Name: Sownthari R P

Week 3 Assignment

1. Mention the result for the mentioned statements using strings.

```
public class StringComparisonExample {
    public static void main(String[] args) {
        // String literals (pooled)
        String str1 = "Hello";
        String str2 = "Hello";

        // New String objects (not pooled)
        String str3 = new String("Hello");
        String str4 = new String("hello");

        // Using ==
        System.out.println("str1 == str2: " + (str1 == str2));
        System.out.println("str1 == str3: " + (str1 == str3));
        // Using equals()
        System.out.println("str1.equals(str3): " + str1.equals(str3));
        System.out.println("str1.equals(str4): " + str1.equals(str4));
        // Using equalsIgnoreCase()
        System.out.println("str1.equalsIgnoreCase(str4): " + str1.equalsIgnoreCase(str4));
    }
}
```

Output:

```
str1 == str2: true
str1 == str3: false
str1.equals(str3): true
str1.equals(str4): false
str1.equalsIgnoreCase(str4): true
```

2. Mention the result for the statements using integers.

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

//Mention what's the result in 1, 2, 3,4 and 5
        // Primitive int
        int int1 = 100;
        int int2 = 100;

        // Integer objects
        Integer intObj1 = 100;
        Integer intObj2 = 100;
        Integer intObj3 = new Integer(100);
        Integer intObj4 = new Integer(200);

        // Using == with primitive int
        System.out.println("int1 == int2: " + (int1 == int2));
        // Using == with Integer objects (within -128 to 127 range)
        System.out.println("intObj1 == intObj2: " + (intObj1 == intObj2));
        // Using == with Integer objects (new instance)
        System.out.println("intObj1 == intObj3: " + (intObj1 == intObj3));
        // Using equals() with Integer objects
        System.out.println("intObj1.equals(intObj3): " + intObj1.equals(intObj3));
        System.out.println("intObj1.equals(intObj4): " + intObj1.equals(intObj4));
    }
}
```

Output:

```
int1 == int2: true
intObj1 == intObj2: true
intObj1 == intObj3: false
intObj1.equals(intObj3): true
```

intObj1.equals(intObj4): false

3. Mention how Basic I/O resources are getting closed and the difference that you implemented earlier in the code - copyBytes.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class TryWithResourcesExample {
//Eliminating finally block to close resources.

    public static void main(String[] args) {
        // File path (adjust the path as needed)
        String filePath = "example.txt";

        // Traditional try-with-resources block
        try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

How Basic I/O Resources Are Getting Closed:

1. **Automatic Resource Management**:
    - In the TryWithResourcesExample, the BufferedReader (which wraps the FileReader) is declared inside the parentheses of the try statement.
    - Java ensures that the BufferedReader is automatically closed at the end of the try block, even if an exception occurs.
    - This eliminates the need for an explicit finally block to close the resource.

Difference Compared to copyBytes.java:

1. **Manual Resource Management**:
    - In copyBytes.java, the FileInputStream and FileOutputStream are manually closed in the finally block.
    - This approach requires explicitly checking if the resource is not null before closing it.
    - The finally block is essential here to ensure that the resources are closed, whether or not an exception occurs.
2. **Simplification with Try-With-Resources**:
    - In the TryWithResourcesExample, the try-with-resources statement handles closing automatically, leading to cleaner, more readable code.
    - There is no need for a finally block or null checks.
    - It reduces the potential for resource leaks, which can occur if an exception is thrown and the resource-closing code is missed in a manual approach.

4.Mention the order for 1,2 and 3 using collections

1. HashSet:

- Order of Elements: Unordered.
- HashSet does not guarantee any specific order of elements. The elements may appear in a seemingly random order depending on the hash codes and the internal hashing mechanism.
- Example Output: HashSet: **[Orange, Grapes, Banana, Apple]**

2. LinkedHashSet:

- Order of Elements: Insertion Order.
- LinkedHashSet maintains the order in which elements are inserted. The elements will appear in the order they were added to the set.
- Example Output: LinkedHashSet: **[Banana, Apple, Orange, Grapes]**

3. TreeSet:

- Order of Elements: Sorted Order.
- TreeSet sorts the elements according to their natural ordering (alphabetical order for strings). It maintains elements in ascending order.
- Example Output: TreeSet: **[Apple, Banana, Grapes, Orange]**