LAB 4 – Mutability

Student Name: Eryk Gloginski **(L00157413)**

Course: BSc Computing

Module: Secure Programming

Lecturer: Maria Griffin

Submission Date: 16/03/2023

1. Introduction:

Mutability is the ability of an object or data structure to be changed after its creation. Mutable objects can be modified in place, which means that their values can be changed without creating a new object. Other than the Mutable objects, there are also Immutable objects. Immutable objects are objects that cannot be changed after their creation. Examples of Immutable objects include Java classes such as String and Integer. Once a String or Integer object is created, their values cannot be changed.

2. Aims:

The aims of this lab are to:

- Use the code provided for the class **Course.java**.
- Find out if the **Course.java** is mutable (suggest changes if it is not) and explain why.
- Use a tester class to see if you can manipulate the details of **Course.java**.
- Document all findings.

3. Method:

The method used in this lab to check if the class **Course.java** is mutable or immutable is manual code review using a tester class called **CourseTester.java** as a helper. Manual code review, as the name implies, involves a manual review of the code by at least one developer to identify and correct problems. It can be done formally or informally and might involve a checklist or other review guidelines. It is useful for detecting issues that automated tools may miss.

4. Results:

I believe that the **Course.java** class is immutable. To verify that, I have created a class called **CourseTester.java** where construct a Course object called testCourse with Test Course as a name and a new Date object.

```java
import java.util.Date;

public class CourseTester {
    Run | Debug
    public static void main(String[] args) {

        // declare testCourse as a new Course object
        Course testCourse = new Course(courseName: "Test Course", new Date());
```

I first attempt to return the **courseName** and then display it. I attempt to change the value of the String after and it's not possible to change the original value because **courseName** is a String object and **Strings are not mutable**.

```java
// get testCourse courseName and check if it's mutable or not
String testCourseName = testCourse.getCourseName();
System.out.println("before changing: " + testCourseName);
testCourseName = "Another Test Course";
System.out.println("testCourseName after changing: " + testCourseName);
// it's a String therefore it is not mutable
System.out.println("testCourse.getCourseName() after changing: " + testCourse.getCourseName());
```

Next, I try to return the **startDate** and display it. I attempt to change it using the **deprecated setYear()** method and it's not possible to change the original value because of two reasons that I can see. **Reason 1 being**: the **startDate** variable is declared as a private final, which means that once it's

initialized in the constructor, it cannot be changed. **Reason 2 being**: the **startDate()** method returns a **new Date** object from the **startDate**, which means that the returned **Date** object is a separate instance from the one that is stored in the **startDate** variable.

```
// get testCourse startDate and check if it's mutable or not
Date testStartDate = testCourse.startDate();
System.out.println("before changing: " + testStartDate);
testStartDate.setYear(year: 133);
System.out.println("testStartDate after changing: " + testStartDate);
// it's not mutable because it's not
System.out.println("testCourse.startDate() after changing: " + testCourse.startDate());
```

```
private final Date startDate;
```

```
public Date startDate() {
    return new Date(startDate.getTime());
}
```

The **output** shown below is the code of the **CourseTester.java** class being run:

```
a4f04967b8809ebe\redhat.java\jdt_ws\LAB2 REPORT_d699488b\bin' 'CourseTester'
before changing: Test Course
testCourseName after changing: Another Test Course
testCourse.getCourseName() after changing: Test Course

before changing: Wed Mar 08 10:24:39 GMT 2023
testStartDate after changing: Tue Mar 08 10:24:39 GMT 2033
testCourse.startDate() after changing: Wed Mar 08 10:24:39 GMT 2023
PS D:\Year 3\Semester 6\Secure Programming\LAB2 REPORT>
```

By checking the Java documentation, we can see that the Date class has 4 deprecated constructors and 18 deprecated methods!

5. Conclusion:

In conclusion, mutability refers to the ability of an object to be modified after its creation while immutability refers to the object's state of not being able to be changed after its creation.

There are plenty of benefits of immutable classes, such as: easier testing, better design, better performance and thread-safety.

The code that we were provided for the Course.java class is an example of an immutable class. The state of the class is set in the constructor and once it's been set, it cannot be changed. The startDate is a private final which makes sure that the variable cannot be modified after the initialization of the object. Additionally, the startDate() method returns a new Date object rather than the original one that was initialized.

6. References:

https://docs.oracle.com/javase/8/docs/api/java/util/Date.html