

## Weekend Assignment in Java

**1. Please find case 1 and mention the result for the mentioned statements using strings?**

**Case1 Code:-**

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

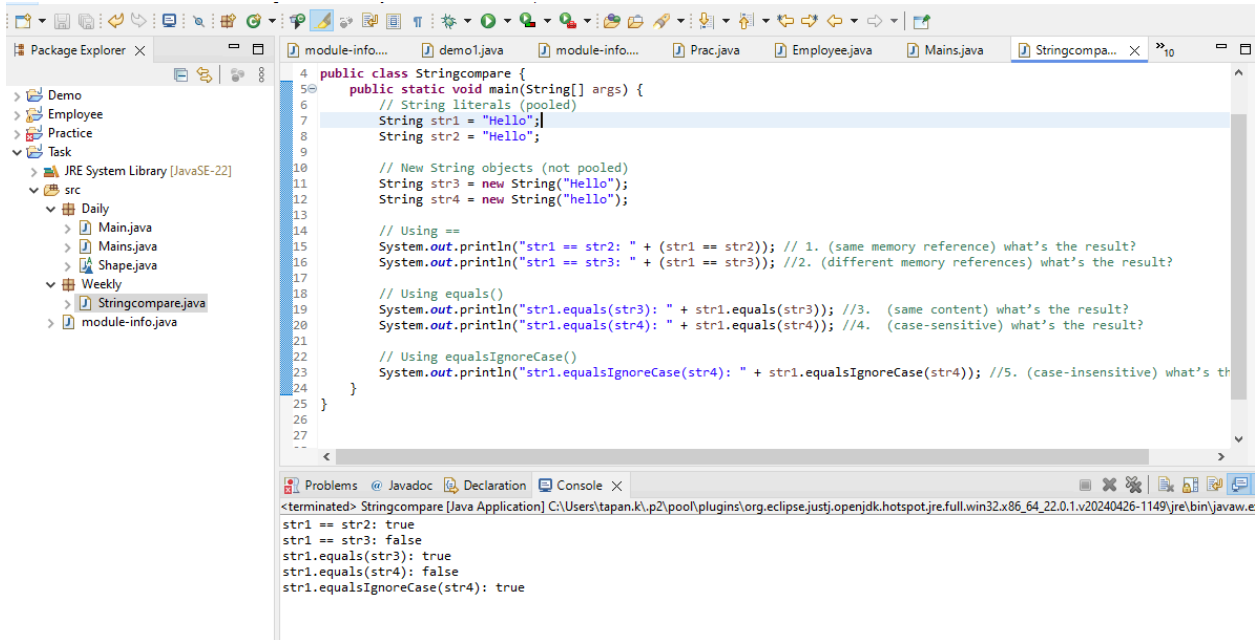
        // New String objects (not pooled)
        String str3 = new String("Hello");
        String str4 = new String("hello");

        // 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?
    }
}
```

**Output:-**



## 2. Find case 2 and mention the result for the statements using integers?

### Case2 Code:-

```

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)
```

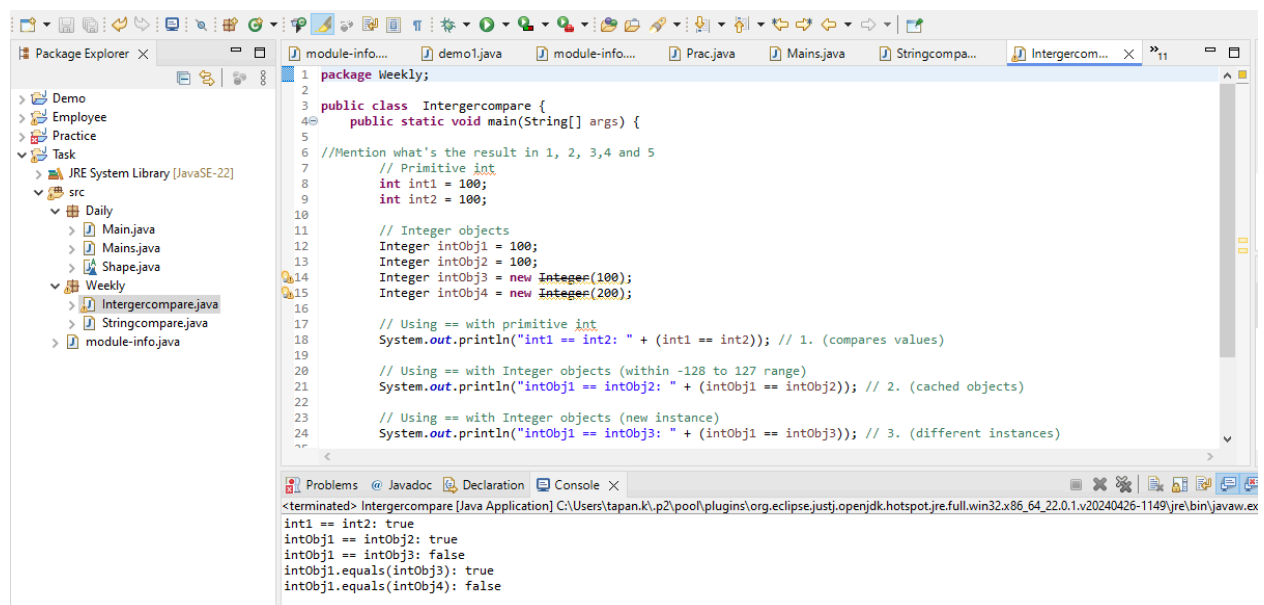
```
// Using equals() with Integer objects
```

```
System.out.println("intObj1.equals(intObj3): " + intObj1.equals(intObj3)); // 4. (same content)
```

```
System.out.println("intObj1.equals(intObj4): " + intObj1.equals(intObj4)); // 5. (different content)
```

```
    }  
}
```

Output:-



The screenshot shows the Eclipse IDE with a project named 'Weekly'. The 'src' folder contains several Java files, including 'Integercompare.java'. The code in 'Integercompare.java' is as follows:

```
1 package Weekly;  
2  
3 public class Integercompare {  
4     public static void main(String[] args) {  
5  
6         //Mention what's the result in 1, 2, 3,4 and 5  
7         // Primitive int  
8         int int1 = 100;  
9         int int2 = 100;  
10  
11        // Integer objects  
12        Integer intObj1 = 100;  
13        Integer intObj2 = 100;  
14        Integer intObj3 = new Integer(100);  
15        Integer intObj4 = new Integer(200);  
16  
17        // Using == with primitive int  
18        System.out.println("int1 == int2: " + (int1 == int2)); // 1. (compares values)  
19  
20        // Using == with Integer objects (within -128 to 127 range)  
21        System.out.println("intObj1 == intObj2: " + (intObj1 == intObj2)); // 2. (cached objects)  
22  
23        // Using == with Integer objects (new instance)  
24        System.out.println("intObj1 == intObj3: " + (intObj1 == intObj3)); // 3. (different instances)  
25    }  
}
```

The console output shows the following results:

```
<terminated> Integercompare [Java Application] C:\Users\tapan.k\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe  
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 ?**

**Case3 Code:-**

```
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();
        }
    }
}

```

Answer:-

## Closing Basic I/O Resources

In **Case 3**, the code uses the **try-with-resources statement**, introduced in Java 7, to manage the closing of the `BufferedReader` resource. The main advantage of this approach is its ability to automatically close resources like file readers, writers, and streams once the try block completes, regardless of whether it exits normally or due to an exception

## Key Differences:

- **Explicit Closing:** In the older approach, the resource (**`BufferedReader`**) must be closed explicitly in the **finally** block, requiring additional error handling in case closing fails.

- **Automatic Closing:** In the try-with-resources version, the resource is declared within the parentheses of the **try** statement, and it is automatically closed when the block exits, whether normally or due to an exception.

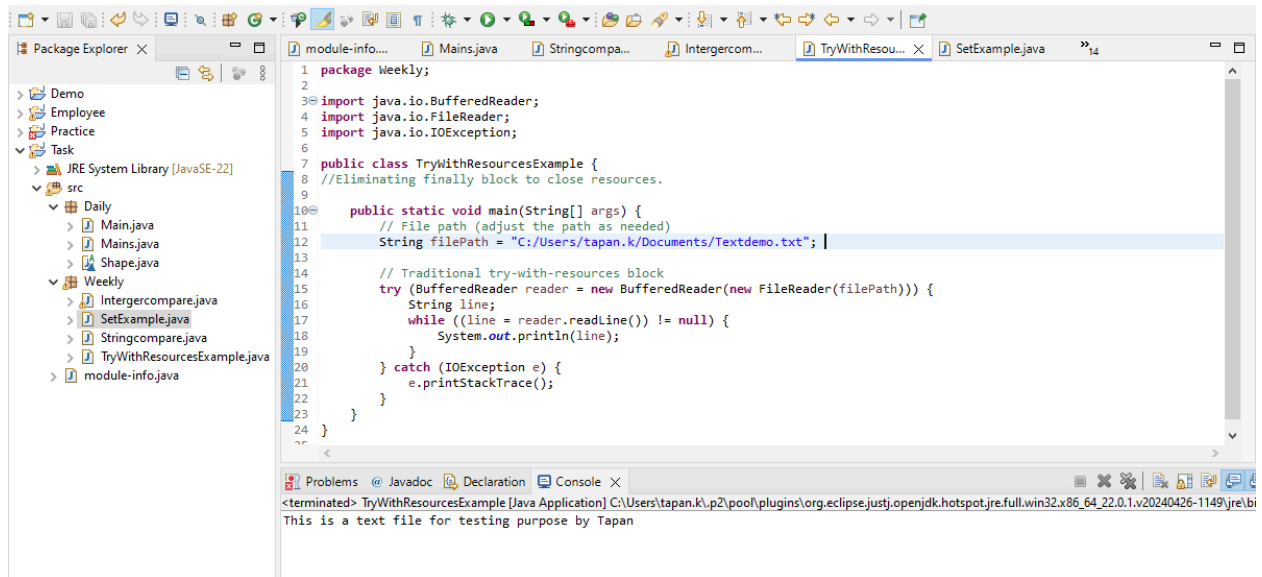
Updated code:

```
package Weekly;
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/tapan.k/Documents/Textdemo.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();
        }
    }
}
```

Output:-



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

##### Case4 Code:-

```
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);

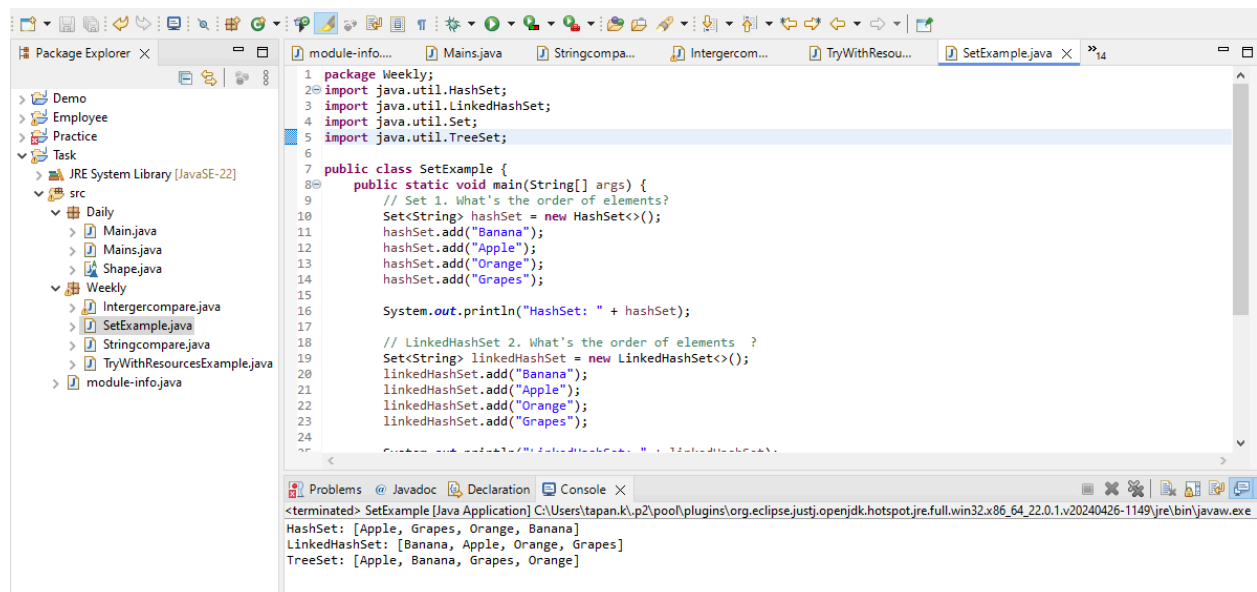
```

```

}
}

```

Output:-



The screenshot shows the Eclipse IDE with the 'SetExample.java' file open. The code in the file is as follows:

```

1 package Weekly;
2 import java.util.HashSet;
3 import java.util.LinkedHashSet;
4 import java.util.Set;
5 import java.util.TreeSet;
6
7 public class SetExample {
8     public static void main(String[] args) {
9         // Set 1. What's the order of elements?
10        Set<String> hashSet = new HashSet<>();
11        hashSet.add("Banana");
12        hashSet.add("Apple");
13        hashSet.add("Orange");
14        hashSet.add("Grapes");
15
16        System.out.println("HashSet: " + hashSet);
17
18        // LinkedHashSet 2. What's the order of elements ?
19        Set<String> linkedHashSet = new LinkedHashSet<>();
20        linkedHashSet.add("Banana");
21        linkedHashSet.add("Apple");
22        linkedHashSet.add("Orange");
23        linkedHashSet.add("Grapes");
24
25        System.out.println("LinkedHashSet: " + linkedHashSet);
26    }
27 }

```

The console output at the bottom shows the following results:

```

<terminated> SetExample [Java Application] C:\Users\tapan.k\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe
HashSet: [Apple, Grapes, Orange, Banana]
LinkedHashSet: [Banana, Apple, Orange, Grapes]
TreeSet: [Apple, Banana, Grapes, Orange]

```

## 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: [Apple, Grapes, Orange, Banana]

## **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]