Practical session 1, Week 6: The Drawing Application's Menu Bar (File I/O)

In this practical you will be completing the implementation of the menu bar in the drawing application. You will be doing this by adding functionality to the menu choices of the menu bar.

Please note that you are expected to complete this practical in a single session.

Task 1

There are 4 menu items in the menu bar. Here is a short description of what each should do:

File → Save
Should save the contents of the canvas in a file (permanent storage). For simplicity, the file name should always be kept the same i.e. the file should be overwritten every time a save is requested.

File → Load
Should load the contents of the above file into the drawing application and redraw the canvas. This operation should replace the previous contents of the canvas.

File → Exit
Should simply terminate the drawing application.

Help → About
Should display a dialog window with some information about the drawing application. In the simplest case, it should display the author's name, the application's version and the year of creation.

Just like almost any interaction with GUI components, the selection of a menu item causes an event to occur. Events from menu items are serviced by instances of classes implementing the familiar `ActionListener` interface. `ActionListener` is the same interface used in the case of buttons such as the "Clear Canvas" or the colour button in our drawing application. Remember that you only need to implement the `actionPerformed` method for this interface. This is the method that will be called when the action event occurs.

In order to associate a menu item with an instance that implements the above listener you need to use the menu item's `addActionListener` method. Again, recall that the same method in the `Button` class was used in order to associate the "Clear Canvas" or colour button objects to their respective listener objects.

To complete Task 1 you need to create outline code for the four listener classes for the menu items above and associate each menu item with an instance of its corresponding listener class.

For a quick test of your outline code, add some code in each listener class in order to verify that your event capture mechanism works for each menu item. For example, you could post an informative message on the message area each time a menu item is selected.

Task 2

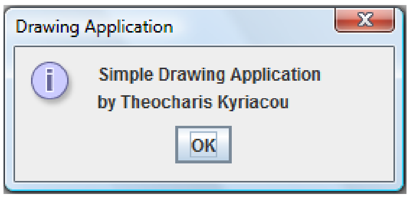
In order to terminate the Java virtual machine (and therefore the execution of your program) you need to call the `exit` (static) method of the `System` class. This method takes an integer as a parameter that it passes to the operating system as a mechanism for your code to indicate the exit status of your program. By convention, exit status 0 indicates normal termination whereas a non-zero value for exit status indicates that your program did not terminate normally.

When you select the **File** → **Exit** menu item therefore you simply need the following command to be executed:

```
...
System.exit(0);
...
```

Task 3

Remember from your lectures that the `JOptionPane` class provides us with a quick way for passing some information to the user or accepting simple input from them by means of *dialog windows*. Just like the colour chooser dialog that you have used in practical 3 these windows save us from quite a bit of programming.


Drawing Application

Simple Drawing Application
by Theocharis Kyriacou

Figure 1: An example of a message dialog window.

For example, for the simple message dialog window shown in figure 1 there is existing code that creates and positions the window, the OK button, the label that says “Simple Drawing...”, the information icon, code that creates a listener for the OK button etc. All this and a lot more (for other types of dialog windows) is conveniently implemented for you in the `JOptionPane` class. You need only to write one line of code to produce a dialog window such as the one above when the **Help** → **About** menu item is selected in the drawing application.

In order to find out how to use the `JOptionPane` class please refer to the class’s API by scrolling down to its entry at the Java 7 documentation page at:

https://docs.oracle.com/javase/7/docs/api/

and/or the simple tutorial at:

https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html

Task 4

In order to implement the saving and loading of the canvas contents you need to recall your recent lecture on file I/O. **You are strongly advised to revise your lecture and the accompanying examples before proceeding on to Task 5 in this practical.**

Task 5

Recall from your code that in order to reconstruct/redraw the canvas contents you need the following data structures:

Data structure name	Data type	Description
`lxy`	`int[][]`	Coordinates of lines
`lineColour`	`Color[]`	Colours of lines
`linesCount`	`int`	Number of lines drawn
`rxy`	`int[][]`	Coordinates of rectangles
`rectangleColour`	`Color[]`	Colours of rectangles
`rectanglesCount`	`int`	Number of rectangles drawn
`oxy`	`int[][]`	Coordinates of ovals
`ovalColour`	`Color[]`	Colours of ovals
`ovalsCount`	`int`	Number of ovals drawn
`fxy`	`int[][]`	Coordinates of freehand pixels
`freehandColour`	`Color[]`	Colour of freehand pixels
`freehandPixelsCount`	`int`	Number of freehand pixels

You may have chosen different names for your data structures but their type and role should be the same as described above.

In order to save the canvas contents in a file you need to save all the data contained in these data structures. The easiest way you can do this is by using an `ObjectOutputStream` object.

An `ObjectOutputStream` object allows you to save an object (such as an array for example) to a stream in a straightforward manner using its `writeObject` method. Objects of different types can be saved in the same file using this same method. Note it is the responsibility of any program that subsequently reads from the file, however, to read the objects back in the same order that they were written (see next task below).

Notice that amongst the data listed in the table above there are a few simple `int` variables. In Java `int` is a **primitive data type** and a simple `int` variable is not an object. Even so, you can still use the `writeObject` method to write an `int` data item to the output stream, the `int` data type is actually saved as an instance of class `Integer`. But this could make things a bit more complicated than they need to be when you come to read the integer later! Fortunately, to simplify things the `ObjectOutputStream` class provides separate methods for writing primitive data types. For writing simple integers for example you can use the `writeInt` method:

```
...
FileOutputStream fos = new FileOutputStream("drawing");
ObjectOutputStream fh = new ObjectOutputStream(fos);

fh.writeInt(linesCount);
...
```

It is important to realise the enormous saving that you benefit from by using the `ObjectOutputStream` class. **Looking at the different examples in your recent lecture on file I/O can you think of a different method (much more cumbersome) to save the above data? You may want to discuss your solution with a demonstrator.**

Using the code examples from your lecture, write some suitable code in the save menu item listener in order to save all the canvas data into a single file using a `ObjectOutputStream`.

Task 6

In order to implement the load menu item you need to use an `ObjectInputStream`. The `ObjectInputStream` class uses the `readObject` method to read an object from the stream. The class also implements different methods for reading primitive data types. For reading integers, for example, the `readInt` method can be used.

By looking at the API of the `ObjectInputStream` class, note that the `readObject` method always returns an instance of class `Object`. Remember that all classes inherit (they are descendants) from class `Object` and therefore any object can be *type cast* to this class. (You may need to refresh you memory on the *casting* of data types in Java?) When you read an object into its corresponding data structure above you will need to type cast it to the data type of that structure. For example:

```
...
FileInputStream fis = new FileInputStream("drawing");
ObjectInputStream fh = new ObjectInputStream(fis);

lxy = (int[][])fh.readObject();
...
```

The important thing to note here, as mentioned before, is that data must be read from the stream in the same order as they were written. **What do you think will happen if you try to read an object of the wrong data type?**

Again with reference to the examples in your recent lecture, write code in the load menu item listener in order to load all the canvas data from the file that you have saved them into in the previous task, this time using the `ObjectInputStream` class.