

**National College of Ireland**

**Project Submission Sheet – 2014/2015**

**School of Computing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name:** | …Steven Gaynor……………………………………………………………………… | | |
| **Student ID:** | …14108038……………………………………………………………………………………………… | | |
| **Programme:** | …HDCLOUD………………………………………………… | **Year:** | ……2014/15… |
| **Module:** | …Software Development………………………………………………………………………… | | |
| **Lecturer:** | …Adriana Chis……………………………………………………………………………………… | | |
| **Submission Due Date:** | …7/12/2014……………………………………………………………………………………………… | | |
| **Project Title:** | …Designing and Implementing “Rock-Paper-Scissors” in Java…………… | | |
| **Word Count:** | …1386……………………………………………………………………………………………………… | | |

**I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.**

**ALL internet material must be referenced in the bibliography section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.**

|  |  |
| --- | --- |
| **Signature:** | ………………………… |
| **Date:** | …………6/12/2014……………………………………………………………………………………………… |

**PLEASE READ THE FOLLOWING INSTRUCTIONS:**

1. Please attach a completed copy of this sheet to each project (including multiple copies).

2. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.

3 Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

|  |  |
| --- | --- |
| **Office Use Only** | |
| Signature: |  |
| Date: |  |
| Penalty Applied (if applicable): |  |

SDEV Project Report

A Report into the design and implementation of a “Rock Paper Scissors” Game in Java

Written by Steven Gaynor (HDCLOUD)

Student Number: 14108038

# Introduction

The main objective of this project is to design a working game of the popular Rock Paper Scissors using the Java programming language. Commencing the process, it was obvious to start with a basic and logical approach to the problem as presented and bit-by-bit start building the elements of the Java code until a fully functional game was completed as per the design brief.

As an object-oriented language, it is clear that Java offers a good framework for the inexperienced software programmer to convert real-world ideas and notions into something that is of practical use for a computer. With this in mind, my first approach to the problem was to write an outline of what was required at a highly conceptual level.

# Inputs, Processes and Outputs

The very starting point of my design phase was to write down some thoughts on what happens in a game of Rock Paper Scissors in the real world. In summary, two players choose a shape from a choice of rock, paper or scissors. They then show each-other the shapes at exactly the same time. There are various permutations that determine the winner or loser. For example, the rock can smash the scissors and the paper can wrap the rock. The two players might decide to play using simple rules such as who wins the most after a certain number of rounds, or who loses their lives first. They could record this on a piece of paper.

At an early stage, my reasoning was that the most efficient way to represent the choice of shapes was through a simple integer (i.e. 1 for rock, 2 for paper and 3 for scissors). If required to be converted to text output, then the number could always be easily converted into a text string for the end user. This integer would be used to calculate the various permutations of the actual game itself. For example, a section of pseudo-code for working out how the rock breaks the scissors was written as follows:

**if** (userHandShape is Rock and computerHandShape is Scissors) {

Print out: “Rock smashes Scissors”;  
userScore increases by 1;  
} **else if** (userHandShape is Scissors and computerHandShape is Rock){  
 Print out: “Rock smashes Scissors”;  
 computerScore increases by 1;}

This pattern of pseudo-code was repeated for all of the other permutations of winning and draws in Rock Paper Scissors.

Having the genesis of the potential logic, I set to work on the flow chart for the program. This also involved determining how to implement the user starting with 3 lives, allowing the user to choose how many rounds per game and also the two-dimensional array used to store the scores from each game.

# Flow Chart Diagram

The main aim of a flow chart diagram is to think about how a Java program will present data, process data and make decisions by following the direction of flow from the start to the termination. The flow chart for my version of the Rock Paper Scissors game went through various versions and was helpful in determining how my initial concepts could be converted into potential classes, and how those classes would interact with each other (flow chart is presented in Figure 1).

During the process of writing the flow chart it seemed interesting to work in an incentive to win the Rock Paper Scissors overall match. Achieving 7 lives would mean the user wins and losing all lives would result in a Game Over. Following either a win or defeat, the score board that had collected results from each game would be shown at the end of the match.

It was also evident that a way of validating the user’s input for choosing their hand shape would need to be written into the flow process. The invalid number flow involves checking the number is an integer that equals 1(rock), 2(paper) or 3(scissors). If this condition isn’t fulfilled, the program must come back to the user until they enter the number as required.

When the user input is valid, this will then be combined with the random number generated by the computer, and used in the game process algorithm by using “if…else” statements.

Another aspect to be implemented was how to provide the user with the choice of how many rounds should occur per game. The logical solution was to loop the whole games process until either the user’s lives reached 7 and the won or all of their lives were lost and they lose. Each game in the loop would consist of the number of rounds as chosen by the user at the start of the game.

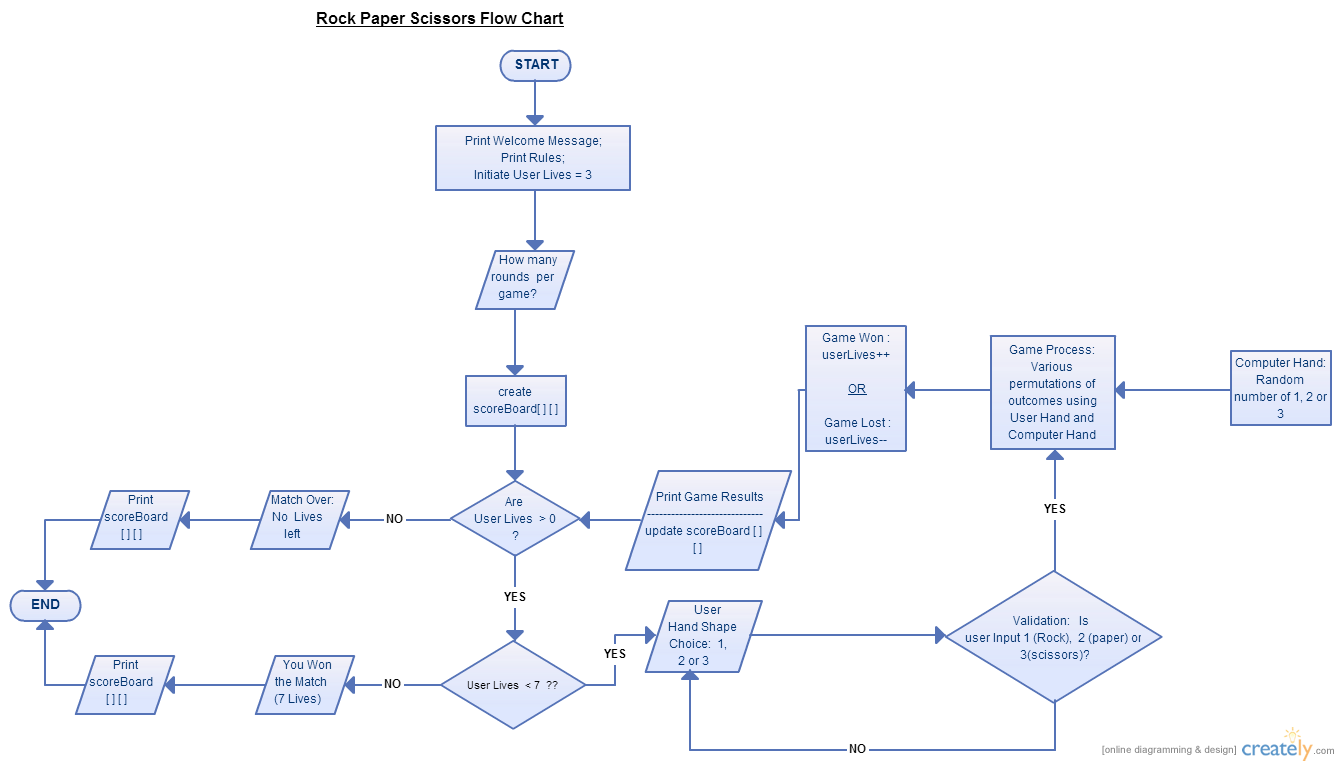


Figure 1: Flow Chart

# Class Diagram

Knowing the valuable information provided by the flow chart, the next step was to determine which classes would go towards ensuring the Rock Paper Scissors game worked in terms of the Java programming language. Initially, the class diagram was quite simple and showed two classes. However, it soon developed when all the various requirements of the program where worked into the code.

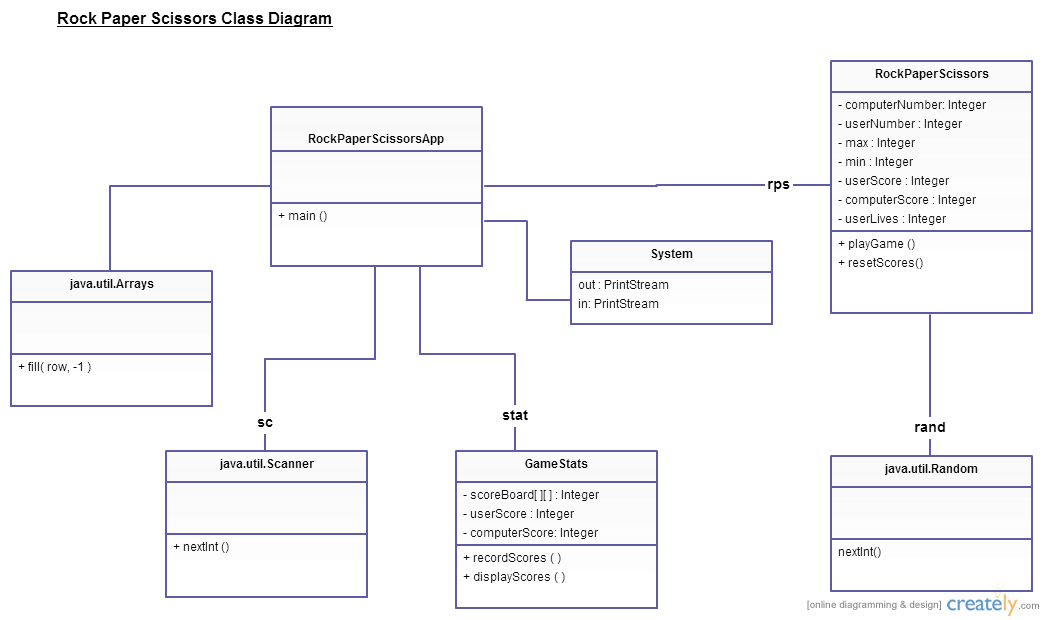


Figure 2: Class Diagram

The class diagram involves the following programmer-defined classes:

* **RockPaperScissorsApp:** Contains the main method. Takes inputs from the user, outputs information and results from processes. Also initialises the score board array with default values.
* **RockPaperScissors:** This is an instantiable class that contains the process of the program. It has two methods, one that contains the game permutations and another that resets the scores for the next game. It also has a constructor that initialises the user lives at 3.
* **GameStats:** This instantiable class records scores and displays them using a two-dimensional array scoreboard. It could potentially be re-used for any other game involving a user and a computer because this class is not specific to the Rock Paper Scissors game.

In addition to the classes that I’ve authored, these Java Standard Classes were also utilised:

* **System:** PrintStream used for output data. Input used in conjunction with Scanner class.
* **Scanner:** Used to take an input from the user. NextInt method utilised for this program.
* **util.Random:** Provided the random number for the computer hand shape.
* **util.Arrays:** Method called fill was used to initialise all fields in the array with -1’s.

The class diagram was helpful in providing an overview of the whole program, and keeping track of what classes, variables, methods and object names were used.

# Coding, Debugging and Testing

It was reasonable to commence writing the code by focusing on the actual game process element of the flow chart. The rest of the elements of the game and additional requirements were then built onto the existing code. Certain bugs were discovered during the testing process and for the most part involved ensuring that syntax errors were corrected, setter and getter methods were correctly utilised and other small items such as getting the array data from instantiable class GameStats over to the App. Again, this involved correctly sequencing the getter and setter method invokations in the main method and also ensuring the method for recording scores in GameStats correctly recorded the scores after each round of games.

# Running the Application

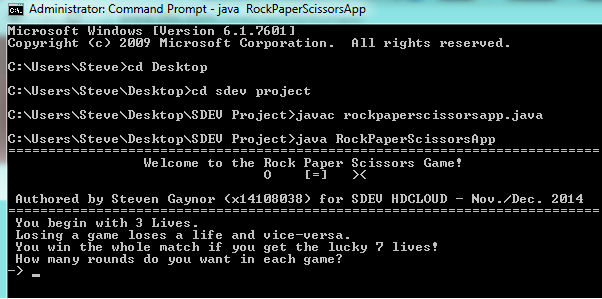
The following screenshots demonstrate the finished program running using the command line prompt. When the RockPaperScissorsApp.java file is first executed, the following welcome message appears, including the rules of the game and the user choice of the amount of rounds:

Figure 3: Welcome Message

In this example, 6 rounds per game was chosen. The actual game process itself then begins, showing what both players hand shapes are and who won:

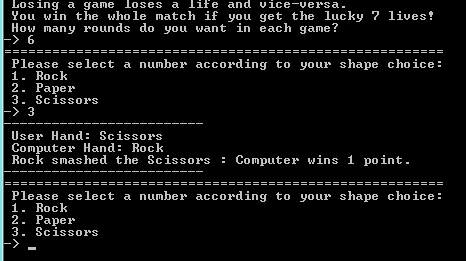


Figure 4: Playing Rock Paper Scissors

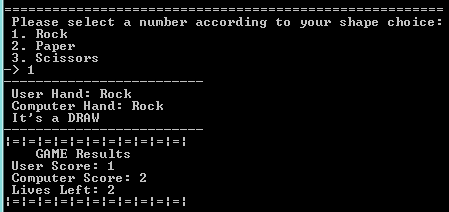
The game continues until all 6 rounds are complete. Then, the game results summary is presented:

Figure 5: Game Results

These scores are stored in the double-array following each game. If the user enters an invalid choice when asked for their next hand shape, the following message (Fig. 6) will display until user enters the correct number:

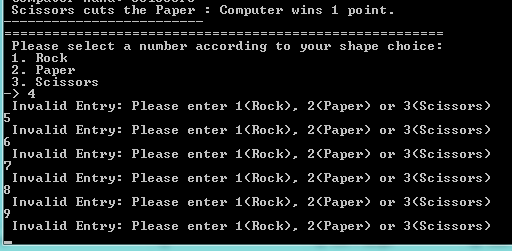


Figure 6: Invalid Number Loop

When the user either wins 7 lives or loses all of their lives, the match terminates and the final scoreboard is displayed, showing the results from each game. In this example, the user achieves 7 lives and wins the overall match. The scores from all previous games are then called back from the score board array and presented to the user:

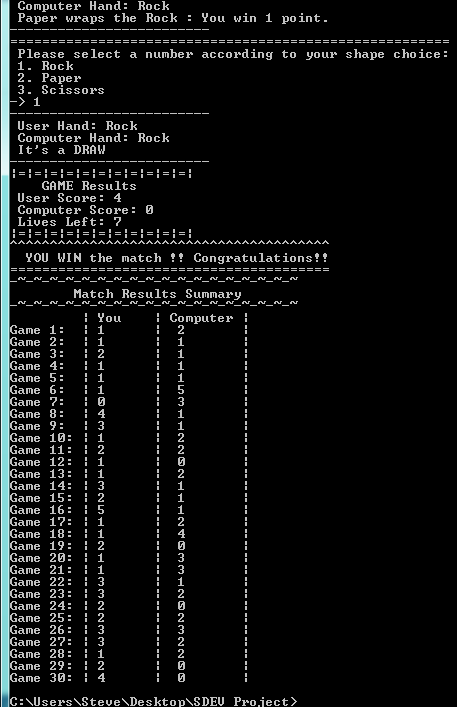


Figure 7: Final Scoreboard