Review of Java and Object Oriented Programming
What do I do?

- Associate Director Robotics Resource Center
- Responsible for the WPILib
  - Used for LabVIEW, C++, and now Java
- Involved with FIRST since 2001
- On the Game Design Software Sub-Committee
- One of the mentors for Team 190
What is WPI?

· Worcester Polytechnic Institute
  – Worcester, Massachusetts
· Undergraduate and graduate university with an emphasis on technology and engineering
Robotics Engineering at WPI

- 3rd largest major on campus based on incoming freshman
- Students learn Mechanical Engineering, Software Engineering, and Electrical Engineering

- I used to say, “It’s not about the robots”. Now it is!
This talk sponsored by
– Eric Arseneau
– CTO/Co-Founder
– eric@assetscience.com
Agenda

- What is it?
- Java
- Look at a sample Program
- Using the Tools
- More Java
- A few sample built in classes
- Differences C++ and Java
- Documentation …
- What’s new for 2011
Goals of Java for FRC

- Provide a safe text based language
- Reduce the barrier to entry
- Have a program environment that matches what students learn in school
- Simplify robot programming even more
What is it?

- Java implementation of the WPILib API for FRC Robots
  - We call it WPILibJ
- Sponsored by Sun Microsystems (Oracle)
- Demonstrated to FIRST in fall 2008
- Released to FRC teams for the 2010 season
- Started by
  - Derek White (Sun)
  - Eric Arseneau (Sun)
  - Brad Miller (WPI)
  - Students from WPI
Java based on Squawk

- Java Virtual Machine for small devices
  - Sun Labs Project – PI – Derek White
  - Java ME CLDC 1.1 IMP 1.0
- Sun Microsystems Labs project / mobile and embedded community project
  - https://squawk.dev.java.net
- Java Operating Environment for small devices
- Bootloader / Micro Kernel
WPILibJ

- Equivalent functionality as C++/LabVIEW
  - Teams can learn about and use object oriented programming techniques
- Framework to support competition programs
- Runs on top of VxWorks
  - Full support for Java threading features
- Parity between C/C++ and LabVIEW libraries
WPILib Features

· Currently supported devices through functions and C++ Classes (more on the way)
  – Access to LabVIEW camera / image processing library
  – High performance A/D converter
  – Speed controls with settable profiles
  – Gyros & Accelerometers
  – Driver Station I/O and Dashboard
  – Relays and solenoids
  – Counters, gear-tooth sensors and encoders
  – Timing
  – Driving based on robot definition
  – Arcade, tank, holonomic (Mecanum) steering
  – Compressor & pressure sensor
Everything is Open Source

- Community ownership - nothing proprietary
  - Allowing community contributors make for a more acceptable standard code base

- Nothing hidden “under the covers”
  - Gives teams a better understanding of how code works
  - Teams can now be more self supporting without relying on a single person/company for support

- Teams can easily extend and adapt code for their own needs
Why Objects?

- Objects are a good way of representing real-world “things”
  - Motors, sensors, driver station, robot drive train, robot subsystems, your whole robot
  - Have data associated with the “thing” and code that operates on the data (member data and methods)
  - Some objects are more specific versions of other objects (subclasses)
Object Oriented Programming

- C++ = C + Objects + other features
  - C++ programs that don’t use the extensions look like C programs
- C++ objects are similar to Java objects
  - Java taught in many high schools
  - Java used on programming AP exams
Classes

- Classes represent *templates* of information
  - Like blank forms
- Objects are filled out forms
  - Filling out the template (blank form) *instantiates* it
Comments are Important

- Comments document what your code does
- Should be in all programs so that you and others will remember how the

```c
/*
 * This code does this.
 */
```
import edu.wpi.first.wpilib.*;

- Imports classes from the WPILib package (collection of programs)
- The name of the package is the reverse of the Internet domain name
Class declarations

```java
public class MySimpleRobot {
    public void run() {
        move();
        turnLeft();
        move();
    }
}
```

· “MySimpleRobot” defines a new class
Adding new methods

- Adding a capability to MySimpleRobot
- run is a new method

```java
public class MySimpleRobot {
    public void run() {
        move();
        turnLeft();
        move();
    }
}
```
public class MySimpleRobot {
    public void run() {
        move();
        turnLeft();
        move();
    }
}

- The method is called run and it’s implementation (code) codes between the braces
- public - this method can be used outside of the class
- void - this method doesn’t return a result
This is the implementation of the method
Each statement ends in a semicolon, and the statements are executed sequentially
Defining New Methods

- Adding methods to a class encapsulates the information
- private methods are only accessible inside the class
- private methods hide information about the implementation (information hiding)

```csharp
private void turnAround() {
    turnLeft();
    turnLeft();
}
```
Control Statements

- Conditional Statements
  - if and if-else

- Iteration statements
  - while - doing something until condition
  - for - doing something a number of times
If Statements

- Starts with the *if* part
- Then the statements to execute if the condition is true
- Can choose between operations with *else*
- Possible conditions: frontIsClear, leftIsClear, beepersPresent, beepersInBag, etc.

```java
if (rightIsClear()) {
    fillPotHole();
}
```
for statement

- Used when you know the number of times to do some operations

```java
for (int i = 0; i < 4; i++)
{
    move(10);
    turnRight(90);
}
```
while Statement

- Repeats operation while a condition

```java
int i = 0;
while (i < 4) {
    move(10);
    turnRight(90);
    i++;
}
```
Airplane Reservation

- What data is stored in a reservation?
- What can you do with a reservation?
- How do you refer to a particular reservation?
Instances and Classes

- Use the new operator to create an instance from a class
  - new returns a reference to the object
  - refer to the member variables and methods using the "." operator

```java
Reservation bradRes = new Reservation();
bradRes.setName("Brad Miller");
```

```java
class Reservation {
    private String name;
    private double baseCost;
    private int seat;
    
    public double getCost() {
        return baseCost * 1.12;
    }
    
    public void setName(String n) {
        name = n;
    }
}
```

WPI

Monday, December 13, 2010
Constructors

- Oops! What’s the value of the other member variables that we didn’t initialize when the class was instantiated?
- Wouldn’t it be nice if the name got filled in when the new happened?
- In fact, wouldn’t it be nice if all the member variables could be filled in when the class was instantiated?
Constructors to the Rescue

- A method with the same name as the class is called when the object is created
- And it can have parameters - cool!
- Now remove setName... maybe

```java
class Reservation {
    Reservation(String n) {
        name = n;
        baseCost = 0.0;
        seat = 0;
    }
    // other stuff in class here too...
}

Reservation bradRes = new Reservation("Brad Miller");
```

Look. Not only is it initializing name, but all the other member variables too!
Motors

- Our programs send PWM signals to motors (actually to the motor speed controllers) to control them
- What would an object look like that controls a motor? What sort of methods would it have?
Let’s call our class PWM since it controls a PWM output on the controller. What should it do?

– We’d like to linearize the output signal and we know the min and max PWM values and the center
– So we can make setSpeed do math to linearize the values
Using a PWM object

// Instantiate the PWM objects
PWM left = new PWM(2);
PWM right = new PWM(3);

// set the speeds for full forward
right.setSpeed(1.0);
left.setSpeed(-1.0);

- Looks pretty straightforward, nice linearized speed controllers
- I’m happy... how about you?
Enter the Jaguar

- Just when we thought everything was OK, we get Jaguars
- Different min and max values
- Different curve to linearize
- We can’t use our PWM object to control Jaguars
A New Class for Jaguars

```java
class PWM {
    public PWM(int portNumber) {...}
    public void setSpeed(double speed) {...}
    public double getSpeed() {...}
}
```
class PWM {
    public PWM(int portNumber) {...}
    public void setSpeed(double speed) {...}
    public double getSpeed() {...}
}
A New Class for Jaguars

```java
class PWM {
    public PWM(int portNumber) {...}
    public void setSpeed(double speed) {...}
    public double getSpeed() {...}
}

class Victor {
    public Victor(int portNumber) {...}
    public void setSpeed(double speed) {...}
    public double getSpeed() {...}
}
```
A New Class for Jaguars

class PWM {
    public PWM(int portNumber) {...}
    public void setSpeed(double speed) {...}
    public double getSpeed() {...}
}

class Victor {
    public Victor(int portNumber) {...}
    public void setSpeed(double speed) {...}
    public double getSpeed() {...}
}

and add...
A New Class for Jaguars

class PWM {
    public PWM(int portNumber) {...}
    public void setSpeed(double speed) {...}
    public double getSpeed() {...}
}

class Victor {
    public Victor(int portNumber) {...}
    public void setSpeed(double speed) {...}
    public double getSpeed() {...}
}

class Jaguar {
    public Jaguar(int portNumber) {...}
    public void setSpeed(double speed) {...}
    public double getSpeed() {...}
}
A New Class for Jaguars

class PWM {
    public PWM(int portNumber) {...}
    public void setSpeed(double speed) {...}
    public double getSpeed() {...}
}

class Victor {
    public Victor(int portNumber) {...}
    public void setSpeed(double speed) {...}
    public double getSpeed() {...}
}

class Jaguar {
    public Jaguar(int portNumber) {...}
    public void setSpeed(double speed) {...}
    public double getSpeed() {...}
}

and add...

But all the linearization code and PWM generation code is duplicated in Victor and Jaguar :-(

Monday, December 13, 2010
We had PWMs

Victors are PWMs with one type of curve

Jaguars are PWMs with another type of curve

The curves can be represented by some values (min and max PWM, min and max deadband, and center)
class PWM {
    public PWM(int portNumber) {...}
    public void setSpeed(double speed) {...}
    public double getSpeed() {...}
    public void setBounds(int min, int max,
                            int highDB, int lowDB,
                            int center) {...}
}

class Jaguar extends PWM {
    public Jaguar(int portNumber) {
        super(portNumber);
        setBounds(...); // Jaguar values
    }
}

class Victor extends PWM {
    public Victor(int portNumber) {
        super(portNumber);
        setBounds(...); // Victor values
    }
}

Jaguar and Victor both “inherit from” or extend PWM
Inheritance

- Some terminology
  - PWM is the **superclass**
  - Jaguar and Victor are **subclasses**

- Jaguar and Victor get every *public (or protected)* method and instance variable from PWM

- Jaguar and Victor have their own constructor that can call the superclass (PWM) constructor
Robot drive trains have a lot in common:

– Some motors (almost always 2 or 4)
– Can be controlled with Joysticks
– Can be controlled with code
– Have some limited drive modes: tank, arcade, meccanum, maybe some more
Design a RobotDrive class

class RobotDrive {
    public RobotDrive(Victor leftMotor, Victor rightMotor) {...}
    public void drive(double speed, double direction) {...}
    public void tank(Joystick leftStick, Joystick rightStick) {...}
    public void arcade(Joystick stick) {...}
}

Here we go again, what about Jaguars?
class RobotDrive {
    public RobotDrive(PWM leftMotor, PWM rightMotor) {...}
    public void drive(double speed, double direction) {...}
    public void tank(Joystick leftStick, Joystick rightStick) {...}
    public void arcade(Joystick stick) {...}
class RobotDrive {
    public RobotDrive(PWM leftMotor, PWM rightMotor) {...} 
    public void drive(double speed, double direction) {...} 
    public void tank(Joystick leftStick, Joystick rightStick) {...} 
    public void arcade(Joystick stick) {...}

    Does this work? After all, Victors and Jaguars are both PWMs
class RobotDrive {
    public RobotDrive(PWM leftMotor, PWM rightMotor) {...}
    public void drive(double speed, double direction) {...}
    public void tank(Joystick leftStick, Joystick rightStick) {...}
    public void arcade(Joystick stick) {...}

    Does this work? After all, Victors and Jaguars are both PWMs

    It does, but it will break soon. Why?
Now There are CAN Jaguars

- CAN connected Jaguars don’t use PWM signaling - you just tell it the speed and the Jaguar does it
- It doesn’t make sense to base the new CANJaguar class on PWM
- So the RobotDrive class can’t operate on PWM objects, it needs something else
Quick overview of Objects
Objects (Classes)

Quadrature Encoder Object

What can it do?
- getCount()
- getPeriod()
- reset()

What does it know?
- Port numbers
- Reversed or not

RobotDrive Object

What can it do?
- drive()
- arcadeDrive()
- tankDrive()

What does it know?
- Port numbers
- Speed controller type

Member Variables

Methods
SimpleRobot class

autonomous() { print message "Autonomous" }

operatorControl() { print message "Operator control" }

isAutonomous() { return the current field state }

isEnabled() { return robot state }

Very boring!
SimpleRobot class

autonomous() { print message “Autonomous” }

operatorControl() { print message “Operator control” }

isAutonomous() { return the current field state }

isEnabled() { return robot state }

Team190Robot class

Very boring!
SimpleRobot class

Subclass of a class

autonomous() { print message "Autonomous" }

operatorControl() { print message "Operator control" }

Team190Robot class

autonomous() { Drive around the field and score point }

operatorControl() { Drive our robot using tank steering }

isAutonomous() { return the current field state }

isEnabled() { return robot state }

Very boring!
Team190Robot class

autonomous() { Drive around the field and score point }

operatorControl() { Drive our robot using tank steering }

isAutonomous() { return the current field state }

isEnabled() { return robot state }

Now very cool!
Objects in sample program

myRobot

RobotDrive Object

Constructor:
RobotDrive(leftmotor, rightmotor);
Methods:
drive(speed, curve);
arcade(joystick);

Joystick Object

Constructor:
Joystick(port);
Methods:
getX();
gety();
getTwist(); …
package edu.wpi.first.wpilibj.templates;
import edu.wpi.first.wpilibj.SimpleRobot;
public class Team190Robot extends SimpleRobot {
    public void autonomous() {
    }
    public void operatorControl() {
    }
}
RobotDrive drive;
Joystick leftStick;
Joystick rightStick;

// Constructor
public Team190Robot() {
    drive = new RobotDrive(1, 2);
    leftStick = new Joystick(1);
    rightStick = new Joystick(2);
}


public void autonomous(void) {
    drive.drive(0.5, 0.0);
    Wait(2000);
    drive.drive(0.0, 0.0);
}

Use the RobotDrive object drive method to go 0.5 speed (forward, half speed) with no turn Then stop.

**Key to the example:**

drive.drive(speed, curve)

- **speed**: a value from -1.0 to 1.0 where 0.0 is stopped
- **curve**: a value from -1.0 to 1.0 where 0.0 is no turn
public void operatorControl()
{
    while (true)
    {
        drive.tankDrive(leftStick, rightStick);
        Timer.delay(0.01);
    }
}
Microedition not Desktop Java

- Microedition is Java used on phones
- No dynamic class loading
- Pre-verification step
- Finalization not implemented
- JNI not supported, JNA instead
- Serialization and RMI not supported
- No User Interface APIs - Swing and AWT
- Threads are supported without thread groups
- No new features such as Generics and Autoboxing, yet…

Monday, December 13, 2010
Cross platform development tools

- NetBeans & Eclipse
  - Run on Windows, Linux, and Mac OS X
- Still need Windows/LabVIEW to image the cRIO
- To install on alternate platforms...
  - Install Java
  - Install NetBeans or Eclipse
  - Install the plugins
Development Tools

- NetBeans
  - Open source IDE
    - Supported by Sun Microsystems and community development
  - Custom FRC plugins

- Eclipse
  - Open source
  - Available for 2011
Creating a Robot Program

1. Choose Project
2. Name and Location

- Project Name: SampleProject
- Project Location: /Users/brad
- Project Folder: /Users/brad/SampleProject
- Project Package: edu.wpi.first.wpilibj.templates
- Robot Class: Team190Robot

SampleProject

- src
  - edu.wpi.first.wpilibj.templates
    - Team190Robot.java

build.xml
First program

```java
package edu.wpi.first.wpilibj.templates;
import edu.wpi.first.wpilibj.SimpleRobot;
public class Team190Robot extends SimpleRobot {
    public void autonomous() {
    }
    public void operatorControl() {
    }
}
```

- Easy template program that you fill in
  - autonomous and operatorControl methods called at the appropriate times
  - You can fill in constructor with initialization code
Compiling the program

- Select “Build Main Project” and that’s it!
  - Designate project as main project if there is more than one

- Program builds and errors are highlighted

```java
init:
No to.jar.file specified.
Using "suite/Team190Robot_1.0.0.jar"
clean:
Created dir: /Users/brad/Team190Robot/build
Compiling 1 source file to /Users/brad/Team190Robot/build
compile:
Created dir: /Users/brad/Team190Robot/j2meclasses
preverify:
Created dir: /Users/brad/Team190Robot/suite
Building jar: /Users/brad/Team190Robot/suite/Team190Robot_1.0.0.jar
jar-app:
BUILD SUCCESSFUL (total time: 2 seconds)
```
Running the program

- Select “Run main project” or use the arrow in the toolbar
  - Program will compile
  - Then get downloaded to the cRIO
- Set your driver station to Autonomous or Teleop and Enabled
- That’s it!
Debugging the code

1. Place breakpoint in your code
2. Click on debug button
3. Wait for “Waiting for connection message”
4. Attach debugger
How Java Programs Start

- OTA server runs at cRIO boot
  - Named JavaVM.out
  - OTA server loads squawk.out (VM) and starts it
  - Squawk loads midlet class from suite file
    - Robot class is derived from midlet
  - Robot program starts

- NetBeans loads programs
  - Verifies the versions of squawk and the OTA server and loads new ones if necessary
  - Loads the user program
  - Reboots the cRIO
Threads built in to Java

- Threads part of the Java language

```java
m_task = new CompressorThread(this);
m_task.start();

/*
private class CompressorThread extends Thread {

    Compressor m_compressor;

    CompressorThread(Compressor comp) {
        m_compressor = comp;
    }

    public void run() {
        while (m_run) {
            if (m_compressor.enabled()) {
                m_compressor.setValue(Im_compressor.get.SetValue());
            }
        }
    }
*/
```
Java brings runtime checking

- In C++ there is no runtime checking for pointers or subscripts
  - Everything is a Downloadable Kernel Module
- The good news:
  - Java provides full runtime checking
- The bad news:
  - Java provides full runtime checking
- It will help you find bugs, but will terminate your program when it finds them
An exception is an event that disrupts the normal flow of the program

A method checks for error condition

Exception is “thrown” when an error occurs

Calling program checks for an exception using a try-catch block
Exceptions are passed up

- If the calling method doesn’t check, it’s passed up the call stack
- If no method catches the exception, the runtime system does
  - But the program terminates
Three types of exceptions

- Checked exception: an error condition a program might encounter and should be checked for
- Error exception: something external to the application that it can’t anticipate or recover from
- Runtime exception: subscript out of bounds or a null pointer. Best to fix the program!
WPILib exception policy

- Unchecked exceptions for program errors like bad port numbers, etc.
  - You can optionally check for these

- Checked exceptions for runtime errors
  - You’re required to check for these

- Implication
  - Bad port numbers will likely kill the program
  - Runtime errors will be caught
Error handling

- Exceptions
- Errors on the dashboard
A few sample built in classes

- Camera and imaging classes
- PID control
2010 Camera/Image Classes

- Based on an image class
  - Image class is an abstract base class
    - ColorImage and MonolImage subclassed from Image
    - HSLImage, RGBImage from ColorImage
    - BinaryImage from MonolImage (for masks)
  - ParticleAnalysisReport object

- AxisCamera class
  - Object oriented interface to camera
  - Web interface settings now in class
    - Brightness
    - Exposure control
AxisCamera Class

- Camera class is singleton
  - AxisCamera.getInstance()
- Several methods to set parameters
  - writeBrightness
  - writeWhiteBalance
  - writeColorLevel
  - writeExposureControl
  - writeExposurePriority
  - writeResolution
  - writeCompression
  - writeRotation
Different image types depending on the application
# Image classes

<table>
<thead>
<tr>
<th>Image Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColorImage (RGB &amp; HSL)</td>
<td>Supports plane extraction and thresholding</td>
</tr>
<tr>
<td>MonolImage</td>
<td>Shape detection</td>
</tr>
<tr>
<td>BinaryImage</td>
<td>Particle detection</td>
</tr>
<tr>
<td>HSLImage</td>
<td>Returned by the Camera</td>
</tr>
</tbody>
</table>

![Image Processing GUI](image.png)
Image methods

• All images have:
  – Height and width
  – Write operation to file

• Color Images
  – Get/Replace color planes
  – Thresholding to extract masks in ranges
  – RGB and HSL images can load from file

• Binary Images
  – Particle analysis reporting
Finding Pink / Green target

- Threshold on an HSLImage for each color to generate BinaryImages
- Apply getOrderedParticleAnalysisReport on the BinaryImages
  - List of particleAnalysisReports for each image
  - Results correspond to each pink/green object
Implementation

- Image operations and images are implemented using C++ library
  - Better memory usage (no garbage collection)
  - Higher performance
  - Less copying of images
- Therefore - images need to be manually freed after use
PID Class

- **PID = Proportional, Integral, Differential**
  - Integral and Differential terms are time-based
- **Interfaces for source and target objects**
PID Interfaces

- Add these interfaces to define input and output

- Sources are:
  - Accelerometer
  - Analog Channel
  - Encoder
  - Gyro
  - Ultrasonic

- Outputs are:
  - Jaguar
  - Victor

```java
public interface PIDSource {
    public double pidGet();
}

public interface PIDOutput {
    public void pidWrite(double output);
}
```
public PIDController(double Kp, double Ki, double Kd,
        PIDSource source, PIDOutput output,
        double period) {
    m_controlLoop = new java.util.Timer();
    m_P = Kp;
    m_I = Ki;
    m_D = Kd;
    m_pidInput = source;
    m_pidOutput = output;
    m_period = period;
    m_controlLoop.schedule(new PIDTask(this), 0L, (long) (m_period * 1000));
}
private class PIDTask extends TimerTask {

    private PIDController m_controller;

    public PIDTask(PIDController controller) {
        m_controller = controller;
    }

    public void run() {
        m_controller.calculate();
    }
}
private synchronized void calculate() {

double input = m_pidInput.pidGet();

// calculations go here

m_pidOutput.pidWrite(m_result);
}

Monday, December 13, 2010
Java and C++ compared

- Java and C++ are very similar
- Can’t tell them apart from a distance
  - Able to port code by editing
- Same classes, same methods (almost)
- If you can write in one language, then you can also write in the other
Java and C++ compared

- Garbage collection
- Header files
- Single inheritance with interfaces
- No pointers (sort of)
- Runtime checking
- New development tools
- Performance
Naming conventions

- Method names
  - C++ - Uppercase first letter, then camel case
  - Java - Lowercase first letter, then camel case

- Names are otherwise the same
No include files

- Java source files only - no includes necessary
- Use import to resolve references to external code
- Packages define ownership, path (and some access) for groups of modules
Special Class Methods

- C++ has constructors and destructors, Java only has constructors
- Constructors (method with class-name)
  - Called automatically when an instance is created
  - Used to initialize resources
Documentation

• Getting Started with Java
• WPILib Users Guide
• Includes JavaDocs for all APIs
• Java Tutorial (at java.sun.com)

• And user generated JavaDoc for your own classes
Source code is included in the distribution

• Look at <user account>/sunspotfrcsdk/lib/WPILibJ/src
  • Open that location as a project
  • Generate JavaDocs from that project
  • Right-click on project and select “Generate Javadocs”

Also on FIRSTForge server

• Details will be available soon
• Uses a Subversion repository
Installing

- Follow instructions on web site
  - http://first.wpi.edu
  - http://first.wpi.edu/FRC/frcjava.html
- You can get the 2010 kit from there today
- 2011 kit will be released at Kickoff
Getting started

1. Download JDK and NetBeans

2. Install software

3. Install plugins
What’s new for 2011

- Eclipse Plugin
- Dashboard
- Motor Safety
Eclipse Plugin

- Eclipse is a more widely used IDE
- Also cross platform
- Eclipse is where all new work will be done
- Work in progress
  - Graphical robot model
  - Graphical state machine programing model
  - Event system
Dashboard

- Automatic dashboard generator
- Call log methods on robot
- Interface on Classmate is auto generated
- Work in progress
  - Save and edit layouts
Motor Safety

- Replacement for user watchdog
  - Same across all languages
- No explicit watchdog feeding needed
- Expiration timers for each motor
- Setting motor values resets timer
- Timer timeout stops motor
Last but not least

- Source Control
- NetBeans & Eclipse support Subversion and Mercurial
  - We will be using SVN for FIRSTForge
- You should *seriously* consider using source control for your teams
  - Can revert if (when) code breaks
  - Can track revisions
  - Multiple people can work on it at the same time
The END

- Go to http://first.wpi.edu/ for more information
- This presentation will be posted