фінальний тест містить питання теоретичного (40) і практичного спрямування (87) загальною кількістю 127 питань, з яких під час іспиту для кожного студента випадковим чином обирається 30 питань. Перелік теоретичних питань наведено у наступному розділі НМК.

Перелік вибраних завдань практичного спрямування:

1. Побудувати систему класів для опису плоских геометричних фігур: кола, квадрата, прямокутника. Передбачити методи для створення об'єктів, переміщення на площині, зміни розмірів і повороту на заданий кут.
2. Побудувати опис класу, що містить інформацію про поштову адресу організації. Передбачити можливість окремої зміни складових частин адреси, створення і знищення об'єктів цього класу.
3. Скласти опис класу для подання комплексних чисел з можливістю ініціалізації дійсної і уявною частин як числами типу double, так і цілими числами. Забезпечити виконання операцій додавання, віднімання і множення комплексних чисел.
4. Скласти опис класу для роботи з ланцюговими списками рядків (рядки довільної довжини) з операціями включення в список, видалення зі списку елемента з заданим значенням, видалення всього списку або кінця списку, починаючи з заданого елемента.
5. Скласти опис класу для об'єктів-векторів, що задаються координатами кінців в тривимірному просторі. Забезпечити операції додавання і віднімання векторів з отриманням нового вектора (суми або різниці), обчислення скалярного добутку двох векторів, довжини вектора, косинуса кута між векторами.
6. Скласти опис класу прямокутників зі сторонами, паралельними вісям координат. Передбачити можливість переміщення прямокутників на площині, зміни розмірів,

побудови найменшого прямокутника, що містить два заданих прямокутника, і прямокутника, що є спільною частиною (перетином) двох прямокутників.

7. Скласти опис класу для визначення одновимірних масивів цілих чисел (векторів). Передбачити можливість звернення до окремого елементу масиву з контролем виходу за межі індексів, можливість завдання довільних меж індексів при створенні об'єкта і виконання операцій поелементного додавання і віднімання масивів з однаковими межами індексів, множення і ділення всіх елементів масиву на скаляр, друку (виведення на екран) елементів масиву за індексами і всього масиву.
8. Скласти опис класу для визначення одновимірних масивів рядків фіксованої довжини. Передбачити можливість звернення до окремих рядків масиву за індексами, контроль виходу за межі індексів, виконання операцій поелементного зчеплення двох масивів з утворенням нового масиву, злиття двох масивів з виключенням повторюваних елементів, друк (виведення на екран) елементів масиву і всього масиву.
9. Скласти опис класу многочленів від однієї змінної, що задаються ступенем многочлена і масивом коефіцієнтів. Передбачити методи для обчислення значення многочлена для заданого аргументу, операції додавання, віднімання і множення многочленів з отриманням нового об'єкта-многочлена, друк (виведення на екран) опису многочлена.
10. Скласти опис класу одновимірних масивів рядків, кожен рядок яких задається довжиною і покажчиком на виділену для неї пам'ять. Передбачити можливість звернення до окремих рядків масиву за індексами, контроль виходу за межі індексів, виконання операцій поелементного зчеплення двох масивів з утворенням нового масиву, злиття двох масивів з виключенням повторюваних елементів, друк (виведення на екран) елементів масиву і всього масиву.
11. Скласти опис об'єктного типу TMatr, що забезпечує розміщення матриці довільного розміру з можливістю зміни числа рядків і стовпців, виведення на екран підматриці будь-якого розміру і всієї матриці.
12. Скласти програму, що працює зі зв'язаними списками. Ми будемо розглядати зв'язаний список як об'єкт, що містить зв'язаний список даних і операцій (методів), які ви можете з ними виконувати. Зв'язаний список даних складається з покажчиків на початок («голову») і кінець («хвіст») зв'язаного списку (в нашому прикладі через його гнучкість використовується двонаправлений зв'язаний список). Кожен елемент зв'язаного списку є реалізацією окремого об'єкта. Методи, необхідні для використання зв'язаного списку, надають наступні операції:
    - створення зв'язаного списку (виділення для нього пам'яті);

- знищення зв'язаного списку (звільнення використовуваної пам'яті);
- ініціалізація зв'язаного списку;
- деініціалізацію зв'язаного списку;
- вставка елемента в середину списку перед існуючим елементом;
- приєднання елемента до кінця зв'язаного списку;
- видалення елемента з зв'язаного списку;
- повернення першого елемента зв'язаного списку;
- повернення останнього елемента зв'язаного списку. Необхідно мати на увазі, що створення і ініціалізація, а також знищення і деініціалізація – це не синоніми. При створенні і знищенні методи create і destroy виділяють і звільняють пам'ять для об'єкта (зв'язаного списку), а методи ініціалізації і деініціалізації initialize і deinitialize тільки ініціалізують і деініціалізують раніше виділені екземпляри об'єкта. Ви можете бачити, як об'єкт зв'язаного списку успадковується об'єктами стека або черги, оскільки чергу і стек можна реалізувати як зв'язаний список з обмеженим числом операцій. Наприклад, можна реалізувати чергу у вигляді зв'язаного списку, в якому елементи можуть додаватися до кінця і вилучатись з початку. Якщо ви таким чином реалізуєте чергу, то потрібно заборонити успадковані методи зв'язаного списку,

13. Визначити об'єкт TFish – акваріумну рибку. Рибка має координати, швидкість, розмір, колір, напрямок руху. Методами об'єкта є:
    - Init – встановлює значення полів об'єкта і малює рибу на екрані методом Draw.
    - Draw – малює рибу у вигляді куточка з вістрям в точці Coord і спрямованого вістрям по ходу руху риби.
    - Look – перевіряє кілька точок на траєкторії риби. Якщо хоч одна з них відрізняється за кольором від води, повертаються її колір і відстань до риби.
    - Run – переміщує рибу в поточному напрямі на відстань, що залежить від поточної швидкості риби. Іноді випадковим чином змінює напрямок руху риби. Якщо риба бачить перешкоду, напрямок руху змінюється, доки перешкода не зникне з поля зору риби.
14. Визначити об'єкт TAquarium, який є місцем проживання риб (див. завдання 13). Він являє собою область екрану, наповнену водою. Риби живуть в акваріумі, тому екземпляри об'єкта TFish мають бути полями об'єкта TAquarium. Методи:
    - Init – вмикає графічний режим, заповнює акваріум водою, камінням і рибами.

- Run – організовує нескінченний цикл, в якому виконується метод Run всіх мешканців акваріума.
- Done – вимикає графічний режим.

15. Визначити два об'єкти TPike і TKarp, які успадковують об'єкт Tfish (див. завдання 15). Обидва вони відрізняються від TFish тим, що по-різному зображують себе на екрані: TPike – у вигляді зеленої стрілки, а TKarp – у вигляді червоного трикутника. Скористайтеся віртуальними методами. Для цього поверніться до визначення TFish і відкоригуйте його, зробивши Draw порожнім і віртуальним.
16. Об'єднати коропів і щук (див. завдання 15) в дві зграї. Зграя – це зв'язаний список риб в динамічній пам'яті. Для зв'язку додайте в об'єкти TPike і TKarp поле Next – покажчик на наступну рибу в зграї. Зробіть акваріум власником не окремих риб, а двох зграй і дозвольте користувачеві поповнювати зграї, вводячи риб з клавіатури. Дозвольте щукам (див. завдання 15) проявити свій поганий характер і поїдати коропів, як тільки вони їх побачать. Тут виникне проблема – встановити, якого саме коропа бачить щука. Вона вирішується шляхом перегляду всієї зграї коропів і пошуку того, чиї координати близькі до координат щуки. Знайдений короп видаляється зі зграї.
17. Скласти програму для гри в шашки. Шашка кожного нового кольору виступає в якості окремого об'єкта. Характеристики шашки – колір і позиція на дошці. Методи – переміщення. Не забудьте про такі об'єкти, як «дамки».
18. Скласти програму для гри в доміно. Як об'єкти виступають кістки доміно. Методи – способи виставлення тієї чи іншої кістки.
19. Скласти програму для гри в шахи. Кожна унікальна шахова фігура виступає в якості окремого об'єкта. Вона характеризується кольором, положенням на дошці, способом переміщення. Передбачити можливість перетворення пішака на ферзя.

Решта завдань практичного спрямування, які пропонуються студентам під час проміжного та підсумкового контролю, запозичені з курсу "Основи програмування на Java" (https://college.page.link/A6vH), який також входить до переліку рекомендованих матеріалів для самостійного опрацювання.

ПИТАННЯ ДЛЯ ПІДГОТОВКИ ДО ЕКЗАМЕНУ

Перелік питань до екзамену:

1. Мова програмування Java. Особливості мови. Поняття про подвійну компіляцію, байт-код, віртуальну машину.
2. Можливості IDE Netbeans стосовно форматування, вставки коду та рефакторингу.
3. Життєвий цикл програмного продукту. UML-проектування в середовищі Netrbeans. Кодогенерація та зворотний інжиніринг.
4. Система типів й пакетів у Java Core. Поняття об’єкту, класу, атрибутів та операцій.
5. Структурна будова класу, класові й екземплярні дані й методи. Ключові слова this та super.
6. Абстрактні класи та інтерфейси. Методи за замовчуванням у інтерфейсах.
7. Застосування ключового слова final у оголошенні класів, методів, полів.
8. Створення методу, способи передачі параметрів, повернення значення з методу, метод зі змінною кількістю параметрів.
9. Базові (фундаментальні) типи даних. Класи обгортки autoboxing/unboxing.
10. Успадкування й перевизначення методів. Приведення типів, оператор instanceof.
11. Внутрішні (inner) класи, їхнє призначення й використання.
12. Вкладені (nested) статичні класи, їхнє призначення й використання.
13. Локальні класи, їхнє призначення й використання.
14. Анонімні класи, їхнє призначення й використання.
15. Оператори розгалуження та циклів.
16. Оператори дострокового виходу з циклу або переходу до наступної ітерації. Завершення роботи програми.
17. Клас Object, його призначення та основні методи.
18. Поняття виключень, обробка й генерація виключень. Ієрархія класів-виключень.
19. Тип перелічення (enum), його призначення.
20. Модифікатори доступу в Java, приклади.
21. Статичні поля й методи та поля й методи об’єкта, приклади.
22. Способи ініціалізації полів об’єкту: у оголошенні, конструкторі й секції ініціалізації. Як відбувається ініціалізація даних екземпляру класу, успадкованих від супер-класу.
23. Основні оператори мови Java. Логічні й побітові операції. Тернарний оператор.
24. Підтримка поліморфізму у Java. Приклади.
25. Поняття пакету (package). Створення й використання пакетів у програмі.

26. Поняття про графічний інтерфейс користувача. Swing, Контейнери та менеджери розмітки.
27. Обробка подій від елементів керування в графічних додатках.
28. Робота з потоками. Файлове введення-виведення.
29. Робота з потоками. Серіалізація та десеріалізація об’єктів.
30. Реалізація багатозадачності. Керування потоками.
31. Консольне введення-виведення. Форматування рядків.
32. Системні властивості. Робота з параметрами командного рядка.
33. Реалізація багатозадачності. Одначасний доступ до даних та синхронізація.
34. Мережеві можливості Java. Сокети.
35. Мережеві можливості Java. Основні класи пакету java.net.
36. Платформа Netbeans. Основні особливості та переваги. Підходи до розробки.
37. Основні принципи ООП, їх реалізація в Java.
38. Документування Java-програм. Javadoc-коментарі.
39. Порівняння Java та Microsoft .NET.
40. Перевизначення та перевантаження методів.

Онлайнові підручники для підготовки до екзамену:

- Вікіпідручник «Объектно-ориентированное программирование» https://college.page.link/mb77
- Вікіпідручник «Java» – https://ru.wikibooks.org/wiki/Java

Тестові завдання:

Дивись курс "Основи програмування на Java" (https://college.page.link/A6vH) – тестові завдання цього курсу лежать в основі фінального тесту з дисципліни, який студенти складають на платформі ClassMarker - https://www.classmarker.com/a/

Додаткові зауваження:

- якщо студент не має поточних не перезданих «двійок» – іспит складається у вигляді фінального тесту на платформі ClassMarker
- якщо є «двійки», «н/а» тощо – студент складає «класичний» іспит (за білетами)

– якщо є сертифікат курсу від Prometheus або Oracle Academy і успішно виконав всі практичні роботи – студент звільняється від екзамену й отримує «відмінно»
– додаткові преференції на екзамені отримують учасники фахових конкурсів, змагань та олімпіад (за умови успішного виконання всіх практичних робіт)

КОМПЛЕКС КОНТРОЛЬНИХ РОБІТ: ДЛЯ ВИЗНАЧЕННЯ ЗАЛИШКОВИХ ЗНАНЬ З ДИСЦИПЛІНИ (ККР)

ЗАГАЛЬНІ ПОЛОЖЕННЯ

Комплексна контрольна робота з дисципліни “Об’єктно-орієнтоване програмування” розроблена згідно програми та навчального плану і передбачає написання 30 варіантів (індивідуально) для групи з 25-28 здобувачів освіти (студентів). Комплексна контрольна робота призначена для визначення якісного рівня підготовки здобувачів освіти з дисципліни.

Виконання завдань роботи розраховане на 2 академічні години. Робота передбачає перевірку знань здобувачів освіти на тому етапі, коли викладання дисципліни “Об’єктно-орієнтоване програмування” завершено, і цілком базується на знаннях, одержаних на лекціях, практичних та лабораторних роботах, індивідуальній роботі та при підготовці завдань самостійної роботи.

Зміст відповідей на поставлені запитання дає уявлення про можливість практичного використання вивченого теоретичного матеріалу, а викладач має змогу перевірити рівень знань здобувачів освіти після вивчення курсу.

АНОТАЦІЯ ДИСЦИПЛІНИ

Програма вивчення нормативної навчальної дисципліни "Об’єктно-орієнтоване програмування" складена відповідно до освітньо-професійної програми спеціальності «Розробка програмного забезпечення».

Предметом вивчення навчальної дисципліни є методи та засоби проектування та розробки складних об’єктно-орієнтованих програмних систем з використанням платформи JAVA.

Міждисциплінарні зв'язки:
- Основи програмної інженерії
- Конструювання програмного забезпечення
- Алгоритми та структури даних
- Основи програмування та алгоритмічні мови

Програма навчальної дисципліни складається з таких змістових модулів:
- Основні поняття ООП. Об’єктно-орієнтований аналіз та проектування. Огляд технології JAVA
- Складні типи даних. Основні алгоритмічні конструкції. Методи
- Інкапсуляція. Успадкування та повторне використання коду
- Розробка об’єктно-орієнтованих програм мовою JAVA та їх документування
- Основні можливості платформи JAVA
- Проектування та реалізація архітектури програм. Забезпечення якості програмних продуктів
- Консоль. Файлова система. Графічний Інтерфейс користувача
- Багатопоточність. Мережеві можливості. Платформа Netbeans

Мета та завдання навчальної дисципліни

Мета викладання навчальної дисципліни "Об’єктно-орієнтоване програмування" – дати здобувачам освіти уявлення, початкові знання та практичні навички з сучасних тенденцій проектування та розробки складних програмних систем на прикладі платформи JAVA.

Основними завданнями вивчення дисципліни "Об’єктно-орієнтоване програмування" є ознайомлення з принципами, методами і засобами об'єктно-орієнтованого програмування, отримання уявлення про платформу та мову програмування JAVA, ознайомлення з шаблонами проектування та типовими архітектурними рішеннями під час проектування програмних систем, основами

документування проектних рішень, формування практичних навичок створення програмних систем за допомогою мови JAVA та IDE NetBeans.

Згідно з вимогами освітньо-професійної програми здобувачі освіти повинні:
знати:

- Основні поняття та методи об'єктно-орієнтованого аналізу та проектування
- Особливості платформи JAVA та синтаксис мови програмування JAVA
- Основні підходи до проектування сучасних програмних систем
- Основні методи забезпечення якості програмних продуктів, види тестування

вміти:

- Застосовувати ключові концепції об'єктно-орієнтованого програмування, такі як інкапсуляція, успадкування та поліморфізм
- Виявляти об'єкти предметної області, розробляти діаграми класів для проблемної області, створювати ієрархію класів, засновану на діаграмах класів
- Проектувати додатки за допомогою UML-діаграм, виконувати кодогенерацію та зворотне проектування
- Розробляти і тестувати JAVA-застосунки, компілювати в байт-код та запускати JAVA-програми на виконання
- Виконувати рефакторинг існуючого коду, налагоджувати, тестувати і профілювати свої проекти
- Документувати свої додатки
- Створювати графічні JAVA-застосунки з використанням відповідних компонентів Swing API
- Реалізувати введення/виведення для роботи з файловою системою
- Створювати багатопоточні JAVA-застосунки
- Використовувати сокети для організації взаємодії в реальному часі за протоколом TCP/IP
- Розробляти додатки для платформи Netbeans

Сформовані компетенції

- ціннісно-змістовна (свідома організація власної діяльності під час вирішення типових завдань обробки даних та розробки простих алгоритмів автоматизації рутинних операцій та типових обрахунків)

– інформаційна (вміння знаходити потрібну інформацію користуючись усіма доступними джерелами і обмінюватись нею з колегами)
– комунікативна (здатність до вдосконалення навичок командної роботи, вміння працювати на результат, доводити власну думку, вести діалог)

На вивчення навчальної дисципліни відводиться 216 години 4 /6 кредити нац./ECTS.

ТЕМАТИЧНИЙ ПЛАН КУРСУ

Змістовий модуль 1. Основні поняття ООП. Об'єктно-орієнтований аналіз та проектування. Огляд технології JAVA

Тема 1.1. Поняття про об'єкти та класи
Тема 1.2. Взаємодія об'єктів
Тема 1.3. Об'єктно-орієнтований аналіз та проектування з використанням UML. Аналіз проблеми та розробка алгоритму її вирішення
Тема 1.4. Огляд технології JAVA. Розробка та тестування JAVA-програми. Декларування, ініціалізація та використання змінних
Тема 1.5. Підготовка середовища виконання та розробки JAVA-застосунків. JRE, SDK та JDK
Тема 1.6. Інші технології подвійного компілювання. Microsoft .NET
Тема 1.7. Підведення підсумків. Підсумкове тестування

Змістовий модуль 2. Складні типи даних. Основні алгоритмічні конструкції.Методи

Тема 2.1. Створення та використання об'єктів
Тема 2.2. Використання операторів та алгоритмічних конструкцій. Використання циклів
Тема 2.3. Розробка та використання методів

Змістовий модуль 3. Інкапсуляція. Успадкування та повторне використання коду

Тема 3.1. Інкапсуляція та конструктори
Тема 3.2. Створення та використання масивів
Тема 3.3. Реалізація успадкування

Змістовий модуль 4. Розробка об'єктно-орієнтованих програм мовою JAVA та їх документування

Тема 4.1. JAVA як платформа. IDE NetBeans
Тема 4.2. Інструменти продуктивності розробника в NetBeans
Тема 4.3. Об'єктно-орієнтоване програмування. Документування програм

Змістовий модуль 5. Основні можливості платформи JAVA

Тема 5.1. Ідентифікатори, ключові слова та типи даних
Тема 5.2. Вирази, керування виконанням програми
Тема 5.3. Поглиблене використання масивів

Змістовий модуль 6. Проектування та реалізація архітектури програм. Забезпечення якості програмних продуктів

Тема 6.1. Проектування ієрархії класів. Використання UML. Особливості створення класів
Тема 6.2. Рефакторинг. Типові архітектурні рішення та антипатерни

Тема 6.3. Обробка помилок та виключень. Відлагодження, тестування та профілювання
Тема 6.4. Колекції та дженерики

Змістовий модуль 7. Консоль. Файлова система. Графічний Інтерфейс користувача
Тема 7.1. Основи вводу-виводу. Робота с консоллю та файловою системою
Тема 7.2. Робота з базами даних. JDBC. Entity Classes.
Тема 7.3. Створення графічного інтерфейсу користувача
Тема 7.4. Обробка подів від інтерфейсних елементів
Тема 7.5. Тонке налагодження інтерфейсу користувача

Змістовий модуль 8. Багатопоточність. Мережеві можливості. Платформа Netbeans
Тема 8.1. Багатопоточність в JAVA
Тема 8.2. Робота з мережею
Тема 8.3. Створення та використання веб-сервісів. REST
Тема 8.4. Створення JAVA-аплетів. JSP
Тема 8.5. Розробка додатків для платформи NetBeans
Тема 8.6. Розповсюдження додатків для платформи NetBeans. Інсталятори. Bootstrap’ери. Портативні додатки
Тема 8.7. Підведення підсумків. Підсумкове тестування

ПЕРЕЛІК ТЕМ ДО ВАРІАНТІВ КОНТРОЛЬНОЇ РОБОТИ

1. Поняття про об’єкти та класи
2. Взаємодія об’єктів
3. Об’єктно-орієнтований аналіз та проектування з використанням UML. Аналіз проблеми та розробка алгоритму її вирішення
4. Огляд технології JAVA. Розробка та тестування JAVA-програми. Декларування, ініціалізація та використання змінних
5. Підготовка середовища виконання та розробки JAVA-застосунків. JRE, SDK та JDK
6. Інші технології подвійного компілювання. Microsoft .NET
7. Створення та використання об’єктів
8. Використання операторів та алгоритмічних конструкцій. Використання циклів
9. Розробка та використання методів
10. Інкапсуляція та конструктори
11. Створення та використання масивів
12. Реалізація успадкування
13. JAVA як платформа. IDE NetBeans
14. Інструменти продуктивності розробника в NetBeans
15. Об’єктно-орієнтоване програмування. Документування програм
16. Ідентифікатори, ключові слова та типи даних
17. Вирази, керування виконанням програми
18. Поглиблене використання масивів
19. Проектування ієрархії класів. Використання UML. Особливості створення класів
20. Рефакторинг. Типові архітектурні рішення та антипатерни
21. Обробка помилок та виключень. Відлагодження, тестування та профілювання
22. Колекції та дженерики
23. Основи вводу-виводу. Робота с консоллю та файловою системою
24. Робота з базами даних. JDBC. Entity Classes.
25. Створення графічного інтерфейсу користувача
26. Обробка подів від інтерфейсних елементів
27. Тонке налагодження інтерфейсу користувача
28. Багатопоточність в JAVA
29. Робота з мережею
30. Створення та використання веб-сервісів. REST
31. Створення JAVA-аплетів. JSP
32. Розробка додатків для платформи NetBeans
33. Розповсюдження додатків для платформи NetBeans. Інсталятори. Bootstrap’ери. Портативні додатки

КРИТЕРІЇ ОЦІНЮВАННЯ ЗНАНЬ ТА ВМІНЬ (У 100-БАЛЬНІЙ ШКАЛІ, ШКАЛІ ECTS ТА ЗА НАЦІОНАЛЬНОЮ ШКАЛОЮ)

- **100–88 балів – оцінка A (“відмінно” – 5)** виставляється за високий рівень знань (допускаються деякі неточності) навчального матеріалу, вміння аналізувати явища, які вивчаються, у їхньому взаємозв’язку і розвитку, чітко, лаконічно, логічно, послідовно відповідати на поставлені запитання, вміння застосовувати теоретичні положення під час розв’язання практичних задач;
- **87–80 балів – оцінка B (“дуже добре” – 4)** виставляється за знання навчального матеріалу вище від середнього рівня, включаючи розрахунки, аргументовані відповіді на поставлені запитання (можлива невелика кількість неточностей), вміння застосовувати теоретичні положення під час розв’язання практичних задач;
- **79–71 бал – оцінка C (“добре” – 4)** виставляється за загалом правильне розуміння навчального матеріалу, включаючи розрахунки, аргументовані відповіді на поставлені запитання, які, однак, містять певні (неістотні) недоліки, за вміння застосовувати теоретичні положення під час розв’язання практичних задач;
- **70–61 бал – оцінка D (“посередньо” – 3)** виставляється за посередні знання навчального матеріалу, малоаргументовані відповіді, слабке застосування теоретичних положень під час розв’язання практичних задач;
- **60–50 балів – оцінка E (“задовільно” – 3)** виставляється за слабкі знання навчального матеріалу, неточні або мало аргументовані відповіді, з порушенням послідовності викладення, за слабке застосування теоретичних положень під час розв’язання практичних задач;
- **49–26 балів – оцінка FX (“незадовільно” – 2)** виставляється за незнання значної частини навчального матеріалу, істотні помилки у відповідях на запитання, невміння застосувати теоретичні положення під час розв’язання практичних задач;
- **25–00 балів – оцінка F (“незадовільно” – 2)** виставляється за незнання значної частини навчального матеріалу, істотні помилки у відповідях на запитання, невміння орієнтуватися під час розв’язання практичних задач, незнання основних фундаментальних положень.

Комплект завдань складається з 30 білетів, кожний з яких містить 2 теоретичних питання і 1 практичне завдання. Кожне запитання і завдання має вагу 33 бали.

ВАРІАНТИ КОНТРОЛЬНИХ РОБІТ

Варіант №1

1. Чи можливо використовувати рядки у виразі switch?
   - ○ так
   - ○ ні
2. Чи можливе наслідування від рядка? Наприклад:
   ```
   class MyString extends String {}
   ```
   - ○ так
   - ○ ні
3. Чи допустимо викликати методи в рядкових літералів? Наприклад:
   ```
   "abc".charAt(0)
   "abc".length()
   ```
   - ○ так
   - ○ ні

Варіант №2

1. Скільки об'єктів буде створено в наступних рядках коду
   ```
   String s1=new String("abc");
   String s2="abc";
   ```
   - ○ один
   - ○ два
   - ○ жодного
2. Який метод краще використовувати для порівняння рядків?
   - ○ equals()
   - ○ hashCode()
   - ○ ==
3. Чи буде зкомпільовано наступний код?
   ```
   class A
   {
   }

   interface Demo extends A
   {
   }
   ```
   - ○ так
   - ○ ні, оскільки A не реалізує інший інтерфейс
   - ○ ні, оскільки інтерфейс не може розширяти клас

Варіант №3

1. Чи буде зкомпільовано наступний код? Оберіть найточнішу відповідь
   ```
   interface A
   {
     int variable=1;
   }
   ```

```
Class B implements A
{
  void methodB()
  {
    variable=2;
  }
}
```

- ○ так
- ○ ні, оскільки variable вже оголошено
- ○ ні, оскільки variable вже оголошено і це константа

2. Наступний код викликає помилку компіляції. Вкажіть причину

```
abstract class A
{
  abstract int demo();
}

class B extends A
{

}
```

- ○ клас B не містить конструктора
- ○ клас A не містить абстрактного конструктора
- ○ клас B не реалізував всі абстрактні методи батьківського класу

3. Клас було оголошено як абстрактний без жодного абстрактного методу. Чи буде він зкомпільований? Оберіть найточнішу відповідь

```
abstract class Demo {}
```

- ○ так
- ○ так, за умови що буде додано абстрактний конструктор
- ○ ні
- ○ тільки розробник може вирішувати чи буде щось зкомпільовано

Варіант №4

1. Вкажіть причину помилки компіляції для наступного коду. Виберіть найточнішу відповідь

```
interface A
{
  void test();
}

class B implements A
{
  void test()
```

```
  {
    System.out.println("Ok");
  }
}
```

- ○ метод test() в класі B повинен бути static
- ○ метод test() в класі B повинен бути final
- ○ метод test() в класі B повинен бути public
- ○ тут не буде помилки компіляції

2. Чи буде зкомпільовано наступний інтерфейс?

```
interface A
{
  private int i;
}
```

- ○ так
- ○ ні

3. Скільки конструкторів матиме наступний клас?

```
public class A
{

}
```

- ○ один
- ○ жодного
- ○ такий клас викличе помилку компіляції

Варіант №5

1. Чи буде зкомпільовано наступний клас?

```
class Test
{
  int test(int i, int d)
  {
    return 0;
  }

  static int test (int i, double d)
  {
    return 0;
  }

  static double test(double i, double d)
  {
    return 0;
  }
}
```

- ○ так
- ○ ні

2. Чи буде зкомпільовано наступний код?

```
class A

{

  void test()

  {

    class B

    {

      static void demo(){ }

    }

  }

}
```

- ○ так
- ○ ні

3. Яким чином отримати доступ до змінної i за межами класу OuterClass

```
class OuterClass

{

  static class InnerClass

  {

    int i;

  }

}
```

- ○ це неможливо
- ○ OuterClass.new InnerClass().i
- ○ OuterClass.InnerClass.i
- ○ new OuterClass.InnerClass().i

Варіант №6

1. Що з наведеного нижче неправда про анонімний внутрішній клас?
   - ○ ви можете створити безліч екземплярів такого класу
   - ○ такі класи не мають імені
   - ○ такий клас може мати лише один екземпляр
2. Чи буде зкомпільовано наступний фрагмент коду?

```
class A

{

  class B

  {

    static void test()

    {

      System.out.println("ok");

    }

  }

}
```

   - ○ так
   - ○ ні
3. Чи буде зкомпільовано наступний фрагмент коду?

```
class Question

{

  void Test()

  {

    enum Demo

    {

    }
```

```
  }

}
```

○ так
○ ні

Варіант №7

1. Що буде виведено на екран після виконання програми?

```
enum Demo

{

  DEMO;

  int i = 10;

  {

    i--;

  }

  {

    --i;

  }

  private Demo()

  {

    i = i + i;
  }

}

public class TestClass

{

  public static void main(String[] args)
```

```
  {

    System.out.println(Demo.DEMO.i);

  }

}
```

- ○ 14
- ○ 15
- ○ 10
- ○ помилка компіляції
- ○ помилка часу виконання
- ○ щось інше

2. Яким чином отримати доступ до константи WINTER за межами класу Demo?

```
public class Demo

{

  public enum Seasons

  {

    SPRING, WINTER;

  }

}
```

- ○ new Demo().Seasons.WINTER
- ○ Demo.Seasons.WINTER
- ○ Це неможливо

3. Що буде виведено на екран після виконання програми?

```
enum Seasons
{
  SPRING, WINTER;

  private Seasons()
   {

    System.out.println("ok");

  }
```

```
}

public class Test

{

  public static void main(String[] args)

  {

    Seasons[] seasons= new Seasons[2];

    for (int i = 0; i < seasons.length; i++)

    {

      System.out.print(seasons[i]);

    }

  }

}
```

- ○ nullnull
- ○ помилка компіляції
- ○ помилка часу виконання
- ○ щось інше

Варіант №8

1. Що буде виведено на екран після виконання програми?

```
enum Seasons

{

  SPRING, WINTER;

  private Seasons()

  {

    System.out.println("ok");

  }
```

```
}
public class Test

{

  public static void main(String[] args)

  {

    Seasons seasons = new Seasons();

  }

}
```

- ○ ok
- ○ помилка компіляції
- ○ помилка часу виконання
- ○ щось інше

2. Що буде виведено на екран після виконання програми?

```
enum D
{

  A, B, C;
  private D()
  {
    System.out.print("*");
  }
}
public class Demo
{
  public static void main(String[] args)
  {
    Enum enu = D.B;
  }
}
```

- ○ ***
- ○ нічого
- ○ буде помилка компіляції

3. Чи можливо використовувати локальні змінні без ініціалізації?
   - ○ так

○ ні
○ так, за умови що змінна final

Варіант №9

1. Чи буде працювати наступний фрагмент класу?

```
public class A

{

  A a = new A();

}
```

○ так
○ ні
○ буде помилка переповнення стеку під час виконання
○ буде помилка компіляції

2. Чи дозволено використовувати this() та super в методах?

```
void test()

{

  this();

  super();

}
```

○ так
○ ні

3. Що відбудеться якщо Ви будете повертати значення в конструкторі

```
class MyClass

{

  int MyClass()

  {

    return 1;

  }

}
```

○ помилка компіляції

- ○ буде працювати як звичайний конструктор
- ○ програма завершиться та поверне -1 як код помилки
- ○ це буде звичайний метод, а не конструктор

Варіант №10

1. Які значення приймуть a та b після виконання main?

```
class Test
{
  int a, b;
  Test()
  {
    a = 10;
    b = 20;
  }
}
class Runner
{
  public static void main(String[] args)
  {
    Test obj1 = new Test();
    Test obj2 = obj1;
    obj1.a += 1;
    obj1.b += 1;
    obj2.a += 1;
    obj2.b += 1;
```

```
  }

}
```
- ○ 11, 21
- ○ 12, 21
- ○ 10, 20
- ○ 2, 2

2. Яким чином створюються екземпляри об'єкту?
   - ○ використовується ключове слово create
   - ○ використовується ключове слово make
   - ○ використовується ключове слово new
   - ○ використовується ключове слово add
3. Що задає поведінку об'єкту?
   - ○ методи
   - ○ поля
   - ○ опис класу

Варіант №11

1. Клас повинен мати принаймні одне поле
   - ○ так
   - ○ ні
2. Що описує стан об'єкту?
   - ○ методи
   - ○ поля
   - ○ опис класу
3. Що з наведено нижче найкраще описує returnType та returnValue

```
returnType methodName()

{

  return returnValue;

}
```
- ○ returnValue повинно бути того ж типу що і returnType
- ○ returnValue повинно бути типу returnType або типом, що є нащадком returnType
- ○ значення returnValue може бути будь якого типу, Java приведе його до returnType
- ○ якщо returnType описано як void то тоді returnValue може бути чим завгодно

Варіант №12

1. Що виведе на екран наступний фрагмент коду?

```
class IntsTest {

int i;

}

class Main {

public static void main(String args[]) {

IntsTest test = new IntsTest();

System.out.println(test.i);

}

}
```

   ○ 0
   ○ null
   ○ помилка компіляції
   ○ помилка виконання
2. В чому відмінність break від continue?
   ○ break перериває цикл повністю, а continue лише одну ітерацію
   ○ continue перериває цикл повністю, а break лише одну ітерацію
   ○ вони ідентичні та взаємозамінні, відмінностей немає
   ○ у Java не використовується break, замість нього є інше слово
3. За допомогою якого оператора у Java можна обчислити залишок від ділення цілих чисел?
   ○ /
   ○ %
   ○ #
   ○ mod

Варіант №13

1. У змінній якого типу можна зберігати число 128?
   ○ byte, short, int, long
   ○ short, int, long
   ○ тільки int або long
2. Як можна назвати змінну?
   ○ goto
   ○ null

- ○ true
- ○ const
- ○ $const_goto
- ○ жодним із запропонованих варіантів

3. Які є базові характеристики кожної змінної
   - ○ ім’я, тип, модульність
   - ○ ім’я, тип, значення
   - ○ ім’я, тип, знак

Варіант №14

1. Які ключові слова з перелічених заборонено використувати (не реалізовано) у Java?
   - ☐ const
   - ☐ goto
   - ☐ abstract
   - ☐ break
2. Чи може ім'я змінної починатися з цифри?
   - ○ так, може
   - ○ ні, не може
   - ○ тільки якщо це нуль
3. Скільки у файлі вихідного коду може бути класів з модифікатором public?
   - ○ завжди хоча б один
   - ○ не більше двох
   - ○ не більше одного
   - ○ у файлі вихідного коду не має бути жодного такого класу, відповідь – нуль

Варіант №15

1. Коли дані не є глобальними, доступними всій програмі, а локальними доступними тільки малій частині програми?
   - ○ при інкапсуляції
   - ○ при поліморфізмі
   - ○ при оголошенні методу
2. Об’єктно-орієнтоване програмування характеризується
   - ☐ наявністю однієї лінійної програми
   - ☐ поділом програми на модулі
   - ☐ всі дані про об'єкт, його зв'язки з іншими об'єктами являють собою одну структурну одиницю
3. ООП дозволяє розширення функцій об'єктів або функцій класу. Це називається ...
   - ○ розширення
   - ○ збільшення місткості
   - ○ віртуальний внутрішній клас
   - ○ масштабованість
   - ○ наслідування

Варіант №16

1. Приховування даних також відомо як ...
   - ○ зв'язування
   - ○ інкапсуляція
   - ○ поліморфізм
2. Під час якого процесу об'єкт отримує властивості іншого об'єкту?
   - ○ інкапсуляція
   - ○ наслідування
   - ○ поліморфізм
3. Що з наведеного не є частиною парадигми ООП?
   - ☐ перевірка типів
   - ☐ багатозадачність
   - ☐ поліморфізм
   - ☐ приховування даних

Варіант №17

1. Поведінка об'єкту визначається через...
   - ○ поля
   - ○ методи
2. Об'єкти мають дві основні характеристики: стан та поведінка. Що з перерахованого описує стан?
   - ○ стан — це визначення самого об'єкту
   - ○ стан дозволяє розрізнити два об'єкти. Стан також відомий як поля
   - ○ стан визначає що може робити об'єкт
3. Що буде виведено на екран в результі виконання фрагменту коду?
   ```
   System.out.println(Integer.valueOf(1).equals(Long.valueOf(1L)));
   ```
   - ○ true
   - ○ false

Варіант №18

1. Що буде виведено на екран в результаті виконання фрагменту коду?
   ```
   Integer i1=100;
   Integer i2=100;
   System.out.println(i1==i2);
   ```
   - ○ true
   - ○ false
2. Які з тверджень про реалізацію hashCode() за замовчуванням правдиві?
   - ☐ клас Object не має реалізації hashCode() за замовчуванням
   - ☐ hashCode() за замовчуванням може повертати однакові значення для різних об'єктів
   - ☐ якщо два різних посилання вказують на один і той же об'єкт, то виклик hashCode() поверне одне й те ж саме значення
   - ☐ hashCode() за замовчуванням поверне випадкове число
   - ☐ серед наведених варіантів немає правильного

3. Який тип виключення не потрібно обробляти на етапі компіляції?
   - ○ Runtime
   - ○ Checked
   - ○ жоден з наведених

Варіант №19

1. Який блок завжди виконується, незалежно від того, було викинуто виключення чи ні?
   - ○ throws
   - ○ finally
   - ○ catch
   - ○ throw
2. Що буде виведено на екран в результаті виконання коду?

```
class Main{
  public static void main(String args[]){
    try {
      throw new String("A");
    }
    catch (int e){
       System.out.println("Exception "+e);
    }
  }
}
```

   - ○ Exception A
   - ○ Exception
   - ○ нічого, буде помилка компіляції
3. Що з наведеного нижче об'єднає рядки str1 та str2?
   - ○ str3 = str1 && str2
   - ○ str3 = str1.concat(str2)
   - ○ str3 = str1.append(str2)
   - ○ всі варіанти помилкові

Варіант №20

1. Що буде виведено на екран після виконання наступного фрагменту коду?

```
String str1="Hello";
String str2="World";
String str3=str1+str2;
String str4="YelloWorld";
System.out.println(str3==str4);
```

   - ○ true
   - ○ false
2. Що виведе наступний фрагмент коду?

```
class Main{
 public static void main(String args[])
```

```
 {
   char chars[]={'a', 'b', 'c'};
   String str = new String(chars);
   System.out.println(str);
 }
}
```

- ○ нічого, буде помилка компіляції
- ○ abc
- ○ a
- ○ c

3. Чи можливо використовувати рядки у виразі switch?
   - ○ так
   - ○ ні

Варіант №21

1. Чи можливе наслідування від рядка? Наприклад:
   ```
   class MyString extends String {}
   ```
   - ○ так
   - ○ ні
2. Які значення приймуть a та b після виконання main?
   ```
   class Test

   {

     int a, b;

     Test()

     {

       a = 10;

       b = 20;

     }

   }

   class Runner

   {

     public static void main(String[] args)
   ```

```
  {
    Test obj1 = new Test();
    Test obj2 = obj1;
    obj1.a += 1;
    obj1.b += 1;
    obj2.a += 1;
    obj2.b += 1;
  }
}
```

- ○ 11, 21
- ○ 12, 21
- ○ 10, 20
- ○ 2, 2

3. Як можна назвати змінну?
   - ○ goto
   - ○ null
   - ○ true
   - ○ const
   - ○ $const_goto
   - ○ жодним із запропонованих варіантів

Варіант №22

1. Чи допустимо викликати методи в рядкових літералів? Наприклад:
   `"abc".charAt(0)`
   `"abc".length()`
   - ○ так
   - ○ ні
2. Яким чином створюються екземпляри об'єкту?
   - ○ використовується ключове слово create
   - ○ використовується ключове слово make
   - ○ використовується ключове слово new
   - ○ використовується ключове слово add
3. Які є базові характеристики кожної змінної
   - ○ ім’я, тип, модульність

○ ім’я, тип, значення
○ ім’я, тип, знак

Варіант №23

1. Скільки об'єктів буде створено в наступних рядках коду
   ```
   String s1=new String("abc");
   String s2="abc";
   ```
   ○ один
   ○ два
   ○ жодного
2. Що задає поведінку об'єкту?
   ○ методи
   ○ поля
   ○ опис класу
3. Які ключові слова з перелічених заборонено використувати (не реалізовано) у Java?
   ☐ const
   ☐ goto
   ☐ abstract
   ☐ break

Варіант №24

1. Який метод краще використовувати для порівняння рядків?
   ○ equals()
   ○ hashCode()
   ○ ==
2. Клас повинен мати принаймні одне поле
   ○ так
   ○ ні
3. Чи може ім'я змінної починатися з цифри?
   ○ так, може
   ○ ні, не може
   ○ тільки якщо це нуль

Варіант №25

1. Чи буде зкомпільовано наступний код?
   ```
   class A
   {
   }

   interface Demo extends A
   {
   }
   ```
   ○ так

- ○ ні, оскільки A не реалізує інший інтерфейс
- ○ ні, оскільки інтерфейс не може розширяти клас

2. Що описує стан об'єкту?
   - ○ методи
   - ○ поля
   - ○ опис класу
3. Скільки у файлі вихідного коду може бути класів з модифікатором public?
   - ○ завжди хоча б один
   - ○ не більше двох
   - ○ не більше одного
   - ○ у файлі вихідного коду не має бути жодного такого класу, відповідь – нуль

Варіант №26

1. Чи буде зкомпільовано наступний код? Оберіть найточнішу відповідь

```
interface A
{
  int variable=1;
}

Class B implements A
{
  void methodB()
  {
    variable=2;
  }
}
```

   - ○ так
   - ○ ні, оскільки variable вже оголошено
   - ○ ні, оскільки variable вже оголошено і це константа

2. Що з наведено нижче найкраще описує returnType та returnValue

```
returnType methodName()

{

  return returnValue;

}
```

   - ○ returnValue повинно бути того ж типу що і returnType
   - ○ returnValue повинно бути типу returnType або типом, що є нащадком returnType
   - ○ значення returnValue може бути будь якого типу, Java приведе його до returnType

○ якщо returnType описано як void то тоді returnValue може бути чим завгодно

3. Коли дані не є глобальними, доступними всій програмі, а локальними доступними тільки малій частині програми?
   ○ при інкапсуляції
   ○ при поліморфізмі
   ○ при оголошенні методу

Варіант №27

1. Наступний код викликає помилку компіляції. Вкажіть причину

```
abstract class A
{
  abstract int demo();
}

class B extends A
{

}
```

   ○ клас B не містить конструктора
   ○ клас A не містить абстрактного конструктора
   ○ клас B не реалізував всі абстрактні методи батьківського класу

2. Що виведе на екран наступний фрагмент коду?

```
class IntsTest {

int i;

}

class Main {

public static void main(String args[]) {

IntsTest test = new IntsTest();

System.out.println(test.i);

}

}
```

   ○ 0
   ○ null

○ помилка компіляції
○ помилка виконання

3. Об’єктно-орієнтоване програмування характеризується
☐ наявністю однієї лінійної програми
☐ поділом програми на модулі
☐ всі дані про об'єкт, його зв'язки з іншими об'єктами являють собою одну структурну одиницю

Варіант №28

1. Клас було оголошено як абстрактний без жодного абстрактного методу. Чи буде він зкомпільований? Оберіть найточнішу відповідь
```
abstract class Demo {}
```
○ так
○ так, за умови що буде додано абстрактний конструктор
○ ні
○ тільки розробник може вирішувати чи буде щось зкомпільовано
2. В чому відмінність break від continue?
○ break перериває цикл повністю, а continue лише одну ітерацію
○ continue перериває цикл повністю, а break лише одну ітерацію
○ вони ідентичні та взаємозамінні, відмінностей немає
○ у Java не використовується break, замість нього є інше слово
3. ООП дозволяє розширення функцій об'єктів або функцій класу. Це називається ...
○ розширення
○ збільшення місткості
○ віртуальний внутрішній клас
○ масштабованість
○ наслідування

Варіант №29

1. Вкажіть причину помилки компіляції для наступного коду. Виберіть найточнішу відповідь
```
interface A
{
  void test();
}

class B implements A
{
  void test()
  {
    System.out.println("Ok");
  }
}
```

- ○ метод test() в класі B повинен бути static
- ○ метод test() в класі B повинен бути final
- ○ метод test() в класі B повинен бути public
- ○ тут не буде помилки компіляції

2. За допомогою якого оператора у Java можна обчислити залишок від ділення цілих чисел?
   - ○ /
   - ○ %
   - ○ #
   - ○ mod
3. Об'єкти мають дві основні характеристики: стан та поведінка. Що з перерахованого описує стан?
   - ○ стан — це визначення самого об'єкту
   - ○ стан дозволяє розрізнити два об'єкти. Стан також відомий як поля
   - ○ стан визначає що може робити об'єкт

Варіант №30

1. Чи буде зкомпільовано наступний інтерфейс?
```
interface A
{
  private int i;
}
```
   - ○ так
   - ○ ні
2. У змінній якого типу можна зберігати число 128?
   - ○ byte, short, int, long
   - ○ short, int, long
   - ○ тільки int або long
3. Що виведе наступний фрагмент коду?
```
class Main{
 public static void main(String args[])
 {
   char chars[]={'a', 'b', 'c'};
   String str = new String(chars);
   System.out.println(str);
 }
}
```
   - ○ нічого, буде помилка компіляції
   - ○ abc
   - ○ a
   - ○ c

ЗРАЗОК ВИКОНАННЯ ВАРІАНТУ КОНТРОЛЬНОГО ЗАВДАННЯ

Варіант 1.

Питання 1: Чи можливо використовувати рядки у виразі switch?
⦿ так
○ ні

Питання 2: Чи можливе наслідування від рядка? Наприклад:
`class MyString extends String {}`
○ так
⦿ ні

Завдання3: Чи допустимо викликати методи в рядкових літералів? Наприклад:
`"abc".charAt(0)`
`"abc".length()`
⦿ так
○ ні

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Олексій Васильєв – Програмування мовою Java – Навчальна книга – Богдан, 2020. – 696 с.; іл.
2. Роберт Мартін – Чистий код. Створення і рефакторинг за допомогою Agile – Фабула, 2019. – 448 с.
3. Кунгурцев О.Б. Основи програмування на мові Java. Середовище NetBeans. Навчальний посібник – Одеса: ВМВ, 2006. – 183с.
4. Арнольд К., Гослинг Д.– Язык программирования Java – СПб.: Питер, 2002.– 250 с.; ил.
5. Ноутон П., Шилдт Г. – Java2. Наиболее полное руководство СПб.: БХВ-Петербург, 2006.– 1034 с.; ил.
6. Дейтел Х.М., Дейтел П.Дж., Сантри С.И. – Технологии программирования на Java. Том 1. Графика. Москва.: Бином-пресс, 2003. – 560с.; ил.
7. Скотт К. Java для студента – СПб.: БХВ-Петербург, 2007.– 448 с.; ил.
8. Флэнаген Д. Java в примерах: Справочник.-2е издание. Спб.: Символ-Плюс, 2003. – 664 с.
9. Нотон П. JAVA: Справ.руководство: Пер.с англ. / Под ред.А.Тихонова. –М.: БИНОМ: Восточ. Кн. Компания, 1996: Восточ. Кн. Компания. – 447с. – (Club Computer)
10. Патрик Нотон, Герберт Шилдт Полный справочник по Java. – McGraw-Hill,1997, Издательство "Диалектика",1997
11. Дейтел Х.М., Дейтел П.Дж., Сантри С.И. – Технологии программирования на Java. Том 2. Распределенные приложения. Москва.: Бином-пресс, 2003. – 464с.; ил.
12. Дейтел Х.М., Дейтел П.Дж., Сантри С.И. – Технологии программирования на Java. Том 3. Корпоративные системы, сервлеты, JSP, Web-сервисы. Москва.: Бином-пресс, 2003. – 672с.; ил.
13. Руководство программиста Enterprise JavaBean v4.0. Inprise Corporation. 1999
14. Хабибулин И.Ш. Создание распределенных приложений на Java 2. – СПб.: БХВ-Петербург, 2002.– 701 с.; ил.
15. Дэвид Флэнэген Java in a Nutshell.- O'Reilly & Associates, Inc., 1997, Издательская группа BHV, Киев, 1998
16. Чен М.С. и др. Программирование на JAVA:1001 совет: Наиболее полное руководство по Java и Visual J++: Пер.с англ. / Чен М.С., Грифис С.В., Изи Э.Ф. – Минск: Попурри, 1997. – 640с.ил.+ Прил.(1диск).
17. Майкл Эферган Java: справочник. – QUE Corporation, 1997, Издательство "Питер Ком", 1998

ОПИС МЕТОДИЧНОГО ТА ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

Для успішного опанування дисципліни «Об'єктно-орієнтоване програмування» кожному здобувачеві освіти потрібен доступ до **персонального комп'ютера або лептопа з наступними характеристиками**:

- процесор: принаймні 2-ядерний Intel або AMD 1.7 Ггц або кращий
- дисковий простір: принаймні 120 Гц вільного місця для встановлення ПЗ
- RAM: принаймні 2 Гб (у випадку використання Linux), 4 Гб або більше (Windows)
- мережна карта з доступом в Інтернет
- відеокарта/монітор: принаймні 11,6" Wide XGA (WXGA) або краший
- Microsoft Mouse або сумісний маніпулятор
- звукова карта з гучномовцями та мікрофоном
- операційна система – будь-яка (Windows 10, Linux, MacOS)
- бажаною є наявність веб-камери

Вільне програмне забезпечення, використовуване під час виконання практичних робіт:

- Java JDK
- IDE NetBeans або IntelliJ IDEA (Community Edition або Ultimate, доступна в рамках партнерської угоди між коледжем та JetBrains)
- easyUML for Netbeans
- Google Chrome
- draw.io (diagrams.net) або StarUML
- Repl.It
- EasyCodeForAll
- Java Visualizer
- Lightshot
- Doxygen
- GitHub (GitHub Enterprise Cloud, доступний в рамках партнерської угоди між коледжем та GitHub Inc.)
- GitHub Desktop або GitKraken (Pro-версія доступна у складі Student Developer Pack в рамках партнерської угоди між коледжем та GitHub Inc.)

РЕКОМЕНДОВАНА ЛІТЕРАТУРА ТА ІНТЕРНЕТ-РЕСУРСИ

Книги та навчальні посібники:

1. Яків Файн. Програмування мовою Java для дітей, батьків, дідусів та бабусь (2015) – http://myflex.org/books/java4kids/JavaKid_ua.zip
2. Копитко М.Ф., Іванків К.С. Основи програмування мовою Java: Тексти лекцій. – Львів: Видавничий центр ЛНУ ім. Івана Франка, 2002. – 83 с.
3. Брнакевич І.Є., Вагін П.П. Програмування мовою Java: використання фундаментальних класів: Тексти лекцій. – Львів: Видавничий центр ЛНУ імені Івана Франка, 2002. – 75 с.
4. Вступ до алгоритмів Томас Г. Кормен, Чарлз Е. Лейзерсон, Роналд Л. Рівест і Кліфорд Стайн. Переклад з англійської (третього видання). К.І.С.,2019 - 1288 стор.
5. Кунгурцев О.Б. Основи програмування на мові Java. Середовище Net Beans. Навчальний посібник-Одеса: ВМВ, 2006. -183с.
6. Будай А. Дизайн-патерни - просто, як двері (2012) [PDF] - https://toloka.to/t34059

Мережеві ресурси:

1. Основи програмування на Java – безкоштовний відеокурс (Prometheus): https://college.page.link/A6vH
2. Java Professional – авторський відеокурс (Бабич О.В, ITVDN): https://college.page.link/Upp4
3. Andrei Dmitriev's Java SE Course: https://edu.netbeans.org/contrib/slides/dmitriev/
4. Освоюємо Java (вікіпідручник): https://college.page.link/GK8t