Object Oriented System Laboratory Assignment-3

Name: Kaustav Mondal

Section: A1

Roll No: 002311001017

1) Write a generic method in Java that takes an array of any data type and sorts the array in ascending order using any sorting algorithm.

Ans:

```
import java.util.Arrays;

public class GenericSort {

        // Generic method to sort an array of any Comparable data type
        public static <T extends Comparable<T>> void sortArray(T[] array) {
        int n = array.length;

        // Bubble sort algorithm
        for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - 1 - i; j++) {
                if (array[j].compareTo(array[j + 1]) > 0) {
                // Swap the elements if they are in the wrong order
                T temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
                }
           }
         }
        }

        // Main method to test the generic sort method
        public static void main(String[] args) {
        // Test with Integer array
        Integer[] intArray = { 5, 2, 8, 1, 3 };
        System.out.println("Before sorting: " + Arrays.toString(intArray));
        sortArray(intArray);
        System.out.println("After sorting: " + Arrays.toString(intArray));

        // Test with String array
        String[] strArray = { "Apple", "Banana", "Orange", "Grape" };
        System.out.println("Before sorting: " + Arrays.toString(strArray));
        sortArray(strArray);
        System.out.println("After sorting: " + Arrays.toString(strArray));
        }
}
```

```
Q1.class
[be2317@localhost Assign-3]$ nano GenericSort.java
[be2317@localhost Assign-3]$ javac GenericSort.java
[be2317@localhost Assign-3]$ java GenericSort
Before sorting: [5, 2, 8, 1, 3]
After sorting: [1, 2, 3, 5, 8]
Before sorting: [Apple, Banana, Orange, Grape]
After sorting: [Apple, Banana, Grape, Orange]
```

2) Write a generic method in Java that takes any type of an array as input and finds the frequency of each data element.

Ans:

```
import java.util.HashMap;
import java.util.Map;

public class FrequencyCounter {

        // Generic method to count the frequency of each element in an array
        public static <T> void countFrequency(T[] array) {
        Map<T, Integer> frequencyMap = new HashMap<>();

        // Loop through the array and count frequencies
        for (T element : array) {
        frequencyMap.put(element, frequencyMap.getOrDefault(element, 0) + 1);
        }

        // Print the frequency of each element
        for (Map.Entry<T, Integer> entry : frequencyMap.entrySet()) {
        System.out.println("Element: " + entry.getKey() + " - Frequency: " + entry.getValue());
        }
        }

        // Main method to test the generic frequency counter
        public static void main(String[] args) {
        // Test with Integer array
        Integer[] intArray = { 1, 2, 2, 3, 1, 1, 4 };
        System.out.println("Frequency of elements in Integer array:");
        countFrequency(intArray);

        // Test with String array
        String[] strArray = { "Apple", "Banana", "Apple", "Orange", "Apple" };
        System.out.println("\nFrequency of elements in String array:");
        countFrequency(strArray);
        }
}
```

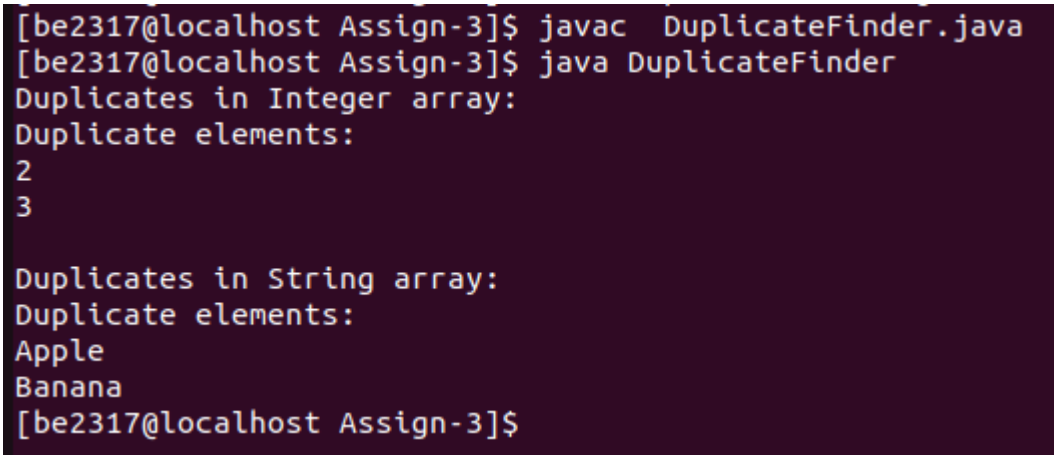
```
[be2317@localhost Assign-3]$ nano FrequencyCounter.java
[be2317@localhost Assign-3]$ javac FrequencyCounter.java
[be2317@localhost Assign-3]$ java FrequencyCounter
Frequency of elements in Integer array:
Element: 1 - Frequency: 3
Element: 2 - Frequency: 2
Element: 3 - Frequency: 1
Element: 4 - Frequency: 1

Frequency of elements in String array:
Element: Apple - Frequency: 3
Element: Orange - Frequency: 1
Element: Banana - Frequency: 1
```

3) Design a generic Java class having a method that takes an array of any data type and prints all the duplicate elements.

Ans:

```
import java.util.HashSet;
import java.util.Set;

public class DuplicateFinder {

        // Generic method to find and print duplicate elements in an array
        public static <T> void printDuplicates(T[] array) {
        Set<T> seen = new HashSet<>();      // Set to store elements we've already
encountered
        Set<T> duplicates = new HashSet<>();  // Set to store duplicate elements

        // Loop through the array to find duplicates
        for (T element : array) {
        if (!seen.add(element)) {
                // If element is already in 'seen', it's a duplicate
                duplicates.add(element);
        }
        }

        // Print the duplicates
        if (duplicates.isEmpty()) {
        System.out.println("No duplicates found.");
        } else {
        System.out.println("Duplicate elements: ");
        for (T duplicate : duplicates) {
                System.out.println(duplicate);
```

```
        }
        }
        }

        // Main method to test the generic method
        public static void main(String[] args) {
        // Test with Integer array
        Integer[] intArray = { 1, 2, 3, 2, 4, 5, 3 };
        System.out.println("Duplicates in Integer array:");
        printDuplicates(intArray);

        // Test with String array
        String[] strArray = { "Apple", "Banana", "Apple", "Orange", "Banana" };
        System.out.println("\nDuplicates in String array:");
        printDuplicates(strArray);
        }
}
```



4) Test the functionalities of different java reflection APIs such as getClass(), getMethods(), getConstructors(), getDeclaredMethod(), getDeclaredField(), setAccessible() etc.

Ans:

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;

class SampleClass {
        private int x;
```

```
	public String name;

	public SampleClass() {
	this.x = 10;
	this.name = "Reflection Example";
	}

	public SampleClass(int x, String name) {
	this.x = x;
	this.name = name;
	}

	private void display() {
	System.out.println("Private Method - display() is called!");
	}

	public void show() {
	System.out.println("Public Method - show() is called!");
	}

	public int getX() {
	return x;
	}
}

public class ReflectionExample {
	public static void main(String[] args) {
	try {
	// 1. Using getClass() method
	Class<?> clazz = SampleClass.class;  // or obj.getClass() if instance is available
	System.out.println("Class Name: " + clazz.getName());

	// 2. Using getMethods() to get public methods
	Method[] methods = clazz.getMethods();
	System.out.println("\nPublic Methods:");
	for (Method method : methods) {
		System.out.println("Method Name: " + method.getName() + ", Modifier: " +
Modifier.toString(method.getModifiers()));
	}

	// 3. Using getDeclaredMethods() to get all methods (including private methods)
	Method[] declaredMethods = clazz.getDeclaredMethods();
	System.out.println("\nDeclared Methods (including private methods):");
	for (Method method : declaredMethods) {
		System.out.println("Method Name: " + method.getName() + ", Modifier: " +
Modifier.toString(method.getModifiers()));
	}
```

```
        // 4. Using getConstructors() to get constructors
        Constructor<?>[] constructors = clazz.getConstructors();
        System.out.println("\nPublic Constructors:");
        for (Constructor<?> constructor : constructors) {
                System.out.println("Constructor: " + constructor.getName());
        }

        // 5. Using getDeclaredConstructor() to get specific constructor (including
private)
        Constructor<?> privateConstructor = clazz.getDeclaredConstructor(int.class,
String.class);
        System.out.println("\nDeclared Constructor (int, String): " +
privateConstructor.getName());

        // 6. Using getDeclaredField() to get private field
        Field field = clazz.getDeclaredField("x");
        System.out.println("\nDeclared Field (private 'x'): " + field.getName());

        // 7. Using setAccessible() to access private field
        field.setAccessible(true);
        SampleClass sample = new SampleClass();
        System.out.println("\nAccessing private field 'x' through reflection: " +
field.get(sample));

        // 8. Invoking a private method using reflection
        Method privateMethod = clazz.getDeclaredMethod("display");
        privateMethod.setAccessible(true);  // Making the private method accessible
        privateMethod.invoke(sample);

        // 9. Accessing public method and invoking it
        Method publicMethod = clazz.getMethod("show");
        publicMethod.invoke(sample);

        // 10. Accessing public field and modifying it
        Field nameField = clazz.getField("name");
        nameField.set(sample, "Updated Name via Reflection");
        System.out.println("\nUpdated public field 'name': " + nameField.get(sample));

        } catch (Exception e) {
        e.printStackTrace();
        }
        }
}
```

```
[be2317@localhost Assign-3]$ javac ReflectionExample.java
[be2317@localhost Assign-3]$ java ReflectionExample
Class Name: SampleClass

Public Methods:
Method Name: getX, Modifier: public
Method Name: show, Modifier: public
Method Name: wait, Modifier: public final
Method Name: wait, Modifier: public final
Method Name: wait, Modifier: public final native
Method Name: equals, Modifier: public
Method Name: toString, Modifier: public
Method Name: hashCode, Modifier: public native
Method Name: getClass, Modifier: public final native
Method Name: notify, Modifier: public final native
Method Name: notifyAll, Modifier: public final native

Declared Methods (including private methods):
Method Name: getX, Modifier: public
Method Name: display, Modifier: private
Method Name: show, Modifier: public

Public Constructors:
Constructor: SampleClass
Constructor: SampleClass

Declared Constructor (int, String): SampleClass

Declared Field (private 'x'): x

Accessing private field 'x' through reflection: 10
Private Method - display() is called!
Public Method - show() is called!
```