Лабораторна робота №4

Інтерфейси, пакети та внутрішні класи в мові Java

Мета лабораторної роботи:	1. Вивчити основні концепції використання інтерфейсів у Java. Ознайомитись із методами використання внутрішніх класів. 2. Вивчити прийоми програмування з використанням інтерфейсів та внутрішніх класів на мові Java.

Зміст роботи.

1. Теоретичні відомості

Короткі теоретичні відомості про інтерфейси, пакети та внутрішні класи в Java можна переглянути тут

Завдання на лабораторну роботу

Завдання 1.

Реалізувати абстрактні класи або інтерфейси, а також успадкування та поліморфізм для класів згідно з варіантом.

Умову Завдання1 згідно з варіантом можна подивитися тут.

Завдання 2.

Реалізувати класи з використанням внутрішніх класів згідно з варіантом.

Умову Завдання 2 згідно з варіантом можна подивитися тут .

Для всіх завдань необхідно розробити тестову програму, де демонструється робота зі створеними класами та тестуються їх методи.

2. Звіт

Повинен містити: постановку завдання, програмний код розв'язання, результат роботи написаної програми

ІНТЕРФЕЙСИ І ВНУТРІШНІ КЛАСИ

Інтерфейси

Інтерфейси подібні до абсолютно абстрактних класів, але не є класами. Жоден з оголошених методів може бути реалізований всередині інтерфейсу. У мові Java існують два види інтерфейсів: інтерфейси, що визначають контракт для класів за допомогою методів, та інтерфейси, реалізація яких автоматично (без реалізації методів) надає класу певних властивостей. До останніх відносяться, наприклад, інтерфейси Cloneable та Serializable, які відповідають за клонування та збереження об'єкта в інформаційному потоці відповідно.

Усі оголошені в інтерфейсі методи автоматично трактуються як public і abstract, проте поля – як public, static і final, навіть якщо вони не оголошені. Клас може реалізовувати будь-яке число інтерфейсів, що вказуються через кому після ключового слова implements, що доповнює визначення класу. Після цього клас зобов'язаний реалізувати всі методи, отримані від інтерфейсів, або оголосити себе абстрактним класом.

На безлічі інтерфейсів також визначено ієрархію спадкування, але вона не має відношення до ієрархії класів.

Визначення інтерфейсу має вигляд:

```
[public] interface Ім'я [extends Ім'я1, Ім'я2,…, Ім'яN] {
                                        /*реалізація інтерфейсу*/}
```

Наприклад:

```
/* Приклад: оголошення інтерфейсів: LineGroup.java, Shape.java */
package chapt06;

public interface LineGroup {
      // за умовчанням public abstract
      double getPerimeter();// оголошення методу
}

package chapt06;

public interface Shape extends LineGroup {
     // int id; // помилка, якщо немає ініціалізації
     //void method(){} /* помилка, оскільки абстрактний метод
                    неспроможна мати тіла! */
      double getSquare(); // оголошення методу
}
```

Для більш простої ідентифікації інтерфейсів у великому проекті у спільноті розробників діє негласна угода про додавання до імені інтерфейсу символу 'I', відповідно до якої замість імені **Shape** можна записати IShape.

Клас, який реалізовуватиме інтерфейс Shape, повинен визначити всі методи з ланцюжка успадкування інтерфейсів. В даному випадку це методи getPerimeter() та getSquare().

Інтерфейси зазвичай оголошуються як public, тому що опис функціональності, що надається ними, може бути використаний у декількох пакетах проекту. Інтерфейси з областю видимості в рамках пакета можуть використовуватися тільки в цьому пакеті і більше.

У мові Java інтерфейси забезпечують більшу частину тієї функціональності, що у C++ представляється з допомогою механізму множинного успадкування. Клас може успадковувати один суперклас і реалізовувати довільну кількість інтерфейсів.

Реалізація інтерфейсів класом може мати вигляд:

```
[доступ] class Ім'яКласа implements Ім'я1, Ім'я2,…, Ім'яN {
                                        /*код класу*/}
```

Тут Имя1, Имя2,…, ИмяN – список використовуваних інтерфейсів. Клас, який реалізує інтерфейс, має надати повну реалізацію всіх методів, оголошених в інтерфейсі. Крім цього, цей клас може оголошувати власні методи. Якщо клас розширює інтерфейс, але повністю не реалізує його методи, цей клас має бути оголошений як abstract.

```
/* Приклад: реалізація інтерфейсу: Rectangle.java */
package chapt06;

public class Rectangle implements Shape {
      private double a, b;

      public Rectangle(double a, double b) {
            this.a = a;
            this.b = b;
      }
      //Реалізація методу з інтерфейсу
      public double getSquare() {//площа прямокутника
```

```
            return a * b;
      }
      //Реалізація методу з інтерфейсу
       public doublegetPerimeter() {
              return2*(a+b);
      }
}
```

/ Приклад: реалізація інтерфейсу: Circle.java */*

```
packagechapt06;

public classCircle implements Shape {
      private double r;

      publicCircle(double r) {
            this.r = r;
      }
      public doublegetSquare() {//площа кола
            returnMath.PI * Math.pow (r, 2);
      }
      public doublegetPerimeter() {
            return2 * Math.PI * r;
      }
}
```

/ Приклад: неповна реалізація інтерфейсу: Triangle.java */*

```
packagechapt06;
```

/ метод getSquare() у даному абстрактному класі не реалізований */*

```
public abstract classTriangle implements Shape {
      private doublea, b, c;

      publicTriangle(double a, double b, double c) {
            this.a = a;
            this.b = b;
            this.c = c;
      }
      public doublegetPerimeter() {
            returna + b + c;
      }
}
```

/ приклад: властивості посилання на інтерфейс: Runner.java */*

```
packagechapt06;

public classRunner {
      public static voidprintFeatures(Shape f) {
         System.out.printf("площа:%.2f периметр: %.2f%n",
                  f.getSquare(), f.getPerimeter());
      }
      public static voidmain(String[] args) {
            Rectangle r = new Rectangle(5, 9.95);
            Circle c = New Circle (7.01);
            printFeatures(r);
            printFeatures(c);
      }
}
```

В результаті буде виведено:

```
площа: 49,75 периметр: 29,90
площа: 154,38 периметр: 44,05
```

Клас Runner містить метод printFeatures(Shape f), який викликає методи об'єкта, що передається йому як параметр. Спочатку йому передається об'єкт, відповідний прямокутнику, потім колу (об'єкти з r). Як метод printFeatures() може обробляти об'єкти двох різних класів? Вся справа в типі аргументу, що передається цьому методу, – класу, що реалізує інтерфейс Shape. Викликати, однак, можна лише ті методи, які було оголошено в інтерфейсі.

У прикладі в класі ShapeCreator використовуються класи та інтерфейси, визначені вище, і оголошується посилання на інтерфейсний тип. Таке посилання може вказувати на екземпляр будь-якого класу, який реалізує оголошений інтерфейс. При виклику методу через таке посилання буде викликатись його реалізована версія, заснована на поточному екземплярі класу. Виконуваний метод розшукується динамічно під час виконання, що

дозволяє створювати класи пізніше за код, який викликає їх методи.

```
/* приклад # 6: динамічний зв'язування методів: ShapeCreator.java */
package chapt06;

public class ShapeCreator {
      public static void main(String[] args) {
            Shape sh; /* Посилання на інтерфейсний
            тип */ Rectangle re = new Rectangle(5,
            9.95); sh = re;
            sh.getPerimeter();//виклик методу класу Rectangle
            Circle cr = New Circle (7.01);
            sh = cr; // Надається посилання на інший об'єкт
            sh.getPerimeter();//виклик методу класу Circle

            // cr = re; // Помилка! різні гілки успадкування
      }
}
```

Неможливо прирівнювати посилання на класи, що знаходяться в різних гілках успадкування, так як не існує є ніякого способу привести один такий тип до іншого. З цієї причини помилку викличе спроба оголошення об'єкта як:

```
Circle c = new Rectangle(1, 5);
```

Пакети

Будь-який клас Java відноситься до певного пакета, який може бути неіменованим (unnamed або de-fault package), якщо оператор package відсутній. Оператор package ім'я, поміщається на початку вихідного програмного файла, визначає іменований пакет, тобто. область у просторі імен класів, де визначаються імена класів, які у цьому файлі. Дія оператора package вказує на місце розташування файлу щодо кореневого каталогу проекту. Наприклад:

```
package chapt06;
```

При цьому програмний файл буде поміщений у підкаталог під назвою chapt06. Ім'я пакета при зверненні до класу з іншого пакета приєднується до класу: chapt06.Student. Усередині зазначеної області можна виділити підобласті:

```
package chapt06.bsu;
```

Загальна форма файлу, що містить вихідний код Java, може бути така:

```
одиночний оператор package (необов'язковий);
будь-яку кількість операторів import
(необов'язкові); одиночний відкритий (public) клас
(необов'язковий) будь-яку кількість класів пакета
(необов'язкові)
```

У реальних проектах пакети часто називаються так:

- зворотний інтернет-адреса виробника програмного забезпечення,а саме для`www.bsu.byвийде by.bsu;`
- далі слідує ім'я проекту, наприклад: eun;
- потім розташовуються пакети, що визначають власне додаток.

При використанні класів перед ім'ям класу через точку треба додавати повне ім'я пакета, до якого належить цей клас. На малюнку наведено не повний список пакетів реального докладання. З назв пакетів можна визначити, які приблизно класи в ньому розташовані, не заглядаючи всередину. При створенні пакета завжди слід керуватися простим правилом: називати його ім'ям простим, але таким, що відображає сенс, логіку поведінки і функціональність об'єднаних у ньому класів.

by.bsu.eun
by.bsu.eun.administration.constants
by.bsu.eun.administration.dbhelpers
by.bsu.eun.common.constants
by.bsu.eun.common.dbhelpers.annboard
by.bsu.eun.common.dbhelpers.courses
by.bsu.eun.common.dbhelpers.guestbook
by.bsu.eun.common.dbhelpers.learnres
by.bsu.eun.common.dbhelpers.messages
by.bsu.eun.common.dbhelpers.news
by.bsu.eun.common.dbhelpers.prepinfo
by.bsu.eun.common.dbhelpers.statistics
by.bsu.eun.common.dbhelpers.subjectmark
by.bsu.eun.common.dbhelpers.subjects
by.bsu.eun.common.dbhelpers.test
by.bsu.eun.common.dbhelpers.users
by.bsu.eun.common.menus
by.bsu.eun.common.objects
by.bsu.eun.common.servlets
by.bsu.eun.common.tools
by.bsu.eun.consultation.constants
by.bsu.eun.consultation.dbhelpers
by.bsu.eun.consultation.objects
by.bsu.eun.core.constants
by.bsu.eun.core.dbhelpers
by.bsu.eun.core.exceptions
by.bsu.eun.core.filters
by.bsu.eun.core.managers
by.bsu.eun.core.taglibs

РисОрганізація пакетів програми

Кожен клас додається до зазначеного пакета при компіляції. Наприклад:

```
// приклад: найпростіший клас у пакеті: CommonObject.java
package by.bsu.eun.objects;

public class CommonObject implements Cloneable {
      public CommonObject() {
            super();
      }
      public Object clone()
            throws CloneNotSupportedException {
            return super.clone();
      }
}
```

Клас починається з вказівки, що він належить пакету by.bsu.eun.objects. Іншими словами, це означає, що файл CommonObject.java знаходиться в каталозі objects, який, у свою чергу, знаходиться в каталозі bsu і так далі. Не можна перейменовувати пакет, не перейменувавши каталог, у якому зберігаються його класи. Щоб отримати доступ до класу з іншого пакета, перед ім'ям такого класу вказується ім'я пакета: by.bsu.eun.objects.CommonObject. Щоб уникнути таких довгих імен, використовується ключове слово import. Наприклад:

```
import by.bsu.eun.objects.CommonObject;
```

або

```
import by.bsu.eun.objects.*;
```

У другому варіанті імпортується весь пакет, що означає можливість доступу до будь-якого класу пакета, але не до підпакета та його класів. У практичному програмуванні слід використовувати індивідуальний

importкласу, щоб при аналізі коду була можливість швидко визначити місцезнаходження використовуваного класу.

Доступ до класу з іншого пакета можна здійснити так:

```
// приклад: доступ до пакету: UserStatistic.java
package by.bsu.eun.usermng;

public class UserStatistic
                  extends by.bsu.eun.objects.CommonObject {
      private long id;
      private int mark;

      public UserStatistic() {
            super();
      }
      public long getId() {
            return id;
      }
      public void setId(long id) {
            this. id = id;
      }
      public int getMark() {
            return mark;
      }
      public void setMark(int mark) {
            this. mark = mark;
      }
}
```

При імпорті класу з іншого пакета рекомендується завжди вказувати повний шлях із зазначенням імені імпортованого класу. Це дозволяє у великому проекті легко знайти визначення класу, якщо виникає необхідність переглянути вихідний код класу.

```
// приклад # 9: доступ до пакету: CreatorStatistic.java
package by.bsu.eun.actions;
import by.bsu.eun.objects.CommonObject;
import by.bsu.eun.usermng.UserStatistic;

public class CreatorStatistic {
   public static UserStatistic createUserStatistic(long id) {
            UserStatistic temp = new UserStatistic();
            temp.setId(id);
            // читання інформації з бази даних з id
            користувача int mark = набуте значення;
            temp.setMark(mark);
            return temp;
      }
      public static void main(String[] args) {
            UserStatistic us = createUserStatistic(71);
            System.out.println(us.getMark());
      }
}
```

Якщо пакет не існує, його необхідно створити до першої компіляції, якщо пакет не вказано, клас добавиться в пакет без імені (unnamed). При цьому unnamed-каталог не створюється. Однак у реальних проектах класи поза пакетами не створюються, і немає причин відступати від цього правила.

Статичний імпорт

Константи та статичні методи класу можна використовувати без вказівки приналежності до класу, якщо застосувати статичний імпорт, як показано в наступному прикладі.

```
// Приклад: статичний імпорт: ImportDemo.java
package chapt06;
import static java.lang.Math.*;


public class ImportDemo {
     public static void main(String[] args) {
```

```
        double radius = 3;
        System.out.println(2 * PI * radius);
        System.out.println(floor(cos(PI/3)));
    }
}
```

Якщо необхідно отримати доступ лише до однієї константи класу чи інтерфейсу, наприклад Math.E, то статичний імпорт виробляється у такому вигляді:

```
import staticjava.lang.Math.E;
import staticjava.lang.Math.cos;//для одного методу
```

Внутрішні класи

Класи можуть взаємодіяти друг з одним як у вигляді успадкування і використання посилань, а й у вигляді організації логічної структури з визначенням одного класу у тілі іншого.

У Java можна визначити (вкласти) один клас усередині визначення іншого класу, що дозволяє групувати класи, логічно пов'язані один з одним, та динамічно керувати доступом до них. З одного боку, обґрунтоване використання в коді внутрішніх класів робить його ефектнішим і зрозумілішим. З іншого боку, застосування внутрішніх класів є одним із способів приховування коду, оскільки внутрішній клас може бути абсолютно недоступний і не видно поза класом-власником. Внутрішні класи також використовуються як блоки прослуховування подій (глава «Події»). Однією з найважливіших причин використання внутрішніх класів є можливість незалежного наслідування внутрішніми класами. Фактично при цьому реалізується множинне успадкування зі своїми перевагами та проблемами.

Як приклади можна розглянути взаємозв'язки класів «Корабель», «Двигун» та «Шлюпка». Об'єкт класу «Двигун» розташований усередині (невидимий ззовні) об'єкта «Корабель» і його діяльність наводить «Корабель» у рух. Обидва ці об'єкти нерозривно пов'язані, тобто запустити «Двигун» можна лише за допомогою об'єкта «Корабль», наприклад, з машинного відділення. Таким чином, перед ініціалізацією об'єкта внутрішнього класу «Двигун» має бути створений об'єкт зовнішнього класу «Корабль».

Клас «Шлюпка» також є логічною частиною класу «Корабель», проте ситуація з його об'єктами простіше через те, що ці об'єкти можуть бути використані незалежно від наявності об'єкта «Корабель». Об'єкт класу «Шлюпка» використовує тільки ім'я (на борту) свого зовнішнього класу. Такий внутрішній клас слід визначати як static. Якщо об'єкт «Шлюпка» використовується без прив'язки до судна, то клас слід визначати як звичайний незалежний клас.

Вкладені класи можуть бути статичними, оголошеними модифікатором static, і нестатичними. Статичні класи можуть звертатися до членів класу, що включає, не безпосередньо, а тільки через його об'єкт. Нестатичні внутрішні класи мають доступ до всіх змінних та методів свого зовнішнього класу-власника.

Внутрішні (inner) класи

Нестатичні вкладені класи називають внутрішніми (inner) класами. Доступ до елементів внутрішнього класу можливий із зовнішнього класу лише через об'єкт внутрішнього класу, який має бути створений у коді методу зовнішнього класу. Об'єкт внутрішнього класу завжди асоціюється (приховано зберігає посилання) з об'єктом зовнішнього класу, що його створив, - так званим зовнішнім (enclosing) об'єктом. Зовнішній та внутрішній класи можуть виглядати, наприклад, так:

```
public classShip {
    // поля та конструктори
    // abstract, final, private, protected - допустимі
    public classEngine { // Визначення внутрішнього класу
        // поля та методи
        public voidlaunch() {
            System.out.println("Запуск двигуна");
        }
    }// кінець оголошення внутрішнього класу
    public voidinit() {// метод зовнішнього класу
      // оголошення об'єкта внутрішнього
        класуEngine eng = new
        Engine(); eng.launch();
    }
}
```

При такому оголошенні об'єкта внутрішнього класу Engine у методі зовнішнього класу Ship немає реальної відмінності від використання будь-якого іншого зовнішнього класу, крім оголошення усередині класу Ship. Використання об'єкта внутрішнього класу поза своїм зовнішнім класом можливе лише за наявності доступу (видимості) та при оголошенні посилання у вигляді:

```
Ship.Engine obj = new Ship().new Engine();
```

Основна відмінність від зовнішнього класу полягає у великих можливостях обмеження видимості внутрішнього класу порівняно із звичайним зовнішнім класом. Внутрішній клас може бути оголошений як private,

що забезпечує його повну невидимість поза класом-власником та надійне приховування реалізації. У цьому випадку посилання obj, наведене вище, оголосити було б не можна. Створити об'єкт такого класу можна лише у методах та логічних блоках зовнішнього класу. Використання protected дозволяє отримати доступ до внутрішнього класу для класу в іншому пакеті, що є суперкласом зовнішнього класу.

Після компіляції об'єктний модуль, який відповідає внутрішньому класу, отримає ім'я **Ship$Engine.class.**

Методи внутрішнього класу мають прямий доступ до всіх полів та методів зовнішнього класу, водночас зовнішній клас може отримати доступ до вмісту внутрішнього класу лише після створення об'єкта внутрішнього класу. Внутрішні класи що неспроможні містити статичні атрибути і методи, крім констант (final static). Внутрішні класи мають право успадковувати інші класи, реалізовувати інтерфейси та у ролі об'єктів успадкування. Допустимо успадкування наступного виду:

```
public classWarShip extends Ship {
     protected classSpecialEngine extends Engine {}
}
```

Якщо внутрішній клас успадковується зазвичай іншим класом (після extends вказується **Ім'яЗовнішньогоКласу.Ім'яВнутрішньогоКласу**), то він втрачає доступ до полів свого зовнішнього класу, в якому він був оголошений.

```
public classMotor extends Ship.Engine {
     publicMotor(Ship obj) {
          obj.super();
     }
}
```

У даному випадку конструктор класу Motor має бути оголошений з параметром типу Ship, що дозволить по-отримати доступ до посилання на внутрішній клас Engine, успадкований класом Motor.

Внутрішні класи дозволяють остаточно вирішити проблему множинного успадкування, коли потрібно успадковувати властивості кількох класів.

При оголошенні внутрішнього класу можна використовувати модифікатори final, abstract, private, protected, public.

Простий приклад практичного застосування взаємодії класу-власника та внутрішнього нестатичного класу проілюстровано на наступному прикладі.

```
/* приклад: взаємодія зовнішнього та внутрішнього класів : Student.java : AnySession.java */
packagechapt06;

public classStudent {
     private int id;
     privateExamResult[] exams;

     publicStudent(int id) {
          this. id = id;
     }

     private classExamResult {// внутрішній клас
          privateString name;
          private int mark;
          private boolean passed;

          publicExamResult(String name) {
               this. name = name;
               passed = false;
          }
          public voidpassExam() {
               passed = true;
```

```
        }
        public voidsetMark(int mark) {
             this. mark = mark;
        }
        public intgetMark() {
             returnmark;
        }
        public intgetPassedMark() {
             final intPASSED_MARK = 4; // «Чарівне» число
             returnPASSED_MARK;
        }
        publicString getName() {
             returnname;
        }
        public booleanisPassed() {
             returnpassed;
        }
    } // Закінчення внутрішнього класу

public voidsetExams(String[] name, int[] marks) {
            if(name.length! = marks.length)
                  throw newIlegallegalArgumentException();
            exams = new ExamResult[name.length];
            for(int i = 0; i < name.length; i++) {
                  exams[i] = new ExamResult(name[i]);
                  exams[i].setMark(marks[i]);
      if(exams[i].getMark() >= exams[i].getPassedMark())
                        exams[i].passExam();
            }
publicString toString() {
            String res = "Студент:" + id + "\n";
      for(int i = 0; i < exams.length; i++)
            if(exams[i].isPassed())
            res += exams[i].getName() + "здав \n";
      else
            res += exams[i].getName() + "не здав \n";

      returnres;
      }
}
packagechapt06;

public classAnySession {
    public static voidmain(String[] args) {
         Student stud = new Student(822201);
         String ex[] = {"Механіка", "Програмування"};
         intmarks[] = {2, 9};
         stud.setExams(ex, marks);
         System.out.println(stud);
    }
}
```

В результаті буде виведено:

```
Студент: 822201
Механіка не здав
Програмування здав
```

Внутрішній клас визначає сутність предметної області “результат іспиту” (клас ExamResult), яка зазвичай безпосередньо пов'язана в інформаційній системі з об'єктом класу Student. Клас ExamResult в даному випадку визначає лише методи доступу до своїх атрибутів і зовсім невидимий поза класом Stu-

dent, який включає методи створення та ініціалізації масиву об'єктів внутрішнього класу з будь-якою кількістю іспитів, який однозначно ідентифікує поточну успішність студента.

Внутрішній клас може бути оголошений усередині методу або логічного блоку зовнішнього класу. Видимість такого класу регулюється областю видимості блоку, де він оголошений. Але внутрішній клас зберігає доступ до всіх полів і методів зовнішнього класу, а також усіх констант, оголошених у поточному блоці коду. Клас, оголошений усередині методу, не може бути оголошений як static, а також не може містити статичні поля та методи.

*/*приклад: внутрішній клас, оголошений усередині методу: TeacherLogic.java*/*

```
packagechapt06;

public abstract classAbstractTeacher {
      private int id;
      publicAbstractTeacher(int id) {
            this. id = id;
      }
      public abstract booleanexcludeStudent(String name);
}
packagechapt06;

public classTeacher extends AbstractTeacher {

      publicTeacher(int id) {
            super(id);
      }
      public booleanexcludeStudent(String name) {
            return false;
      }
}
packagechapt06;

public classTeacherCreator {
      publicTeacherCreator(){}

      publicAbstractTeacher createTeacher(int id) {
            // Оголошення класу всередині методу
         classDean extends AbstractTeacher {
                  Dean(int id){
                        super(id);
                  }
            public booleanexcludeStudent(String name) {
                        if(name! = null) {
                        // Зміна статусу студента в базі даних
                              return true;
                        }
                        else return false;
                  }
            } // кінець внутрішнього класу

            if(isDeanId(id))
                  return newDean(id);
            else return newTeacher(id);
      }
      private static booleanisDeanId(int id) {
            // Перевірка декана з БД або
            return(id == 777);
      }
}
packagechapt06;

public classTeacherLogic {
      public static voidexcludeProcess(int deanId,
                  String name) {

            AbstractTeacher teacher =
            newTeacherCreator().createTeacher(deanId);
```

```
            System.out.println("Студент: " + name
            + "відрахований:" + teacher.excludeStudent(name));
      }
      public static voidmain(String[] args) {
            excludeProcess(700, "Балаганов");
            виключитипроцес(777, "Балаганів");
      }
}
```

В результаті буде виведено:

```
Студент: Балаганов відрахований: false
Студент: Балаганов відрахований: true
```

Клас Dean оголошений у методі createTeacher(int id), і відповідно об'єкти цього класу можна створювати тільки всередині цього методу, з іншого місця зовнішнього класу внутрішній клас недоступний. Однак існує можливість отримати посилання на клас, оголошений усередині методу, та використовувати його специфічні властивості. При компіляції цього коду з внутрішнім класом асоціюється об'єктний модуль складним ім'ям **TeacherCreator$1Dean**, проте однозначно визначальним зв'язок між зовнішнім та внутрішнім класами. Цифра 1 в імені говорить про те, що в інших методах класу можуть бути оголошені внутрішні класи з таким самим ім'ям.

Вкладені (nested) класи

Якщо немає потреби у зв'язку об'єкта внутрішнього класу з об'єктом зовнішнього класу, тобто сенс зробити такий клас статичним.

Вкладений клас логічно пов'язаний із класом-власником, але може бути використаний незалежно від нього.

При оголошенні такого внутрішнього класу є службове слово static, і такий клас називається вкладеним (nested). Якщо клас вкладено в інтерфейс, він стає статичним за умовчанням. Такий клас здатний успадковувати інші класи, реалізовувати інтерфейси і бути об'єктом успадкування для будь-якого класу, який має необхідні права доступу. У той же час, статичний вкладений клас для доступу до нестатичних членам і методам зовнішнього класу має створювати об'єкт зовнішнього класу, а безпосередньо має доступ лише до статичних полів та методів зовнішнього класу. Для створення об'єкта вкладеного класу об'єкт зовнішнього класу створювати не потрібно. Підклас вкладеного класу неспроможний успадкувати можливість доступу до членам зовнішнього класу, якими наділений його суперклас.

/ приклад # 13: вкладений клас: Ship.java: RunnerShip.java */*

```
packagechapt06;

public classShip {
      private int id;
      // abstract, final, private, protected - допустимі
      public static classLifeBoat {
            public static voiddown() {
                  System.out.println("шлюпки на воду!");
            }
            public voidswim() {
                  System.out.println("відплиття шлюпки");
            }
      }
}
packagechapt06;

public classRunnerShip {
      public static voidmain(String[] args) {
            // Виклик статичного методу
            Ship.LifeBoat.down();
            // Створення об'єкта статичного класу
            Ship.LifeBoat lf = New Ship.LifeBoat();
            // Виклик звичайного методу

            lf.swim();
      }
}
```

Статичний метод вкладеного класу викликається за умови повного відносного шляху до нього. Об'єкт lf вкладеного класу створюється з використанням імені зовнішнього класу без виклику його конструктора.

Клас, вкладений у інтерфейс, за умовчанням статичний. На нього не накладається жодних особливих обмежень, і він може містити поля та методи як статичні, так і нестатичні.

/ Приклад # 14 : клас вкладений в інтерфейс: Faculty.java : University.java */*

```
packagechapt06;
```

```
public interfaceUniversity {
      intNUMBER_FACULTY = 20;

      classLearningDepartment {// static за замовчуванням
            public intidChief;

            public static voidassignPlan(int idFaculty) {
                  // Реалізація
            }
            public voidacceptProgram() {
                  // Реалізація
            }
      }
}
```

Такий внутрішній клас використовує простір імен інтерфейсу.

Анонімні (anonymous) класи

Анонімні (безіменні) класи застосовуються для надання унікальної функціональності окремо взятому об'єкту для обробки подій, реалізації блоків прослуховування тощо. Можна оголосити анонімний клас, який розширюватиме інший клас або реалізовуватиме інтерфейс при оголошенні одного, єдиного об'єкта, коли решті об'єктів цього класу відповідатиме реалізація методу, визначена в самому класі. Оголошення анонімного класу виконується одночасно зі створенням його об'єкта за допомогою оператора new.

Анонімні класи ефективно використовуються, як правило, для реалізації (перевизначення) кількох методів та створення власних методів об'єкта. Цей прийом ефективний у разі, коли необхідне перевизначення методу, але створювати новий клас немає необхідності через вузьку область (або одноразове) застосування методу.

Конструктори анонімних класів не можна визначати та перевизначати. Анонімні класи допускають вологість один одного, що може сильно заплутати код і зробити ці конструкції незрозумілими.

/ приклад # 15: анонімні класи: TypeQuest.java: RunnerAnonym.java */*

```
packagechapt06;

public classTypeQuest {
      private intid = 1;

      publicTypeQuest() {
      }
      publicTypeQuest(int id) {
            this. id = id;
      }
      public voidaddNewType() {
            // Реалізація
            System.out.println(
            "Додано питання на відповідність");
      }
}
packagechapt06;

public classRunnerAnonym {
      public static voidmain(String[] args) {

       TypeQuest unique = new TypeQuest() {// анонімний клас #1
                  public voidaddNewType() {
                        // Нова реалізація методу
                        System.out.println(
                  "додане питання з вільною відповіддю");
                  }
            }; // кінець оголошення анонімного класу
            unique.addNewType();

            newTypeQuest(71) {// анонімний клас #2
                  privateString name = "Drag&Drop";

                  public voidaddNewType() {
                        // Нова реалізація методу #2
            System.out.println("доданий" + getName());
                  }
```

```
                String getName() {
                      returnname;
                }
          }.addNewType();

          TypeQuest standard = новий TypeQuest(35);
          standard.addNewType();
    }
}
```

В результаті буде виведено:

```
додано питання з вільною відповіддю
додано Drag&Drop
додано питання на відповідність
```

При запуску програми відбувається оголошення об'єкта unique із застосуванням анонімного класу, в якому перевизначається метод addNewType(). Виклик цього методу на об'єкті unique призводить до виклику версії методу з анонімного класу, який компілюється в об'єктний модуль з ім'ям RunnerAnonym$1. Процес створення другого об'єкта з анонімним типом застосовується у програмуванні значно частіше, особливо при реалізації класів-адаптерів та реалізації інтерфейсів у блоках прослуховування. У цьому ж оголошенні продемонстровано можливість оголошення в анонімному класі полів та методів, які доступні об'єкту поза цим класом.

Для перерахування оголошення анонімного внутрішнього класу виглядає дещо інакше, оскільки ініціалізація всіх елементів відбувається при першому зверненні до типу. Тому і анонімний клас реалізується лише всередині оголошення типу enum, як це зроблено в прикладі.

/ приклад # 16: анонімний клас у перерахуванні: EnumRunner.java */*

```
packagechapt06;

enumShape {
    RECTANGLE, SQUARE,
    TRIANGLE{// анонімний клас
          public doublegetSquare() {// версія для TRIANGLE
                returna*b/2;
          }
    };
    public doublea, b;

    public voidsetShape(double a, double b){
          if((a<=0 || b<=0) || a!=b && this==SQUARE)
                throw new IllegalArgumentException();
          else
                this.a = a;
                this.b = b;
    }
    public doublegetSquare(){// версія для RECTANGLE та SQUARE returna * b;
    }
    publicString getParameters() {
          return"a="+a+", b="+b;

    }
}
public classEnumRunner {
    public static voidmain(String[] args) {
          inti = 4;
          for(Shape f : Shape.values()) {
                f.setShape(3, i--);
    System.out.println(f.name()+"-> " + f.getParameters()
                      + "площа =" + f.getSquare());
          }
    }
}
```

В результаті буде виведено:

```
RECTANGLE-> a=3.0, b=4.0 площа= 12.0
SQUARE-> a=3.0, b=3.0 площа= 9.0
TRIANGLE-> a=3.0, b=2.0 площа= 3.0
```

Об'єктний модуль для такого анонімного класу буде скомпілюванийз ім'ям Shape $1.

Завдання1

Реалізувати абстрактні класи та інтерфейси, а також успадкування та поліморфізм для наступних класів:

1. **interface Абітурієнт←abstract class Студент←class Студент-Заочник**.
2. **interface Співробітник←class Інженер←class Керівник**.
3. **interface Будівля← abstract class Громадська Будівля← class Театр**.
4. **interface Mobile← abstract class Siemens Mobile← class Model**.
5. **interface Корабель ← abstract class Військовий Корабель ← class Авіаносець**.
6. **interface Лікар← class Хірург← class Нейрохірург**.
7. **interface Корабель← class Вантажний Корабель← class Танкер**.
8. **interface Меблі← abstract class Шафа← class Книжкова Шафа**.
9. **interface Фільм← class Вітчизняний Фільм← class Комедія**.
10. **interface Тканина← abstract class Одяг← class Костюм**.
11. **interface Техніка← abstract class Плеєр← class Відеоплеєр**.
12. **interface Транспортне Засіб ← abstract class Громадський Транспорт← class Трамвай**.
13. **interface Пристрій Друку← class Принтер← class Лазерний Принтер**.
14. **interface Папір ← abstract class Зошит ← class Зошит Для Малювання**.
15. **interface Джерело Світла← class Лампа← class Настільна Лампа**.
16. **interface Лікар← class Хірург← class Нейрохірург**.
17. **interface Корабель← class Вантажний Корабель← class Танкер**.
18. **interface Меблі← abstract class Шафа← class Книжкова Шафа**.
19. **interface Фільм← class Вітчизняний Фільм← class Комедія**.
20. **interface Тканина← abstract class Одяг← class Костюм**.
21. **interface Техніка← abstract class Плеєр← class Відеоплеєр**

Завдання 2

1. Створити клас **Notepad** з внутрішнім класом або класами, за допомогою об'єктів якого можуть зберігатися кілька записів на одну дату.

2. Створити клас **Payment** (купівля) із внутрішнім класом, за допомогою об'єктів якого можна сформувати покупку з кількох товарів.

3. Створити клас **Account** (рахунок) із внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про всі операції з рахунком (зняття, платежі, надходження).

4. Створити клас **Залікова Книга** з внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про сесії, заліки, іспити.

5. Створити клас **Department** (відділ фірми) з внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про всі посади відділу та всіх співробітників, які коли-небудь займали конкретну посаду.

6. Створити клас **Catalog** (каталог) із внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про історію видач книги читачам.

7. Створити клас **City** (місто) з внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про проспекти, вулиці, площі.

8. Створити клас **CD** (mp3-диск) з внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про каталоги, підкаталоги та записи.

9. Створити клас **Mobile** з внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про моделі телефонів та їх властивості.

10. Створити клас **Художня Виставка** з внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про картини, авторів та час проведення виставок.

11. Створити клас **Календар** із внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про вихідні та святкові дні.

12. Створити клас **Shop** (магазин) з внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про відділи, товари та послуги.

13. Створити клас **Довідкова Служба Громадського Транспорту** з внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про час, лінії маршрутів та вартість проїзду.

14. Створити клас **Computer** (комп'ютер) з внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про операційну систему, процесор та оперативну пам'ять.

15. Створити клас **Park** (парк) з внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про атракціони, час їх роботи та вартість.

16. Створити клас **Cinema** (кіно) з внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про адреси кінотеатрів, фільми та час сеансів.

17. Створити клас **Програма Передач** із внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про назву телеканалів та програм.

18. Створити клас **Фільм** з внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про тривалість, жанр і режисерів фільму.

19. Створити клас **Залікова Книга** з внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про сесії, заліки, іспити.

20. Створити клас **Department** (відділ фірми) з внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про всі посади відділу та всіх співробітників, які коли-небудь займали конкретну посаду.

21. Створити клас **Catalog** (каталог) із внутрішнім класом, за допомогою об'єктів якого можна зберігати інформацію про історію видач книги читачам.