**CA 3 - Lab 2 – Mutability Report**

Daniel Gallagher

L00158616

[L00158616@atu.ie](mailto:L00158616@atu.ie)

7th May 2023

# Introduction

In this lab, I'll look at the idea of immutability and how it affects the security and dependability of code. To determine whether the provided code is mutable or immutable, I will analyse it and make the necessary adjustments to ensure immutability. I'll investigate the Date object and its implications as well.

# Aims/Objectives

1. Determine if the given program is mutable or immutable and modify it if necessary.
2. Create a tester class to demonstrate the ability or inability to manipulate the course details.
3. Research the Date object and its implications.
4. Write conclusions about the findings and the impact of immutability in programming.

# Method

**Step 1: Analyse the given code for mutable or immutable behaviour.**

Using the attributes and methods of the provided Course class, I carefully examined it in this step to find mutable components (Fig. 1). I specifically looked for the presence of mutable objects like the Date object, access modifiers, and the use of final keywords.

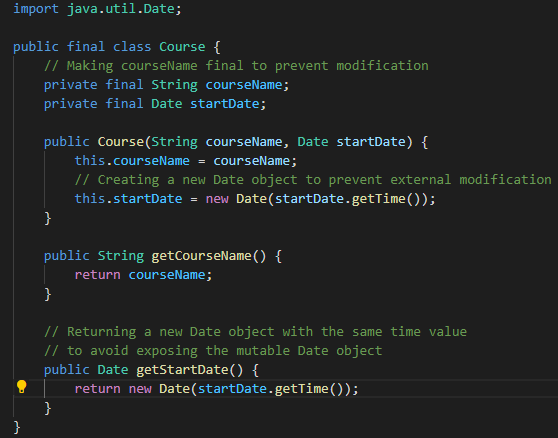
A picture containing text, screenshot, software, display

Description automatically generated

*Fig 1 – The initial Course class*

**Step 2: Modify the code if required to ensure immutability.**

I determined that the "courseName" attribute was mutable because it had not been declared final based on the analysis from Step 1 of the analysis process. I made the required adjustments to the Course class, which included handling the mutable Date object in the constructor and getter method (Fig. 2) and adding the final keyword to the "courseName" attribute.



*Fig 2 – The modified Course class with immutability*

**Step 3: Create a tester class to demonstrate the code's behaviour.**

To show the immutability of the modified Course class, I created a tester class called CourseTester in this step (Fig. 3). The tester class creates a Course object and tries to change its properties. The initial and modified values are then printed to demonstrate the Course object's immutability.

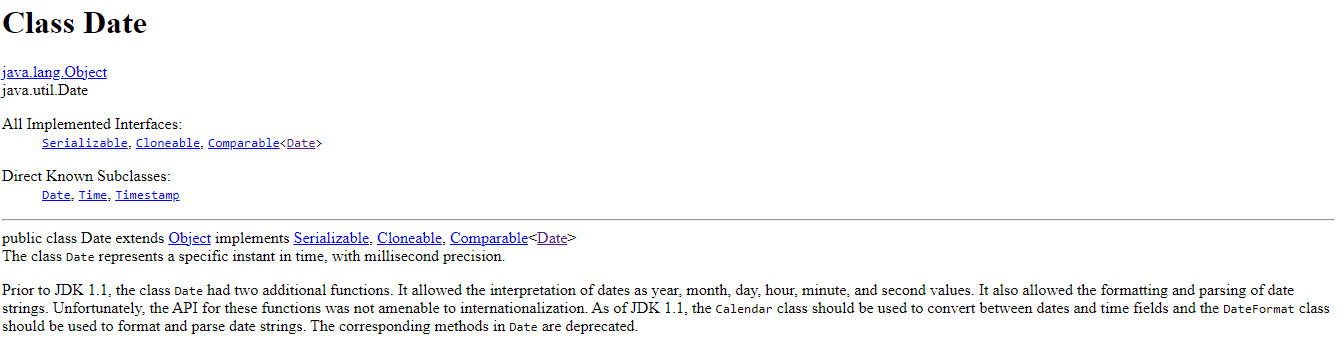
A picture containing text, screenshot, software, font

Description automatically generated

*Fig 3 – The CourseTester class that tests the immutability of the Course class.*

**Step 4: Research the Date object.**

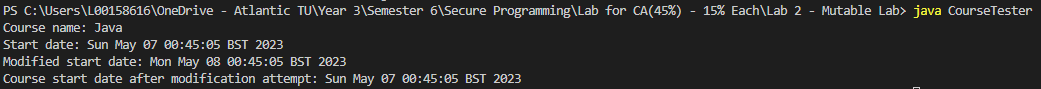
To gain a better understanding of the Date object and its implications on immutability, I conducted research by reviewing the Java documentation and other online resources (Fig. 4). I focused on the mutable nature of the Date object and explored ways to handle it when creating immutable classes.



*Fig 4 – Java documentation on the Date object.*

# Results and Testing

After analysing and modifying the code, I created a tester class to demonstrate the behaviour of the modified code. The output of the CourseTester is shown below in Figure 5.



*Fig 5 - Output of the CourseTester.java*

**Is it Immutable or Mutable?**

The given code was mutable because the "courseName" attribute in the provided code could be changed after the object was created because it was not declared final. By making the "courseName" attribute final, eliminating any setters that may have existed, and handling the mutable Date object defensively, I was able to make the class immutable.

**Date Object**

Because the Date object is mutable, using it in an immutable class may present problems. To avoid external modification, the provided code uses a new Date object in the constructor and returns a new Date object with the same time value for the "getStartDate()" method. To further improve code security and dependability, I looked into other options, such as utilizing the java.time package, which offers immutable date and time classes like LocalDate, LocalTime, and LocalDateTime.

# Conclusion

Immutability is crucial for upholding code security, dependability, and the ease of object state reasoning. The "courseName" attribute was made final, which rendered the given code immutable. The mutable Date object was also handled by me by creating a new object as needed, preventing outside modifications. The significance of immutability and the potential dangers of mutable objects like the Date object have both been illustrated in this lab. To ensure robust and maintainable code, immutability must be considered when designing classes, especially when working with shared data or concurrent programming.

# References

* Oracle. (2021). Date (Java SE 10 & JDK 10). Retrieved from <https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/util/Date.html>.