

Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Кафедра «Електронних обчислювальних машин»



Звіт
з лабораторної роботи № 3
з дисципліни: «Кросплатформенні засоби програмування»
на тему: «СПАДКУВАННЯ ТА ІНТЕРФЕЙСИ»

Виконав:

студент групи КІ-306

Хмільовський С. Р.

Прийняв:

доцент кафедри ЕОМ

Іванов Ю. С.

Мета роботи: ознайомитися з спадкуванням та інтерфейсами у мові Java.

Завдання (варіант № 22)

1. Написати та налагодити програму на мові Java, що розширює клас, що реалізований у лабораторній роботі No2, для реалізації предметної області заданої варіантом (**Автомат → Штурмова гвинтівка**).
Суперклас, що реалізований у лабораторній роботі No2, зробити абстрактним. Розроблений підклас має забезпечувати механізми свого коректного функціонування та реалізовувати мінімум один інтерфейс. Програма має розміщуватися в пакеті Група.Прізвище.Lab3 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.
2. Автоматично згенерувати документацію до розробленого пакету.
3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.
4. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.
5. Дати відповідь на контрольні запитання.

Вихідний код програми:

Файл CustomAssaultRifle.java

```
//package CustomAssaultRifle;  
  
import java.io.IOException;  
import java.util.Scanner;  
  
/**  
 * The CustomAssaultRifle class extends the abstract class AssaultRifle and  
 * implements the ShotEngine interface.  
 * This class provides extended shooting methods and uses them to display  
 * shooting information.  
 *  
 * @author Khmilovskiy Stanislav  
 * @version 1.0  
 * @since 1.0  
 * @see AssaultRifle  
 * @see ShotEngine  
 */  
  
public class CustomAssaultRifle extends AssaultRifle implements ShotEngine {
```

```

/**
 * Constructor for the CustomAssaultRifle class.
 *
 * @param model          The model of the rifle
 * @param ammunition      The amount of ammunition
 * @param enlargedMagazine A flag for an enlarged magazine
 * @throws IOException Thrown in case of I/O errors
 */
public CustomAssaultRifle(String model, int ammunition, boolean
enlargedMagazine) throws IOException {
    super(model, ammunition, enlargedMagazine);
}

/**
 * Creates a new CustomAssaultRifle object with the specified parameters.
 *
 * @param model          the model of the assault rifle.
 * @param ammunition      the initial ammunition count.
 * @param caliber         the caliber of the assault rifle.
 * @param weight          the weight of the assault rifle in kilograms.
 * @param price           the price of the assault rifle in dollars.
 * @param scope           {@code true} if the assault rifle has a scope,
{@code false} otherwise.
 * @param muffler         {@code true} if the assault rifle has a muffler,
{@code false} otherwise.
 * @param enlargedMagazine {@code true} if the assault rifle has an enlarged
magazine, {@code false} otherwise.
 * @throws IOException if an error occurs while opening the log file.
 */
public CustomAssaultRifle(String model, int ammunition, double caliber,
double weight, double price, boolean scope, boolean muffler, boolean
enlargedMagazine) throws IOException {
    super(model, ammunition, caliber, weight, price, scope, muffler,
enlargedMagazine);
}

/**
 * Extended automatic fire method with specified accuracy.
 *
 * @param bullets The number of bullets to fire
 * @param accuracy The accuracy of the shots, ranging from 0.0 to 1.0
 */
//Розширений метод automaticFire:
public void automaticFire(int bullets, double accuracy) {
    int bulletsHit = (int) (bullets * accuracy);
    System.out.println("Number of bullets hit: " + bulletsHit);
    automaticFire(bullets);
}

/**
 * Extended burst fire method with a specified sound.
 *
 * @param bullets The number of bullets to fire
 * @param sound    The sound of the shots
 */
//Розширений метод burstFire:
public void burstFire(int bullets, String sound) {
    System.out.println("Shooting sound: ");
    for (int i = 0; i < bullets; i++) {
        System.out.println(sound+"!");
    }
    burstFire(bullets);
}

/**
 * Extended single shot method with a specified target.
 *

```

```

    * @param bullets The number of bullets to fire
    * @param target The target being fired upon
    */
    //Розширений метод singleFire:
    public void singleFire(int bullets, String target) {
        System.out.println("Firing at " + target + "!");
        singleFire(bullets);
    }

    /**
     * Method for testing the CustomAssaultRifle class.
     *
     * @param args Command-line arguments
     * @throws IOException Thrown in case of I/O errors
     */
    public static void main(String[] args) throws IOException {
        CustomAssaultRifle rifle = new CustomAssaultRifle("AK-47", 30, true);
        Scanner scanner = new Scanner(System.in);

        //Default TESTING:
        rifle.singleFire(10, "Enemy target");
        System.out.println();
        rifle.burstFire(3, "Boom");
        System.out.println();
        rifle.automaticFire(11, 0.2);

        //Input TESTING:
        /*System.out.print("Enter target: ");
        String target = scanner.nextLine();
        rifle.singleFire(10, target);

        System.out.print("Enter shot sound: ");
        String sound = scanner.nextLine();
        rifle.burstFire(10, sound);

        System.out.println("Enter the accuracy percentage: ");
        double accuracy = scanner.nextDouble();
        rifle.automaticFire(10, accuracy);
        */
    }
}

```

Файл ShotEngine.java

```

//package CustomAssaultRifle;

//Інтерфейс "ShotEngine" містить в собі прототипи розширених методів
/**
 * The ShotEngine interface defines prototypes for extended shooting methods.
 * Classes implementing this interface should provide implementations for these
 * methods.
 *
 * @author Khmilovskiy Stanislav
 * @version 1.0
 * @since 1.0
 */
public interface ShotEngine {
    /**
     * Prototype for an automatic fire method.
     *
     * @param bullets The number of bullets to fire automatically
     */
    void automaticFire(int bullets);

    /**

```

```

    * Prototype for a burst fire method.
    *
    * @param bullets The number of bullets to fire in bursts
    */
void burstFire(int bullets);

/**
 * Prototype for a single shot method.
 *
 * @param bullets The number of single shots to fire
 */
void singleFire(int bullets);
}

```

Результат виконання програми:


```
Firing at Enemy target!
Single shot! (1 bullet used)
Single shot! (1 bullet used)
Single shot! (1 bullet used)
Single shot! (1 bullet used)
Single shot! (1 bullet used)
Single shot! (1 bullet used)
Single shot! (1 bullet used)
Single shot! (1 bullet used)
Single shot! (1 bullet used)
Single shot! (1 bullet used)
Single shot! (1 bullet used)
Remaining ammunition: 20

Shooting sound:
Boom!
Boom!
Boom!
Burst fire! (3 bullets used)
Burst fire! (3 bullets used)
Burst fire! (3 bullets used)
Remaining ammunition: 11

Number of bullets hit: 2
Automatic Fire! (Bullet used: 1)
Automatic Fire! (Bullet used: 2)
Automatic Fire! (Bullet used: 3)
Automatic Fire! (Bullet used: 4)
Automatic Fire! (Bullet used: 5)
Automatic Fire! (Bullet used: 6)
Automatic Fire! (Bullet used: 8)
Automatic Fire! (Bullet used: 9)
Automatic Fire! (Bullet used: 10)
Automatic Fire! (Bullet used: 11)
Remaining ammunition: 0

Process finished with exit code 0
```

Вміст файлу log.txt:



log: Блокнот

Файл Редагування Формат Вигляд Довідка

Fired 10 bullets in 10 single shots.
Fired 9 bullets in 3 bursts.
Fired 11 bullets in automatic mode.

Фрагмент згенерованої документації JavaDoc:

PACKAGE

CLASS

TREE

INDEX

HELP

SUMMARY: NESTED | FIELD | CONSTR | METHODDETAIL: FIELD | CONSTR | METHOD

SEARCH

Interface ShotEngine

All Known Implementing Classes:
CustomAssaultRifle

```
public interface ShotEngine
```

The ShotEngine interface defines prototypes for extended shooting methods. Classes implementing this interface should provide implementations for these methods.

Since:
1.0

Method Summary

All Methods	Instance Methods	Abstract Methods
Modifier and Type	Method	Description
void	<code>automaticFire(int bullets)</code>	Prototype for an automatic fire method.
void	<code>burstFire(int bullets)</code>	Prototype for a burst fire method.
void	<code>singleFire(int bullets)</code>	Prototype for a single shot method.

Method Details

`automaticFire`

```
void automaticFire(int bullets)
```

Відповіді на контрольні запитання:

1. Синтаксис реалізації спадкування.

```
class Subclass extends Superclass {  
    // код підкласу  
}
```

2. Що таке суперклас та підклас?

Суперклас (або батьківський клас) - це клас, від якого спадковується інший клас, який називається підкласом (або дочірнім класом). Підклас успадковує всі поля і методи суперкласу і може розширювати їх або навіть перевизначити.

3. Як звернутися до членів суперкласу з підкласу?

Для звернення до членів суперкласу з підкласу можна використовувати ключове слово “super”. Наприклад, “super.methodName()” викликає метод суперкласу.

4. Коли використовується статичне зв’язування при виклику методу?

Статичне зв’язування (або раннє зв’язування) відбувається під час компіляції. Воно використовується, коли метод, який викликається, визначається на етапі компіляції на основі типу посилання.

5. Як відбувається динамічне зв'язування при виклику методу?

Динамічне зв'язування (або пізнє зв'язування) відбувається під час виконання програми. Воно використовується, коли метод, який викликається, визначається на етапі виконання на основі реального типу об'єкта.

6. Що таке абстрактний клас та як його реалізувати?

Абстрактний клас - це клас, який не може бути створений напряду, але може містити абстрактні (незавершені) методи. Щоб оголосити абстрактний клас, використовуйте ключове слово `abstract`.

Приклад реалізації:

```
abstract class MyAbstractClass {  
    // абстрактний метод  
    public abstract void myMethod();  
}
```

7. Для чого використовується ключове слово `instanceof`?

Ключове слово `instanceof` використовується для перевірки, чи об'єкт є екземпляром певного класу або підкласу. Воно повертає `true`, якщо об'єкт є екземпляром класу, і `false`, якщо ні.

8. Як перевірити чи клас є підкласом іншого класу?

Використати ключове слово `extends` при оголошенні класу для показу спадкування.

9. Що таке інтерфейс?

Інтерфейс в Java - це абстрактний тип, який оголошує набір методів, але не містить їх реалізації. Класи можуть реалізовувати інтерфейси, надаючи конкретну реалізацію методів, вказаних в інтерфейсі.

10. Як оголосити та застосувати інтерфейс?

Щоб оголосити інтерфейс, використовуйте ключове слово `interface`.

Приклад реалізації:

```
interface MyInterface {
```

```
void myMethod();  
}
```

Для застосування інтерфейсу в класі використовуйте ключове слово `implements`.

Приклад реалізації:

```
class MyClass implements MyInterface {  
    public void myMethod() {  
        // реалізація методу  
    }  
}
```

Висновок:

На даній лабораторній роботі я отримав навички роботи з концепціями спадкування та інтерфейсами в мові програмування Java. Також на цій лабораторній роботі я ознайомився з цими важливими аспектами об'єктно-орієнтованого програмування та зрозумів їх роль у створенні більш структурованих і гнучких програм.