

Day-19

Name: Amit Kumar Parhi

Email: amitykparhi07@mail.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.wipro.assign19;

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);
    }

    public static void main(String[] args) {
        Pair<Integer, String> intStringPair = new Pair<>(42, "Hello");
        System.out.println("Original Pair: " +
            intStringPair.getFirst() + ", " + intStringPair.getSecond());

        Pair<String, Integer> reversedPair = intStringPair.reverse();
        System.out.println("Reversed Pair: " + reversedPair.getFirst()
            + ", " + reversedPair.getSecond());
    }
}
```

## Output:



The screenshot shows an IDE with the Package Explorer on the left, the Editor in the center, and the Console at the bottom. The Package Explorer shows a project named 'DSAAAssignment' with a package 'com.wipro.assign19' containing a file 'Pair.java'. The Editor shows the following code:

```
1 package com.wipro.assign19;
2
3 public class Pair<T, U> {
4     private T first;
5     private U second;
6
7     public Pair(T first, U second) {
8         this.first = first;
9         this.second = second;
10    }
11
12    public T getFirst() {
13        return first;
14    }
15
16    public U getSecond() {
17        return second;
18    }
19
20
21    public Pair<U, T> reverse() {
22        return new Pair<>((second, first));
23    }
24
25    public static void main(String[] args) {
26        Pair<Integer, String> intStringPair = new Pair<>(42, "Hello");
```

The Console at the bottom shows the output of the program:

```
<terminated> Pair (1) [Java Application] C:\Users\vaish\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (Jun 2, 2024, 10:21:35 PM - 10:21:36 PM)
Original Pair: 42, Hello
Reversed Pair: Hello, 42
```

## 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.wipro.assign19;

import java.util.Arrays;

public class Swap {
    public static <T> void swap(T[] array, int left, int right) {
        if (array == null || left < 0 || right < 0 || left >=
array.length || right >= array.length) {
            throw new IllegalArgumentException("Invalid indices for
swapping.");
        }

        T temp = array[left];
        array[left] = array[right];
        array[right] = temp;
    }

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

```

Integer[] intArray = {1, 2, 3, 4, 5};
swap(intArray, 1, 3);
System.out.println("Swapped Int array: " +
Arrays.toString(intArray));

```

```

String[] stringArray = {"apple", "banana", "cherry", "date"};
swap(stringArray, 0, 2);
System.out.println("Swapped String array: " +
Arrays.toString(stringArray));
}
}

```

Output:

The screenshot shows an IDE with the following components:

- Package Explorer:** Shows a project structure with packages like `com.wipro.assign19` and `com.wipro.ds`. The `Swap.java` file is selected.
- Editor:** Displays the code for `Swap.java`. The code includes a generic `swap` method and a `main` method that demonstrates its usage with integer and string arrays.
- Console:** Shows the output of the program:
 

```

Swapped Int array: [1, 4, 3, 2, 5]
Swapped String array: [cherry, banana, apple, date]
      
```

### 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.wipro.assign19;
```

```
import java.lang.reflect.Field;

public class Reflect {
    private int PField = 42;

    public static void main(String[] args) throws
NoSuchFieldException, IllegalAccessException {
        Reflect instance = new Reflect();

        Class<?> clazz = instance.getClass();

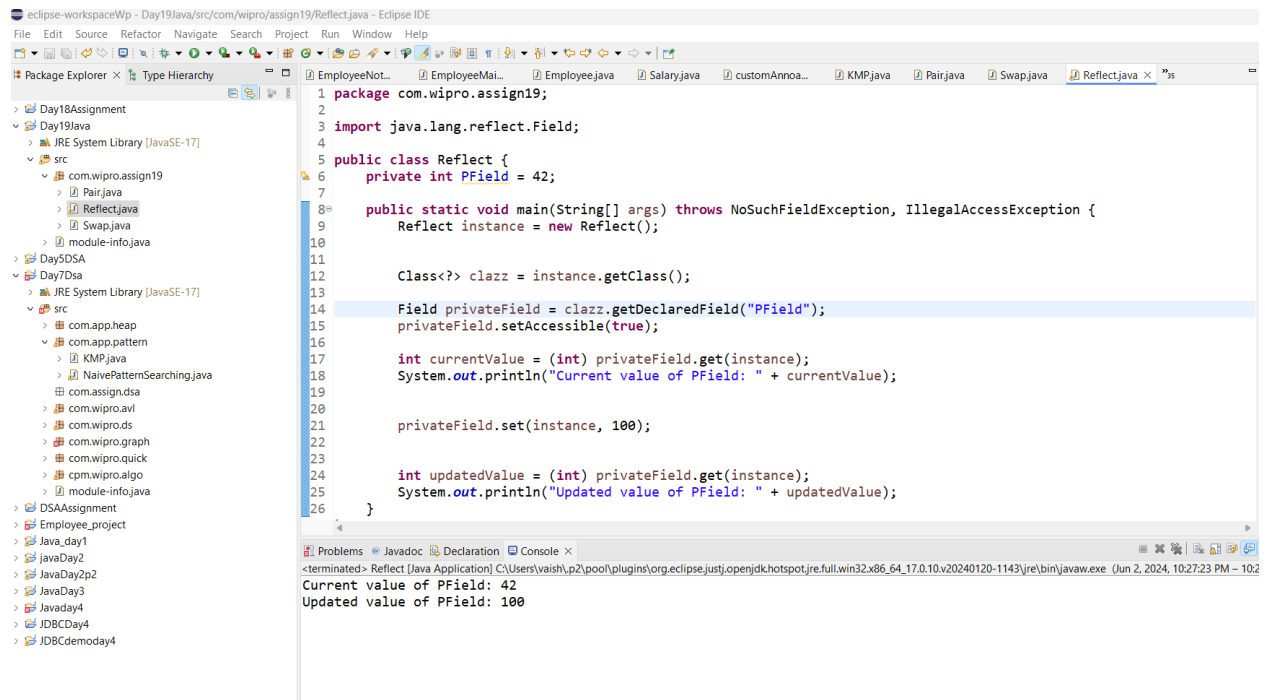
        Field privateField = clazz.getDeclaredField("PField");
        privateField.setAccessible(true);

        int currentValue = (int) privateField.get(instance);
        System.out.println("Current value of PField: " +
currentValue);

        privateField.set(instance, 100);

        int updatedValue = (int) privateField.get(instance);
        System.out.println("Updated value of PField: " +
updatedValue);
    }
}
```

## Output:



The screenshot shows the Eclipse IDE with the 'Reflect.java' file open. The code defines a 'Reflect' class with a private static field 'PField' initialized to 42. The 'main' method uses reflection to access this field, prints its current value (42), updates it to 100, and prints the updated value (100). The console output at the bottom confirms these actions.

```
1 package com.wipro.assign19;
2
3 import java.lang.reflect.Field;
4
5 public class Reflect {
6     private int PField = 42;
7
8     public static void main(String[] args) throws NoSuchFieldException, IllegalAccessException {
9         Reflect instance = new Reflect();
10
11
12         Class<?> clazz = instance.getClass();
13
14         Field privateField = clazz.getDeclaredField("PField");
15         privateField.setAccessible(true);
16
17         int currentValue = (int) privateField.get(instance);
18         System.out.println("Current value of PField: " + currentValue);
19
20
21         privateField.set(instance, 100);
22
23
24         int updatedValue = (int) privateField.get(instance);
25         System.out.println("Updated value of PField: " + updatedValue);
26     }
27 }
```

Console Output:

```
<terminated> Reflect [Java Application] C:\Users\vaish\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (Jun 2, 2024, 10:27:23 PM - 10:27:23 PM)
Current value of PField: 42
Updated value of PField: 100
```

## 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.wipro.assign19;

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

public class SortPerson {
    public static void main(String[] args) {
        List<Person> personList = new ArrayList<>();
        personList.add(new Person("Lily", 25));
        personList.add(new Person("Marshal", 29));
        personList.add(new Person("Robin", 22));
        personList.add(new Person("Ted", 27));

        personList.sort(Comparator.comparingInt(Person::getAge));
    }
}
```

```

        System.out.println("Sorted list by age:");
        for (Person person : personList) {
            System.out.println(person.getName() + " age:" +
person.getAge());
        }
    }
}

```

**Output:**

```

1 package com.wipro.assign19;
2
3 import java.util.ArrayList;
4 import java.util.Comparator;
5 import java.util.List;
6
7 public class SortPerson {
8     public static void main(String[] args) {
9         List<Person> personList = new ArrayList<>();
10        personList.add(new Person("Lily", 25));
11        personList.add(new Person("Marshal", 29));
12        personList.add(new Person("Robin", 22));
13        personList.add(new Person("Ted", 27));
14
15
16
17        personList.sort(Comparator.comparingInt(Person::getAge));
18
19
20        System.out.println("Sorted list by age:");
21        for (Person person : personList) {
22            System.out.println(person.getName() + " age:" + person.getAge());
23        }
24    }
25 }
26

```

Console Output:

```

Sorted list by age:
Robin age:22
Lily age:25
Ted age:27
Marshal age:29

```

## 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.wipro.assign19;

import java.util.function.Consumer;
import java.util.function.Function;
import java.util.function.Predicate;
import java.util.function.Supplier;

```

```

public class FunctionalInterf {
    public boolean testPerson(Predicate<Person> predicate, Person
person) {
        return predicate.test(person);
    }

    public <R> R applyFunction(Function<Person, R> function,
Person person) {
        return function.apply(person);
    }

    public void acceptConsumer(Consumer<Person> consumer, Person
person) {
        consumer.accept(person);
    }

    public Person getFromSupplier(Supplier<Person> supplier) {
        return supplier.get();
    }

    public static void main(String[] args) {
        FunctionalInterf operations = new FunctionalInterf();
        Person person = new Person("John Doe", 25);

        Predicate<Person> isAdult = p -> p.getAge() >= 18;

        Function<Person, String> ageBetween22And30 = p -> {
            if (p.getAge() > 22 && p.getAge() < 30) {
                return p.getName();
            }
            return null;
        };

        Consumer<Person> printPerson = p ->
System.out.println("Person: " + p.getName());

        Supplier<Person> personSupplier = () -> new
Person("Barney stinson", 28);

        if (operations.testPerson(isAdult, person)) {

```

```

        System.out.println(person.getName() + " is an
adult.");
    }

    String personName =
operations.applyFunction(ageBetween22And30, person);
    if (personName != null) {
        System.out.println(personName + " is between 22 and
30 years old.");
    }

    operations.acceptConsumer(printPerson, person);

    Person newPerson =
operations.getFromSupplier(personSupplier);
    System.out.println("New person from supplier: " +
newPerson.getName());

    if (operations.testPerson(isAdult, newPerson)) {
        System.out.println(newPerson.getName() + " is an
adult.");
        personName =
operations.applyFunction(ageBetween22And30, newPerson);
        if (personName != null) {
            System.out.println(personName + " is between 22
and 30 years old.");
        }
        operations.acceptConsumer(printPerson, newPerson);
    }
}
}

```



## Output:

The screenshot displays an IDE with a project explorer on the left and a code editor on the right. The project explorer shows a project named 'DSAAssignment' with various sub-projects and source files. The code editor shows a Java file with the following code:

```
46
47
48     if (operations.testPerson(isAdult, person)) {
49         System.out.println(person.getName() + " is an adult.");
50     }
51
52     String personName = operations.applyFunction(ageBetween22And30, person);
53     if (personName != null) {
54         System.out.println(personName + " is between 22 and 30 years old.");
55     }
56
57     operations.acceptConsumer(printPerson, person);
58
59     Person newPerson = operations.getFromSupplier(personSupplier);
60     System.out.println("New person from supplier: " + newPerson.getName());
61
62
63     if (operations.testPerson(isAdult, newPerson)) {
64         System.out.println(newPerson.getName() + " is an adult.");
```

The output window at the bottom shows the following text:

```
<terminated> FunctionalInterf [Java Application] C:\Users\vaish.p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (Jun 2, 2024, 10:49:1)
John Doe is between 22 and 30 years old.
Person: John Doe
New person from supplier: Barney stinson
Barney stinson is an adult.
Barney stinson is between 22 and 30 years old.
Person: Barney stinson
```