Міністерство освіти і науки України Національний університет

«Львівська політехніка» Кафедра

«Електронних обчислювальних машин»



Звіт з лабораторної роботи № 2

дисципліни: «Кросплатформенні засоби програмування»

на тему: «Класи та пакети»

**Виконав:** студент групи KI-304 Мох М. П.

**Прийняв:** Олексів М. В.

Львів – 2023

**Мета роботи:** 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 (12. Водойма). Програма має задовольняти наступним вимогам:
   * програма має розміщуватися в пакеті Група.Прізвище.Lab3;
   * 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.

1. Автоматично згенерувати документацію до розробленого пакету.
2. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації.
3. Дати відповідь на контрольні запитання

**Силка на GitHub:** <https://github.com/MarianMoh/LabsMohKI-304>

# Варіант 12: Водойма

**Код програми:**

Файл Lab2:

package KI304.Moh.Lab2;  
  
*/\*\*  
 \* The {@code Lab2} class serves as a demonstration of the functionality of the {@code Reservoir} class.  
 \* It creates a {@code Reservoir} object, performs various operations on it, and displays the results.  
 \*  
 \* @author Marian Moh  
 \* @version 1.0  
 \* @since 2023-11-26  
 \*/*public class Lab2 {  
  
 */\*\*  
 \* The main method serves as the entry point of the program.  
 \* It creates a {@code Reservoir} object, performs various operations on it, and displays the results.  
 \*  
 \* @param args The command line arguments.  
 \*/* public static void main(String[] args) {  
 // Create object reservoir  
 Reservoir reservoir = new Reservoir("lake",50000, 15);  
 // Show info about this object  
 reservoir.showInfo();  
  
 // Error in rename method, because the name is invalid  
 reservoir.rename("123");  
 // Good example of rename  
 reservoir.rename("White lake");  
  
 // Add season  
 reservoir.addSeason();  
 // Next Season and show info  
 reservoir.nextSeason();  
  
 // Error in addSeason, because the season is already exists  
 reservoir.addSeason();  
 // Next season  
 reservoir.nextSeason();  
  
 // Delete season  
 reservoir.deleteSeason();  
 // Error in nextSeason, because season was deleted  
 reservoir.nextSeason();  
  
 // Dry up our reservoir  
 reservoir.dry();  
 // Program already finished, we won't see this message  
 System.*out*.println("Program is dead!");  
 }  
}

Файл Reservoir.java

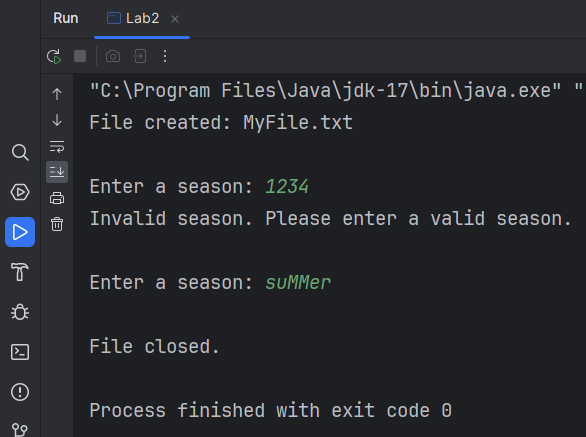
package KI304.Moh.Lab2;  
  
import java.io.File;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.Scanner;  
  
*/\*\*  
 \* The {@code Reservoir} class represents a reservoir with various properties  
 \* such as name, amount of water, water temperature, and current season.  
 \* It provides methods to perform operations on the reservoir.  
 \*  
 \* @author Marian Moh  
 \* @version 1.0  
 \* @since 2023-11-25  
 \*/*public class Reservoir {  
  
 */\*\* The name of the reservoir. \*/* private String name;  
  
 */\*\* The current season of the reservoir. \*/* private Season season;  
  
 */\*\* The amount of water in the reservoir. \*/* private int amountOfWater;  
  
 */\*\* The temperature of the water in the reservoir. \*/* private int waterTemperature;  
  
 */\*\* Flag indicating whether the water in the reservoir is frozen. \*/* private boolean waterIsFrozen;  
  
 */\*\* The FileWriter used to log information to a file. \*/* private FileWriter fileWriter;  
  
 */\*\* The constant file name for logging. \*/* private static final String *FILE\_NAME* = "MyFile.txt";  
  
 */\*\*  
 \* Default constructor for the {@code Reservoir} class.  
 \* Initializes the reservoir and creates a log file.  
 \*/* public Reservoir() {  
 createFile();  
 season = Season.*UNKNOWN*;  
 waterIsFrozen = waterTemperature < 0;  
 }  
  
 */\*\*  
 \* Parameterized constructor for the {@code Reservoir} class.  
 \* Initializes the reservoir with specified properties and creates a log file.  
 \*  
 \* @param name The name of the reservoir.  
 \* @param amountOfWater The amount of water in the reservoir.  
 \* @param waterTemperature The temperature of the water in the reservoir.  
 \*/* public Reservoir(String name, int amountOfWater, int waterTemperature) {  
 this();  
 this.name = name;  
 this.amountOfWater = amountOfWater;  
 this.waterTemperature = waterTemperature;  
 }  
  
 */\*\*  
 \* Creates a log file for the reservoir.  
 \* If the file does not exist, it will be created.  
 \*/* private void createFile() {  
 File file = new File(*FILE\_NAME*);  
 try {  
 if (!file.exists() && file.createNewFile()) {  
 System.*out*.println("File created: " + *FILE\_NAME* + "\n");  
 }  
 fileWriter = new FileWriter(file);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Logs a message to the file associated with the reservoir.  
 \*  
 \* @param message The message to be logged.  
 \*/* public void log(String message) {  
 try {  
 fileWriter.write(message + "\n");  
 fileWriter.flush();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Closes the file associated with the reservoir.  
 \* If the file is not null, it will be closed.  
 \*/* public void closeFile() {  
 try {  
 if (fileWriter != null) {  
 fileWriter.close();  
 System.*out*.println("\nFile closed.");  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Displays information about the reservoir, including its name, amount of water,  
 \* water temperature, current season, and whether the water is frozen.  
 \*/* public void showInfo() {  
 log("\nName: " + name);  
 log("Amount of water: " + amountOfWater);  
 log("Water temperature: " + waterTemperature);  
  
 if(!seasonIsUnknown()) {  
 log("Season is: " + season);  
 }  
 if(waterIsFrozen){  
 log("Water is frozen");  
 }  
 }  
  
 */\*\*  
 \* Renames the reservoir with a new name, if the provided name is valid.  
 \*  
 \* @param newName The new name for the reservoir.  
 \*/* public void rename(String newName) {  
 if (newName.matches("^[a-zA-Z ]{3,}$")) {  
 name = newName;  
 log("\nNew name: " + newName);  
 } else {  
 log("\n- Error! New name \"" + newName + "\" is incorrect :(");  
 }  
 }  
  
 */\*\*  
 \* Drains the reservoir, setting its properties to default values,  
 \* logs a message, closes the log file, and exits the program.  
 \*/* public void dry() {  
 name = "Unknown";  
 amountOfWater = 0;  
 waterTemperature = 0;  
  
 log("\n-\_- The reservoir has dried up...");  
 closeFile();  
 System.*exit*(0);  
 }  
  
 */\*\*  
 \* Adds a new season to the reservoir, if the current season is unknown.  
 \* Prompts the user to enter a season and logs the new season.  
 \*/* public void addSeason() {  
 if(seasonIsUnknown()) {  
 Scanner scanner = new Scanner(System.*in*);  
 String strOfSeason = null;  
  
 while (true) {  
 try {  
 System.*out*.print("Enter a season: ");  
 strOfSeason = scanner.nextLine().toUpperCase();  
 season = Season.*valueOf*(strOfSeason);  
  
 log("\nNew season is: " + season);  
 break;  
 } catch (IllegalArgumentException e) {  
 System.*out*.println("Invalid season. Please enter a valid season.\n");  
 log("\nError! Invalid season: " + strOfSeason);  
 }  
 }  
 showInfo();  
 } else {  
 log("\n- Error! You already have the season: " + season);  
 }  
 }  
  
 */\*\*  
 \* Deletes the current season of the reservoir and logs the action.  
 \*/* public void deleteSeason() {  
 season = Season.*UNKNOWN*;  
 log("\nSeason was deleted");  
 }  
  
 */\*\*  
 \* Changes the current season of the reservoir based on the current season.  
 \* Logs the change and provides information about the new season.  
 \*/* public void nextSeason() {  
 if(!seasonIsUnknown()) {  
 log("\nSeason was changed");  
 switch (season) {  
 case *WINTER* -> springIsComing();  
 case *SPRING* -> summerIsComing();  
 case *SUMMER* -> autumnIsComing();  
 case *AUTUMN* -> winterIsComing();  
 }  
 } else {  
 log("\n- Error! You don't have a season to change it");  
 }  
 }  
  
 */\*\*  
 \* Checks if the current season of the reservoir is unknown.  
 \*  
 \* @return {@code true} if the season is unknown, {@code false} otherwise.  
 \*/* public boolean seasonIsUnknown() {  
 return season.equals(Season.*UNKNOWN*);  
 }  
  
 */\*\*  
 \* Sets the properties of the reservoir for the winter season.  
 \* Logs the change and displays information about the reservoir.  
 \*/* private void winterIsComing() {  
 season = Season.*WINTER*;  
 waterTemperature = 0;  
 waterIsFrozen = true;  
 showInfo();  
 }  
  
 */\*\*  
 \* Sets the properties of the reservoir for the spring season.  
 \* Logs the change, adjusts the amount of water, and displays information about the reservoir.  
 \*/* private void springIsComing() {  
 season = Season.*SPRING*;  
 waterTemperature = 14;  
 waterIsFrozen = false;  
 amountOfWater = (int) (amountOfWater \* 1.2);  
 showInfo();  
 }

*/\*\*  
 \* Sets the properties of the reservoir for the summer season.  
 \* Logs the change, adjusts the amount of water, and displays information about the reservoir.  
 \* If the amount of water is less than 5000, the reservoir dries up.  
 \*/* private void summerIsComing() {  
 season = Season.*SUMMER*;  
 waterTemperature = 20;  
 waterIsFrozen = false;  
 amountOfWater = (int) (amountOfWater \* 0.8);  
  
 if(amountOfWater < 5000) {  
 dry();  
 }  
 showInfo();  
 }  
  
 */\*\*  
 \* Sets the properties of the reservoir for the autumn season.  
 \* Logs the change and displays information about the reservoir.  
 \*/* private void autumnIsComing() {  
 season = Season.*AUTUMN*;  
 waterTemperature = 10;  
 waterIsFrozen = false;  
 showInfo();  
 }  
}

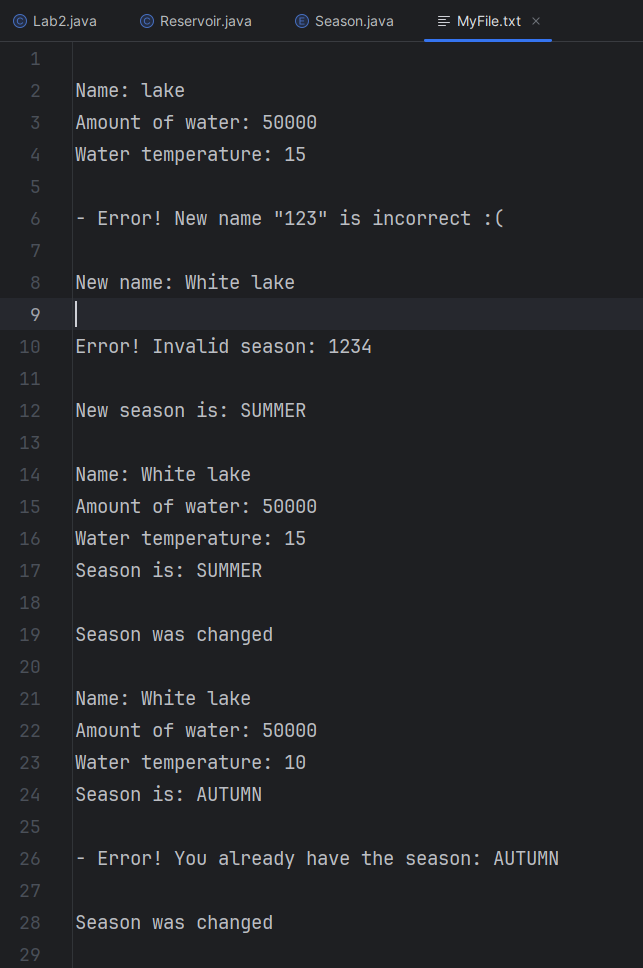
Файл Season.java

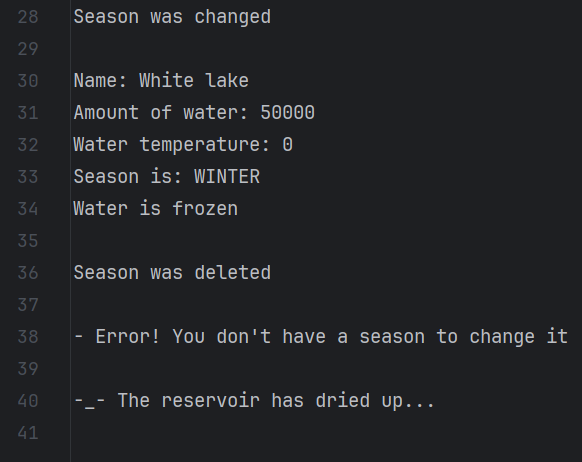
package KI304.Moh.Lab2; */\*\*  
 \* The {@code Season} enum represents the different seasons.  
 \* It includes UNKNOWN, WINTER, SPRING, SUMMER, and AUTUMN.  
 \*/*public enum Season {  
 *UNKNOWN*, *WINTER*, *SPRING*, *SUMMER*, *AUTUMN*}

# Результат виконання програми:



Текстовий файл:





**Фрагмент згенерованої документації:**

# 

**Відповіді на контрольні запитання:**

1. Синтаксис визначення класу.
   * public class ClassName {

// Class members (fields, methods, constructors)

}

1. Синтаксис визначення методу.
   * public returnType methodName(parameters) {

// Method body

}

1. Синтаксис оголошення поля.
   * accessModifier dataType fieldName;
2. Як оголосити та ініціалізувати константне поле?
   * public static final dataType CONSTANT\_NAME = initial\_value;
3. Які є способи ініціалізації полів?
   * Явна ініціалізація при оголошенні поля.
   * Ініціалізація у конструкторі класу.
   * Ініціалізація у блоку ініціалізації (конструкторі, статичному або звичайному).
4. Синтаксис визначення конструктора.
   * public ClassName(parameters) {

// Constructor body

}

1. Синтаксис оголошення пакету.
   * package packageName.subpackage;
2. Як підключити до програми класи, що визначені в зовнішніх пакетах?
   * Вказати повне ім'я класу перед використанням (наприклад, java.util.Date today = new java.util.Date();).
   * Використовувати оператор import для підключення класів з інших пакетів, щоб уникнути повторення повного імені класу.
3. В чому суть статичного імпорту пакетів?
   * Статичний імпорт дозволяє підключити статичні методи і поля класів без повного імені класу.
   * Завдяки статичному імпорту, можна використовувати статичні члени класу, не додаваючи перед ними ім'я класу.
4. Які вимоги ставляться до файлів і каталогів при використанні пакетів?
   * Назви пакетів повинні відповідати структурі каталогів.
   * Назви загальнодоступних класів повинні співпадати з назвами файлів, де вони розміщені.
   * Після компіляції ієрархія каталогів проекту повинна відповідати ієрархії пакетів.
   * Для компіляції та запуску програми слід використовувати шляхи до файлів та пакетів.

# Висновок

У ході виконання даної лабораторної роботи, отримав цінні навички розробки класів та пакетів у мові програмування Java. Ця лабораторна робота надала мені можливість ознайомитися з базовими конструкціями Java, такими як оголошення класів, методів та полів. Я навчився правильно структурувати свій код, визначати доступ до класів та їх членів, а також використовувати модифікатори доступу для керування видимістю.