****

**‘Over & Under’**

**Java Program Report**

**Software Development 1**

**SET07402**

Metric Number – 40456376

Word Count - 772

Contents

[1. Introduction 2](#_Toc26717241)

[2. Program Structure 2](#_Toc26717242)

[2.1 Figure 1 – ‘Over and Under’ Class Descriptions 2](#_Toc26717243)

[2.2 Figure 2 – Code from the ‘Over and Under’ GameMain Class 3](#_Toc26717244)

[2.3 Figure 3 – Code from the game() method in the Game Class 3](#_Toc26717245)

[2.2 Figure 3 – ‘Over & Under’ Class Diagram 4](#_Toc26717246)

[3. Extensibility 5](#_Toc26717247)

[4. Advanced Features 5](#_Toc26717248)

[5. Conclusion 5](#_Toc26717249)

1. Introduction

It essential that a program is robust, extensible and soundly programmed in order to limit errors, provide future expandability, and to mitigate against potential exploits. ‘Over and Under’ is a betting game which in the real world could be manipulated with malicious intent in order to benefit the user monetarily. For this reason, the program (including its classes, methods and variables) has been designed in such a way as to avoid pitfalls which might permit these exploits. This report will describe an implementation of the ‘Over and Under’ program by explaining the logic behind each class, method and variable with the aid of a Unified Modelling Language (UML) Class Diagram. In addition to this, the report will explore why this program should be considered extensible and upgradeable. Finally, it will explain the advanced features included in the program and the approach taken in order to implement them.

1. Program Structure

When developing ‘Over and Under’ it was important that the program was structured in a logical way, as it should be with all programs developed using an object-oriented programming (OOP) language. This meant that the program had to have sensible classes, useful reusable methods, and practically named variables that followed all conventions and expected syntax. As seen in Figure 2, every class in the program derives from the ‘Game’ Superclass which in turn derives from the ‘GameMain’ Superclass. ‘GameMain’ contains the main() method which calls the game() method from the ‘Game’ class which then runs the game in the correct order by calling methods from each subclass as needed. By starting the program this way, it meant that there was no need to have main() methods in each class which could quickly become cumbersome as the program grows and develops further, as would be expected in a real-world setting.

## **2.1 Figure 1 – ‘Over and Under’ Class Descriptions**

|  |  |
| --- | --- |
| **Class Name** | **Description** |
| GameMain (Superclass) | This class holds the main() method and calls the game() method from the Game superclass to run the program/start the game. |
| \* Game (Superclass) | This class inherits all of the methods from the Player, PlayerGuess, Stake, Dice and LeagueTable classes. Using these methods, the Game class is able to run through each of its own methods in the correct order to play the game. The anotherGo() method loops through the game again if the user choses so. |
| \*Stake | This class is simple, containing getters and setters to be used by the Game superclass and a Stake object constructor. |
| \*Player | This class includes all of the necessary information that is needed about the player of the game. The playerInfo() method collects some of this information by using input dialogs. |
| \*PlayerGuess | This class includes methods to ask the player what their guess is and to turn that guess into a String for use in the Game class. |
| LeagueTable | This class utilises an ArrayList, Comparator, LinkedHashMap and a TreeMap in order to print the High scores of the game in order each time the game is run. It uses the Map .clear() method to clear the sorted map each time in order to avoid duplication of the High score list. |
| \*Dice | This class creates a dice object, setting the maximum and minimum it can roll, and utilises an algorithm which makes use of the Math.random() method to assign a number to the getRoll() method. This is then called by the Game class as and when needed. |

\*Includes getters and setters in order to ensure global variables can remain private, improving program security.

## **2.2 Figure 2 – Code from the ‘Over and Under’ GameMain Class**

**package** overAndUnder;

**public** **class** GameMain {

**public** **static** **void** main(String[] args) {

Game g1 = **new** Game (); //Creating an object of the class game

g1.game(); //Starts the program

}

}

## **2.3 Figure 3 – Code from the game() method in the Game Class**

**public** **void** game() {

gameStarts();//This is the main start screen for the game

gameRules();//Tells the player the rules and asks if they understand

chooseStake();//This allows the user to choose if they want to change the stake of the game

playerStartStats();//Sets player starting data

gameInfo();//Asks the player for their name & alias

//The game plays when the player has enough money/turns to play the game

**while** (Game.*player1*.getPlayerTurns() > 0 && Game.*player1*.getMoneyInBank() > 0) {

gamePlay();

}

gameOver(); //Game ends and the players place in the League Table is displayed

anotherGo();//Player will be asked if they want to play again or not

//This checks if the player wants to play again

**while** (getPlayerGoesAgain() == **true**) {

gameAgain();//Restarts the game if the player wants to play again

}

**while**(getPlayerGoesAgain() == **false**) {

System.*exit*(0);//exits the game if the player does not want to play again

}

}

## **2.2 Figure 3 – ‘Over & Under’ Class Diagram**

A close up of text on a white background

Description automatically generated

1. Extensibility

‘Over and Under’ was designed with extensibility in mind, as any good program should be. There are several elements of the program that make it extensible, most notably so the fact that each subclass (Dice, Player, LeagueTable, Stake, PlayerGuess) only contains the information that would be relevant to that object in real life. This means that if another game were to be designed to sit alongside ‘Over & Under’ in the program, with its own ruleset and goals, these classes could be used to help create it with very little tinkering by a future software engineer needed. Similarly, if the ‘Over and Under’ game were to expand its rules and goals, it could be done with relative ease.

In addition to this, by creating a Player class which houses all of the information about a player, this means that the program could be expanded to include several players who could play the game at once. The addition of the league table would also provide a way for these players to compare their performance and would provide a competitive drive to play the game again. Further expansion could come in the form of additional modes ‘stake’ modes which is made possible by the fact that the code within the Stake object can edited easily.

1. Advanced Features

The advanced features in this program provide proof of its extensible nature and include: a league table and the ability for the player to change the stakes of the game if they so desire.

**League Table**

In order to implement the league table, a TreeMap to add both the players game data and the data of other placeholder players to was created. This data was then sorted using a bespoke comparator and placed into two ArrayList’s (Key & Value) by using a for loop to iterate over it. The data from each ArrayList was then displayed for the user using a dialog box and then immediately cleared using the .clear method to ensure that if the player chose to play the game again that the data was not duplicated.

**Changing the Stake**

To enable user to change the stake, a ‘Stake’ class was created which held getters and setters for the stake object as well as a stake constructor. These getters and setters are called by the Game class when the user chooses their stake. For example, if the player choses to play the normal mode the code would be set to:

* getTakeTurnMoney(1);
* getStakeWin2(2);
* getStakeWin5(5);

While if the player opted to play the ‘money to burn’ mode the code would be set to:

* getTakeTurnMoney(3);
* getStakeWin2(6);
* getStakeWin5(15);

1. Conclusion

In conclusion, ‘Over and Under’ has been developed in such a way as to ensure future expandability and security. By using sensible classes, methods and variable names the program should be easy to understand if further changes/upgrades are deemed necessary – the advanced features provide proof of this.