МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання контрольної роботи

та самостійному вивченню навчальної дисципліни

ОБ'ЄКТНО – ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

для студентів 3 курсу заочної форми навчання

напрямку підготовки – комп'ютерні науки

Узгоджено

Зав. навч.- конс. центру

_____ВОЛОШИНА О.В.

Затверджено на засіданні

кафедри інформаційних технологій

Протокол №____від _____ 2016 р.

Зав. кафедрою _____ ПРЕПЕЛИЦЯ Г.П.

ОДЕСА 2017

МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання контрольної роботи

та самостійному вивченню навчальної дисципліни

ОБ'ЄКТНО – ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

для студентів 3 курсу заочної форми навчання

напрямку підготовки – комп'ютерні науки

Узгоджено

У навчально –

консультаційному центрі

ОДЕСА 2017

МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання контрольної роботи

та самостійному вивченню навчальної дисципліни

ОБ'ЄКТНО – ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

для студентів 3 курсу заочної форми навчання

напрямку підготовки – комп'ютерні науки

ОДЕСА 2017

МЕТОДИЧНІ ВКАЗІВКИ до виконання контрольної роботита самостійному вивченню навчальної дисципліни ОБ’ЄКТНО – ОРІЄНТОВАНЕ ПРОГРАМУВАННЯдля студентів 3 курсу заочної форми навчаннянапрямку підготовки – комп’ютерні науки.

Укладач:

Рольщіков В. Б., старший викладач кафедри інформаційних технологій,

Штефан Н. З., асистент кафедри інформаційних технологій

ОДЕкУ, 2017, 82с., укр. мова.

ЗМІСТ

ПЕРЕДМОВА

Мета та завдання навчальної дисципліни

Вивчення студентами сучасних об’єктно-орієнтованих технологій програмування, та ознайомлення з відповідними мовами програмування, насамперед з програмуванням мовою JAVA, а також з основами C#. Одержання ними теоретичних знань та навичок з програмування, виготовлення, тестування, налагодження та запровадження програмних продуктів.

Предметом дисципліни є одержання студентами початкових навичок в створенні програм з застосуванням об’єктно-орієнтованої парадигми програмування шляхом слухання лекцій, виконання лабораторних робіт та завдань учбової практики. Велика практична значущість створення програмного забезпечення на основі об’єктного підходу до програмування визначається тим, що сучасні мови програмування у своїй більшості є цілком об’єктно-орієнтованими або у значний мірі підтримують парадигму об’єктного програмування. Такий підхід значною мірою зменшує вірогідність

Завдання дисципліни: навчити студентів створювати програмне забезпечення на основі об’єктного підходу до програмування ЕОМ; надання студентам навичок у налагодженні об’єктних програм; ознайомлення студентів із структурою та властивостями об’єктів.

За результатами вивчення навчальної дисципліни студент повинен**знати:** принципи побудови класів та їхніх об’єктів, їх властивості (абстракцію, інкапсуляцію, спадковість та поліморфізм), елементи основних мов об’єктного програмування, насамперед JAVA.

вміти: засобами мови програмування Java створювати власні класи, створювати екземпляри об’єктив від цих класів, передавати повідомлення об’єктам та відповідним чином провадити їх обробку, вміти обробляти виключення, що виникають в програмі.

Дисципліна базується на: алгоритмічні мові та основи програмування, об’єктно-орієнтоване програмування (змістовий модуль №1 – об’єктно-орієнтоване моделювання).

Дисципліна забезпечує вивчення: операційні системи, технології розподілених систем та паралельних обчислень.

Компетенції щодо навчальної дисципліни:

- **знання** основ парадигми об’єктно-орієнтованого програмування, засобів керування властивостями об’єктів, сферу застосовності програм, які написані за допомогою цієї парадигми;
- **вміння** створювати і програмно моделювати складні системи різноманітних об’єктів навколишнього середовища з існуючими проміж них зв’язками і взаємодіями, які найбільш повно відображають фізичні та суспільні системи реального світу.

1 ЗАГАЛЬНА ЧАСТИНА

Зміст дисципліни

Програма лекційного модуля

Розділ 1 – Технологія ООП (основні поняття)

1 Основи обєктно – орієнтованої мови програмування

2 Абстрагування даних та інкапсуляція

3 Конструктори і методи класів

4 Перевантаження конструкторів та методів

5 Статичні та константні члени класів

Розділ 2 – Технологія ООП (поліморфізм)

6 Просте успадкування класів

7 Композиція обєктів супроти успадкування

8 Реалізація поліморфізму

9 Інтерфейси та множине успадкування

Розділ 3 – Оброблення виключень

9 Вкладені та внутрішні класи

10 Оброблення виняткових ситуацій

11 Параметризовані класи та методи

Знання, якими повинні оволодіти студенти

- основні поняття мови об'єктно-орієнтованого програмування Java, як виконується інкапсуляція даних, принципи створення конструкторів і методів класів та їх перевантаження;
- як виконується просте успадкування класів, у яких випадках необхідно робити успадкування, а у яких створювати композицію об'єктів, як реалізується поліморфізм і виконується множинне успадкування у мові Java;
- навіщо потрібні вкладені та внутрішні класи, як вони функціонують,

як проводиться оброблення виняткових ситуацій, як програмувати та застосовувати параметризовані класи та методи.

Програма практичного модуля

Теми лабораторних робіт:

1. Класи та обєкти. Способи ініцювання полів.
2. Застосування конструкторів ініцювання та копіювання обєктів, перевантаження методів.
3. Спадкування членів класу. Перевизначення методів.
4. Абстрактні класи і їх реалізація.
5. Опис та реалізація інтерфейсів.
6. Обробка виключень, що виникають під час виконання програми.
7. Внутрішні класи та їх реалізація.

Після виконання лабораторних робіт студент повинен:

- вміти описувати класи, що містять статичні й нестатичні компоненти, задавати значення полів за допомогою блоків ініціалізації, створювати об’єкти і масиви об’єктів, розробляти різні типи конструкторів, будувати класи, що використовують перевантажені методи;
- вмітибудувати ієрархію спадкування, звертатися до методів суперкласу й перевизначати їхні методи, застосовувати прийоми побудови ієрархії з використанням абстрактних класів, описувати конкретні реалізації таких класів та методів, працювати з масивами типу абстрактного класу, описувати та застосовувати інтерфейси, створювати класи, що реалізують інтерфейс і працювати з об’єктами типу інтерфейс;
- вміти розробляти механізми обробки виключень в JAVA і будувати структуру класу з урахуванням можливостей перехоплення й обробки виключень, а також збуджувати їх штучно для

повідомлення зовнішньому коду про виникнення позаштатної ситуації, здійснювати побудову класів, що містять у своєму складі внутрішні класи.

Методичне забезпечення для виконання всіх змістовних модулів включає усебе методичні вказівки «Застосування засобів середовища розробки Java при виконанні лабораторних робіт з об'єктно-орієнтованого програмування» та «Методичні вказівки до виконання лабораторних робіт з ООП» (частини перша, друга, третя).

Всі лабораторні роботи виконуються **в комп'ютерних класах** кафедри інформаційних технологій із застосуванням **локальної мережі робочих станцій** на базі ОС Windows і середовища програмування Eclips.

2 Література

Базова

1. Императивное программирование объектно-ориентированное моделирование: Java, UML, OCL: учебное пособие для студентов высших учебных заведений / А.Ф.Верлань, И.А.Чмырь, С.Д.Кузниченко, Л.Б.Коваленко. – Одесса: Экология, 2013. – 432 с.

2. Шилдт Г. Java 8. Полное руководство; 9-е изд.: Пер. с англ. – М. : ООО "И.Д. Вильямс", 2015. – 1376 с.: ил.

Допоміжна

1. Specification: JSR-337 Java® SE 8 Release Contents ("Specification"), Version: 8, Final Release: March 2014 – Redwood City: Oracle America Inc., 2014. – 760 p.

2. Эккель Б. Философия Java. Библиотека программиста. 4-е изд. Спб.: Питер, 2009. – 640 с.: ил.

3. Вайсфельд М. Объектно-ориентированное мышление. – СПб.: Питер, 2014. – 304 с.: ил.

4. Java. Методы программирования: уч.-мет. пособие / И.Н.Блинов, В.С.Романчик. – Минск: издательство «Четыре четверти», 2013. – 896 с.

Методичне забезпечення

1. Рольщиков В.Б. Застосування засобів середовища розробки Java при виконанні лабораторних робіт з об’єктно-орієнтованого програмування // методичні вказівки – Одеса: ОДЕКУ, 2011 – 125 с.

2. Рольщиков В.Б., Шуптар Н.Й. Методичні вказівки до виконання лабораторних робіт для студентів II курсу денної форми навчання з дисципліни “Об’єктно-оріентоване програмування” // частина 1 – Одеса: ОДЕКУ, 2013 – 40 с.

3. Рольщиков В.Б., Шуптар Н.Й. Методичні вказівки до виконання лабораторних робіт для студентів II курсу денної форми навчання з дисципліни “Об’єктно-оріентоване програмування” // частина 2 – Одеса: ОДЕКУ, 2014 – 36 с.

4. Рольщиков В.Б., Шуптар Н.Й. Методичні вказівки до виконання лабораторних робіт для студентів II курсу денної форми навчання з дисципліни “Об’єктно-оріентоване програмування” // частина 3 – Одеса: ОДЕКУ, 2015 – 38 с.

Інформаційні ресурси

http://java.com/ru/

http://www.eclipse.org/

http://mymanual.narod.ru/

http://www.cyberforum.ru/java/

http://progopedia.ru/language/java/

http://www.helloworld.ru/show.php?curraz=34

http://developers.sun.ru/

http://bestjava.3dn.ru/publ/1-1-0-1

http://www.excode.ru/art259p1.html

http://java.sun.com/javame/index.jsp

http://developers.sun.ru/forum/viewforum.php?f=13

http://www.javenue.info/themes/java-gui

http://onedeveloper.ru/search?w=Java%20EE

3 Індивідуальні завдання

Для заочної форми навчання робочою навчальною програмою дисципліни передбачається виконання міжсесійної контрольної роботи.

Тематика міжсесійної контрольної роботи присвячена питанням створення об’єктно-орієнтованих них програм з застосуванням парадигми об’єктного програмування мовою Java. Завдання на контрольну роботу передбачають розробку деякої системи починаючи зі створення відповідного інтерфейсу, створення низки класів, які різними способами реалізують цей інтерфейс з урахуванням обробки можливих виняткових ситуацій і створення тестового класу для перевірки роботи системи.

Контроль за виконанням КР виконується поетапно шляхом відправлення кожного виконаного завдання на електрону адресу кафедри інформаційних технологій (***kaf-infotech@odeku.edu.ua***) не пізніше вказаного сроку (див. табл.). Титульний аркуш для КР наведен у додатку.

Таблиця 3.1 – Види самостійної роботи та форми її контролю

Розділ	Теми	Термін представлення частини контрольної роботи на перевірку	Бали
1. Технологія ООП (Основні поняття)	1.1 Основи обєктно – орієнтованої мови програмування 1.2 Абстрагування даних та інкапсуляція 1.3 Конструктори і методи класів 1.4 Перевантаження конструкторів та методів	листопад	25

	1.5 Статичні та константні члени класів *Виконання 1 завдання КР*		
2. Технологія ООП (поліморфізм)	2.1 Просте успадкування класів 2.2 Композиція обєктів супроти успадкування 2.3 Реалізація поліморфізму 2.4 Інтерфейси та множине успадкування *Виконання 2 завдання КР*	лютий	45
3. Оброблення виключень	3.1 Вкладені та внутрішні класи 3.2 Оброблення виняткових ситуацій 3.3 Параметризовані класи та методи *Виконання 3 завдання КР*	квітень	30
			Всього 100

4 Методи навчання і контролю

Навчання ведеться за допомогою читання лекцій, проведення лабораторних робот та за рахунок самостійної роботи студентів.

Організація поточного, семестрового та підсумкового контролю знаньпроводиться згідно з «Положенням про організацію поточного та підсумкового контролю знань студентів заочної форми навчання ОДЕКУ ».

Поточний контроль знань та вмінь студентів виконується шляхом усного опитування на кожній лабораторній роботі, оцінювання якості і терміну виконання лабораторних робіт, проведення тестового контролю знань з лекційного модуля. Приблизний перелік запитань до тестового контролю наведений нижче:

1. Загальні положення об’єктно-орієнтованої парадигми програмування
2. Основні поняття ООП
3. Збереження даних у пам’яті
4. Поняття і властивості об’єкта, оголошення простого класу у мові JAVA
5. Модифікатори оголошення класу
6. Поля класу їх модифікатори та конструкція ініціювання
7. Статичні поля та поля **final**
8. Управління доступом до членів класу, модифікатори доступу
9. Створення об’єктів
10. Блоки ініціювання та ініціалізація статичних полів
11. Конструктори їх відміна від методів класу
12. Методи класу, їх оголошення та модифікатори
13. Виклик методів, передача та повернення параметрів
14. Застосування методів для керування доступом
15. Вираз **this** і його застосування
16. Перевантаження методів, метод **main** та методи **native**
17. Спадкування, супер та субкласи
18. Конструктори субкласів
19. Порядок ініціювання полів субкласів
20. Перевизначення методів класів

21. Приховування полів та доступ до успадкованих членів
22. Службове слово **super** і його застосування
23. Сумісність, перетворення та перевірка типів
24. Особливості застосування модифікаторів **protected** та **final**
25. Модифікатор методів **abstract** і абстрактні класи
26. Клас Object і його методи
27. Клонування об’єктів
28. Тип даних, що перелічуються (**enum**)
29. Наслідування або композиція
30. Принципи проектування класу, що підлягає спадкуванню
31. Поняття інтерфейсу, стандартні інтерфейси мови JAVA
32. Оголошення інтерфейсу, константи та методи у ньому
33. Розширення інтерфейсів, спадкування та приховування констант
34. Спадкування, перевизначення і перевантаження методів у інтерфейсах
35. Робота з інтерфейсами та їх реалізація
36. Пусті інтерфейси та правила застосування інтерфейсів
37. Виключення та їх види
38. Створення типів виключень
39. Інструкція **throw**, передача управління
40. Висловлювання **throws** і перевизначення методів
41. Блок **try**
42. Висловлювання **finally**
43. Правила застосування виключень
44. Статичні вкладені класи і інтерфейси
45. Внутрішні класи
46. Доступ до полів внутрішніх та зовнішніх класів
47. Спадкування внутрішніх класів
48. Спадкування і приховування у контексті зовнішнього класу
49. Локальні внутрішні класи

50. Анонімні локальні класи
51. Вкладеність у інтерфейсах та змінні в них

На протязі заліково - єкзаменаційноїсесії виконується постійний обов'язковий контроль відвідування лекційних та лабораторних занять, оцінювання відповідей на запитання під час проведення лекцій. Наприкінці семестру проведення іспиту.

6 Методика оцінювання студентів

Поточна та підсумкова оцінка рівня знань студентів здійснюється за модульною системою. Розподіл балів за змістовими модулями з теоретичної і практичної частини курсу, яку може набрати студент, наведений у табл. 6.1.

При проведенні міжсесійного контролю студент вважається атестованим, якщо він набрав не менше 50% від максимально можливої суми балів за завдання, що завершені у атестаційний період.

Таблиця 6.1 – Розподіл балів, які отримують студенти

<table>
<tr><td colspan="2">Поточне тестування та самостійна робота</td><td>Підсум-ковий тест (іспит)</td><td>Сума</td></tr>
<tr><td>Індивідуальне завдання</td><td>Лабораторні роботи</td><td rowspan="2">100</td><td rowspan="2">300</td></tr>
<tr><td>100</td><td>100</td></tr>
</table>

Студент **вважається таким що не виконав навчального плану** з дисципліни «Об'єктно-орієнтоване програмування» доки він має перелічені нижче заборгованості і не ліквідував їх:

- набрано менше 50% балів від суми балів за виконання лабораторних робіт;

- набрано менше 50% балів від суми балів за виконання індивідуального завдання.

Для студентів, які виконали навчальний план, формується інтегральна сума балів – сума балів одержаних з теоретичної та практичної частин курсу.

Студенти виконують тестову іспитову контрольну роботи з курсу.

Екзаменаційний білет містить 20 тестових запитань, які мають різні ваги приблизно в такому співвідношенні: 12 запитань з вагою 5; 6 запитань з вагою 4 і 2 запитання з вагою 8. Тобто вірні відповіді на всі запитання з першої групи забезпечують студентові 60% від повної суми балів, вірні відповіді на другу групу додають ще 24% і останні 2 питання доповнюють відповідь до 100 відсотків.

Іспит проводиться на комп'ютерном обладнанні кафедри ІТ.

Інтегральна оцінка з дисципліни виставляється згідно «Положення про проведення підсумкового контролю знань студентів» затвердженого Методичною радою університету.

Для студентів заочної форми навчання загальна кількісна**інтегральна оцінка** з дисципліни згідно до «Положення про організацію поточного та підсумкового контролю знань студентів заочної форми навчання ОДЕКУ», накопичена підсумкова оцінка засвоєння студентом навчальної дисципліни розраховується за формулою:

ПО = 0,5ОПК + 0,25(ОЗЕ + ОМ),

де:

ОПК – кількісна оцінка (у відсотках від максимально можливої – 100%) заходу підсумкового контролю;

ОЗЕ – кількісна оцінка (у відсотках від максимально можливої - 100%) заходів контролю СРС під час проведення аудиторних занять;

ОМ – кількісна оцінка (у відсотках від максимально можливої) за межсесійну контрольну роботу.

При підсумковій атестації використовується шкала відповідності сумарної суми балів (%) підсумкової атестації з дисципліни у формі іспиту (табл.7.2).

7 Методичне забезпечення

1. Рольщиков В.Б. Застосування засобів середовища розробки Java при виконанні лабораторних робіт з об’єктно-орієнтованого програмування // методичні вказівки – Одеса: ОДЕКУ, 2011 – 125 с.

2. Рольщиков В.Б., Шуптар Н.Й. Методичні вказівки до виконання лабораторних робіт для студентів II курсу денної форми навчання з дисципліни “Об’єктно-оріентоване програмування” // частина 1 – Одеса: ОДЕКУ, 2013 – 40 с.

3. Рольщиков В.Б., Шуптар Н.Й. Методичні вказівки до виконання лабораторних робіт для студентів II курсу денної форми навчання з дисципліни “Об’єктно-оріентоване програмування” // частина 2 – Одеса: ОДЕКУ, 2014 – 36 с.

4. Рольщиков В.Б., Шуптар Н.Й. Методичні вказівки до виконання лабораторних робіт длястудентів II курсу денної форми навчання з дисципліни “Об’єктно-оріентоване програмування” // частина 3 – Одеса: ОДЕКУ, 2015 – 38 с.

Таблиця 7.2 – Шкала оцінювання: національна та ECTS

Сума балів за всі види навчальної діяльності	ОцінкаECTS	Оцінка за національною шкалою
		для екзамену, курсового проекту (роботи), практики
1	2	3
90 – 100	**A**	відмінно
82 – 89,9	**B**	добре
74 – 81,9	**C**	
64 – 73,9	**D**	задовільно
60 – 63,9	**E**	
35 – 59,9	**FX**	незадовільно з можливістю повторного складання
0 – 34,9	**F**	незадовільно з обов’язковим повторним вивченням дисципліни

2 ОРГАНІЗАЦІЯ САМОСТІЙНОЇ РОБОТИ

2.1 Вказівки до виконання завдань з прикладами

Методичні вказівки до виконання контрольної роботи (завдання 1)

Література до виконання завдання: [1] – стор. 184 – 196, [3] – стор. 232 – 283, [4] – стор. 353 – 382, а також [2]. Крім того, нижче наведені деякі відомості з принципів побудови класів, описання їх властивостей та методів, а також робота з конструкторами.

Метоюданої частиниєствоpення класів за умовами варіанту завдання.

1. Створення класу: властивості та методи.

Розглянемо приклад створення найпростішого класу. Давайте з його допомогою змоделюємо окружності на координатної площині. Кожна така окружність, буде визначатися своїм центром (тобто крапкою з двома числовими координатами) і радіусом (тобто його довжиною, що подається у вигляді числа).Таким чином, коло на координатної площині характеризують 3 дійсні числа. Значить в нашому класі повинно бути три відповідних властивості.Поки наділимо клас наступними можливостями: створену на основі класу окружність має бути можна зробити висновок на екран (у вигляді опису її характеристик), переміщати (тобто здійснювати перетворення руху, змінюючи координати її центру) і масштабувати (тобто здійснювати перетворення подібності , змінюючи радіус кола).

```
// описуємо окремий новий клас
classCircle{
  // властивості класу
  publicdoublex;// абсцисса центру
  publicdoubley;// ордіната центру
  publicdoubler;// радіус
  // властивості класу
  // вивод параметрів кола
  publicvoidprintCircle(){
    System.out.println("Окружность с центром ("+x+";"+y+") и
радиусом "+r);
  }
  // переміщує центр, рух окружності
  publicvoidmoveCircle(doublea,doubleb){
    x=x+a;
    y=y+b;
  }
  // масштабується, виконуємо перетворення подібності з
коефіцієнтом k
  publicvoidzoomCircle(doublek){
    r=r*k;
  }
}
// описуємо основний клас, що містить метод main
publicclassMain{
  publicstaticvoidmain(String[]args){
    // Створюємо об'єкт (окружність класу Circle), у неї буде
нульовою
        // Радіус і центр в (0.0; 0.0), оскільки всі
властивості отримають
        // Значення за замовчуванням
    Circleo1 =newCircle();
    // виводимо на екран параметри кола
    o1.printCircle();
    // Міняємо абсциссу центру, звертался до властивості x
```

```
    o1.x=3;
    // Міняємо радіус, звертається до властивості r
    o1.r=12.3;
    // виводимо на екран оновлені параметри кола
    o1.printCircle();
    // Створюємо інший об'єкт того ж класу
    Circleo2 =newCircle();
    o2.r=3.14;
    o2.zoomCircle(1.66);
    o2.printCircle();// Окружность с центром (0.0;0.0) и
радиусом 5.2124
  }
}
```

1 Конструктори

Коли ми створюємо об'єкт командою Circle o1 = new Circle (); використовується так званий конструктор за замовчуванням (або конструктор безпараметрів) - це спеціальний метод класу, ми його не визначали явно, але навіть якщо його не визначити він створюється автоматично, виконується при створенні кожного нового об'єкта і привласнює початкові значення його властивостям (инициализирует їх). Значення за замовчуванням для властивостей залежать від їх типу (0 або 0.0 для чисельних типів, false для логічного типу і т.д.).

Конструктор за замовчуванням можна описати явно і при цьому поставити начальне значення для властивостей нового об'єкта, відмінні від значень за замовчуванням.

Конструктори можуть мати будь-які модифікатори доступу: public, protected, private, або модифікатор може бути відсутнім (часто мається на увазі package або friendly), крім цього, конструктор не може бути abstract, final, native, static, orsynchronized.

Повертаються типи теж відрізняються. Методи можуть повертати будь-який правильний тип, або нічого не повертати, в даному випадку

повертається тип описується як void. Конструктори ж не мають повертається типу, вони не можуть повертати навіть тип void.

І на закінчення, методи і конструктори мають різні імена. Конструктори мають однакові імена з ім'ям класу в якому описані, а методи, за домовленістю, мають імена відмінні від імені класу. Якщо Java програма написана за правилами мови, імена методів починаються з маленької літери, конструкторів - з великою. І ще, ім'я конструктора є іменником, так як імена класів зазвичай є іменниками, імена ж методів зазвичай є дієсловами.

```
classCircle{
    publicdouble x;// абсцисса центра
    publicdouble y;// ордината центра
    publicdouble r;// радиус

    publicvoid printCircle(){
        System.out.println("Окружность с центром ("+x+";"+y+") и
радиусом "+r);
    }
    publicvoid moveCircle(double a,double b){
        x = x + a;
        y = y + b;
    }
    publicvoid zoomCircle(double k){
        r = r * k;
    }
   // Конструктор за замовчуванням, тепер відразу після
створення об'єкта будемо
     // Отримувати окружність одиничного радіуса з центром в
початку координат
    publicCircle(){
        x =0.0;
        y =0.0;
        r =1.0;
    }
```

```
}

publicclassMain{
    publicstaticvoid main(String[] args){
        Circle o1 =newCircle();
        o1.printCircle();// Окружность с центром (0.0;0.0) и
радиусом 1.0
    }
}
```

За допомогою перевантаження можна створити додаткові варіанти конструкторів. Наприклад, зручно мати конструктор, який дозволить при створенні об'єкта явно вказувати координати його центру і довжину радіуса.

Описати подібний конструктор можна на додаток до основного наступним чином

```
    publicCircle(double a,double b,double s){
        x = a;
        y = b;
        r = s;
    }
```

Теперприствореннїоб'єктівможнакористуватисябудь-якимконструкторомнавибір:

```
     Circleo1 =newCircle();
    o1.printCircle();// Окружность с центром (0.0;0.0) и
радиусом 1.0
    Circleo2 =newCircle(1,-1,14);
    o2.printCircle();// Окружность с центром (1.0;-1.0) и
радиусом 14.0
```

Потрібно враховувати наступний факт: якщо в класі описаний явно хоча б один конструктор з параметрами, то конструктор за замовчуванням (без параметрів) створюватися автоматичні вже не буде (його в такій ситуації

треба описувати явно). Хоча, якщо вам потрібно тільки конструктор з параметрами (як другий з нашого прикладу), то можна обійтися і зовсім без конструктора за замовчуванням (описати в класі тільки один конструктор з параметрами).

3 Доступ до членів класу з тіла методів

Додамо в наш клас метод, що обчислює площу тієї окружності, до якої метод застосований. Методбудеописуватитак:

```
publicdouble squareCircle(){
    double s =Math.PI * r * r;
    return s;
}
```

Результатроботиметодуможнапобачитивтакийспосіб

```
    System.out.println("Площадь круга o2:
"+o2.squareCircle());//615.75...
```

Зверніть увагу: всередині кожного методу класу доступні властивості того об'єкта, для якого метод буде викликатися. Тобто якщо ми викликаємо метод для об'єкта o2, то всередині методу при його виконанні ми будемо працювати саме з властивостями об'єкта o2 (o2.x буде доступно x, o2.r буде доступно як r і т.д.).

Може виникнути ситуація, коли для формальних параметрів методу ви захочете використовувати імена, які вже належать до властивостей класу.

Наприклад, можна було б почати опис методу для масштабування таким чином:

```
publicvoid zoomCircle(double r){…
```

Як же в такому випадку звертатися до властивостей об'єкта (адже імена цих властивостей перекриваються формальним параметром)?

Рішення такої неоднозначності існує: до будь-якої властивості всередині методу можна звертатися не тільки по імені, але і через посилання this. Тобто всередині методу можна написати x = 13;, а можна this.x = 13; - Ефект буде ідентичний. Відповідно, коли ім'я формального параметра перекриває ім'я властивості, до імені властивості потрібно звертатися через посилання this. Тоді метод можна переписати таким чином:

```
publicvoidzoomCircle(doubler){
  this.r=this.r*r;
}
```

Зрозуміло, що зручніше не допускати перекривання імен властивостей іменами локальних параметрів в методах. Іноді, втім, потрібно всередині методу застосувати якийсь інший метод до поточного об'єкту, тоді без посилання this не обійтися.

Додамо в клас метод, який перевіряє, чи збігаються дві окружності по площі.

У цьому методі повинні брати участь два об'єкти: той, для якого метод викликаний і другий учасник порівняння, який може бути переданий в метод через параметр. При цьому параметр буде мати відповідний тип (не якийсь вбудований, а у вигляді класу Circle).

Методможнаописатитак:

```
publicboolean equalsCircle(Circle cir){
    if(this.squareCircle()== cir.squareCircle()){
        returntrue;
    }else{
        returnfalse;
    }
}
```

Прикладвикористанняметоду:

```
        if(o1.equalsCircle(o2)){
            System.out.println("Круги o2 и o1
имеютравнуюплощадь");
        }else{
            System.out.println("Круги o2 и o1
имеютразличнуюплощадь");
        }
```

4 Завдання

1. Створіть в класі *Circle* метод, що обчислює довжину кола.

2. Створіть у класі *Circle* метод, що переміщає центр кола в випадкову точку квадрата координатної площини з діагоналлю від [-99; -99] до [99; 99]. Зверніть увагу на те, що потрібно створити звичайний метод, який можна застосовувати до вже існуючого об'єкта, а не конструктор створює новий об'єкт.

3. Змініть в класі *Circle* конструктор за замовчуванням так, щоб в момент створення об'єкту з його допомогою, координати центру і радіус кола користувач вводив з клавіатури.

4. Створіть у класі *Circle* метод, що обчислює відстань між центрами двох кіл.

5. Створіть в класі *Circle* метод, який перевіряє, чи стосуються окружності в одній точці. Врахуйте, що можливий варіант, коли одна окружність міститься всередині іншої і при цьому все одно можливо дотик в одній точці.

5 Приклад виконання завдання

```
classCircle{
    publicdouble x;// абсцисса центру
    publicdouble y;// ордината центру
    publicdouble r;// радіус

    publicvoid printCircle(){
        System.out.println("Окружность с центром ("+x+";"+y+") и
радиусом "+r);
    }
    publicvoid moveCircle(double a,double b){
        x = x + a;
        y = y + b;
    }
    publicvoid zoomCircle(double r){
        this.r =this.r * r;
    }
    publicCircle(){
        x =0.0;
        y =0.0;
        r =1.0;
    }
    publicCircle(double a,double b,double s){
        x = a;
        y = b;
        r = s;
    }
// Метод обчислює площу круга
    publicdouble squareCircle(){
        double s =Math.PI * r * r;
        return s;
    }
    // метод перевірки на рівтістьпо площині
```

```
    publicboolean equalsCircle(Circle cir){
        if(this.squareCircle()== cir.squareCircle()){
            returntrue;
        }else{
            returnfalse;
        }
    }
}

publicclassMain{
    publicstaticvoid main(String[] args){
        Circle o1 =newCircle();
        o1.printCircle();// Окружность с центром (0.0;0.0) и
радиусом 1.0
        Circle o2 =newCircle(1,-1,14);
        o2.printCircle();// Окружность с центром (1.0;-1.0) и
радиусом 14.0
        System.out.println("Площадь круга o2:
"+o2.squareCircle());//615.75...
        o1.zoomCircle(14);
        if(o1.equalsCircle(o2)){
            System.out.println("Круги o2 и o1 имеют равную
площадь");
        }else{
            System.out.println("Круги o2 и o1 имеют различную
площадь");
        }
    }
}
```

Полями створюваного класу можуть бути не тільки змінні вбудованих типів, але і об'єкти будь-яких інших доступних класів.При цьому задіявши методи існуючих класів, можна значно спростити створення нового класу.

Розглянемо приклад, в якому клас кіл створюється з використанням класу точок (одним з полів класу кіл є об'єкт-точка):

```
importjava.util.Scanner;

classPoint{
  publicdoublex;// абсцисса точки
  publicdoubley;// ордината точки

  // возвращает строку с описанием точки
  publicStringtoString(){
    return"("+x+";"+y+")";
   }
   // выводит на экран описание точки
  publicvoidprint(){
    System.out.print(this.toString());
  }
  // метод перемещает точку на указанный вектор
  publicvoidmove(doublea,doubleb){
    x=x+a;
    y=y+b;
  }
  // метод изменяет координаты точки на указанные
  publicvoidset(doublea,doubleb){
    x=a;
    y=b;
  }
  // конструктор по умолчанию, создающий точку с указанными
пользователем координатами
  publicPoint(){
    booleanerr;
    do{
      err=false;
      System.out.print("Введите абсциссу точки: ");
```

```
      Scannerscan=newScanner(System.in);
      if(scan.hasNextDouble()){
        x=scan.nextDouble();
      }else{
        System.out.println("Вы ввели не число, попробуйте
снова");
        err=true;
      }
    }while(err);
    do{
      err=false;
      Scannerscan=newScanner(System.in);
      System.out.print("Введите ординату точки: ");
      if(scan.hasNextDouble()){
        y=scan.nextDouble();
      }else{
        System.out.println("Вы ввели не число, попробуйте
снова");
        err=true;
      }
    }while(err);
  }
  // конструктор, создающий точку с указанными координатами
  publicPoint(doublea,doubleb){
    x=a;
    y=b;
  }
  // метод вычисляющий расстояние между точками
  publicdoublelength(Pointp){
    returnMath.sqrt(Math.pow(p.x-x,2)+Math.pow(p.y-y,2));
  }
  // метод проверяющий совпадают ли точки
  publicbooleanequalsPoint(Pointp){
    if(this.x==p.x&&this.y==p.y){
      returntrue;
```

```
    }else{
      returnfalse;
    }
  }
}

classCircle{
  publicdoubler;// радиус
  publicPointc;// центр

  // возвращает строку с описанием окружности
  publicStringtoString(){
    return"Окружность с центром в точке "+c+" и радиусом "+r;
  }
  // выводит на экран описание окружности
  publicvoidprint(){
    System.out.print(this.toString());
  }
  // метод перемещает центр окружности на указанный вектор
  publicvoidmove(doublea,doubleb){
    c.move(a,b);
  }
  // метод изменяет окружность, перемещая центр в указанные
координаты и меняя радиус
  publicvoidset(doublea,doubleb,doublem){
    c.set(a,b);
    r=m;
  }
   // метод изменяет окружность, перемещая центр в указанную
точку и меняя радиус
  publicvoidset(Pointp,doublem){
    c.set(p.x,p.y);
    r=m;
  }
  // конструктор по умолчанию, создающий окружность с указанными
```

```
пользователем параметрами
  Circle(){
    System.out.println("Задайте центр окружности:");
    c=newPoint();
    booleanerr;
    do{
      err=false;
       Scannerscan=newScanner(System.in);
      System.out.print("Задайте радиус: ");
      if(scan.hasNextDouble()){
        r=scan.nextDouble();
        if(r<=0){
          System.out.println("Радиус окружности должен быть
положительным");
          err=true;
        }
      }else{
        System.out.println("Вы ввели не число, попробуйте
снова");
        err=true;
      }
    }while(err);
  }
  Circle(doublea,doubleb,doublem){
    c.set(a,b);
    r=m;
  }
  // метод вычисляющий длину окружности
  publicdoublelength(Pointp){
    return2*Math.PI*r;
  }
  // метод проверяющий, совпадают ли две окружности
  publicbooleanequalsCircle(Circleo){
    if(this.r==o.r&&c.equalsPoint(o.c)){
      returntrue;
```

```
    }else{
      returnfalse;
    }
  }
}

publicclassMain{
  publicstaticvoidmain(String[]args){
    Circleo1 =newCircle();
    o1.print();
  }
}
```

Звернітьувагунате, щовметодахкласукілмивикористовувалиметодикласуточок, щозначноспростилопобудовадругогокласу.

Контрольніпитання

1. Дайтевизначенняоб’єктно-орієнтованогопрограмування.
2. Поняттякласутаоб’єктавООП.
3. ПоняттяполятаметодувООП.
4. Статичнітанестатичнікомпонентикласу.
5. Дайтевизначенняблокуініціалізації.
6. Дайтевизначенняконструкторукласу.
7. Дайтевизначенняконструктора.
8. Якіособливостіоголошенняконструктора?
9. Длячогопотрібніконструктори?
10. Щотакеперевантаженняметодів?
11. Якпередатимасиввякостіаргументаметода?

Методичнівказівкидовиконанняконтрольноїроботи (завдання 2)

Літературадовиконаннязавдання:[1] – стор. 184 – 196, [3] – стор. 232 – 283, [4] – стор. 353 – 382, а також [2]. Крімтого, нижченаведенідеяківідомостізуспадкуваннякласів, атакожпринципипобудовиабстрактнихкласівтаінтерфейсів, наведеніприкладитощо.

Метоючастиниєстворенняабстрактнихкласівтаінтерфейсівзаумовамивариантузавдання.

1 Абстрактні класи

Особливістю абстрактних класів являється наявність у їх складі хоча б одного абстрактного методу, тобто такого методу, реалізація якого не розписується, а вказується тільки його назва та тип значення, що повертається, якщо таке є. Водночас, в такому класі можуть міститися поля і конкретні методи, які, також як і абстрактні, будуть успадковані класами нащадками.

Абстрактний клас не може мати представників - для цього його необхідно наслідувати в підклас в якому запропонувати конкретну реалізацію абстрактних методів. У зв'язку з цим у складі абстрактного класу немає конструкторів.

Для того, щоб перетворити клас у абстрактний перед його ім'ям треба вказати модифікатор abstract.

Розглянемо приклад абстрактного класу A:

```
abstractclass A {
    int p1;
    A(){
        p1 =1;
    }
    voidprint(){
```

```
        System.out.println(p1);
    }
}
class B extends A {
}
publicclassMain{
    publicstaticvoid main(String[] args){
        A ob1;
        // ошибка: ob1 = new A();
        B ob2 =new B();// будет вызван конструктов по умолчанию
из A
        ob2.print();
    }
}
```

Java дозволить описати конструктори в класі A, але не дозволить ними скористатися (бо заборонено створювати об'єкти абстрактного класу). Зверніть увагу на те, що оголошення змінної ob1 як посилання, на об'єкт класу A теж не забороняється.

2 Абстрактні методи

Абстрактним називається метод, який не має реалізації в даному класі. Після круглих дужок, де перераховані його аргументи, ставиться не відкрита фігурна дужка, щоб почати блок опису методу, а крапка з комою. Тобто опис у абстрактноно методу відсутній. Перед ім'ям методу вказується при цьому модифікатор abstract.

Який сенс у створенні методу без реалізації? Адже його не можна буде використовувати. Якщо успадкувати клас і в нащадках перевизначити метод, задавши там його опис, то для об'єктів класів нащадків метод можна буде викликати (і працювати будуть описані в класах нащадках реалізації).

Щоб виключити можливість використання абстрактного методу, в Java введено таку вимогу: абстрактним є клас, в якому присутній хоча б один абстрактний метод.

Коли ж доречно використовувати абстрактні методи і класи? Спочатку розглянемо приклад ієрархії класів домашніх тварин, де немає ні абстрактних класів, ні абстрактних методів.

```
classPet{
    String name;
    int age;
    boolean hungry;
    void voice(){
    }
    void food(){
        hungry =false;
    }
}
classSnakeextendsPet{
    double length;
    void voice(){
        System.out.println("Шшш-ш-ш");
    }
}
classDogextendsPet{
    void voice(){
        System.out.println("Гав-гав");
    }
}
classPatrolDogextendsDog{
    void voice(){
        System.out.println("Ррр-р-р");
    }
}
classCatextendsPet{
    void voice(){
        System.out.println("Мяу-мяу");
    }
}
```

```
classFishextendsPet{
}
publicclassMain{
    publicstaticvoid main(String[] args){
        Pet zorka =newPet();
        zorka.food();
        Fish nemo =newFish();
        nemo.voice();
    }
}
```

Оскількинемаєякогосьзагальногозвуку, якийвидавалибвсідомашнітварини, томивкласі Pet нестализадаватиякусьреалізауіюметоду voice (), всередініметодунеробитьсязовсімнічого, алетимнеменшуньогоєтіло, відокремленийблокомзфігурнихдужок. Метод voice () хорошийпретендентнате, щобстатиабстрактним.

Крімтого, навряд чи можна завести собі домашню тваринуневизначеного виду, тобто у вас вдома цілком могли б жити Cat, Dog абонавіть Snake, але навряд чи вибзмогли завести тварину Pet, що є незрозумілоким.

Відповідно, в рамках реального завдання навряд чи буде потрібно створювати об'єкти класу Pet, а значить його можна зробити абстрактним. Розглянемо приклад з участю абстрактного класу і абстрактного методу:

```
abstractclassPet{
    String name;
    int age;
    boolean hungry;
    abstractvoid voice();
    void food(){
        hungry =false;
    }
}
```

```
classSnakeextendsPet{
    double length;
    void voice(){
        System.out.println("Шшш-ш-ш");
    }
}
classDogextendsPet{
    void voice(){
        System.out.println("Гав-гав");
    }
}
classPatrolDogextendsDog{
    void voice(){
        System.out.println("Ррр-р-р");
    }
}
classCatextendsPet{
    void voice(){
        System.out.println("Мяу-мяу");
    }
}
classFishextendsPet{
    void voice(){
    }
}
publicclassMain{
    publicstaticvoid main(String[] args){
        // ошибка: Pet zorka = new Pet();
        Fish nemo =newFish();
        nemo.voice();
    }
}
```

Звернітьувагу, щотепер, по-перше, минеможемостворюватиоб'єктиабстрактногокласу Pet, а, по-друге,

реалізаціяметоду voice () повиннабутиувсіхйогонащадків (хочабпорожняреалізація), якінеєабстрактнимикласами.

Хоча, ми могли б створити абстрактного нащадка:

```
abstractclassFishextendsPet{
}
```

Аленемоглибстворюватиоб'єктівкласуFish, намдовелосяброзширюватиклас, щобврезультатіотриматинеабстрактнеіствoрюватинайогоосновіоб'єкти. наприклад:

```
classGoldenFishextendsFish{
    void voice(){
    }
}
```

4 Інтерфейси

Інтерфейс - це повністю абстрактний клас, який не містить нічого крім декларацій методів із зазначенням їх семантики (типу значення, що повертається і кількості, типу і структури набору аргументів), а також набору констант, які за замовчуванням являються статичними і незмінними (static final). Опис інтерфейсу починається з ключового слова interface.

Інтерфейси призначені для рішення проблем з множинним спадкуванням - будь-який клас, незалежно від його положення в його класовій ієрархії, може реалізувати будь-яку кількість інтерфейсів. Для цього при описі класу необхідно перерахувати набір реалізованих інтерфейсів після ключового слова implements і реалізувати весь набір інтерфейсних методів у даному класі, в точності дотримуючись їх семантики. Дані методи повинні бути позначені модифікатором public.

Подібний підхід дозволяє створювати набір (масив) об'єктів, що має тип інтерфейсу, де кожен його елемент має тип кожногокласу, що реалізє цей інтерфейс. Це дозволяє викликати для елементів масиву весь набір інтерфейсних методів, причому конкретна реалізація кожного методу буде призначена динамічно залежно від типу конкретного елементу.

Таким чином, інтерфейс описує протокол взаємодії між різними класами через загальний для них усіх набір методів, де кожен клас реалізує їх відповідно до логіки його роботи і призначення.

Розглянемо наступний приклад:

```
interfaceInstruments{
    finalstaticString key ="До мажор";
    publicvoid play();
}
classDrumimplementsInstruments{
    publicvoid play(){
        System.out.println("бум бац бац бум бац бац");
    }
}
classGuitarimplementsInstruments{
    publicvoid play(){
        System.out.println("до ми соль до ре до");
    }
}
```

Оскільки всі властивості інтерфейсу повинні бути константними і статичними, а всі методи загальнодоступними, то відповідні модифікатори перед властивостями і методами дозволяється не вказувати. Тобто інтерфейс можна було описати так:

```
interfaceInstruments{
  staticpublicStringkey="Домажор";
```

```
  voidplay();
}
```

Але коли метод play () буде описуватися в реализующем інтерфейс класі, перед ним все одно необхідно буде явно вказати модифікатор public.

5 Множинне успадкування інтерфейсів

Java не підтримує множинне успадкування класів. Це пояснюється тим, що таке спадкування породжує деякі проблеми.

Найчастіше вказуються неоднозначності, що виникають при так званому «ромбоподібному» спадкуванні, коли один клас «A» є спадкоємцем двох інших класів «B» і «C», які в свою чергу обидва є спадкоємцями класу «D». Проблема допустимості множинного спадкоємства криється в наступному. Припустимо, що в батьку A визначався якийсь метод m1 (). І цей же метод ми викликаємо для об'єкта класу D. А що якщо m1 () був перевизначений в класах B і C. Яка реалізація з трьох буде працювати при виклику методу m1 () для об'єкта класу D? Від неодназначно можна було б позбутися зажадавши в описаному випадку при виклику уточнювати при виклику, який з методів потрібно (так і зроблено в деяких мовах), але в Java від множинного спадкоємства класів вирішили відмовитися.

Замість множинного успадкування класів в Java введено множинне спадкування інтерфейсів, яке часічно вирішує проблеми (але, як буде показано в прикладі далі, на жаль, не всі).

Розглянемо приклад, де реалізовано два інтерфейси з методами доступними для вантажного та для легкового транспорту. Клас Pickup (пікап) повинен володіти як можливістю перевезення вантажів, так і пасажирів, тому він реалізує відразу обидва інтерфейса:

```
interfacePassangersAuto{
    void transportPassangers();
}
```

```
interfaceCargoAuto{
    void transportCargo();
}
classTruckimplementsCargoAuto{
    finalstaticint a =1;
    publicvoid transportCargo(){
        System.out.println("Везу груз");
    }
}
classSedanimplementsPassangersAuto{
    publicvoid transportPassangers(){
        System.out.println("Везу пассажиров");
    }
}
classPickupimplementsCargoAuto,PassangersAuto{
    publicvoid transportCargo(){
        System.out.println("Везу груз");
    }
    publicvoid transportPassangers(){
        System.out.println("Везу пассажиров");
    }
}
```

Часто всю відкриту частину класу (тобтозагальнодоступні методи) зумовлює як раз в інтерфейсі. Тоді глянувши на один лише інтерфейс можна зрозуміти які ж методи повинні використовуватися для взаємодії з об'єктами даного класу. Тобто інтерфейси цілком відповідають принципам інкапсуляцуіі. Як, втім, і принципу поліморфізму. Адже в декількох класах метод деякого інтерфейсу може бути реалізований по-різному, хоча і з одним і тим же ім'ям.

Але інтерфейси, як говорилося вище, не є досконалим інструментом позбавленим всяких недоліків. Розглянемо приклад, коли у нас є два

інтерфейси, в кожному з яких є властивості з однаковими іменами (але, можливо, різними значеннями) і методи з однаковими іменами.

Успадкувавши клас від пари цих інтерфейсів ми не зможемо звертатися до властивості його об'єктів безпосередньо, без вказівки того, який з двох інтерфейсів ми мали на увазі. Це обмеження існує тому, що в інтерфейсах властивостям може даватися різний початкове значення і, відповідно, програма не зможе визначити яке ж значення вибрати.

Також до властивості можна звернутися як до статичної властивості одного з інтерфейсів (зрозуміло, це можна робити і якщо у властивостей були б різні імена).

Проблема зникне, якщо перед зверненням до властивості ми наведемо об'єкт до одного з батьківських інтерфейсів (нагадаємо, що будь-який об'єкт можна явно привести до класу або інтерфейсу його батька прямого або транзитивного).

На жаль, створити окремі реалізації для двох однойменних методів з різних інтерфейсів в класі спадкоємця не вийде (щоб потім ними можна було користуватися через той же приведення об'єктів до потрібного інтерфейсу). Якщо клас реалізує кілька інтерфейсів, в яких є однойменні методи, то в ньому може здаватися лише одна загальна для всіх реалізація цих методів (і це вже обмежує поліморфізм при множині спадкування через інтерфейси в Java).

Отже, кодприкладу:

```
interfaceInterface1{
    int someField =100;
    String someMethod();
}
interfaceInterface2{
    int someField =200;
    String someMethod();
}
classSomeClassimplementsInterface1,Interface2{
```

```
    publicString someMethod(){
        return"It Works";
    }
}
publicclassMain{
    publicstaticvoid main(String[] args){
        SomeClass a =newSomeClass();
        System.out.println( a.someMethod());// It works
        System.out.println( a.someField );// ошибка
        System.out.println(((Interface1) a).someField );// 100
        System.out.println(Interface1.someField );// 100
    }
}
```

Контрольні питання

1. Поясніть використання ключового слова "extends".
2. Як конструктори підкласів можуть викликати методи суперкласу?
3. Поясніть суть використання перевизначених методів?
4. Яка різниця між перевантаженим та перевизначеним методом?
5. Дайте визначення статичному методу?
6. В чому полягає особливість абстрактних класів?
7. Чому у складі абстрактних класів не має конструкторів?
8. Коли виникає необхідність у використанні абстрактних класів?
9. Дайте визначення інтерфейсу.
10. Яке призначення інтерфейсів?
11. Який модифікатор доступу слід використовувати для інтерфейс них методів?

Методичні вказівки до виконання контрольної роботи (завдання 3)

Література до виконання завдання:[3] – стор. 119-135, [4] – стор. 353 – 382, а також [2]. Крім того, нижче наведені деякі відомості з успадкування класів, а також принципи побудови абстрактних класів та інтерфейсів, наведені приклади тощо.

Метою завдання єознайомлення з особливостями побудови класів, що містять у своєму складі внутрішні класи, навчитися будувати подібні моделі і працювати з ними, розв’язуючи практичні задачі.

Нестатичні вкладені класи прийнято називати внутрішніми (inner) класами. Доступ до елементів внутрішнього класу можливий з зовнішнього класу тільки через об'єкт внутрішнього класу, який повинен бути створений в коді методу зовнішнього класу. Об'єкт внутрішнього класу завжди асоціюється (приховано зберігає посилання) з створив його об'єктом зовнішнього класу - так званим зовнішнім (enclosing) об'єктом.

Наприклад, є клас Book, всередині якого визначено клас Publisher:

```
classBook{

    String name;
    String author;
    intyear;
    publicPublisher publisher;

    Book(String name, String author, intyear, String publ){
```

```
        this.name = name;
        this.author = author;
        this.year = year;
        publisher = newPublisher(publ);
    }

    classPublisher{

        publicString name;
        publicBook book;

        publicPublisher(String name){
            book=Book.this;
            this.name=name;
        }
    }
}
```

Внутрішній клас поводиться як звичайний клас за тим винятком, що його об'єкти можуть бути створені тільки всередині зовнішнього класу.

Внутрішній клас має доступ до всіх полів зовнішнього класу, в тому числі закритим за допомогою модифікатора private. А саме посилання на зовнішній клас з внутрішнього можна отримати за допомогою виразу Book.this, де спочатку йде ім'я зовнішнього класу.

Тепер використовуємо класи:

```
Book b1 = newBook("Война и мир", "Л. Н. Толстой", 1863,
"ХудКнига");
System.out.println(b1.publisher.name);
```

Об'єкти внутрішніх класів можуть бути створені тільки в тому класі, в якому внутрішні класи визначені. В інших зовнішніх класах об'єкти внутрішнього класу створити не можна.

Ще однією особливостей внутрішніх класів є те, що їх можна оголосити всередині будь-якого контексту, в тому числі всередині методу і навіть в циклі:

```
classBook{

    String name;
    String author;
    intyear;

    Book(String name, String author, intyear){

        this.name = name;
        this.author = author;
        this.year = year;
    }
    publicvoidsetPublisher(String publ){

        classPublisher{
            voiddisplayInfo(){

                System.out.println("Издатель: "+ publ);
            }
        }

        Publisher publisher = newPublisher();
        publisher.displayInfo();
    }
}
```

Потім при використанні нам досить викликати метод setPublisher об'єкта Book:

```
Book b1 = newBook("Война и мир", "Л. Н. Толстой", 1863);
```

```
b1.setPublisher("ООО ХудКнига");
```

ЗАВДАННЯ ДО КОНТРОЛЬНОЇ РОБОТИ

Контрольна робота складається з трьох частин, кожна з яких базується на попередній і змінює (або доповнює) її.

Вибір варіанта

Варіант вибирається відповідно до номера залікової книжки. Номер варіанта визначається як остання цифра залікової книжки, ділення на 2. Результат округляється до більшого цілого. Наприклад, якщо остання цифра залікової книжки 5, то номер варіанта - 3.

Зміст звіту

Звіт надається у електроному вигляді на почтову скриню кафедри на перевірку у виглядіи:

1) Титульний аркуш

2) Завдання на виконання

3) UML-модель реалізації для кожного завдання.

4) Вихідний код кожного завдання на мові Java з вказівку коментарів до коду.

5) Вихідний код класу (ів), з використання якого (их) вироблялося

тестування конструкторів і методів реалізованих класів.

2.2 ВАРІАНТИ ЗАВДАНЬ

ЗАВДАННЯ 1

1. Створіть пакет. Ім'я створюваного пакета повинно відповідати варіанту.
2. Для кожної використовуваної змінної або константи, атрибута, методу, класу вибирати імена відповідно до конвенціями іменування.
3. Реалізуйте класи відповідно до варіанта.
4. Визначте поля класу як приватні (доступні тільки усередині класу), а методи і конструктори - публічними (доступні всім іншим класам).
5. При використанні значень за замовчуванням (наприклад, при реалізації конструкторів) для цих самих значень використовувати статичні публічні константи. Імена констант записуються наступним чином: всі букви заголовні, слова розділяються між собою символами підкреслення.
6. Створіть окремий клас для тестування конструкторів і викликів методів створених відповідно до варіанта класів.
7. Якщо написано масив - значить масив, а не ArrayList.

Варіант № 0

1. Пакет - *policlinic*.
2. Створіть публічний клас *OutpatientsCard* - карти людини,

зареєстрованого в поліклініці. Клас не зберігає в явному вигляді інформацію про поліклініці.

- Кожен пацієнт характеризується ім'ям, прізвищем, адресою проживання;
- Кожен пацієнт має страховий поліс, і характеризується його номером;
- Конструктор може приймати ім'я і прізвище (номер поліса = 0 і адреса - порожній);
- Конструктор може приймати ім'я, прізвище та адресу (номер поліса = 0);
- Конструктор може приймати ім'я, прізвище, адресу та номер поліса;
- Створіть метод отримання імені;
- Створіть метод зміни імені;
- Створіть метод отримання прізвища;
- Створіть метод зміни прізвища;
- Створіть метод отримання номера страхового поліса;
- Створіть метод зміни номера страхового поліса;
- Створіть метод отримання адреси;
- Створіть метод зміни адреси;

ЗСтворіть публічний клас *Polyclinic* - поліклініки деякого району.

- Поліклініка характеризується номером і адресою;
- Клас зберігає явно масив карт пацієнтів, зареєстрованих в поліклініці;
- Конструктор може приймати номер і адресу поліклініки (довжина масиву пацієнтів = 0)
- Конструктор може приймати номер і адресу, а так же масив пацієнтів;
- Створіть метод отримання номера поліклініки;
- Створіть метод зміни номера поліклініки;

- Створіть метод отримання адреси поліклініки;
- Створіть метод зміни адреси поліклініки;
- Створіть метод, який повертає загальне число пацієнтів, зареєстрованих в поліклініці;
- Створіть метод, який повертає посилання карту пацієнта за номером страхового поліса;
- Створіть метод, який повертає посилання на масив карт пацієнтів, які проживають за адресою, що вказана;
- Створіть метод видалення карти (приймає в якості вхідного параметра номер поліса, видаляє відповідний цими даними елемент з масиву карт);
- Створіть метод додавання карти (приймає в якості вхідного параметра посилання на екземпляр класу, розширює масив карт шляхом додавання нового елемента в кінець масиву);
- Створіть метод, який повертає масив карт;
- Створіть метод, який повертає масив карт, відсортованого за адресами;

Варіант № 1

1. Пакет - *university*.

2. Створіть публічний клас *Student* - студента деякої спеціальності деякого університету. Клас не зберігає явно інформацію про спеціальності, номер групи \ потоку, предметах, університеті.

- кожен студент характеризується ім'ям, прізвищем, роком надходження, унікальним 6-ти значний номером залікової книжки.
- Конструктор може приймати ім'я і прізвище. При цьому номер залікової книжки - 0;

- Конструктор, який приймає ім'я, прізвище, номер залікової книжки;
- Створіть метод отримання імені;
- Створіть метод зміни імені;
- Створіть метод отримання прізвища;
- Створіть метод зміни прізвища;
- Створіть метод отримання номера залікової книжки;
- Створіть метод зміни номера залікової книжки;
- Створіть метод отримання року надходження;
- Створіть метод зміни року надходження;

3 Створіть публічний клас *Group* - студентської групи. Клас не зберігає явно спеціальність і ім'я університету.

- Кожна група має свій номер (унікальний, в межах спеціальності)
- Клас зберігає явно масив студентів;
- Конструктор може приймати номер групи (в цьому випадку кількість студентів = 0)
- Конструктор може приймати номер групи, кількість студентів (масив студентів тільки ініціюється, але елементи його - порожні);
- Конструктор може приймати масив студентів;
- Створіть метод отримання номера групи;
- Створіть метод зміни номера групи;
- Створіть метод, який повертає загальне число студентів групи;
- Створіть метод, який повертає посилання на студента за номером залікової книжки;
- Створіть метод видалення студента (приймає в якості вхідного параметра номер залікової книжки студента, якого потрібно видалити, видаляє відповідний цими даними елемент з масиву студентів);
- Створіть метод додавання студента (приймає в якості вхідного

параметра посилання на екземпляр класу Student, розширює масив студентів шляхом додавання нового елемента в кінець масиву);

- Створіть метод, який повертає масив студентів;
- Створіть метод, який повертає масив студентів, відсортований за іменами (і якщо однакові прізвища - то по іменах);

Варіант № 2

1. Пакет - organization.

2 Створіть публічний клас Employee - працівника деякої організації. Клас не зберігає явно номер або ім'я підрозділи і організації, в якій працює працівник.

- кожен працівник займає певну посаду;
- Кожен працівник отримує певну платню;
- Кожен працівник характеризується ім'ям і прізвищем;
- Конструктор може приймати ім'я і прізвище (посада - інженер, платню - 30к руб.);
- Конструктор може приймати ім'я, прізвище, посада, платню;
- Створіть метод отримання імені;
- Створіть метод зміни імені;
- Створіть метод отримання прізвища;
- Створіть метод зміни прізвища;
- Створіть метод отримання посади;
- Створіть метод зміни посади;
- Створіть метод отримання платні;
- Створіть метод зміни платні.

3 Створіть публічний клас Department - підрозділи деякої організації. Клас не зберігає явно номер підрозділу і ім'я організації, частиною якої є.

- різні підрозділи мають різні імена;
- Клас зберігає явно масив своїх працівників;
- Конструктор може приймати ім'я підрозділи (в цьому випадку кількість працівників = 0);
- Конструктор може приймати масив працівників;
- Створіть метод отримання імені підрозділу;
- Створіть метод зміни імені підрозділу;
- Створіть метод, який повертає загальне число працівників підрозділу;
- Створіть метод, який повертає сумарну зарплату всіх працівників, що відносяться до даного підрозділу;
- Створіть метод, який повертає посилання на працівника по прізвища та імені;
- Створіть метод звільнення працівника (приймає в якості вхідних параметрів прізвище, ім'я, посаду працівника, якого потрібно видалити, видаляє відповідний цими даними елемент з масиву працівників);
- Створіть метод прийому працівника на роботу (приймає в якості вхідних параметрів посилання на екземпляр класу Employee, розширює масив працівників шляхом додавання нового елемента в кінець масиву);
- Створіть метод, який повертає масив працівників відділу;
- Створіть метод, який повертає масив працівників відділу, відсортований за іменами (і якщо однакові прізвища - то по іменах).

Варіант № 3

1. Пакет - *logistics*.

2. Створіть публічний клас *Truck* - вантажного автомобіля. Клас не зберігає явно інформацію про вантаж, водії, маршруті.

- кожен вантажний автомобіль характеризується держ. реєстраційним номером, маркою, вантажопідйомністю (т), об'ємом кузова (куб. м).
- конструктор може приймати держ. рег. номер (інші поля приймають значення за умовчанням деякої моделі вантажного автомобіля, обраній студентом)
- конструктор може приймати держ. реєстраційний номер, марку, вантажопідйомність (т), об'ємом кузова (куб. м)
- створіть метод отримання держ. рег. номера
- створіть метод зміни держ. рег. номера
- створіть метод отримання марки
- створіть метод зміни марки
- створіть метод отримання вантажопідйомності
- створіть метод зміни вантажопідйомності
- створіть метод отримання обсягу кузова
- створіть метод зміни обсягу кузова

3 Створіть публічний клас *TruckFleet* - парку автомобілів деякої логістичної організації. Клас не зберігає явно інформацію про вантажі, водіях, маршрутах. класс явным образом хранит массив грузовых автомобилей;

- конструктор може приймати число вантажних автомобілів (в цьому випадку инициализируется відповідний масив, але самі елементи не инициализируются);
- конструктор може приймати масив вантажних авто;
- створіть метод, який повертає загальне число вантажівок;
- створіть метод, який повертає посилання на вантажівку по його держ. рег. номеру;
- створіть метод, який повертає посилання на масив вантажівок,

менше заданої вантажопідйомності;

- створіть метод, який повертає посилання на масив вантажівок, менше заданого обсягу кузова;
- створіть метод видалення вантажівки (приймає в якості вхідного параметра держ. рег. номер, видаляє відповідний цими даними елемент з масиву вантажівок);
- створіть метод додавання вантажівки (приймає в якості вхідного параметра посилання на екземпляр класу Truck, розширює масив шляхом додавання нового елемента в кінець масиву);
- створіть метод, який повертає масив всіх авто;
- створіть метод, який повертає масив вантажівок, відсортоване за вантажопідйомністю.

Вариант № 4

1. Пакет - *text*.

2. Створіть публічний клас *Paragraph* - абзацу текстового документа. Клас не зберігає явно розташування в тексті, число рядків.

- кожен абзац характеризується рядком, безпосередньо містить весь текст абзацу;
- кожен абзац характеризується відступом нового рядка (число символів, а не сантиметри);
- конструктор за замовчуванням (без параметрів) створює «порожній» абзац - характеризується символом нового рядка, і відступом = 0;
- конструктор може приймати значення відступу (в цьому випадку рядок - порожня);
- конструктор може приймати значення відступу і рядок - текст.
- створіть метод отримання рядка тексту;

- створіть метод змінити текстову;
- створіть метод отримання відступу троки;
- створіть метод зміни відступу троки;

3 Створіть публічний клас *Text* - тексту.

- клас характеризується максимальним числом символів в рядку;
- клас явно зберігає в собі масив абзаців;
- конструктор за замовчуванням (довжина масиву абзаців = 0, число символів 80);
- конструктор може приймати масив абзаців (число символів в рядку = 80);
- конструктор може приймати масив абзаців і число символів в рядку;
- створіть метод, який повертає загальне число абзаців;
- створіть метод, який повертає загальне число рядків тексту;
- створіть метод, який повертає посилання на абзац по його номеру (номер абзацу = номеру в масиві);
- створіть метод, вставляють абзац після абзацу (приймає посилання на новий абзац та номер абзацу, після якого потрібно вставити новий)
- створіть метод видалення абзацу по його номеру;
- створіть метод зміни абзацу за його номером (приймає посилання на новий абзац та номер абзацу, який потрібно замінити новим);
- створіть метод, який повертає масив абзаців

Варіант № 5

1. Пакет - *bank*.

2. Створіть публічний клас *Account* - рахунки банку. Клас не містить в явному вигляді інформації про його «власника»

- кожен рахунок характеризується своїм унікальним номером, залишком коштів на рахунку (в рублях);
- конструктор може приймати номер рахунку (залишок = 0);
- конструктор може приймати номер рахунку, залишком коштів на рахунку
- створіть метод отримання номера рахунку
- створіть метод зміни номера рахунку
- створіть метод отримання залишку коштів на рахунку
- створіть метод зміни залишку коштів на рахунку

3 Створіть публічний клас *Client* - клієнта банку.

- клієнт характеризується ім'ям, прізвищем, серією і номером паспорта;
- клас явно зберігає масив рахунків;
- конструктор може приймати ім'я, прізвище, паспортні дані (розмір масиву рахунків = 0);
- конструктор може приймати ім'я, прізвище, паспортні дані та масив рахунків;
- створіть метод, який повертає посилання на рахунок по його унікальним номером;
- створіть метод, який повертає масив всіх рахунків;
- створіть метод, який повертає сумарний залишок на всіх рахунках;
- створіть метод, який повертає масив рахунків з позитивним залишком на рахунку;
- створіть метод видалення рахунку по його номеру;
- створіть метод додавання рахунку (приймає в якості вхідного параметра посилання на рахунок, розширює масив рахунків шляхом додавання нового рахунку в кінець масиву):
- створіть метод зменшення розміру залишку рахунку (приймає посилання на рахунок і розмір суми);

- створіть метод збільшення розміру залишку рахунку (приймає посилання на рахунок і розмір суми);

ЗАВДАННЯ 2

1. У вже створеному пакеті реалізуйте (змініть) класи і інтерфейси відповідно до варіанта.
2. Як і раніше, поля класу - приватні (доступні тільки усередині класу).
3. При використанні значень за замовчуванням (наприклад, при реалізації конструкторів) для цих самих значень використовувати статичні публічні константи. Імена констант записуються наступним чином: всі букви заголовні, слова розділяються між собою символами підкреслення.
4. Створіть окремий клас для тестування конструкторів і викликів методів створених відповідно до варіанта класів.
5. Поля - дати, повинні мати тип java.util.Date.
6. При перейменування чогось слід користуватися засобами рефакторінга середовища розробки.

Варіант № 0

1. Пакет - *policlinic*.

2. Створіть абстрактний клас *MedicalInsurancePolicy* - поліса мед. страхування. Клас містить:

приватні поля:

- Номер поліса
- Назва страхової компанії

конструктори:

- Без параметрів (назви компанії немає, номер поліса 0)
- З 2-ма параметрами: номером поліса і назвою компанії.

методи:

- Геттери і сеттери полів

3. Створіть клас *ObligatoryMedicalInsurancePolicy* - поліса обов'язкового медичного страхування. Цей клас розширює (успадковує) клас *MedicalInsurancePolicy*. Додаткових елементів цей клас не пропонує.

4. Створіть клас *VoluntaryMedicalInsurancePolicy* - поліса добровільного медичного страхування. Цей клас також розширює (успадковує) клас *MedicalInsurancePolicy*. Цей клас визначає додаткові елементи:

Приватні поля:

- Загальна сума страховки;
- Виплачена страхова сума;
- конструктори:
- без параметрів (загальна сума страховки - 100 000, виплачена стразового сума = 0)
- з одним параметром - загальна сума страховки (виплачена страхова сума = 0)

методи:

- Геттери і сеттери полів

5.Змініть поля, конструктори, методи класу *OutpatientsCard*, пов'язані з номером мед. поліса. Тепер цей клас повинен працювати не з числом (номер мед. Полісу), а з екземпляром класу *MedicalInsurancePolicy*.

- приватне поле, що містить номер поліса, потрібно видалити. Замість нього додати поле типу MedicalInsurancePolicy.
- Конструктори, які беруть номер поліса, тепер приймають в якості вхідного параметра посилання типу

MedicalInsurancePolicy.

- Методи отримання номер поліса змінити на методи отримання посилання на поліс (тип MedicallnsurancePolicy)

7. Змініть клас *Policlinic*.
   - змініть реалізацію методу, який повертає посилання на карту пацієнта за номером страхового поліса, а також методу видалення карти пацієнта за номером поліса. Методи і раніше приймають номер страхового поліса, змінюється тільки реалізація.

7.Перейменуйтекласс*OutpatientsCard*в*SocialOutpatientsCard.*

8.Створить інтерфейс,*OutpatientsCard*,що містить методи:

- отримання імені;
- зміни імені;
- отримання прізвища;
- зміни прізвища;
- отримання адреси;
- зміни адреси;
- отримання посилання на мед. поліс
- зміна посилання на мед. Поліс

9 Класс *SocialOutpatientsCard*повинен реалізовувати інтерфейс*Out-patientsCard.*

10 Створіть клас *Bill* - рахунок за лікування, який містить 3 приватних поля:

- дата (екземпляр класу java.util.Date),
- Сума за надані в цей день медичні послуги,
- Вид медичної послуги (перерахування, можливі значення цього перерахування - стоматологія, ендокринологія, хірургія, проф. Огляд, і т.п. (придумайте самостійно ще кілька варіантів), а також відкриті геттери і сеттери до них.

11 Створіть клас *PaidOutpatientsCard* - карти людини, що користується

платними послугами поліклініки. Клас повинен реалізувати інтерфейс *OutpatientsCard* і, крім методів і полів, необхідних для реалізації цього інтерфейсу, він повинен містити наступні елементи:

- приватний список (клас ArrayList <Bill>), що містить інформацію про те скільки і коли пацієнт залишив грошей в поліклініці (іншими словами, список містить екземпляри класу Bill)
- конструктор без параметрів (який ініціює список нульової довжини)
- конструктор може приймати ім'я і прізвище (поліс = null і адреса - порожній, ініціюється список нульової довжини);
- конструктор може приймати ім'я, прізвище та адресу (поліс = null, ініціюється список нульової довжини);
- конструктор може приймати ім'я, прізвище, адресу та посилання на екземпляр поліса (MedicalInsurancePolicy);
- конструктор, що приймає ім'я, прізвище, адресу, посилання на екземпляр поліса і список рахунків (екземпляр ArrayList <Bill>)
- геттер і сетер для списку рахунків
- метод, який повертає загальну суму, заплачену пацієнтом поліклініці
- метод, який повертає список рахунків в заданий день (якщо рахунок тільки один, повертає список з одного елемента, якщо рахунків в заданий день не було, повертає список з 0 елементів)
- метод, який додає рахунок в кінець списку рахунків
- метод, що видаляє рахунок з відповідною датою і розміром платежу (передаються в якості параметрів методу)

12 Змініть клас *Policlinic*. Додайте методи:

- метод, який повертає загальне число пацієнтів, що

обслуговуються за полісами ОМС

- метод, який повертає загальне число пацієнтів, що обслуговуються за полісами ДМС
- метод, який повертає загальне число пацієнтів, хоча б один раз користувалися платними послугами поліклініки (не через ДМС)
- метод, який повертає загальну суму перерахувань (оплат за рахунком) в заданий місяць і рік.

Варіант № 1

1. Пакет - *university.*

2. Створіть клас *Payment* - плата за навчання, що містить: приватні поля:

- дата (екземпляр класу java.util.Date),
- розмір суми, переведеної студентом на рахунок університету конструктори:
- без параметрів (дата містить null, сума - 0)
- з двома параметрами - датою і сумою методи:
- геттери і сеттери для приватних полів

3. Створіть клас *ContractStudent*, що розширює (успадковує) клас *Student*. Цей клас додає:

- приватне поле - зв'язний список платежів (клас *LinkedList <Payment>*)
- приватне поле - вартість навчання за семестр (в завданні передбачається що ця вартість протягом усього періоду навчання змінюватись не буде)
- конструктор без параметрів (який ініціює список нульової довжини)
- конструктор може приймати ім'я і прізвище (номер залікової книжки - 0, який ініціює список нульової довжини);

- конструктор, що приймає ім'я, прізвище, номер залікової книжки (який ініціює список нульової довжини);
- конструктор, що приймає ім'я, прізвище, номер залікової книжки, список платежів (екземпляр класу *LinkedList <Payment>*)
- геттер і сетер для списку платежів і вартості навчання
- метод, який повертає розмір заборгованості студента на поточний момент (метод не приймає параметрів. Виходимо з припущення, що плата за навчання не змінюється. Знаючи поточну дату і рік надходження, можна з'ясувати скільки семестрів студент провчився і визначити суму, яку він повинен був внести за весь термін навчання).
- метод, який додає платіж в кінець списку платежів
- метод, що видаляє платіж з відповідною датою і розміром платежу

4. Створіть інтерфейс *Event* - заходи, в якому брав участь студент, що описує методи:
   - геттери і сеттери для дати проведення заходу (працюють з екземплярами класу java.util.Date)
   - геттери і сеттери для назви міста, в якому проводився захід
5. Визначте клас *Olympiad,* який реалізує інтерфейс Event.

Містить приватні поля - дата і назва міста, а так само місце (ціле число), яке зайняв студент на олімпіаді.

Реалізувати методи доступу (геттери і сеттери) для приватних полів.

6. Визначте клас *Conference,* який реалізує інтерфейс Event.

Містить приватні поля - дата і назва міста, а так само назва доповіді (статті), з яким (якою) студент виступав на конференції.

Реалізувати методи доступу (геттери і сеттери) для приватних полів.

7. Визначте клас *Competition*, який реалізує інтерфейс *Event*.

Містить приватні поля - дата і назва міста, а так само назва проекту і виграна сума (0 - якщо немає =)))

Реализовать методы доступа (геттеры и сеттеры) для приватных полей.

8. Визначте інтерфейс *Activist* - учасника різних конкурсів, олімпіад і т.п. Інтерфейс визначає наступні методи:

- метод, який повертає загальну кількість заходів, в яких брав участь студент
- метод, який повертає число призових місць, зайнятих на олімпіадах
- метод, який повертає число доповідей на конференціях
- метод, який повертає рядок, що складається з назв проектів (розділених переходом на новий рядок), за які студент отримав винагороду на змаганнях

9. Змініть клас *Studen*t. Він повинен реалізовувати інтерфейс *Activist.* Для реалізації методів інтерфейсу, додати приватний поле - список подій (клас ArrayList <Event>) в яких брав участь студент.Помимо методов, реализуемых в соответствии с интерфейсом Activist, добавить следующие методы:

- вставки інформації про подію в кінець списку подій
- видалення події зі списку за датою
- пошуку події за датою (метод повертає посилання на подію)

Крім того, внесіть зміни в конструктори класу, для того, щоб вони ініціювали список подій.

10. Змініть клас *Group*.

Додайте методи: метод, возвращающий список студентов - активистов (которые хоть один раз участвовали в каком-то мероприятии)

- метод, який повертає список «привілейованих» студентів - які хоч раз займали призове місце на олімпіаді або в конкурсі.
- метод, який повертає число активістів в групі
- метод, який повертає число бюджетників в групі

- метод, який повертає число контрактників в групі
- метод, який повертає список боржників (які сплатили за вчасно рахунок за контрактом)

Варіант № 2

1. Пакет - *organization.*
2. Створітьперерахування*JobTitles*назвпосад, передбачитинаступніпосади: начальникпідрозділу (DepartmentBoss), інженер (Engineer), секретар (Clerk), директор (BigBoss). Можете додати ще посад по¬желанію.
3. Клас *Employee* зробіть абстрактним. Додайте наступні елементи:
   - приватні поля: дата прийому на роботу (екземпляр класу Date);
   - конструктори: конструктор, який приймає ім'я, прізвище, посада, платню, дату прийому;
   - методи: гетер и сеттер даты приема;
   - абстрактний відкритий метод, який повертає щомісячну премію;

Змініть:

- приватне поле, що містить посаду співробітника, повинно бути екземпляром перерахування *JobTitles*.
- змініть конструктори і методи, що працюють з назвою посади, відповідно до того, що поле тепер має тип перерахування.

4. Створіть клас *FullDayEmployee* штатного співробітника, який розширює (успадковує) клас *Employee*.
   - Додайте реалізацію методу, який повертає щомісячну премію. Вона обчислюється як число повних років, які пропрацював співробітник в компанії, ділення на 20. Крім того, якщо зарплата нараховується в січні (тобто поточний місяць - січень), премія збільшується на розмір окладу (для визначення поточної дати використовується клас *Calendar*).

- Додайте такі ж конструктори, що і в класі *Employee*

5. Створіть клас *HalfDayEmployee* - зовнішнього сумісника, який розширює (успадковує) клас Employee.

- Додайте реалізацію методу, який повертає щомісячну премію. Цей метод повертає 0.
- Додайте такі ж конструктори, що і в класі Employee.

6. Створіть клас *BusinessTravel* - відрядження. Цей клас містить:

Приватний поля:

- Дата відбуття з підприємства у відрядження
- дата прибуття
- Вартість трансферу до місця і назад
- добові

Конструктори:

- конструктор за замовчуванням, не ініціює поля (порожній конструктор)
- конструктор з параметрами: дата відбуття з підприємства у відрядження, дата прибуття, вартість трансферу до місця і назад, добові.

Методи:

- відкриті гетери і сеттери полів;
- метод повертає число повних днів між датами відбуття і прибуття;
- метод, який повертає загальну суму витрачених на відрядження грошей (трансфер + добові * к-ть днів)

7. Створіть інтерфейс *BusinessTraveller* - працівника, що направляється у відрядження. Цей інтерфейс описує наступні методи:

- метод додавання інформації про відрядження;
- метод видалення інформації про відрядження (приймає параметр дату відбуття);
- метод, який повертає (посилання на) екземпляр класу

BusinessTravel за датою (якщо введена дата потрапляє в інтервал між початком і кінцем відрядження);

- метод, який повертає середню тривалість відряджень працівника;
- метод, який повертає середній інтервал між відрядженнями в днях.

8. Клас *FullDayEmployee* повинен реалізовувати інтерфейс BusinessTraveller. Для цього визначте приватне поле типу ArrayList <BusinessTravel> - цей список буде містити всю інформацію про відрядження співробітника. При створенні екземпляра співробітника, в конструкторах, це поле ініціюється списком нульової довжини. На основі цього поля реалізуйте методи, описувані в інтерфейсі.

9. Замініть клас*Department*.

Додайте методи:

- Відкритий метод, який повертає список (*ArrayList <FullDayEmployee>*) штатних співробітників;
- Відкритий метод, який повертає список (*ArrayList <HalfDayEmployee>*) зовнішніх сумісників;
- Відкритий метод, який повертає список (*ArrayList <BusinessTraveller>*) співробітників, які перебувають у відрядженні а зараз;
- Відкритий метод, який повертає список (*ArrayList <BusinessTraveller>*) співробітників, які перебувають у відрядженні зазначеного числа (виступає параметром методу);

Варіант № 3

1. Пакет - *logistics*.
2. Створіть абстрактний клас *ContainerCargo* - вантажу, що знаходиться в контейнері. Цей клас містить: приватне поле, що містить вага контейнера з вантажем (вага) в кілограмах

конструктори:

- без параметрів
- приймає один параметр - вага методи:
- відкриті методи отримання і зміни ваги
- відкритий абстрактний метод, який повертає обсяг контейнера
- відкритий абстрактний метод, який повертає обсяг контейнера в заданих одиницях виміру (одиниця виміру передається як вхідний параметр методу, і має тип *VolumeUnitEnumeration*)

3. Створіть перерахований *VolumeUnitEnumeration* - одиниць виміру обсягу. Можливі значення перерахування: CubicMetre, Littre, CubicCentimetre.

4. Клас *BoxedCargo* Створіть - вантажу, поміщеного в контейнер прямокутної форми. Цей клас розширює (успадковує) клас *ContainerCargo*. Він містить такі елементи:

приватні поля:

- висота
- ширина
- Довжина (все три поля хранят значения в метрах) конструктори:
- конструктор без параметрів,
- конструктор, що приймає параметри: вага, висота, ширина, довжина.

відкриті методи:

- гетери і сеттери полів: висота, ширина, довжина
- метод, який повертає обсяг контейнера
- метод, який повертає обсяг контейнера в заданих одиницях виміру (одиниця виміру передається як вхідний параметр методу, і має тип VolumeUnitEnumeration)

5. Клас *TankedCargo* Створіть - рідкого вантажу в контейнері циліндричної форми. Цей клас розширює (успадковує) клас *ContainerCargo*. Він містить такі елементи:

приватні поля:

- висота
- Радіус(обидва поля зберігають значення в метрах)

конструктори:

- конструктор без параметрів,
- конструктор, що приймає параметри: вага, висота, радіус.

відкриті методи:

- гетери і сеттери полів: висота, радіус
- метод, який повертає обсяг контейнера
- метод, який повертає обсяг контейнера в заданих одиницях виміру (одиниця виміру передається як вхідний параметр методу, і має тип VolumeUnitEnumeration)

6. Створіть *CargoTransport* Інтерфейс, що описує такі методи:

- метод отримання рег. номера (рядок)
- метод зміни рег. номера
- метод отримання марки (рядок)
- метод зміни марки
- метод отримання вантажопідйомності (дійсне число)
- метод зміни вантажопідйомності
- метод отримання максимально можливого сумарного обсягу перевезеного вантажу (дійсне число)
- метод зміни максимально можливого сумарного обсягу перевезеного вантажу
- метод, який повертає масив контейнерів, що перевозяться транспортом
- метод, який приймає масив контейнерів, що перевозяться транспортом
- метод, який додає контейнер до загального вантажу (метод приймає посилання на додається контейнер)
- метод, що видаляє контейнер (метод приймає посилання на

видаляється контейнер)

- метод, який повертає сумарний обсяг контейнерів, що перевозяться транспортом, в заданих одиницях виміру (одиниця виміру передається як вхідний параметр методу, і має тип *VolumeUnitEnumeration*)
- метод, який повертає посилання на найбільш важкий контейнер

7. Створіть клас *CargoShip* вантажного судна. Клас повинен реалізовувати інтерфейс *CargoTransport*. Цей клас містить:

приватні поля:

- рег. номер
- марка
- максимальна вантажопідйомність
- максимальний обсяг вантажу, що перевозиться
- список вантажів, що перевозяться (екземпляр класу LinkedList <ContainerCargo>)

конструктори:

- конструктор може приймати рег. номер (інші поля приймають значення за умовчанням, які обираються студентом, список инициализируется порожнім - число елементів 0)
- конструктор може приймати реєстраційний номер, марку, вантажопідйомність (т), обсяг (куб. м) (список инициализируется порожнім - число елементів 0) методи:
- Методи, що реалізують інтерфейс *CargoTransport*

8. Змініть клас Truck - він повинен реалізувати інтерфейс CargoTransport. Замініть масив на клас ArrayList <ContainerCargo>. Додайте відповідні реалізації методів.

9. Змініть клас *TruckFleet*

- Перейменуйте його в CargoDeliveryBase
- Він повинен працювати з елементами типу CargoTransport (і може містити об'єкти як типу CargoShip, так і Truck).

Варіант № 4

1. Пакет - library.

2. Створіть клас *Printing* - деякого друкованого видання.

Клас містить наступні елементи:

- приватні поля, що містять назву і рік видання
- 2 конструктора: по-замовчуванню і з параметрами (назва, рік)
- гетери і сеттери назви і року видання
- метод отримання «віку» друкованого видання (для отримання інформації про поточну дату використовувати класу java.util.Calendar)

3. Створіть перерахування *GenreEnumeration* - літературного жанру. Можливі значення: постапокаліптика, кіберпанк, фентезі …. Жанри описувані перерахуванням вибираються студентом самостійно.

4. Створіть клас *Book*. Цей клас розширює (успадковує) клас Printing. Крім назви і року видання кожна книжка характеризується Прізвищем автора і жанром. Клас додає наступні елементи:

- приватні поля, що містять прізвище автора та жанр (екземпляр перерахування GenreEnumeration)
- конструктори: без параметрів і з параметрами (прізвище, назва, рік видання, жанр)
- гетери і сеттери полів

5. Створіть клас *Article* - статті в журналі (збірнику статей). Цей клас характеризується прізвищем автора та назвою.

Приватний поля:

- прізвище автора
- назва статті

конструктори:

- за замовчуванням

- з 2-ма параметрами (автор, назва) методи:
- гетери і сеттери для полів

6. Створіть клас *ScienceDigest* - збірки статей. Цей клас розширює (успадковує) клас Printing. Крім назви збірки і року видання, збірник характеризується списком статей (екземпляр класу ArrayList <Article>. Елементи класу:

- приватне поле - список статей (клас ArrayList <Article>)
- конструктор за замовчуванням і з параметрами (назва збірки, рік, список статей)
- методи:
- гетер і сетер длясписку статей,
- методи додавання статті в кінець списку (приймають в якості параметра посилання на статтю)
- метод видалення статті зі списку (метод приймає як параметр назва статті)
- метод пошуку статті під назвою автора. Повертає посилання на список статей ArrayList <Article> зазначеного автора.
- метод пошуку статті за назвою. Повертає посилання на статтю.

7. Створіть клас *FreeLibraryCard* - читацького квитка. Карта містить інформацію про читача - ім'я, прізвище, список друкованої продукції, які в даним момент у нього на руках (він їх читає - кеп). Клас містить наступні елементи:

Приватні поля:

- прізвище читача,
- ім'я читача,
- список видань onHandList (екземпляр класу LinkedList <Printing>), які читач ще читає (не повернув)

Конструктори:

- по-замовчуванню (ініціює порожній список)
- з параметрами: прізвище, ім'я, список відкриті методи:

- сеттери і гетери полів
- додавання видання в список onHandList (методи приймають посилання на об'єкт типу Printing)
- видалення видання зі списку onHandList (метод приймає назву видання)

8. Створіть інтерфейс *LibraryCard* - читацького квитка. Інтерфейс визначає засоби:

- гетери і сетер імені та прізвища читача
- додавання видання в список onHandList (методи приймають посилання на об'єкт типу Printing)
- видалення видання зі списку onHandList (метод приймає назву видання)

9. Клас *FreeLibrary Card* повинен реалізовувати інтерфейс *Library Card*

10. Створіть клас *VipLibraryCard*, який реалізує інтерфейс *LibraryCard*. Цей клас відрізняється від класу *FreeLibraryCard* тим, що він заснований на двох списках типу ArrayList <Printing>, а не LinkedList <Printing >.

11. Створіть клас *Library* - бібліотека. Бібліотека містить 2 реєстру: 1) реєстр друкованих видань і 2) реєстр читацьких квитків. Елементи класу Library:

Приватні поля:

- список друкованої продукції (LinkedList <Printing>).
- списокчитацькихквитків (ArrayList <Library Card>).

Конструктори:

- по-замовчуванню (створює списки нульової довжини)
- з параметрами (список друкованої продукції, список читацьких квитків); Відкриті методи:
- метод, який реалізує пошук в реєстрі видання за параметрами (назва, прізвище автора, рік видання - шукає як книги, так і статті)

- метод, який реалізує пошук в реєстрі картки читача за параметрами (прізвище та ім'я читача)

Вариант № 5

1. Пакет - bank.
2. Створіть клас *AccountNumberGenerator*. Цей клас містить:
   - статичну приватну змінну типу int, з початковим значенням 0;
   - відкритий статичний метод getNext () - який збільшує значення статичної змінної на 1 і повертає її нове значення;
   - відкритий статичний метод getCurrent (), який повертає значення статичної змінної;
   - відкритий метод reset (), що встановлює значення змінної в 0;

Цей клас потрібно використовувати для генерації номерів рахунків.

3. Створіть перерахування *Currency* - валюти. Перерахування містить елементи (валюти) - USD (долар США), EUR (євро), JOY (Йена), TRY (Ліра), AED (Дірхам), RUB (Рубль). Можете додати ще валют.
4. Клас *Account* зробіть абстрактним. Додайте наступні елементи: поля:
   - приватне поле, що містить комісію за обслуговування.
   - приватне поле, що містить валюту рахунку. конструктори:
   - конструктор, що приймає 3 параметра - номер рахунку, залишок на рахунку, розмір комісії за обслуговування рахунку (валюта - рубль)
   - конструктор, що приймає 4 параметра - номер рахунку, залишок на рахунку, розмір комісії за обслуговування рахунку, валюта

методи:

- частное поле, что містіть комісію за обслуговування.

- частное поле, что містіть валюту Рахунку.

конструктори:

- конструктор, что пріймає 3 параметра - номер Рахунку, Залишок на Рахунку, розмір КОМІСІЇ за обслуговування Рахунку (валюта - рубль)
- конструктор, что пріймає 4 параметра - номер Рахунку, Залишок на Рахунку, розмір КОМІСІЇ за обслуговування Рахунку, валюта

видалить метод:

- Зміни значення залишку коштів на рахунку (сеттер)

5. Створіть клас *DebitAccount*, дебетові карти. Цей клас розширює (успадковує) клас *Account*.

Цей клас додає свої методи і поля, і не перекриває методи і поля суперкласу. Клас визначає аналогічні суперкласу конструктори, в яких просто викликає відповідний конструктор суперкласу.

6. Створіть клас *CreditAccount*, кредитної карти. Цей клас розширює (успадковує) клас *Account*. Додайте наступні елементи:

поля:

- приватне поле - процентна ставка (річна, у відсотках)
- приватне поле - ліміт по кредитній карті
- приватне поле - нараховані відсотки
- приватне поле - нараховані комісійні Конструктори:
- аналогічні суперкласу конструктори, в яких просто викликає відповідний конструктор суперкласу, інші змінні = 0;
- конструктор, що приймає параметри: номер рахунку, залишок на рахунку, розмір комісії за обслуговування рахунку, валюта, процентна ставка, ліміт по карті (в заданій валюті), нараховані відсотки і комісійні = 0;

Методи:

- гетери і сеттери процентної ставки і ліміту по карті.
- гетери для нарахованих відсотків і комісійних за обслуговування.

- метод нарахування відсотків (якщо залишок менше ліміту по карті - збільшується сума нарахованих відсотків на величину = «(ліміт - залишок) * (процентна ставка / число днів в поточному році) / 100». Для визначення числа днів в поточному році використовувати клас Calendar.

Перевизначити методи:

- метод, що віднімає комісію з залишку - замість зменшення залишку на величину комісії, метод повинен збільшити «нараховані комісійні» на величину комісійних.
- метод поповнення рахунку - замість просто збільшення залишку, кошти спочатку йдуть на погашення нарахованих комісійних, потім нарахованих відсотків, потім на поповнення залишку.

7. Створіть інтерфейс *Client*. Інтерфейс визначає такі методи:
   - метод, який повертає посилання на рахунок по його унікальним номером;
   - метод, який повертає список (клас ArrayList <Account>) всіх рахунків;
   - метод, який повертає список (клас ArrayList <Account>) рахунків дебетових карт;
   - метод, який повертає список (клас ArrayList <Account>) рахунків кредитних карт;
   - метод, який повертає сумарний залишок на всіх дебетових рахунках;
   - метод, який повертає суму боргу клієнта (сума нарахованих відсотків і комісійних за всіма кредитними рахунками, а також негативний залишок по картах)
   - метод, який повертає список (клас ArrayList <Account>) рахунків з позитивним залишком на рахунку;
   - метод видалення рахунку по його номеру;
   - метод додавання рахунку (приймає в якості вхідного параметра

посилання на рахунок):

- метод списування коштів з рахунку (приймає номер рахунку і розмір суми);
- метод поповнення рахунку (приймає номер рахунку і розмір суми);

8. Змінитьклас *Client*, перейменуйте його вкласс в *NaturalClient*.Клас повинен реалізувати інтерфейс *Client*. Для цього:

- замість масиву рахунків використовуйте список рахунків (клас ArrayList <Account>).
- змініть конструктори
- реалізуйте методи інтерфейсу

полякасу - ім'я, прізвище, серія та номер паспорта, а так само відповідні гетери і сеттери залиште.

ЗАВДАННЯ 3

1. У вже створеному пакеті реалізуйте внутрішні класи і продемонструйте відповідно до варіанта.

Вариант № 0

1. Пакет - *policlinic*.

2. В поліклініці деякого міста почали використовувати новий апарат УЗІ, який характеризується кількістю часу у «стані роботи» та у «стані очікування». Для роботи апарату потрібне енергопостачання, причому у стані очікування його расходується на 35 відсотків менше, ніж під час роботи. Після експлуатації протягом місяцязав. лабораторією цікавить питання: скільки потрібно виділити коштів на оплату рахунку по енергопостачанню. Ціну за 1 кВт та кількість часу роботи УЗІ у двух станів студент обирає самостійно.

Вариант № 1

1. Пакет - *university*.

2. Для классу Studentзамінити поле «номер залікової книжки» на єкземпляр классу Record_book. У кожного студента є залікова книжка. В кінці семестру в ній проставляються оцінки за єкзамени, заліки та курсові роботи студента. Рейтінг студента напряму залежить від середього балу, який він отримав в кінці семестру та врахування участі у університетських заходах (спортивних змаганнях, олімпіадах та наукових семінарах).Прорахувати рейтінг для декількох студентів та обрати найуспішнішого з них.

Вариант № 2

1. Пакет - organization

У кожного підрозділу організації є своє приміщення на фіксовану кількіть робочих місць. Після відкриття філіалу, організація прийняла на роботу нових працівників у два підрозділа (бугалтерія та інформаціоний відділ). Керівництво підприємства цікавить питання: чи вистачить новим працівникам робочих міст і чи є потреба закласти у бютжет наступного року підготування нового приміщення на таку ж кількіть мість у разі повторного розширення штату цього року.

Вариант № 3

1. Пакет - *logistics*.

2. Для класу *Truch*додатиполе пробіг. Додати класс Engine (двигун), який характеризується потужністю та датой заміни мастила. Кожні

100000 км пробігу авто (або кожен рік – в залежності від того, який факт наступить першим) потрібна заміна мастила. Після повернення декількох вантажівок з рейсу трастпортного інстектора виробництва цікавить: хто з них потребує тех. огляду з урахуванням дати останнього. Пробіг кожної вантажівки студент обирає власноруч.

Вариант № 4

1. Пакет - Library

2. Змінити класс ScienceDigest (додати поле «максимальна кількість сторінок»), відтепер він працює з об єктом класу Article. Кожна стаття характеризується обємом. Потрібно підготувати новий номер журналу до видання. Для точної верстки видання потрібно підрахувати кількість публікацій, на які були подані заяви, та врахувати інформацію на правах реклами (розміри якої студент обираю власноруч). У разі перевищення допустимої кількості сторінок решту статей перенести до наступного номеру журналу, а залишок вільного містя заповнити рекламою.

Вариант № 5

1. Пакет - Bank

2. В банку деакого населеного пункту експлуатується банкомат, завдяки якому клієнт може отримати інформацію про стан рахунку та зняти гроші у декількох номіналах купюр. У разі недостатньої кількості купюр у банкоматі виникає потреба «заправити» банкомат. Після робочого дня та обслуговування клієнтів банкомату у адміністрації банку виникає питання: остаток коштів в банкоматі та які номінали купюр є найбільший попит у клієнтів.

ДОДАТОК 1

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ
Навчально-консультаційний центр заочної освіти

Контрольна робота № _____

по _____ варіант _____
(назва дисципліни)
студент _____ курсу, спеціальність _____
_____
(прізвище, ім'я, по батькові)

Студентський квиток № _____
Електрона адреса_____
_____

«___» _____ 20__ р.

П.І.Б. студента	Дата отримання завдання СРС	Дати виконання етапів КР по РП							П.І.Б. Підпис викладача
	НКЦ/кафедра/викладач/мережа Internet	Дати фактичного виконання							
1. Петров В.С.	25.05.2016г.	3.10	3.11	3.12	3.01	3.02	3.03	3.04	
	кафедра								

Дата реєстрування контрольної роботи в НКЦ _____ печать

Дата реєстрування контрольної роботи на кафедрі _____

Результати оцінювання контрольної роботи викладачем за шкалою ВНЗ, національною шкалою та шкалою ECTS

Зворотній бік: рецензія на самостійне завдання студента