DAY 19 JAVA ASSIGNMENT

Day 19:

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.

Generics.java

```java
package assignment;

public class Generics<T, U> {
    private T first;
    private U second;
    public Generics(T first, U second) {
        this.first = first;
        this.second = second;
    }
    public T getFirst() {
        return first;
    }
    public U getSecond() {
        return second;
    }
    public Generics<U, T> reverse() {
        return new Generics<>(second, first);
    }
    public static void main(String[] args) {
    Generics<String, Integer> pair = new Generics<>("Hello", 123);
    System.out.println("Original Pair: " + pair.getFirst() + ", " + pair.gets
    Generics<Integer, String> reversedPair = pair.reverse();
    System.out.println("Reversed Pair: " + reversedPair.getFirst() + ", " + r
    }
}
```

Console

```
<terminated> Generics [Java Application]
Original Pair: Hello, 123
Reversed Pair: 123, Hello
```

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.

GenericArray.java

```java
package assignment;

public class GenericArray {
    public static <T> void swapElements(T[] array, int index1, int index2) {
        if (index1 < 0 || index1 >= array.length || index2 < 0 || index2 >= array.length) {
            throw new IllegalArgumentException("Invalid indices");
        }
        T temp = array[index1];
        array[index1] = array[index2];
        array[index2] = temp;
    }

    public static void main(String[] args) {
    String[] stringArray = {"apple", "banana", "cherry"};
    System.out.println("Original String array: " + java.util.Arrays.toString(stringArray));
    swapElements(stringArray, 0, 2);
    System.out.println("After swapping: " + java.util.Arrays.toString(stringArray));
    Integer[] intArray = {1, 2, 3, 4, 5};
    System.out.println("\nOriginal Integer array: " + java.util.Arrays.toString(intArray));
    swapElements(intArray, 1, 3);
    System.out.println("After swapping: " + java.util.Arrays.toString(intArray));
    Character[] charArray = {'a', 'b', 'c', 'd'};
    System.out.println("\nOriginal Character array: " + java.util.Arrays.toString(charArray));
    swapElements(charArray, 0, 3);
    System.out.println("After swapping: " + java.util.Arrays.toString(charArray));
    }
}
```

Console

```
<terminated> GenericArray [Java Application] C:\Program Files\Ja
Original String array: [apple, banana, cherry]
After swapping: [cherry, banana, apple]

Original Integer array: [1, 2, 3, 4, 5]
After swapping: [1, 4, 3, 2, 5]

Original Character array: [a, b, c, d]
After swapping: [d, b, c, a]
```

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

GenericArray.java | *Reflection.java

```java
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
class MyClass {
    private int privateField;
    public MyClass(int privateField) {
        this.privateField = privateField;
    }
    private void privateMethod() {
        System.out.println("Private method called");
    }
}
public class Reflection {
    public static void main(String[] args) throws Exception {
        Class<?> myClass = MyClass.class;
        System.out.println("Constructors:");
        Constructor<?>[] constructors = myClass.getDeclaredConstructors();
        for (Constructor<?> constructor : constructors) {
            System.out.println(constructor);
        }
        System.out.println("\nFields:");
        Field[] fields = myClass.getDeclaredFields();
        for (Field field : fields) {
            System.out.println(field);
        }
        System.out.println("\nAccessing private field:");
        MyClass obj = new MyClass(10);
        Field privateField = myClass.getDeclaredField("privateField");
        privateField.setAccessible(true);
        int fieldValue = (int) privateField.get(obj);
        System.out.println("Original value of privateField: " + fieldValue);
        privateField.set(obj, 20);
        fieldValue = (int) privateField.get(obj);
        System.out.println("Modified value of privateField: " + fieldValue);
        System.out.println("\nMethods:");
        Method[] methods = myClass.getDeclaredMethods();
        for (Method method : methods) {
            System.out.println(method);
        }
        System.out.println("\nInvoking private method:");
        Method privateMethod = myClass.getDeclaredMethod("privateMethod");
        privateMethod.setAccessible(true);
        privateMethod.invoke(obj);
```

Console

```
<terminated> Reflection [Java Application] C:\Program Files\Java\j
Constructors:
public assignment.MyClass(int)

Fields:
private int assignment.MyClass.privateField

Accessing private field:
Original value of privateField: 10
Modified value of privateField: 20

Methods:
private void assignment.MyClass.privateMethod()

Invoking private method:
Private method called
```

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..

LambdaExpressions.java

```java
package assignment;

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

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 int getAge() {
        return age;
    }
    @Override
    public String toString() {
        return "Person{" +"name='" + name + '\'' +", age=" + age +'}';
    }
}
public class LambdaExpressions {
    public static void main(String[] args) {
        List<Person> people = new ArrayList<>();
        people.add(new Person("Hope", 30));
        people.add(new Person("Bob", 25));
        people.add(new Person("Chathy", 35));
        Collections.sort(people, Comparator.comparingInt(Person::getAge));
        System.out.println("Sorted by age:");
        for (Person person : people) {
            System.out.println(person);
        }
    }
}
```

Console

```
<terminated> LambdaExpressions [Java Appl
Sorted by age:
Person{name='Bob', age=25}
Person{name='Hope', age=30}
Person{name='Chathy', 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.

Functionalinterface.java

```java
package assignment;

import java.util.function.Consumer;
import java.util.function.Function;
import java.util.function.Predicate;
import java.util.function.Supplier;
class PersonA {
    private String name;
    private int age;

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

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
    public String toString() {
        return "Person{" +"name='" + name + '\'' +", age=" + age +'}';
    }
}
public class Functionalinterface {
    public static void processPerson(PersonA person,
                                     Predicate<PersonA> predicate,
                                     Function<PersonA, String> function,
                                     Consumer<String> consumer,
                                     Supplier<Integer> supplier) {
        if (predicate.test(person)) {
            String result = function.apply(person);
            consumer.accept(result);
            int suppliedValue = supplier.get();
            System.out.println("Supplied value: " + suppliedValue);
        } else {
            System.out.println("Predicate condition not met for " + person.getName());
        }
    }
```

Console

```
<terminated> Functionalinterface [Java
Alice
Supplied value: 30
```

```java
    }
    public static void main(String[] args) {
        PersonA person = new PersonA("Alice", 30);
        Predicate<PersonA> isAdult = p -> p.getAge() >= 18;
        Function<PersonA, String> getNameFunction = PersonA::getName;
        Consumer<String> printNameConsumer = System.out::println;
        Supplier<Integer> ageSupplier = person::getAge;
        processPerson(person, isAdult, getNameFunction, printNameConsumer, ageSupplier);
    }
}
```