Day 27 Assignment

Name: Mehul Anjikhane Email: mehulanjikhane13@gmail.com

**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.**

**package** generics;

**public** **class** Pair<T1, T2> {

**private** **final** T1 firstValue;

**private** **final** T2 secondValue;

**public** Pair(T1 firstValue, T2 secondValue) {

**this**.firstValue = firstValue;

**this**.secondValue = secondValue;

}

**public** T1 getInitialValue() {

**return** firstValue;

}

**public** T2 getSecondaryValue() {

**return** secondValue;

}

**public** Pair<T2, T1> getSwappedHolder() {

**return** **new** Pair<>(secondValue, firstValue);

}

**public** **static** **void** main(String[] args) {

Pair<String, Integer> originalholder = **new** Pair<> ("John", 30);

System.***out***.println("Original pair: (" + originalholder.getInitialValue() + ", " + originalholder.getSecondaryValue() + ")");

Pair<Integer, String> swappedHolder = originalholder.getSwappedHolder();

System.***out***.println("Reversed pair: (" + swappedHolder.getInitialValue() + ", " + swappedHolder.getSecondaryValue() + ")");

}

}

**Output:**

Original pair: (John, 30)

Reversed pair: (30, John)

**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.**

**package** generics;

**import** java.util.Arrays;

**public** **class** SwapElements {

**public** **static** <T> **void** switchElements(T[] arr, **int** index1, **int** index2) {

T temp = arr[index1];

arr[index1] = arr[index2];

arr[index2] = temp;

}

**public** **static** **void** main(String[] args) {

String[] fruits = { "apple", "banana", "cherry" };

System.***out***.println("Original fruits: " + String.*join*(", ", fruits));

*switchElements*(fruits, 0, 2);

System.***out***.println("Swapped fruits: " + String.*join*(", ", fruits));

Double[] weights = { 5.2, 7.1, 4.8 System.***out***.println("Original weights: " + String.*join*(", ", Arrays.*toString*(weights)));

*switchElements*(weights, 1, 2);

System.***out***.println("Swapped weights: " + String.*join*(", ", Arrays.*toString*(weights)));

}

}

**Output:**

Original fruits: apple, banana, cherry

Swapped fruits: cherry, banana, apple

Original weights: [5.2, 7.1, 4.8]

Swapped weights: [5.2, 4.8, 7.1]

**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**

**package** reflectionapi;

import java.lang.reflect.Constructor;

import java.lang.reflect.Field;

import java.lang.reflect.Method;

**class** Book {

**private** String title;

**private** String author;

**public** Book(String title, String author) {

**this**.title = title;

**this**.author = author;

}

**public** String getTitle() {

**return** title;

}

**public** **void** setTitle(String title) {

**this**.title = title;

}

**public** String getAuthor() {

**return** author;

}

**public** **void** setAuthor(String author) {

**this**.author = author;

}

**private** **void** printDetails() {

System.***out***.println("Book title: " + title + ", Author: " + author);

}

}

public class ReflectionExample {

public static void main(String[] args) {

try {

// Assuming Book.class is in the same package (reflectionapi)

Class<?> bookClass = Class.forName("reflectionapi.Book");

// Inspect methods

System.out.println("\*\* Methods of " + bookClass.getSimpleName() + " \*\*");

Method[] methods = bookClass.getDeclaredMethods();

for (Method method : methods) {

System.out.println(method);

}

// Inspect fields

System.out.println("\n\*\* Fields of " + bookClass.getSimpleName() + " \*\*");

Field[] fields = bookClass.getDeclaredFields();

for (Field field : fields) {

System.out.println(field);

}

// Inspect constructors

System.out.println("\n\*\* Constructors of " + bookClass.getSimpleName() + " \*\*");

Constructor<?>[] constructors = bookClass.getConstructors();

for (Constructor<?> constructor : constructors) {

System.out.println(constructor);

}

// Modify private fields and invoke private method

Object bookInstance = bookClass.getConstructor(String.class, String.class).newInstance("The Lord of the Rings", "J.R.R. Tolkien"); // Example constructor with arguments

Field titleField = bookClass.getDeclaredField("title");

titleField.setAccessible(true);

titleField.set(bookInstance, "The Hitchhiker's Guide to the Galaxy");

Field authorField = bookClass.getDeclaredField("author");

authorField.setAccessible(true);

authorField.set(bookInstance, "Douglas Adams");

Method printDetailsMethod = bookClass.getDeclaredMethod("printDetails");

printDetailsMethod.setAccessible(true);

printDetailsMethod.invoke(bookInstance);

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Output:**

\*\* Methods of Book \*\*

public java.lang.String reflectionapi.Book.getAuthor()

public void reflectionapi.Book.setAuthor(java.lang.String)

public java.lang.String reflectionapi.Book.getTitle()

public void reflectionapi.Book.setTitle(java.lang.String)

private void reflectionapi.Book.printDetails()

\*\* Fields of Book \*\*

private java.lang.String reflectionapi.Book.title

private java.lang.String reflectionapi.Book.author

\*\* Constructors of Book \*\*

public reflectionapi.Book(java.lang.String,java.lang.String)

Book title: The Hitchhiker's Guide to the Galaxy, Author: Douglas Adams

**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 lambda\_expression;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

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 int getAge() {

return age;

}

@Override

public String toString() {

return name + " (" + age + ")";

}

public static void main(String[] args) {

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

people.add(new Person("Mehul", 23));

people.add(new Person("Nikhil", 19));

people.add(new Person("Tukaram", 51));

System.out.println("Before sorting: " + people);

// Using lambda expression to sort by age

Comparator<Person> ageComparator = (p1, p2) -> p1.getAge() -

p2.getAge();

Collections.sort(people, ageComparator);

System.out.println("After sorting by age: " + people);

}

}

**Output:**

Before sorting: [Mehul (23), Nikhil (19), Tukaram (51)]

After sorting by age: [Nikhil (19), Mehul (23), Tukaram (51)]

**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.**

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 operateOnPerson1(Person1 person, Predicate<Person1>

predicate, Function<Person1, String> function,

Consumer<Person1> consumer, Supplier<Person1> supplier) {

// Predicate to check a condition on the person

if (predicate.test(person)) {

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

} else {

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

}

// Function to apply an operation and return a result

String result = function.apply(person);

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

// Consumer to perform an operation on the person

consumer.accept(person);

// Supplier to provide a new person object

Person1 newPerson = supplier.get();

System.out.println("Supplier provided: " + newPerson);

}

public static void main(String[] args) {

Person1 person = new Person1("Vikram", 20);

Predicate<Person1> ageCheck = p -> p.getAge() > 25;

Function<Person1, String> nameExtractor = Person1::getName;

Consumer<Person1> namePrinter = p -> System.out.println("Person's name: " + p.getName());

Supplier<Person1> personSupplier = () -> new Person1("Yash", 17);

operateOnPerson1(person, ageCheck, nameExtractor, namePrinter,

personSupplier);

}

}

**Output:**

Predicate test failed.

Function result: Vikram

Person's name: Vikram

Supplier provided: Yash (17)