Object Oriented Programming

Assignment 1

Game Development

Marks available: 100

Percentage of overall marks from this assignment: 10%

Date issued: 28/09/20

Deadline: 16/10/20, 17:00

Introduction

For this assignment you will be creating some simple computer games.

Submission instructions

Submit your work by the deadline above as a **zip** archive[1]. Submit only .java source files in your archive unless told otherwise.

Make absolutely sure you have are not submitting .class files instead of .java files.

Where applicable, your class for each of the exercises must have the name shown in the exercise heading.

Each Java file must have a package declaration at the topic of it (these declarations must be on the very first line of the file). The final part of this declaration varies according to the assignment and exercise. The package declaration for this assignment is:

```
package com.bham.pij.assignments.rps;
```

Do not copy and paste this package declaration from this pdf! You will get unexpected characters in your files that will render them untestable. Type it in.

Your package declaration must be on the very first line of your file. Do not put anything above the package declaration, not even comments.

[1] If you create the zip file on MacOS, make sure it can be opened again properly before you submit it.

Do not forget to add the semicolon at the end of this line. All characters in a package name should be lower case.

In each of these exercises, you are not restricted to only creating the required methods but you must create **at least** the required methods.

All of the required methods must be `public`.

None of the methods you create should be declared as `static`.

Do not use any of the Java collections for these exercises, e.g. `ArrayList`.

Do not use the `Arrays` class.

Do not use the regular expressions[2] in this assignment (you may pass single characters to methods that take a regular expression if you want to, but no more than a single character).

Do not use non-ASCII characters anywhere in your files, even as comments.

Do not output to the screen using `System.out.println()` unless otherwise instructed[3].

All strings can be treated as case insensitive unless otherwise stated.

IMPORTANT: Do NOT put any code that gets **user input** in the required methods below. If you do, your work can't be tested. This means that you should not use `Scanner` in any required method or in any method called by a required method, etc. Please ask if unsure about this.

Exercise 1: `RockPaperScissors` [50 marks]

For this exercise you can use the already-prepared Java file `RockPaperScissors.java` that can be downloaded from Canvas. The file shows you, in the comments, where you need to add your code. Do not change anything else in this file.

The game of 'Rock Paper Scissors'

Rock Paper Scissors (RPS) is a simple game for two players. In the real world it is played by two people. For each turn, each person simultaneously makes the approximate shape of a rock, paper or scissors behind their back. At the count of three, both players reveal the shape they created to the other player. The winner is based on the following rules:

Rock beats scissors (because scissors can't cut a rock).
Scissors beat paper (because scissors can cut paper).
Paper beats rock (because you can wrap the paper around the rock).

If the players create the same shape, the game is a draw.

[2] If you don't know what regular expressions are, don't worry about it.
[3] You can use it for your own debugging but make sure you delete those lines of code before you submit

The game created for you

The file `RockPaperScissors.java` contains a game of RPS. You do not have to create the game. You will just add some code to it. **The game will not work properly until you have added your code.**

The game works as follows. When you run the program the user is presented with a request to press the 'r' key to choose to play 'rock', or the 'p' key to play 'paper', or the 's' key to play 'scissors'. Once the player has pressed one of those keys, the game will randomly choose one of the three (rock, paper, scissors) to play as 'the computer'. It then prints the winner of that 'round' to the screen and waits for the player to play again. The player presses 'x' to quit the game. When they do that the game prints the number of rounds won by the player and the number of rounds won by the computer to the screen.

What you need to do

You will add code to four methods. The methods you need to add code to are clearly identified in the file `RockPaperScissors.java`. The requirements for each method are stated in that source file as comments, and also presented more clearly in the HTML documentation for this class that can be downloaded from Canvas. The HTML documentation states what each method needs to do. You need to add the code to each method to make it function as required.

You will learn fully about methods during this module but please bear in the mind the following so you can get started.

As an example, let's look at the following method:

```
public boolean isValidTurn(int turn)
```

You will see that this is the first line of that method. We call this the *signature*.

This signature has the following parts:

`public`: You will learn fully what this means later on. Simply put, it just means that the method is 'available' to be used by the rest of the program. For now we simply add that as the very first part of the signature.

`boolean`: This is the type of the *return value* of the method. This means that this method **must** return a boolean value, which means it must return the value true or false.

`isValidTurn`: This is the name of the method. This is the name that other parts of the program must use in order to use this method.

`int turn`: This is the parameter that is being passed to this method. It is of type `int` and its name is `turn`.

`( )`: The round brackets contain the parameters of the method. In this case there is only one (called `turn`, as above).

Look at the method signature again. This method will 'answer a question'. The question is "is this turn (the one passed as a parameter) a valid one". If the turn is valid it will answer that question with 'yes' by returning the value `true`. If it is not valid it will answer with 'no' by returning `false`.

As you will see in the HTML documentation, a valid turn has the integer value 1, 2 or 3 (one each for rock, paper and scissors). Any value that is not in the range 1 to 3 (inclusive) is, therefore, invalid. The method `isValidTurn` needs to check if the parameter is in the range 1 to 3. If it is it should return `true`, and if not it should return `false`.

Important: All of the methods you need to create for this assignment already exist in the files. You should add your code where stated in the files. Please note, however, that in order for the files to compile without errors as they stand, `return` statements have been added to the methods. You will need to delete those and add your own. You will learn about these topics soon so don't worry.

In summary, therefore, you need to get the file `RockPaperScissors.java` and the HTML documentation. You then need to look in the file and the documentation to see which methods you need to complete. Then you need to add the code to those methods to make them work as specified in the documentation.

Exercise 2: `Hangman` [40 marks]

For this exercise you can use the already-prepared Java file `Hangman.java` that can be downloaded from Canvas. The file shows you, in the comments, where you need to add your code. Do not change anything else in this file.

The game of 'Hangman'

Hangman is a guessing game. In the real world, one player chooses a target word that the other player(s) must guess. This target is written down on paper (usually) as a series of underscores representing each letter. This shows the guessing player how many letters are in the word but not the word itself.

The guessing player guesses announces letters they believe might be in the word. If the guess is correct, the player who set the target writes that letter on the paper in all positions in which it exists in the target word. If the letter does not exist in the target word, the player who set the target instead draws one piece of the 'gallows'[4]. If the target-setting player has drawn the entire gallows before the guessing player has guessed the word correctly, the guessing player has lost the game.

The game created for you

The file `Hangman.java` contains a game of Hangman. You do not have to create the game. You will just add some code to it. **The game will not work properly until you have added your code.**

The game works as follows. When you run the program the user is presented with a request to choose a letter, and a representation of the target word (this is the 'current guess'). When the player presses a key, the game will redraw the current guess with the new letter added, if it is in the target word. If the letter is not in the target word the game will increment the number of gallows pieces and print that value on the screen.

What you need to do

As for the previous exercise, you need to look at the Java file provided and the documentation and complete the required methods according to the specification in the documentation.

[4]There is no fixed number of gallows pieces. Every player seems to choose their own style of gallows.

Exercise 3: `FalloutTerminal` [10 marks]

This exercise is quite a bit harder than the previous two. It is only worth 10% of the marks for this assignment (which is itself only worth 10% of the module marks). Attempt this exercise only when you are happy with your solutions to the first two exercises.

This game is based on a 'mini game' from the video game series 'Fallout' (Bethesda Studios). You don't need to know anything about that game to attempt this exercise, but if you google 'Fallout terminal hacking' and select 'images' you will see some examples of what this mini game looks like. Also, if you are familiar with the game 'Mastermind' (and related), that is very similar.

The game of 'Fallout Terminal'

In this game you must 'hack' a terminal. You hack the terminal by guessing the password. You have four guesses in each game. If you get the password right within four guesses you are granted access to the terminal. If you fail to get the password right you are 'locked out' of the terminal.

In the figure you can see this game running[5]. You can see that the terminal screen appears to be showing a 'memory dump'. Each 'section' of memory starts with a memory address written in hexadecimal, for example, `0xADC0`[6].The `0x` at the start of the number is not part of the number but signifies that the number is in hexadecimal format. This address is the address of only the character immediately adjacent to it. All of the other characters have their own memory address. Each memory address is one byte (8 bits) higher than the previous one.

It is assumed that the password we require is somewhere in this memory dump. You can see that amongst the rndom characters there are some words. All of these words are 8 characters in length. One of them is

[5]Our version, not the real one.

[6]If you don't know much about hexadecimal, now is the time to look it up.

the password. The passwords is randomly selected from one of the visibile words each time the game is run.

In order to guess the password the player must enter an attempt via the keyboard. The guess entered must be one of the words shown in the terminal. If the guess is the actual password, access will be granted[7]. If the guess is incorrect, the terminal will state how many characters in the guess are correct. This is referred to as 'Likeness'. For example, if the password is PRESTIGE and the guess is MARRIAGE the likeness is 2 (the 'G' and the 'E' at the end). To count as a likeness, the characters have to be in the exact same position as in the password. If a letter is present but in a different position, that doesn't count.

What you need to do

There is no documentation for this exercise because you only need to create one method. You can use the file `FalloutTerminal.java` for the rest of the code.

The method you need to create has the following signature:

```
public String buildDisplayLine(int startAddress, float wordProbability)
```

This method creates a **single line of the display**. If you look at the figure, you can see that a single line of the display has two sections of memory in it, because there are two addresses. Each line has some random characters and *possibly* a word. This is your only responsibility in this exercise.

This method receives as parameters the starting address for the line (`startAddress`) and the probability that the line will contain a word (`wordProbability`). It returns the line as a `String`. There is a maximum of one word per line.

The parameter `startAddress` contains the address of the first character in the line. You will see from the figure that halfway along the line there is another address. You need to compute the value of this and include it in the line. You can compute it using the following information. Each character is one byte in size. The whole line has 60 characters (excluding addresses and spaces).

The parameter `wordProbability` will be a value in the range 0-1. If the value of this parameter is 0.6 or less (≤ 0.6) then the line must contain a random word. You can use the provided method `getRandomWord()` to do this. Once you have chosen a random word, you must 'save' it by calling the method `addSelectedWord()` passing the string containing the word as a paramter. If the line does not contain a word it should contain only of printable ASCII characters that are not alphanumeric or white space characters. Look at the figure to see the type of characters that should be included[8].

In words, the format of the string that should be returned is as follows:

1. The starting hexadecimal address.
2. A space character.
3. 30 characters, possibly including a word.
4. A space character.

[7] Unfortunately, nothing will happen in this game when access is granted.
[8] If you don't know anything about ASCII, now is the time to look that up too.

5. The hexadecimal address of the 31st character.

6. A space character.

7. 30 characters, possibly including a word.

Remember though: there is a maxiumum of one word per line.

If the line contains a word, that word should not be split across the two sections of the line. It must be fully within the left hand or right hand section.

Notes:

You will see in the figure above that no words are repeated. This is not a requirement of this exercise but you could implement it so it works that way. It looks better that way.

Setting up Eclipse for the assignment

In Eclipse click 'File' and then 'New' → 'Java Project'. Give the project a name. Any name will do. Make sure 'Use default location' is checked (unles you know what you are doing). Where it says 'Use an execution environment JRE', make sure 'JavaSE-11' is selected. Also make sure that 'Create separate folders for sources and class files' is checked. Now click 'Finish'. On the next dialog, 'New module-info.java', click 'Don't cCeate'. Your project should now be visible in the 'Package Explorer'.

In the 'Package Explorer', right-click on the 'src' folder in your new project (you will first need to expand the project by clicking on it). Now select 'New' → 'Package'.

In the 'Name' field enter `com.bham.pij.assignments.rps`. Check you have typed this in exactly as shown. Now click 'Finish'.

You can do the next step a number of ways. Here's one way. Copy the three Java files provided for this assignment to the folder {Your-project-name}/src/com/bham/pij/assignments/rps. You must replace {Your-project-name} with whatever you called the project. Note that the path here is expressed as a unix (and Mac) path. Whichever system you are on, use a 'File Explorer' to find the folder.

Once you have copied the files to the correct location, go back into Eclipse, select your project again and press 'F5' (or right-click on the project and choose 'Refresh'). The three files should now appear in your project, under the package you just created.

To run each program (one of the three provided), right-click on the Java file for that program in the package explorer, and then select 'Run As' → 'Java Application'. The programs should run when you do this but they won't work properly yet. That is your job by completing this assignment!

Test Details and Marking Scheme

This section shows the overall requirements for each exercise, the tests that will be undertaken by the automatic marking system, and the marking scheme.

RockPaperScissors

Required class name:

com.bham.pij.assignments.rps.RockPaperScissors

Exercise marks:

50.0

Required methods:

public boolean isValidTurn(int)
public int getPlayerTurn(String)
public String getWinner(int, int)
public String getTurnAsString(int)

Test list for this exercise:

Test 1
Method tested: isValidTurn()
Test purpose: This test checks only valid inputs to the isValidTurn() method.
Test fails because: Your isValid() method returns false for one or more valid inputs when it should return true. Check that you considered edge cases.
Marks for this test: 3.0

Test 2
Method tested: isValidTurn()
Test purpose: This test checks only invalid inputs to the isValidTurn() method.
Test fails because: Your isValid() method returns true for one or more invalid inputs when it should return true. Check that you considered edge cases.
Marks for this test: 4.0

Test 3
Method tested: getPlayerTurn()
Test purpose: This test checks only invalid inputs to the getPlayerTurn() method.
Test fails because: Your getPlayerTurn() method returns incorrect results for one or more invalid inputs.
Marks for this test: 4.0

Test 4
Method tested: getPlayerTurn()
Test purpose: This test checks only valid inputs to the getPlayerTurn() method.
Test fails because: Your getPlayerTurn() method returns correct results for one or more valid inputs.
Marks for this test: 4.0

Test 5
Method tested: getWinner()
Test purpose: This test checks only valid inputs to the getWinner() method with the player as winner.
Test fails because: Your getWinner() method returns correct results for one or more valid inputs with the player as winner.
Marks for this test: 9.0

Test 6
Method tested: getWinner()
Test purpose: This test checks only valid inputs to the getWinner() method with the computer as winner.

Test fails because: Your getWinner() method returns correct results for one or more valid inputs with the computer as winner.
Marks for this test: 9.0

Test 7
Method tested: getWinner()
Test purpose: This test checks only valid inputs to the getWinner() method that should result in a draw.
Test fails because: Your getWinner() method returns correct results for one or more valid inputs that should result in a draw.
Marks for this test: 9.0

Test 8
Method tested: getTurnAsString()
Test purpose: This test checks your getTurnAsString() method returns the correct results for valid values.
Test fails because: Your getTurnAsString() method returns incorrect results for one or more inputs.
Marks for this test: 4.0

Test 9
Method tested: getTurnAsString()
Test purpose: This test checks your getTurnAsString() method returns the correct results for invalid values.
Test fails because: Your getTurnAsString() method returns incorrect results for one or more inputs.
Marks for this test: 4.0

Total test marks for this exercise: 50.0

Hangman

Required class name:
com.bham.pij.assignments.rps.Hangman
Exercise marks:
40.0
Required methods:
public boolean targetContains(String, char)
public boolean hasFoundTarget(String, String)
public String getCurrentGuessForDisplay(String)
public String updateCurrentGuess(String, String, char)

Test list for this exercise:

Test 1
Method tested: targetContains()
Test purpose: This test checks your targetContains() method with only valid values.
Test fails because: Your targetContains() method did not return the correct value for a valid input.
Marks for this test: 5.0

Test 2
Method tested: targetContains()
Test purpose: This test checks your targetContains() method with only invalid values.

Test fails because: Your targetContains() method did not return the correct value for an invalid input.
Marks for this test: 5.0

Test 3
Method tested: hasFoundCurrent()
Test purpose: This test checks your hasFoundCurrent() method correctly identifies when the player has found the target word.
Test fails because: Your hasFoundCurrent() method does not correctly identify that the guess matches the target.
Marks for this test: 5.0

Test 4
Method tested: hasFoundCurrent()
Test purpose: This test checks your hasFoundCurrent() method correctly identifies when the player has not found the target word.
Test fails because: Your hasFoundCurrent() method does not correctly identify that the guess doesn't match the target.
Marks for this test: 5.0

Test 5
Method tested: updateCurrentGuess()
Test purpose: This test checks your updateCurrentGuess() method correctly updates the guess with a valid letter.
Test fails because: Your updateCurrentGuess() method does not correctly update the guess with a valid letter
Marks for this test: 6.0

Test 6
Method tested: updateCurrentGuess()
Test purpose: This test checks your updateCurrentGuess() method correctly does not update the guess with an invalid letter.
Test fails because: Your updateCurrentGuess() method updates the guess with a invalid letter
Marks for this test: 7.0

Test 7
Method tested: getCurrentGuessForDisplay()
Test purpose: This test checks that your getCurrentGuessForDisplay() method returns the correct String.
Test fails because: Your getCurrentGuessForDisplay() method does not return the correct String.
Marks for this test: 7.0

Total test marks for this exercise: 40.0

FalloutTerminal

Required class name:
com.bham.pij.assignments.rps.FalloutTerminal
Exercise marks:
10.0
Required methods:

public String buildDisplayLine(String,String)

Test list for this exercise:

Test 1
Method tested: buildDisplayLine()
Test purpose: This test checks that your buildDisplayLine() method returns a string of the correct length.
Test fails because: Your buildDisplayLine() method returns a string of an incorrect length.
Marks for this test: 2.0

Test 2
Method tested: buildDisplayLine()
Test purpose: This test checks that your buildDisplayLine() method returns a string containing the correct addresses.
Test fails because: Your buildDisplayLine() method returns a string with incorrect addresses.
Marks for this test: 2.0

Test 3
Method tested: buildDisplayLine()
Test purpose: This test checks that your buildDisplayLine() method returns a string containing random characters.
Test fails because: Your buildDisplayLine() method returns an incorrect string.
Marks for this test: 3.0

Test 4
Method tested: buildDisplayLine()
Test purpose: This test checks that your buildDisplayLine() method returns a string containing random characters.
Test fails because: Your buildDisplayLine() method returns an incorrect string.
Marks for this test: 3.0

Total test marks for this exercise: 10.0