CS 444 - Project 2 - Experience with More leJOS Classes

Experiment:

Can you get a better feel for some of these leJOS classes and methods entering pre-written classes as a team, and then modifying them? (And, of course, running them on your team's robot, debugging, etc.)

Method:

For each provided class below, one team member should be the **reader**, one team member should be the **typer**, and one team member should be the **explainer**.

- The **reader** reads the next line of code out loud; the **typer** types it in; the **explainer** says what it is doing. All look out for typos, discuss any questions/comments any team member has along the way.
- Team members may switch roles as often as the team would like.
- For modifications, the team should work in a "normal" trio-programming mode (one typing, two saying what to type).

NOTE: you need to give a proper javadoc comment for every class and method. Additional internal comments below are for explanation, and you may CHOOSE which, if any, you would like to include as you type them in; you may also add ADDITIONAL internal comments!

Stage 1 - ButtonEventPlay plus

Purpose: to see the Java 1.1 event handling model in action and to see how you can specify actions to be done when an NXT button is acted upon.

- Using the method above, implement `ButtonEventPlay.java` below. Include all present team members' names in an:

  `@author adapted by ...`

  in the class' javadoc comment.
- Verify that it works; try it out.
- MODIFY it in some interesting way of your choice; verify that it works as you would like.
- Demonstrate it to me, and make sure I check off that you have done so.
  - [You MAY move on to Stage 2 while you are waiting for me to get to you.]
- Submit your resulting `ButtonEventPlay.java` with a homework number of **21**.

```
import lejos.nxt.*;
import lejos.util.*;


/**
  application class in which each button
  push causes an identifying string to be printed
  to the NXT screen

  @author Sharon Tuttle
  @version 2015-02-15
**/

public class ButtonEventPlay
{
    /**
      set up buttons to print an identifying string
      to the NXT screen when they are pressed, and
      run for about 30 seconds

      @param args not used
    **/

    public static void main(String[] args)
    {
        System.out.println("Button Play");
        System.out.println("RUNS 30 SEC");
        System.out.println("click buttons!");

        // creating an instance of this class, from which to
        //     call the desired non-static method
        //     from this static main method

        ButtonEventPlay buttonEventPlayInstance = new ButtonEventPlay();
        buttonEventPlayInstance.setUp();

        // uh oh, how make this QUIT effectively? ...trying just
        //     ending program after about 30 more seconds
        // (don't want a button push to end program THIS time!)

        Delay.msDelay(30000);
    }
```

```
/**
  set up the buttons so they can identify themselves
  to the screen when pressed and released
**/

private void setUp()
{
    // set a button listener for each button
    //    on the front of the NXT brick

    Button.ENTER.addButtonListener(new ButtonAction("Enter"));
    Button.ESCAPE.addButtonListener(new ButtonAction("Escape"));
    Button.LEFT.addButtonListener(new ButtonAction("Left"));
    Button.RIGHT.addButtonListener(new ButtonAction("Right"));
}

/**
    private inner class, to set up ButtonListeners

    (since implements ButtonListener, MUST implement
    methods buttonPressed and buttonReleased with the
    headers shown -- BUT OK for it to have OTHER
    methods/data in addition, here demo'd by its having
    a non-default constructor and a data field)

**/

private class ButtonAction implements ButtonListener
{
    // data field

    private String displayLabel;

    /**
      set the desired button label for this button listener

      @param desiredLabel the desired button display label
```

```
        **/

        public ButtonAction(String desiredLabel)
        {
            this.displayLabel = desiredLabel;
        }

        /**
           when "sensitive" button is pressed,
           clear the screen, and
           print "[its label] PRESSED" on line 0 of the screen

           @param b the calling button
        **/

        public void buttonPressed(Button b)
        {
            LCD.clear();
            LCD.drawString(this.displayLabel + " PRESSED", 0, 0);
        }

        /**
           when "sensitive" button is released,
           print "[its label] RELEASED" on line 1 of the screen

           @param b the calling button
        **/

        public void buttonReleased(Button b)
        {
            LCD.drawString(this.displayLabel + " RELEASED", 0, 1);
        }
    }
}
```

Stage 2 - SquareTracer plus

Purpose: to see some of the `DifferentialPilot` methods in action; to see your robot ALMOST travel in a square path...

- Using the method given at the beginning of the handout, implement `SquareTracer.java` below. Include all present team members' names in an:

  `@author adapted by ...`

  in the class' javadoc comment.
- Verify that it works; try it out.
- MODIFY it in some interesting way of your choice; verify that it works as you would like.
- Demonstrate it to me, and make sure I check off that you have done so.
  - [You MAY move on to Stage 2 while you are waiting for me to get to you.]
- Submit your resulting `SquareTracer.java` with a homework number of **22**.

```
import lejos.nxt.*;
import lejos.robotics.navigation.*;

/**
 * travel in a square path
 *
 * @author www.lejos.org - Roger
 * @author adapted by Sharon Tuttle
 * @version 2015-02-17
 */

public class SquareTracer
{
    // data field

    private DifferentialPilot pilot;

    /**
     * travel in a square path, where each side of the
     * square is a given length
     *
     * param sideLength length of a side of the square path to
     *     travel
     */

    public void  drawSquare(float sideLength)
    {
        for(int i=0; i<4; i++)
        {
            pilot.travel(sideLength);
            pilot.rotate(90);
        }
    }

    /**
     * create a pilot and have it travel a square path
     *
     * @param args not used
     */

    public static void main(String[] args)
    {
        SquareTracer sqTracer = new SquareTracer();
        sqTracer.pilot = new DifferentialPilot(56f, 120f, Motor.B, Motor.C);
        System.out.println("Press Button");
        Button.waitForAnyPress();

        for (int i=0; i<5; i++)
        {
            LCD.clear();
            sqTracer.drawSquare(100 * i);

            System.out.println("Press Button");
            Button.waitForAnyPress();
        }
    }
}
```

Stage 3 - TravelTest plus

Purpose: to try out using a touch sensor, to see some of the `DifferentialPilot` methods in action, and to see your robot (maybe) react based on sensor input.

- In the basic robot instruction manual, we want to KLUGE a set of the directions given to add a touch sensor to your current robot.

- GO TO pages **32-33** of the booklet.
- USING A TOUCH SENSOR INSTEAD OF THE REFLECTED LIGHT SENSOR SHOWN, do ONLY substeps 1 through **5** of step 22.
- then, turn to pages **37-38** in the booklet, and attach the touch sensor as shown on steps 24-25 (facing forward, NOT downward).
- Make sure you note to which sensor port you happen to connect it... 8 -)

• Using the method given at the beginning of the handout, implement `TravelTest.java` below. Include all present team members' names in an:

  `@author adapted by ...`

  in the class' javadoc comment.

• Verify that it works; try it out.

• MODIFY it in some interesting way of your choice; verify that it works as you would like.

• Demonstrate it to me, and make sure I check off that you have done so.
  - [You MAY move on to Stage 4 while you are waiting for me to get to you.]

• Submit your resulting `TravelTest.java` with a homework number of **23**.

```
import lejos.nxt.*;
import lejos.robotics.navigation.*;

/**
    Robot that, three times:
    goes forward a set distance
    UNLESS it hits something on the way,
    in which case it backs up and stops

    @author www.lejos.org
    @author adapted by S. Tuttle
    @author impl'd by <present team member names>
    @version 2015-02-15 (completed after class)
**/

public class TravelTest
{
    // data fields

    private DifferentialPilot pilot;
    private TouchSensor bump = new TouchSensor(SensorPort.S3);
```

```
/**
   each time this is called, go forward either 1000
   millimeters or until bumping into something
**/

public void go()
{
    // try to travel 1000 mm (because called DifferentialPilot
    //    constructor in main with measurements given in mm)
    // (BUT return control to this method immediately)

    pilot.travel(1000, true);

    // if touch sensor registers as bumped before
    //     1000 mm traveled, go backward 200 mm
    //     and stop

    while (pilot.isMoving())
    {
        if (bump.isPressed())
        {
            System.out.println("PRESSED!: " +
               pilot.getMovement().getDistanceTraveled());

            // go backward 200 mm and stop

            pilot.travel(-200);
            pilot.stop();
        }
    }

    System.out.println(
        pilot.getMovement().getDistanceTraveled());
    Button.waitForAnyPress();
}
```

```
    /**
     set up an instance of THIS class
     and make it go 3 times

     @param args not used
    */

    public static void main(String[] args)
    {
        TravelTest traveler = new TravelTest();

        // happening to give wheel diameter and track distance
        //    (distance between center of the two wheel)
        //    in millimeters; the f below means I want float
        //    versions of those named constants, which is the type
        //    the constructor wants for those)
        // whatever units you use for these HERE will affect
        //    the units assumed in other DifferentialPilot
        //    methods such as travel (used in go method above)

        traveler.pilot =
            new DifferentialPilot( 56f, 120f,
                   Motor.B, Motor.C);
        System.out.println("about to call go 1st time");
        Button.waitForAnyPress();
        traveler.go();

        System.out.println("about to call go 2nd time");
        Button.waitForAnyPress();
        traveler.go();

        System.out.println("about to call go 3rd time");
        Button.waitForAnyPress();
        traveler.go();
    }
}
```

Stage 4 - ObjectDetectPlay plus

Purpose: to try out using an ultrasonic sensor, and to learn a bit about the `FeatureListener` interface and the `RangeFeatureDetector` class (both from the leJOS package `lejos.robotics.objectdetection`)

- In the basic robot instruction manual, we want to add an ultrasonic sensor to your current robot.
  - GO TO pages **28-30** of the booklet.
  - do steps 20-21.
  - Make sure you note to which sensor port you happen to connect it... 8 -)
- Using the method given at the beginning of the handout, implement `ObjectDetectPlay.java` below. Include all present team members' names in an:

  `@author adapted by ...`

  in the class' javadoc comment.
- Verify that it works; try it out.
- MODIFY it in some interesting way of your choice; verify that it works as you would like.
- Demonstrate it to me, and make sure I check off that you have done so.
  - [develop additional class(es) while you are waiting for me to get to you.]
- Submit your resulting `ObjectDetectPlay.java` with a homework number of **24**.

```
import lejos.nxt.*;

/* note the new package involved! */

import lejos.robotics.objectdetection.*;

/**
   experiment with the Ultrasonic sensor, AND the FeatureListener
   interface and the RangeFeatureDetector class (both from leJOS
   package lejos.robotics.objectdetection)

   @author www.lejos.org
   @author adapted by Sharon Tuttle
   @version 2015-02-17
*/

public class ObjectDetectPlay
```

```
{
    // data field

    /*
       only try to detect features within 80 -- what? what units? cm?
       away
    */

    private static final int MAX_DETECT = 80;

    /**
       when its ultrasonic sensor detects a feature --
       play a sound, and display its distance away?

       @param args not used
    */

    public static void main(String[] args)
    {
        System.out.println("Sensor Play");

        ObjectDetectPlay objectDetectInstance = new ObjectDetectPlay();
        objectDetectInstance.setUp();

        System.out.println("click Enter to quit");
        Button.ENTER.waitForPressAndRelease();
    }

    /**
       make ultrasonic sensor sensitive to detected features
       within a certain range
    */

    private void setUp()
    {
        /*
           create an UltrasonicSensor object based on the
```

```
              sensor port to which the sensor is connected
       */

     UltrasonicSensor ultraSensor = new UltrasonicSensor(SensorPort.S2);

       /*
           RangeFeatureDetector constructor expects:
           *   a range finder (here, the ultrasonic sensor object)
           *   the maximum distance to report a detection (in cm?)
           *   the delay between scanning the sensor (in millisec?)
       */

       RangeFeatureDetector featDetector = new RangeFeatureDetector(
                                               ultraSensor, MAX_DETECT, 500);

       /*
          add a FeatureListener to specify what should happen when
          a feature is detected by the ultrasonic sensor
       */

       featDetector.addListener(new FeatureDetected());
   }


   /**
      when a feature is detected, play a tone and display
      its distance from the ultrasonic sensor on the NXT screen
   */

   private class FeatureDetected implements FeatureListener
   {
       /**
         this will be called when a sensitive UltraSonic sensor
         is in range of a feature, playing a tone and displaying how far
         away that feature is

         <p> this method is required for the FeatureListener interface </p>
```

```
          @param feature detected feature
          @param detector ultrasonic sensor
        */

        public void featureDetected(Feature feature,
                                    FeatureDetector detector)
        {
            // how far away is the detected feature?

            int range = (int)feature.getRangeReading().getRange();

            // play a tone and display detected feature's distance

            Sound.playTone(1200 - (range * 10), 100);
            System.out.println("Range:" + range);
        }
    }
}
```

Stage 5 - Class Demonstrations

As a team, EITHER:

- select the team's favorite of your modified versions from Stages 1-4 to demonstrate to the class, OR
- IF you have time, create a NEW class using some of the features/classes you practiced with in Stages 1-4 to demonstrate to the class.
  - IF you choose this option, be sure the class includes all of the participating team members' names, and submit it with a homework number of **25** prior to the class demonstration.

At a class time to be determined:

- each team will demonstrate their selected class running on their team's robot
- each team member should plan to briefly tell the class about one of the methods being used in their program (what the method's name is, what class it is in, and what it is doing in their program).