Національний університет «Львівська Політехніка»

Інститут комп’ютерних технологій, автоматики та метрології

Кафедра електронних обчислювальних машин



Звіт

Про виконання лабораторної роботи №3

З дисципліни «Кросплатформлені засоби програмування»

**Виконав:**

студент групи КІ-305

Панасюк Т.Ю.

**Перевірив:**

Доцент кафедри ЕОМ

Іванов Ю. С.

Львів – 2023

**Тема:** Спадкування та інтерфейси.

**Мета:** Ознайомитися з спадкуванням та інтерфейсами у мові Java.

**Завдання:**

1. Написати та налагодити програму на мові Java, що розширює клас, що реалізований у лабораторній роботі №2, для реалізації предметної області заданої варіантом. Суперклас, що реалізований у лабораторній роботі №2, 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.

**Завдання згідно варіанту:**

**Варіант: 21**

**Виконання:**

**Код програми:**

**Pistol.java:**

**package** KI305.Panasiuk.Lab4;

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

/\*\*

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

\* firing, reloading, and managing ammunition.

\* **@author** taraspanasiuk

\*/

**public** **abstract** **class** Pistol **extends** Object **implements** Fireable{

**private** String name;

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

**private** Company company;

**private** Trigger trigger;

**private** Magazine magazine;

/\*\*

\* Default constructor for Pistol.

\*/

**public** Pistol() {

name = "Unknown";

weight = 1.0;

company = **new** Company();

trigger = **new** Trigger();

magazine = **new** Magazine();

}

/\*\*

\* 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, **double** weight, Company company, Trigger trigger, Magazine magazine) {

**this**.name = name;

**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 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 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;

}

/\*\*

\* Toggle the safety of the pistol.

\*/

**public** **abstract** **void** toggleSafety();

/\*\*

\* Aim the pistol at the target.

\*/

**public** **abstract** **void** aim();

/\*\*

\* 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.

\*

\*/

@Override

**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.

\*/

@Override

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

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

fire();

}

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

}

/\*\*

\* Reload the pistol.

\*/

@Override

**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.

\*/

@Override

**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.");

}

}

/\*\*

\* 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;

}

}

**Trigger.java:**

**package** KI305.Panasiuk.Lab4;

/\*\*

\* 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;

}

}

**WaterPistol.java:**

**package** KI305.Panasiuk.Lab4;

/\*\*

\* The WaterPistol class represents a water-based toy pistol.

\* **@author** taraspanasiuk

\*/

**public** **class** WaterPistol **extends** Pistol {

**private** **int** waterCapacity;

/\*\*

\* Default constructor for WaterPistol.

\*/

**public** WaterPistol() {

**super**();

waterCapacity = 0;

}

/\*\*

\* Parameterized constructor for WaterPistol.

\*

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

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

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

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

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

\* **@param** waterCapacity The water capacity of the water pistol.

\*/

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

**super**(name, weight, company, trigger, magazine);

**this**.waterCapacity = waterCapacity;

}

/\*\*

\* Get the water capacity of the water pistol.

\*

\* **@return** The water capacity of the water pistol.

\*/

**public** **int** getWaterCapacity() {

**return** waterCapacity;

}

/\*\*

\* Set the water capacity of the water pistol.

\*

\* **@param** waterCapacity The water capacity of the water pistol.

\*/

**public** **void** setWaterCapacity(**int** waterCapacity) {

**this**.waterCapacity = waterCapacity;

}

/\*\*

\* Spray water from the water pistol.

\*/

**public** **void** sprayWater() {

**if** (waterCapacity > 0) {

System.***out***.println("Spray! " + getName() + " sprayed water.");

waterCapacity--;

logActivity(getName() + " sprayed water.");

} **else** {

System.***out***.println(getName() + " is out of water.");

logActivity(getName() + " attempted to spray water but ran out.");

}

}

/\*\*

\* Refill the water pistol with water.

\*

\* **@param** waterToAdd The amount of water to add.

\*/

**public** **void** refillWater(**int** waterToAdd) {

**if** (waterToAdd > 0) {

waterCapacity += waterToAdd;

System.***out***.println("Refilled " + getName() + " with " + waterToAdd + " units of water.");

logActivity(getName() + " refilled with " + waterToAdd + " units of water.");

} **else** {

System.***out***.println("Invalid amount of water to refill.");

logActivity(getName() + " attempted to refill with an invalid amount of water.");

}

}

/\*\*

\* Toggle the safety of the water pistol.

\*/

@Override

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

System.***out***.println("Toggle safety for " + getName() + " water pistol.");

}

/\*\*

\* Aim the water pistol at the target.

\*/

@Override

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

System.***out***.println("Aim " + getName() + " water pistol at the target.");

}

}

**WaterPistolDriver.java:**

**package** KI305.Panasiuk.Lab4;

/\*\*

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

\* **@author** taraspanasiuk

\*/

**public** **class** WaterPistolDriver {

/\*\*

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

\* for demonstrating the functionality of a WaterPistol object.

\*

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

\*/

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

// Creating a water pistol object

WaterPistol myWaterPistol = **new** WaterPistol("Super Soaker", 0.5, **new** Company(), **new** Trigger(), **new** Magazine(), 500);

// Demonstrating water pistol properties

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

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

System.***out***.println("Water Capacity: " + myWaterPistol.getWaterCapacity() + " ml");

// Refilling the water pistol

myWaterPistol.refillWater(250);

// Spraying water

System.***out***.println("\nSpraying Water:");

myWaterPistol.sprayWater();

myWaterPistol.sprayWater();

myWaterPistol.sprayWater();

myWaterPistol.sprayWater();

// Checking for water

**boolean** hasWater = myWaterPistol.getWaterCapacity() > 0;

System.***out***.println("\nWater in the water pistol: " + (hasWater ? "Yes" : "No"));

// Checking safety status and toggling it

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

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

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

myWaterPistol.toggleSafety();

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

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

// Checking water pistol status (loaded or not)

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

**boolean** isWaterPistolLoaded = myWaterPistol.isLoaded();

System.***out***.println("Water pistol is loaded: " + (isWaterPistolLoaded ? "Yes" : "No"));

// Changing magazine capacity and checking it

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

myWaterPistol.setWaterCapacity(750);

**int** newWaterCapacity = myWaterPistol.getWaterCapacity();

System.***out***.println("Water capacity: " + newWaterCapacity);

// Reloading the water pistol (in the context of a water pistol, this means refilling water)

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

myWaterPistol.refillWater(500);

// Aiming

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

myWaterPistol.aim();

// Clearing the water pistol (in the context of a water pistol, this doesn't apply)

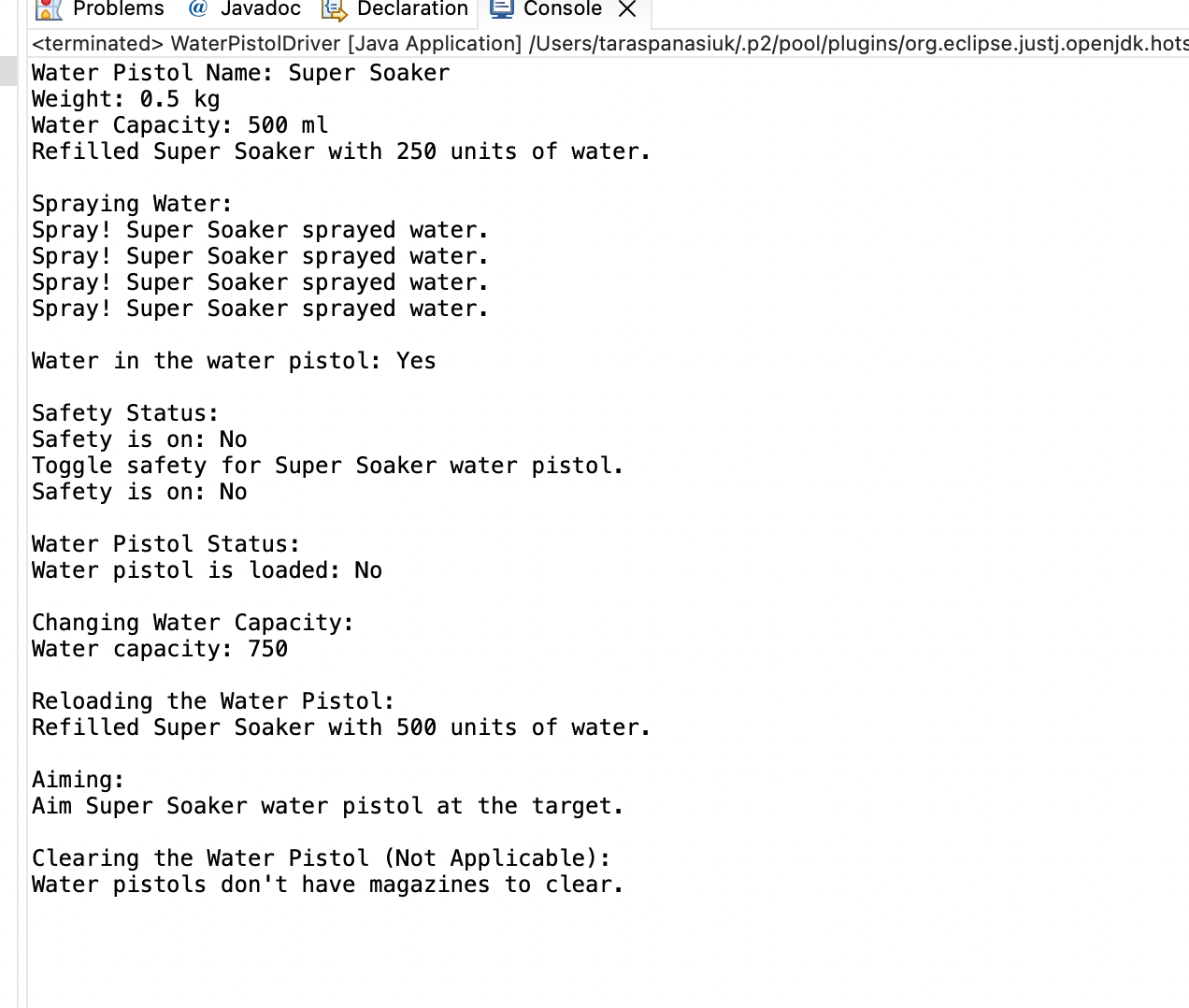
System.***out***.println("\nClearing the Water Pistol (Not Applicable):");

System.***out***.println("Water pistols don't have magazines to clear.");

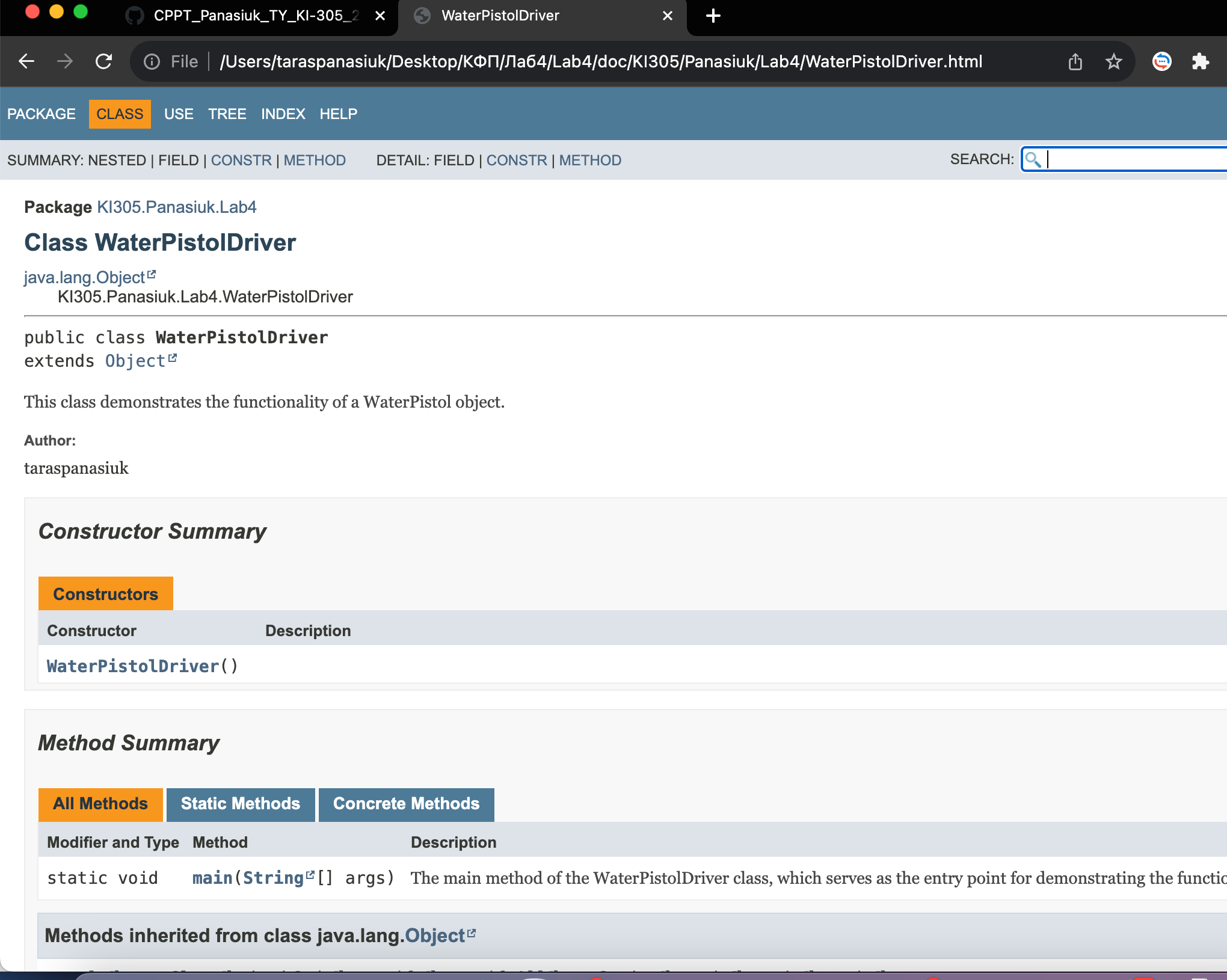
}

}

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



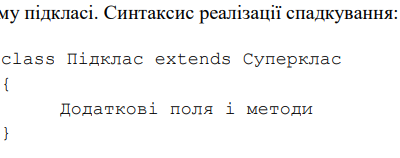
**Фрагмент генерації документації:**



**КОНТРОЛЬНІ ПИТАННЯ**

1. Синтаксис реалізації спадкування.

**Відповідь:**

****

1. Що таке суперклас та підклас?**Відповідь:**

В термінах мови Java базовий клас найчастіше називається суперкласом, а похідний клас – підкласом

1. Як звернутися до членів суперкласу з підкласу?

**Відповідь:**

Для доступу до членів суперкласу з підкласу в Java використовуються ключові слова super та ідентифікатори класів.

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

**Відповідь:**

Якщо метод є приватним, статичним, фінальним або конструктором, то для нього застосовується механізм статичного зв’язування. Механізм статичного зв’язування передбачає визначення методу, який необхідно викликати, на етапі компіляції.

1. Як відбувається динамічне зв’язування при виклику методу?

**Відповідь:**

Якщо для виклику методу використовується динамічне зв’язування, то віртуальна машина повинна викликати версію методу, що відповідає фактичному типу об’єкту на який посилається об’єктна змінна. Оскільки на пошук необхідного методу потрібно багато часу, то віртуальна машина заздалегідь створює для кожного класу таблицю методів, в якій перелічуються сигнатури 5 всіх методів і фактичні методи, що підлягають виклику. При виклику методу віртуальна машина просто переглядає таблицю методів.

1. Що таке абстрактний клас та як його реалізувати?

**Відповідь:**

Абстрактні класи призначені бути основою для розробки ієрархій класів та не

дозволяють створювати об’єкти свого класу. Вони реалізуються за допомогою ключового

слова abstract.

Щоб створити абстрактний клас у Java, використовується ключове слово abstract:

abstract class AbstractClass {

// Абстрактний метод без реалізації (тіло відсутнє)

public abstract void abstractMethod();

}

1. Для чого використовується ключове слово instanceof?

**Відповідь:**

Наявність бодай одної такої виключної ситуації призводить до аварійного завершення програми. Щоб уникнути цього слід перед приведенням типів використати оператор instanceof, який повертає true, якщо посилання посилається на об’єкт фактичний тип якого є не вищим в ієрархії типів, ніж вказаний у операторі instanceof, і false у протилежному випадку.

1. Як перевірити чи клас є підкласом іншого класу?

**Відповідь:**

У Java використовується ключове слово instanceof для перевірки, чи є об'єкт екземпляром певного класу або чи є певний клас підкласом іншого класу.

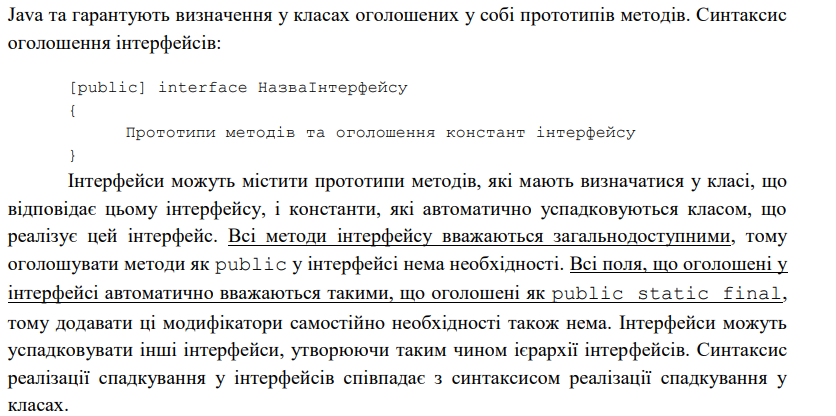
1. Що таке інтерфейс?

**Відповідь:**

Інтерфейси вказують що повинен робити клас не вказуючи як саме він це повинен робити. Інтерфейси покликані компенсувати відсутність множинного спадкування у мові Java та гарантують визначення у класах оголошених у собі прототипів методів.

1. Як оголосити та застосувати інтерфейс?

**Відповідь:**

****

**Висновок:** на цій лабораторній роботі я ознайомився з спадкуванням та інтерфейсами у мові Java. Перетворив попередній клас водойма у абстрактний і реалізував деякі його методи у новоствореному класі море. Також створив інтерфейс «Приріст кораблів» і реалізував його метод в класі море також.