Coursework commentaries 2015–2016

CO2220 Graphical object-oriented and internet programming in Java

Coursework assignment 1

General remarks

Please note the following model answers are provided with this report:

Part (b): MovingCirclesGUI.java

Part (c):

The following files are as given out with the assignment:

- AbstractBook.java
- ASIN.java
- Book.java
- BookID.java
- BookLength.java
- EBook.java
- ISBN.java
- Main.java
- audiobooks.csv
- books.csv
- ebooks.csv

The following files have been completed with the answers to the assignment:

- AudioBook.java (with added toString() method)
- AudioBookCSVParser.java (with completed parseBook(String) method)
- BookCSVParser.java (with completed parseBook(String) method)
- EBookCSVParser.java (with completed parseBook(String) method)
- BookAuthorComparator.java (revised from that given with the assignment)
- BookSalesComparator.java (part of the answer, not given with the assignment)

Comments on specific questions

Part a: Identifying a problem with an animation

The first part of the coursework assignment asked students to explain why a program with a simple animation did not work, despite being almost identical to another program that did work. Most students successfully named the error as being in the

```
while (true) repaint();
```

statement of the `paintComponent(Graphics)` method in the `RandomColDrawPanel` inner class. However, many students could not

explain why this caused the program to fail. In fact this was so common that just over half of students achieved full credit for this part of the coursework assignment.

Firstly, when `repaint()` is placed in `actionPerformed(ActionEvent)` the system will update when it detects a change. If the calculations for the position of the circle are in a loop, then there will be multiple changes in the position of the circle, and after each one the method will call `repaint()`. If the call to `repaint()` is in the loop, this would represent a misunderstanding of how the system works. However, if the `repaint()` statement was not in `actionPerformed(ActionEvent)` but instead in the `go()` method, then it would need to be in a loop. For example:

```
while (true) drawPanel.repaint();
```

The second reason is very well described in this excellent sample answer:

> Most swing-handling code runs on the event-dispatch thread (EDT) which is a special thread managed by swing, such that the programmer does not need to create it. In the AnimationEg_2.java program, the paintComponent(Graphics g) method is such an event, and contains within it another such event, the call to repaint(). When repaint() is invoked on a component, an asynchronous call is created to action this request later on the EDT [1] – which presumes that the JVM will dedicate processing resources to this thread. Prior to the invocation of the repaint() method, there is an infinite loop – the while (true) construct – which effectively monopolises the event queue, preventing the invocation of repaint() from ever being executed by the EDT. This can be illustrated more clearly by replacing the 'while (true)' construct with 'for (int i=0; i<500000000; i++)': it then takes over 30 seconds before the first invocation of repaint() is processed, i.e. the loop must 'cycle' in its entirety before the EDT can process the next item in the event queue and the first repaint() invocation can be executed – since this never happens with the 'while (true)' construct, the EDT is permanently executing the infinite loop and is unable to process any more items in the event queue.

Part b: MovingCircleGUI.java

Part (b) asked students to develop a simple GUI, where pressing a `JButton` stopped and started an animation. This was done well, with most students achieving most or all of the marks available.

Question 1: Add a `JButton` to the Frame using BorderLayout

Question 3: Set the text on the `JButton` to "Click me to start the animation"

Both these questions were very simple to do, and could be combined into one statement as follows:

```
animateCircleButton = new JButton("Click me to start the
animation");
```

Every student who attempted both questions received full credit for them.

Question 2: Write an inner class to implement the `ActionListener` interface

To answer this question, students were expected to both write the inner class and add it to the `JButton`, for example, if the `JButton` was called `animateCircleButton`:

```
animateCircleButton.addActionListener(new
AnimateCircleListener());
```

Almost every student who attempted this question received full marks; however, a very small number lost credit by directly implementing the `ActionListener` interface.

Question 4: The animation

Students were asked to animate the orange circle that the `CircleDrawPanel` placed on the `JFrame`, and could choose from a number of ways to do this. Most popular was making the circle move either across the frame and back again, or down the frame and back up again. Less popular was making the circle start moving with a random trajectory, then rebounding whenever it hit a side, and the least popular was making randomly placed circles appear. Of those attempting this question, 90 per cent received full credit. Reasons for losing credit included:

- the animation started without a button click from the user
- the animation was not stopped by a button click
- the animation was started by the user, but stopped by itself (i.e. time limited by a loop).

Question 5: Keeping the circle visible in the `drawPanel`

Quite often the circle moved past the edge of the `JFrame`, meaning that a part of it, usually a small part, would be hidden by the edge. There were two reasons for this. First, some students used `frame.getWidth()` and `frame.getHeight()`; that is, the width and height of the `JFrame` which did not give completely accurate results. Using `this.getHeight()` or `this.getWidth()` in the `paintComponent(Graphics)` method of the `CircleDrawPanel` would invoke the getters for the drawPanel rather than the `JFrame`, and give more accurate results, since it was the drawPanel that the circle was placed on. The second reason was that some students attempted to work out the position of the circle numerically without using the getters at all; that is, using 500 for the width and 500 for the height in their calculations, because this was the size of the `JFrame`. This was a mistake because the circle is placed on the drawPanel, which itself is placed on the `JFrame`. The drawPanel is not exactly the same size as the `JFrame`. Some students compensated for this by reducing the width and height in their calculations to 490 x 490 say, but using the getters would have given completely accurate results, since Java knows how much space is allotted to the drawPanel without having to guess.

These mistakes were so common that only about two-thirds of students received full credit for this question.

Question 6: Use `Thread.sleep` to slow the animation

This was again a very simple thing to do, and almost all students received full credit. The most common reason for losing marks was making no attempt at the question. Students could use other means to slow the animation and receive full credit.

Question 7: The user can click the button to stop and start the animation repeatedly

Question 8: The text on the button changes appropriately with each button click

Most students achieved full credit for both questions. They could both be answered simply in the `actionPerformed(ActionEvent)` method as follows:

```
class AnimateCircleListener implements ActionListener{
   public void actionPerformed (ActionEvent e){
      start = !start;
      //a boolean instance variable initialised to 'false'

      if (start) animateCircleButton.setText("Click me to stop
      the animation");
      if (!start) animateCircleButton.setText("Click me to start
      the animation");
      drawPanel.repaint();
   }
}
```

Some students wrote `actionPerformed(ActionEvent)` methods that were much more complicated than necessary. Learning to keep your code as simple as possible will help you in the future. Also when the `actionPerformed(ActionEvent)` method is: (1) listening to one component only; and (2) invoked whenever the button is clicked, it really is not necessary to query the source of the event. In fact, it should not be necessary to query the event source even with more than one component to listen to, as the solution would be to write another inner class to listen to the other component.

Part c: A program to sort and display lists of books

Questions 1, 3 and 4: The `parseBook()` methods

The first task was to write the body of the `parseBook(String)` method in the `BookCSVParser` class given to students. The method given to students was as follows:

```
private static Book parseBook(String line) {
   return null;
}
```

The method needed to be completed so that when given a String comprising a line of text read in from the `books.csv` file (also given with the coursework assignment), it would return a `Book` object with its fields composed of the data from the String (the `Book` and other associated classes were also given with the coursework). This could be answered with:

```
private static Book parseBook(String line) {
   String[] details = line.split(";\\s*");

   String title = details[0].trim();
   String auth = details[1].trim();
   String pub = details[2].trim();
   String genre = details[3].trim();
   String isbn = details[4].trim();
   int year = Integer.parseInt(details[5].trim());
   int pages = Integer.parseInt(details[6].trim());

   Book b = new Book(title, auth, new ISBN(isbn), pub, genre,
      year, new BookLength(pages,"pages"));
   int sales = Integer.parseInt(details[7].trim());
   b.setSales(sales);

   return b;
}
```

Most students wrote something similar to the above, starting with splitting the line using `String.split()` and using `String.trim()` to deal with any trailing white spaces; almost three-quarters of students attempting the question received full marks for this. The most common mistake was forgetting to give a value to the `sales` field as the question stated that all of the fields of the `Book` object must have a value. The `sales` field was a field of the parent class of the `Book` object, `AbstractBook`. Since `Book` inherited from `AbstractBook` it inherited the `sales` field and therefore, this should have been given a value in the `parseBook(String)` method. The `sales` field was not included in the constructor of the `Book` class, and this was probably the reason that many students overlooked it, thinking that the method had done its job once it had made a `Book` object. While it is easy to imagine making this mistake on a first attempt at the method, failing to realise the error and correct it suggests poor testing on the part of these students. The developer had written a setter for the `sales` instance variable as it is the only value that can change, and this (`setSales(int)`) should have been used to give the `sales` field a value, as has been done in the above `parseBook(String)` method.

In fact, the `sales` field should have been included in the `AbstractBook` constructor. Ideally the class would have had two constructors. The first would not include the `sales` field in the parameters, but would set it to zero (since a book may not have any sales). The second constructor would include the `sales` variable in the parameter list, allowing the user to give it a value. The less than perfect `AbstractBook` class reflects that in the real world developers often have to work on classes that they did not write and cannot change.

Another common mistake was forgetting to trim some or all of the `Strings`, resulting in problems with sorting because of the white spaces in the `String`; or compilation errors when attempting to parse a `String` with trailing white spaces to an `int`.

The `parseBook(String)` methods for the `EBookCSVParser` and `AudioBookCSVParser` classes were written in a very similar way to the method for the `BookCSVParser` class, except that the `AudioBook` class had one additional field, narrator, which needed to be filled, and also had an `ASIN` rather than an `ISBN`. Similarly, the `EBook` class had an `ASIN` instead of an `ISBN`. The same common mistakes were seen as above, plus many students set the wrong value for the `String` part of the `BookLength` variable. Either the book length was given in "pages" or it was not given a value at all. In fact, the length of `EBooks` is given in the number of words and `AudioBook` length should have been given in minutes. Students could have deduced this information from the headings at the top of the `ebooks.csv` and the `audiobooks.csv` files.

Question 2: The `BookAuthorComparator` class

The task was to change the class so that it could sort not only `Book` objects by author, but `eBook` and `audioBook` objects too. The change that was needed was very simple, as most students attempting the question realised. It was only necessary to change the object in the angled brackets from `Book`, to the parent class of `Book`, `EBook` and `AudioBook`: `AbstractBook`. Once done, due to polymorphism, the class would act on the `AbstractBook` class and all of its child classes.

```
import java.util.Comparator;
//change Book to AbstractBook in the angled brackets below.
public class BookAuthorComparator implements Comparator<Book>{
   @Override
   public int compare(Book o1, Book o2) {
      String n1 = o1.getAuthor();
      String n2 = o2.getAuthor();

      return n1.compareTo(n2);
   }
}
```

Question 5: A `toString()` method for the AudioBook class

All students who attempted this question got some credit, and a large percentage got full marks. By far the most common mistake was failing to include the `narrator` field in the method. Some students also forgot the `sales` field and/or the `asin` too.

Question 6: The `BookSalesComparator` class

While there were students who could not make their comparator work, a large proportion gained full marks. Some students lost credit by sorting on the wrong field (e.g. `author` rather than `sales`) and others lost credit by attempting to sort the `sales` field as `Strings`; for example:

```
import java.util.Comparator;
public class BookSalesComparator2 implements
Comparator<AbstractBook>{
   public int compare(AbstractBook o1, AbstractBook o2) {

      int s1 = o1.getSales();
      int s2 = o2.getSales();
      String t1=Integer.toString(s1);
      String t2=Integer.toString(s2);
      return t1.compareTo(t2);
   }
}
```

Other students wrote a `Comparator` that only worked on the `Book` class, or wrote one `Comparator` for `Book`, another for `AudioBook` and a third for `EBook`. These students had failed to appreciate that as `sales` was a field of the parent class, `AbstractBook`, a `Comparator` written to work on `AbstractBook` would work on each of its child classes.

Conclusion

Part (c) was quite challenging and those students who achieved a good mark did well.

Coursework assignment 2

General remarks

Please note the following model answers are provided with this report:

Part (a):

- Grocery.java (with toString() method)
- Fruit.java
- FruitUtils.java
- FruitTest.java
- fruits.csv

Part (b):

The following files are as given out with the assignment:

- Book.java
- BookClient.java
- BookFileParser.java
- BookServer.java
- ServerUtils.java
- books.ser

The following files have been completed with the answers to the assignment:

- BookServerEngine.java (includes added search() method)
- History.java
- ClientGUI.java (with completed HistoryListener class)

Comments on specific questions

Part a

Question 1: `toString()` for Grocery

In question 1 students were asked to write a `toString()` method for the `Grocery` class, and this was done well by all. Almost all students successfully attempted the task, with a very small minority losing marks by writing a `toString()` method that did not use `String.format()` as the question required. The following is an example of a `toString()` method that would have gained full credit:

```
public String toString() {
   return String.format("%-10s %s%n%-10s %s%n", "Name:", name,
      "Producer:", producer);
}
```

Question 2: The Fruit class

Again almost all students gained full credit for this question. There was one mistake made by a small minority of students, which was copying the

```
   private static final long serialVersionUID = 1L;
```

variable from the Grocery class into the Fruit class, despite being asked to write a class with only the one instance variable called `pricePerKilo`, a `double`. However, no deduction for this was made, as arguably this was not a mistake since objects of the `Fruit` class needed to be serializable. (The variable is used to provide a version number for deserialization purposes. The version number is used to compare the serialized class to the class it is being deserialized to in order to check that class attributes and methods have not changed from the time of serialization, which would mean that deserialization was not possible and an `InvalidClassException` would be thrown. If a `serialVersionUID` is not set by the developer then it is automatically set by the JVM, and modified when class details change.)

Question 3: Constructors for Fruit

Students were asked to add two constructors for the Fruit class. The first should have had three parameters, and used the `super` keyword. The intention of this question was to deepen students' understanding of the use of keywords in constructors.

The `super` keyword can be used in a constructor to call the superclass constructor. Since the superclass (`Grocery`) constructor only had two parameters, students were expected to first call the `Grocery` constructor to set the values of the `name` and `producer` fields, and then set the value of the `double` field `pricePerKilo` locally, as follows:

```
public Fruit(String name, String producer, double price) {
   super(name, producer);
   pricePerKilo = price; //this line missing in some answers
}
```

In fact, most students successfully wrote this constructor; the only common mistake was that a minority of students did not set the value of `pricePerKilo`, so it defaulted to zero (note that instance variables have default values while local variables do not). However, just over a quarter of students achieved full credit for this question, mostly because many struggled with writing the second constructor. Students were asked to use the `this` keyword, for a two parameter constructor that set the value of `pricePerKilo` to -1 (intended to be a default value for the field, so that the user seeing -1 would know that a price needed to be set).

Since the `this` keyword can be used in constructors to call another constructor in the same class, students were expected to write a very simple constructor, as follows:

```
public Fruit(String name, String producer) {
   this(name, producer, -1);
   //calls the three argument constructor in Fruit
}
```

It was notable that the majority of students could not do this, despite the recommended reading covering keywords and constructors. Many students lost some credit by using the `this` keyword as follows:

```
public Fruit(String name, String producer){
   super(name, producer);
   this.pricePerKilo=-1;
}
```

Question 4: toString() for the Fruit class

Most students gained full credit for this question. There were two common errors seen: (1) not using `String.format()` and (2) not using the `toString()` from the `Grocery` class to do some of the work, and thus contradicting both the coursework requirements and the software developer's principle, DRY – Don't Repeat Yourself. The following is an example of a `toString()` method that would have received full credit:

```
public String toString() {
   String s = super.toString();
   //above calls the super class's toString() method
   return String.format("%s%-10s %.2f %s%n", s, "Price:",
      pricePerKilo, "(per kilo)");
}
```

Question 5: FruitUtils, a static utility class

This was answered badly on the whole, with just over a third of students gaining full credit, and nearly a quarter, gaining half marks or less. The major error was that many students did not understand that the methods in the class should either take a `List` or an `ArrayList` as a parameter (the `displayFruits()`, `serializeToDisk()` and `writeFruitsToFile()` methods) or should return a `List` or an `ArrayList` (the `fromSerialized()` and `parseFruitsFromCSV()` methods). What they should not be doing was operating on a local `ArrayList` or `List` variable; doing so made them of limited use outside of the class. Summary comments and common errors follow:

- the `FruitUtils` class should not extend `Fruit`
- `displayFruits()` should not be an instance method and should take an `ArrayList` parameter (some students had their method make a local `ArrayList`, then either display the empty `List` or populate it using `parseFruitsFromCSV()`). Additionally, this method should not return an `ArrayList`
- `parseFruitsFromCSV()`:
    - the method assumes that the file always contains three potential parameters, so causes a run time error when trying to parse an empty space to a double
    - the method uses the wrong logic to decide whether there is a String that could be parsed to a double, not realising that the third String in the line in the file might exist, but be empty, thus causing a run time error when trying to parse an empty space to a double
    - the method should take the file name as a parameter (it is of limited use if it does not)
    - the method should not be used to fill an `ArrayList` class variable in the `FruitUtils` class
    - the method should return an `ArrayList`
- `writeFruitsToFile()`: the method should take an `ArrayList` as a parameter, allowing the user to tell it which `ArrayList` to save to a file
- `serializeToDisk()`:
    - the method should not be saving to a .csv file, or to a .txt file, as this gives the impression of a text file, which is not the case

- the method should not be saving to a file with the same name as the `writeFruitsToFile()` method, since one file will overwrite the other
- the method should have an `ArrayList` parameter allowing the user to tell it which `ArrayList` to serialize

- `fromSerialized()`: the method should take a file name parameter or it is of limited use, and it should also return an `ArrayList`.

Question 6: Exceptions

Some students demonstrated a good grasp of exceptions, and how they can be used to let the developer know what has gone wrong, or how they can be used to allow for detecting potential but unpredictable problems (such as a file not being found) and recover from them. However, some students did not put much thought into this and made their exceptions as generic as possible. Despite this three-quarters gained full credit since the question just asked that try/catch be used to handle exceptions. Common errors were that some students wrote methods that both threw and handled exceptions depending on their type; or wrote methods that did not handle exceptions at all. Puzzlingly, some students gained partial credit by handling exceptions in some of their methods, and throwing them in others.

Question 7: A constructor for the `FruitUtils` class

Students were asked to write a constructor for the `FruitUtils` class with an appropriate access level. The `FruitUtils` class was supposed to be composed of only static methods. The recommended reading for the coursework assignment pointed out that sometimes developers write private, empty, constructors for static utility classes in order to prevent instantiation of these classes. Despite this, less than half achieved full credit for this question. There was only one correct answer, as follows:

```
private FruitUtils() {}
```

Question 8: the `FruitTest` class

Less than half of students achieved full credit for this question. Many students had created problems for themselves by the way they had written their `FruitUtils` class – use of a local `ArrayList` made it difficult (and sometimes impossible) to test the methods from outside the class. In particular, when the methods to deserialize and to read in from the file did not return `ArrayLists`, testing their output was often achieved by the student displaying the arraylist from within the method itself – a workaround, – but one that did not receive full credit. Other common issues were:

- the `toString()` methods of `Fruit` and `Grocery` should have been explicitly tested by making objects of the class and using them to call the methods, e.g.

```
Fruit f = new Fruit("Apple", "Down Dale Farms", 15.5);
System.out.println(f);

Grocery g = new Grocery("Leek", "Up Dale Farms");
System.out.println(g);
```

- testing the method to deserialize but not demonstrating its success by displaying the resulting `ArrayList`.

Other errors included (1) not testing all methods and (2) making a `FruitUtils` object to use in the testing statements. This was done by students who had clearly not undertaken the recommended reading and hence did not realise that static utility classes should not be instantiated.

Question 9: Giving classes and methods the correct names

All but two students gained full credit here.

Part b

The `BookFileParser` class was given with the coursework assignment. Some students might have seen the following warning when the class was compiled:

```
BookFileParser.java uses unchecked or unsafe
operations.

Note: Recompile with -Xlint:unchecked for details.
```

The warning came about because the class was casting an object to a parameterized list in the `fromSerialized(String) method`. See here for further details:

http://stackoverflow.com/questions/16514148/java-unchecked-or-unsafe-operations-message

Question 1: The `search()` method for the BookServerEngine class

Of those attempting this question, most students gained full credit. Credit was lost by some students who had problems with searching the entire list of books, and implemented a search method that only looked at the first item in the `List`. In a related error, a minority wrote methods that returned the first matching term from the `List` when it was found, ending the search so that subsequent matches could not be found. Some students had problems with case insensitivity; a good search method should have been written to make sure that matches would not have been missed when the only difference between the search term and the substring in the `List` was the case of one or more letters (e.g. gone/Gone). Most of these students had made the search `String` lower case, but did not make the `List` item lower case, while one had made no attempt to address upper case/lower case mismatches.

Some students made mistakes with error messages. A few did not implement an error message to the user when the search term was not found, unhelpfully returning nothing. Some students had their error message in the search loop. This meant that the user would see a message about not finding the search term for as many items in the `ArrayList` that did not match the search term, which was not really helpful; the user just needed to know found or not found, and if found, to see all the matches.

Question 2: The `History` and `HistoryListener` classes

This question was the most challenging of the assignment; only just over half of those attempting it received full credit. A few students could not make the classes work at all, while some achieved partial success, making minor errors. For example, when scrolling up through commands entered into the `ClientGUI`, once the user reached the first command entered, the list should terminate with further presses of the up arrow having no effect and the first command entered staying in the `JTextField` until the user pressed down. In some cases, further presses of the up arrow made the list repeat in a nonterminating loop.

Scrolling back down through the commands was supposed to terminate with the user returning to an empty `JTextField`. Quite often the `JTextField` would not be empty once the user had scrolled all the way down, but would instead display the most recent command entered. In a related error, in some cases scrolling up through commands ended with a blank `JTextField`, when it should have ended with the first command that was entered by the user. Other errors included where the user re-entered a command, the command did not show up as the first command (i.e. the most recently entered) in the list, but retained its old position. Another error seen was recording commands in the wrong order, that is, when scrolling up the user would see the oldest command first, and the most recently entered command last. Another minor mistake was that when scrolling up the most recently entered command was missing, but when scrolling back down it would appear.

This question was very challenging, and all those who made it work, even partially, are to be congratulated.

Conclusion

Some students could have achieved a better mark if they had paid attention to the recommended reading indicated in the coursework assignment. In general though, part (b) was very challenging, so well done to those who completed it successfully, and to those who made a good attempt at it.