Programming the 2010 FRC Control System in Java

Keith M. Hughes
Kickoff, 2010

DON'T PANIC!

There is a lot to cover here in little time, so things may go by too fast, or you may need more time to understand the concepts.

But have no fear.

These slides will be available at

http://www.robotbrains.org

Follow the links.

Most stuff will only make sense when you start doing it.

The Controller Hardware

The FRC controller is quite powerful.

- 400 MHz processor.
- 64 Megabyte memory, 32 megabytes for user programs.
- Full multi-tasking, real-time operating system (can do more than one thing at a time).

Many complicated operations are done in hardware (e.g. quadrature encoding).

Programming has been made simpler so that you can concentrate more on the robot problem at hand and not so much on the processor and operating system.

Programming Choices

Do your programmers think they might want to do programming some day?

- **For a professional career**
- **Build websites (Javascript)**

Then Java or C++ is an excellent choice. Not all is as hard as everyone claims.

If no one on team wants to study programming in school or work in the software field, or is interested more in a non-programming field (like engineering), then LabView is an excellent choice. A lot like MindStorms Lego programming.

Why Use Java?

There are many reasons to use Java.

- Easier to use than C++.
- The development environment runs on Windows, Mac, and Linux.
- More likely to have Java taught in high school than C++ or LabView.
- Java used a lot in the programming industry.

Java

- Language used in the software industry.
- Is a descendant of *C* and *C++*.
- Many similar languages, like Javascript (used in web pages), *C#*.

```
public class Team9901Robot2009 extends SimpleRobot
{
     RobotDrive robotDrive = new RobotDrive(1, 2);
     Joystick stick = new Joystick(1);

     public void operatorControl() {
          getWatchdog().setEnabled(true);
          while (isOperatorControl())
          {
               getWatchdog().feed();
               robotDrive.arcadeDrive(stick);
          }
     }
}
```

Java Is a Descendant of C/C++

Many Java constructs are the same in C and C++.

- `if` statements
- `while` loops
- `for` loops
- Use of curly braces (`{` and `}`).

Java Starter Code

Use the templates provided with Netbeans to start with. These set up the environment most of the way for you.

For the simplest robot, use the Simple Robot Template.

For something closer to the old way of programming the FRC controller, consider the Iterative Robot Template.

A Simple Robot Controller

```
package org.team9901;

import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.RobotDrive;
import edu.wpi.first.wpilibj.SimpleRobot;
import edu.wpi.first.wpilibj.Timer;

public class Team9901Robot2009 extends SimpleRobot
{
    private RobotDrive robotDrive = new RobotDrive(1, 2);
    private Joystick stickLeft = new Joystick(1);
    private Joystick stickRight = new Joystick(2);
```

A Simple Robot Controller, Continued

```
public Team9901Robot2009() {
    getWatchdog().setExpiration(0.1);
}

void operatorControl() {
    getWatchdog().setEnabled(true);

    while (isOperatorControl()) {
        getWatchdog().feed();

        robotDrive.tankDrive(stickLeft, stickRight);
    }

}
```

A Simple Robot Controller, Continued

```
    void autonomous(void)
    {
        getWatchdog().setEnabled(false);

        robotDrive.drive(0.5, 0.0);   // drive forwards half speed
        Timer.delay(2.0);             // for 2 seconds
        robotDrive.drive(0.0, 0.0);   // stop robot
        Timer.delay(2.0);             // for 2 seconds
        robotDrive.drive(0.0, 0.75);  // turn at three quarters speed
        Timer.delay(2.0);             // for 2 seconds
        robotDrive.drive(0.0, 0.0);   // and stop
    }
}
```

Where Is WPILib?

The `import` directives tells your program where things are defined in WPILib.

These must be in your program at the top before your class definitions and after the `package` line.

```
package org.team9901;

import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.RobotDrive;
import edu.wpi.first.wpilibj.SimpleRobot;
import edu.wpi.first.wpilibj.Timer;

public class Team9901Robot2009 extends SimpleRobot
```

Object-Oriented Programming

Terms like

- **inherits/extends**
- **class**
- **instance variables**
- **constructor**

are used a lot in object-oriented programming.

If the terms don't make sense, don't worry about them for now. You can, and should, understand them later, but you don't have to immediately.

Just know for now that you have to write things in a certain way for your robot to work.

Your Robot Class

```
public class Team9901Robot2009 extends SimpleRobot
{
    private RobotDrive robotDrive = new RobotDrive(1,2);
    private Joystick stickLeft = new Joystick(1);
    private Joystick stickRight = new Joystick(2);
```

Your robot program is called a class in Java.

A class is a collection of data and functions which work on that data.

The name of this robot program is `Team9901Robot2009`.

It extends `SimpleRobot`, which is also a collection of data and functions that `Team9901Robot2009` gets to use as though they were defined in `Team9901Robot2009`, like `getWatchdog()`.

Instance variables

```
public class Team9901Robot2009 extends SimpleRobot
{
    private RobotDrive robotDrive = new RobotDrive(1,2);
    private Joystick stickLeft = new Joystick(1);
    private Joystick stickRight = new Joystick(2);
```

This robot class has 3 instance variables, the data for your robot class.

- `robotDrive`
- `stickLeft`
- `stickRight`

This means this robot has a RobotDrive (which can be 2 or 4 PWM signals for your drive system) and two Joysticks.

It could also have an Accelerometer, a Jaguar (speed controller), or whatever.

2 Wheel Drive

```
public class Team9901Robot2009 extends SimpleRobot
{
    private RobotDrive robotDrive = new RobotDrive(1,2);
    private Joystick stickLeft = new Joystick(1);
    private Joystick stickRight = new Joystick(2);
```

The line with RobotDrive on it says that you have a 2 wheel drive robot where

- the left motor is driven by PWM 1
- the right motor is driven by PWM 2

I Want 4 Wheel Drive!

To get 4 motors, you just tell the RobotDrive constructor the numbers of the 4 PWM outputs you want to use.

Suppose, just to be silly, we put

- the left, front motor on PWM 2
- the left, back motor on PWM 4
- the right, front motor on PWM 6
- the right, back motor on PWM 8

```
public class Team9901Robot2009 extends SimpleRobot
{
    private RobotDrive robotDrive = new RobotDrive(2, 4, 6, 8);
    private Joystick stickLeft = new Joystick(1);
    private Joystick stickRight = new Joystick(2);
```

Initializing Your Robot

Any code you want to run only once and initialize your robot goes into a special kind of function called a constructor.

Here, the watchdog timer is set and nothing else.

```
public Team9901Robot2009() {
    getWatchdog().setExpiration(0.1);
}
```

Initializing Your Robot

Anything else you want done only once to initialize the robot should go inside the constructor function.

You could initialize gyroscopes, air compressors, etc, in the constructor.

Here the sensitivity of the gyroscope is set along with the watchdog expiration time.

```
public Team9901Robot2009() {
    getWatchdog().setExpiration(0.1);

    gyro.setSensitivity(0.2);
}
```

Initializing Your Robot

A very important thing that every robot must do is start the User Watchdog.

Here the watchdog is told that it must be fed every 1/10th of a second.

```
public Team9901Robot2009() {
    getWatchdog().setExpiration(0.1);
}
```

The argument for `setExpiration()` is in seconds.

Example of Inheritence

The class `Team9901Robot2009` does not define the function `getWatchdog()`.

```
public Team9901Robot2009() {
    getWatchdog().setExpiration(0.1);
}
```

The class `SimpleRobot` has a definition the function `getWatchdog()`. Because `Team9901Robot2009` extends `SimpleRobot`, functions in `Team9901Robot2009` can use `getWatchdog()`.

Operator Control

The controller will run the `operatorControl()` function when the driver station tells the robot it is in teleoperated mode.

Here the watchdog is re-enabled in case someone turned it off. Notice the watchdog is fed every time the loop goes around again.

Also, each time around the loop we tell the drive motors what speeds the joysticks are requesting.

```
void operatorControl() {
    getWatchdog().setEnabled(true);

    while (isOperatorControl()) {
        getWatchdog().feed();

        robotDrive.tankDrive(stickLeft, stickRight);
    }
}
```

Arcade Mode

Want Arcade Mode? It is an easy change.

```
public class Team9901Robot2009 extends SimpleRobot
{
    private RobotDrive robotDrive = new RobotDrive(1, 2);
    private Joystick stick = new Joystick(1);

    public Team9901Robot2009() {
        getWatchdog().setExpiration(0.1);
    }

    public void operatorControl() {
        getWatchdog().setEnabled(true);
        while (isOperatorControl()) {
            getWatchdog().feed();
            robotDrive.arcadeDrive(stick);
        }
    }
```

Only 1 joystick defined.

Changed from `tankDrive()` to `arcadeDrive()` and used the one joystick.

A Simple Autonomous Mode

The controller will run the `autonomous()` function whenever the driver station is set to Auto.

```
public void autonomous() {
    getWatchdog().setEnabled(false);

    robotDrive.drive(0.5, 0.0);   // drive forwards half speed
    Timer.delay(2.0);             // for 2 seconds
    robotDrive.drive(0.0, 0.0);   // stop robot
    Timer.delay(2.0);             // for 2 seconds
    robotDrive.drive(0.0, 0.75);  // turn at three quarters speed
    Timer.delay(2.0);             // for 2 seconds
    robotDrive.drive(0.0, 0.0);   // and stop
}
```

Autonomous Mode Gotcha

The autonomous() method will not automatically stop when the driver station switches to teleoperated mode.

Which means your robot could keep moving after the autonomous period is over.

Oops!

Can check with `isAutonomous()` function. Or make sure your program doesn't run longer than the autonomous period.

Making The Program Pause

```
Timer.delay(2.0);
```

The `Timer.delay()` function tells the program to pause. The argument is in seconds.

The motors don't pause, just the program.

Driving The Robot

```
robotDrive.drive(straight, turn);
```

The `drive()` function on a `RobotDrive` object tells the motors at what speed to turn.

The first number, `straight`, tells the robot how fast to move forward or backwards.

The second number, `turn`, tells the robot how fast to turn.

Both can be working at the same time, the robot moves forwards or backwards while turning.

Driving The Robot

```
robotDrive.drive(straight, turn);
```

The numbers `straight` and `turn` are between -1.0 and 1.0.

- **0.0 means stop.**
- **1.0 means 100% power forward.**
- **-1.0 means 100% power reverse.**
- **0.5 is 50% forward.**
- **-0.31412 is 31.412% backward.**

You get the idea.

Driving The Robot

```
robotDrive.drive(straight, turn);
```

If you put in a value bigger than 1 or smaller than -1, the robot doesn't go in the opposite direction you thought it would. Values are clamped at their maximum and minimum values.

- -21.45 is the same as -1.0.
- 123.7 is the same as 1.0.

A Fancier Autonomous Mode

```
void autonomous(void)
{
    getWatchdog().setEnabled(false);

    // Drive the robot in a square until autonomous mode is over.
    int state = 1;
    while (isAutonomous()) {
        if (state == 1) {
            robotDrive.drive(0.5, 0.0);
            Timer.delay(2.0);
            state = 2;
        } else {
            // This will have to be tuned to make the robot turn right.
            robotDrive.drive(0.0, 0.75);
            Timer.delay(2.0);
            state = 1;
        }
    }
}
```

Making Programs Readable

```
public class Team9901Robot2009 extends SimpleRobot
{
    RobotDrive robotDrive = new RobotDrive(1, 2);
    Joystick stickLeft = new Joystick(1);
    Joystick stickRight = new Joystick(2);

    public Team9901Robot2009() {
        getWatchdog().setExpiration(0.1);
    }
```

What does 1 mean? What does 2 mean? Programmers call these Magic Numbers.

Making Your Program Readable

```
public class Team9901Robot2009 extends SimpleRobot
{
    static final int LEFT_MOTOR_PWM = 1;
    static final int RIGHT_MOTOR_PWM = 2;
    static final int LEFT_JOYSTICK_USB = 1;
    static final int RIGHT_JOYSTICK_USB = 2;

    private RobotDrive robotDrive =
        new RobotDrive(LEFT_MOTOR_PWM, RIGHT_MOTOR_PWM);
    private Joystick stickLeft = new Joystick(LEFT_JOYSTICK_USB);
    private Joystick stickRight = new Joystick(RIGHT_JOYSTICK_USB);

    public Team9901Robot2009() {
        getWatchdog().setExpiration(0.1);
    }
```

So Many Objects, So Little...

Be sure to look through the Javadoc. There are objects for

- Speed controllers: Jaguar, Victor
- Joysticks
- Accelerometers
- Digital Inputs
- Analog Inputs
- Gear Sensors
- Compressors
- Solenoids
- Ultrasonic Sensors
- The Camera

More Information

Obviously the National Instruments site and the FIRST forums.

http://www.chiefdelphi.com

A forum for FIRST teams to discuss all aspects of FIRST.

Use your favorite search engine to look up Java Programming, or Java Tutorial.

http://www.robotbrains.org

Lots of resources, including presentations from previous workshops.

Look for the Programming In RobotWorld online book. Uses a BASIC-like language, but the programming concepts are the same.

Have Fun!