University of London International Programmes
Computing and Information Systems/Creative Computing
CO2220 Graphical object-oriented and internet programming in Java
Coursework assignment 1 2015–16

Introduction

This is Coursework assignment 1 of two for 2015–16. Parts a. and b. of Coursework assignment 1 ask that you demonstrate an understanding of simple graphics and animation, inner classes and the *ActionListener* interface. Part c. looks at the object-oriented programming paradigm, including inheritance, interfaces, classes (including abstract classes) and methods.

IMPORTANT NOTE: In order to complete part (c) you will need to be using Java 8. Using previous versions of Java will mean that the *Main* class will not compile.

Electronic files you should have:

Part a.

- *Animation_Eg1.java*
- *Animation_Eg2.java*

Part b.

- *MovingCircleGUI.java*

Part c.

<u>Book classes</u>

- *AbstractBook.java*
- *AudioBook.java*
- *Book.java*
- *Ebook.java*

<u>Book field classes and interfaces</u>

- *BookID.java*
- *ASIN.java*
- *ISBN.java*
- *BookLength.java*

<u>Parser classes</u>

- *AudioBookCSVParser.java*
- *BookCSVParser.java*
- *EbookCSVParser.java*

<u>Other classes and files</u>

- *BookAuthorComparator.java*
- *Main.java*
- *audiobooks.csv*
- *books.csv*
- *ebooks.csv*

Comments
You may find it useful to put a comment at the start of a method to explain what the method should do, or you may find it useful to comment particularly complicated code, but please do not comment every line of your code as this does not usually make your code any more readable and usually less so. Code in a high-level language such as Java is designed to be readable, reducing the need for comments; in particular, the examiner does not want to see comments such as the following:

```
frame.setSize(500,500);  // sets the size of the JFrame
frame.setVisible(true);  // makes the JFrame visible
```

Deliverables – very important
There is one mark 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 49/50 marks.

There is one mark for handing in just the .java files asked for without wrapping them up in a directory structure – that is, students who upload their directories can only achieve 49/50 marks.

Please make sure that you give in electronic versions of your .java files. Class files are **not** needed. Any student uploading the class file rather than the java file will not gain any marks for that part of the coursework assignment. Every year at least one student hands in only their class files, most likely by mistake, and sadly it is then not possible to give that student any marks for the relevant part of the coursework assignment, so please be careful about what you upload.

A pdf or Word document is required for this coursework assignment. Please do **NOT** include your .java files in the pdf or Word document. Any student uploading a pdf or other document and no .java files will receive zero marks for those parts of the coursework assignment that require .java files to be given in.

Please note carefully what you are asked to give in at the end of each part of the coursework assignment. Please note that this supersedes the generic hand in instructions given by the university. Full marks can only be achieved by handing in all the files asked for, including the .java files, which must be uncompressed and individual files – that is, **NOT** contained within a directory structure.

Coursework assignment 1

PART A

1.	Compile and run the *Animation_Eg1.java* program and the *Animation_Eg2.java* program. You should note that despite being almost the same, one program shows an animation of an oval whose colour constantly changes randomly, and the other does not work. Can you explain, in one paragraph, why the *Animation_Eg2* class does not work? *Please note a paragraph has at most eight sentences.*	[6 marks]

Reading for part a.
http://www.oracle.com/technetwork/java/painting-140037.html

Deliverable for part a.

- A pdf or Word or OpenOffice document with the answer to the above question, named with your ID number (ie, for a student whose ID number was 00000000, the file should be: 00000000.pdf; or 00000000.doc; or 00000000.odt). Please put your name and ID number at the top of your document, before your answer to the question.

PART B

Compile and run the *MovingCircleGUI.java* file. You should note that it uses an inner class to place a draw panel onto a `JFrame`. The draw panel displays an orange circle. You may change the colour of the circle if you wish. See: http://docs.oracle.com/javase/7/docs/api/java/awt/Color.html for the static Java Colors available, and methods to make your own if you prefer.

Make the following changes to the *MovingCircleGUI* class:

1.	Add a `JButton` to the SOUTH region of the `JFrame` using *BorderLayout*	[3 marks]
2.	Write an inner class called A*nimateCircleListener* to	[3 marks]

	implement the `ActionListener` interface, and to listen to the `JButton` in the SOUTH region. *NOTE: you can only get full credit for this part of the question by writing an inner class.*	
3.	Set the text on the button to read “Click me to start the animation”.	[1 mark]
4.	Make the necessary changes to the *MovingCircleGUI* class such that when the button is clicked the circle is animated. The animation continues until the user clicks again on the button to stop the animation. The circle should stay the same size and colour. You may choose ***one*** of the following ways to animate the circle: • Randomly placed circles keep appearing until the user clicks the button to stop the animation. • The circle moves across the draw panel until it touches the opposite side, then it moves back until it again touches the side it started at and keeps going across and back again until stopped by the user. • The circle moves down the draw panel until it touches the bottom edge, then moves back up until it touches the top edge, and keeps going up and down until stopped by the user. • The circle starts moving on a random trajectory. When it hits a side it rebounds on a new trajectory until it hits another edge and rebounds again. The circle continues to bounce around inside the draw panel until stopped by the user.	[4 marks]

5. Make sure that the area of each circle is always drawn completely inside the draw panel. In order to complete this task you should use the methods *getWidth()* and *getHeight()* to get the height and width of the draw panel. It should be possible for the circle to touch the edges of the draw panel and/or the button, without any part of it disappearing behind an edge.	[4 marks]
6. Use *Thread.sleep* to slow the animation to a pace that seems reasonable to you.	[3 marks]
7. The user should be able to click the button to stop and start the animation as many times as they wish to.	[3 marks]
8. The text on the button should change each time the user clicks the button, reading either “Click me to start the animation” or “Click me to stop the animation” as appropriate.	[3 marks]

Reading for part b.
- Chapter 12 of *Head First Java*, pages 353–385.
- Chapter 11 of Volume 1 of the subject guide, pages 115–117;
- Chapter 12 of Volume 1 of the subject guide, pages 131 and 132.

Deliverable for part b.
- Electronic file of your program: *MovingCircleGUI.java*

PART C

Notes on the *Main* class and associated files.

Java version
In order to complete part (c) you will need to be using Java 8. Using previous versions of Java will mean that the *Main* class will not compile.

List interface
You will note that in the files you have been given, `ArrayLists` are declared to be of type `List`, as are methods that return an `ArrayList`. `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
https://cebulko.com/Programming-Resources/Effective%20Java%20-%202nd%20Edition.pdf
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 in the constructor.

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 c. questions
Compile and run the *Main.java* file. You will find that it outputs the following:

```
LIST OF BOOKS BY AUTHOR

Exception in thread "main" java.lang.NullPointerException
```

This is because of the *parseBook()* method in the *BookCSVParser* class. This method is called by the *parseBookChart()* method to turn a line read in from the *books.csv* file into a *Book* object. The method returns `null`, meaning that the `ArrayList` created and returned to the *Main* class by the *parseBookChart()* method contains objects that all have the value `null`, hence the exception.

1. Write the body of the *parseBook()* method in the *BookCSVParser* class, such that the *parseBook()* method returns	[3 marks]

	a *Book* object with all of its fields (instance variables) given a value.	
2.	Once you have written the *parseBook()* method, re-compile *BookCSVParser* and then compile and run *Main.java*. You should see a list of books, sorted by the author's first name. The sorting is being done using the *BookAuthorComparator* class. Amend this class (do not change its name) so that it can be used to sort not just objects of the *Book* class, but also *EBook* and *AudioBook* objects.	[3 marks]
3.	Write the *parseBook()* method for the *EBookCSVParser* class.	[3 marks]
4.	Write the *parseBook()* method for the *AudioCSVParser* class.	[3 marks]
5.	Remove the comment delimiters around the second and third blocks of code in the Main.java file. Compile and run the *Main* program again and this time you should see a list of *Books*, followed by a list of *EBooks*, then *AudioBooks*, each list sorted by author name. You should note that the *narrator* field is missing from the list of *AudioBooks*. Write a *toString()* method for the *AudioBook* class such that the *narrator* field is included in every object in the list of *AudioBooks*.	[3 marks]
6.	Write a *BookSalesComparator* class that can be used to sort *Books*, *EBooks* and *AudioBooks* into the order given by their *sales* field; that is, into bestselling order, with the book with the highest *sales* figure	[3 marks]

starting the list, and the book with the lowest *sales* ending it. Test your new *Comparator* by removing the comment delimiters around the final three blocks of code in the *Main* class, recompiling and running the class.	

Reading for part c.
http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html
For more on the Comparator interface, see also the 2014–15 Computing CO2220 Coursework report.

http://docs.oracle.com/javase/7/docs/api/java/lang/String.html
http://www.w3schools.com/jsref/jsref_split.asp
See the above for String methods including String.split()

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

- Chapter 3
- Chapter 5, pages 39–41
- Chapter 7, pages 63–65
- Chapter 8, pages 73–74
- Chapter 9, pages 95–100.

Deliverables for part c.
Electronic copy of the following six classes:

- *AudioBook.java (with added toString() method)*

- <u>Parser classes (with completed *parseBook()* methods)</u>
- *AudioBookCSVParser.java*
- *BookCSVParser.java*
- *EbookCSVParser.java*

- <u>Comparators</u>
- *BookAuthorComparator.java (revised)*
- *BookSalesComparator.java (new)*

MARKS FOR COURSEWORK ASSIGNMENT 1

The marks for each section of the coursework assignment are clearly displayed against each question and add up to 50. There are another 50 marks available for CO2220 Coursework assignment 2.

The marks breakdown given by each question is there to help you, so that you know how many marks you can potentially achieve even if you cannot successfully implement or answer every sub-question.

Mark for handing in uncompressed files	1 mark
Mark for handing in files **not** enclosed in a directory structure	1 mark
Total marks for part a.	6 marks
Total marks for part b.	24 marks
Total marks for part c.	18 marks
Total marks for Coursework assignment 1	**[50 marks]**

[END OF COURSEWORK ASSIGNMENT 1]