Manjula Nannuri

Day 20_Assignment

Task 1: Generics and Type Safety

Create a generic Pair class that holds two objects of different types, and write a method to return a reversed version of the pair.

```
public class Pair<T, U> {
    private T first;
    private U second;
    public Pair(T first, U second) {
        this.first = first;
        this.second = second;
    }
    public T getFirst() {
        return first;
    }
    public void setFirst(T first) {
        this.first = first;
    }
    public U getSecond() {
        return second;
    }
    public void setSecond(U second) {
        this.second = second;
    }
    public Pair<U, T> reverse() {
        return new Pair<>(second, first);

    }

    public String toString() {

        return "Pair{" + "first=" + first + ", second=" + second +'}';

    }

    public static void main(String[] args) {

     Pair<Integer, String> originalPair = new Pair<>(1, "one");

        System.out.println("Original Pair: " + originalPair);

        Pair<String, Integer> reversedPair = originalPair.reverse();

        System.out.println("Reversed Pair: " + reversedPair);

    }

}
```

Output :

```
Original Pair: Pair{first=1, second=one}

Reversed Pair: Pair{first=one, second=1}
```

Task 2: Generic Classes and Methods

Implement a generic method that swaps the positions of two elements in an array, regardless of their type, and demonstrate its usage with different object types.

```
public class ArrayUtils {

 public static <T> void swap(T[] array, int index1, int index2) {

        if (array == null || index1 < 0 || index2 < 0 || index1 >= array.length || index2 >= array.length) {

            throw new IllegalArgumentException("Invalid index or array");

        }

        T temp = array[index1];
        array[index1] = array[index2];

        array[index2] = temp;

    }

    public static <T> void printArray(T[] array) {

        for (T element : array) {

System.out.print(element + " ");

        }

System.out.println();

    }



    public static void main(String[] args) {

Integer[] intArray = {1, 2, 3, 4, 5};

System.out.println("Original Integer array:");

printArray(intArray);

swap(intArray, 1, 3);

System.out.println("Integer array after swap:");

printArray(intArray);



String[] strArray = {"apple", "banana", "mango", "date"};

System.out.println("Original String array:");

printArray(strArray);

swap(strArray, 0, 2);

System.out.println("String array after swap:");

printArray(strArray);

Double[] doubleArray = {1.1, 2.2, 3.3, 4.4};
```

```
System.out.println("Original Double array:");
printArray(doubleArray);
swap(doubleArray, 1, 3);
System.out.println("Double array after swap:");
printArray(doubleArray);
    }
}
```

Output:

```
Original Integer array:
1 2 3 4 5
Integer array after swap:
1 4 3 2 5
Original String array:
apple banana mango  date
String array after swap:
mango banana apple date
Original Double array:
1.1 2.2 3.3 4.4
Double array after swap:
1.1 4.4 3.3 2.2
```

Task 3: Reflection API

Use reflection to inspect a class's methods, fields, and constructors, and modify the access level of a private field, setting its value during runtime.

Here iam taking a person as a class:

```
public class Person {
    private String name;
    private int age;

    public Person() {
    }
    public Person(String name, int age) {
        this.name = name;
       this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public intgetAge() {
        return age;
    }

    public void setAge(int age) {
this.age = age;
```

```
    }
    private void printPrivateMessage() {
System.out.println("This is a private method.");
    }
}
```

ReflectionExample Class:

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
public class ReflectionExample {
    public static void main(String[] args) {
        try {
  Class<?>personClass = Class.forName("Person");
System.out.println("Constructors:");
            Constructor<?>[] constructors = personClass.getConstructors();
            for (Constructor<?>constructor : constructors) {
System.out.println(constructor);
            }
System.out.println("\nMethods:");
Method[] methods = personClass.getMethods();
            for (Method method : methods) {
System.out.println(method);
            }
System.out.println("\nFields:");
Field[] fields = personClass.getDeclaredFields();
```

```
        for (Field field : fields) {
System.out.println(field);
        }
        Object personInstance = personClass.getConstructor().newInstance();
        Field nameField = personClass.getDeclaredField("name");
        nameField.setAccessible(true);
        nameField.set(personInstance, "John Doe");
        Method getNameMethod = personClass.getMethod("getName");
        String name = (String) getNameMethod.invoke(personInstance);
         System.out.println("\nModified name field: " + name);
        Field ageField = personClass.getDeclaredField("age");
       ageField.setAccessible(true);
      ageField.setInt(personInstance, 30);
        Method getAgeMethod = personClass.getMethod("getAge");
       int age = (int) getAgeMethod.invoke(personInstance);
       System.out.println("Modified age field: " + age);
      Method privateMethod = personClass.getDeclaredMethod("printPrivateMessage");
    privateMethod.setAccessible(true);
    privateMethod.invoke(personInstance);
     } catch (Exception e) {
    e.printStackTrace();
     }
   }
}
```

Output:

```
Constructors:
public Person()
public Person(java.lang.String,int)

Methods:
public java.lang.StringPerson.getName()
public void Person.setName(java.lang.String)
public intPerson.getAge()
public void Person.setAge(int)
public final void java.lang.Object.wait(long,int) throws java.lang.InterruptedException
public final void java.lang.Object.wait() throws java.lang.InterruptedException
public final native void java.lang.Object.wait(long) throws java.lang.InterruptedException
public booleanjava.lang.Object.equals(java.lang.Object)
public java.lang.Stringjava.lang.Object.toString()
public native intjava.lang.Object.hashCode()
public final native java.lang.Class<?>java.lang.Object.getClass()
public final native void java.lang.Object.notify()
public final native void java.lang.Object.notifyAll()

Fields:
private java.lang.String Person.name
private intPerson.age

Modified name field: John Doe
Modified age field: 30
```

This is a private method.

Task 4: Lambda Expressions

Implement a Comparator for a Person class using a lambda expression, and sort a list of Person objects by their age..

```
public class Person {
    private String name;
    private int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public intgetAge() {
        return age;
    }
    public String toString() {
        return "Person{name='" + name + "', age=" + age + "}";
    }
}

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

public class LambdaComparatorExample {

    public static void main(String[] args) {

    List<Person> persons = new ArrayList<>();

persons.add(new Person("Amit", 30));

persons.add(new Person("Bobby", 25));

persons.add(new Person("Charmi", 35));

persons.add(new Person("Devi", 20));

persons.sort((Person p1, Person p2) ->Integer.compare(p1.getAge(), p2.getAge()));

persons.forEach(System.out::println);

    }

}
```

Output:

When you run the LambdaComparatorExample class, you should see output similar to:

```
Person{name='Devi', age=20}

Person{name='Bobby', age=25}

Person{name='Amit', age=30}

Person{name='Charmi', age=35}
```

Task 5: Functional Interfaces

Create a method that accepts functions as parameters using Predicate, Function, Consumer, and Supplier interfaces to operate on a Person object.

```
public class Person {
    private String name;

    private int age;

    public Person(String name, int age) {

        this.name = name;

       this.age = age;

    }

    public String getName() {

        return name;

    }

    public intgetAge() {

        return age;

    }

    public void setName(String name) {

        this.name = name;

    }

    public void setAge(int age) {

this.age = age;

    }

    public String toString() {

        return "Person{name='" + name + "', age=" + age + '}';

    }

}

import java.util.function.Consumer;

import java.util.function.Function;

import java.util.function.Predicate;
import java.util.function.Supplier;

public class PersonOperations {

  public static void operateOnPerson(

      Predicate<Person> predicate,

      Function<Person, String> function,

      Consumer<Person> consumer,

      Supplier<Person> supplier) {

    Person person = supplier.get();

System.out.println("Initial person: " + person);

    if (predicate.test(person)) {

System.out.println("Predicate test passed");

      String result = function.apply(person);

System.out.println("Function result: " + result);

    } else {

System.out.println("Predicate test failed");

    }

consumer.accept(person);

System.out.println("Person after consumer: " + person);

  }

  public static void main(String[] args) {

    Predicate<Person>isAdult = p ->p.getAge() >= 18;

    Function<Person, String>getName = Person::getName;

    Consumer<Person>makeOlder = p ->p.setAge(p.getAge() + 10);

    Supplier<Person>personSupplier = () -> new Person("John Doe", 55);

   operateOnPerson(isAdult, getName, makeOlder, personSupplier);
```

```
    }

}
```

Output:

```
  Initial person: Person{name='John Doe', age=55}

Predicate test passed

Function result: John Doe

Person after consumer: Person{name='John Doe', age=65}
```