

Міністерство освіти і науки України
Національний технічний університет України «Київський
політехнічний інститут імені Ігоря Сікорського»

Звіт

з лабораторної роботи № 4

з дисципліни

«Мова програмування Java»

Виконала студентка групи ЗП_зп-41
Павлюк Р.В.

Перевірив Орленко С.П.

Київ 2025

Для виконання лабораторної роботи я обрала Варіант 2. Відповідно до завдання, я реалізувала ієрархію Java-класів для вольєрів з різними видами тварин, використовуючи принципи узагальненого програмування (Generics).

У файлі Animal.java я реалізувала ієрархію класів тварин

- Animal: Базовий абстрактний клас, що реалізує Serializable для можливості збереження у файл.
- Mammal та Bird: Проміжні класи для ссавців та птахів.
- Lion, Eagle, Zebra, Giraffe: Конкретні класи тварин.
- Ungulate: Клас для копитних, який об'єднує зебр та жирафів для їх спільногого утримання

```
task_4_1 > 📄 Animal.java
1   import java.io.Serializable;
2
3   abstract class Animal implements Serializable {
4       private String name;
5       public Animal(String name) { this.name = name; }
6       public String getName() { return name; }
7   }
8
9   class Mammal extends Animal { public Mammal(String name) { super(name); } }
10  class Bird extends Animal { public Bird(String name) { super(name); } }
11
12  class Lion extends Mammal { public Lion(String name) { super(name); } }
13  class Ungulate extends Mammal { public Ungulate(String name) { super(name); } }
14
15  class Zebra extends Ungulate { public Zebra(String name) { super(name); } }
16  class Giraffe extends Ungulate { public Giraffe(String name) { super(name); } }
17  class Eagle extends Bird { public Eagle(String name) { super(name); } }
```

Файл Cage.java містить узагальнений клас Cage<T extends Animal>, який відповідає за логіку окремого вольєра

- Обмеження типів: Використано Generics, щоб гарантувати, що у вольєрі будуть лише дозволені види тварин
- Місткість: Реалізовано змінні та методи для контролю максимальної та поточної кількості місць.
- addAnimal(): Метод для додавання тварини, який ініціює виключну ситуацію, якщо вольєр заповнений.
- removeAnimal(): Метод для вилучення тварини з перевіркою її наявності у вольєрі (ініціює Exception, якщо тварини немає).

```

task_4_1 > Cage.java
1 import java.util.*;
2 import java.io.Serializable;
3
4 public class Cage<T extends Animal> implements Serializable {
5     private int maxCapacity;
6     private List<T> animals = new ArrayList<>();
7
8     public Cage(int maxCapacity) {
9         this.maxCapacity = maxCapacity;
10    }
11
12     public void addAnimal(T animal) throws Exception {
13         if (animals.size() >= maxCapacity) {
14             throw new Exception("Вольєр переповнений!");
15         }
16         animals.add(animal);
17     }
18
19     public void removeAnimal(T animal) throws Exception {
20         if (!animals.contains(animal)) {
21             throw new Exception("Цієї тварини немає у вольєрі!");
22         }
23         animals.remove(animal);
24     }
25
26     public int getMaxCapacity() { return maxCapacity; }
27     public int getOccupiedPlaces() { return animals.size(); }
28     public List<T> getAnimals() { return animals; }
29 }

```

У файлі Zoo.java реалізовано логіку управління всіма вольєрами зоопарку:

- Список вольєрів: Використано List<Cage<? extends Animal>>, що дозволяє зберігати в одному списку вольєри з різними типами тварин (використання Wildcards).
- addCage(): Метод для додавання нового вольєра до зоопарку.
- getCountOfAnimals(): Функція, яка ітерує всі вольєри та підраховує загальну кількість тварин у зоопарку.

```

task_4_1 > Zoo.java
1 import java.util.*;
2
3 public class Zoo {
4     public List<Cage<? extends Animal>> cages = new ArrayList<>();
5
6     public void addCage(Cage<? extends Animal> cage) {
7         cages.add(cage);
8     }
9

```

```

9
10    public int getCountOfAnimals() {
11        int total = 0;
12        for (Cage<? extends Animal> cage : cages) {
13            total += cage.getOccupiedPlaces();
14        }
15        return total;
16    }
17 }

```

Файл ZooManager.java відповідає за роботу з даними та виконання вимог щодо збереження інформації:

- Збереження: Використано ObjectOutputStream для запису об'єктів (тварин) у файл.
- Читання: Використано ObjectInputStream для відновлення даних із файлу.

```

task_4_1 > ZooManager.java
1  import java.io.*;
2  import java.util.List;
3
4  public class ZooManager {
5      public static void saveAnimalsToFile(List<Animal> animals, String filename) throws IOException {
6          try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filename))) {
7              oos.writeObject(animals);
8          }
9      }
10
11     @SuppressWarnings("unchecked")
12     public static List<Animal> loadAnimalsFromFile(String filename) throws IOException, ClassNotFoundException {
13         try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filename))) {
14             return (List<Animal>) ois.readObject();
15         }
16     }
17 }

```

Файл ZooTest.java містить в собі клас для проведення модульних тестів та демонстрації роботи програми:

- Тестування наповнення: Перевірка успішного додавання тварин у відповідні вольєри.
- Тестування виключчних ситуацій: Створення сценаріїв, де вольєр переповнюється, для перевірки коректності обробки помилок

```

task_4_1 > ZooTest.java
1  public class ZooTest {
2      public static void main(String[] args) {
3          try {
4              // Створюємо вольєри
5              Cage<Lion> lionCage = new Cage<>(2);
6              Cage<Ungulate> ungulateCage = new Cage<>(3);
7              Cage<Bird> birdCage = new Cage<>(5);
8
9              // Додаємо тварин
10             lionCage.addAnimal(new Lion("Сімба"));
11             ungulateCage.addAnimal(new Zebra("Марті"));

```

```
12     ungulateCage.addAnimal(new Giraffe("Мелман"));
13     birdCage.addAnimal(new Eagle("Зевс"));
14
15     // Робота з класом Zoo
16     Zoo myZoo = new Zoo();
17     myZoo.addCage(lionCage);
18     myZoo.addCage(ungulateCage);
19     myZoo.addCage(birdCage);
20
21     System.out.println("Всього тварин у зоопарку: " + myZoo.getCountOfAnimals());
22
23     // Перевірка виключної ситуації при переповненні
24     lionCage.addAnimal(new Lion("Муфаса"));
25     try {
26         lionCage.addAnimal(new Lion("Скар"));
27     } catch (Exception e) {
28         System.out.println("Очікувана помилка: " + e.getMessage());
29     }
30
31 } catch (Exception e) {
32     e.printStackTrace();
33 }
34 }
35 }
```

Результати компіляції та демонстрація роботи програми наведені нижче

```
Caused by: java.lang.StackOverflowError
● ruszlanapavliuk@POL02-0349 task_4_1 % java ZooTest
Всього тварин у зоопарку: 4
Очікувана помилка: Вольєр переповнений!
```