

Assignments

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.

The screenshot displays the Eclipse IDE with a Java project named 'wiproAssignment'. The 'Package Explorer' on the left shows the project structure, including a package 'com.wipro.selectors' containing a 'Generics.java' file. The 'Main Editor' shows the code for 'Generics.java'.

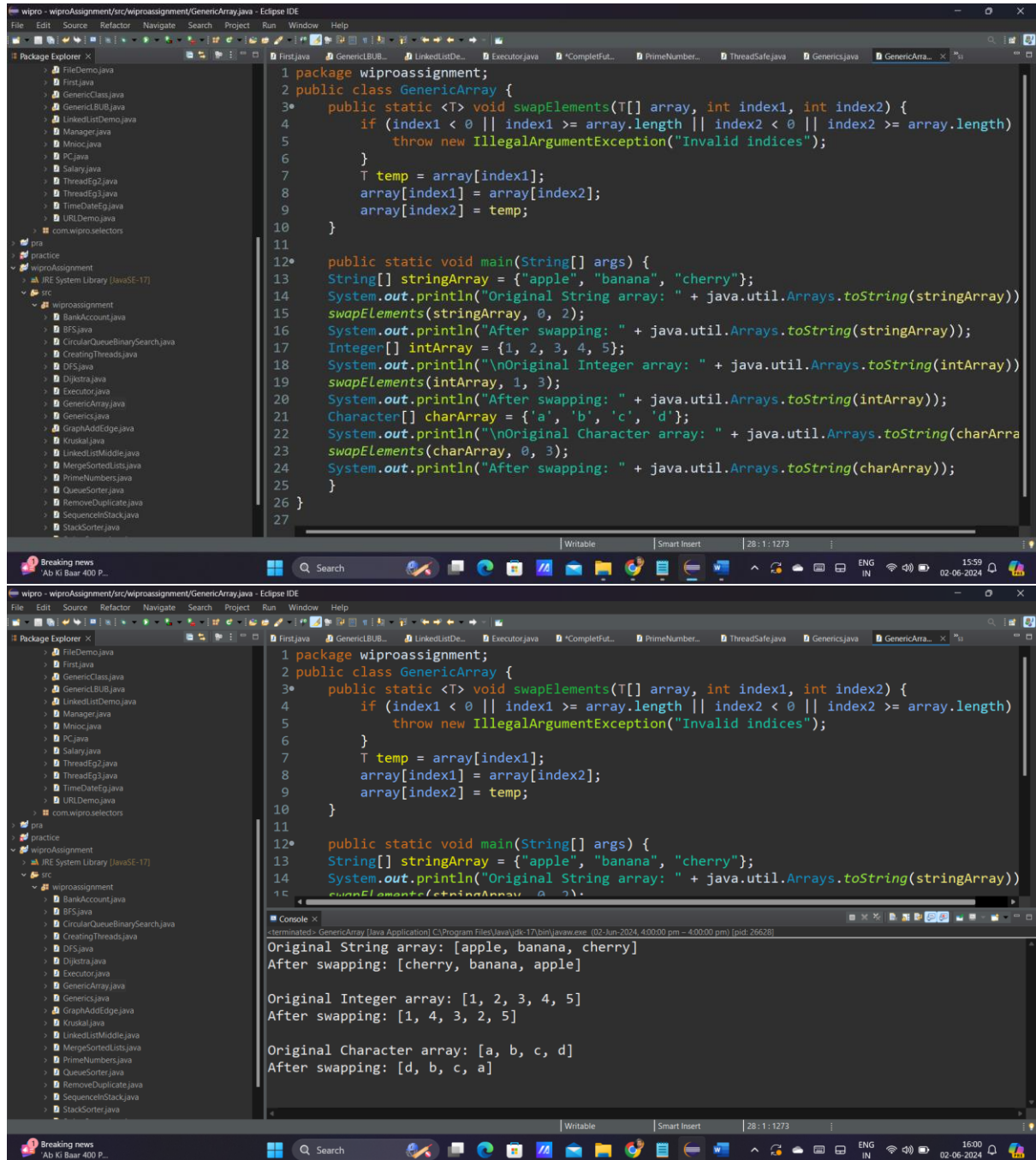
```
1 package wiproassignment;
2
3 public class Generics<T, U> {
4     private T first;
5     private U second;
6     public Generics(T first, U second) {
7         this.first = first;
8         this.second = second;
9     }
10    public T getFirst() {
11        return first;
12    }
13    public U getSecond() {
14        return second;
15    }
16    public Generics<U, T> reverse() {
17        return new Generics<>(second, first);
18    }
19    public static void main(String[] args) {
20        Generics<String, Integer> pair = new Generics<>("Hello", 123);
21        System.out.println("Original Pair: " + pair.getFirst() + ", " + pair.getSecond());
22        Generics<Integer, String> reversedPair = pair.reverse();
23        System.out.println("Reversed Pair: " + reversedPair.getFirst() + ", " + reversedPair.getSecond());
24    }
25 }
26
```

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

```
<terminated> Generics [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (02-Jun-2024, 3:42:25 pm - 3:42:25 pm) [pid: 35368]
Original Pair: Hello, 123
Reversed Pair: 123, Hello
```

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.



```
1 package wiproassignment;
2 public class GenericArray {
3     public static <T> void swapElements(T[] array, int index1, int index2) {
4         if (index1 < 0 || index1 >= array.length || index2 < 0 || index2 >= array.length)
5             throw new IllegalArgumentException("Invalid indices");
6         T temp = array[index1];
7         array[index1] = array[index2];
8         array[index2] = temp;
9     }
10 }
11
12 public static void main(String[] args) {
13     String[] stringArray = {"apple", "banana", "cherry"};
14     System.out.println("Original String array: " + java.util.Arrays.toString(stringArray));
15     swapElements(stringArray, 0, 2);
16     System.out.println("After swapping: " + java.util.Arrays.toString(stringArray));
17     Integer[] intArray = {1, 2, 3, 4, 5};
18     System.out.println("\nOriginal Integer array: " + java.util.Arrays.toString(intArray));
19     swapElements(intArray, 1, 3);
20     System.out.println("After swapping: " + java.util.Arrays.toString(intArray));
21     Character[] charArray = {'a', 'b', 'c', 'd'};
22     System.out.println("\nOriginal Character array: " + java.util.Arrays.toString(charArray));
23     swapElements(charArray, 0, 3);
24     System.out.println("After swapping: " + java.util.Arrays.toString(charArray));
25 }
26 }
27 }
```

Console Output:

```
<terminated> GenericArray [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (02-jun-2024, 4:00:00 pm - 4:00:00 pm) [pid: 25628]
Original String array: [apple, banana, cherry]
After swapping: [cherry, banana, apple]

Original Integer array: [1, 2, 3, 4, 5]
After swapping: [1, 4, 3, 2, 5]

Original Character array: [a, b, c, d]
After swapping: [d, b, c, a]
```

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

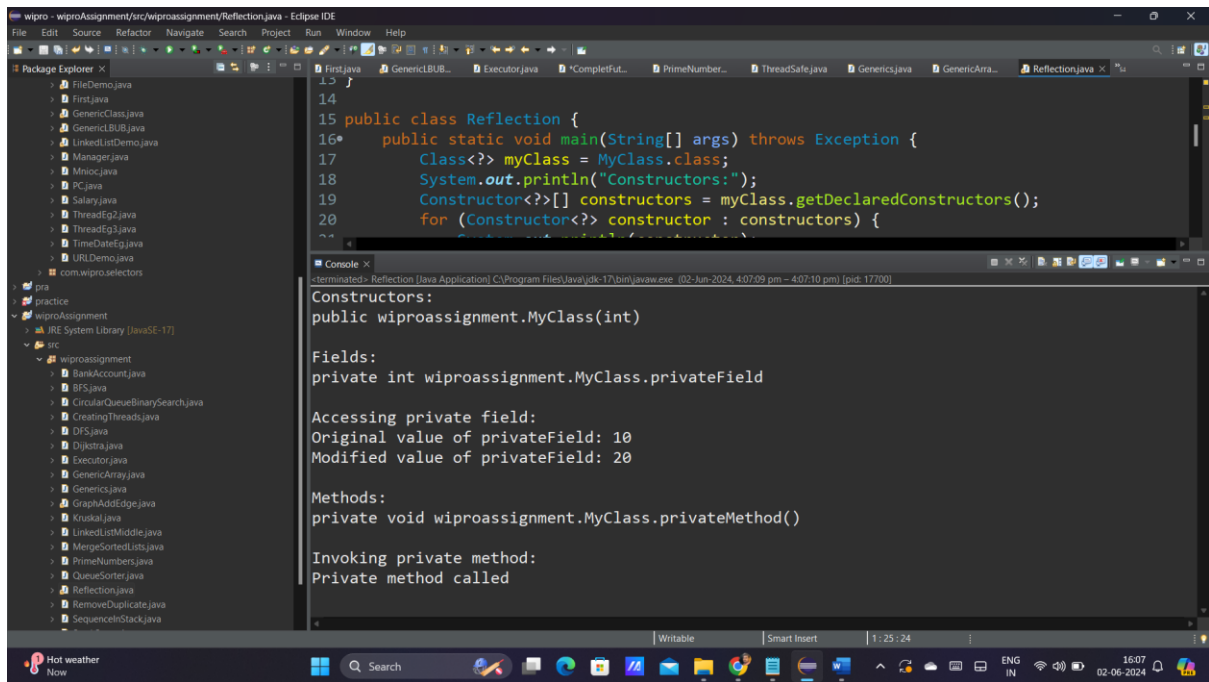
The image displays two screenshots of the Eclipse IDE, showing the development of a Java application that uses reflection to inspect a class.

Top Screenshot: The IDE shows the package explorer on the left with a project named "wiproAssignment". The main editor displays the code for "Reflection.java". The code defines a "MyClass" with a private field "privateField" and a private method "privateMethod". The "Reflection" class has a static "main" method that uses reflection to inspect "MyClass".

```
1 package wiproassignment;
2 import java.lang.reflect.Constructor;
3 import java.lang.reflect.Field;
4 import java.lang.reflect.Method;
5 class MyClass {
6     private int privateField;
7     public MyClass(int privateField) {
8         this.privateField = privateField;
9     }
10    private void privateMethod() {
11        System.out.println("Private method called");
12    }
13 }
14
15 public class Reflection {
16     public static void main(String[] args) throws Exception {
17         Class<?> myClass = MyClass.class;
18         System.out.println("Constructors:");
19         Constructor<?>[] constructors = myClass.getDeclaredConstructors();
20         for (Constructor<?> constructor : constructors) {
21             System.out.println(constructor);
22         }
23         System.out.println("\nFields:");
24         Field[] fields = myClass.getDeclaredFields();
25         for (Field field : fields) {
26             System.out.println(field);
27         }
28     }
29 }
```

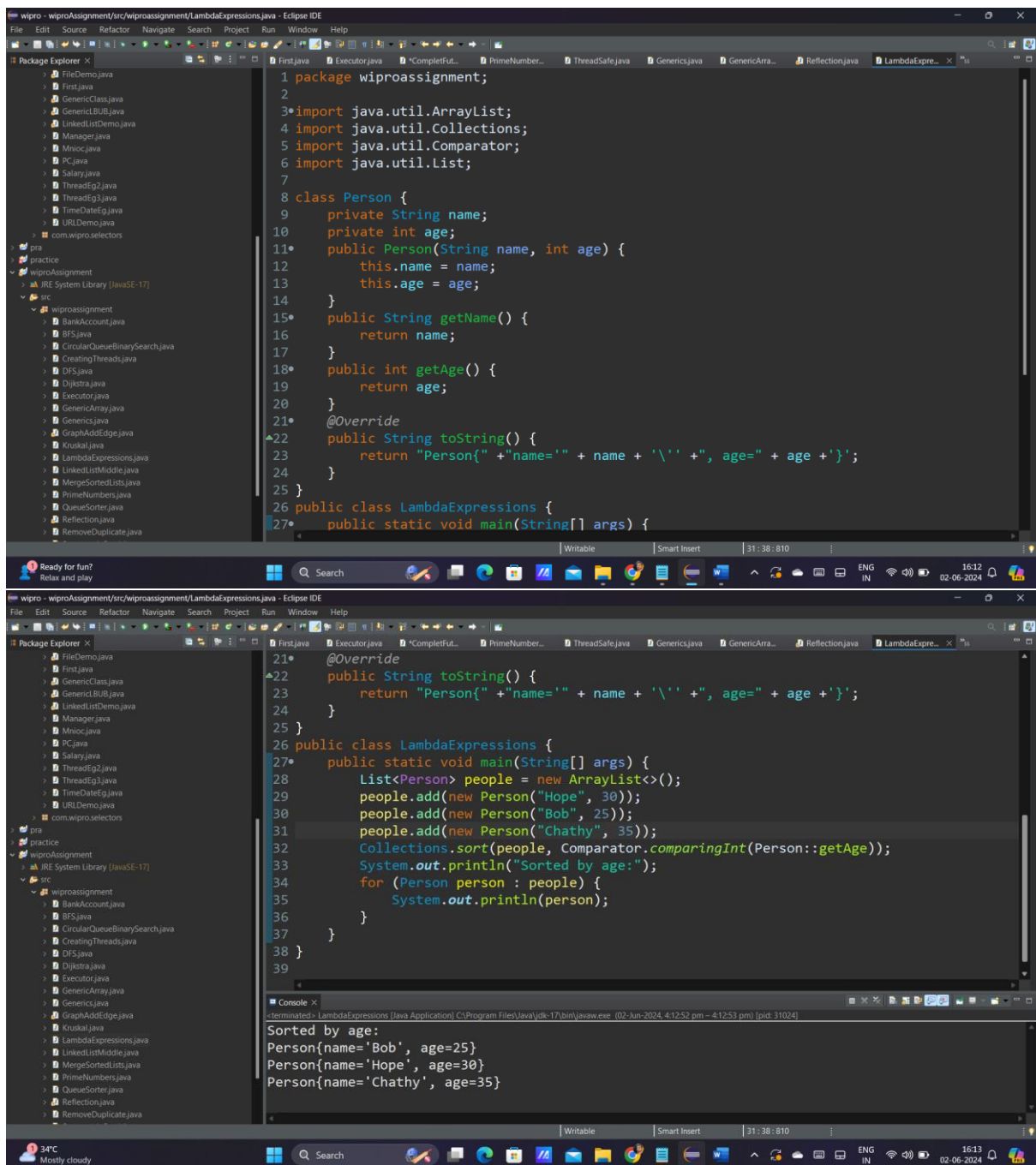
Bottom Screenshot: The IDE shows the same project, but the code in "Reflection.java" is extended to demonstrate accessing private fields and methods.

```
22 }
23 System.out.println("\nFields:");
24 Field[] fields = myClass.getDeclaredFields();
25 for (Field field : fields) {
26     System.out.println(field);
27 }
28 System.out.println("\nAccessing private field:");
29 MyClass obj = new MyClass(10);
30 Field privateField = myClass.getDeclaredField("privateField");
31 privateField.setAccessible(true);
32 int fieldValue = (int) privateField.get(obj);
33 System.out.println("Original value of privateField: " + fieldValue);
34 privateField.set(obj, 20);
35 fieldValue = (int) privateField.get(obj);
36 System.out.println("Modified value of privateField: " + fieldValue);
37 System.out.println("\nMethods:");
38 Method[] methods = myClass.getDeclaredMethods();
39 for (Method method : methods) {
40     System.out.println(method);
41 }
42 System.out.println("\nInvoking private method:");
43 Method privateMethod = myClass.getDeclaredMethod("privateMethod");
44 privateMethod.setAccessible(true);
45 privateMethod.invoke(obj);
46 }
47 }
48 }
```



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..



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.


```
wipro - wiproAssignment/src/wiproAssignment/FunctionalInterface.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer
  LinkedListDemo.java
  Manager.java
  Minoc.java
  PC.java
  Salary.java
  ThreadEg2.java
  ThreadEg3.java
  TimeDateEg.java
  URLLDemo.java
  com.wipro.selectors
  pra
  practice
  wiproAssignment
    IRE System Library (JavaSE-17)
    src
      wiproAssignment
        BankAccount.java
        BFS.java
        CircularQueueBinarySearch.java
        CreatingThreads.java
        DFS.java
        Dijkstra.java
        Executor.java
        FunctionalInterface.java
        GenericArray.java
        Generics.java
        GraphAddEdge.java
        Kruskal.java
        LambdaExpressions.java
        LinkedListMiddle.java
        MergeSortedList.java
        PrimeNumbers.java
        QueueSorter.java
        Reflection.java
        RemoveDuplicate.java
        SequenceInStack.java
        StackSorter.java
        StringOperations.java

21 package wiproAssignment;
22 import java.util.function.Consumer;
23 import java.util.function.Function;
24 import java.util.function.Predicate;
25 import java.util.function.Supplier;
26 class PersonA {
27     private String name;
28     private int age;
29
30     public PersonA(String name, int age) {
31         this.name = name;
32         this.age = age;
33     }
34
35     public String getName() {
36         return name;
37     }
38
39     public int getAge() {
40         return age;
41     }
42
43     public String toString() {
44         return "Person{" +
45             "name=" + name + '\'' +
46             ", age=" + age +
47             '\'';
48     }
49 }
```

```
wipro - wiproAssignment/src/wiproAssignment/FunctionalInterface.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer
  LinkedListDemo.java
  Manager.java
  Minoc.java
  PC.java
  Salary.java
  ThreadEg2.java
  ThreadEg3.java
  TimeDateEg.java
  URLLDemo.java
  com.wipro.selectors
  pra
  practice
  wiproAssignment
    IRE System Library (JavaSE-17)
    src
      wiproAssignment
        BankAccount.java
        BFS.java
        CircularQueueBinarySearch.java
        CreatingThreads.java
        DFS.java
        Dijkstra.java
        Executor.java
        FunctionalInterface.java
        GenericArray.java
        Generics.java
        GraphAddEdge.java
        Kruskal.java
        LambdaExpressions.java
        LinkedListMiddle.java
        MergeSortedList.java
        QueueSorter.java
        Reflection.java
        RemoveDuplicate.java
        SequenceInStack.java
        StackSorter.java
        StringOperations.java

25 }
26 public class FunctionalInterface {
27     public static void processPerson(PersonA person,
28                                     Predicate<PersonA> predicate,
29                                     Function<PersonA, String> function,
30                                     Consumer<String> consumer,
31                                     Supplier<Integer> supplier) {
32         if (predicate.test(person)) {
33             String result = function.apply(person);
34             consumer.accept(result);
35             int suppliedValue = supplier.get();
36             System.out.println("Supplied value: " + suppliedValue);
37         } else {
38             System.out.println("Predicate condition not met for " + person.getName());
39         }
40     }
41
42     public static void main(String[] args) {
43         PersonA person = new PersonA("Alice", 30);
44         Predicate<PersonA> isAdult = p -> p.getAge() >= 18;
45         Function<PersonA, String> getNameFunction = PersonA::getName;
46         Consumer<String> printNameConsumer = System.out::println;
47         Supplier<Integer> ageSupplier = person::getAge;
48         processPerson(person, isAdult, getNameFunction, printNameConsumer, ageSupplier);
49     }
50 }
51 }
```

