1.00 Lecture 26

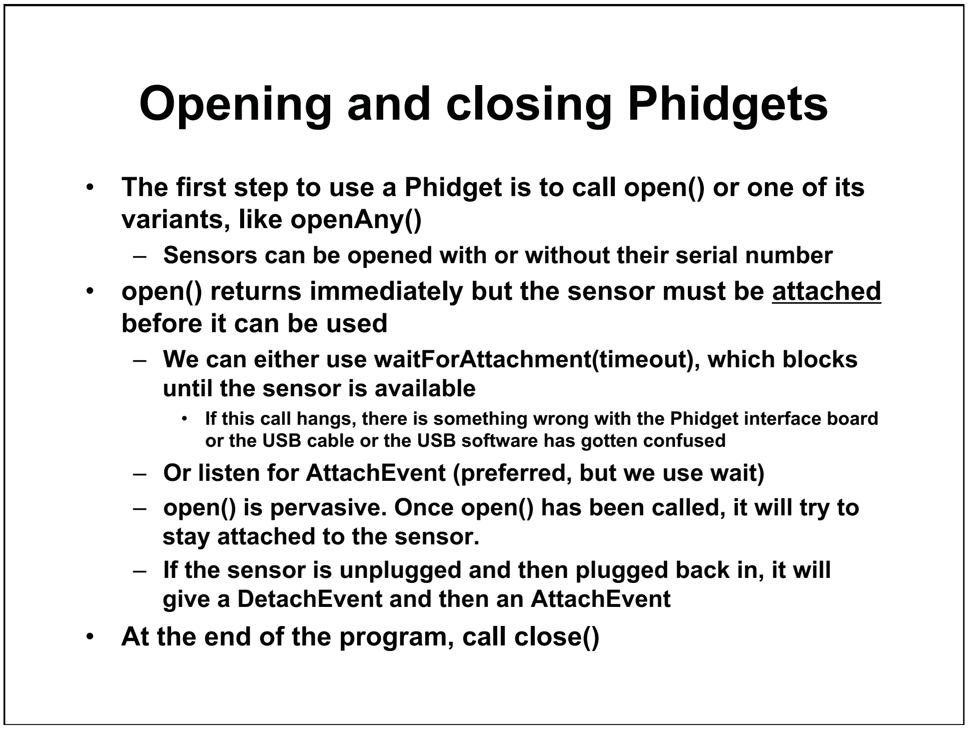
Introduction to Sensors (Phidgets) II

Reading for next time: None

Jar files

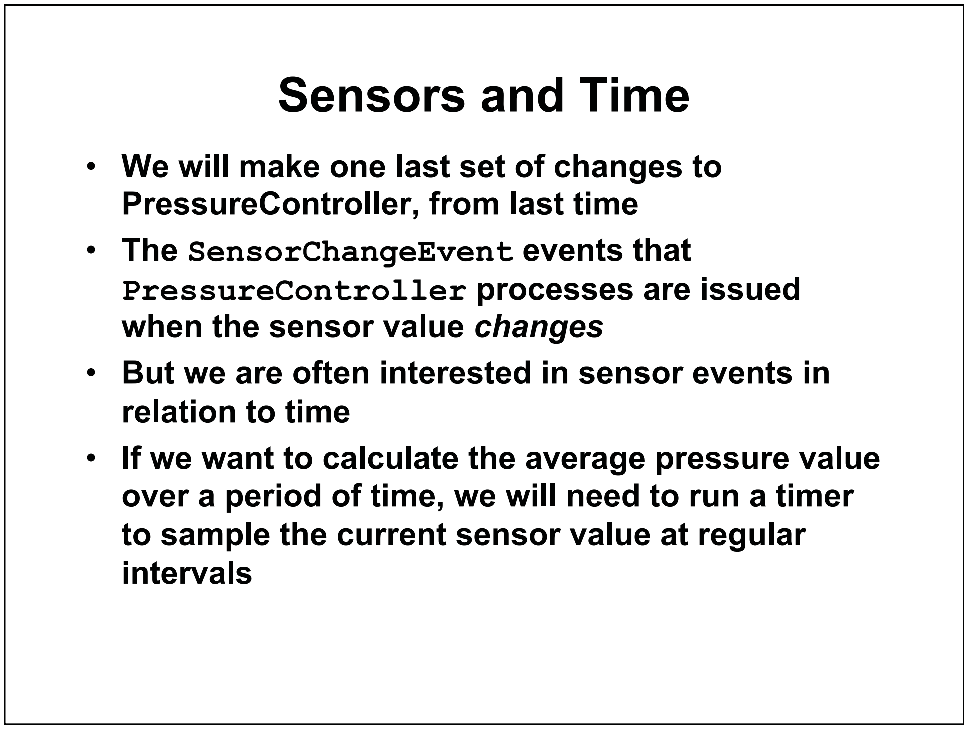
- You must put the phidgets21.jar file into the Java project for each lecture that uses Phidgets
  - And other projects in which you use Phidgets
- Steps:
  - Open the Java Properties/Java Build Path popup by right clicking on the project
  - Click "Add External Jars…" and navigate to where you unzipped the phidget21.jar file last time
  - Select it and click Open, and then OK

- Side note: Jar (Java archive) files contain Java .class files and are easy to create for GUI apps
  - Right click on project in Eclipse
  - Select Export; specify ‘launch configuration’ (which program with a main() to use) and destination (folder) to write .jar file
  - Try it after class with, e.g., BallController from lecture 22



Opening and closing Phidgets

- The first step to use a Phidget is to call open() or one of its variants, like openAny()
  - Sensors can be opened with or without their serial number
- open() returns immediately but the sensor must be attached before it can be used
  - We can either use waitForAttachment(timeout), which blocks until the sensor is available
    - If this call hangs, there is something wrong with the Phidget interface board or the USB cable or the USB software has gotten confused
  - Or listen for AttachEvent (preferred, but we use wait)
  - open() is pervasive. Once open() has been called, it will try to stay attached to the sensor.
  - If the sensor is unplugged and then plugged back in, it will give a DetachEvent and then an AttachEvent
- At the end of the program, call close()

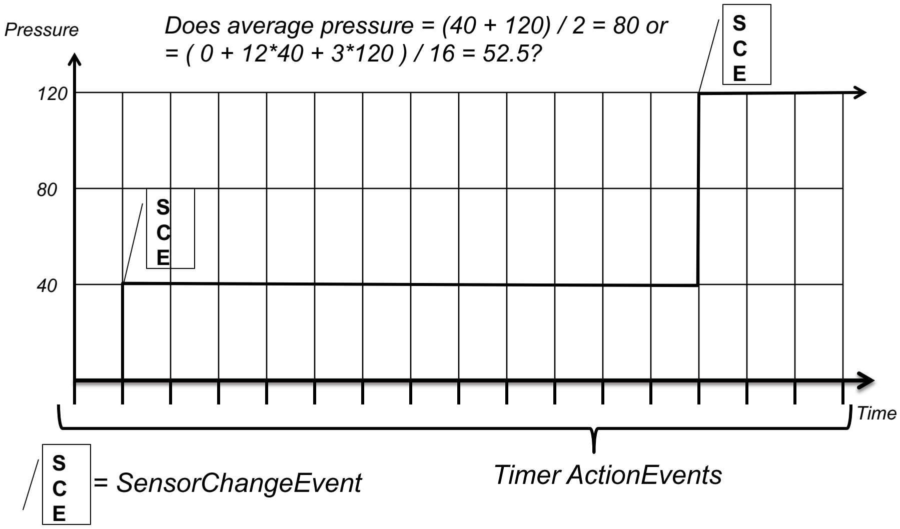


Sensors and Time

- We will make one last set of changes to PressureController, from last time
- The `SensorChangeEvent` events that `PressureController` processes are issued when the sensor value *changes*
- But we are often interested in sensor events in relation to time
- If we want to calculate the average pressure value over a period of time, we will need to run a timer to sample the current sensor value at regular intervals

Sensors and Time, 2

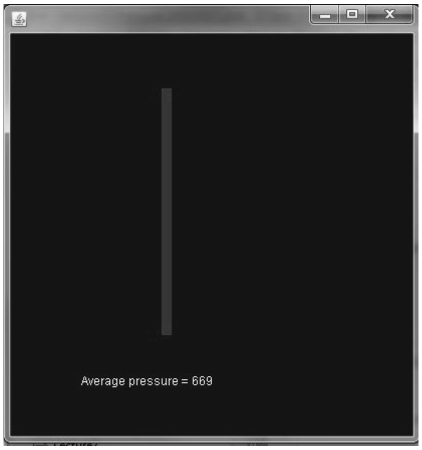
*Does average pressure = (40 + 120) / 2 = 80 or = (0 + 12*40 + 3*120) / 16 = 52.5?*



Pressure Averages

- **Next exercise based on `PressureAvgController1`**
  - **Based on PressureController1, the solution from last time**
  - **You will add code to sample the pressure every 10 milliseconds for 5 seconds (500 events)**
  - **And to calculate a running average that is displayed on the `PressureAvgView1` display**
- **It doesn't start the timer until the first non-zero pressure reading arrives, with value > 10**
- **Each `SensorChangeEvent` sets the current pressure value (pressure) and calls for a repaint of the display**
- **Each `ActionEvent` from the timer updates the average pressure and also calls for a repaint**

PressureAvgView1 Sample



```
Average pressure = 669
```

PressureAvgView1

```
public class PressureAvgView1 extends JPanel {
   private PressureController1 c;     // Reference to controller (MVC)
   public PressureAvgView1( PressureController1 pac ) {
     c= pac;
     setBackground(Color.BLUE);
     setPreferredSize(new Dimension(400,400));
   }
   public void paintComponent( Graphics g ) {
     super.paintComponent( g );
     Graphics2D g2= (Graphics2D) g;
     double x= 150;               // 150 pixels from upper left corner
     double height= ((double) c.getPressure()/1000.0) * 300;
     double width= 10;            // width of rectangle, x direction
     double y= 300 - height;      // top of rectangle, y direction
     Rectangle2D.Double rect= new Rectangle2D.Double(x,y,width,height);
     g2.setPaint( Color.red );
     g2.fill( rect );
     g2.setPaint( Color.white );
     g2.drawString("Average pressure= "+c.getAveragePressure(),70,350);
} }
```

Exercise 1a

- Download PressureAvgController1, and read it
  - It’s the solution (without LED) from the last lecture
- In PressureAvgController1, make the following changes:
  - Class declaration: implements ActionListener (for Timer)
  - Data members: Add:
    - int count: number of events processed. You will quit after 500 events.
    - long pressureSum. Initialize at 0, increment at each sensor reading. Double might be more convenient, but we often use ints with sensors
    - Timer timer
  - Constructor:
    - Create new Timer: events every 10 milliseconds, this as listener
  - Write getAvgPressure() method
    - Use pressureSum and count.
    - This will be called by PressureAvgView1
- Compile but don’t run this.

Exercise 1b

- In PressureAvgController1:
  - In sensorChanged() method:
    - if sensor value > 10, start the timer: timer.start()
    - (Extra calls to timer.start() have no effect. Or check timer.isRunning())
  - In closeIntfcKit() method: replace “closing…” with printing the average pressure to console. Use getAvgPressure()
- Compile but don’t run it yet.

Exercise 1c

- In PressureAvgController1:
  - Write actionPerformed() method to handle timer events
    - Increment count
    - Increment pressureSum
    - Repaint view
    - Call closeIntfcKit() when count = 500
- Compile and run this.

Exercise 2: Two sensors

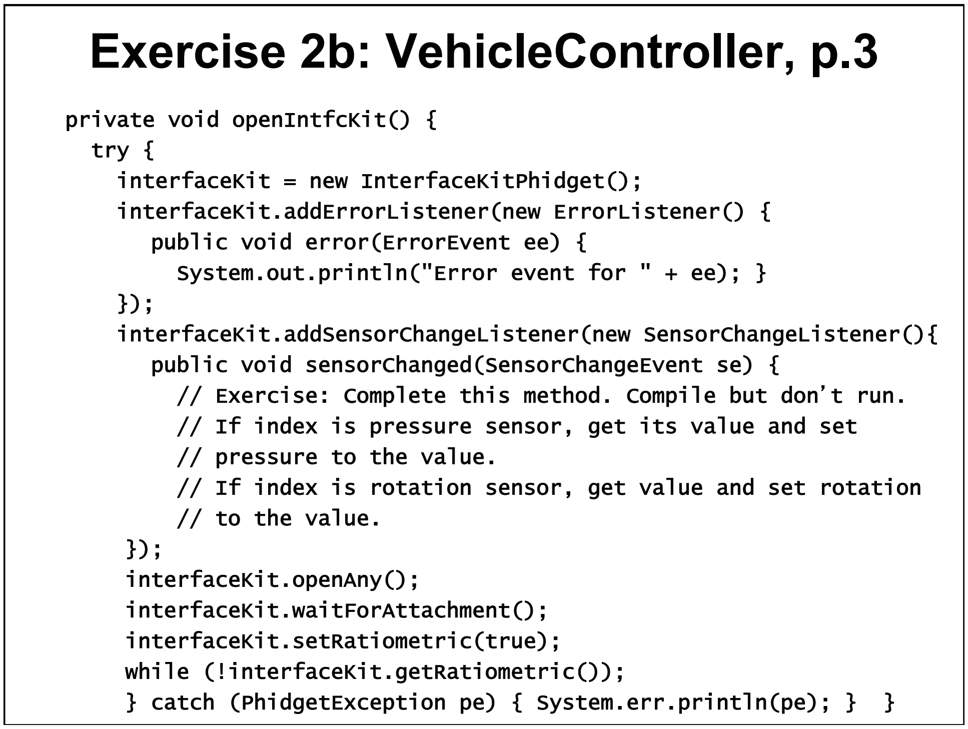
- Place rotation sensor on analog input 2
- Download:
  - VehicleController, VehicleModel, VehicleView
- Controller manages force and rotation sensor events to drive a simple vehicle. We’ll complete it.
  - Rotation sensor controls steering
  - Pressure sensor controls velocity
- View shows vehicle direction, speed, path. It’s complete.
  - Vehicle displayed as icon using Path2D
  - Must stay within display boundaries of view
- Model computes changes in speed, direction from sensor inputs. We will complete this.
  - Vehicle must be able to stop
  - Vehicle can’t turn if it’s not moving

VehicleController

```
import com.phidgets.*;
import com.phidgets.event.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class VehicleController extends JFrame implements
  ActionListener {
   private InterfaceKitPhidget interfaceKit;
   private VehicleView  view;
   private VehicleModel model;
   private int pressure= 0;
   private int pressureIndex= 1; // Sensor on analog 1
   private int rotation= 0;
   private int rotationIndex= 2; // Sensor on analog 2
   private Timer tick;           // Timer to update GUI
   public static final int WIDTH= 800;  // Size of view, model
   public static final int HEIGHT= 800;
```

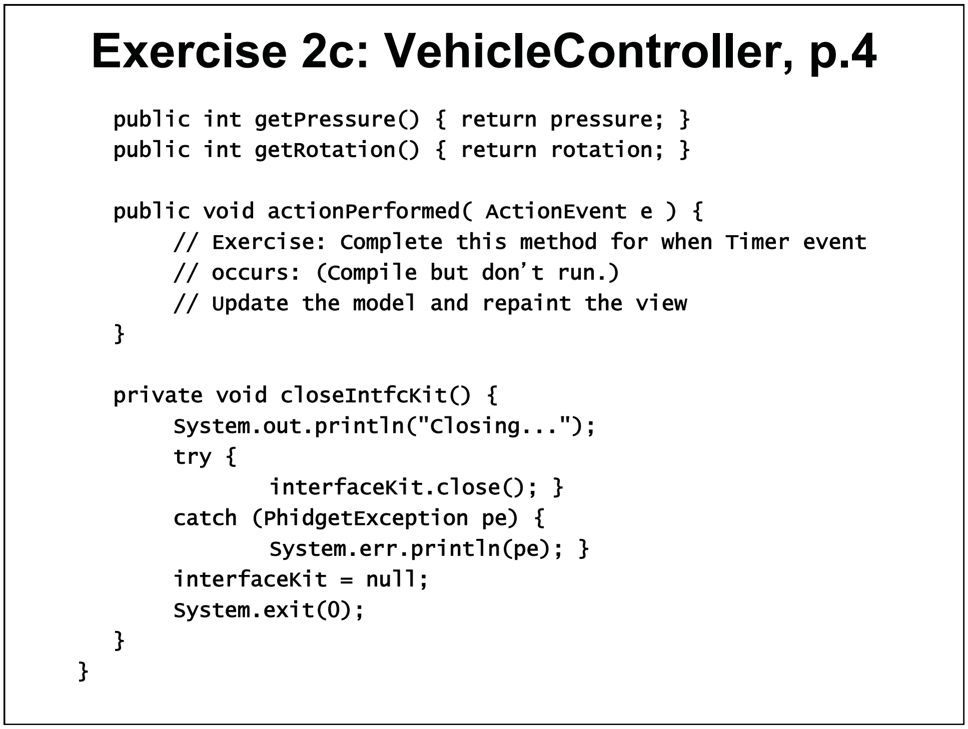
Exercise 2a: VehicleController, p.2

```
   public static void main(String[] args) {
        VehicleController vc = new VehicleController();
        vc.pack();
        vc.setVisible(true);
        vc.openIntfcKit();
   }

   public VehicleController() {
        model= new VehicleModel(this, WIDTH, HEIGHT);
        view = new VehicleView(model, WIDTH, HEIGHT);
        Container c= getContentPane();
        c.add(view, BorderLayout.CENTER);
        addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent we) {
                        closeIntfcKit();
                }
        });
      // Exercise: Create Timer with events every 0.05 seconds
      // using VehicleController as listener, and start it
   }  // Compile it but don’t run it
```



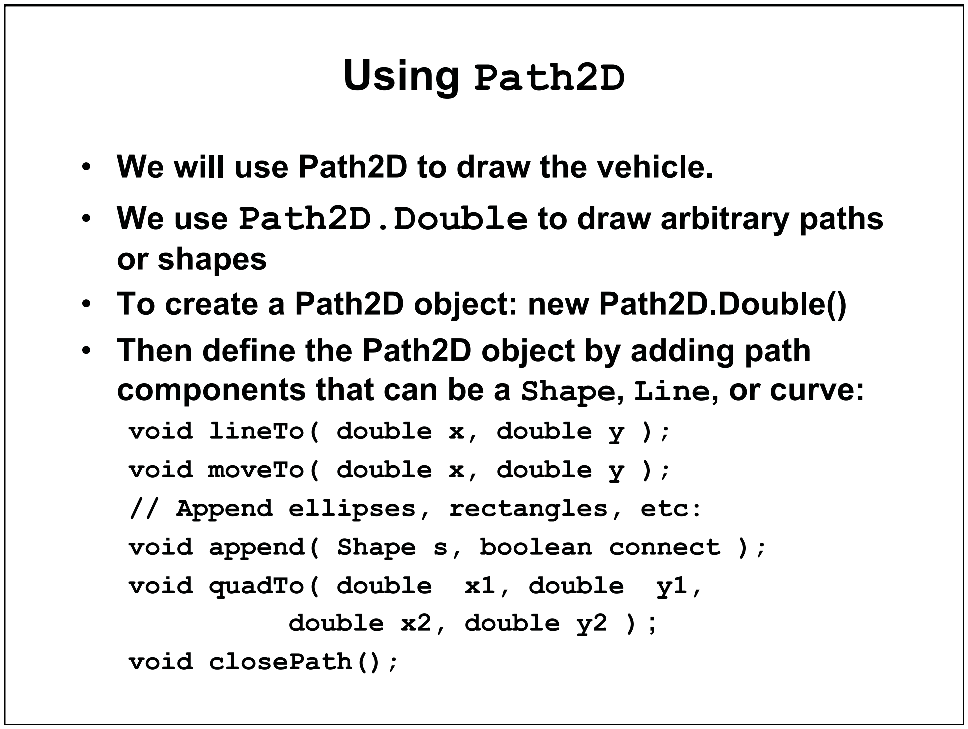
Exercise 2b: VehicleController, p.3

```
private void openIntfcKit() {
  try {
    interfaceKit = new InterfaceKitPhidget();
    interfaceKit.addErrorListener(new ErrorListener() {
       public void error(ErrorEvent ee) {
         System.out.println("Error event for " + ee); }
    });
    interfaceKit.addSensorChangeListener(new SensorChangeListener(){
       public void sensorChanged(SensorChangeEvent se) {
         // Exercise: Complete this method. Compile but don't run.
         // If index is pressure sensor, get its value and set
         // pressure to the value.
         // If index is rotation sensor, get value and set rotation
         // to the value.
     });
     interfaceKit.openAny();
     interfaceKit.waitForAttachment();
     interfaceKit.setRatiometric(true);
     while (!interfaceKit.getRatiometric());
     } catch (PhidgetException pe) { System.err.println(pe); }  }
```



Exercise 2c: VehicleController, p.4

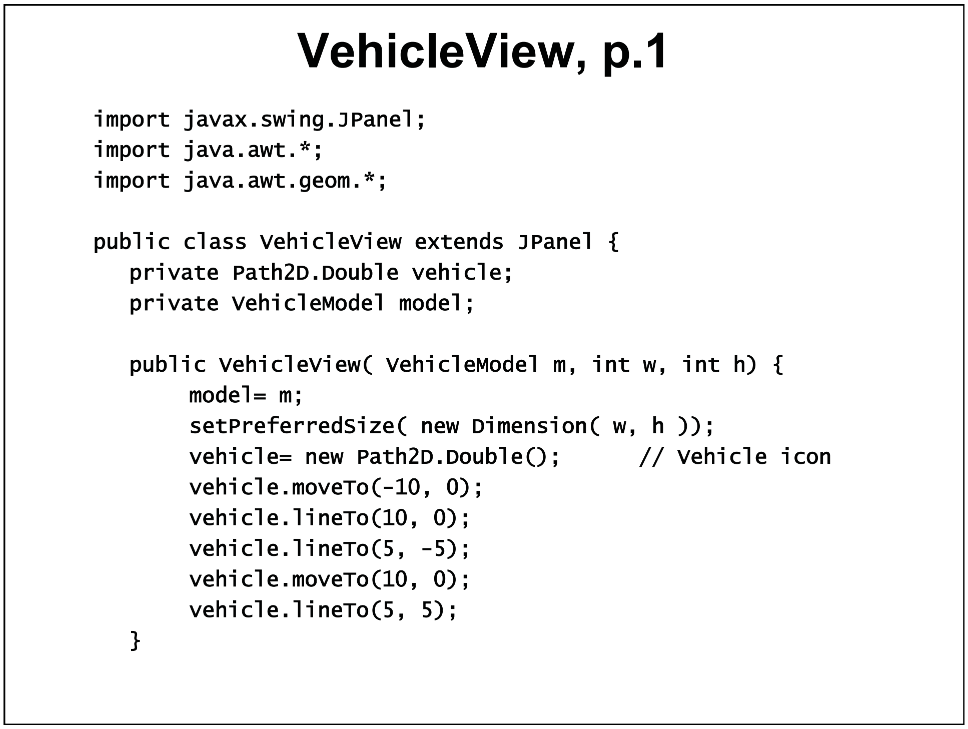
```
   public int getPressure() { return pressure; }
   public int getRotation() { return rotation; }

   public void actionPerformed( ActionEvent e ) {
        // Exercise: Complete this method for when Timer event
        // occurs: (Compile but don't run.)
        // Update the model and repaint the view
   }

   private void closeIntfcKit() {
        System.out.println("Closing...");
        try {
                interfaceKit.close(); }
        catch (PhidgetException pe) {
                System.err.println(pe); }
        interfaceKit = null;
        System.exit(0);
   }
}
```



Using `Path2D`

- **We will use Path2D to draw the vehicle.**
- **We use `Path2D.Double` to draw arbitrary paths or shapes**
- **To create a Path2D object: new Path2D.Double()**
- **Then define the Path2D object by adding path components that can be a `Shape`, `Line`, or curve:**

```
void lineTo( double x, double y );
void moveTo( double x, double y );
// Append ellipses, rectangles, etc:
void append( Shape s, boolean connect );
void quadTo( double  x1, double  y1,
          double x2, double y2 ) ;
void closePath();
```



VehicleView, p.1

```
import javax.swing.JPanel;
import java.awt.*;
import java.awt.geom.*;

public class VehicleView extends JPanel {
   private Path2D.Double vehicle;
   private VehicleModel model;

   public VehicleView( VehicleModel m, int w, int h) {
        model= m;
        setPreferredSize( new Dimension( w, h ));
        vehicle= new Path2D.Double();      // Vehicle icon
        vehicle.moveTo(-10, 0);
        vehicle.lineTo(10, 0);
        vehicle.lineTo(5, -5);
        vehicle.moveTo(10, 0);
        vehicle.lineTo(5, 5);
   }
```

VehicleView, p.2

```
   public void paintComponent( Graphics g ) {
        super.paintComponent( g );
        Graphics2D g2= (Graphics2D) g;
        g2.setPaint( Color.blue );
        g2.setStroke(new BasicStroke(2));

        // No explicit AffineTransform. Use Graphics2D methods
        g2.translate(model.getVehicleX(), model.getVehicleY());
        g2.rotate( model.getVehicleDir() );
        g2.draw( vehicle );
   }
}
```

VehicleModel, p.1

```
public class VehicleModel {       // VehicleModel1 in solution
   private int width;                       // 800
   private int height;                      // 800
   private double vehicleX;
   private double vehicleY;
   private double vehicleDir;               // Radians
   private double speed= 0;
   private double speedF= 0.0005;           // Scale factor
   private int speedThreshold= 10;          // Min sensor value
   private double directionF= 0.0005;       // Scale factor
   private int directionCtr= 500;           // Center of rotate
   private VehicleController sensors;       //  sensor (0-1000)

   public VehicleModel(VehicleController vs, int w, int h) {
        sensors= vs;
        width= w;
        height= h;
        vehicleX= width/2;           // Place in center of view
        vehicleY= height/2;          // which is also center of
        vehicleDir= 0;               // area vehicle can drive in
   }    // And getVehicleX(), getVehicleY(), getVehicleDir()
```

Exercise 2d: VehicleModel, p.2

- **Complete updateModel() in VehicleModel, which is called when an event occurs:**

```
public void updateModel() {
     int p= sensors.getPressure();  // 0-1000
     int r= sensors.getRotation();  // 0-1000, 0-300 degrees
     // Complete this method.
}
```

  - **Check if pressure sensor value above speedThreshold**
  - **If so, set speed= pressure times scale factor (speedF)**
  - **Set vehicle direction= f(rotation sensor) * speed *scale factor**
    - **This is the trickiest part. Experiment, or use:**
    - **vehicleDirection -= (r - directionCtr) * speed * directionF**
  - **Increment vehicle x position by speed * cos(direction)**
  - **Increment vehicle y position by speed * sin(direction)**
  - **Make sure vehicle x and y are between 0 and width or height**
  - **If pressure sensor less than speedThreshold, set speed = 0**
- **Compile and run it.**