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.

ANS:

```
public class GenericAndTypeSafety<U, V> {

    private final U first;

    private final V second;



    public GenericAndTypeSafety(U first, V second) {

        this.first = first;

        this.second = second;

    }



    public U getFirst() {

        return first;

    }



    public V getSecond() {

        return second;

    }



    public GenericAndTypeSafety<V, U> reverse() {

        return new GenericAndTypeSafety<>(second, first);

    }
    @Override

    public String toString() {

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

    }


    public static void main(String[] args) {

        GenericAndTypeSafety<Integer, String> pair = new GenericAndTypeSafety<>(1, "one");

        System.out.println("Original pair: " + pair);

        GenericAndTypeSafety<String, Integer> reversedPair = pair.reverse();

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

    }

}

//code_by_RUBY
```

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.

ANS:

```
import java.util.Arrays;


public class GenericClassesAndMethods {


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

        T temp = array[index1];

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

    }


    public static void main(String[] args) {

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

        swap(intArray, 0, 4);

        System.out.println("Swapped Integer array: " + Arrays.toString(intArray));


        String[] strArray = {"a", "b", "c", "d", "e"};

        swap(strArray, 1, 3);

        System.out.println("Swapped String array: " + Arrays.toString(strArray));

    }

}

//code_by_RUBY
```

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.

ANS:

```
import java.lang.reflect.Constructor;

import java.lang.reflect.Field;

import java.lang.reflect.Method;


class Person {

    private String name;
    private int age;

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

    public void printDetails() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}

public class Reflection {

    public static void main(String[] args) throws Exception {
        Class<Person> personClass = Person.class;



        Method[] methods = personClass.getDeclaredMethods();
        System.out.println("Methods:");
        for (Method method : methods) {
            System.out.println(method.getName());
        }
        Field[] fields = personClass.getDeclaredFields();

        System.out.println("\nFields:");

        for (Field field : fields) {

            System.out.println(field.getName());

        }



        Constructor<?>[] constructors = personClass.getDeclaredConstructors();

        System.out.println("\nConstructors:");

        for (Constructor<?> constructor : constructors) {

            System.out.println(constructor);

        }



        Person person = new Person("John", 30);

        Field ageField = personClass.getDeclaredField("age");

        ageField.setAccessible(true);

        ageField.set(person, 35);

        person.printDetails();

    }

}


//code_by_RUBY
```

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..

ANS:

```
import java.util.ArrayList;

import java.util.Comparator;

import java.util.List;


class Person1 {

    String name;

    int age;


    Person1(String name, int age) {

        this.name = name;

        this.age = age;

    }


    @Override

    public String toString() {

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

    }

}


public class Lambda {


    public static void main(String[] args) {
        List<Person1> people = new ArrayList<>();

        people.add(new Person1("Alice", 34));

        people.add(new Person1("Bob", 25));

        people.add(new Person1("Charlie", 29));


        people.sort(Comparator.comparingInt(p -> p.age));


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

    }

}

//code_by_RUBY
```

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.

ANS:

```
import java.util.function.Consumer;

import java.util.function.Function;

import java.util.function.Predicate;

import java.util.function.Supplier;


class Person2 {

    String name;

    int age;
    Person2(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{name='" + name + "', age=" + age + '}';
    }
}

public class FunctionalInterface {

    public static void processPerson(Person2 person, Predicate<Person2> predicate,
                                     Function<Person2, String> function, Consumer<String> consumer,
                                     Supplier<Person2> supplier) {

        if (predicate.test(person)) {
            String result = function.apply(person);
            consumer.accept(result);
        }

        Person2 newPerson = supplier.get();
        System.out.println("New Person: " + newPerson);
    }
    public static void main(String[] args) {
        Person2 person = new Person2("Alice", 30);


        Predicate<Person2> agePredicate = p -> p.age > 25;
        Function<Person2, String> nameFunction = p -> "Name: " + p.name;
        Consumer<String> printConsumer = System.out::println;
        Supplier<Person2> personSupplier = () -> new Person2("Bob", 40);


        processPerson(person, agePredicate, nameFunction, printConsumer, personSupplier);
    }
}
//code_by_RUBY
```