



# Myro/Java

Douglas Harms, DePauw University

IPRE Winter Workshop - Georgia Tech

8 January 2011

# Overview and Goals

- Implement the Myro API in Java
- Maintain the flavor of both Myro and Java
- Rewrite *Learning Computing with Robots* for Java
- Develop assignments and labs
- Offer section of CSI using Myro/Java in Fall 2011

# Current Status

- Movement
- Sensors
- Camera and imaging
- Gamepad and joystick
- Simple GUI
  - dialog boxes
  - key/mouse events

# Movement

```
void forward()
```

Starts the Scribbler moving forward at full speed with no rotational movement.

```
void forward(double speed)
```

Starts the Scribbler moving forward at a specified speed with no rotational movement.

```
void forward(double speed, double numSeconds)
```

Moves the Scribbler in a forward direction at a specified speed with no rotational movement for a specified amount of time.

# Movement - Specifications

```
public void forward(double speed, double numSeconds)
```

Moves the Scribbler in a forward direction at a specified speed with no rotational movement for a specified amount of time. The Scribbler will stop moving at the end of the specified time period. This method will not return until the specified time period has occurred.

**Precondition:** scribblerConnected, speed between -1.0 (inclusive) and 1.0 (inclusive), numSeconds > 0.0

**Parameters:** speed - Specifies the forward speed. Positive values specify forward movement (1.0 is full forward speed), negative values specify backward movement (-1.0 is full backward speed). numSeconds - Specifies the length of time to move, in seconds.

# Sensors

```
public int getObstacle(int whichSensor)
```

Read one of the Fluke's IR obstacle sensors.

**Precondition:** flukeConnected, whichSensor is SENSOR\_IR\_LEFT (or 0), SENSOR\_IR\_CENTER (or 2), or SENSOR\_IR\_RIGHT (or 1)

**Parameters:** whichSensor - Selects the Fluke IR sensor. Should be SENSOR\_IR\_LEFT (or 0), SENSOR\_IR\_CENTER (or 2), or SENSOR\_IR\_RIGHT (or 1)

**Returns:** The value of the selected sensor. A low value means there are no obstacles detected, a high value means there is an obstacle detected. The return value is in the range 0..6400.

# Sensor Constants

```
static int SENSOR IR CENTER
```

Constant passed to getObstacle to select the center IR sensor.

```
static int SENSOR IR LEFT
```

Constant passed to getIR or getObstacle to select the left IR sensor.

```
static int SENSOR IR RIGHT
```

Constant passed to getIR or getObstacle to select the right IR sensor.

# Camera

```
public Myro.MyroImage takePicture(int imageType)
```

Takes a picture with Fluke's camera. The imageType parameter determines what kind of picture is taken.

**Precondition:** flukeConnected, imageType is IMAGE\_COLOR (or 0), IMAGE\_GRAY (or 1), or IMAGE\_BLOB (or 2)

**Parameters:** imageType - Specifies the type of picture to take: IMAGE\_COLOR, IMAGE\_GRAY, or IMAGE\_BLOB

# Images

- `java.awt.Color get(int x, int y)`  
Returns the RGB color of pixel (x, y).
- `int getGray(int x, int y)`  
Returns the grayscale value of pixel (x, y).
- `int getType()`  
Returns the type (i.e., Color or Grayscale) of this image.
- `int height()`  
Returns the height of the image.
- `int width()`  
Returns the width of the image.

# Images

- void set(int x,int y,java.awt.Color color)  
Sets the color of pixel (x,y) to an RGB color.
- void setGray(int x, int y, int grayLevel)  
Sets the color of pixel (x,y) to a grayscale color.
- void show(int x, int y)  
Causes this image to be visible in a window located at a specified location on the screen.
- void hide()  
Causes this image to be invisible (i.e., the window disappears from the screen).

# Images

- Myro.MyroBlobSpec getUserDefinedBlob()  
Allows the user to select a rectangular area of the image used to define a blob.
- Myro.MyroBlobSpec defineBlob(int xlow, int ylow, int width, int height)

Calculate a blob based on a rectangular area of the image.

- void save(java.lang.String filename)  
Save the image to a file.

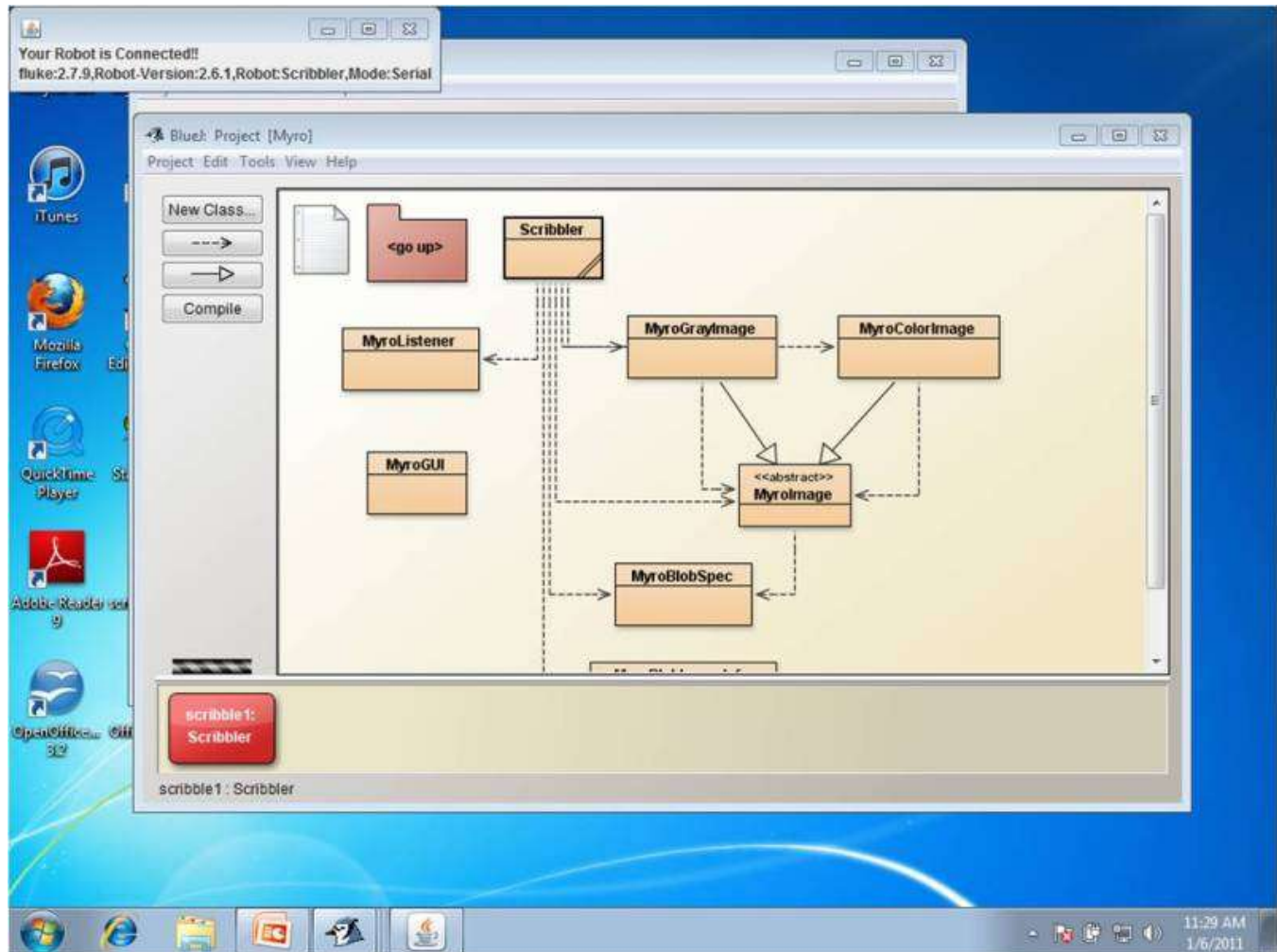
# Miscellaneous

- boolean connect(java.lang.String portName)  
Connect the Scribbler to port portName.
- void close()  
Close the connection between the computer and the Scribbler.
- boolean portOpened()  
Indicates whether the port connecting to the robot has been opened.
- boolean scribblerConnected()  
Returns whether the scribbler is currently connected.
- boolean flukeConnected()  
Indicates whether the robot has a Fluke board attached to it.

# Miscellaneous

- void gamepad()  
Opens a window that permits the user to control the movement of the Scribbler.
- void joyStick()  
Opens a window that permits the user to control the movement of the Scribbler.
- void senses()  
Opens a window that continually displays the sensor values of the Scribbler and/or Fluke.

# Programming – BlueJ Workbench



# Programming-BlueJ Workbench

The screenshot displays the BlueJ Workbench interface. On the left, a sidebar shows the project structure with a 'Project' pane. The main workspace is divided into two panes. The top pane shows a class hierarchy diagram with the following classes and relationships:

- MyroGrayImage** and **MyroColorImage** are subclasses of **MyroImage** (indicated by solid lines with hollow triangle heads).
- MyroImage** is an abstract class (indicated by <<abstract>> in its name).
- MyroImage** has a dashed dependency arrow pointing to **MyroBlobSpec**.
- MyroImage** has a dashed dependency arrow pointing to **MyroColorImage**.
- MyroImage** has a dashed dependency arrow pointing to **MyroGrayImage**.

The bottom pane shows a list of methods for the selected class, **MyroImage**. The methods are:

```
inherited from Object
void autoCamera()
void backward()
void backward(double speed)
void backward(double speed, double numSeconds)
void beep(double duration, int frequency)
void beep(double duration, int frequency1, int frequency2)
void close()
void configureBlob(MyroBlobSpec)
boolean connect(String portName)
void darkenCamera(int level)
boolean flukeConnected()
void forward(double speed)
void forward()
void forward(double speed, double numSeconds)
void gamepad()
double getBattery()
MyroBlobImageInfo getBlob()
int getBright(int whichSensor)
double getFudge()
boolean[] getIR()
boolean getIR(int whichIR)
String getInfo()
int getLight(int whichLight)
int[] getLight()
boolean[] getLine()
boolean getLine(int whichSensor)
String getName()
int getObstacle(int whichSensor)
boolean getStall()
more methods
Inspect
Remove
```

The desktop background is blue with a white arc. The taskbar at the bottom shows the system clock as 11:36 AM on 1/6/2011.

# Programming-BlueJ Codepad

The screenshot displays the BlueJ IDE interface. At the top, a status bar indicates "Your Robot is Connected!!" with details: "fluke:2.7.9, Robot-Version:2.6.1, Robot: Scribbler, Mode: Serial". The main workspace shows a class hierarchy diagram for the "BlueJ: Project [Myro]" project. The diagram includes the following classes and relationships:

- <go up>**: A red folder icon representing the project root.
- Scribbler**: A class box connected to the project root.
- MyroListener**: A class box connected to Scribbler.
- MyroGUI**: A class box connected to Scribbler.
- MyroBlobSpec**: A class box connected to Scribbler.
- MyroImage**: An abstract class box (labeled <<abstract>>) connected to Scribbler.
- MyroGrayImage**: A class box that inherits from MyroImage.
- MyroColorImage**: A class box that inherits from MyroImage.

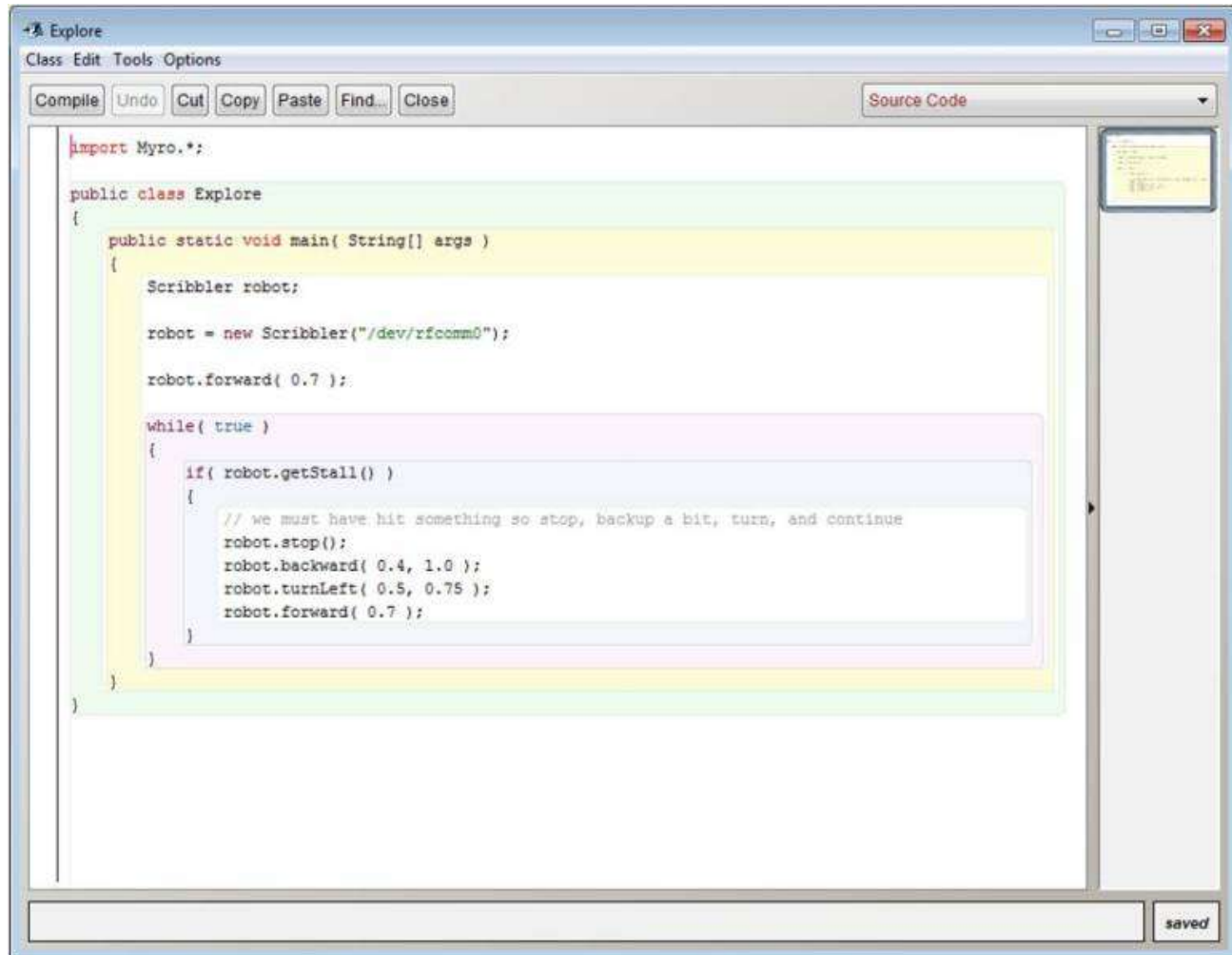
Relationships are shown with solid lines for inheritance (MyroImage to MyroGrayImage and MyroColorImage) and dashed lines for other associations (Scribbler to MyroListener, MyroGUI, MyroBlobSpec, and MyroImage).

Below the diagram, a code editor shows the following code snippet:

```
scribble1.takePicture(0).show();
```

The IDE also features a toolbar with "New Class...", "Compile", and other buttons, and a taskbar at the bottom showing the system clock as 11:43 AM on 1/6/2011.

# Programming-Java Program



The screenshot shows a Java IDE window titled "Explore". The menu bar includes "Class", "Edit", "Tools", and "Options". Below the menu bar is a toolbar with buttons for "Compile", "Undo", "Cut", "Copy", "Paste", "Find...", and "Close". A "Source Code" dropdown menu is visible on the right side of the toolbar. The main editor area contains the following Java code:

```
import Myro.*;

public class Explore
{
    public static void main( String[] args )
    {
        Scribbler robot;

        robot = new Scribbler("/dev/rfcomm0");

        robot.forward( 0.7 );

        while( true )
        {
            if( robot.getStall() )
            {
                // we must have hit something so stop, backup a bit, turn, and continue
                robot.stop();
                robot.backward( 0.4, 1.0 );
                robot.turnLeft( 0.5, 0.75 );
                robot.forward( 0.7 );
            }
        }
    }
}
```

A "saved" button is located in the bottom right corner of the IDE window.

# Myro/Java Code

- Available using svn from <http://svn.cs.brynmawr.edu/Myro/trunk/Java/>
- Repository contains Myro/Java code, binaries for libraries (e.g., serial), and installation instructions
- Tested on:
  - Windows XP – 32 and 64 bit
  - Windows 7 – 32 and 64 bit
  - MacOSX – 32 bit Tiger (10.4)
  - Linux – 32 and 64 bit Fedora
- Contact
  - Doug Harms (dharms@depauw.edu)



**Questions?**