

OOP COURSEWORK REPORT – csi-5-oop

Student ID: 4214293



Contents

[Introduction 2](#_Toc184835532)

[Classes and their attributes and methods: 3](#_Toc184835533)

[ Square Class: 3](#_Toc184835534)

[ Board Class: 4](#_Toc184835535)

[ TemplateShip Class: 6](#_Toc184835536)

[ AmmoDump Class: 7](#_Toc184835537)

[ Submarine Class: 8](#_Toc184835538)

[ Tank Class: 8](#_Toc184835539)

[ Destroyer Class: 9](#_Toc184835540)

[ Battleship (Sea) Class: 9](#_Toc184835541)

[ AircraftCarrier Class: 10](#_Toc184835542)

[ Base (Land) Class: 10](#_Toc184835543)

[ Aeroplane Class: 11](#_Toc184835544)

[ BoardButton Class: 12](#_Toc184835545)

[Conclusion 13](#_Toc184835546)

[References 14](#_Toc184835547)

# Introduction

For this assignment, I have been assigned to add additional changes and improvements to the battleships application that I had developed over the course of the module. The main goal of this coursework is to enhance and modify the gameplay of the Battleships game by implementing land and sea parts on the board with the use of OOP techniques.

The changes that will need to be made to original game change, is introducing new ship types such as Tank, Base, Ammo Dump, Destroyer, Aeroplane, Battleship, Submarine, Aircraft Carrier, and Tank. Another thing to consider is implementing a new attribute SquareType which will determine which is Land or Sea square type. Each ship will have different movements and placements whilst playing the game.

The OOP techniques that will be used in the game is inheritance, polymorphism, encapsulation, abstraction which will allow the game to have modular, maintainable, and reusable code with utilising an OOP approach. By adding in new classes and methods as well as modifying old classes and methods will ensure that new rules in the game will align with the original framework.

This report discusses the ideas of these principles are used to enhance the game mechanics and gameplay of the Battleships game. Code snippets will be provided along with the explanation of the new game features being added into the game itself.

# Classes and their attributes and methods:

## Square Class:

Since there are two types of different lands trying to be implemented, land and sea, I would need to incorporate a type of object that will be differentiated between the land and sea. To do this I will need to add an attribute in the Square class called squareType, this will allow the program to differentiate between land and sea, allowing it to know which type of Ship will be placed onto the type of land.

Code Implementation:

A screenshot of a computer program

Description automatically generated

**Figure 1.1 – Code implementation of the Square class**

Figure 1.1 shows the implementation of the square class, and inside the class, it contains the attributes, constructors and the different methods being used. The reason why I decided to implement the Square class like this is because, if I were to use another class to implement this, I would not be utilising the Square class’s functionality. With the help of the Enum (squareType) which allows me to check the land type and place those specific ships on those land types accordingly. The benefit of using Enum is its extensibility, meaning that if I needed to add different land types into the game, then it could easily be done so by typing the different land types inside the Terrian class itself. Enums also can improve the readability of the code and make it easier to maintain in the long run – if the code requires more features (Glushach, 2023b).

## Board Class:

A computer screen shot of a program

Description automatically generated

**Figure 1.2 – Different implementation of the Board Constructor**

Changes in the Board Constructor will need to be made to accommodate the changes for the Square class. For instance, as the 2D board is being initialized, each Square cell object will be half SEA or LAND-type square. Overall, it iterates through the rows and columns of the board, by setting predefined regions.

To load the new layout of the board, a new method called loadBoard() will be used to load the layout of the board which will be used to play the game.

The getSquare() method will check and return the SquareType depending on the coordinates that are marked on the board.

placeShip() method, will also need to be changed to accommodate the changes that have been made within the game since we have the Square class with an object type SquareType. The change that will need to be made is that we need to loop through each square of the board and check to see which square the ship will take based on its square type with the help of the board attribute.

Another thing which can be used to prevent invalid ship placements depending on the specific square types is using exception handling. The try part of the code will contain the code that is likely to throw an exception, and the catch will handle specific types of expectations depending on the user input. For example, when the exception is thrown or when there is a missing input by the user, like a different data type other than an integer greater than the range.

Another alternative is using IllegalArgumentException, which is thrown when the user enters invalid coordinates or tries to place a specific ship on the wrong square of the board.

The new canPlaceOnSquare() method from the TemplateShip Class will help in checking whether that ship can be placed in that specific square. Worst case, say any square is not a suitable terrain for the ship to be placed in, otherwise, the program will break out of the code and will continue to search for a valid placement for that ship and its SquareType.

On the other hand, if the loop is completed successfully without encountering a collision on the board, the ship can be placed on the board.

An Isvalidmove() method will check the type when deciding the valid moves. In this case of the game, a land ship type can’t go to the sea part of the board, thus making the move invalid. To do this, it will check the ship type and its SquareType and will adjust accordingly throwing an exception error to tell the user that the move they have inputted/selected is invalid.

## TemplateShip Class:

Since Ammo Dump Class is a component of the Battleships game that will be used whilst the game is being played. To implement this class into the game, I have decided to create an AmmoDump Class that uses the Battleships.ship package and inherits properties from the TemplateShip class.

Before importing the TemplateShip Class I realised that there was no attribute related to a square count, and squareType being initialised in the TemplateShip Constructor method.

A screen shot of a computer

Description automatically generated

**Figure 1.3 – TemplateShip changes**

Figure 1.3 shows the changes made to the TemplateShip class, now that squarecount, and type attributes have been initialized it can be used when the other ship type classes inherit properties from.

Adding in a new method called canPlaceOnSquare(), which will be used to check whether the ship can be placed on a particular square type (i.e. Land or Sea). This can be determined by the type of Ship and the Square type itself.

## AmmoDump Class:

A screen shot of a computer program

Description automatically generated

**Figure 1.4– AmmoDump Class implementation**

The AmmoDump class inherits properties from the TemplateShip Class, allowing for code reusability, i.e being able to reuse the code from parent classes (in this case the TemplateShip Class), which reduces time spent on creating new and separate classes (Glushach, 2023a).

The same implementation for inheritance from the TemplateShip class applies to the other Ship classes.

## Submarine Class:

A computer screen shot of a program code

Description automatically generated

**Figure 1.5 – Submarine Class implementation**

The Submarine Class before was inheriting properties from the SimpleShip Class, but now it is inheriting from TemplateShip. The reason why I choose to go with all the Ship Classes inheriting from TemplateShip is because it includes all the methods and attributes that are needed to play the game. This is due to it being able to enhance its functionality with the help of grid-based templates like the Board class, and the visualizations of the ships on the board.

## Tank Class:

A screen shot of a computer program

Description automatically generated

**Figure 1.6 – Tank Class implementation**

For the game, I needed to create a new class called tank, which is a land-based class. It inherits from its parent class TemplateShip and has a pattern of 1 only, meaning that it only can move one place at a time on the land part of the board.

## Destroyer Class:

A screen shot of a computer program

Description automatically generated

**Figure 1.7 – Destroyer Class implementation**

The destroyer ship before inherited from the Simpleship class but now instead inherits from the TemplateShip. Inside the Destroyer constructor I have added in the square count to be equal to 4 and the squareType to be equal to Sea.

## Battleship (Sea) Class:

A screen shot of a computer program

Description automatically generated

**Figure 1.8 – Battleship Class implementation**

The Battleship also inherited from the Simpleship class before, now inherits from the TemplateShip. Inside the Battleship constructor, I have added the square count to be equal to 4 and the TerrianType to be equal to Sea.

## AircraftCarrier Class:

A computer screen shot of a program code

Description automatically generated

**Figure 1.9 – AircraftCarrier Class implementation**

## Base (Land) Class:

A screen shot of a computer program

Description automatically generated

**Figure 1.10 – Base (Land) Class implementation**

To avoid confusion whilst playing the game the Base and Battleship Ship type have different symbols for Base in this case Bl and for Battleship Bs.

## Aeroplane Class:

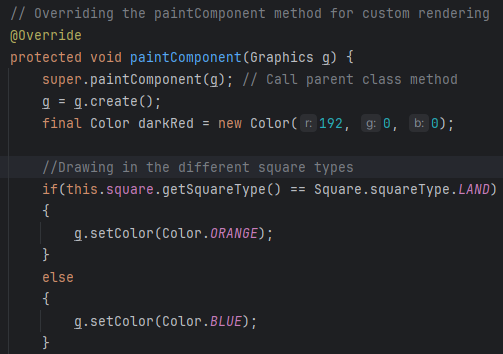
A screenshot of a computer program

Description automatically generated

**Figure 1.11 – Aeroplane Class implementation**

## BoardButton Class:

In the BoardButton Class the changes which need to be made is to the paintComponent() method, to allow it visually display the different square type representations on the board itself.



**Figure 1.11 – paintComponent method changes**

The code snippet above, will check whether the SquareType is a Land type or Sea type. If it is a Land Square type it has an orange square colour, and if it is a Sea Square type it has a blue square colour.

# Conclusion

The changes that have been made to the Battleships, the use of inheritance between the TemplateShip class (parent class), and child classes, (Ammo Dump, Destroyer, Aeroplane, Battleship (Sea), Base (Land), Submarine, Aircraft Carrier, and the Tank) all inherit properties from the TemplateShip class. Inheritance from the parent class essentially allows to create of an extensible and reusable code without needing to create a new separate class, as this leads to being more time-efficient (GeeksForGeeks, 2017). Another example, in the battleships code is TemplateShip inherits properties from the Ship class. The ship class provided a foundation for the TemplateShip class and as a result, the TemplateShip class became more extensible in terms of adding more functionality, methods and attributes.

Encapsulation, in the Square class, hides the details of the implementation of the code structure such as the methods being used and it allows for other classes to use the object SquareType from that class (Shahzad, 2013). The benefit of using encapsulation in OOP, provides a way for the developer to be able to store values in those attributes, and hide them while

The TemplateShip class also allows polymorphism and inheritance, to occur. The idea of objects from different child classes are treated as the objects that come from the parent class. Allowing for an extensible and flexible approach to the Battleships game (Glushach, 2023a).

In the TemplateShip class, the methods getWidth() and getHeight() are overridden, since these methods were originally in the Ship class. Overriding allows for the changes to be made to the superclass, and as a result, those changes are made in the child class for those methods that are overridden. Thus, making the code more maintainable and reducing the risks of errors from taking effect on the code itself (Leon Lovett, 2023).

The TemplateShip class, provides a foundation for the ship objects, by hiding the most common attributes and methods used by the other Ship classes. With the help of abstraction, it allows for focus on specific characteristics of each ship that will be used in the game, without needing to understand the details of how the ships are being implemented.

Overall, the use of OOP reduces errors, by having separate classes and their attribute, and methods, which in turn makes debugging code easier, as this ensures that the code is maintained over time. The design of the code also allows for updates and changes to be made to the game more easily compared to a non-OOP approach.

# References

1. Coursera (2023). *What Is Encapsulation?* [online] Coursera. Available at: https://www.coursera.org/articles/encapsulation [Accessed 22 Nov. 2024].
2. GeeksForGeeks (2017). *Inheritance in Java - GeeksforGeeks*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/inheritance-in-java/ [Accessed 22 Nov. 2024].
3. Glushach, R. (2023a). *Breaking Down the Pros and Cons of Inheritance and Composition in Object-Oriented Programming*. [online] Medium. Available at: https://romanglushach.medium.com/breaking-down-the-pros-and-cons-of-inheritance-and-composition-in-object-oriented-programming-9e79eb811c2c [Accessed 19 Nov. 2024].
4. Glushach, R. (2023b). *Unlocking the Benefits of Java’s Enums and Records for Efficient Programming*. [online] Medium. Available at: https://romanglushach.medium.com/unlocking-the-benefits-of-javas-enums-and-records-for-efficient-programming-85c30b5de4e [Accessed 13 Nov. 2024].
5. Leon Lovett. (2023). *The Purpose and Benefits of Method Overriding in OOP - Leon Lovett*. [online] Available at: https://leonlovett.dev/the-purpose-and-benefits-of-method-overriding-in-oop/ [Accessed 24 Nov. 2024].
6. Shahzad (2013). *Why encapsulation is an important feature of OOP languages?* [online] Stack Overflow. Available at: https://stackoverflow.com/questions/18300953/why-encapsulation-is-an-important-feature-of-oop-languages [Accessed 22 Nov. 2024].