Name : Prathik Balaji N
Week 3 Task

1. Please find case 1 and 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)); // 1. (same memory reference) what's the result?
        System.out.println("str1 == str3: " + (str1 == str3)); //2. (different memory references) what's the result?

        // Using equals()
        System.out.println("str1.equals(str3): " + str1.equals(str3)); //3.  (same content) what's the result?
        System.out.println("str1.equals(str4): " + str1.equals(str4)); //4.  (case-sensitive) what's the result?

        // Using equalsIgnoreCase()
        System.out.println("str1.equalsIgnoreCase(str4): " + str1.equalsIgnoreCase(str4)); //5. (case-insensitive) what's the result?
    }
}
```

Result :

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

str1.equals(str4): false
str1.equalsIgnoreCase(str4): true

2.Find case 2 and 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)); // 1. (compares values)

       // Using == with Integer objects (within -128 to 127 range)
       System.out.println("intObj1 == intObj2: " + (intObj1 == intObj2)); // 2. (cached
objects)

       // Using == with Integer objects (new instance)
       System.out.println("intObj1 == intObj3: " + (intObj1 == intObj3)); // 3. (different
instances)

       // Using equals() with Integer objects
       System.out.println("intObj1.equals(intObj3): " + intObj1.equals(intObj3)); // 4. (same
content)
       System.out.println("intObj1.equals(intObj4): " + intObj1.equals(intObj4)); // 5.
(different content)
```

```
    }
}
```

Result :

```
int1 == int2: true
intObj1 == intObj2: true
intObj1 == intObj3: false
intObj1.equals(intObj3): true
intObj1.equals(intObj4): false
```

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

```
package Samp;

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 = "C:\\Users\\Prathik.b\\Documents\\prathik.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();
        }
```

```
    }
}
```

Result :

Hi Prathik

Automatic Resource Management:

The try-with-resources statement automatically manages and closes the resources (like BufferedReader) that implement the AutoCloseable interface.

No finally Block Needed:

There's no need for a finally block to explicitly close resources, as it was done in traditional try-catch-finally constructs.

Traditional Try-Catch-Finally Block:

Resources like InputStream and OutputStream are explicitly closed in a finally block.

If multiple resources are opened, they must be closed individually in the finally block, often requiring null checks to avoid NullPointerException.

Try-With-Resources Approach:

Resources are declared within the parentheses of the try statement and are automatically closed when the block exits.

No need for a finally block or manual resource management.

4.Find case 4 and mention the order for 1,2 and 3 using collections

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

public class SetExample {
    public static void main(String[] args) {
        // Set 1. What's the order of elements?
        Set<String> hashSet = new HashSet<>();
        hashSet.add("Banana");
        hashSet.add("Apple");
        hashSet.add("Orange");
        hashSet.add("Grapes");

        System.out.println("HashSet: " + hashSet);

        // LinkedHashSet 2. What's the order of elements  ?
        Set<String> linkedHashSet = new LinkedHashSet<>();
        linkedHashSet.add("Banana");
        linkedHashSet.add("Apple");
        linkedHashSet.add("Orange");
        linkedHashSet.add("Grapes");

        System.out.println("LinkedHashSet: " + linkedHashSet);

        // TreeSet 1. What's the order of elements  ?
        Set<String> treeSet = new TreeSet<>();
        treeSet.add("Banana");
        treeSet.add("Apple");
        treeSet.add("Orange");
        treeSet.add("Grapes");

        System.out.println("TreeSet: " + treeSet);
    }
```

}

HashSet :

HashSet: [Banana, Apple, Orange, Grapes]

Order of Elements: HashSet does not maintain any order of elements. The order in which elements are stored and retrieved is unpredictable and can vary depending on the internal hash function.

Output: The elements are printed in no particular order.

LinkedHashSet :

LinkedHashSet: [Banana, Apple, Orange, Grapes]

Order of Elements: LinkedHashSet maintains the insertion order. This means that elements are stored and retrieved in the order in which they were added.

Output: The elements are printed in the exact order they were inserted (Banana, Apple, Orange, Grapes).

TreeSet :

TreeSet: [Apple, Banana, Grapes, Orange]

Order of Elements: TreeSet stores elements in a sorted order according to their natural ordering (e.g., alphabetical order for strings) or according to a specified comparator.

Output: The elements are printed in sorted order (Apple, Banana, Grapes, Orange).