**Digvijay Thakare Day\_19 Core Java**

[**digvijaythakare2017@gmail.com**](mailto:digvijaythakare2017@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.**

**Solution-**

package com.epwiproday\_19;

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<>("Day\_19Assignment", 456789);

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

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

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

}

}

**Output-**

Original Pair: (Day\_19Assignment, 456789)

Reversed Pair: (456789, Day\_19Assignment)

**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**

**Solution-**

package com.epwiproday\_19;

import java.util.Arrays;

public class ArrayUtills {

// Generic method to swap elements in an array

public static <T> void swap(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) {

// Demonstrating usage with different object types

// Integer array

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

System.***out***.println("Original Integer Array: " + Arrays.*toString*(intArray));

*swap*(intArray, 1, 3);

System.***out***.println("After Swapping: " + Arrays.*toString*(intArray));

// String array

String[] stringArray = {"apple", "banana", "orange"};

System.***out***.println("Original String Array: " + Arrays.*toString*(stringArray));

*swap*(stringArray, 0, 2);

System.***out***.println("After Swapping: " + Arrays.*toString*(stringArray));

// Character array

Character[] charArray = {'a', 'b', 'c', 'd'};

System.***out***.println("Original Character Array: " + Arrays.*toString*(charArray));

*swap*(charArray, 1, 3);

System.***out***.println("After Swapping: " + Arrays.*toString*(charArray));

}

}

**Output-**

Original Integer Array: [1, 2, 3, 4, 5]

After Swapping: [1, 4, 3, 2, 5]

Original String Array: [apple, banana, orange]

After Swapping: [orange, banana, apple]

Original Character Array: [a, b, c, d]

After Swapping: [a, d, c, b]

**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**

**Solution-**

package com.epwiproday\_19;

import java.lang.reflect.Constructor;

import java.lang.reflect.Field;

import java.lang.reflect.Method;

public class ReflectionExample {

private String privateField;

public ReflectionExample(String privateField) {

this.privateField = privateField;

}

public void publicMethod() {

System.***out***.println("Inside publicMethod()");

}

private void privateMethod() {

System.***out***.println("Inside privateMethod()");

}

public static void main(String[] args) throws Exception {

// Inspecting methods, fields, and constructors of the class

Class<?> clazz = ReflectionExample.class;

// Getting declared fields

System.***out***.println("Declared Fields:");

Field[] fields = clazz.getDeclaredFields();

for (Field field : fields) {

System.***out***.println("Field Name: " + field.getName() + ", Type: " + field.getType());

}

// Getting declared methods

System.***out***.println("\nDeclared Methods:");

Method[] methods = clazz.getDeclaredMethods();

for (Method method : methods) {

System.***out***.println("Method Name: " + method.getName());

}

// Getting declared constructors

System.***out***.println("\nDeclared Constructors:");

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

for (Constructor<?> constructor : constructors) {

System.***out***.println("Constructor: " + constructor);

}

// Modifying the access level of a private field and setting its value during runtime

ReflectionExample obj = new ReflectionExample("Initial Value");

Field privateField = clazz.getDeclaredField("privateField");

privateField.setAccessible(true); // Set the field accessible

System.***out***.println("\nInitial value of privateField: " + privateField.get(obj));

// Setting new value to the private field

privateField.set(obj, "New Value");

System.***out***.println("Modified value of privateField: " + privateField.get(obj));

}

}

**Output-**

Declared Fields:

Field Name: privateField, Type: class java.lang.String

Declared Methods:

Method Name: main

Method Name: privateMethod

Method Name: publicMethod

Declared Constructors:

Constructor: public com.epwiproday\_19.ReflectionExample(java.lang.String)

Initial value of privateField: Initial Value

Modified value of privateField: New Value

**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.**

**Solution-**

package com.epwiproday\_19;

import java.util.Comparator;

import java.util.List;

import java.util.ArrayList;

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 class Main {

public static void main(String[] args) {

// Create a list of Person objects

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

people.add(new Person("Alice", 30));

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

people.add(new Person("Charlie", 35));

people.add(new Person("David", 20));

// Sort the list of Person objects by their age using a lambda expression

people.sort(Comparator.*comparingInt*(Person::getAge));

// Print the sorted list

System.***out***.println("Sorted List of People by Age:");

for (Person person : people) {

System.***out***.println(person);

}

}

}

**Output-**

Sorted List of People by Age:

David (20)

Bob (25)

Alice (30)

Charlie (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.**

**Solution-**

package com.epwiproday\_19;

import java.util.function.Predicate;

import java.util.function.Function;

import java.util.function.Consumer;

import java.util.function.Supplier;

class PersonOperations {

class Person1 {

private String name;

private int age;

public Person1(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 + ")";

}

}

// Method to accept functions as parameters

public void processPerson(Person1 person,

Predicate<Person1> predicate,

Function<Person1, String> function,

Consumer<String> consumer,

Supplier<Integer> supplier) {

// Apply predicate to the person

if (predicate.test(person)) {

// Apply function to the person and get result

String result = function.apply(person);

// Consume the result

consumer.accept(result);

} else {

// Get age from supplier if predicate fails

int age = supplier.get();

System.***out***.println("Predicate failed. Default age: " + age);

}

}

public static void main(String[] args) {

// Create an instance of PersonOperations

PersonOperations personOperations = new PersonOperations();

// Create a Person1 object

Person1 person = personOperations.new Person1("Alice", 30);

// Define Predicate to check if age is greater than 25

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

// Define Function to transform Person1 into a string

Function<Person1, String> function = p -> p.getName() + " is " + p.getAge() + " years old.";

// Define Consumer to print the result

Consumer<String> consumer = System.***out***::println;

// Define Supplier to provide default age

Supplier<Integer> defaultAgeSupplier = () -> 25;

// Process the Person1 object using the functions

personOperations.processPerson(person, agePredicate, function, consumer, defaultAgeSupplier);

}

}

**Output-**

Alice is 30 years old.