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.

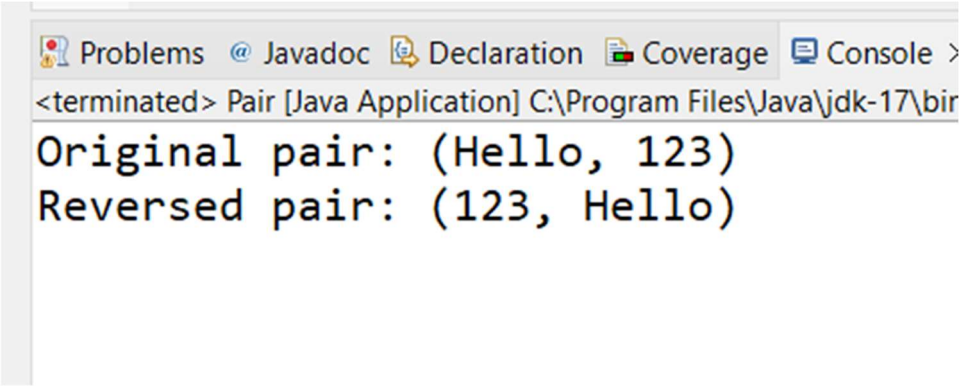
```java
package com.assignment.day19;

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 U getSecond() {
        return second;
    }
    public Pair<U, T> reverse() {
        return new Pair<>(second, first);
    }
    @Override
    public String toString() {
        return "(" + first + ", " + second + ")";
    }
    public static void main(String[] args) {
        Pair<String, Integer> pair = new Pair<>("Hello", 123);
        System.out.println("Original pair: " + pair);
        System.out.println("Reversed pair: " + pair.reverse());
    }
}
```

Output:



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.

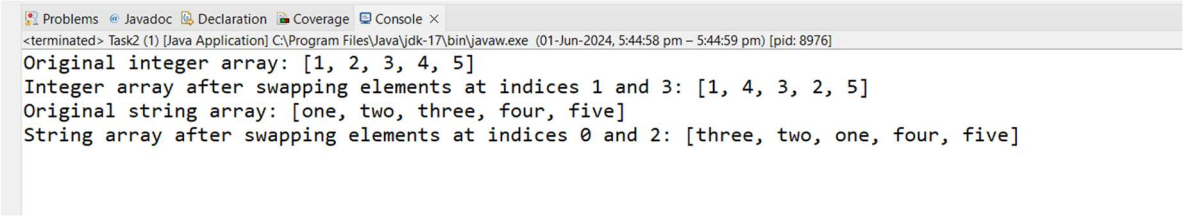
```java
package com.assignment.day19;
import java.util.Arrays;

public class Task2 {
    public static <T> void swap(T[] array, int i, int j) {
        T temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    public static void main(String[] args) {
        Integer[] intArray = {1, 2, 3, 4, 5};
        System.out.println("Original integer array: " + Arrays.toString(intArray));
        swap(intArray, 1, 3);
        System.out.println("Integer array after swapping elements at indices 1 and 3: " + Arrays.toString(intArray));

        String[] strArray = {"one", "two", "three", "four", "five"};
        System.out.println("Original string array: " + Arrays.toString(strArray));
        swap(strArray, 0, 2);
        System.out.println("String array after swapping elements at indices 0 and 2: " + Arrays.toString(strArray));
    }
}
```

Output:



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

```java
package com.assignment.day19;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
public class Task3 {
    private int number1;
    protected int number2;
    public String name;
    public Task3(int number1, int number2, String name) {
        this.number1 = number1;
        this.number2 = number2;
        this.name = name;
    }
    public static void main(String[] args) throws Exception {
        // Inspecting class fields
        Class<Task3> clazz = Task3.class;
        System.out.println("Fields:");
        for (Field field : clazz.getDeclaredFields()) {
            System.out.println(field.getName() + ", Access Modifier: " + Modifier.toString(field.getModifiers()));
        }
        Task3 instance = new Task3(0, 0, "");
        Field privateField = clazz.getDeclaredField("number1");
        privateField.setAccessible(true); // Allow access to private field
        privateField.setInt(instance, 42); // Set value of private field
        System.out.println("\nModified number1 field value: " + instance.number1);
        Task3 instance2 = new Task3(0, 0, "");
        Field protectedField = clazz.getDeclaredField("number2");
        protectedField.setAccessible(true); // Allow access to private field
        protectedField.setInt(instance, 52); // Set value of private field
        System.out.println("\nModified number2 field value: " + instance.number2);
    }
}
```

Output:

```
<terminated> Task3 (1) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw
Fields:
number1, Access Modifier: private
number2, Access Modifier: protected
name, Access Modifier: public

Modified number1 field value: 42

Modified number2 field value: 52
```

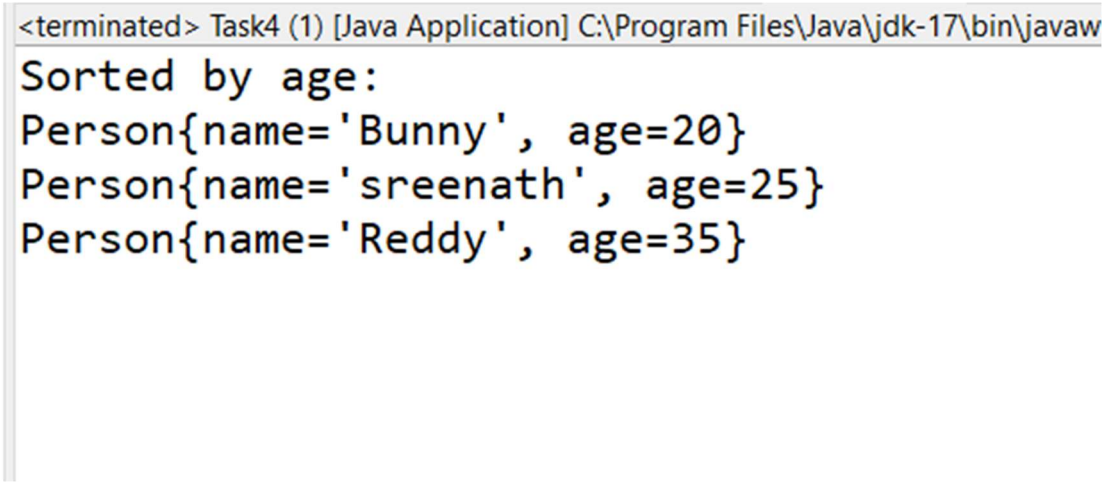
Task 4: Lambda Expressions
Implement a Comparator for a Person class using a lambda expression, and sort a list of Person objects by their age.

```
package com.assignment.day19;
import java.util.ArrayList;
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 Task4 {
    public static void main(String[] args) {
        // Create a list of Person objects
        List<Person> personList = new ArrayList<>();
        personList.add(new Person("Bunny", 20));
        personList.add(new Person("sreenath", 25));
```

```
        List<Person> personList = new ArrayList<>();
        personList.add(new Person("Bunny", 20));
        personList.add(new Person("sreenath", 25));
        personList.add(new Person("Reddy", 35));
        // Sort the list by age using a Comparator with lambda expression
        personList.sort(Comparator.comparing(Person::getAge));
        // Print the sorted list
        System.out.println("Sorted by age:");
        personList.forEach(System.out::println);
    }
}
```

Output:



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.

```java
package com.assignment.day19;

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

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

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

```java
public class Task5 {
    public static void main(String[] args) throws Exception {

        System.out.println(" Predicate to check if person is adult");

        Predicate<Persons> isAdult = person -> person.age >= 18;

        System.out.println(" Function to get person's name");
        Function<Persons, String> getName = person -> person.name;

        System.out.println(" Consumer to print person");
        Consumer<Persons> printPerson = person -> System.out.println(person);
        System.out.println("Supplier to create a new person");
        Supplier<Persons> createPerson = () -> new Persons("Sreenath", 24);

        Persons person = createPerson.get();

        // Use the consumer to print the person
        printPerson.accept(person);

        // Use the function to get the person's name
        String name = getName.apply(person);
        System.out.println("Person's name: " + name);

        // Use the predicate to check if the person is an adult
        boolean adult = isAdult.test(person);
        System.out.println("Is person an adult? " + adult);
    }
}
```

Output:

```
<terminated> Task5 (1) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (01-Jun-2(
 Predicate to check if person is adult
 Function to get person's name
 Consumer to print person
Supplier to create a new person
Persons{name='Sreenath', age=24}
Person's name: Sreenath
Is person an adult? true
```