University of London
Computing and Information Systems/Creative Computing
CO2220 Graphical object-oriented and internet programming in Java
Coursework assignment 1 2018–19

Introduction

This is coursework assignment 1 (of 2 coursework assignments total) for 2018–19. Part A looks at the object-oriented programming paradigm: classes, constructors; inheritance; instance methods plus getting your classes to work together to produce a desired result. Part B asks that you demonstrate an understanding of simple graphics, inner classes, events and the *ActionListener* interface.

IMPORTANT NOTE: Please use at least Java 8 to compile and test your programs. Later versions of Java are also fine.

Electronic files you should have:

Part A

- *Hangman.java*
- *AbstractRandomWordGame.java*
- *WordGamesArcade.java*
- *gamedictionary.txt*

Part B

- *AnimatedGUI.java*

What you should submit: very important

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 submit **electronic versions** of your Java files since you cannot gain any marks without submitting the .**java** files asked for. Class files are **not** needed, and any student submitting only a class file will **not** receive any marks for that part of the coursework assignment. **Please be careful about what you upload as you could fail if you submit incorrectly**.

- Please only hand in the .java files asked for, and not any additional files.

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

There is one mark allocated for handing in uncompressed files. Therefore, students who hand in zipped or .tar files or any other form of compressed files can only score 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 a directory 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 *WordSleuth.java*) ***exactly*** as you have been asked to name them.

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

- *JSmith_WordGamesArcade.java*
- *cwk1-WordGamesArcade.java*
- *JSmith-109-assignment1-WordGamesArcade.java*
- *WordGamesArcade .java*

Using a file name that differs from the class name leads to a compilation error caused by a file name / class name mismatch.

The examiners intend to compile and run your Java programmes, for this reason programmes that do not compile will not receive any marks.

You are asked to give your classes certain names, please follow these instructions carefully. If your file does not compile ***for any reason*** (except file name / class name mismatch) you will receive no marks for that part of the assignment. In particular, files that contain Java classes that cannot be compiled because they are the wrong type (*e.g.* PDFs) will not be given any marks.

CO2220 Coursework assignment 1

1.0 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 in 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 focussing 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 classes. However Martin himself notes that names can always be improved, and that 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 remember the following rules outlined by Martin:

1.1 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."

1.2 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.”

1.3 Comments

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

1.4 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**. For example, a method `getOos()` will make an `ObjectOuputStream` if one does not already exist, so should be called `createOrReturnOos()`

1.5 General

- **Obscured intent.** Make the code as expressive as possible so that its intention is clear from a first reading.
- **Put conditional statements into a method to make their intention and effect clear.** For example:

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

  GOOD `if(guessedWordIsTooShort(guessedWord))`

2.0 The list interface

You will note that in the *Hangman* class, the *guesses* variable is of type `List`, but is implemented as an `ArrayList` in the *initialiseGameState()* method.

`List` is an interface: http://docs.oracle.com/javase/7/docs/api/java/util/List.html

Explanation below, modified from Effective Java, 2nd edition by Joshua Bloch:

> You should use interfaces rather than classes as parameter types. More generally, you should favour the use of interfaces rather than classes to refer to objects. If appropriate interface types exist, then parameters, return values, variables, and fields should all be declared using interface types.
>
> Get in the habit of typing this:

```
// Good - uses interface as type
List<Subscriber> subscribers = new ArrayList<Subscriber>();
```

> rather than this:

```
// Bad - uses class as type!
ArrayList<Subscriber> subscribers = new
ArrayList<Subscriber>();
```

If you get into the habit of using interfaces as types, your program will be much more flexible. If you decide that you want to switch implementations, all you have to do is change the class name.

For example, the first declaration could be changed to read:
`List<Subscriber> subscribers = new LinkedList<Subscriber>();`
and all of the surrounding code would continue to work. The surrounding code was unaware of the old implementation type, so it would be oblivious to the change.

Part A

Please complete the following tasks:

1. " Compile the *WordGamesArcade* class and use it to play the *Hangman* game. Read the code for the game and for the *WordGamesArcade*. Write a comment at the top of *Hangman* explaining what improvements or enhancements you would make to the game. Improvements could be, for example, to the names of variables and methods, as readability can always be improved, while enhancements could be to the playability of the game. "

[6 marks]

2. " Read the *AbstractRandomWordGame* class, and note the four comments with the *loadWords()* method. Make any changes you think are needed based on your reading of the code, the rules given by Martin, and the comments. "

   You need to understand the comments in order to act on them. Once you have done so there should be no comments in the method, as dealing with the comments should not introduce the necessity to write new comments. However, you may, if you wish, write a comment for the examiner explaining what you have done and why. Start your comment with //COMMENT or with /*COMMENT.

   In answering this question, note that:

   - TODO comments are notes written by the developer, and are always deleted once dealt with;
   - Take note of the rules from Martin given in 1.1 to 1.5, and that Martin believes that comments should be used sparingly, and that the names of variables and methods should tell you their purpose without the need for a comment to explain it;
   - Private methods cannot be overridden by subclasses, but protected ones can.

[6 marks]

3. " Write a class called *WordSleuth.java* that, like Hangman, inherits from the *AbstractRandomWordGame* class. The rules of the *Word Sleuth* game are as follows: "

   a. "The game chooses a word at random, from the *gamedictionary.txt* file. The game works out the player's possible top score by doubling the length of the word. The game works out the player's total number of guesses by

(length of the word)/2 +1. The result of the division should be rounded up. Hence for an 8 letter word the player would have 5 guesses, and for a 9 letter word the player would have 6 guesses. The game tells the player three things:

- the letter that the word starts with
- how many letters the word has
- how many guesses the player has

b. The player is invited to guess the word.

c. "If the player guesses right, they are told their score, and the game ends. If they guess wrong then one point is deducted from the player's possible score.

d. If the player has guessed wrong then they are given a letter from the word, randomly chosen. The player is not told where the letter occurs in the word. The player is invited to guess the word again.

e. "Steps 3 and 4 are repeated until (1) the game ends as the player has guessed the word (go to step 7), or (2) until the player has been given half the letters in the word, rounded up, ie for words with an odd length the player gets half the number of chars in the word + 0.5. For example 3 letters from 5 letter words, and 4 letters from 7 letter words.

f. " If the player has all the letters they can have, and has not guessed correctly, then the game deducts 2 marks from their possible score, and rearranges the letters they have been given into the order they appear in the word. The letters from the word are displayed to the player without spaces. The player is told that this is the order of letters in the word, and that this is their final guess.

g. At the end of the game the player receives an appropriate message, telling them whether they have won or lost, and their score, if any. If the player has lost, then the game tells them what the word was that they failed to guess.

[18 marks]

4. " When choosing random letters to show to the user, make sure that a letter in a particular position in the word cannot be chosen twice. Letters are distinct in their placement in the word, so in a word such as *stellar,* the 'l' in the third place is distinct from the 'l' in the fourth place, counting from zero. Hence in the case of stellar it would be fine to give the player 'l' twice, but giving them 'a' twice would be an error. [6 marks]

5. " Change the *WordGamesArcade* class such that the Word Sleuth game is added to the menu as the second option. When the user picks option 2, the game starts to play.

[6 marks]

6. " When adding new methods and variables, or renaming existing ones, please remember to make your names as meaningful as possible, following the advice from *Clean Code.* [6 marks]

Reading for part A
The following chapters and pages of Volume 1 of the subject guide, and associated *Head First Java* readings:

- Chapter 2, sections 2.1, 2.2 and 2.3 (inheritance and instance variables)
- Chapter 3 (Object programming)
- Chapter 5, sections 5.1 – 5.6 (Object behaviour)
- Chapter 7, sections 7.1 – 7.5 (The Java library, the `ArrayList`, `boolean` expressions, packages and imports)
- Chapter 8, sections 8.1 – 8.4.2 (Inheritance)
- Chapter 9, sections 9.1 – 9.4 (Abstraction)
- Section 10.3, 10.4 and 10.5 (constructors, including the *this* keyword and superclass constructors).

Deliverables for part A
Please put your name and student number as a comment at the top of your Java files. Please only hand in the files asked for.

Please submit an electronic copy of the following:
- *WordSleuth.java*
- *Hangman.java* (with answer to question 1)
- *AbstractRandomWordGame.java* (with answer to question 2)
- *WordGamesArcade.java* (with answer to question 5)

Part B

Compile and run the *AnimatedGUI* class.

You should see a red square and a blue circle in a `JFrame`, together with a button. The red square moves across the frame from left to right and then exits. The blue circle grows until blue completely fills the draw panel. The button can be pressed at any time to stop the animation. Once the animation has stopped, further clicks on the button have no effect.

Your task is to separate the animations, such that the blue circle can be stopped and started from growing with one button, and the red square can be stopped and started with another. Hence each animation can be stopped and started independently of the other.

In the following questions you are asked to use inner classes. Note that inner classes have unrestricted access to their containing classes instance variables, including private variables.

Please complete the following tasks:

1. ' Separate the animation so that each component has its own button that starts or stops the animation independently of the other component. Each animation can be stopped and re-started as many times as the user wishes. To do this you will need to do several things including the following:

a.
- Change the text on the button
- Add a second button with appropriate text
- Add new instance variables (these should be initialised in the constructor)
- Use inner classes to implement `ActionListener`. You should use a separate inner class to listen to each button.

[18 marks]

b. Make any other necessary changes such that the animation of the red square is independent of the animation of the blue circle. This means that one button can be clicked to stop the animation of the red square, while the blue circle continues growing. The other button can be clicked to stop the blue circle growing, while the red square continues to move.

[6 marks]

c. Make any changes necessary to ensure that each animation can be stopped and restarted as many times as the user wishes.

[6 marks]

2.

a. Change the animation of the square so that once its leading edge hits the right edge of the frame the animation reverses and it moves back across the frame. Once its leading edge touches the left edge of the frame it once again rebounds and starts moving back across the frame again. This continues indefinitely.

[6 marks]

b. Change the animation of the circle so that once the frame is entirely filled with the circle, the animation reverses and the circle shrinks back to its original size. Once it has reached its original size it starts growing again, until it once again fills the frame. This continues indefinitely.

[6 marks]

3. ' When adding new methods and variables, or renaming existing ones, please remember to make your names as meaningful as possible, following the advice from *Clean Code.*

[6 marks]

Reading for part B

- Sections 11.1 – 11.6 of volume 1 of the subject guide
- Sections 12.1 – 12.3 of volume one the subject guide.
- Chapter 12 of *Head First Java*, pages 353-385 only

Deliverable for part B

- An electronic copy of your revised program: *AnimatedGUI.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 CO2220 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 two marks available for submitting uncompressed files and for submitting files that are not contained in a directory. There are two marks for submitting files with the correct names. This amounts to 100 marks altogether. There are another 100 marks available from coursework assignment 2.

Total marks for part A	[48 marks]
Total marks for part B	[48 marks]
Mark for submitting uncompressed files	[1 mark]
Mark for submitting standalone files; namely, files **not** enclosed in a directory	[1 mark]
Mark for submitting files with the correct names	[2 marks]
Total marks for coursework assignment 1	**[100 marks]**

[END OF COURSEWORK ASSIGNMENT 1]