University of London

Computing and Information Systems/Creative Computing

CO1109 Introduction to Java and object-oriented programming

Coursework assignment 1 2018–19

Introduction

This is coursework assignment 1 (of two coursework assignments in total) for 2018–19. The assignment asks that you demonstrate an understanding of random numbers, variables including class variables, user input with the *Scanner* class, the *if – else* statement, loops and static methods with their return types. In addition, this assignment concentrates on how to write code that is readable.

Electronic files you should have:

- *GamesArcade.java*
- *WordScramble.java*

Note that user input is not verified in the classes that you have been given, so the wrong input (a `String` where a method expects an `int`, say) can cause the *GamesArcade* to end with an exception. You should ignore this as there are no marks in this assignment for verifying user input, or for recovering from the effects of invalid input.

What you should hand in: very important

There is one mark allocated for handing in uncompressed files – that is, students who hand in zipped or .tar files or any other form of compressed files can only score a maximum of 99/100 marks.

There is one mark allocated for handing in just the .java files asked for without putting them in a directory; students who upload their files in directories can only achieve 99/100 marks.

There are two marks for (1) not changing the names of the classes given to you, and (2) for naming any classes that you have to write (in this assignment *NumberGuessingGame*) ***exactly*** as you have been asked to name them.

Anything added to the names given means that your files are wrongly named, for example the following count as wrong names:

- *JSmith_GamesArcade.java*
- *cwk1-GamesArcade.java*
- *JSmith-109-assignment1-GamesArcade.java*
- *GamesArcade .java*

At the end of each section there is a list of files to be handed in – **please note the hand-in requirements supersede the generic University of London instructions**. Please make sure that you give in **electronic versions** of your .java files, since you cannot gain any marks without handing in the .**java** files asked for. Class files are **not** needed, and any student giving in only a class file will not receive any marks for that part of the coursework assignment, **so please be careful about what you upload as you could fail if you submit incorrectly**.

Programs that do not compile will not receive any marks.

The examiners will compile and run your Java programs; students who hand in files containing their Java classes that cannot be compiled (*e.g.* PDFs) will not be given any marks for that part of the assignment.

Please put your name and student number as a comment at the top of each .java file that you hand in.

Coursework assignment 1

1.0 THE QUESTIONS

1.1 Introduction

Compile the two classes that you have been given and run the *GamesArcade*. You should see the following menu:

```
GAMES ARCADE

Choose one of the following options:
1. Number Guessing Game
2. Word Scramble
3. Quit

Enter your choice:
```

You will find that if you choose option 1, the output you will see is:

```
Enter your choice: 1
I don't know how to do that.

Choose one of the following options:
1. Number Guessing Game
2. Word Scramble
3. Quit

Enter your choice:
```

Your task is to write the number guessing game and add it to the menu of the *GamesArcade*. You will find the game described in section 8.9.14 of volume 1 of the subject guide, and you will find the solution as number 60 in *Appendix C, Answers to exercises*. The solution in the guide is quite straightforward, with all statements

written in the main method. In this assignment, you are asked to ignore the solution in the subject guide, and write a solution with methods. This is because good object-oriented code has methods, with the main method used to test the methods written.

1.2 Readable code

This assignment is following the advice given by Robert C. Martin in his book *Clean Code: A Handbook of Agile Software Craftsmanship* (published 2008 by Prentice Hall, ISBN 978-0132350884). Mr Martin describes a system for writing readable code; there are others, but this is the one that this assignment will be focusing on.

Mr Martin writes:

> One difference between a smart programmer and a professional programmer is that the professional understands that *clarity is king*. [...] We want to use the popular paperback model whereby the author is responsible for making himself clear and not the academic model where it is the scholar's job to dig the meaning out of the paper.

Mr Martin writes that 'making your code readable is as important as making it executable'. He believes that names of variables, methods and classes are a major part of what makes our code readable:

> The name of a variable, function or class should answer all the big questions. It should tell you why it exists, what it does, and how it is used. If a name requires a comment, then the name does not reveal its intent.

Martin dislikes comments, noting that as code is updated comments are rarely updated at the same time; so, however helpful a comment at the start, once a class has been in use for a while any comments are likely to be outdated and confusing. He believes that code should be written with names that make the intent clear, such that comments are redundant.

You should note that the programmer has tried to follow the advice given by Martin in writing the *GamesArcade* and the *WordScramble* classes. However, Martin himself notes that names can always be improved, and we should not be afraid to keep refining our code.

See the appendix for an example of renaming a simple class to make it more readable.

When answering questions in this assignment, please refer to the following rules from Martin:

1.3 Formatting

> "You should take care that your code is nicely formatted. You should choose a set of simple rules that govern the layout of your code, and then you should consistently apply those rules. […] It helps to have an automated tool that can apply those formatting rules for you."

1.4 Methods

- **Method should do one thing only**. If your method does more than one thing, break it into separate methods.

- **Do not repeat yourself** – if you find yourself writing the same code more than once, put it into a method.
- **Too many arguments (parameters to a method).** "No argument is best, followed by one, two and three. More than three is very questionable and should be avoided with prejudice."

1.5 Comments

If you do write a comment, make sure it is grammatical, short, does not state the obvious and is really needed.

1.6 Names

- **Choose descriptive names** "names in software are 90% of what makes software readable".
- **Unambiguous names** "choose names that make the workings of a function or variable unambiguous".
- **Names should describe side effects**, *e.g.* a method `getOos()` will make an `ObjectOuputStream` if one does not already exist, so should be called `createOrReturnOos()`

1.7 General

- **Obscured intent** – make the code as expressive as possible such that its intention is clear from a first reading.
- **Put conditional statements into a method to make their intention and effect clear**, *e.g.*

  BAD `if (guessedWord.length()< shortestLength)`
  GOOD `if(guessedWordIsTooShort(guessedWord))`

1.8 The assignment

Write a class called *NumberGuessingGame.* The game should implement the instructions given in exercise 8.9.14 in volume 1 of the subject guide. In addition to the instructions in the guide:

- Each time the game is played the class should choose a random `int` for the user to guess from a range of numbers.
- The range of numbers to randomly choose from should be at least 0–5,000, but you can choose a higher upper bound if you wish.
- The subject guide provides an answer to the exercise, which you should ignore. You will get no credit if you hand in the answer given in the guide.

Answer the following questions:

1. You are not asked to make any changes to the *WordScramble* class. Test the game as it is, and write a comment at the top of the *WordScramble* class explaining how the game could be improved. Please do not write more than 2 paragraphs, with a paragraph defined as a block of text with no more than 8 sentences. [6 marks]

2. Write the *NumberGuessingGame* as described above. When writing your class, answer the following questions:

a. The class should have the following 4 methods:

```
askUserToGuessRandomNumber();
readAndReturnUserGuessAtRandomNumber();
getAndReturnFeedbackMessageForUser();
showFeedbackMessageToUser();
```

Note that any parameters in the methods listed above have been omitted, so it is up to you to decide what parameters each method should have, if any. [24 marks]

b. Your class should have a method that loops until the user guesses the number. [6 marks]

c. Write any other methods that you need in order to ensure that you follow the advice about methods given by Martin and summarised in 1.4 above. In particular, make sure to write a method that, when called, plays the game without the user needing to call any further methods. [12 marks]

d. Format your code so that it is readable. As Martin writes 'choose a set of simple rules [and] consistently apply those rules'. Note the example in the appendix, *GradeCalculator*, lays out the code such that the start and end of the class is clear, the start and end of each method is clear, and the block of code with each *if* statement is clear. In addition, the code in the classes that you have been given has been formatted carefully. [12 marks]

e. Write a main method and use it to test your class. You should have written the class such that you need to call one method only in order to run the game. Hence your main method should have one statement only in it. [12 marks]

f. Change the *GamesArcade* class such that if the user chooses menu option 1 the *NumberGuessingGame* that you have written is run. [12 marks]

g. Make sure that you apply the rules given in sections 1.4, 1.5, 1.6 and 1.7 above, as far as possible. [12 marks]

A note on class variables

The classes you have been given use class variables. Class variables are static variables that (among other things) can be accessed and updated by any static method in the class. They are declared outside of a method. For example, the *GamesArcade* class has the `boolean` class variable *keepPlaying*. Note that *keepPlaying* is declared and initialised inside the class, but outside of all of the

methods. Class variables can be accessed by static methods within the class without needing to be included in the method's parameter list.

Reading (all from volume 1 of the subject guide)

- Sections 2.7.1 and 2.7.2 (comments, and how to write a comment over more than one line).
- Sections 4.3–4.5 (variables)
- Sections 5.3–5.6 (methods and random numbers) and section 5.8 (method signatures)
- Sections 6.3 and 6.4 (user input using the Scanner class)
- Chapter 7 *Boolean expressions and conditional statements*
- Chapter 8 *Simple loops*
- Sections 9.1–9.3.3 (calling methods, void and typed methods)
- Chapter 12 *Defining your own methods* sections 12.3–12.10 and 12.1212.13 (non-void or *typed* methods).

Deliverables

Electronic copy of:

- *GamesArcade.java*
- *NumberGuessingGame.java*
- *WordScramble.java*

Appendix

A simple example of renaming methods and variables for greater readability.

Original

```
public class Calculator{

     public static void calc(int x){
          if (x >= 70){
               System.out.println("grade = A");
               return;
          }
          if (x >= 60){
               System.out.println("grade = B");
               return;
          }
          if (x >= 50){
               System.out.println("grade = C");
               return;
          }
          if (x >= 40){
               System.out.println("grade = D");
               return;
          }
          if (x<40) System.out.println("grade = F");
     }

     public static void main(String[] args) {
          calc(90);
          calc(53);
          calc(30);
     }
}
```

Renamed

```
public class GradeCalculator {

     public static void calculateAndPrintGrade(int finalMark){
          if (finalMark >= 70){
               System.out.println("grade = A");
               return;
          }
          if (finalMark >= 60){
               System.out.println("grade = B");
               return;
          }
          if (finalMark >= 50){
               System.out.println("grade = C");
               return;
          }
          if (finalMark >= 40){
               System.out.println("grade = D");
               return;
          }
          if (finalMark<40) System.out.println("grade = F");
     }

     public static void main(String[] args) {
          calculateAndPrintGrade(90);
          calculateAndPrintGrade(53);
          calculateAndPrintGrade(30);
     }
}
```

Marks for CO1109 coursework assignment 1

The marks for each section of coursework assignment 1 are clearly displayed against each question and add up to 96. There are another 4 marks available for giving in uncompressed .java files, for giving in files that are not contained in a directory, and for giving in files with the correct names. This amounts to 100 marks altogether. There are another 100 marks available from coursework assignment 2.

Total marks for questions 1 and 2	[96 marks]
Mark for giving in uncompressed files	[1 mark]
Mark for giving in standalone files; namely, files **not** enclosed in a directory	[1 mark]
Marks for giving classes the correct names	[2 marks]
Total marks for coursework assignment 1	**[100 marks]**

[END OF COURSEWORK ASSIGNMENT 1]