Homework 6 part 2: Turtle Etch-A-Sketch (40pts)

DUE DATE: Friday March 28, 7pm with 5 hour grace period

Now that you have done part 1 of Homework 6, Train Your Turtle to Draw on Command, you are ready to move to the second portion of the assignment. Use your own solution to part 1 or the provided one to complete part 2. Every time you see a *CHECKPOINT* marker, you should definitely stop and check the functionality of your work. It is also recommended that you compile often as well.

TurtleDrawingFileIO

The first thing to do for this homework is to convert the methods from `TurtleDrawing` into `static` methods for `TurtleDrawingFileIO`. If you followed the instructions for naming for homework 6 part 1, this portion of the homework should be as simple as copy, paste and change the method header. You also have to go through and change what some of the methods return. The following table (Table 6.1) outlines the changes you need to make:

Table 6.1 Conversions for the TurtleDrawingFileIO class

Original Method Header	New Method Header	What Else to Change
public void extractCommands(String turtleCommands) throws java.io.FileNotFoundException	public static LinkedList<TurtleCommand> extractCommands(String turtleCommands) throws java.io.FileNotFoundException	Add LinkedList<TurtleCommand> commands = new LinkedList<TurtleCommand>(); at the beginning of the method Return the LinkedList commands at the end
public void addMoreCommandsFromFile(String moreCommands) throws java.io.FileNotFoundException	public static LinkedList<TurtleCommand> addMoreCommandsFromFile(LinkedList<TurtleCommand> commands, String moreCommands) throws java.io.FileNotFoundException	Return the LinkedList commands at the end
public void outputToFile(String outputFile) throws java.io.FileNotFoundException, java.io.IOException	public static void outputToFile(LinkedList<TurtleCommand> commands, String outputFile) throws java.io.FileNotFoundException, java.io.IOException	Nothing
public String toString()	public static String convertCommandsToString(LinkedList<TurtleCommand> commands)	Nothing

CHECKPOINT
After you have completed the conversion, test your methods in the `main` method of `TurtleDrawingFileIO` to make sure everything still works before moving on (Remember that all of do not need to write a constructor for `TurtleDrawingFileIO`).

TurtlePanel

Write a new class called `TurtlePanel` that `extends JPanel`. In the `TurtlePanel` class, you will be required to write two new constructors and at least four new methods: `drawCommands`, `drawCommand`, `save`, `load` and `clear`.

public TurtlePanel()

The default constructor will set up a `World` of default size 500 by 500. Remember to hide the `World` and add it to the panel (just do `this.add(world)` where `world` is a `World`). You also need to create a new `Turtle` and place him on the `World`. You need to save this `Turtle`'s pen color and also set his speed to be 50. Since `TurtlePanel` will keep track of a `LinkedList` of `TurtleCommand` objects, it must be initialize to an empty `LinkedList` in the constructor. Because you will need to access these variables in other methods, they will all need to be global variables (also known as attributes).

public TurtlePanel(int width, int height)

The only difference in this constructor is that the `World` will not be a default 500 by 500 but instead is created based on the parameters.

public void setTurtleSpeed(int turtleSpeed) and public int getTurtleSpeed()

You should write a getter and a setter for the `Turtle`'s speed, because it will be useful later on.

public void drawCommands(LinkedList<TurtleCommand> commands)

Take a look at `TurtleDrawing` and copy and paste the `drawCommands` method into `TurtlePanel`. Just modify the method header so that takes in a `LinkedList` of commands to draw. Remove the lines regarding what do about a world command since it is no longer necessary and also the lines to pause after each command. You should also add the incoming the `LinkedList` of new commands to the list of old commands.

CHECKPOINT

Time for another checkpoint to check your work. You should write another main method for this class, but it is a little different from any you have written before. We will walk you through this first one, but you will be expected to write similar ones for later checkpoints.

First you should create a new `TurtlePanel`. Since every GUI component must exist in a `JFrame` in one way or another, you also need to create a `JFrame`. You should get in the habit of setting the default close operation to the static final value, `JFrame.EXIT_ON_CLOSE`, because it will close the process when you click the X on the window. Now comes some pretty standard lines. You have to get what is currently on the frame and add your `TurtlePanel` to it. Then you pack up the frame and set it to be visible. Then extract commands from a text file and pass the resulting `LinkedList` into the `drawCommands` method for your `TurtlePanel`. See the following page for the code.

CHECKPOINT (Continued)

```
public static void main(String[]args)
    throws java.io.FileNotFoundException{
    TurtlePanel panel = new TurtlePanel();
    JFrame frame = new JFrame ("My Turtle Panel");
    frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().add(panel);
    frame.pack();
    frame.setVisible(true);
    LinkedList<TurtleCommand> commands =
        TurtleDrawingFileIO.extractCommands("commands2.txt");
    panel.drawCommands(commands);
}
```

public void save() throws java.io.FileNotFoundException, java.io.IOException

This method is where you will save your drawing to a text file. Instead of passing in a file path into the method, you should use `FileChooser.pickAFile()` to get your file path. You actually type in file name if a text file does not exist already. Then you should the `outputToFile` method `from TurtleDrawingFileIO` passing in the commands `LinkedList` and the file path you got from the `pickAFile` method.

CHECKPOINT

Test your save method by adding it to the main method you have already written.

public void clear()

This method is where you clear the `World`. To remove all lines from the `World`, you have to remove the `Turtle`. Do `world.remove(turtle)` where `world` is the `World` and `turtle` is the `Turtle`. Then you have to put a new `Turtle` on the `World`. Remember to hide him again.

You also have to set the `Turtle`'s pen color to the default pen color saved from the constructor otherwise his pen color will be different (just how the `Turtle` class works).

CHECKPOINT
Test your `clear` method by adding it to the `main` method you have already written.

public void load() throws java.io.FileNotFoundException

This is the method where you load commands from a file and draw them on the screen. First use the `clear` method you wrote before to clear the screen. Reset the commands `LinkedList` to an empty `LinkedList`. Then similar to the save method, use `FileChooser.pickAFile()` to get your file path. Pass this into the `extractCommands` method from `TurtleDrawingFileIO`. Finally pass the resulting `LinkedList` into the `drawCommands` method.

CHECKPOINT
Comment out the lines testing the `save` and `clear` methods. Then test your `load` method by adding it to the `main` method.

public void drawCommand(String command)

This is a method that takes in String that `is "right", "left", "up"` and `"down"` and will create and draw a series of commands. If the `"right"` is passed in, the first command will be a `"setHeading"` with the parameter 90 and then the second command will be a `"forward"` command with the parameter equal to the `Turtle`'s current speed. The other `String`s work similarly (see Table 6.2). After creating the commands, add them into a `LinkedList` of new commands and pass that `LinkedList` into the `drawCommands` method.

Table 6.2 Headings for the different inputs of the drawCommand method

String	Heading
right	90
left	270
up	0
down	180

CHECKPOINT
Comment out the lines testing the `load` method. Then test your `drawCommand` method in the `main` method by passing in the 4 different possible inputs.

TurtleSketchPanel

`TurtleSketchPanel` is a class that `extends JPanel` and contains the `TurtlePanel` and all the other GUI components. We will go on to describe each component of the `TurtleSketchPanel` and each of their roles in the great scheme of things, but their placement in the `TurtleSketchPanel` is entirely up to you. You will be required to use at least TWO different types of LayoutManagers. All your GUI components especially those that will have `ActionListener`s attached should be global variables. Read the following overview and we will go in a more step-by-step procedure afterwards.

The overview

1. Buttons
    - up, down, left and right buttons to represent the different directions you can move your Turtle.
    - A save button to save your work to a text file.
    - A load button to load commands from a text file.
    - A shake button to clear the workspace (world).

2. JLabel
    - A speed label to label your speed text field.

3. JTextField
    - A speed text field to take an amount for the Turtle to move forward each time.

4. TurtlePanel
    - A TurtlePanel that will house the drawing itself.

public TurtleSketchPanel()
This constructor is where you will be initializing your components, adding `ActionListener`s to them and adding them to the panel.

CHECKPOINT
It is our strong recommendation to do this incrementally adding one component at a time. First add the `TurtlePanel` and then start adding the other components one-by-one. Get one working before moving on to another. You will need to write a `main` method like the one for `TurtlePanel` to see the final resulting panel.

private class ButtonListener implements ActionListener

You will need to write a private inner class called `ButtonListener` to handle all of the button actions in your panel. You have already had some examples of this in class and recitation. Use those examples as a guide to writing this one. Table 6.3 outlines what actions will happen when each button is pressed.

CHECKPOINT

Again you should do this incrementally. Make sure the `ButtonListener` is working for one button before moving on to another button.

private class TextListener implements ActionListener

You will need to write another private inner class called `TextListener` to handle all of the text actions in your panel. This class is essentially the same as the `ButtonListener` class.

CHECKPOINT

Make sure the `TextListener` is working before moving on.

Table 6.3 ButtonListener and TextListener and actions

GUI Component	Actions
Right button	drawCommand is called with `"right"`
Left button	drawCommand is called with `"left"`
Up button	drawCommand is called with `"up"`
Down button	drawCommand is called with `"down"`
Shake button	clear is called
Save button	save is called
Load button	load is called
Speed field	The turtle's speed is set to the input

TurtleEtchASketch

This class will extend `JFrame` and serve as our main container for the `TurtleSketchPanel`. You need to write one constructor and a `main` method.

public TurtleEtchASketch(String title)

This constructor will just call the `super` constructor with the same input.

The main method

The `main` method for `TurtleEtchASketch` is essentially the same as the main method for `TurtleSketchPanel` except the `JFrame` is now `TurtleEtchASketch` and you are required to give the constructor an appropriate title for the frame.

Extra Credit

- +5 pts Make it so that the Turtle will move with keyboard input instead of buttons
- +5 pts Make it so that the Turtle can be dragged with a mouse click to another portion of the screen
- Other creative ideas will also be rewarded!

What to Turn In

- `TurtleDrawingFileIO.java`
- `TurtlePanel.java`
- `TurtleSketchPanel.java`
- `TurtleEtchASketch.java`

Where to Turn In

- T-square