Assignment One (A1): Boing! File I/O, Inheritance, Generics, and Queues

Daniel Archambault

Number of Credits: 10% of the 15 credit module
Electronic Submission Deadline:
Friday March 22nd, 2019 @ 11am

Learning Outcome(s): To gain experience in working with a larger code base. Also, in working with queues, inheritance, Generics, and file I/O.

1 Overview

In this assignment, you will be completing an application that will draw a bunch of shapes and make them fly around the canvas, bouncing off the "walls" of the window! Don't worry, you don't have to write all of the code to cause this to happen. We give you a lot of code that you can use for this assignment. Inside `BoingFinal.zip`, you should have the following code:

1. `Main.java`
2. `BouncingShapesWindow.java`
3. `QueueTester.java`
4. `Queue.java`
5. `QueueElement.java`
6. `ReadShapeFile.java`
7. `ClosedShape.java`
8. `Circle.java`
9. `Oval.java`

That is nine classes! By the end of the assignment there will be eleven. For some of you, this is probably the largest program that you have dealt with and you may find it a bit intimidating and not know where to start. On the other hand, you will have a sense of accomplishment once you have completed it. Don't fret. In this first venture into large programs that you can't fit into your head all at once (well, at least, I can't), I'll lead you through step by step.

The key to success in this assignment is concentrating on a small number of classes at each step and not worrying about the remaining code (only its ADT). Also, I will lead you through this assignment indicating where you should start and providing a path through it. If you are having difficulty, please follow the steps described in Section 2 in order to complete the assignment.

2 Steps to Complete A1

These are the steps to complete assignment one. I have ordered them in increasing difficulty and in a way that ensures you can complete the assignment. The first three steps are core and of similar difficulty. You can complete them in any order, but I would recommend the order specified. The fourth step is an advanced step and should not be attempted unless the core steps have been successfully completed. The final step provides an even greater challenge. **Do not attempt steps 4 and 5 unless you have the first three core steps working.** After each step, you should save and backup your code. This will prevent loss of work!

All five steps are worth 90%. The remaining 10% will be awarded for coding style. Javadoc is required for all classes you create and all methods of those classes.

2.1 Step 1: Make a File Reader

For this step, you will only need to modify, and thus focus on, the following class:

- `ReadShapeFile.java`

You will also need to look at **only the constructors** of the following classes:

- `Circle.java`

- `Oval.java`

Pretend the rest of the code does not exist. This is how we break large programs into smaller components that a human can understand and work with. It's also how the rest of engineering works as well. Think about how the parts of a car engine are compartmentalised nicely into components. You don't need to hold the entire functionality of the engine in your head at one time – just the component you are working on and how it interfaces with the rest of the software system.

It is very important that you **only modify code in the files your are asked to modify**. If your development environment **suggests to change code outside these files don't do it** – even if it makes things magically compile and the red text disappear. Please assume the mistake is in the code you have just written and not in the code supplied.

In this part of the assignment, you'll create a file reader that reads the shape files supplied to you. Each line of the shape file specifies a shape. The format of this line differs depending on the shape to be created. Fortunately, the first entry of the line always indicates what shape is to be created from this input. For example, to create a circle:

```
circle <px> <py> <vx> <vy> <filled?> <diameter>
<r> <g> <b> <insertion time>
```

To create an Oval, the line in the file would be as follows:

```
oval <px> <py> <vx> <vy> <filled?> <width>
<height> <r> <g> <b> <insertion time>
```

All these entries are on a single line of the file. I have broken them into two lines for readability. These numbers mean the following:

- `<px> <py>` - The starting position of the shape in the plane
- `<vx> <vy>` - The velocity of the shape as a vector
- `<filled?>` - True if the shape is filled and false otherwise
- `<r> <g> <b>` - The colour of the shape
- `<insertion time>` - time in milliseconds since the start of the program after which the shape is inserted

For circle only, you have:

- `<diameter>` - the diameter of the circle

For oval only, you have:

- `<width> <height>` - the width and height (major and minor axis) of the oval.

In `ReadShapeFile.java`, write methods to read files consisting of **circles** and **ovals only**. Also, you can **assume that the file is sorted by insertion time**. This means that the lines of the file must be in increasing `<insertion time>` order.

I would start by trying to read a single line of the file followed by multiple lines, printing them to the screen to ensure you are reading the data correctly. Then, create instances of the shape objects and print them out. You can do this by calling the `toString ()` method that has already been given to you.

You will need to throw an exception if the file does not exist. However, you can assume that each line of the file has the correct syntax. To test your code, please try to run the `TwoRedCircles.txt` file. Start by printing out each line your read to the terminal to make sure you are reading the data correctly. Then, create instances of the objects and use the `toString ()` method to print out the object and make sure that it is being created correctly.

This part of the assignment is worth a total of **22 marks**. Part marks will be given if correct information is printed to the screen using `toString ()`. You will be given **3 marks** if you submit your input files with your program.

In later stages, we will ask you to modify the file format. *Make sure that the input files are updated so that these three basic files work with your program. Submit these updated files with your code.*

2.2 Step 2: Complete the Hierarchy

For this step, you will need to create two classes:

- `Square.java`
- `Rect.java`

These two classes will inherit from `ClosedShape.java` which you can begin

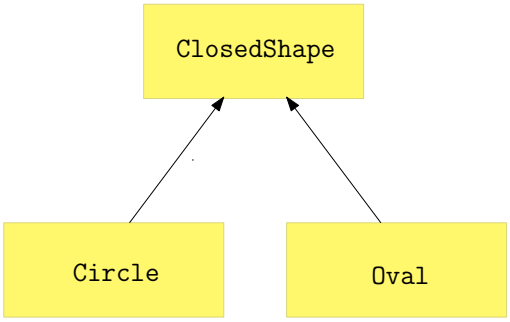

Figure 1: The current hierarchy has the following structure. Open `Circle.java` and `Oval.java` to see the use of `extends`.

reading. The classes will mirror two classes already in the inheritance hierarchy:

- `Circle.java`
- `Oval.java`

But instead of drawing a circle and oval to the screen, they should draw a square and rectangle to the screen. Currently, the hierarchy is as specified in Figure 1 with Circle and Oval as subclasses.

In order to create these classes, you should probably look at the `GraphicsContext` class in the Java API. This class is is located here:

`https://docs.oracle.com/javase/8/javafx/api/javafx/scene/canvas/GraphicsContext.html`

Also, look at how the graphics object is used in `Circle` and `Oval` when completing your classes.

Once you have completed writing these two classes, you will need to modify `ReadShapeFile.java` to read `Square` and `Rect` from files. The format of the lines of these files should be as follows:

```
square <px> <py> <vx> <vy> <filled?> <side> <r>
<g> <b> <insertion time>

rect <px> <py> <vx> <vy> <filled?> <width>
<height> <r> <g> <b> <insertion time>
```

In this format `<side>` means the length of a side in the square. All other entries are the same as in Section 2.1.

Testing this step is a bit complicated, but it's similar to the file reader. I would suggest first testing the file reader and then the shapes you have created.

You should be able to read all the shape files supplied with the assignment. I would start with `TwoRedCircles.txt`, then `ExampleShapesStill.txt` and move on to `ExampleShapes.txt`. In your file reader, print out what you are reading in for `square` and `rect` and make sure that it is correct. Then, create an instance of the class you have just written and load it with the information you have read from the file. Verify that the information stored in the instance of the class is correct by calling `toString ()`. Bugs may be in your `toString ()` method, so check with your file reader output.

In this part of the assignment, **7 marks** will be given for the `Square` class. Likewise, **7 marks** will be given for the `Rect` class. A final **1 mark** is given if shape files containing rectangles and squares can be read properly from disk. To get full marks, your code must be able to read files with these two shapes and display them on the screen.

This part of the assignment is worth a total of **15 marks**.

2.3 Step 3: Complete the Queue

For this step, you will only need to modify, and thus focus on, the following class:

- `Queue.java`

You will only need to read (*please, don't modify*) the following files:

- `QueueElement.java`
- `QueueTester.java`

As usual, pretend the rest of the code does not exist.

In this step, you will complete the `Queue.java` file so that it implements a queue with a linked list. In the class notes, we already have presented the queue abstract data structure. There are detailed notes on how one can implement a queue with a linked list. You can use these notes as a guide to complete the implementation.

Unlike in lecture, this Queue will be implemented as a java generic (lucky you)! Review the generic notes to

implement the Queue generic. The QueueElement is already implemented with a generic. You can study it to learn a bit about how generics work.

If you see the following:

```
Note: XXXX.java uses unchecked or unsafe
operations.
Note: Recompile with -Xlint:unchecked for
details.
```

there is an error in your code. Usually, a generic type parameter is missing (e.g. `Queue` instead of `Queue<T>`) Follow the instructions and go back and fix it.

Open the `Queue.java` file and complete the implementation of the Queue data structure. You must use the `QueueElement` class in order to implement your queue and your queue must be implemented through a linked list. Other solutions that do not involve using QueueElement to implement the Queue will receive zero marks. In order to complete this step, you will need to complete the following methods:

- `isEmpty ()` - checks to see if the queue is empty
- `peek ()` - returns the element on the front of the queue
- `enqueue (T e)` - places `e` on the back of the queue
- `dequeue ()` - removes the element on the front of the queue
- `print ()` - prints the content of the queue in order from front to back

To test your queue, I have supplied a `QueueTester.java` class. This class will test your queue and try out all of the operations. When your queue works reasonably well, the output should be as indicated in `0_QueueTesterOutput.pdf`.

This test is good, but you should play around with your queue a bit to make sure that it works. Also, your queue should not be written to conquer this test file. Your queue should be written in the most general way to achieve top marks.

When you have your queue working properly, you will need to modify the `ReadShapeFile.java` class to use the instance of the queue created for you to load shapes onto it. You can test that everything is loading properly by using the `print ()` method.

Now, you should try to create your first animation. Please try to run the `TwoRedCircles.txt` file. Your program should draw two red circles that bounce around the window with one being inserted immediately and a second a few seconds afterwards.

Once you have the two circles working properly, you can check all the other shapes. **Please do not proceed to this step if you do not have the two circles working properly.**

Start with `ExampleShapesStill.txt` as the shapes don't move. You can check insertion times and visually inspect the properties of the shapes to ensure that they are correct. Then, try `ExampleShapes.txt`.

Shapes must stay inside the boundary of the window when the bounce off the walls. You will lose marks if part of your shape exits the window.

This part of the assignment is worth a total of **25 marks**.

At this point, back up your work. Please make sure you have a working assignment before attempting tasks 4 and 5.

3 Step 4: Create Your Own Shape

Do not attempt this step until all of steps 1-3 work correctly.

Do not worry if you are unable to complete this advanced step.

You have already extended the `ClosedShape` class to create square and circle. For **15 marks**, do the following:

First, go to the Java API `GraphicsContext` again and select a shape that is not a circle, square, rectangle, roundrect, 3DRect, or oval. Text counts as a shape.

`https://docs.oracle.com/javase/8/javafx/api/javafx/scene/canvas/GraphicsContext.html`

Learn how to use this shape from the instructions in the API. Then, create your own class (and correspond-

ing new type of line in the file format). The program should be able to read your newly selected shape from files and animate it on the screen. You are to write a new shape file that uses your newly introduced shape and the old ones in the scene. Supply this shape file with your submitted code for the assignment.

To do this you will need to modify:

- `ReadShapeFile.java`
- the class you introduce for your newly created shape

Shapes must stay inside the boundary of the window. You will lose marks if part of your shape exits the window.

Make sure that all the input files are updated so that these three basic files work with your program. If you do not submit your updated input files, you will lose all three marks for the input files.

4 Step 5: Pulsing Shapes

Do not attempt this step until all of steps 1-4 work correctly.

Do not worry if you are unable to complete this advanced step.

As a final task, make all your shapes alternate/pulse slowly between two sizes. The sizes should be reasonable. The shape should grow and then shrink back to its original size. In the file format, there should be a boolean variable to indicate if the shape pulses. Please update all of the input files so that they work properly with your new format, create a new file that demonstrates your pulsing shape, and submit all files with the solution to your assignment.

To do this you will need to modify:

- `BouncingShapesWindow.java`
- `ReadShapeFile.java`
- `ClosedShape.java`

To accomplish this, you should **not** modify any of the `draw ()` methods anywhere in the code. Instead, you should find another way.

`BouncingShapesWindow` does not follow coding conventions. Leave it as such but appreciate why style is important when modifying this file.

In this part of the assignment, **10 marks** will be given if your shapes pulse between two sizes as they move.

Make sure that all the input files are updated so that these three basic files work with your program. If you do not submit your updated input files, you will lose all three marks for the input files.

5 Submission Instructions

Submission of the assignment is **electronic only**. In order to submit, create a `.zip` file of all your code including your input files. Include all code that was given to you. All of your code should be in a directory named `src`. All of your input files should be placed one directory back from `src` like they were when they were given to you. *Submit only input files that work with your program. If you changed the file format, you must update TwoRedCircles.txt, ExampleShapesStill.txt, ExampleShapes.txt so that they work properly. You will lose marks if you don't submit working input files.*. The only format we will accept is `.zip`. Do not submit any other format. You will lose marks.

After submitting to blackboard **please check** that your uploaded `.zip` file on Blackboard is **readable and contains all files necessary for submission**. Multiple submission attempts are allowed. If there is an error in your submission, please upload it again before the deadline. If all or some of the files necessary for your submission are not present or are unreadable, you will lose marks (or potentially get a zero). After the deadline, we will only use version(s) of your assignment that have been submitted to Blackboard before the deadline. No other versions of your assignment will be accepted. No late submissions of the assignment are accepted without extenuating circumstances.

Feedback for this assignment will be given through Blackboard. In order to see the full rubric, please access it through the assignment page on Blackboard.

In order to submit to Blackboard, click on the red link in the assignments folder and follow the instructions to submit your zip file. Happy coding!