Assignment-1: Choosing the right software process to build good-quality software, and ensuring quality using test automation

SOFE 3980U

Ashad Ahmed

100745913

Project Specifications:

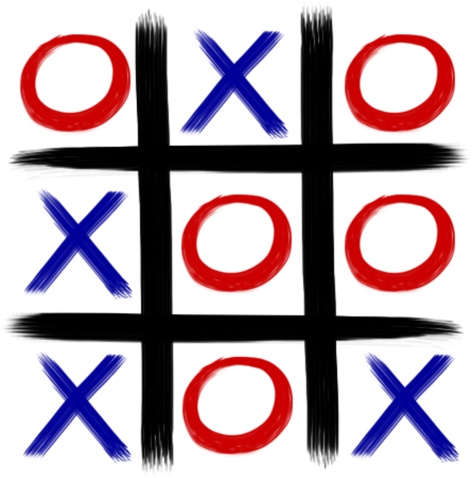
The game created for this project was Tic Tac Toe. Although this code was created by me, the website: GeeksforGeeks was used as a guide to learn the process [1]. It is sourced in the references section. The programming language chosen for the game was **Java** and the IDE was **Eclipse.exe**. The reason for this was that Java has the **Junit 5** automated testing framework built for it, as well as the fact that with Eclipse, Junit testing is inbuilt, and no further installation is required to use this testing framework.

The software engineering process used for development of this program was the **Waterfall process**. This was because the requirements for this project, and the technology used to create it were not prone to sudden changes and feedback that requires maintenance after development has completed. The reason this was chosen over the Agile process is because that process is useful for short phases, requires minimal formal documentation and requires the software to constantly be open to changes and modifications even after deployment. For this project, these elements were not required or necessary. It was chosen over the Incremental process was because the incremental process is more effective when extremely long-term planning with a large team is involved, for a very large-scale project that needs to be further divided into smaller modules to be tackled in their phases individually. Tic Tac Toe is a simple game and does not need to be broken down like this.

Game Functionalities and development Using Waterfall Process:

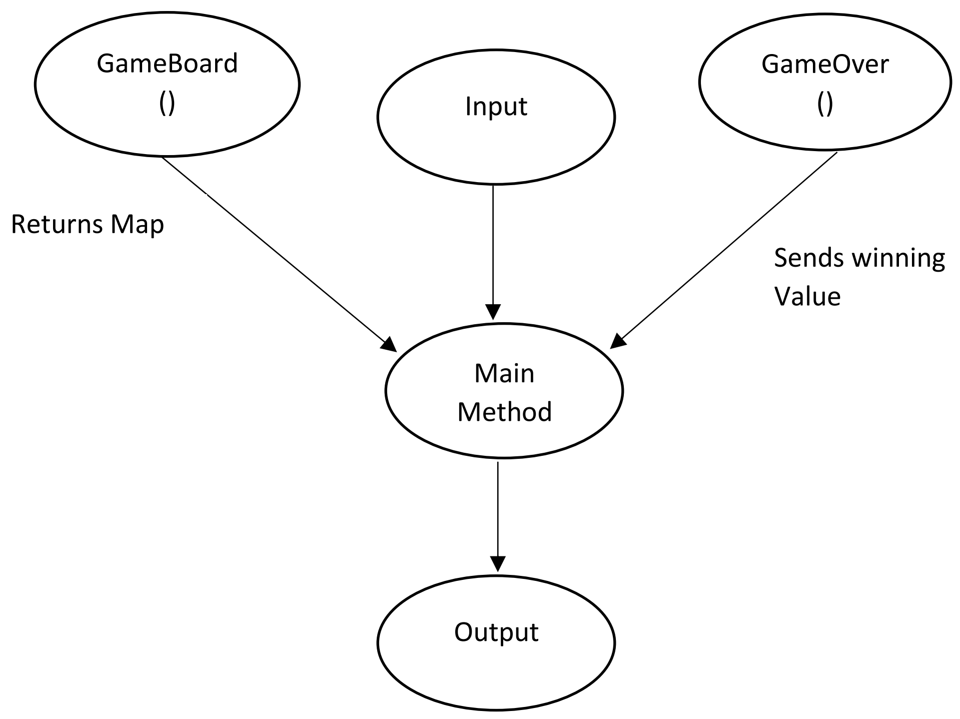
Requirements analysis and specification phase:

The requirements of this project were straightforward and simple. A simple gaming software in any programming language was to be designed and automated testing was to be manually written and performed on that software afterwards. The game selected for this specific project, was Tic Tac Toe. A simple game that involves two players trying to create a row of either three X's or O's across a 3 by 3 map. This row could be horizontal, vertical, or diagonal. If no rows are formed by the time the map is filled, the game is a draw.



Design Phase:

The basic structure of the program in Java was through the usage of three methods: A DisplayBoard method that simply stores the basic layout of the 3 by 3 map shown in the image above, a GameOver method that checks all array slot entry combinations that form a winning row (or none in case of a draw) and a Main method that prompts user input, calling upon the previous two functions to print the board, each turn, and the winner once the game has ended. The structure is simplified in the diagram below:



Implementation and Unit Testing Phase:

Coding these classes went smoothly. Originally the plan was only to implement a winner and no draw, however this was added during the implementation phase, as the game needed to be stopped once all elements of the array were filled up with no rows formed. At the very end, a conditional loop was also implemented in the Main method with a question to prompt the user(s) to restart the game by answering “Y” to the question in case they wanted to play again without rerunning the entire program.

Integration and System Testing Phase:

After completion, the usage of automated tests was implemented, to improve the functionality of the game and ensure every method it was divided into was running flawlessly. Seven tests were written, all of which assessed all the row formation orientations using both X and O to see if they correctly returned the expected value from the GameOver function, whether it was a winner amongst the two (and which one) or a draw. All tests passed.

Operation and Maintenance Phase:

As this project is not being sold to anyone for regular use, rather, it is a one-time submission that does not need to be run constantly after being created and evaluated, there was no maintenance phase in this project. The product was created using the above phases as a guide and after testing, it was finished.

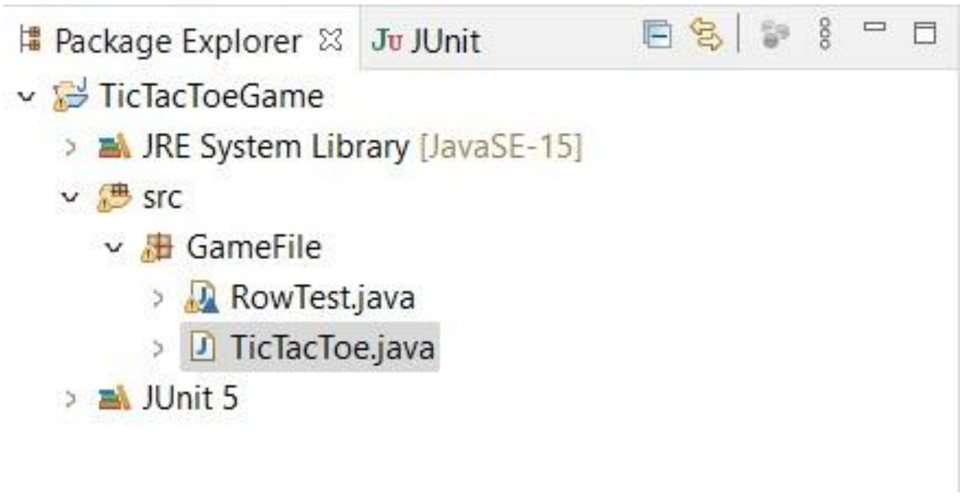
Test Automation Challenges:

While writing challenges, the most time-consuming portion was the planning portion as I was struggling with ideas of what kind of tests to perform on this game, since it does not include many methods that return numeric values based on user input. I eventually decided to use my GameOver method since that was the only method returning variables to the main class. All my tests simply check row combinations and whether the GameOver method returns the correct expected String (Whether X, O, or D) based on the combinations that have been made. Another challenge I encountered during test writing, was that my GameBoard array was initially in the main method as a private static array, so the tests were unable to access it. To fix this, I declared it at the beginning of my game class and changed it to a public array, which caused the tests to start working as expected.

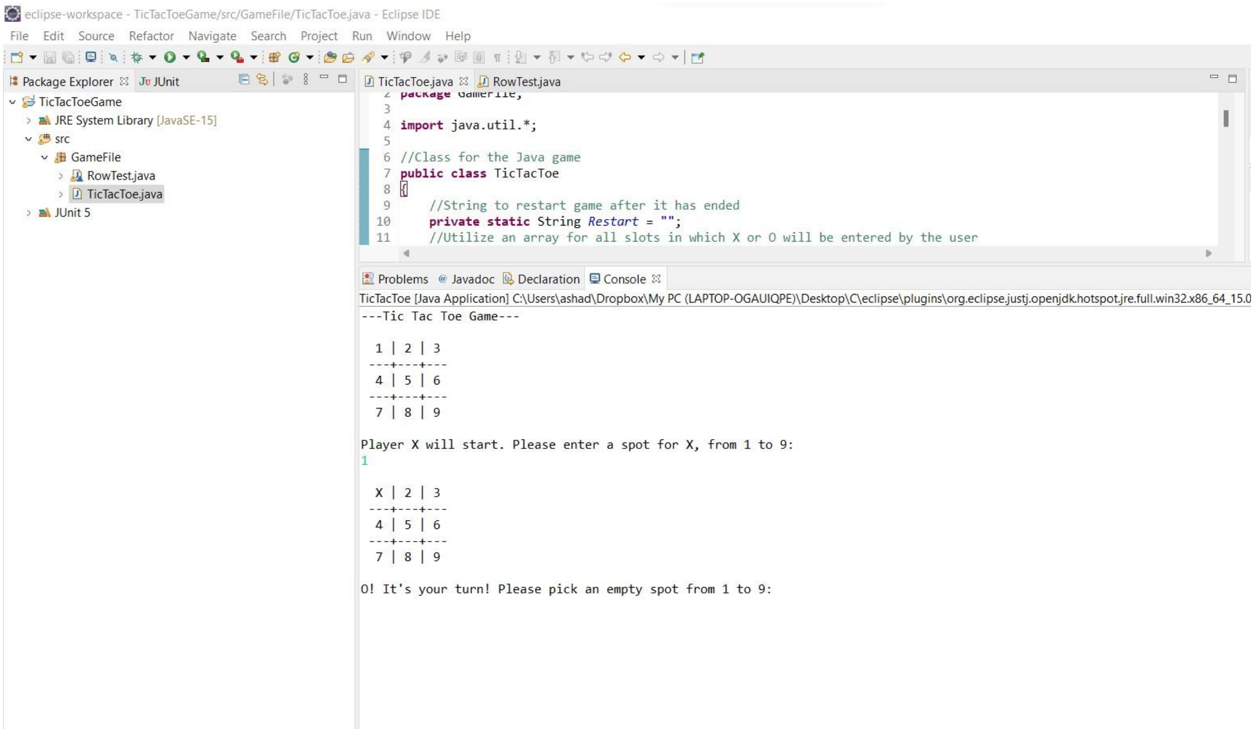
How to use:

TicTacToe.java:

The process for the usage of this game is very simple. There are only two classes, one that runs the game and one for all the tests performed on it. Open this project in Eclipse.exe and click on the TicTacToe.java file in the GameFile package as shown below:



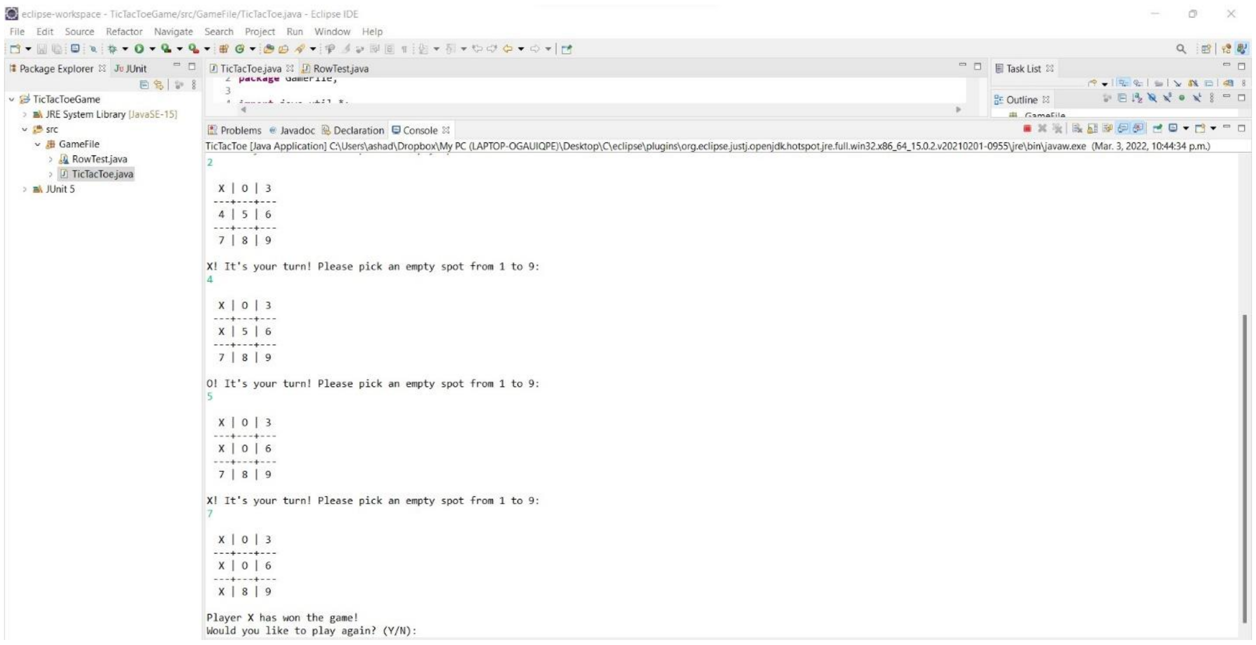
Next, run the file. It will prompt the player who chose X to start with the first turn. The welcome message shown below will be displayed with a display of the three-by-three map and all numbered slots. After entering a number that spot will be replaced with X on the map, which will be displayed again, and the next turn will be prompted. It carries on like this until one player makes a row.


eclipse-workspace - TicTacToeGame/src/GameFile/TicTacToe.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
TicTacToeGame
JRE System Library [JavaSE-15]
src
GameFile
RowTest.java
TicTacToe.java
JUnit 5
TicTacToe.java
RowTest.java
Problems
Javadoc
Declaration
Console

```java
package GameFile;

import java.util.*;

//Class for the Java game
public class TicTacToe
{
    //String to restart game after it has ended
    private static String Restart = "";
    //Utilize an array for all slots in which X or O will be entered by the user
```

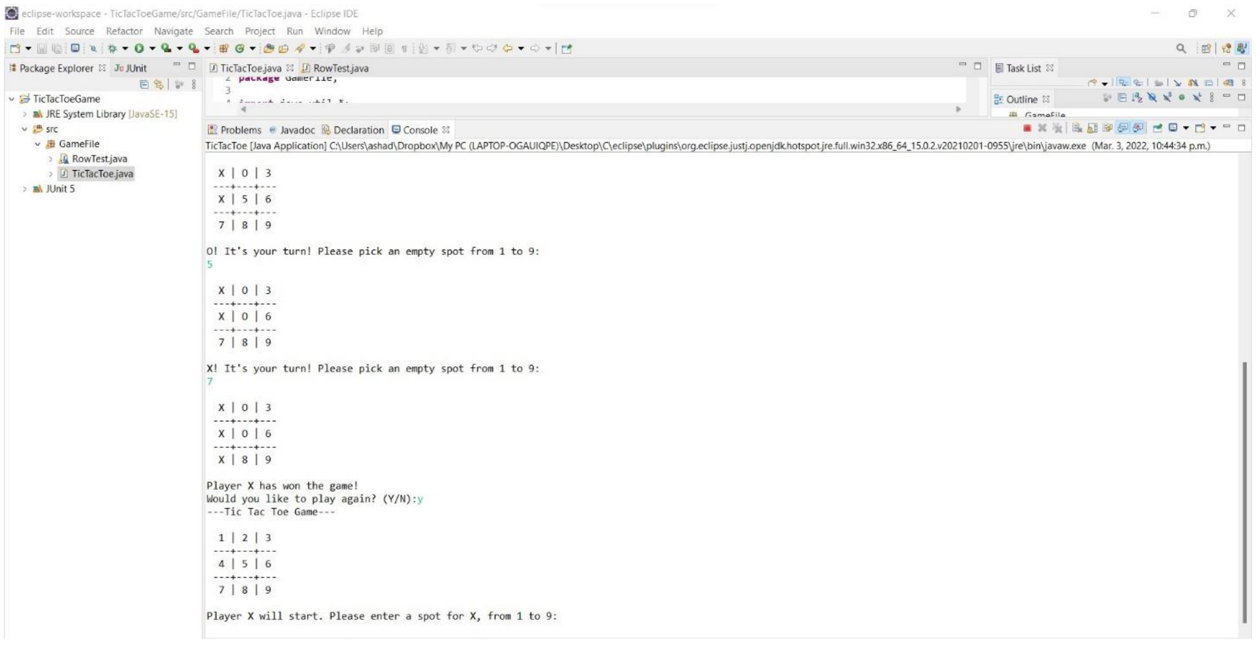
```
TicTacToe [Java Application] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\C\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0
---Tic Tac Toe Game---

 1 | 2 | 3
---+---+---
 4 | 5 | 6
---+---+---
 7 | 8 | 9

Player X will start. Please enter a spot for X, from 1 to 9:
1

 X | 2 | 3
---+---+---
 4 | 5 | 6
---+---+---
 7 | 8 | 9

O! It's your turn! Please pick an empty spot from 1 to 9:
```

After a row is made, a message informing the winner they won will be displayed and the program will ask if the user(s) want to start a new game.



```
TicTacToe [Java Application] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\C\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (Mar. 3, 2022, 10:44:34 p.m.)
2

 X | 0 | 3
---+---+---
 4 | 5 | 6
---+---+---
 7 | 8 | 9

X! It's your turn! Please pick an empty spot from 1 to 9:
4

 X | 0 | 3
---+---+---
 X | 5 | 6
---+---+---
 7 | 8 | 9

O! It's your turn! Please pick an empty spot from 1 to 9:
5

 X | 0 | 3
---+---+---
 X | 0 | 6
---+---+---
 7 | 8 | 9

X! It's your turn! Please pick an empty spot from 1 to 9:
7

 X | 0 | 3
---+---+---
 X | 0 | 6
---+---+---
 X | 8 | 9

Player X has won the game!
Would you like to play again? (Y/N):
```

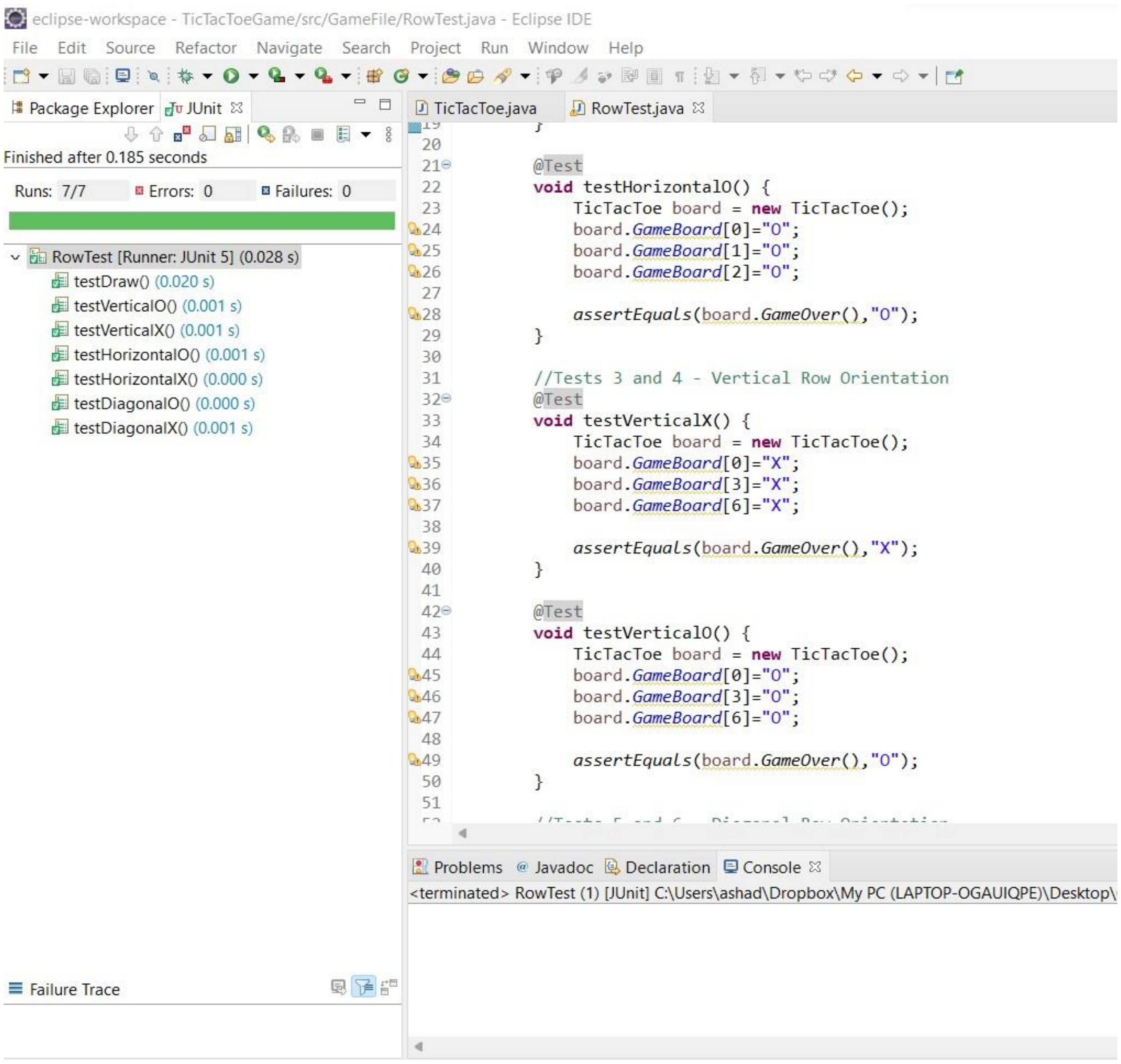
If the user presses the “y” key the game will start with a welcome message all over again. If any other key is pressed, the program stops running.



```
TicTacToe [Java Application] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\C\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (Mar. 3, 2022, 10:44:34 p.m.)

 X | 0 | 3
---+---+---
 X | 5 | 6
---+---+---
 7 | 8 | 9

O! It's your turn! Please pick an empty spot from 1 to 9:
5

 X | 0 | 3
---+---+---
 X | 0 | 6
---+---+---
 7 | 8 | 9

X! It's your turn! Please pick an empty spot from 1 to 9:
7

 X | 0 | 3
---+---+---
 X | 0 | 6
---+---+---
 X | 8 | 9

Player X has won the game!
Would you like to play again? (Y/N):y
---Tic Tac Toe Game---

 1 | 2 | 3
---+---+---
 4 | 5 | 6
---+---+---
 7 | 8 | 9

Player X will start. Please enter a spot for X, from 1 to 9:
```

RowTest.java:

After testing the game class, open RowTest.java. Simply running this class just like the last one, will display in the left corner, all the tests in the class and whether they work properly. The Errors field will have a 0 beside it and the bar that indicates test result on all the methods, will be green.


eclipse-workspace - TicTacToeGame/src/GameFile/RowTest.java - Eclipse IDE

```
Finished after 0.185 seconds

Runs: 7/7      Errors: 0      Failures: 0

RowTest [Runner: JUnit 5] (0.028 s)
    testDraw() (0.020 s)
    testVerticalO() (0.001 s)
    testVerticalX() (0.001 s)
    testHorizontalO() (0.001 s)
    testHorizontalX() (0.000 s)
    testDiagonalO() (0.000 s)
    testDiagonalX() (0.001 s)
```

```java
    }

    @Test
    void testHorizontalO() {
        TicTacToe board = new TicTacToe();
        board.GameBoard[0]="O";
        board.GameBoard[1]="O";
        board.GameBoard[2]="O";

        assertEquals(board.GameOver(),"O");
    }

    //Tests 3 and 4 - Vertical Row Orientation
    @Test
    void testVerticalX() {
        TicTacToe board = new TicTacToe();
        board.GameBoard[0]="X";
        board.GameBoard[3]="X";
        board.GameBoard[6]="X";

        assertEquals(board.GameOver(),"X");
    }

    @Test
    void testVerticalO() {
        TicTacToe board = new TicTacToe();
        board.GameBoard[0]="O";
        board.GameBoard[3]="O";
        board.GameBoard[6]="O";

        assertEquals(board.GameOver(),"O");
    }
```

```
<terminated> RowTest (1) [JUnit] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\
```

References [IEEE Format]:

[1] SAKSHIKULSHRESHTHA, “Tic-tac-toe game in Java,” *GeeksforGeeks*, 13-Oct-2020. [Online]. Available: https://www.geeksforgeeks.org/tic-tac-toe-game-in-java/. [Accessed: 03-Mar-2022].

[2] S. Somani, “Tic-tac-toe game in Java,” *GeeksforGeeks*, 03-Jan-2022. [Online]. Available: https://www.geeksforgeeks.org/tic-tac-toe-game-in-java/. [Accessed: 03-Mar-2022].

Assignment 2: Static and Dynamic Analysis

SOFE 3980U

Ashad Ahmed

100745913

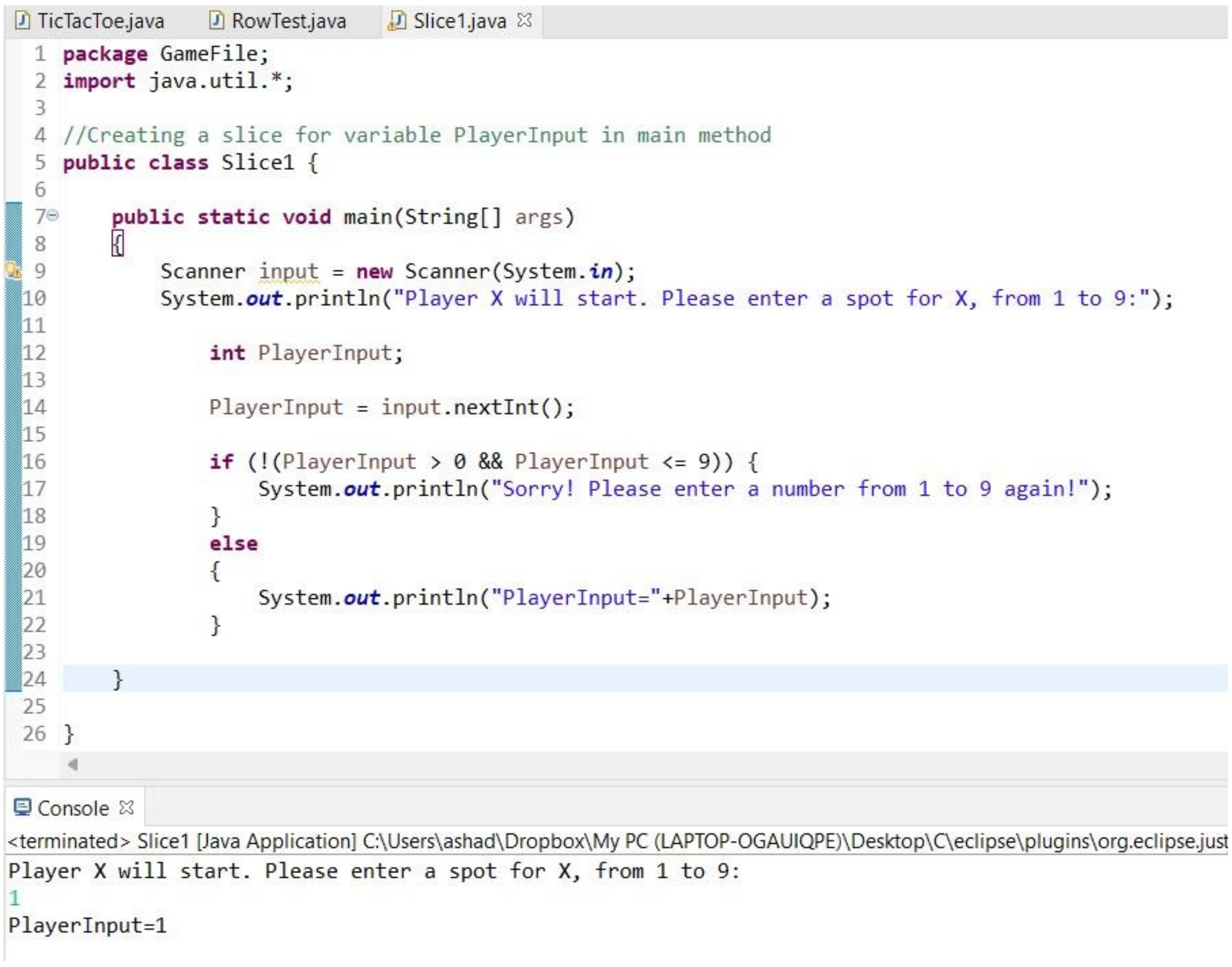
Introduction

Program Slicing is a concept in software testing that involves selecting small segments or lines of program statements and performing tests on these statements to see if they affect the variables within them as expected. This assignment instructs us to perform slicing on variables from the Tic Tac Toe game created within the last assignment.

Tic Tac Toe Program Slicing

Manual slicing was used in this assignment. A slice for each variable was created in a separate class to see if the statements work accurately.

This meant one for *PlayerInput* variable:

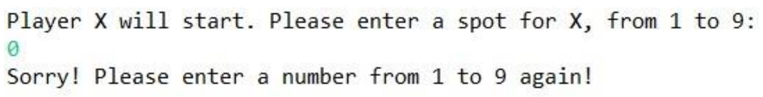


TicTacToe.java RowTest.java Slice1.java

```java
package GameFile;
import java.util.*;

//Creating a slice for variable PlayerInput in main method
public class Slice1 {

    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        System.out.println("Player X will start. Please enter a spot for X, from 1 to 9:");

            int PlayerInput;

            PlayerInput = input.nextInt();

            if (!(PlayerInput > 0 && PlayerInput <= 9)) {
                System.out.println("Sorry! Please enter a number from 1 to 9 again!");
            }
            else
            {
                System.out.println("PlayerInput="+PlayerInput);
            }

    }

}
```

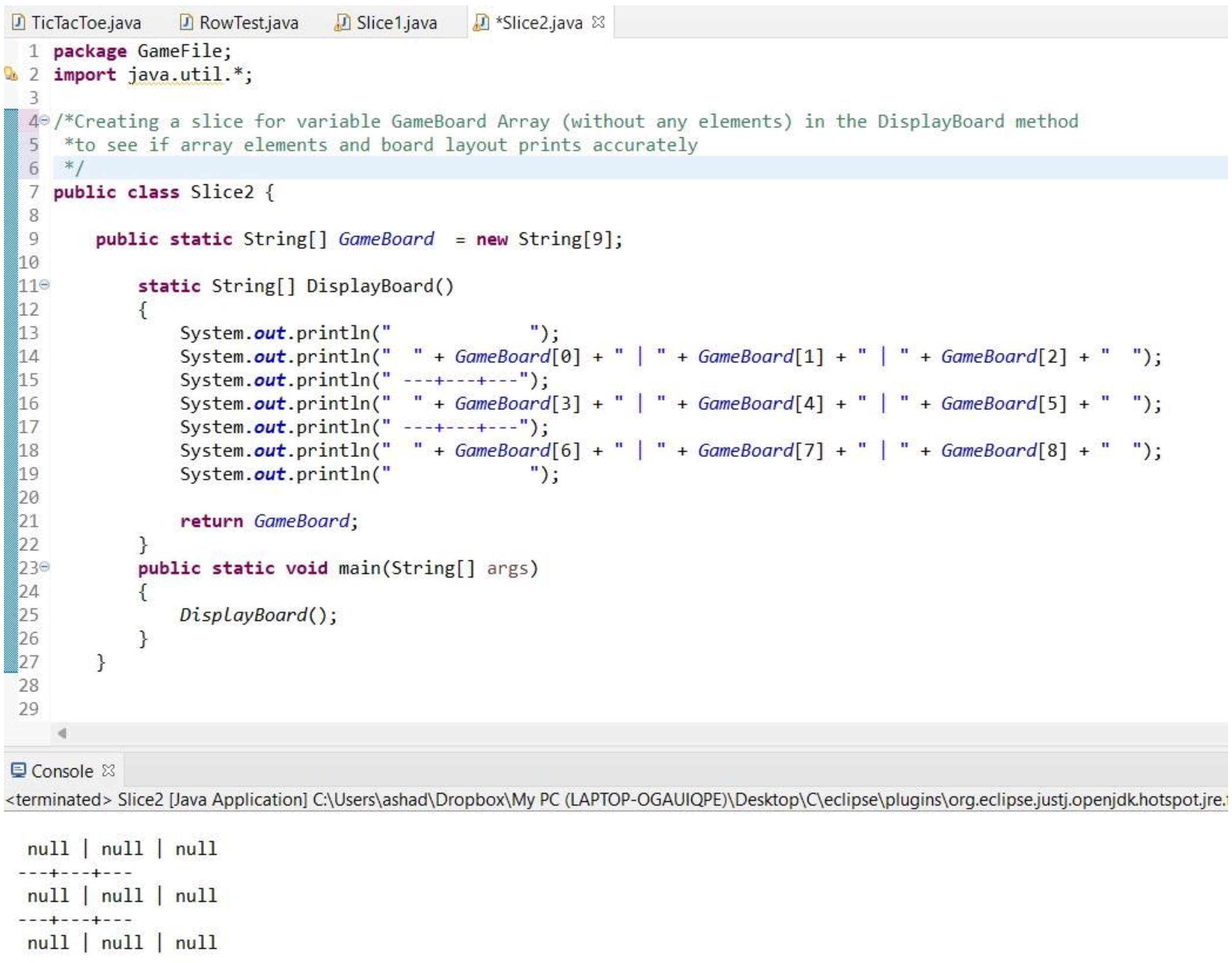
Console

```
<terminated> Slice1 [Java Application] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\C\eclipse\plugins\org.eclipse.just
Player X will start. Please enter a spot for X, from 1 to 9:
1
PlayerInput=1
```



```
Player X will start. Please enter a spot for X, from 1 to 9:
0
Sorry! Please enter a number from 1 to 9 again!
```

Another slice was created for the *GameBoard* array and the *DisplayBoard* method to see if the board layout and array element positioning is correct:

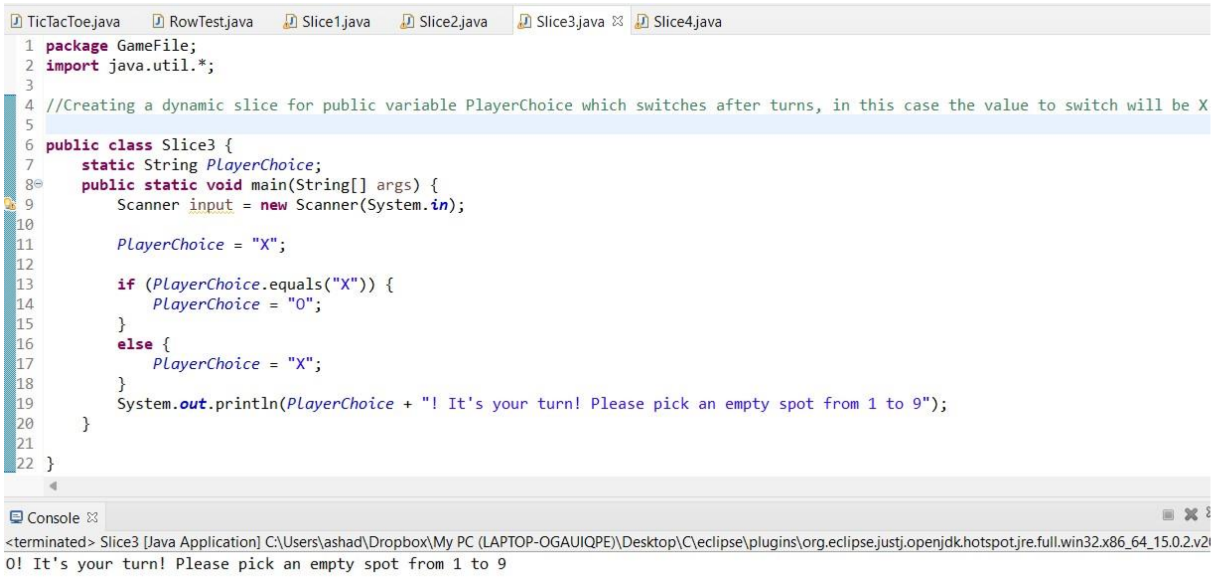


```java
package GameFile;
import java.util.*;

/*Creating a slice for variable GameBoard Array (without any elements) in the DisplayBoard method
 *to see if array elements and board layout prints accurately
 */
public class Slice2 {

    public static String[] GameBoard  = new String[9];

        static String[] DisplayBoard()
        {
            System.out.println("             ");
            System.out.println("  " + GameBoard[0] + " | " + GameBoard[1] + " | " + GameBoard[2] + "  ");
            System.out.println(" ---+---+---");
            System.out.println("  " + GameBoard[3] + " | " + GameBoard[4] + " | " + GameBoard[5] + "  ");
            System.out.println(" ---+---+---");
            System.out.println("  " + GameBoard[6] + " | " + GameBoard[7] + " | " + GameBoard[8] + "  ");
            System.out.println("             ");

            return GameBoard;
        }
        public static void main(String[] args)
        {
            DisplayBoard();
        }
    }


```

```
<terminated> Slice2 [Java Application] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\C\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.

  null | null | null
 ---+---+---
  null | null | null
 ---+---+---
  null | null | null
```

Another slice for *PlayerChoice* variable:

Note that for this slice, the purpose is not to take, input, it is only to display the correct choice (X or O) in the message that asks for input in the console.

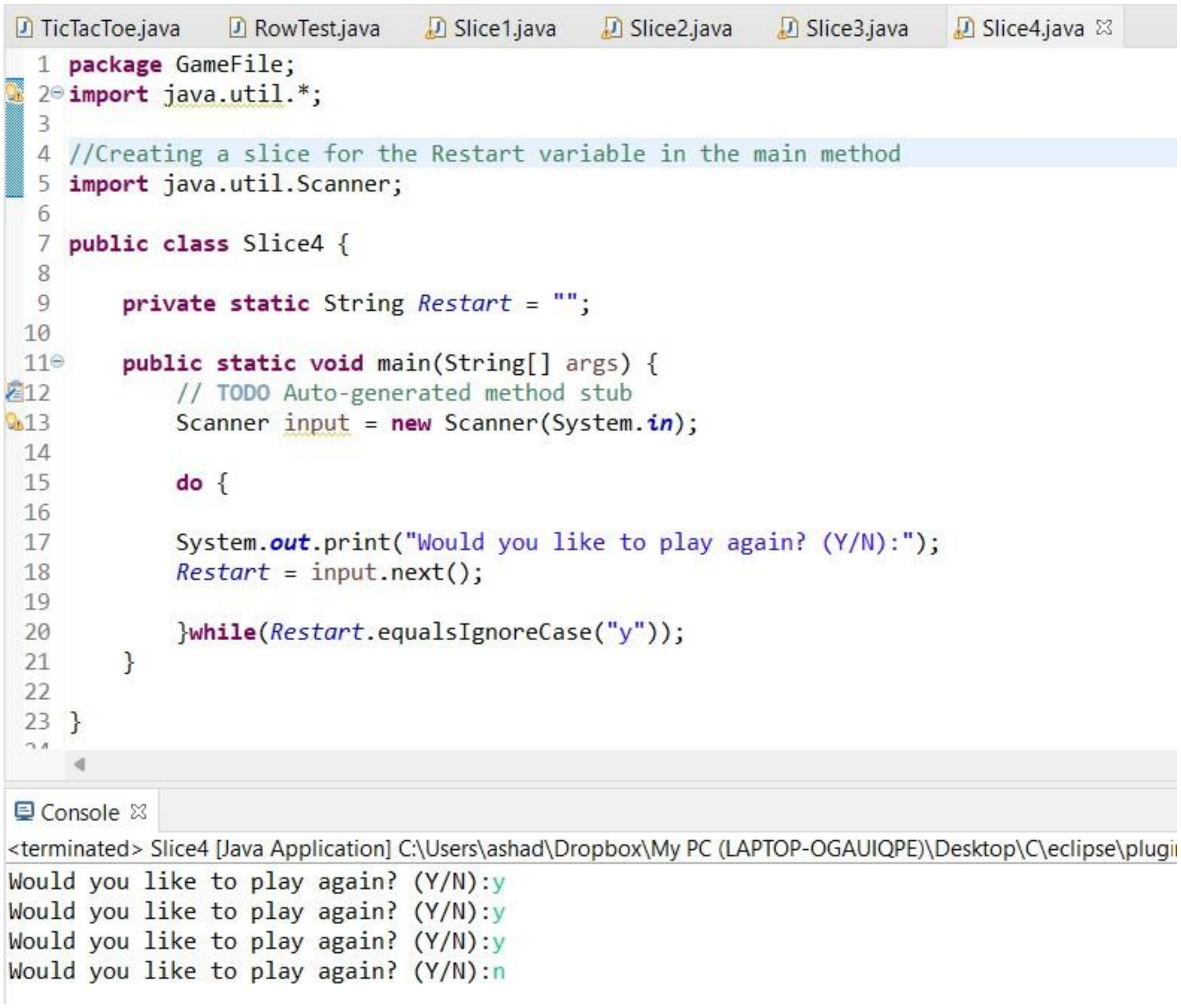

TicTacToe.java RowTest.java Slice1.java Slice2.java Slice3.java Slice4.java

```java
package GameFile;
import java.util.*;

//Creating a dynamic slice for public variable PlayerChoice which switches after turns, in this case the value to switch will be X

public class Slice3 {
    static String PlayerChoice;
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        PlayerChoice = "X";

        if (PlayerChoice.equals("X")) {
            PlayerChoice = "O";
        }
        else {
            PlayerChoice = "X";
        }
        System.out.println(PlayerChoice + "! It's your turn! Please pick an empty spot from 1 to 9");
    }

}
```

Console

```
<terminated> Slice3 [Java Application] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\C\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v2
O! It's your turn! Please pick an empty spot from 1 to 9
```

Another slice for Restart variable:

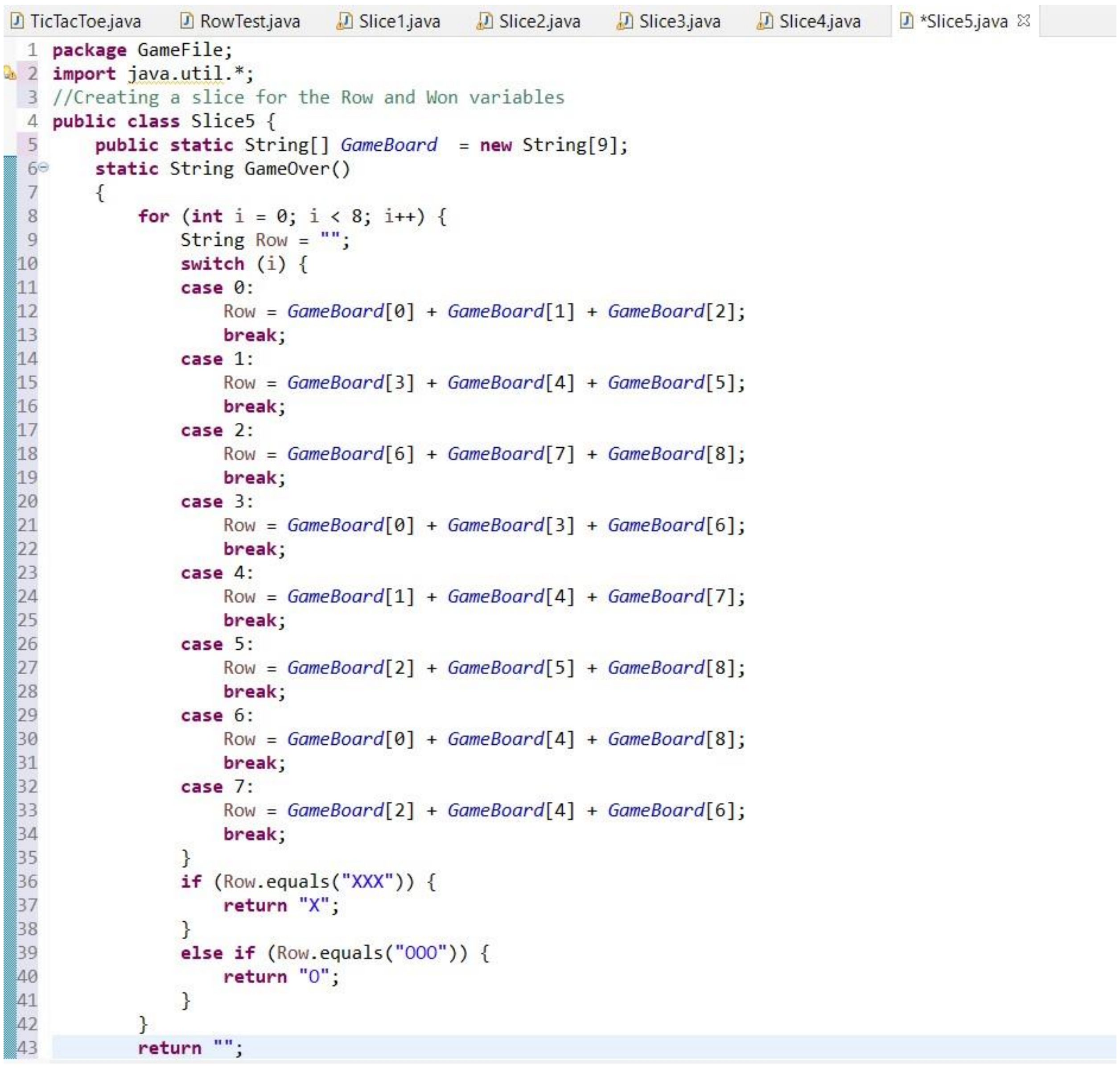

TicTacToe.java RowTest.java Slice1.java Slice2.java Slice3.java Slice4.java

```java
package GameFile;
import java.util.*;

//Creating a slice for the Restart variable in the main method
import java.util.Scanner;

public class Slice4 {

    private static String Restart = "";

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner input = new Scanner(System.in);

        do {

        System.out.print("Would you like to play again? (Y/N):");
        Restart = input.next();

        }while(Restart.equalsIgnoreCase("y"));
    }

}
```

Console

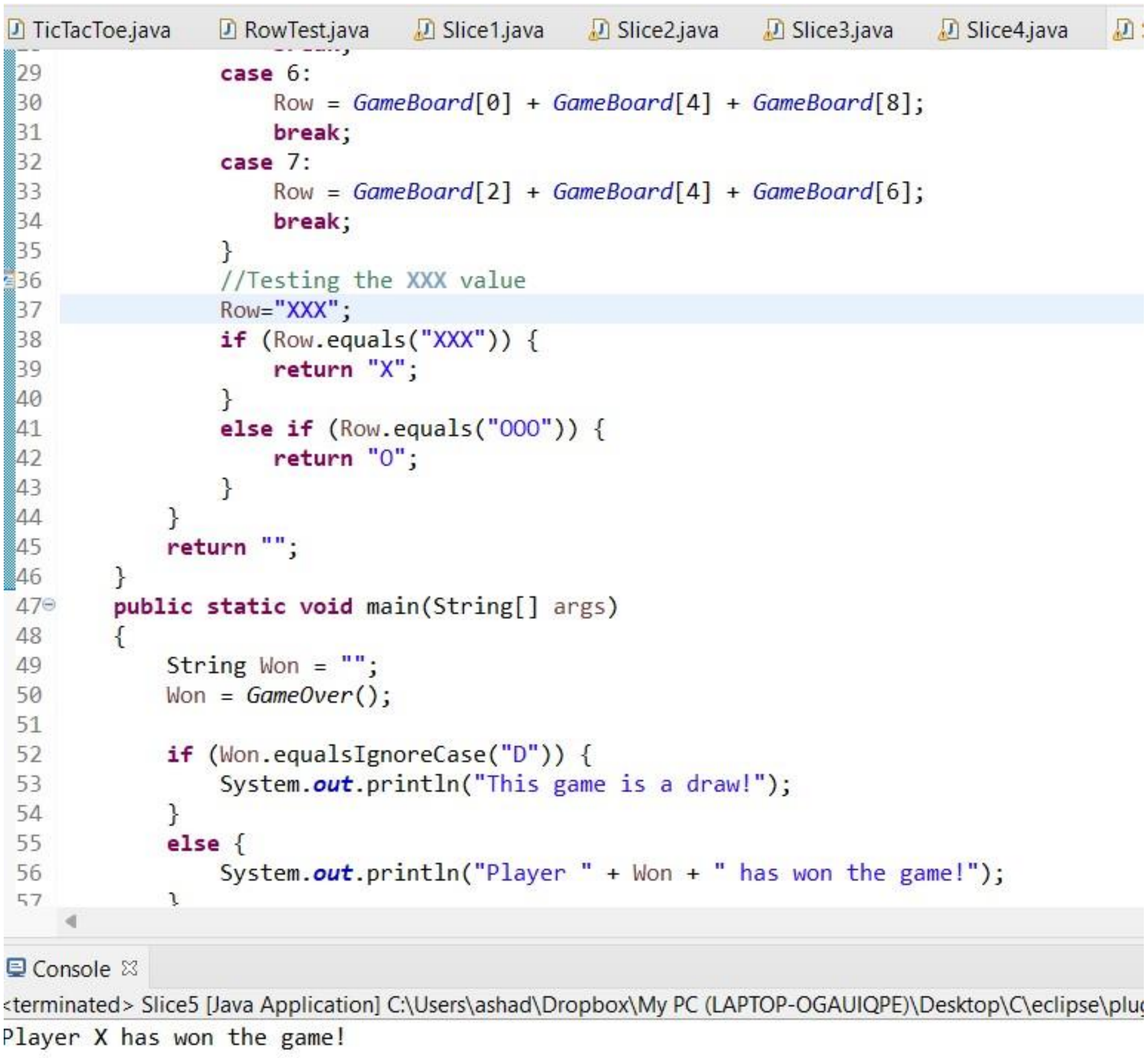
```
<terminated> Slice4 [Java Application] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\C\eclipse\plugi
Would you like to play again? (Y/N):y
Would you like to play again? (Y/N):y
Would you like to play again? (Y/N):y
Would you like to play again? (Y/N):n
```

The final slice for the program tests the Row and Won Variables:



TicTacToe.java RowTest.java Slice1.java Slice2.java Slice3.java Slice4.java *Slice5.java

```java
package GameFile;
import java.util.*;
//Creating a slice for the Row and Won variables
public class Slice5 {
    public static String[] GameBoard  = new String[9];
    static String GameOver()
    {
        for (int i = 0; i < 8; i++) {
            String Row = "";
            switch (i) {
            case 0:
                Row = GameBoard[0] + GameBoard[1] + GameBoard[2];
                break;
            case 1:
                Row = GameBoard[3] + GameBoard[4] + GameBoard[5];
                break;
            case 2:
                Row = GameBoard[6] + GameBoard[7] + GameBoard[8];
                break;
            case 3:
                Row = GameBoard[0] + GameBoard[3] + GameBoard[6];
                break;
            case 4:
                Row = GameBoard[1] + GameBoard[4] + GameBoard[7];
                break;
            case 5:
                Row = GameBoard[2] + GameBoard[5] + GameBoard[8];
                break;
            case 6:
                Row = GameBoard[0] + GameBoard[4] + GameBoard[8];
                break;
            case 7:
                Row = GameBoard[2] + GameBoard[4] + GameBoard[6];
                break;
            }
            if (Row.equals("XXX")) {
                return "X";
            }
            else if (Row.equals("OOO")) {
                return "O";
            }
        }
        return "";
```



```java
            case 6:
                Row = GameBoard[0] + GameBoard[4] + GameBoard[8];
                break;
            case 7:
                Row = GameBoard[2] + GameBoard[4] + GameBoard[6];
                break;
            }
            //Testing the XXX value
            Row="XXX";
            if (Row.equals("XXX")) {
                return "X";
            }
            else if (Row.equals("OOO")) {
                return "O";
            }
        }
        return "";
    }
    public static void main(String[] args)
    {
        String Won = "";
        Won = GameOver();

        if (Won.equalsIgnoreCase("D")) {
            System.out.println("This game is a draw!");
        }
        else {
            System.out.println("Player " + Won + " has won the game!");
```

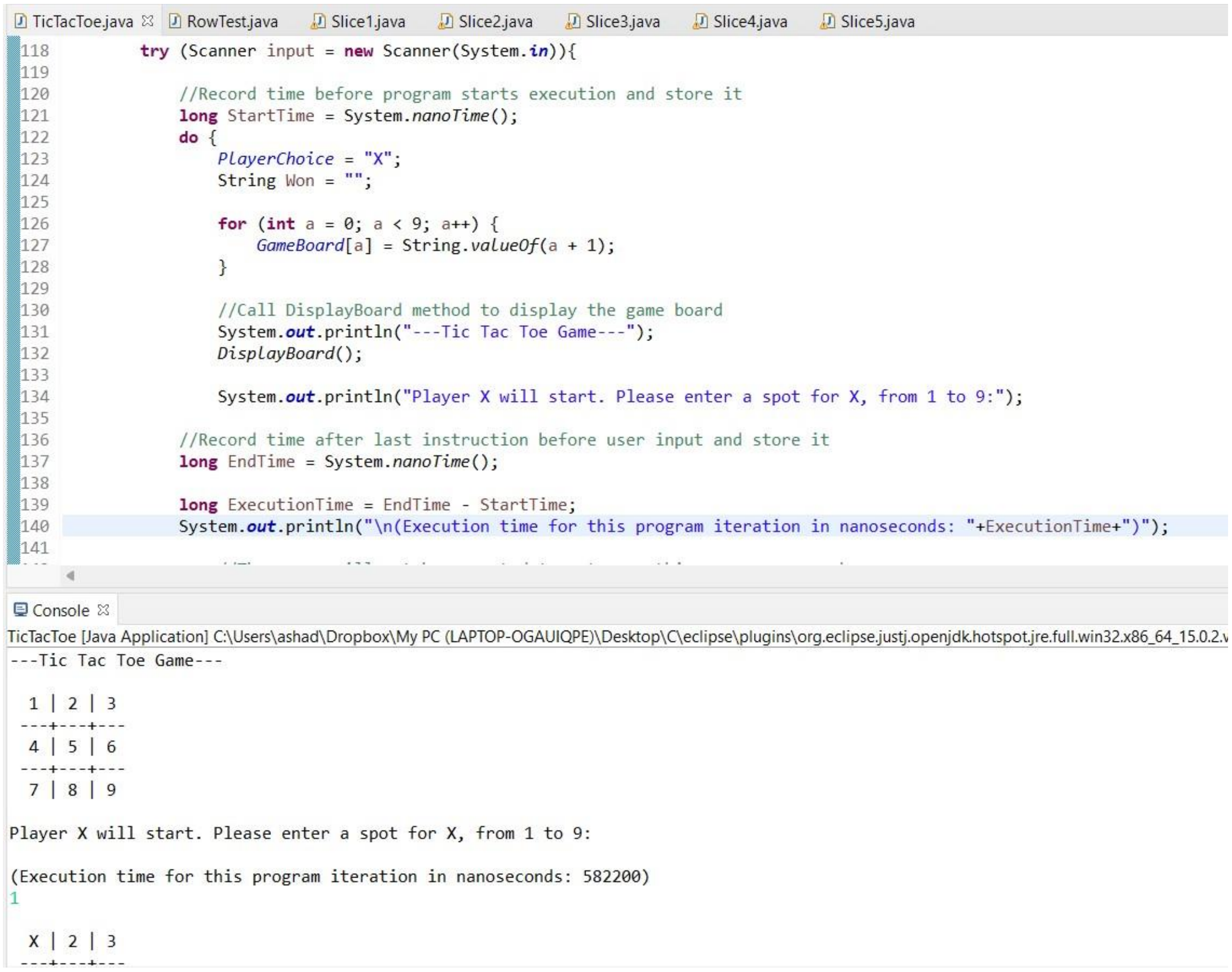
```
<terminated> Slice5 [Java Application] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\C\eclipse\plu
Player X has won the game!
```

Instrumentation

Instrumentation is a dynamic analysis technique in which measurement probes are added to a program to monitor it's execution or monitor it while it is running. In this assignment, a timer was added to the main Tic Tac Toe file and all program slices created in part a, to observe the amount of time it takes to execute the program. This is measured in nanoseconds as the execution window is usually under a second.

TicTacToe.java

For the game class, the main method is the method which fetches all other method functions, asks for input, and then displays output, which is why a variable to record current time value right before the first instruction and the last instruction before user input is required, was implemented in the main method. These start and end times were subtracted from each other to get the duration of execution. Note that in the execution, a side message in brackets indicating the time is displayed, as this is not a message that is needed to be interacted with and it is simply there to notify user of the duration. The rest of the program runs as before. This same strategy is used to calculate execution time of all the slices as well.



TicTacToe.java | RowTest.java | Slice1.java | Slice2.java | Slice3.java | Slice4.java | Slice5.java

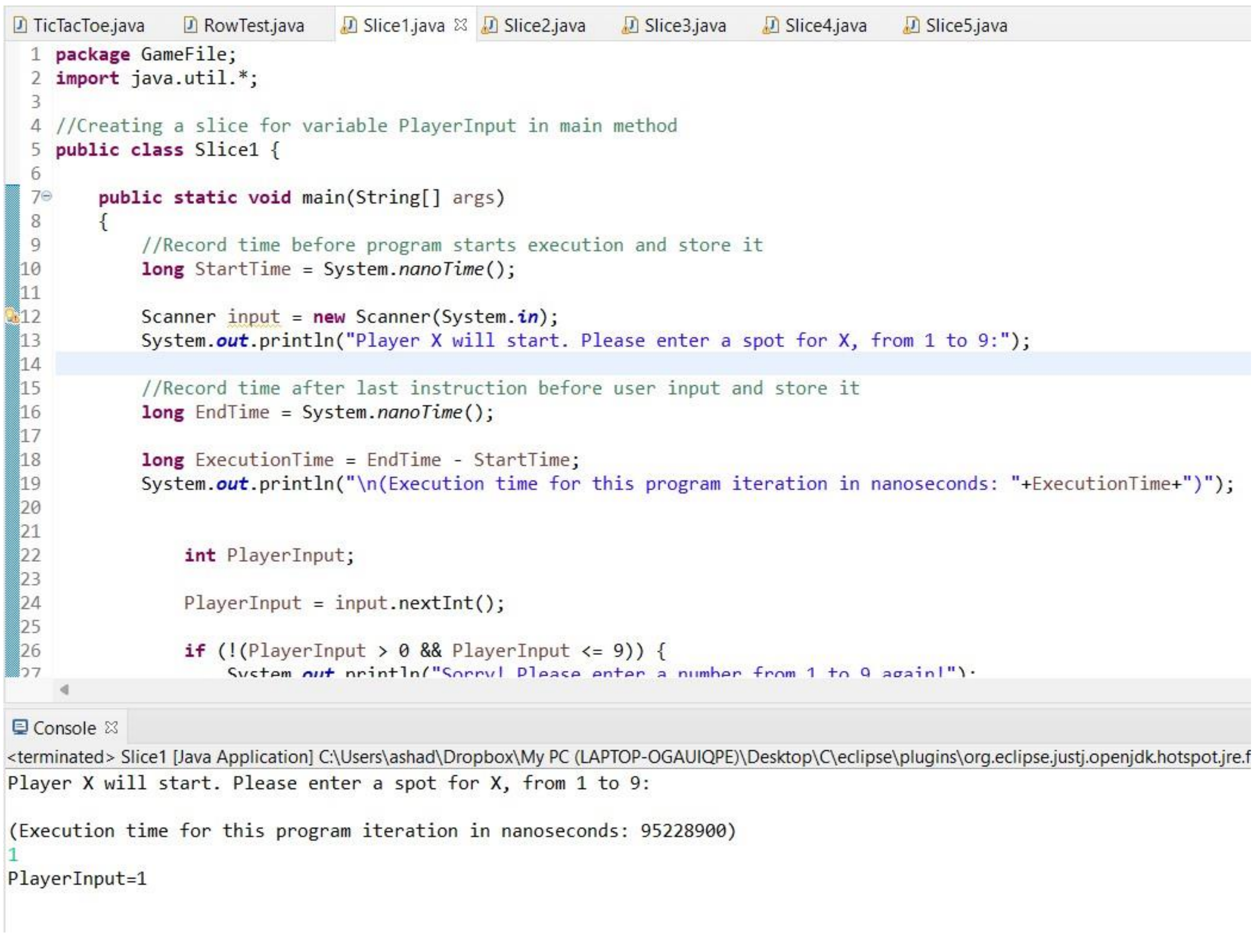
```java
        try (Scanner input = new Scanner(System.in)){

            //Record time before program starts execution and store it
            long StartTime = System.nanoTime();
            do {
                PlayerChoice = "X";
                String Won = "";

                for (int a = 0; a < 9; a++) {
                    GameBoard[a] = String.valueOf(a + 1);
                }

                //Call DisplayBoard method to display the game board
                System.out.println("---Tic Tac Toe Game---");
                DisplayBoard();

                System.out.println("Player X will start. Please enter a spot for X, from 1 to 9:");

            //Record time after last instruction before user input and store it
            long EndTime = System.nanoTime();

            long ExecutionTime = EndTime - StartTime;
            System.out.println("\n(Execution time for this program iteration in nanoseconds: "+ExecutionTime+")");
```

Console

TicTacToe [Java Application] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\C\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v

```
---Tic Tac Toe Game---

 1 | 2 | 3
---+---+---
 4 | 5 | 6
---+---+---
 7 | 8 | 9

Player X will start. Please enter a spot for X, from 1 to 9:

(Execution time for this program iteration in nanoseconds: 582200)
1

 X | 2 | 3
---+---+---
```

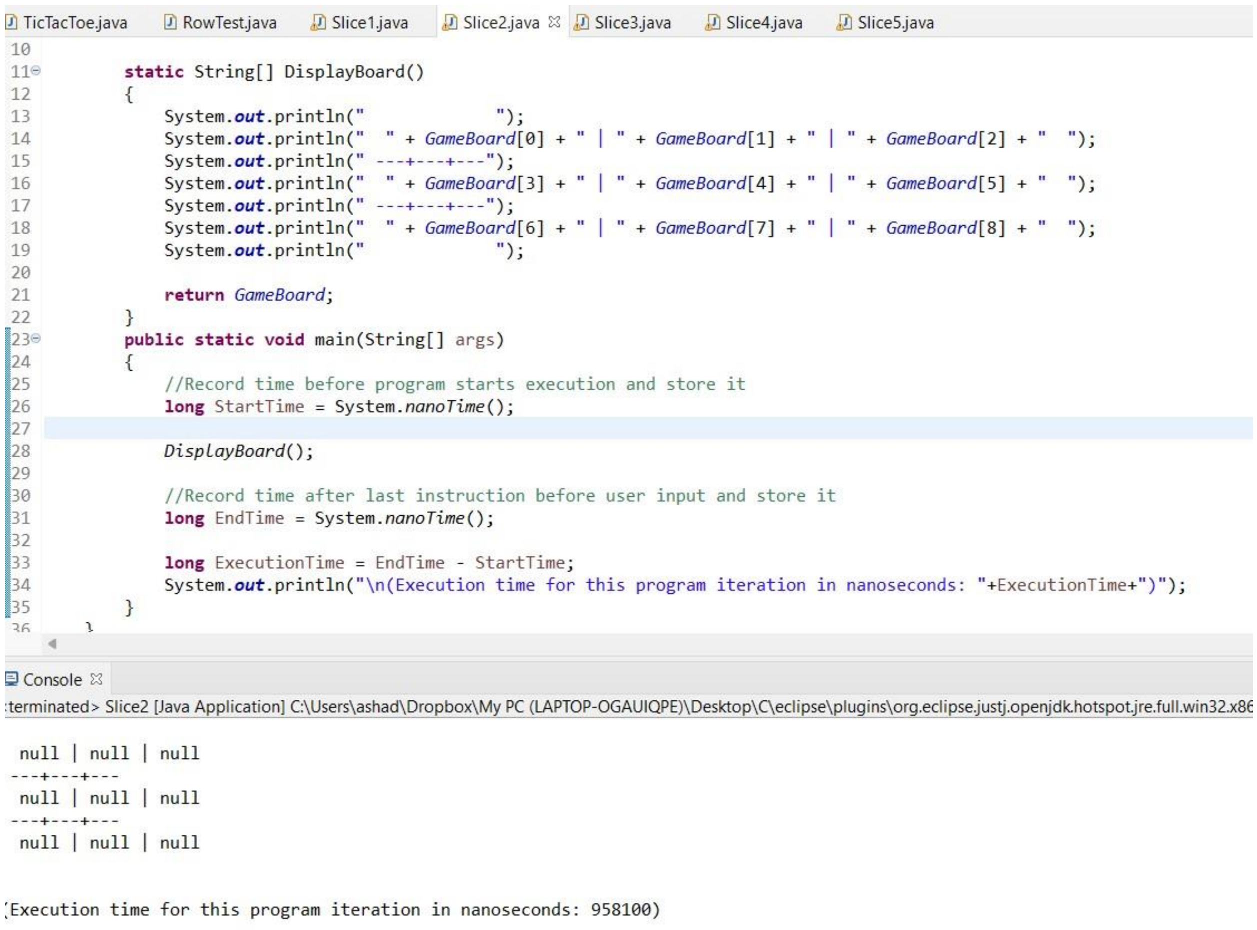
Slice1.java



```java
package GameFile;
import java.util.*;

//Creating a slice for variable PlayerInput in main method
public class Slice1 {

    public static void main(String[] args)
    {
        //Record time before program starts execution and store it
        long StartTime = System.nanoTime();

        Scanner input = new Scanner(System.in);
        System.out.println("Player X will start. Please enter a spot for X, from 1 to 9:");

        //Record time after last instruction before user input and store it
        long EndTime = System.nanoTime();

        long ExecutionTime = EndTime - StartTime;
        System.out.println("\n(Execution time for this program iteration in nanoseconds: "+ExecutionTime+")");


            int PlayerInput;

            PlayerInput = input.nextInt();

            if (!(PlayerInput > 0 && PlayerInput <= 9)) {
                System.out.println("Sorry! Please enter a number from 1 to 9 again!");
```

```
<terminated> Slice1 [Java Application] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\C\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.f
Player X will start. Please enter a spot for X, from 1 to 9:

(Execution time for this program iteration in nanoseconds: 95228900)
1
PlayerInput=1
```

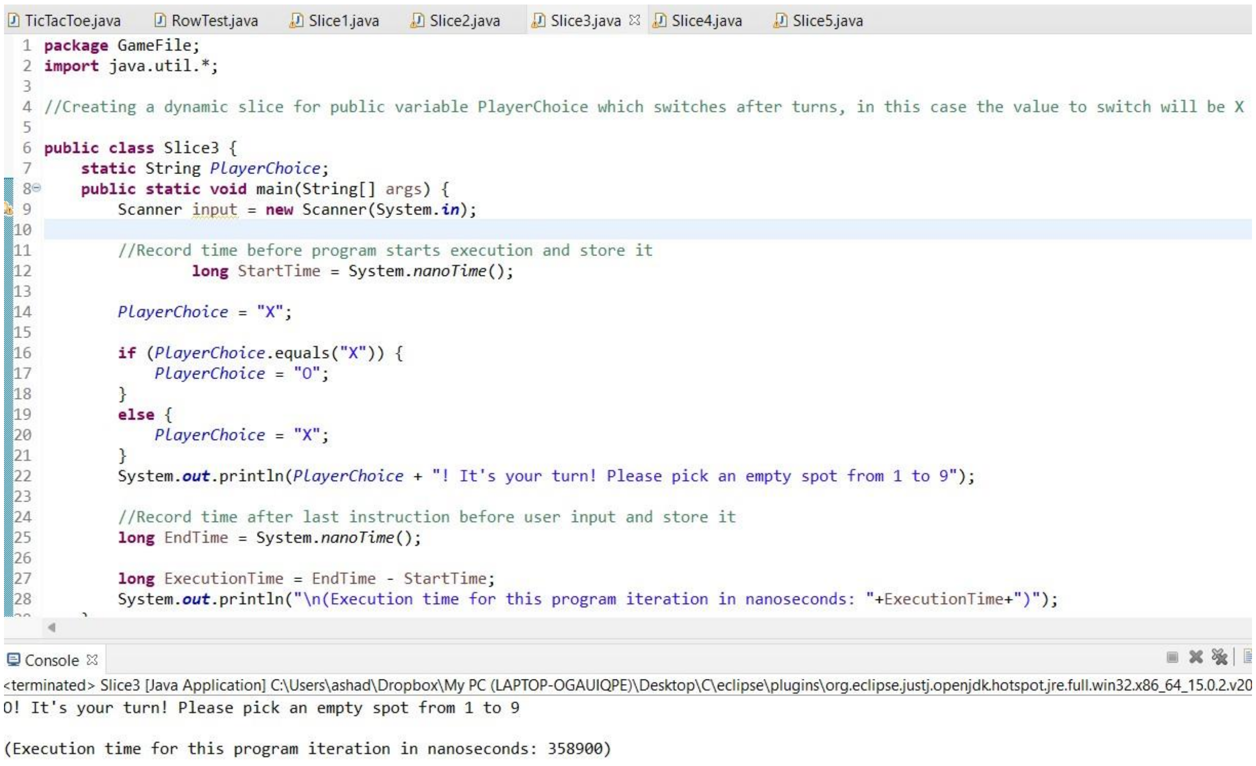
Slice2.java



```java

        static String[] DisplayBoard()
        {
            System.out.println("             ");
            System.out.println("  " + GameBoard[0] + " | " + GameBoard[1] + " | " + GameBoard[2] + "  ");
            System.out.println(" ---+---+---");
            System.out.println("  " + GameBoard[3] + " | " + GameBoard[4] + " | " + GameBoard[5] + "  ");
            System.out.println(" ---+---+---");
            System.out.println("  " + GameBoard[6] + " | " + GameBoard[7] + " | " + GameBoard[8] + "  ");
            System.out.println("             ");

            return GameBoard;
        }
        public static void main(String[] args)
        {
            //Record time before program starts execution and store it
            long StartTime = System.nanoTime();

            DisplayBoard();

            //Record time after last instruction before user input and store it
            long EndTime = System.nanoTime();

            long ExecutionTime = EndTime - StartTime;
            System.out.println("\n(Execution time for this program iteration in nanoseconds: "+ExecutionTime+")");
        }
    }
```

```
<terminated> Slice2 [Java Application] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\C\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86

 null | null | null
---+---+---
 null | null | null
---+---+---
 null | null | null


(Execution time for this program iteration in nanoseconds: 958100)
```

Slice3.java

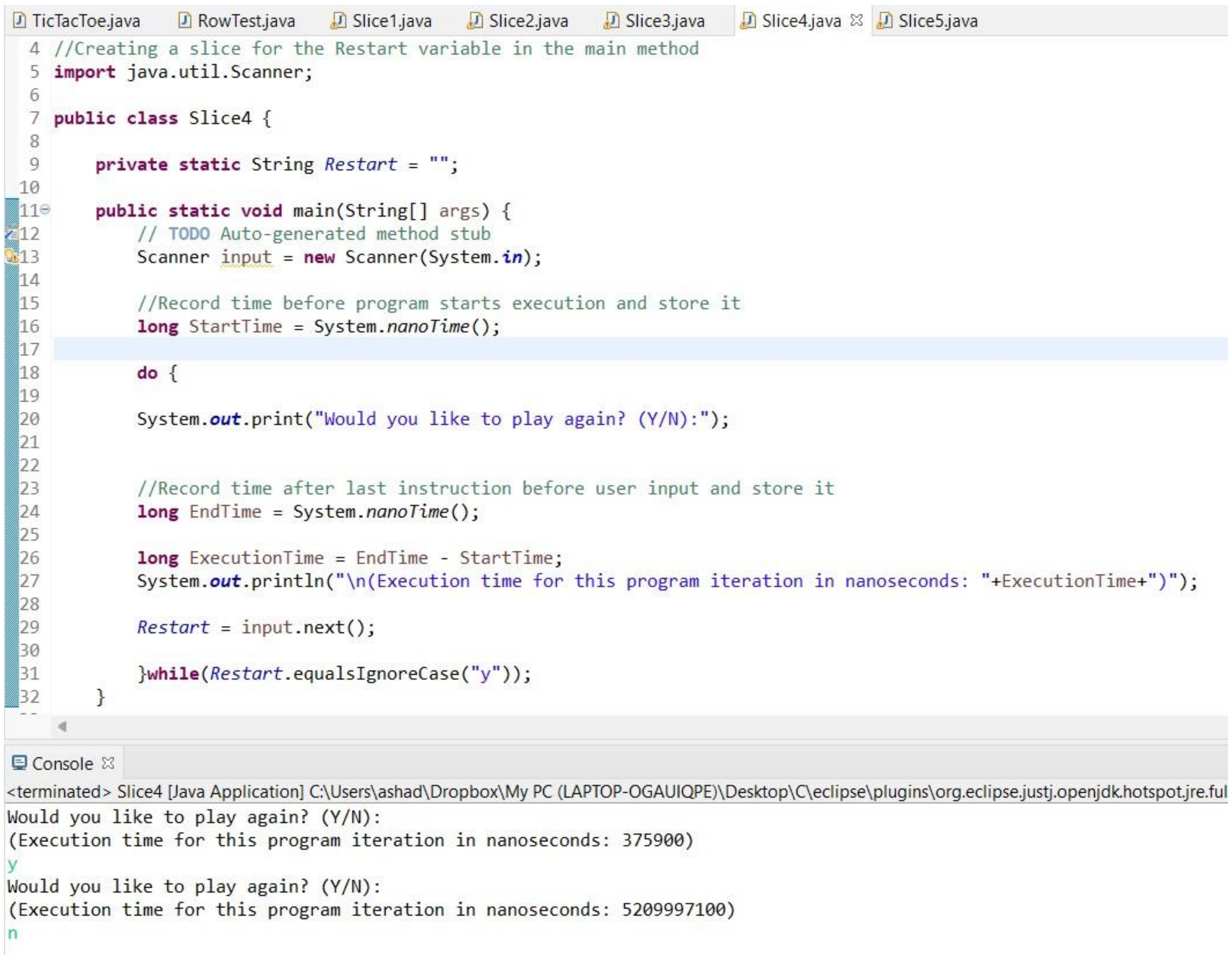

TicTacToe.java RowTest.java Slice1.java Slice2.java Slice3.java Slice4.java Slice5.java

```java
package GameFile;
import java.util.*;

//Creating a dynamic slice for public variable PlayerChoice which switches after turns, in this case the value to switch will be X

public class Slice3 {
    static String PlayerChoice;
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        //Record time before program starts execution and store it
                long StartTime = System.nanoTime();

        PlayerChoice = "X";

        if (PlayerChoice.equals("X")) {
            PlayerChoice = "O";
        }
        else {
            PlayerChoice = "X";
        }
        System.out.println(PlayerChoice + "! It's your turn! Please pick an empty spot from 1 to 9");

        //Record time after last instruction before user input and store it
        long EndTime = System.nanoTime();

        long ExecutionTime = EndTime - StartTime;
        System.out.println("\n(Execution time for this program iteration in nanoseconds: "+ExecutionTime+")");
```

Console

```
<terminated> Slice3 [Java Application] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\C\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20
O! It's your turn! Please pick an empty spot from 1 to 9

(Execution time for this program iteration in nanoseconds: 358900)
```

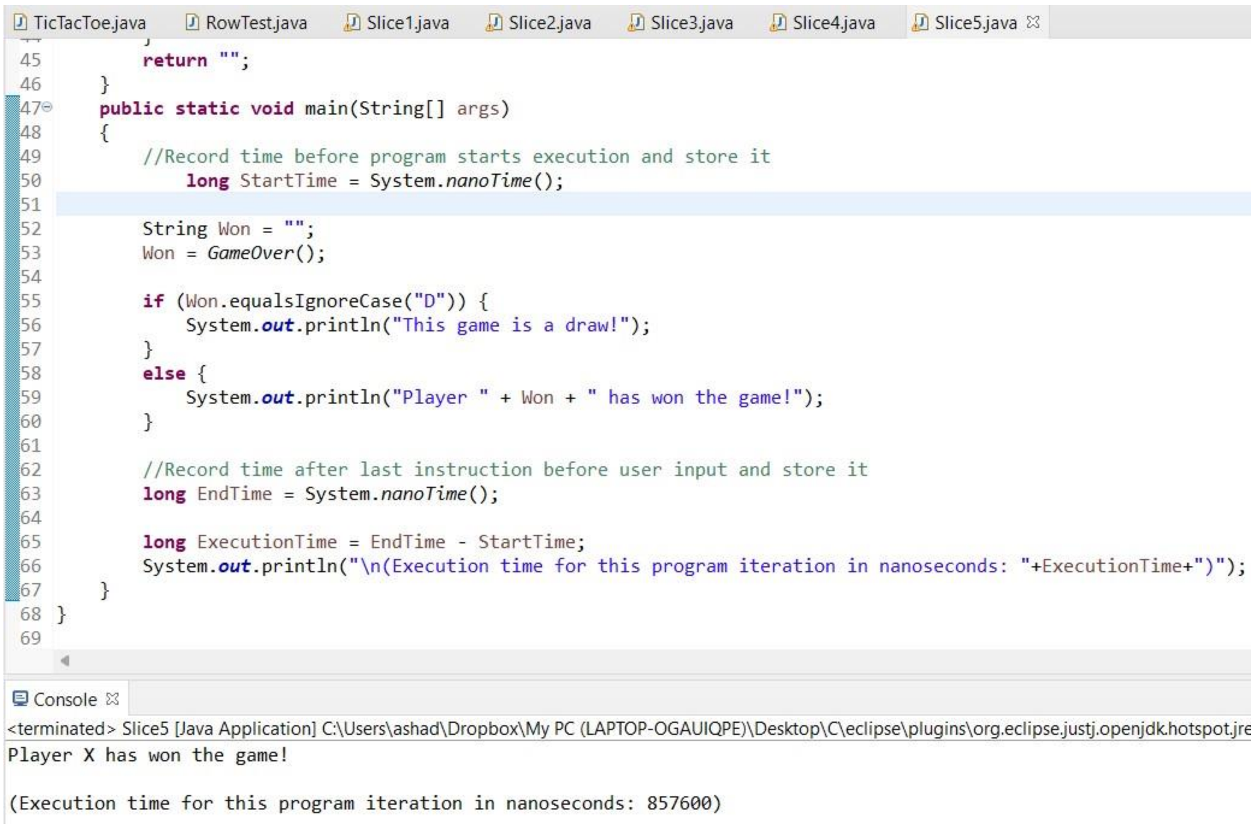
Slice4.java



```java
//Creating a slice for the Restart variable in the main method
import java.util.Scanner;

public class Slice4 {

    private static String Restart = "";

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner input = new Scanner(System.in);

        //Record time before program starts execution and store it
        long StartTime = System.nanoTime();

        do {

        System.out.print("Would you like to play again? (Y/N):");


        //Record time after last instruction before user input and store it
        long EndTime = System.nanoTime();

        long ExecutionTime = EndTime - StartTime;
        System.out.println("\n(Execution time for this program iteration in nanoseconds: "+ExecutionTime+")");

        Restart = input.next();

        }while(Restart.equalsIgnoreCase("y"));
    }
```

```
<terminated> Slice4 [Java Application] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\C\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.ful
Would you like to play again? (Y/N):
(Execution time for this program iteration in nanoseconds: 375900)
y
Would you like to play again? (Y/N):
(Execution time for this program iteration in nanoseconds: 5209997100)
n
```

Slice5.java



```java
        return "";
    }
    public static void main(String[] args)
    {
        //Record time before program starts execution and store it
            long StartTime = System.nanoTime();

        String Won = "";
        Won = GameOver();

        if (Won.equalsIgnoreCase("D")) {
            System.out.println("This game is a draw!");
        }
        else {
            System.out.println("Player " + Won + " has won the game!");
        }

        //Record time after last instruction before user input and store it
        long EndTime = System.nanoTime();

        long ExecutionTime = EndTime - StartTime;
        System.out.println("\n(Execution time for this program iteration in nanoseconds: "+ExecutionTime+")");
    }
}
```

```
<terminated> Slice5 [Java Application] C:\Users\ashad\Dropbox\My PC (LAPTOP-OGAUIQPE)\Desktop\C\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre
Player X has won the game!

(Execution time for this program iteration in nanoseconds: 857600)
```

Challenges and Solutions Discovered

The tasks to perform in this assignment went through many changes, so one of the main challenges was to adapt to the changing requirements of this assignment. Initially, we were tasked to create a program that reads our application for us and creates slices. However, this was changed later to allow manual slicing input as well, which is what this submission focuses on. Other than this, in part a, since this game did not have a lot of variables and classes to create slices of, it was a challenge to choose these variables, but with some changes and additions to the original game class it was solved. For part b, Instrumentation was to be used, however, when researching this concept in dynamic analysis, information seemed to be scarce on the internet. The main challenge was to understand it before coding, which was an extremely time-consuming task as these concepts seem to be outdated.