Національний університет «Львівська Політехніка»

Інститут комп’ютерних технологій, автоматики та метрології

Кафедра електронних обчислювальних машин



Звіт

Про виконання лабораторної роботи №2

З дисципліни «Кросплатформлені засоби програмування»

**Виконав:**

студент групи КІ-305

Панасюк Т.Ю.

**Перевірив:**

Доцент кафедри ЕОМ

Іванов Ю. С.

Львів – 2023

**Тема:** Класи та пакети.

**Мета:** Ознайомитися з процесом розробки класів та пакетів мовою Java.

**Завдання:**

1. Написати та налагодити програму на мові Java, що реалізує у вигляді класу предметну область згідно варіанту. Програма має задовольняти наступним вимогам:  
 • програма має розміщуватися в пакеті Група.Прізвище.Lab2;  
 • клас має містити мінімум 3 поля, що є об’єктами класів, які описують складові частини предметної області;  
 • клас має містити кілька конструкторів та мінімум 10 методів;  
 • для тестування і демонстрації роботи розробленого класу розробити клас-драйвер;  
 • методи класу мають вести протокол своєї діяльності, що записується у файл;  
 • розробити механізм коректного завершення роботи з файлом (не надіятися на метод finalize());  
 • програма має володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.  
2. Автоматично згенерувати документацію до розробленої програми.  
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.

**Завдання згідно варіанту:**

**Варіант: 21**

**Код програми:**

**Pistol.java:**

**package** KI305.Panasiuk.Lab3;

**import** java.io.\*;

/\*\*

\* The Pistol class represents a firearm with various properties and methods for

\* firing, reloading, and managing ammunition.

\* **@author** taraspanasiuk

\*/

**public** **class** Pistol {

**public** **static** **int** *count* = 0;

**private** String name;

**private** **int** caliber;

**private** **double** weight;

**private** Company company;

**private** Trigger trigger;

**private** Magazine magazine;

/\*\*

\* Default constructor for Pistol.

\*/

**public** Pistol() {

name = "Unknown";

caliber = 9;

weight = 1.0;

company = **new** Company();

trigger = **new** Trigger();

magazine = **new** Magazine();

*count* += 1;

}

/\*\*

\* Parameterized constructor for Pistol.

\*

\* **@param** name The name of the pistol.

\* **@param** caliber The caliber of the pistol.

\* **@param** weight The weight of the pistol.

\* **@param** company The manufacturing company of the pistol.

\* **@param** trigger The trigger type of the pistol.

\* **@param** magazine The magazine associated with the pistol.

\*/

**public** Pistol(String name, **int** caliber, **double** weight, Company company, Trigger trigger, Magazine magazine) {

**this**.name = name;

**this**.caliber = caliber;

**this**.weight = weight;

**this**.company = company;

**this**.trigger = trigger;

**this**.magazine = magazine;

}

/\*\*

\* Get the name of the pistol.

\*

\* **@return** The name of the pistol.

\*/

**public** String getName() {

**return** name;

}

/\*\*

\* Get the caliber of the pistol.

\*

\* **@return** The caliber of the pistol.

\*/

**public** **int** getCaliber() {

**return** caliber;

}

/\*\*

\* Get the weight of the pistol.

\*

\* **@return** The weight of the pistol.

\*/

**public** **double** getWeight() {

**return** weight;

}

/\*\*

\* Get the manufacturing company of the pistol.

\*

\* **@return** The manufacturing company of the pistol.

\*/

**public** Company getCompany() {

**return** company;

}

/\*\*

\* Get the trigger type of the pistol.

\*

\* **@return** The trigger type of the pistol.

\*/

**public** Trigger getTrigger() {

**return** trigger;

}

/\*\*

\* Get the magazine associated with the pistol.

\*

\* **@return** The magazine associated with the pistol.

\*/

**public** Magazine getMagazine() {

**return** magazine;

}

/\*\*

\* Set the name of the pistol.

\*

\* **@param** name The name of the pistol.

\*/

**public** **void** setName(String name) {

**this**.name = name;

}

/\*\*

\* Set the caliber of the pistol.

\*

\* **@param** caliber The caliber of the pistol.

\*/

**public** **void** setCaliber(**int** caliber) {

**this**.caliber = caliber;

}

/\*\*

\* Set the weight of the pistol.

\*

\* **@param** weight The weight of the pistol.

\*/

**public** **void** setWeight(**double** weight) {

**this**.weight = weight;

}

/\*\*

\* Set the manufacturing company of the pistol.

\*

\* **@param** company The manufacturing company of the pistol.

\*/

**public** **void** setCompany(Company company) {

**this**.company = company;

}

/\*\*

\* Set the trigger type of the pistol.

\*

\* **@param** trigger The trigger type of the pistol.

\*/

**public** **void** setTrigger(Trigger trigger) {

**this**.trigger = trigger;

}

/\*\*

\* Set the magazine associated with the pistol.

\*

\* **@param** magazine The magazine associated with the pistol.

\*/

**public** **void** setMagazine(Magazine magazine) {

**this**.magazine = magazine;

}

/\*\*

\* Log an activity to a file.

\*

\* **@param** activity The activity to be logged.

\*/

**public** **void** logActivity(String activity) {

**try** {

FileWriter fileWriter = **new** FileWriter("pistol\_log.txt", **true**);

fileWriter.write(activity + "\n");

fileWriter.close();

} **catch** (IOException e) {

System.***err***.println("Error writing to file: " + e.getMessage());

}

}

/\*\*

\* Fire one round from the pistol.

\*/

**public** **void** fire() {

**if** (magazine.getCurrentRounds() > 0) {

System.***out***.println("Bang! " + name + " fired a round.");

magazine.fireRounds(1);

logActivity(name + " fired a round.");

} **else** {

System.***out***.println("Click! " + name + " is out of ammo.");

logActivity(name + " attempted to fire but ran out of ammo.");

}

}

/\*\*

\* Fire multiple rounds from the pistol.

\*

\* **@param** rounds The number of rounds to fire.

\*/

**public** **void** fireMultipleTimes(**int** rounds) {

**for** (**int** i = 0; i < rounds; i++) {

fire();

}

logActivity(name + " fired " + rounds + " rounds.");

}

/\*\*

\* Toggle the safety of the pistol.

\*/

**public** **void** toggleSafety() {

trigger.setSafety(!trigger.isSafety());

String safetyStatus = trigger.isSafety() ? "enabled" : "disabled";

logActivity(name + " toggled the safety. Status: " + safetyStatus);

}

/\*\*

\* Reload the pistol.

\*/

**public** **void** reload() {

**if** (magazine.getCurrentRounds() < magazine.getCapacity()) {

**int** roundsToLoad = magazine.getCapacity() - magazine.getCurrentRounds();

System.***out***.println("Reloading " + name + " with " + roundsToLoad + " rounds.");

magazine.loadRounds(roundsToLoad);

logActivity(name + " reloaded. Loaded " + roundsToLoad + " rounds.");

} **else** {

System.***out***.println(name + " is already fully loaded.");

logActivity(name + " attempted to reload, but the magazine is already fully loaded.");

}

}

/\*\*

\* Load ammunition into the pistol's magazine.

\*

\* **@param** roundsToAdd The number of rounds to load.

\*/

**public** **void** loadAmmo(**int** roundsToAdd) {

**if** (roundsToAdd > 0) {

magazine.loadRounds(roundsToAdd);

System.***out***.println("Loaded " + roundsToAdd + " rounds into " + name + "'s magazine.");

logActivity(name + " loaded " + roundsToAdd + " rounds into the magazine.");

} **else** {

System.***out***.println("Invalid number of rounds to load.");

logActivity(name + " attempted to load an invalid number of rounds.");

}

}

/\*\*

\* Aim the pistol at the target.

\*/

**public** **void** aim() {

System.***out***.println("Aiming " + name + " at the target.");

logActivity(name + " is aiming at the target.");

}

/\*\*

\* Clear the pistol's magazine.

\*/

**public** **void** clearMagazine() {

**int** roundsInMagazine = magazine.getCurrentRounds();

**if** (roundsInMagazine > 0) {

System.***out***.println("Clearing " + name + "'s magazine. Removed " + roundsInMagazine + " rounds.");

magazine.setCurrentRounds(0);

logActivity(name + " cleared the magazine. Removed " + roundsInMagazine + " rounds.");

} **else** {

System.***out***.println(name + "'s magazine is already empty.");

logActivity(name + " attempted to clear the magazine, but it is already empty.");

}

}

/\*\*

\* Change the magazine capacity of the pistol.

\*

\* **@param** newCapacity The new capacity of the magazine.

\*/

**public** **void** changeMagazineCapacity(**int** newCapacity) {

magazine.setCapacity(newCapacity);

logActivity(name + " changed the magazine capacity to " + newCapacity + " rounds.");

}

/\*\*

\* Check if the pistol is loaded.

\*

\* **@return** True if the pistol is loaded, false otherwise.

\*/

**public** **boolean** isLoaded() {

**boolean** loaded = magazine.getCurrentRounds() > 0;

**if** (loaded) {

logActivity(name + " checked the pistol's status. It is loaded.");

} **else** {

logActivity(name + " checked the pistol's status. It is not loaded.");

}

**return** loaded;

}

}

**Magazine.java:**

**package** KI305.Panasiuk.Lab3;

/\*\*

\* The Magazine class represents a firearm magazine that holds rounds.

\* **@author** taraspanasiuk

\*/

**public** **class** Magazine {

**private** **int** capacity;

**private** **int** currentRounds;

/\*\*

\* Default constructor for a Magazine.

\* Sets the default capacity to 10 rounds and initializes currentRounds to 0.

\*/

**public** Magazine() {

capacity = 10;

currentRounds = 0;

}

/\*\*

\* Parameterized constructor for a Magazine.

\*

\* **@param** capacity The maximum capacity of the magazine.

\* **@param** currentRounds The current number of rounds in the magazine.

\*/

**public** Magazine(**int** capacity, **int** currentRounds) {

**this**.capacity = capacity;

**this**.currentRounds = currentRounds;

}

/\*\*

\* Get the maximum capacity of the magazine.

\*

\* **@return** The maximum capacity of the magazine.

\*/

**public** **int** getCapacity() {

**return** capacity;

}

/\*\*

\* Set the maximum capacity of the magazine.

\*

\* **@param** capacity The maximum capacity of the magazine.

\*/

**public** **void** setCapacity(**int** capacity) {

**this**.capacity = capacity;

}

/\*\*

\* Get the current number of rounds in the magazine.

\*

\* **@return** The current number of rounds in the magazine.

\*/

**public** **int** getCurrentRounds() {

**return** currentRounds;

}

/\*\*

\* Set the current number of rounds in the magazine.

\*

\* **@param** currentRounds The current number of rounds in the magazine.

\*/

**public** **void** setCurrentRounds(**int** currentRounds) {

**this**.currentRounds = currentRounds;

}

/\*\*

\* Load rounds into the magazine.

\*

\* **@param** rounds The number of rounds to load into the magazine.

\*/

**public** **void** loadRounds(**int** rounds) {

**if** (rounds > 0 && rounds <= capacity) {

currentRounds = rounds;

} **else** {

System.***out***.println("Invalid number of rounds to load.");

}

}

/\*\*

\* Fire rounds from the magazine.

\*

\* **@param** rounds The number of rounds to fire from the magazine.

\*/

**public** **void** fireRounds(**int** rounds) {

**if** (rounds > 0 && rounds <= currentRounds) {

currentRounds -= rounds;

System.***out***.println("Fired " + rounds + " rounds.");

} **else** {

System.***out***.println("Not enough rounds to fire.");

}

}

}

**Trigger.java:**

**package** KI305.Panasiuk.Lab3;

/\*\*

\* The Trigger class represents the trigger mechanism of a firearm.

\* **@author** taraspanasiuk

\*/

**public** **class** Trigger {

**private** String type;

**private** **boolean** safety;

/\*\*

\* Default constructor for a Trigger.

\* Initializes the trigger type to "Standard" and safety to false.

\*/

**public** Trigger() {

type = "Standard";

safety = **false**;

}

/\*\*

\* Parameterized constructor for a Trigger.

\*

\* **@param** type The type of the trigger.

\* **@param** safety Indicates whether the safety is engaged or not.

\*/

**public** Trigger(String type, **boolean** safety) {

**this**.type = type;

**this**.safety = safety;

}

/\*\*

\* Get the type of the trigger.

\*

\* **@return** The type of the trigger.

\*/

**public** String getType() {

**return** type;

}

/\*\*

\* Set the type of the trigger.

\*

\* **@param** type The type of the trigger.

\*/

**public** **void** setType(String type) {

**this**.type = type;

}

/\*\*

\* Check if the safety is engaged.

\*

\* **@return** True if the safety is engaged, false otherwise.

\*/

**public** **boolean** isSafety() {

**return** safety;

}

/\*\*

\* Set the safety status of the trigger.

\*

\* **@param** safety True to engage the safety, false to disengage.

\*/

**public** **void** setSafety(**boolean** safety) {

**this**.safety = safety;

}

}

**PistolDeiver.java**

**package** KI305.Panasiuk.Lab3;

/\*\*

\* This class demonstrates the functionality of a Pistol object.

\* **@author** taraspanasiuk

\*/

**public** **class** PistolDriver {

/\*\*

\* The main method of the PistolDriver class, which serves as the entry point

\* for demonstrating the functionality of a Pistol object.

\*

\* **@param** args The command-line arguments (not used in this program).

\*/

**public** **static** **void** main(String[] args) {

// Creating a pistol object

Pistol myPistol = **new** Pistol("Glock 19", 9, 1.0, **new** Company(), **new** Trigger(), **new** Magazine());

// Demonstrating pistol properties

System.***out***.println("Pistol Name: " + myPistol.getName());

System.***out***.println("Caliber: " + myPistol.getCaliber());

System.***out***.println("Weight: " + myPistol.getWeight() + " kg");

// Loading the magazine

myPistol.loadAmmo(10);

// Firing

System.***out***.println("\nFiring:");

myPistol.fire();

myPistol.fire();

myPistol.fire();

myPistol.fireMultipleTimes(3); // Fire 3 more rounds

// Checking for ammunition

**boolean** hasAmmo = myPistol.isLoaded();

System.***out***.println("\nRounds in the magazine: " + (hasAmmo ? "Yes" : "No"));

// Checking safety status and toggling it

System.***out***.println("\nSafety Status:");

**boolean** isSafetyOn = myPistol.getTrigger().isSafety();

System.***out***.println("Safety is on: " + (isSafetyOn ? "Yes" : "No"));

myPistol.toggleSafety();

isSafetyOn = myPistol.getTrigger().isSafety();

System.***out***.println("Safety is on: " + (isSafetyOn ? "Yes" : "No"));

// Checking pistol status (loaded or not)

System.***out***.println("\nPistol Status:");

**boolean** isPistolLoaded = myPistol.isLoaded();

System.***out***.println("Pistol is loaded: " + (isPistolLoaded ? "Yes" : "No"));

// Changing magazine capacity and checking it

System.***out***.println("\nChanging Magazine Capacity:");

myPistol.changeMagazineCapacity(15);

**int** newCapacity = myPistol.getMagazine().getCapacity();

System.***out***.println("Magazine capacity: " + newCapacity);

// Reloading the magazine

System.***out***.println("\nReloading the Magazine:");

myPistol.reload();

// Aiming

System.***out***.println("\nAiming:");

myPistol.aim();

// Clearing the magazine

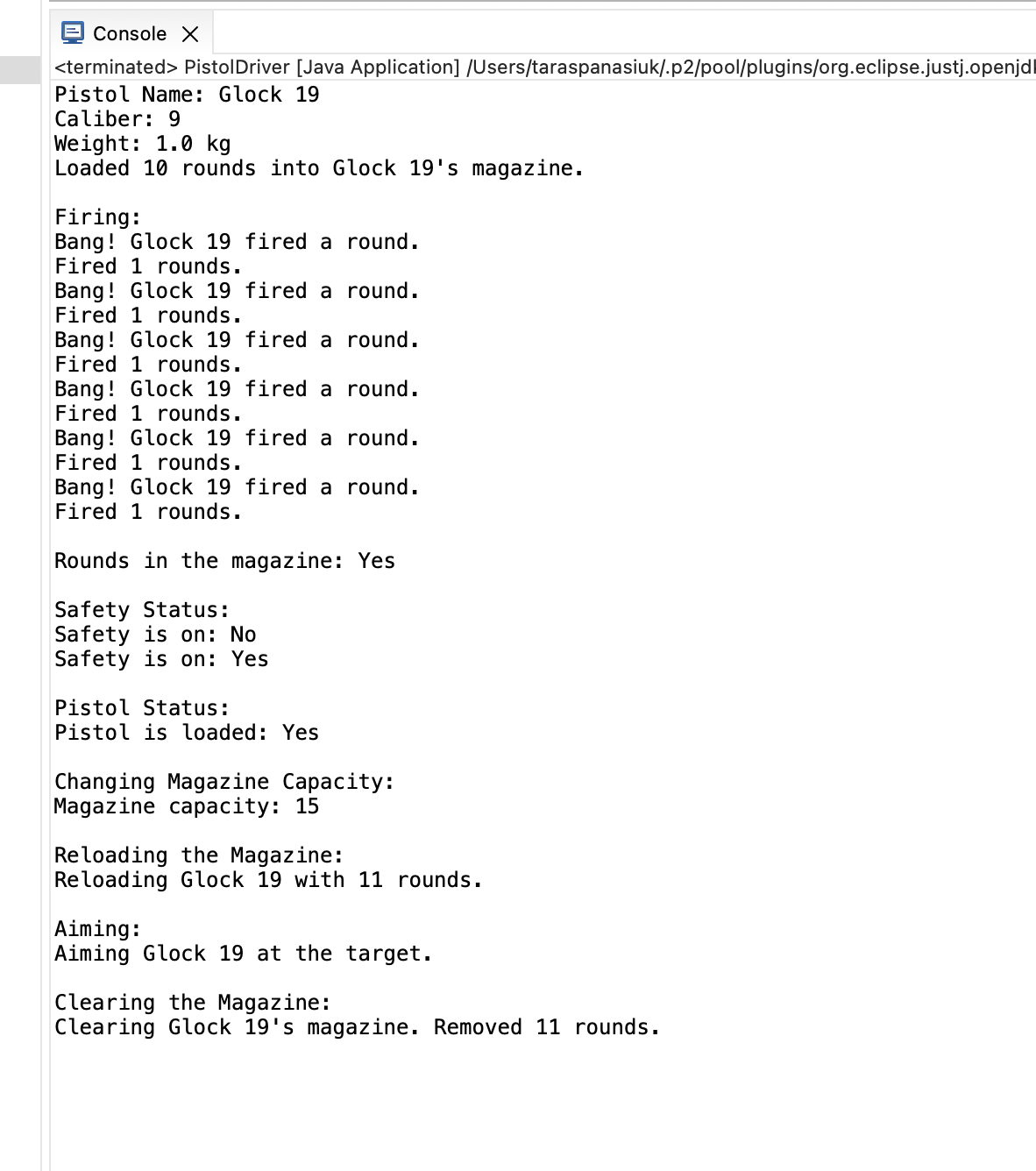
System.***out***.println("\nClearing the Magazine:");

myPistol.clearMagazine();

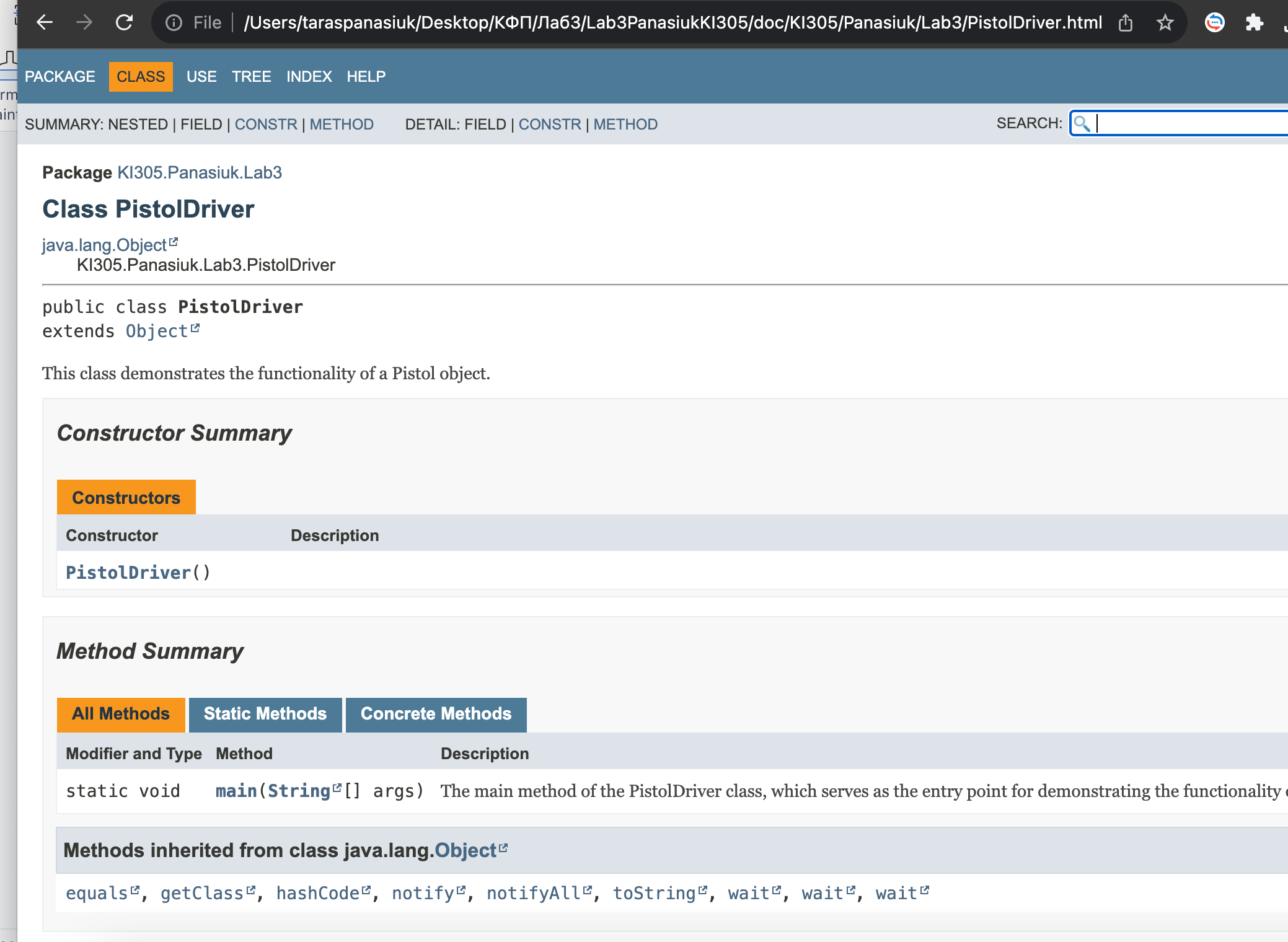
}

}

**Результат роботи програми у консолі та файлі:**

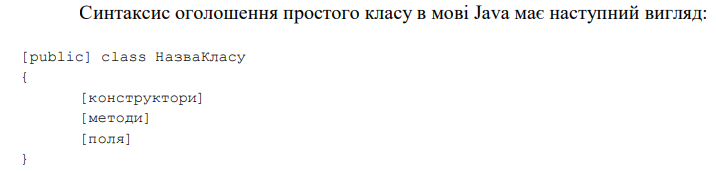


**Фрагмент генерації документації:**



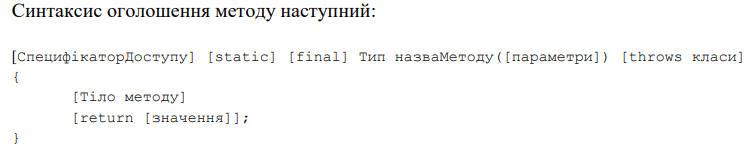
**КОНТРОЛЬНІ ПИТАННЯ**

1. Синтаксис визначення класу.

**Відповідь: **

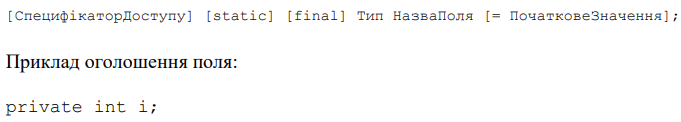
1. Синтаксис визначення методу.

**Відповідь:**

****

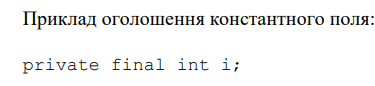
1. Синтаксис оголошення поля.

**Відповідь:**



1. Як оголосити та ініціалізувати константне поле?

**Відповідь:**

****

Ініціалізація при оголошенні:

При оголошенні поля, ви можете встановити його значення. Наприклад:

java

Copy code

public class MyClass {

// Константне поле ініціалізується при оголошенні

private final int MY\_CONSTANT = 10;

}

Ініціалізація в конструкторі:

Якщо значення константи вам потрібно встановити під час створення об'єкта класу, ви можете ініціалізувати його у конструкторі. Однак, після першої ініціалізації константне поле не можна змінювати.

java

Copy code

public class MyClass {

private final int MY\_CONSTANT;

public MyClass(int value) {

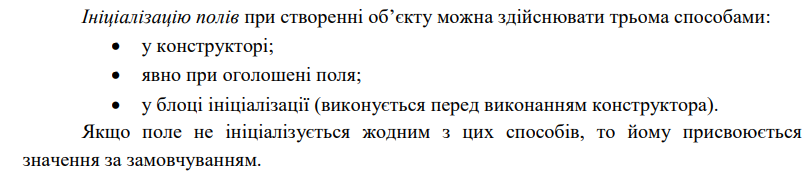
MY\_CONSTANT = value;

}

}

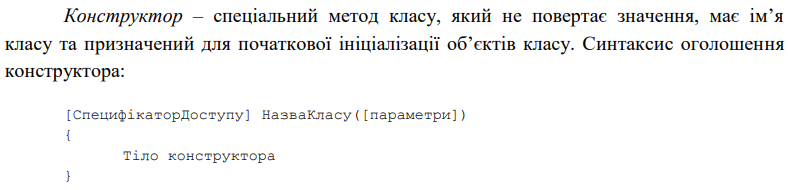
1. Які є способи ініціалізації полів?

**Відповідь:**



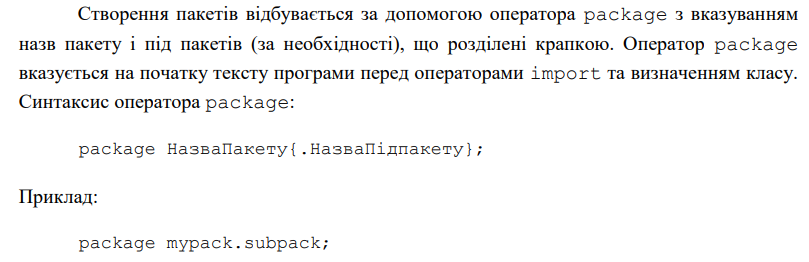
1. Синтаксис визначення конструктора.

**Відповідь:**



1. Синтаксис оголошення пакету.

**Відповідь:**



1. Як підключити до програми класи, що визначені в зовнішніх пакетах?

**Відповідь:**

Щоб використовувати класи, що належать зовнішнім пакетам у вашій програмі, вам потрібно вказати їх імпорт.

1. В чому суть статичного імпорту пакетів?

**Відповідь:**

Статичний імпорт в Java - це механізм, який дозволяє коротше викликати статичні методи і статичні поля з класу, не вказуючи повністю кваліфіковане ім'я класу.

1. Які вимоги ставляться до файлів і каталогів при використанні пакетів?

**Відповідь:**

При використанні пакетів у Java ставляються певні вимоги до організації файлів і каталогів:

1. Ієрархія пакетів: Файли повинні бути розміщені у відповідних пакетах згідно з ієрархією пакетів. Це означає, що розташування файлів відображає структуру пакетів.
2. Директорія пакету: Файли Java повинні розташовуватися у відповідних папках пакетів. Наприклад, якщо ви маєте пакет com.example, ваш файл Java повинен розташовуватися у відповідній структурі каталогів, такій як com/example/YourClass.java.
3. Ім'я файлу: Ім'я файлу Java повинно відповідати імені класу, яке він містить. Наприклад, якщо ваш клас називається YourClass, то файл повинен мати ім'я YourClass.java.
4. Імпорти і пакети: У файлі Java можуть бути імпорти і заявки package. Це використовується для підключення класів з інших пакетів або для визначення поточного пакету, від якого відповідні класи його визначають.
5. Відносні шляхи: При використанні імпортів вам необхідно вказувати повний шлях до класу відносно його розташування в пакеті. Наприклад: import com.example.packageName.ClassName;.

**Висновок:** на цій лабораторній роботі я ознайомитися з процесом розробки класів та пакетів мовою Java.