Name: ______________________________ Section: ____

ISTE-121 Java – Graphics
Lab 12c

Objectives:

Practice graphics with tracking the mouse, and what it takes to do some above beginner graphics. The objective of this lab is to have you practice graphics; learn about drawing points for making lines on the screen.

Step 1 – See Demo, Download & Look

Download the lab starter files from MyCourses. Do not compile yet. Look at the code to see what methods are in Mousy.java and the MousySizes interface. Ignore the Trajectory class for now. Many methods are defined in Mousy and an inner class. See what they are, **do not uncomment any code or jump ahead** until the instructions say so.

Step 2 – Compile & run

Now compile and run the code. Follow these steps to really see what is happening:

1. Move the mouse around the JFrame – not impressive, is it.
2. Click and drag the mouse around the JFrame – Hello! Freaky ugly
   You are getting the blur because paintComponent() did not call super.paintComponent(g), which did not call the super's update() method.
3. Add `super.paintComponent(g);`
   to the existing `public void paintComponent(Graphics g)` method in Mousy.java.
4. Compile and run again. Does it look better?
5. In MouseMove class answer what causes this to listen to the mouse?

    a. What mouse Listener is used? ______________________________

    b. What mouse Adapter is used? ______________________________

    c. What mouse method is used? ______________________________

    d. What method makes the paint happen? ______________________

Step 3 – Report the position of the mouse

Look at the MouseMove class. MouseMove class extends JPanel, when adding in the constructor a listener, it is added to the JPanel.

Within the addMouseMotionListener, below the STEP 3 comment and before the mouseDragged method, add this method: (variable counter is already defined for you)

```
public void mouseMoved(MouseEvent mEvent) {

  System.out.printf("mouseMoved() %d times to (%d,%d)%n",
          ++counter, mEvent.getX(), mEvent.getY() );
}
```

Compile and run. Now simply move the mouse around the JFrame, don't click and hold until you see the output from the mouseMoved() method.

You may comment these lines out later if they get too annoying, the mouseMoved() won't be needed for the rest of this lab.

Step 4 – Create the Blue and Red bases

Drawing some blue and red boxes on the screen is done in the paint method.
To remove all the calculations, but not all the learning, here is what you need to add to the paintComponent method. Remember to use the graphics object, like the "Hello!" uses it.

- Set the drawing color to `BLUE`
- Draw a rectangle, see Graphics class for the method definition. Then substitute the parameters for (x, y, width, height) with the following defined variables:

x	= `startX`
y	= `startY – BASE_HEIGHT`
width	= `BASE_WIDTH`
height	= `BASE_HEIGHT`

- Set the drawing color to `RED`
- Draw another rectangle with this information

x	= `BASE_X`
y	= `BASE_Y – BASE_HEIGHT`
width	= `BASE_WIDTH`
height	= `BASE_HEIGHT`

Compile and run again. Should have Hello when click/drag, a blue and red box.

Step 5 – Drawing lines

In the paint method, we want to draw a line from the starting X and Y (lower left corner of the blue box) to where the mouse is being dragged, or in this case, the Hello! word is being moved. The reason will be answered later.

Looking at the Graphics Javadocs, fine how to draw a simple line with two X, Y pairs. Use the following information for drawing the line. Place this in the paintComponent method. Compile/Run but don't click or move the mouse over the JFrame

```
x1 = startX
y1 = startY
x2 = x
y2 = y
```

We've added drawing a line, but why is it going where it is?

Should be obvious, the paintComponent() method was run when the frame was displayed. So the line was drawn with whatever X and Y were stored in the values.
A couple of ways to get rid of this line. Easiest is to set the 'x' and 'y' to 'startX' and 'startY', rather than zero, when the variables were defined as MouseMove attributes.

Now we get Hello! sitting in the blue box. Why? ____________________________

Comment out the Hello!, we don't need it any more.
Compile and run, should look good now.

Step 6 – Limit dragging to inside the blue box

Objective: We only want to draw the line, while the mouse is being dragged (click-drag) in the blue box.

In the mouseDragged method it gets the X, and Y values, then repaints. Add an "if" statement to limit the X, Y positions to inside the blue box. In the "if" place the existing repaint(). So you don't waste time on figuring out the provided code and variables, add this code.

```
if(x < startX+BASE_WIDTH && x>startX &&
   y > startY-BASE_HEIGHT && y < startY ) {
      repaint();     // <<< existing repaint()
}
```

You should have only ONE repaint() in the mouseDragged method.

Compile and test to see if the line drawing is limited to the blue box. Notice, once leaving the blue box, the drawing stops.

Step 7 – Use the Trajectory class – Now the real action begins!

Objective: Moving the mouse around is cool enough, but let's add a mouse listener that when the mouse is released, it shoots a projectile from the blue box to the red box.

In the MouseMove constructor, there is some STEP 7 CODE commented out. We will be adding another mouse listener similar to the addMouseMotionListener and MouseMotionAdapter above the STEP 7 CODE comment.

Between the STEP 7 CODE comment and the if statement below it, fill in by following these instructions:

- Add an `addMouseListener`, with an anonymous inner class with an object of `MouseAdapter`. (Follow the addMouseMotionListener in the code above)
- Method in MouseAdapter is `mouseReleased(MouseEvent me)`
- In the mouseReleased method add:
  ```
  int mx = me.getX();
  int my = me.getY();
  ```
- Following this code is the commented "if" statement which contains shoot(mx,my). Uncomment this at this time. And ensure the "if" statement is within the mouseReleased method.

Compile. Don't run, it won't do anything interesting.
If this does not compile, check the { } and ending the addMouseListener with });
Still doesn't compile? Ask for help.

Before running the code, look at the shoot() method. The angle of the line drawn in the blue box determines the angle of the shot. The length of the line determines the velocity.

Understand that the code does what the comments say, you don't have to know all the details at this time. You have the trajectory class for your use later.

Almost done… Objective: Draw the trajectory line from the blue box out.
The shoot method loaded two arrays, "pointsX" and "pointsY" with (x,y) points for drawing a polygon line in the JPanel. Add drawing a Polyline into the paint method (Javadocs for polyline is right above the rectangle drawing method), and use these variables:

xPoints = `pointsX`
yPoints = `pointsY`
nPoints = You figure how to supply how many points in the array.

Now compile and run.
Does there seem to be a delay of drawing? After the mouse was released, was a repaint issued? This would go after the shoot finished. Add this and compile/run again.

It should all work now.

Step 8 – Change Trajectory so the projectile stops at the red box's edge.

Lab 11c – Instructor / TA: ________________