Lab 06 – Rock, Paper, Scissors

Recommended Group Brainstorm (NO computers during this time)

Good programmers think before they begin coding. Part I of this assignment involves brainstorming with a group of peers with no computers to talk about a strategy for solving this week's lab. Breakup into groups based on your seating (3-4 people per group) and brainstorm about how to solve the problems below. Make sure everyone understands the problem and sketch out potential ways to move toward a solution. You may find it helpful to look over the required readings for this week.

Note: Brainstorms are to help you get started with the lab and get you in the habit of designing before you code. You won't submit them to eLC.

Introduction

In the game *Rock, Paper, Scissors*, two players each choose either "rock," "paper," or "scissors," and then show their choice to the other player at the same time. Each choice wins against one other choice and loses to one other choice:

- Rock wins against scissors, but loses against paper.
- Paper wins against rock, but loses against scissors.
- Scissors win against paper, but lose against rock.
- If both players make the same choice, the result is a tie.

Assignment

In this lab, you will implement a version of *Rock, Paper, Scissors* where the user plays against the computer. The user puts in their choice as a String (either **`"rock"`**, **`"paper"`**, or **`"scissors"`** - not case sensitive), and the computer will make its choice at random. The winner should be determined based on the rules above.

The user should be able to play **multiple rounds** of *Rock, Paper, Scissors* against the computer. The program should first **ask the user how many wins they want to play**. Your program should keep track of the score of both the computer and the user. **If there is a tie, neither player gets a point**. The program should keep running until either the player or the computer wins the number of games they specified at the start. So, for example, if the user inputs 3 when prompted, the game should continue until either the user *or* the computer wins three rounds (*not* once three rounds have been played).

After each turn, the program should print who won (either the player or the computer) and the score after that turn. Show the player's score first (for example, if the computer has won once and the player has not won, show (0-1)). More examples of this can be seen in the sample runs below.

Instructions

Start by creating a new class named `RockPaperScissors`. Download the file `ComputerOpponent.java` from the CSCI 1301 site. It contains the code that generates the computer's move for you. Once you have downloaded it, move the file into the src folder of your RockPaperScissors project. It should be located directly alongside your `RockPaperScissors.java` file in the same src folder.

Once you have both files in Eclipse, feel free to explore `ComputerOpponent.java`. There's nothing in the code beyond anything we've covered already. Notice the variable,

```
static boolean TESTING_MODE = false;
```

When `TESTING_MODE` is false, `ComputerOpponent.java` will generate computer moves randomly using `Math.random()`. However, if you change `TESTING_MODE` to `true`, the moves that the computer generates will always come in a fixed order: *rock*, *paper*, *scissors*, *rock*, *paper*, *scissors*, *rock*, *paper*, *scissors*, … and so on. You may find it useful to change `TESTING_MODE` to `true` when you are testing and debugging your program, since the computer's moves can be predicted. This will also help our TA's when they are grading your code.

We will use our own version of `ComputerOpponent.java` when grading your lab, so don't worry about messing with the functionality of that file (but make sure that all of the code *you* write stays in `RockPaperScissors.java`). ***Do not* rename `ComputerOpponent.java`, as doing so will cause a compilation error when we try to grade your lab.**

Once both java files are in the same src folder, you can get the computer move from `ComputerOpponent.java` by using its **`getMove()` method.** This method will return a String value which will be either `"rock"`, `"paper"`, or `"scissors"` (each being all lowercase). Here is the line of code to get the computer's move:

```
String computerMove = ComputerOpponent.getMove();
```

On each turn, you should prompt the user for their move and retrieve it via the `nextLine()` method. The game should be able to recognize "rock", "paper", and "scissors" *regardless* of capitalization. If the user types something else, ask them to try again. **Do not update either player's score if the user did not enter a valid move**.

Considerations While Brainstorming

Your program will be doing a lot of the same things over and over again, which means that a loop will be part of your code. Consider the following when brainstorming:

- How do we keep track of the player's and computer's scores?
- Which tasks belong *inside* the loop?
    - When you write your code, it may be easier to write the code for *one* iteration of the loop first, then putting that code inside a loop statement.
- Which tasks belong *outside* the loop?
- Should our call to `ComputerOpponent.getMove()` be *inside* or *outside* the loop? What happens if we don't put it where it should be (i.e., inside the loop when it should be outside, or outside the loop when it should be inside)?
- What should the condition for the loop be?
    - In other words, under what circumstances should the loop keep going?
- How do we handle unintended input, like `Spock` or `gun`?

  - We can do better than simply ending the game immediately!

Additional Requirements

These are things that make the graders lives easier, and ultimately, you will see in the real world as well. Remember the teaching staff does not want to touch your code after they gave you requirements; they want to see the perfect results they asked for! Here is a checklist of things you can **lose points** for:

- (10 points) If the source file(s)/class(es) are named incorrectly (case matters!)
- (10 points) If your source file(s) have a package declaration at the top
- (10 points) If any source file you submit is missing your Statement of Academic Honesty at the top of the source file. All submitted source code files must contain your Statement of Academic Honesty at the top of each file.
- (10 points) If you have more than one instance of Scanner in your program. Your program should only have one instance of Scanner.
- (15 points) Inconsistent I/O (input/output) that does not match our instructions or examples exactly (unless otherwise stated in an assignment's instructions). Your program's I/O (order, wording, formatting, etc.) must match our examples and instructions.
- (100 points) If the source file(s) are not submitted before the specified deadline's late period ends (48 hours after the deadline) or if they do not compile.
- (25 points) Late penalties will be deducted as per the course syllabus.
- If your (10 points) comments or (10 points) variables are "lacking"
  - Here, "lacking" means that you or a TA can find **any** lines of code or variables that take more than 10 seconds to understand, and there is no comment, or the variable name does not make sense (variable names like **b**, **bb**, **bbb**, etc. **will almost never be acceptable)**
- (10 points) Indentation is not consistent throughout your source code
  - Refresh your memory of indentation patterns in chapter 2 in the course textbook
  - Be careful of a combination of tabs and spaces in your files (use one or the other)!
- (100 points) For this assignment, you are **NOT** permitted to use a **break** or **continue** statement inside of the body of a loop, and using one of these statements in the body of a loop will result in a grade of zero on this assignment.

If any of the above do not make sense to you, talk to a TA, or ask on Piazza!

Submission Guidelines

Once complete, submit ***only*** your completed **`RockPaperScissors.java`** to eLC. You do *not* need to submit your `ComputerPlayer.java` file, or any `.class` files. Be sure to include the Academic Honesty Policy comment at the top of your file. All instructions and requirements must be followed; otherwise, points maybe deducted.

Examples

Bold text represents the user's input; all other text should be printed by your program.

Winning against the computer:

```
Points to win: 5
Rock, paper, or scissors? rock
The computer chose scissors, so you win! (1-0)
Rock, paper, or scissors? paper
The computer chose paper, so it's a tie. (1-0)
Rock, paper, or scissors? scissors
The computer chose rock, so you lose. (1-1)
Rock, paper, or scissors? paper
The computer chose rock, so you win! (2-1)
Rock, paper, or scissors? rock
The computer chose scissors, so you win! (3-1)
Rock, paper, or scissors? rock
The computer chose scissors, so you win! (4-1)
Rock, paper, or scissors? scissors
The computer chose paper, so you win! (5-1)
Congratulations! You won!
```

Losing to the computer:

```
Points to win: 3
Rock, paper, or scissors? rock
The computer chose paper, so you lose. (0-1)
Rock, paper, or scissors? paper
The computer chose scissors, so you lose. (0-2)
Rock, paper, or scissors? scissors
The computer chose rock, so you lose. (0-3)
Sorry, you lost. Better luck next time!
```

Edge cases for user input:

```
Points to win: 3
Rock, paper, or scissors? pApEr
The computer chose rock, so you win! (1-0)
Rock, paper, or scissors? scisor
Please choose 'rock', 'paper', or 'scissors'!
Rock, paper, or scissors?     scissors
The computer chose scissors, so it's a tie. (1-0)
Rock, paper, or scissors? sCISSORS
The computer chose scissors, so it's a tie. (1-0)
(and so on...)
```