Assignment 2 - Java Basics

1. **Now, let's move on to some code with a graphical interface.** Download starter.zip from Blackboard. Unzip it. On Linux, the command is:

   ```
   unzip starter.zip
   ```

   (It is recommended you learn to use the shell for simple tasks, like zipping and unzipping folders, instead of some graphical program. Graphical tools are very difficult to use in scripts. Those who learn to use the shell can automate such tasks. In software development, automation is **very** important.)

   Now, build and run this program. You should notice that I included both the .bash and .bat build files. You will only need to use one of them, depending on your operating system. You should see a button. When you push the button, it prints something in the console window.

   Read through all of the provided code, and follow any directions that are found in the comments as well.

2. **Examine Game.java**. Observe the following points. If any of these points are unclear, use a search engine find some information to help you understand it better:
   - Java programs begins in "main".
   - The main function makes a "new Game" object. That means it allocates a portion of memory big enough to hold an instance of "Game".
   - We call such allocated portions of memory "class instances" or "objects".
   - When you make a new object, it implicitly calls the constructor. A constructor is a method with the same name as the class. The usual purpose of a constructor is to initialize the values in a new object.
   - The Game constructor makes two local variables named "controller" and "view". A variable is a reference (or pointer) to some place in memory. In this case, these variables reference two new objects of type "Controller" and "View".
   - Note that it is a common convention to start type names with a capital letter and to start variable names with a lowercase letter.
   - "this" refers to the object that is being initialized by the constructor.
   - Notice that the Game class extends JFrame. JFrame is a class that comes with the Java class library. By extending JFrame, the Game class inherits all the methods and member variables in the JFrame class. A JFrame is the frame around a graphical program. It is basically basically a window with a close button and a maximize button.

3. **Examine View.java**. Observe the following points. An important part of this exercise is learning the language that I will use to communicate with you about Java code. So, read these points carefully and try to understand. Again, using a search engine is a good idea:
   - View extends JPanel. A JPanel represents the big gray area inside the JFrame. It is the place where the graphical components of your program go.
   - The View class has a member variable of type "JButton", named "b1". A member variable is a variable stored in an object.
   - Notice that the View constructor requires a parameter. This parameter is passed to it when a "new View()" is instantiated.
   - The View constructor instantiates a new JButton and uses the member variable, b1, to reference this new object.

- The method "addActionListener" is a member of the JButton class. This method tells the button which object will handle the event that occurs when someone presses the button. We will use our controller object to handle this event, so we pass a reference to our controller object to the addActionListener method.
- We also add b1 to "this" object, meaning we add the button to the panel. If you don't do that, it won't appear in the window.

4. **Examine Controller.java**. Here are some points to observe:
   - This class "implements ActionListener". That means "This class is capable of handling ActionEvents, such as when someone pushes a button." In order to implement ActionListener, a class must provide a method named "actionPerformed", which is the method that will handle the event.
   - The "actionPerformed" method just prints a silly message to the console. "System" is a class provided by the Java class library that provides some important system functionality. "out" is a reference to an object that represents where text will be printed. By default, "out" is connected to your console, but it is possible to redirect it to go to other places.

5. **Make the button remove itself** when you press it. Change the text of the button to something else. (I do not care what you change it to.)

   It is highly recommended that you build and run after every little change that you make. That way, if the code breaks, you won't have to guess about which change is responsible. Also, it doesn't hurt to make full backups at regular intervals.

   In the Controller class (in Controller.java), add a class member variable to reference the View:

   ```
   private View view;
   ```

   (If you need an example, the View class has a class member variable in it that you can look at. Generally, you should put all of the member variables at the top of the class, before the methods. The constructor is usually the first method after the member variables. However, you can put things wherever you want, as long as it works.)

   Add a method to the Controller class that lets the caller set the object that "view" references.

   ```
   void setView(View v)
   {
       view = v;
   }
   ```

   Call that setter method from the View constructor:

   ```
   c.setView(this);
   ```

   Now, the view and controller objects both have references to each other.

   Add a method to the View class that removes the button:

   ```
   void removeButton()
   {
       this.remove(b1);
       this.repaint();
   }
   ```

   Finally, change the behavior of the button so that it removes itself. So replace this line,

   ```
   System.out.println("Hey! I said not to push that button!");
   ```

   with

   ```
   view.removeButton();
   ```

When you run the program, you should see that pushing the button makes it disappear.

6. **Load a turtle image and change the background color of the View**. Download turtle.png from Blackboard and save it in the folder with your code. Add a member variable to your view to reference this image:

```
private BufferedImage turtle_image;
```

Add the following code to the View constructor to load this image from disk:

```
try
{
    this.turtle_image = ImageIO.read(new File("turtle.png"));
}
catch(Exception e)
{
    e.printStackTrace(System.err);
    System.exit(1);
}
```

But, where is the turtle? Well, we loaded it into memory, but we never actually told the program to draw it. So, right now, it's just filling up some memory. (Computers are not generally intelligent. They do not do obvious things unless they are explicitly instructed to do them.)

Add a method to the View class that will draw it:

```
public void paintComponent(Graphics g)
{
    g.fillRect(0, 0, this.getWidth(), this.getHeight());
    g.drawImage(this.turtle_image, 50, 200, null);
}
```

paintComponent is already a method in the JPanel class, which the View class extends. By putting this method in your View class, you are overriding the one in JPanel, so it calls yours instead.

Notice that the background is now black. Let's change it to a cyan color by inserting this line

```
g.setColor(new Color(128, 255, 255));
```

just before the call to "g.fillRect". Now, when you build, you might get the following compiler error:

```
View.java:31: error: cannot find symbol
                g.setColor(new Color(128, 255, 255));
                               ^
  symbol:   class Color
  location: class View
1 error
```

It cannot find the symbol "Color" because we have not told it where to look. Add this line near the top of the file:

```
import java.awt.Color;
```

This line tells it that the "Color" symbol is found in the namespace "java.awt". A "namespace" is a hierarchical system that Java uses to keep all of its classes organized. Now the program should build without errors, and when you run, you will have a Cyan background.

7. **Make a new class to represent the state of the game and draw the turtle**. Add a new file to your project named "Model.java". Put the following contents in this file:

```
public class Model
{
        private int turtleX;
```

```
        private int turtleY;
        private int destX;
        private int destY;

        public Model()
        {
        }

        public void update()
        {
                // Move the turtle
                if(this.turtleX < this.destX)
                        this.turtleX += 1;
                else if(this.turtleX > this.destX)
                        this.turtleX -= 1;
                if(this.turtleY < this.destY)
                        this.turtleY += 1;
                else if(this.turtleY > this.destY)
                        this.turtleY -= 1;
        }

        public void setDestination(int x, int y)
        {
                this.destX = x;
                this.destY = y;
        }

        public int getTurtleX()
        {
                return turtleX;
        }

        public int getTurtleY()
        {
                return turtleY;
        }
}
```

Add any additional getters and setters as necessary.

Add "Model.java" to the list of files that your build script builds.

In the Game constructor, declare and instantiate a new Model object.

Also, in Controller, add a class member variable to reference the Model object:

```
private Model model;
```

Add a new parameter of type Model to the Controller constructor (change what's currently there), and use it to initialize the "model" member variable:

```
public Controller(Model m)
{
    model = m;
}
```

Now, if you try to build, it will tell you:

```
Game.java:11: error: constructor Controller in class Controller
                cannot be applied to given types;
                Controller controller = new Controller();
                                        ^
  required: Model
  found: no arguments
  reason: actual and formal argument lists differ in length
```

This means there is a problem in Game.java on line 11. (It might be a different line number for you.) The problem is that the Controller constructor requires one parameter value, but you are

currently supplying zero. (You are supposed to be able to determine that by reading the error message. In this case, the error message is not worded very clearly, so it helps to have some experience reading error messages. Start building that experience now by carefully reading the error messages that go with each problem that you fix. Don't just be happy to get it working--a little time spent learning to understand the compiler will pay off in the long run.) Now, fix this problem by passing the model object when you instantiate your controller:

```
public Game()
{
    Model model = new Model();
    Controller controller = new Controller(model);
```

The View is also going to need a reference to the model. So, go ahead and do that, just like you did with the Controller: add a private class member variable of type Model to your View and initialize it by passing a model object to the View constructor.

Adjust the paintComponent method to draw the turtle where the model says the turtle is located:

```
g.drawImage(this.turtle_image, model.getTurtleX(), model.getTurtleY(), null);
```

Change the title of the JFrame to "Turtle wars!"

8. **Animate it**. Then, add the following method to your Game class:

```
public void run()
{
        while(true)
        {
                controller.update();
                model.update();
                view.repaint(); // This will indirectly call View.paintComponent
                Toolkit.getDefaultToolkit().sync(); // Updates screen

                // Go to sleep for 50 milliseconds
                try
                {
                        Thread.sleep(50);
                } catch(Exception e) {
                        e.printStackTrace();
                        System.exit(1);
                }
                System.out.println("hello there"); // remove this line
        }
}
```

When you build, now, you will get an error about "cannot find symbol". Read that error message and see if you can figure out what it is saying. (Especially note the line number and the name of the symbol it cannot find.)

...

Okay, I'll tell you what it means. It is saying that you cannot access "model" or "view" here, because those variables were declared in a different place. Now, see if you can figure out how to fix this problem. (If you can't figure it out, scroll way down.)

Call your run method:

```
public static void main(String[] args)
{
        Game g = new Game();
```

```
            g.run();
    }
```

Now it will redraw the view every 50 milliseconds. (1000 milliseconds-per-second / 50 milliseconds-per-event = 20 frames-per-second). Change it to update at a rate of 25 frames per second. *You're going to have to do some math here!* Be sure to update any relevant comments as well.

However, it is difficult to tell that it is drawing many times per second because nothing moves. So, let's fix that. Make your Controller class also implement the MouseListener interface:

```
class Controller implements ActionListener, MouseListener
{
```

and add the following methods:

```
        public void mousePressed(MouseEvent e)
        {
                model.setDestination(e.getX(), e.getY());
        }

        public void mouseReleased(MouseEvent e) {    }
        public void mouseEntered(MouseEvent e) {    }
        public void mouseExited(MouseEvent e) {    }
        public void mouseClicked(MouseEvent e) {    }
```

and add this line to your Game constructor to tell it that the Controller is in charge of handling mouse clicks

```
                view.addMouseListener(controller);
```

Now, when you build and run, you should see that you can control the turtle by clicking somewhere inside the window. The turtle should move to the location where you clicked. (If the animation appears jerky, try wiggling your mouse inside the window. Some Java virtual machines try to reduce computational requirements by skipping screen refreshes. When the mouse moves, however, they know you are watching, so they don't try to get away with that.)

9. **Let's give it keyboard controls too.** Start by making Controller implement the KeyListener interface.

```
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;

class Controller implements ActionListener, MouseListener, KeyListener
```

Then, add these member variables,

```
        private boolean keyLeft;
        private boolean keyRight;
        private boolean keyUp;
        private boolean keyDown;
```

and these methods:

```
        public void keyPressed(KeyEvent e)
        {
                switch(e.getKeyCode())
                {
                        case KeyEvent.VK_RIGHT: keyRight = true; break;
                        case KeyEvent.VK_LEFT: keyLeft = true; break;
                        case KeyEvent.VK_UP: keyUp = true; break;
                        case KeyEvent.VK_DOWN: keyDown = true; break;
```

```
                }
        }

        public void keyReleased(KeyEvent e)
        {
                switch(e.getKeyCode())
                {
                        case KeyEvent.VK_RIGHT: keyRight = false; break;
                        case KeyEvent.VK_LEFT: keyLeft = false; break;
                        case KeyEvent.VK_UP: keyUp = false; break;
                        case KeyEvent.VK_DOWN: keyDown = false; break;
                }
        }

        public void keyTyped(KeyEvent e)
        {
        }

        public void update()
        {
                if(keyRight)
                        //call a method in model that sets the destination to destX++
                if(keyLeft)
                        //call a method in model that sets the destination to destX--
                if(keyDown)
                        //do the same for destY++
                if(keyUp)
                        //do the same for destY--
        }
```

There are many ways you can accomplish the update above - I personally added four new methods in Model to handle this movement.

Also, add this line to your Game constructor to tell it that the Controller is in charge of handling key events:

```
this.addKeyListener(controller);
```

10. **Make the turtle move 4 times faster** by changing each "1" or "++" in Model.java to "4" or "+=4" (there will be 8 places you are updating this information!).

    Tweak the code so that the turtle does not overshoot its destination, then jump back and forth trying to get there. For example, instead of always moving 4 pixels to the right, like this:

    ```
    turtleX += 4;
    ```

    you could move 4 pixels to the right, or the distance to the destination, whichever is smaller, like this:

    ```
    turtleX += Math.min(4, destX - turtleX);
    ```

    (Do something similar for moving left, up, or down.)
    Note that the turtle should move at the same speed at the end of this step regardless of the input (mouse or keyboard).

    Be sure to test throughly and make sure your turtle doesn't fly off the edge of the screen due to a negative number issue!

11. **Clean up your code**. Add your **name**, **date**, and **assignment description** at the top of each of the .java files. Make sure your tabulations are consistent. Add comments, so you will remember what various things do (for your own benefit--we are not going to check for this). It is a good idea to group your code into "blocks" separated by blank lines, such that each block of lines performs one task. Then you can start each block with a comment describing the one task that the block

performs. If a particular line within a "block" requires annotation, add a comment for that line at the end of the line. Future assignments will build on top of this one, so it is worth your time to keep your code clean. Also, future assignments will not provide nearly as explicit instructions as this one, so be sure you understood what you did. **Read through the code and follow any instructions that are also given in there.**

12. **Submit your code.** Make a zip archive named Assignment2.zip containing only your source code and images in it, and no additional folders. If you don't know how to make a zip archive on your favorite operating system, try using Google to find some instructions.

    The zip archive should contain exactly six files: Game.java, Model.java, View.java, Controller.java, your build script, and turtle.png. It should **NOT** contain any generated files. For example, it should not contain any files with a ".class" extension, or any hidden _MACOSX folders. The build script should be named "build.bash" or "build.bat". **Be sure you have updated your build script to explicitly include *all* the files you are now compiling!**

    Before you submit your archive file, make sure it still works. Try unzipping your archive into another folder. Make sure it only contains the files expected. Run your script. Did it work? If not, figure out what is wrong and fix it. If your code does not work, the graders will not debug it--that's your job. Finally, go the course web site on Blackboard (https://learn.uark.edu/) and submit your zip archive for Assignment2.

13. Make sure your code meets each of these requirements. Please keep in mind that this is a very large class and the TAs do not have time to fix your code, rename files, etc.:
    1. Single zip file named Assignment2.zip (step 12)
        - Contains exactly and only Game.java, Controller.java, Model.java, View.java and build.bat (or build.bash), with no additional folders
        - Use a similar zip command as found in homework 1 for reference
    2. Build script has the correct commands to build all files explicitly and runs the code (step 7)
        - Windows (.bat) and Linux (.bash) are different!
    3. **The code compiles (image can be found)**
        - Do not skip testing in step 12!
    4. Name, the date, and the assignment description is at the top of the file (step 11)
    5. Turtle moves at the same speed regardless of input type (and doesn't fly across the screen at any time) (step 10)
    6. Read through the provided code and followed any directions found there (step 1)
    7. Using public and private correctly throughout the code
    8. The button removes itself when pressed (step 5)
    9. Background is blue and there is a turtle (step 6)
    10. Frame rate has been updated (step 8)

Answer to the challenge in step 8:

Local variables cannot be accessed in other methods. Member variables, however, can be accessed anywhere in the class. So, you need to change "model" and "view" to be member variables instead of local variables. Here is an example of a local variable:

```
class Game
{
        public Game()
        {
                Model model = new Model();
```

Here is an example of a member variable:

```
class Game
{
        private Model model;

        public Game()
        {
                model = new Model();
```

You need to make a similar change for "view" as well.