Task 3 - Files and Parsing

Prepared by William Reid
September 18, 2014

1 JFileChooser

When opening and saving files, Java provides us with a way of graphically choosing files and folders. Instead of designing our own interface for opening and saving files, the **JFileChooser** package provides much of this functionality for us. In addition with the **FileNameExtensionFilter** class, the code below opens an interface element for the user to select either a JPG or GIF image to open. It then prints out the name of the file chosen (if one was chosen).

```
JFileChooser chooser = new JFileChooser();
    FileNameExtensionFilter filter =
      new FileNameExtensionFilter("JPG & GIF Images", "jpg", "gif");
    chooser.setFileFilter(filter);
    int returnVal = chooser.showOpenDialog(parent);
    if(returnVal == JFileChooser.APPROVE_OPTION) {
       System.out.println("You chose to open this file: "
         +  chooser.getSelectedFile().getName());
    }
```

2 Adding a JFileChooser - (10 marks)

In your DrawFrame class, add to your listener private classes such that when the *Open* and *Save* functions are used from the menu, they allow the user to graphically pick a file. When the user clicks "OK", print out the name of the file (as well as the menu option that was chosen) to the console.

For the Open and Save options they must:

1. Allow the user to select only TXT files **(2 marks)**
2. Print out the current file chosen **(2 marks)**
3. Print the menu option chosen **(1 mark)**
4. Print ONLY the menu option chosen when no file is chosen (user clicks "Cancel") **(2 marks)**

3 The Shape Class - (18 marks)

Create a class called *Shape.java* that extends the *Polygon* class. This class will hold all information pertaining to every shape we will make in this program. For this exercise, we can assume that all shapes we will make can be defined by one or more lines.

Your Shape class must:

- Be able to store the type of item ("line", "square" etc.) **(2 marks)**
- Be able to store the shape colour **(1 mark)**
- Be able to add and remove X and Y coordinates **(4 marks)**
- Not be able to change the name of the shape **(1 mark)**

Your class will have two constructors and these will match the two constructors found in the *Polygon* class **(4 marks)**:

http://docs.oracle.com/javase/7/docs/api/java/awt/Polygon.html

Think about how you might be able to use some of the functions provided by *Polygon* instead of creating your own (primarily to do with adding and removing points).

3.1 Adding Your Shape to DrawPanel - (4 marks)

In your DrawPanel class, remove the code that draws a line from the *paintComponent()* method. In the same method, create a new line using your *Shape* class. Once you have created your shape, add code that will draw the shape onto the DrawPanel. Methods exist in the **Graphics** package that may be able to assist you:

http://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html

3.2 Adding Multiple Shapes - (8 marks)

In your DrawPanel class, add a function *addShapes()*. This method must create at least 2 shapes and store them in some local variable (perhaps an Array or preferably an **ArrayList**). Choose your data structure carefully as we want the system to be as extendible as possible.

In your *paintComponent()* method, modify the code such that it extracts every shape from the data structure you have created and adds them to the drawing surface **(2 marks)**.

4 Reading Files - (20 marks)

Create a class called *ShapeParser.java*. The constructor for this class does not have to do anything. You will need two other functions:

1. *readFile* - Will take in a file name as a string, and open it for reading. It will also return an ArrayList of Shapes. **(2 marks)**

2. *writeFile* - Will take in a file name and an ArrayList of Shapes. It will write these Shapes to a new file, given by the supplied file name. **(2 marks)**

For now, your *writeFile* function does not have to do anything (please add all required parameters to the function), we will be adding to this later. The *readFile* must open a text file for reading **(6 marks)** and create the shapes described in the text file **(7 marks)**. A sample text file is supplied with this task - it is in the format:

type: (X1, Y1), (X2, Y2), ... (Xn, Yn), colour

Points *(X1, Y1), (X2, Y2)* are required but the remaining points are optional and may not be specified for a shape. To open the file, you will need to use Java's **BufferedReader** class - the following reference should assist you:

http://stackoverflow.com/questions/3806062/how-to-open-a-txt-file-and-read-numbers-in-java

From here, you will need to use the functions contained in Java's **String** library to split up the line to obtain its information. With this information you should create a new shape and add it to a local data structure. Do this for every ling you find before returning the list of shapes to the caller **(2 marks)**.

Exceptions

An exception is what most people would call an error. It is usually something that a program cannot automatically handle on its own and requires guidance to repair. If no such guidance is available - the program will crash. **Catching** exceptions tells the program that we are able to provide it some guidance when specific errors occur. The following tutorial may help your understanding:

http://docs.oracle.com/javase/tutorial/essential/exceptions/

Sometimes classes require us to catch exceptions - the BufferedReader class is one such example. It is important that we catch these two primary exceptions when reading in a file as they frequently occur. For each of these exceptions, just print out a nice error message briefly stating what the problem is, before exiting the program - *System.exit(1)*;

4.1 Modifying DrawFrame - (10 marks)

Modify your *DrawFrame* constructor to create a *ShapeParser* object (place this code after the DrawPanel is added to the frame) **(2 marks)**. Pass it the sample file supplied with this task and assign the returned data structure to a local variable **(4 marks)**. To test that it works, iterate through the returned data structure and print out the name and colour of each shape (this can be performed in the *DrawFrame* constructor) **(4 marks)**.

Note: The DrawFrame class does not need to store this data structure locally!*

4.2 Modifying DrawPanel - (8 marks)

Modify the *addShape()* method in the *DrawPanel* class such that it takes in a data structure containing Shape objects **(2 marks)**. Remove the code from the *DrawFrame* constructor that prints out the name and colour of each shape and place this into the DrawPanel *addShape()* method **(1 mark)**.

Adjust the code in the *DrawFrame* constructor such that the creation of the DrawPanel and adding it to the content pane is done in two steps - this will allow us to operate on the DrawPanel from inside the DrawFrame **(2 marks)**.

Pass the DrawPanel the data structure containing the shapes obtained from the *ShapeParser* class. The DrawPanel class MUST store this data locally - you will need to adjust the DrawPanel's *addShape()* function to no longer create new Shapes, if you have not already done so **(3 marks)**.

4.2.1 Testing - (2 marks)

If everything has been named and implemented correctly, the DrawPanel should now add Shapes to the drawing surface that have been specified in the text file. The code we implemented in the *paintComponent()* method in **Exercise 3.2** should already draw any shapes in the local data structure (that stores Shapes) onto the drawing surface. If this does not occur, find the source of the errors and report the process in a text file named *debug.txt*

To test your program further, try and pass the *ShapeParser* a file that does not exist and see if it behaves as we expect.

5 Checking your Code (10 marks)

**Note: For each file - 1 mark each goes to comments, layout/structure and conforming to the function/variable naming standards. This does not include Draw.java as this has not been changed. DrawFrame.java and DrawPanel.java will not be marked on layout/structure*

Review your code for layout, structure, comments and how it measures against the supplied standards. When you have completed this check, ensure it is uploaded to the Git repository (you should be committing your work once you have completed each class).

Notify me that you have finished and I will check your work, giving feedback on how things can be improved, including your total mark out of **90**. After I have given feedback you may be asked to modify the code (based on my comments). Once you have completed this and notified me, will move on to the next task. Good luck!

6 Program Understanding

Do you know what we have achieved in these classes? Do you understand how files are read in what we can achieve with exceptions? Create a document called *Questions.txt* that lists questions of things you need clarification on. It may be useful to make this document before you start such that you can add and remove to it as you continue through these tasks.