3 – WEEK – TASK (10-08-2024 TO 11-08-2024)

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

Case1: (Code)

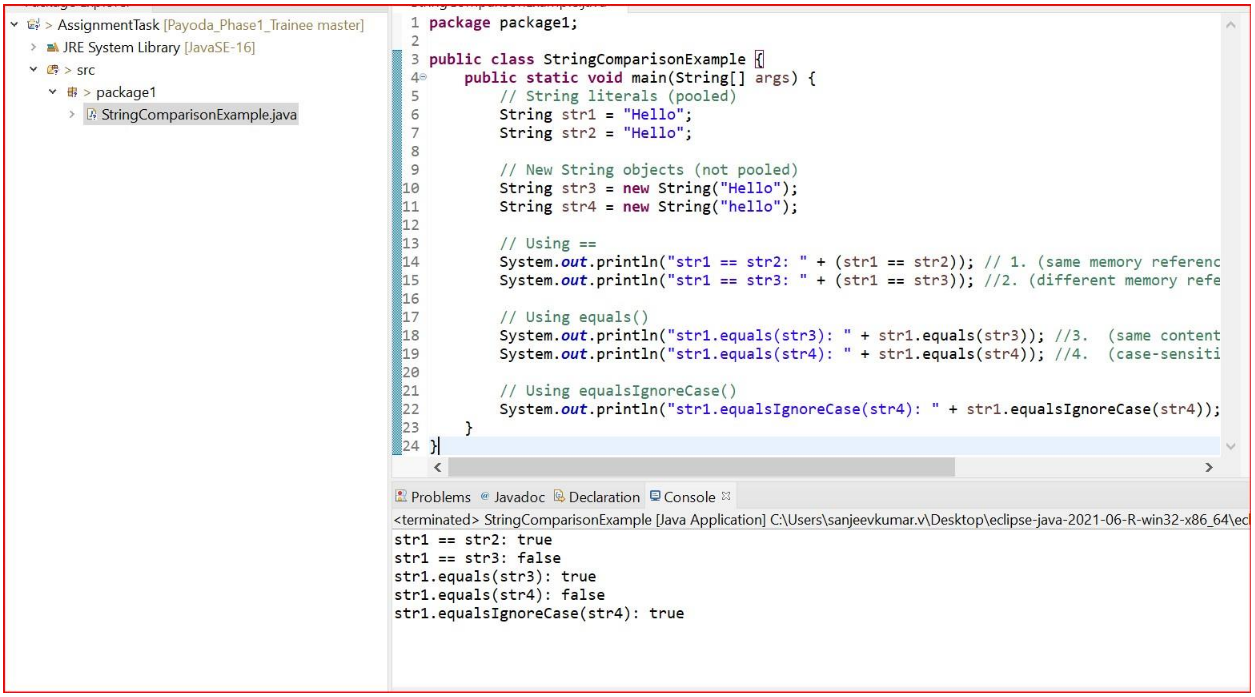
```
public class StringComparisonExample {
    public static void main(String[] args) {
        // String literals (pooled)
        String str1 = "Hello";
        String str2 = "Hello";

        // 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?
    }
}
```

Answer:

- str1 == str2: true
- str1 == str3: false

- str1.equals(str3): true
- str1.equals(str4): false
- str1.equalsIgnoreCase(str4): true

Output:



AssignmentTask [Payoda_Phase1_Trainee master]
JRE System Library [JavaSE-16]
src
package1
StringComparisonExample.java
Problems
Javadoc
Declaration
Console

```java
package package1;

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 referenc
        System.out.println("str1 == str3: " + (str1 == str3)); //2. (different memory refe

        // Using equals()
        System.out.println("str1.equals(str3): " + str1.equals(str3)); //3.  (same content
        System.out.println("str1.equals(str4): " + str1.equals(str4)); //4.  (case-sensiti

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

```
<terminated> StringComparisonExample [Java Application] C:\Users\sanjeevkumar.v\Desktop\eclipse-java-2021-06-R-win32-x86_64\ec
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**

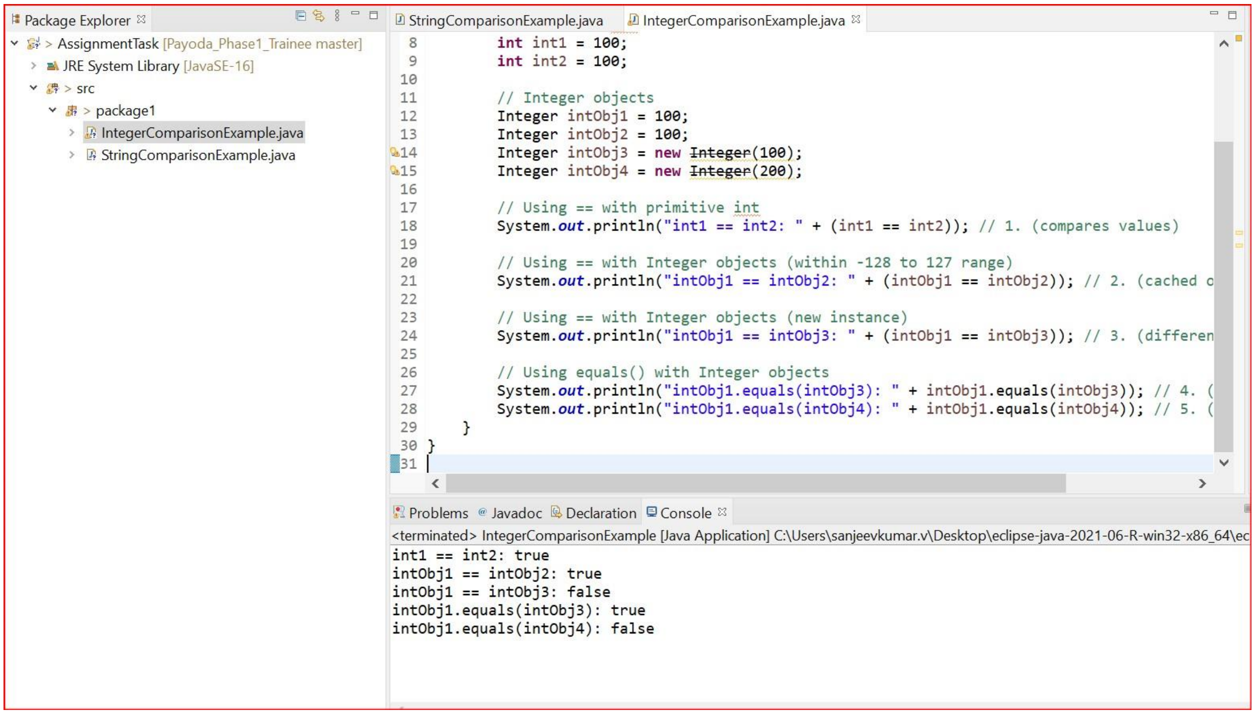
```
public class IntegerComparisonExample {
  public static void main(String[] args) {

//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)
    }
}
```

Answer:

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

Output:


Package Explorer
AssignmentTask [Payoda_Phase1_Trainee master]
JRE System Library [JavaSE-16]
src
package1
IntegerComparisonExample.java
StringComparisonExample.java
StringComparisonExample.java
IntegerComparisonExample.java
Problems
Javadoc
Declaration
Console

```java
        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 o

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

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

```
<terminated> IntegerComparisonExample [Java Application] C:\Users\sanjeevkumar.v\Desktop\eclipse-java-2021-06-R-win32-x86_64\ec
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**

```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();
    }
  }
}
```

Answer:

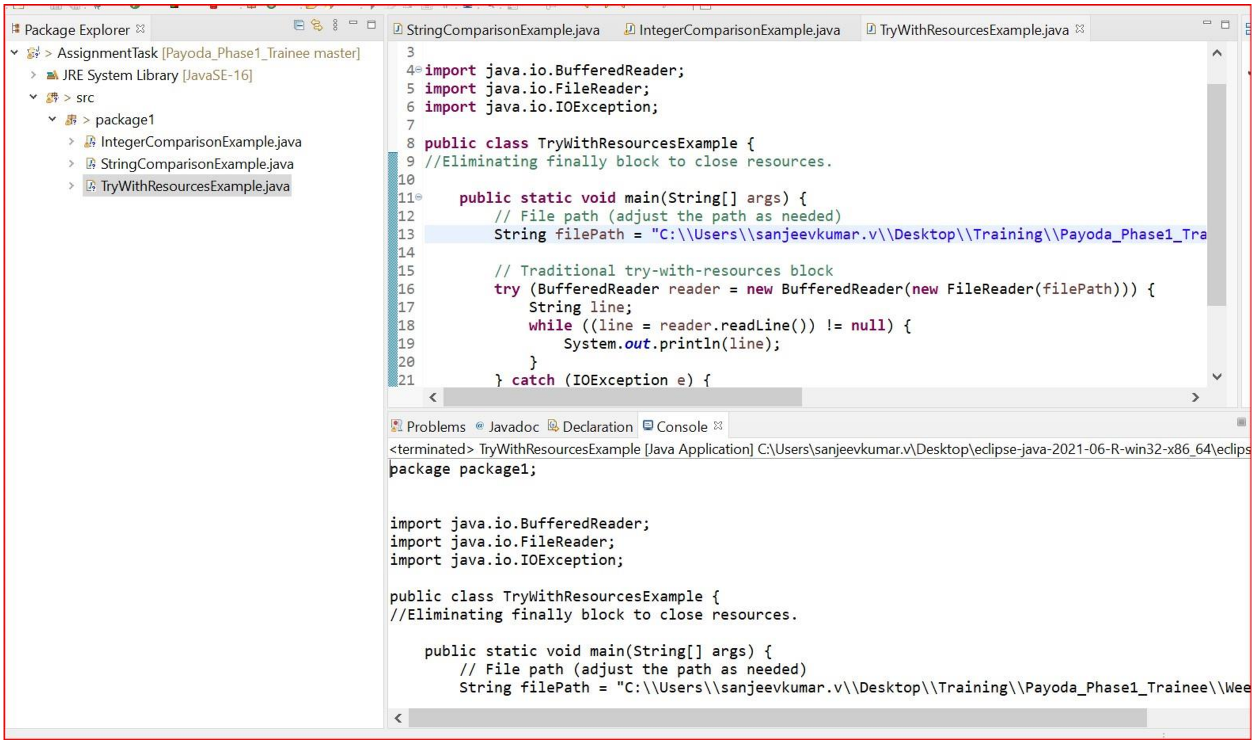
managing I/O resources such as files, streams, and readers is crucial to avoid resource leaks and ensure proper cleanup. The code provided uses the try-with-resources statement, which simplifies the management of such resources.

Automatic Resource Management:

In the try-with-resources statement, the `BufferedReader` and `FileReader` are both declared inside the parentheses of the `try` block. This ensures that:

- **Automatic Closing:** Both `BufferedReader` and `FileReader` are automatically closed when the try block exits, whether normally or due to an exception. This is managed by Java's AutoCloseable interface, which `BufferedReader` and `FileReader` implement.
- **No Need for Finally Block:** You do not need a `finally` block to explicitly close resources. This reduces boilerplate code and the risk of forgetting to close resources.

Output:


Package Explorer
AssignmentTask [Payoda_Phase1_Trainee master]
JRE System Library [JavaSE-16]
src
package1
IntegerComparisonExample.java
StringComparisonExample.java
TryWithResourcesExample.java
StringComparisonExample.java
IntegerComparisonExample.java
TryWithResourcesExample.java
Problems
Javadoc
Declaration
Console

```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 = "C:\\Users\\sanjeevkumar.v\\Desktop\\Training\\Payoda_Phase1_Tra

        // 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) {
```

```
<terminated> TryWithResourcesExample [Java Application] C:\Users\sanjeevkumar.v\Desktop\eclipse-java-2021-06-R-win32-x86_64\eclips
package package1;


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\\sanjeevkumar.v\\Desktop\\Training\\Payoda_Phase1_Trainee\\Wee
```

Difference from Traditional Resource Management

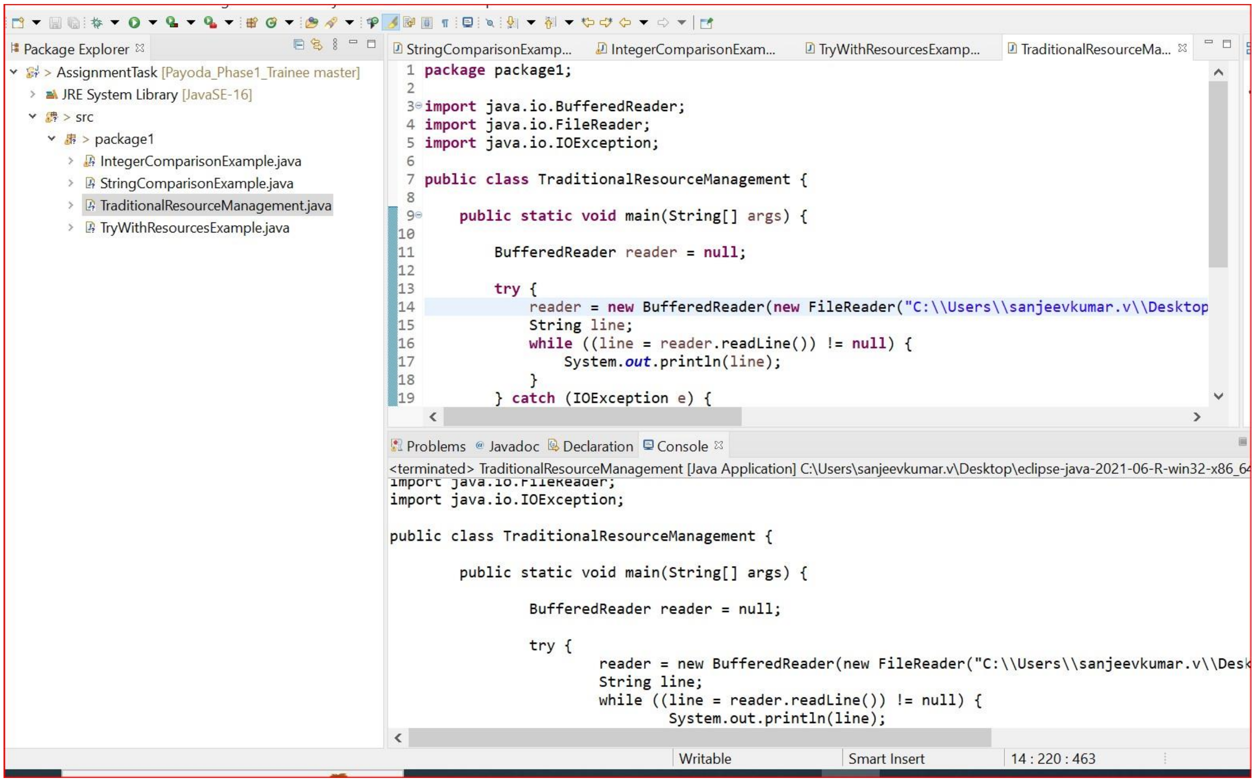
In traditional resource management, you would need to explicitly close resources using a `finally` block, which looks like this:

Code:

```java
package package1;

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

public class TraditionalResourceManagement {

public static void main(String[] args) {

BufferedReader reader = null;

try {
reader = new BufferedReader(new FileReader("example.txt"));
String line;
while ((line = reader.readLine()) != null) {
```

```
System.out.println(line);
}
} catch (IOException e) {
e.printStackTrace();
} finally {
if (reader != null) {
try {
reader.close();
} catch (IOException e) {
e.printStackTrace();
}
}
}
}
}
```

Output:



Package Explorer
> AssignmentTask [Payoda_Phase1_Trainee master]
> JRE System Library [JavaSE-16]
> src
> package1
IntegerComparisonExample.java
StringComparisonExample.java
TraditionalResourceManagement.java
TryWithResourcesExample.java
StringComparisonExamp...
IntegerComparisonExam...
TryWithResourcesExamp...
TraditionalResourceMa...
Problems
Javadoc
Declaration
Console
Writable
Smart Insert
14 : 220 : 463

```
package package1;

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

public class TraditionalResourceManagement {

    public static void main(String[] args) {

        BufferedReader reader = null;

        try {
            reader = new BufferedReader(new FileReader("C:\\Users\\sanjeevkumar.v\\Desktop
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
```

<terminated> TraditionalResourceManagement [Java Application] C:\Users\sanjeevkumar.v\Desktop\eclipse-java-2021-06-R-win32-x86_64

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

public class TraditionalResourceManagement {

        public static void main(String[] args) {

                BufferedReader reader = null;

                try {
                        reader = new BufferedReader(new FileReader("C:\\Users\\sanjeevkumar.v\\Desk
                        String line;
                        while ((line = reader.readLine()) != null) {
                                System.out.println(line);
```

Key Differences:

a. **Explicit vs. Implicit Resource Closure:**
   - **Traditional:** You must explicitly close each resource in the `finally` block.

   - **Try-with-Resources:** Resources are automatically closed at the end of the `try` block.

b. **Error Handling:**
   - **Traditional:** You need additional try-catch blocks within the `finally` block to handle potential exceptions when closing resources.
   - **Try-with-Resources:** The resources are closed automatically and any exceptions thrown during closing are suppressed, allowing the original exception to be propagated.

c. **Code Simplicity:**
   - **Traditional:** Requires more boilerplate code to manage resource closure and error handling.
   - **Try-with-Resources:** Simplifies code, making it more readable and less error-prone.

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);
    }
}
```

Answer:

Explanation of Set Types and Their Order:

HashSet:

- Does not guarantee any specific order of elements.
- The order of elements may appear random and can change over time.
- It provides constant time performance for basic operations (add, remove, contains).

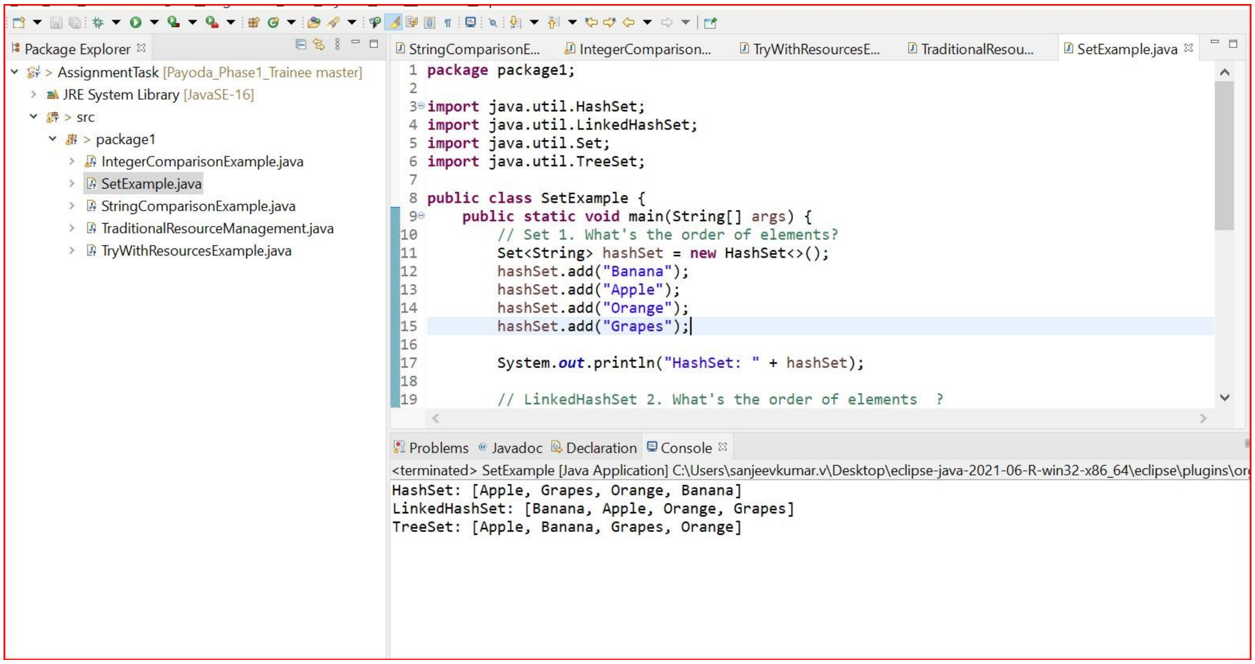
LinkedHashSet:

- Maintains the order of elements as they were inserted.
- Offers predictable iteration order.
- Provides performance similar to `HashSet` with the added benefit of predictable ordering.

TreeSet:

- Stores elements in a sorted order according to their natural ordering or a specified comparator.
- Implements a NavigableSet which allows for efficient searching and retrieval.
- Slower than `HashSet` and `LinkedHashSet` due to the need to maintain sorted order.

Output:


Package Explorer
AssignmentTask [Payoda_Phase1_Trainee master]
JRE System Library [JavaSE-16]
src
package1
IntegerComparisonExample.java
SetExample.java
StringComparisonExample.java
TraditionalResourceManagement.java
TryWithResourcesExample.java
StringComparisonE...
IntegerComparison...
TryWithResourcesE...
TraditionalResou...
SetExample.java
Problems
Javadoc
Declaration
Console

```java
package package1;

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  ?
```

<terminated> SetExample [Java Application] C:\Users\sanjeevkumar.v\Desktop\eclipse-java-2021-06-R-win32-x86_64\eclipse\plugins\or

```
HashSet: [Apple, Grapes, Orange, Banana]
LinkedHashSet: [Banana, Apple, Orange, Grapes]
TreeSet: [Apple, Banana, Grapes, Orange]
```