FRC

FIRST® Robotics Competition

λ Lambda Corps

FIRST Robotics Team # 1895

Introduction to Eclipse, RobotBuilder and JAVA

Dave Frederick, Mentor

Team 1895, Manassas, VA

November 14, 2015 Version 1.0

Goal of Workshop

- Demonstrate a process to create a basic Robot using Eclipse and RobotBuilder

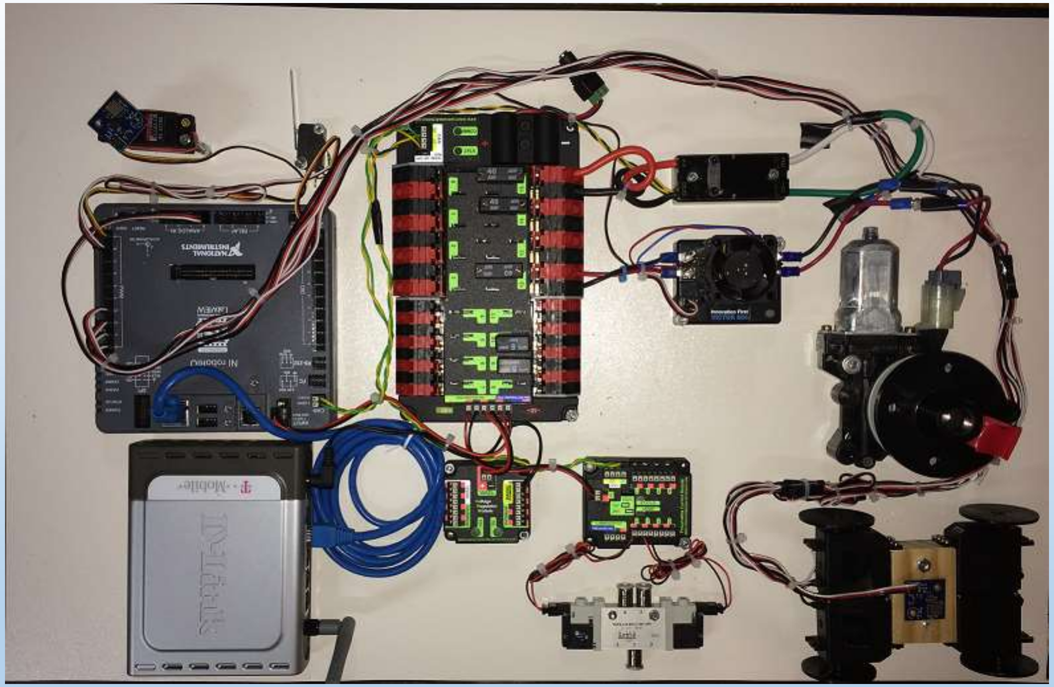
WPI (Worcester Polytechnic Institute) Robot Builder Resource:

- https://wpilib.screenstepslive.com/s/4485/m/26402

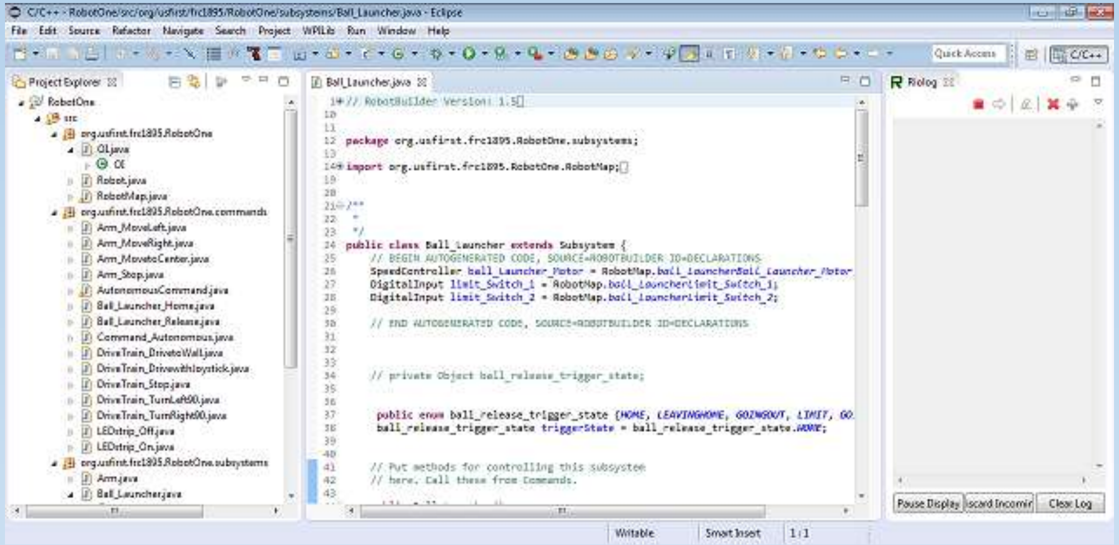
Process to Build Robot Software - Overview

1. Design the Robot on Paper
2. Build Robot in RobotBuilder
   - Create subsystems, commands and Operator Interface
3. Import RobotBuilder into Eclipse
4. Finish the Software in Eclipse
   - Create methods within the subsystems to implement commands
5. Deploy, Test, Troubleshoot
6. Expand, Revise, Enhance

Development Environment



- Hardware
  - Practice Board (RoboRio, Power system, motors, servos, pneumatics …)
- Software
  - Eclipse – Luna with WIPlib Extensions
  - JAVA – SDK (Software Development Kit)
  - FIRST Drivers Station
- Network
  - Basic for demonstration

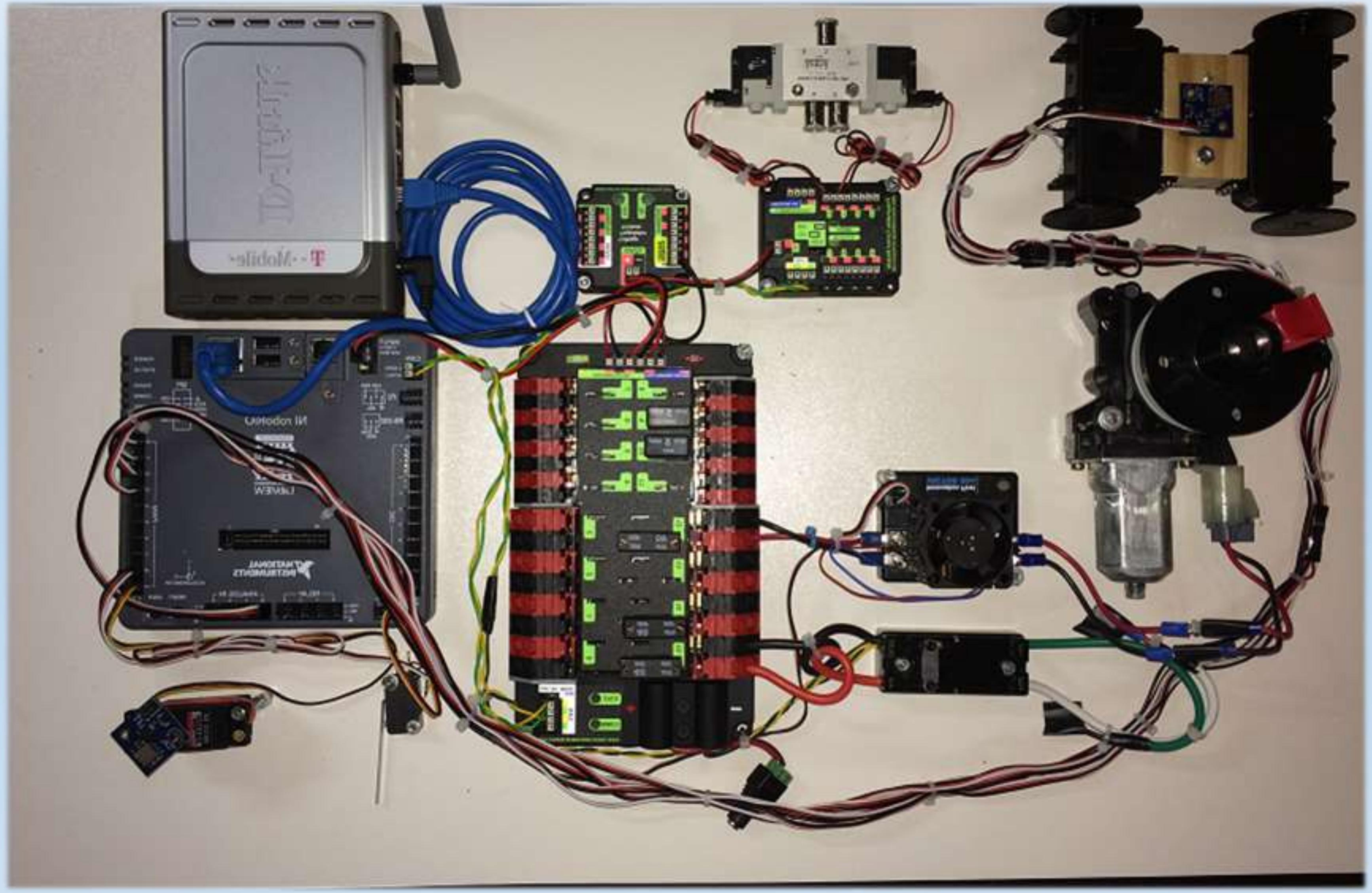


C/C++ - RobotOne/src/org/usfirst/frc1895/RobotOne/subsystems/Ball_Launcher.java - Eclipse
File Edit Source Refactor Navigate Search Project WPILib Run Window Help
Quick Access
C/C++
Project Explorer
Ball_Launcher.java
Riolog
Pause Display
iscard Incomin
Clear Log
Writable
Smart Insert
1 : 1

```
RobotOne
  src
    org.usfirst.frc1895.RobotOne
      OI.java
        OI
      Robot.java
      RobotMap.java
    org.usfirst.frc1895.RobotOne.commands
      Arm_MoveLeft.java
      Arm_MoveRight.java
      Arm_MovetoCenter.java
      Arm_Stop.java
      AutonomousCommand.java
      Ball_Launcher_Home.java
      Ball_Launcher_Release.java
      Command_Autonomous.java
      DriveTrain_DrivetoWall.java
      DriveTrain_DrivewithJoystick.java
      DriveTrain_Stop.java
      DriveTrain_TurnLeft90.java
      DriveTrain_TurnRight90.java
      LEDstrip_Off.java
      LEDstrip_On.java
    org.usfirst.frc1895.RobotOne.subsystems
      Arm.java
      Ball_Launcher.java
```

```
// RobotBuilder Version: 1.5


package org.usfirst.frc1895.RobotOne.subsystems;

import org.usfirst.frc1895.RobotOne.RobotMap;


/**
 *
 */
public class Ball_Launcher extends Subsystem {
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
    SpeedController ball_Launcher_Motor = RobotMap.ball_LauncherBall_Launcher_Motor
    DigitalInput limit_Switch_1 = RobotMap.ball_LauncherLimit_Switch_1;
    DigitalInput limit_Switch_2 = RobotMap.ball_LauncherLimit_Switch_2;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS



    // private Object ball_release_trigger_state;


     public enum ball_release_trigger_state {HOME, LEAVINGHOME, GOINGOUT, LIMIT, GO
     ball_release_trigger_state triggerState = ball_release_trigger_state.HOME;


    // Put methods for controlling this subsystem
    // here. Call these from Commands.
```

RoboRio Practice Board



Step 1: Design the Robot

Understand the Game then Design the Robot

- Define the Requirements for the Robot
  - Understand how to play the Game and scoring
  - Decide how your Team will play the game
  - Decide what your robot needs to do
- Create an initial design of the Robot
  - Drive Train, Shooter/Manipulator, Drivers Station, automation
- Decompose the Robot into Subsystems
  - Identify Subsystems and the User Interface
- Software High Level Design
  - Determine what functions/actions the subsystems must do
  - Determine the best approach for the User Interface

Design the Robot

Given high level design of subsystems and functional requirements

- Software Detailed Design
  - Document the functions to be performed
  - Document the User Interface
    - Joystick / Gamepad / Buttons, Camera displays
    - On-Robot indicators
  - **Determine where automation can assist**
  - **Determine what sensors are needed**
  - Create commands to drive the subsystems
  - Link in the User Interface
  - Create *methods* for the subsystems

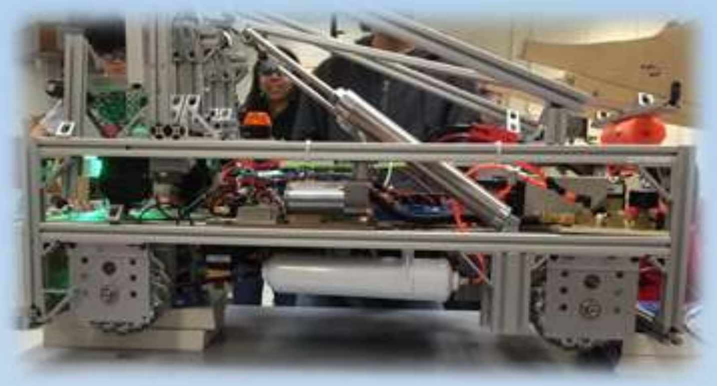
Design the Robot – Drive Train

Drive Train Options:

- Tank drive vs. Mecanum vs. Swerve drive

Items to consider for each option:

- Complexity, Robot agility, Time to build, Weight
- User Interface
  - Tank Drive, Arcade Drive
  - JoyStick, Gamepad
  - Display of first person video
- Software
  - Capabilities during Autonomous *(Drive straight for an exact distance, turn precisely, …)*
- Sensors:
  - Gyro, Accelerometer, Shaft Encoders



Design the Robot - Shooter/Manipulator Subsystem

Shooter/Manipulator Subsystem:

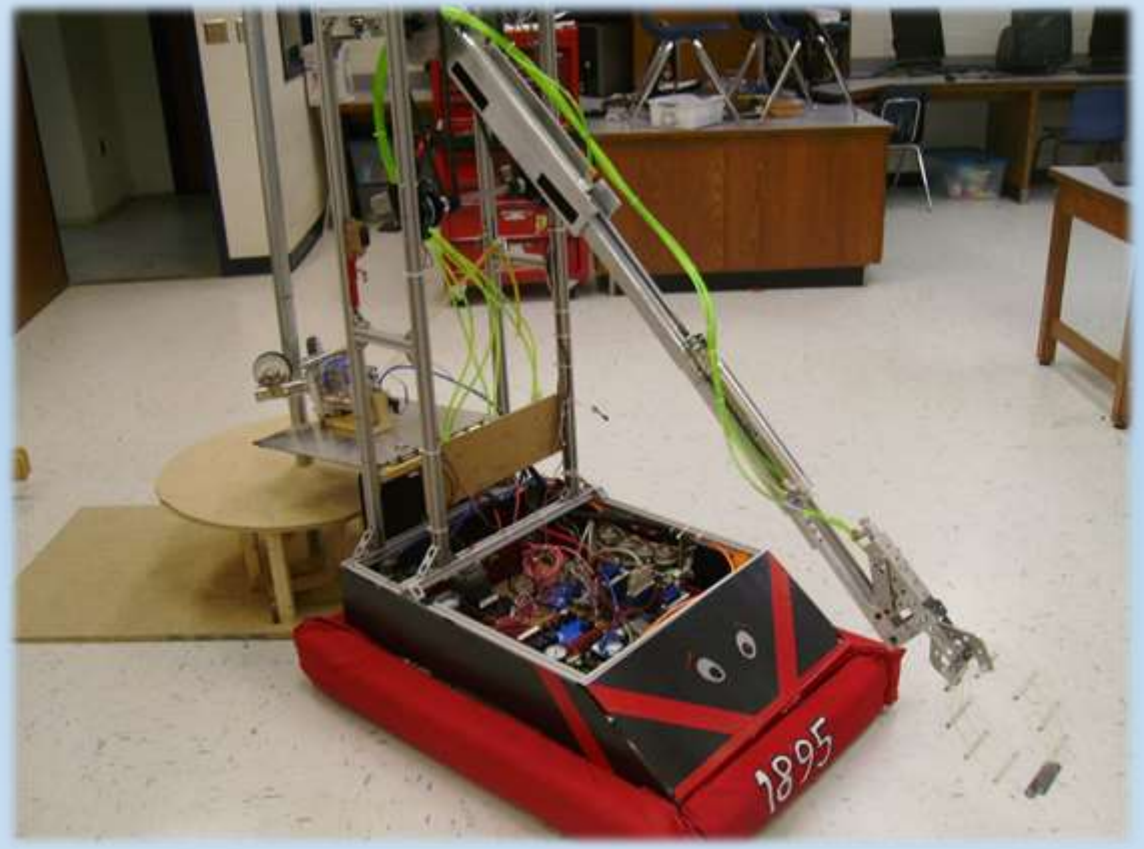
- Ball Launcher, Arm/claw, Kicker, Climber

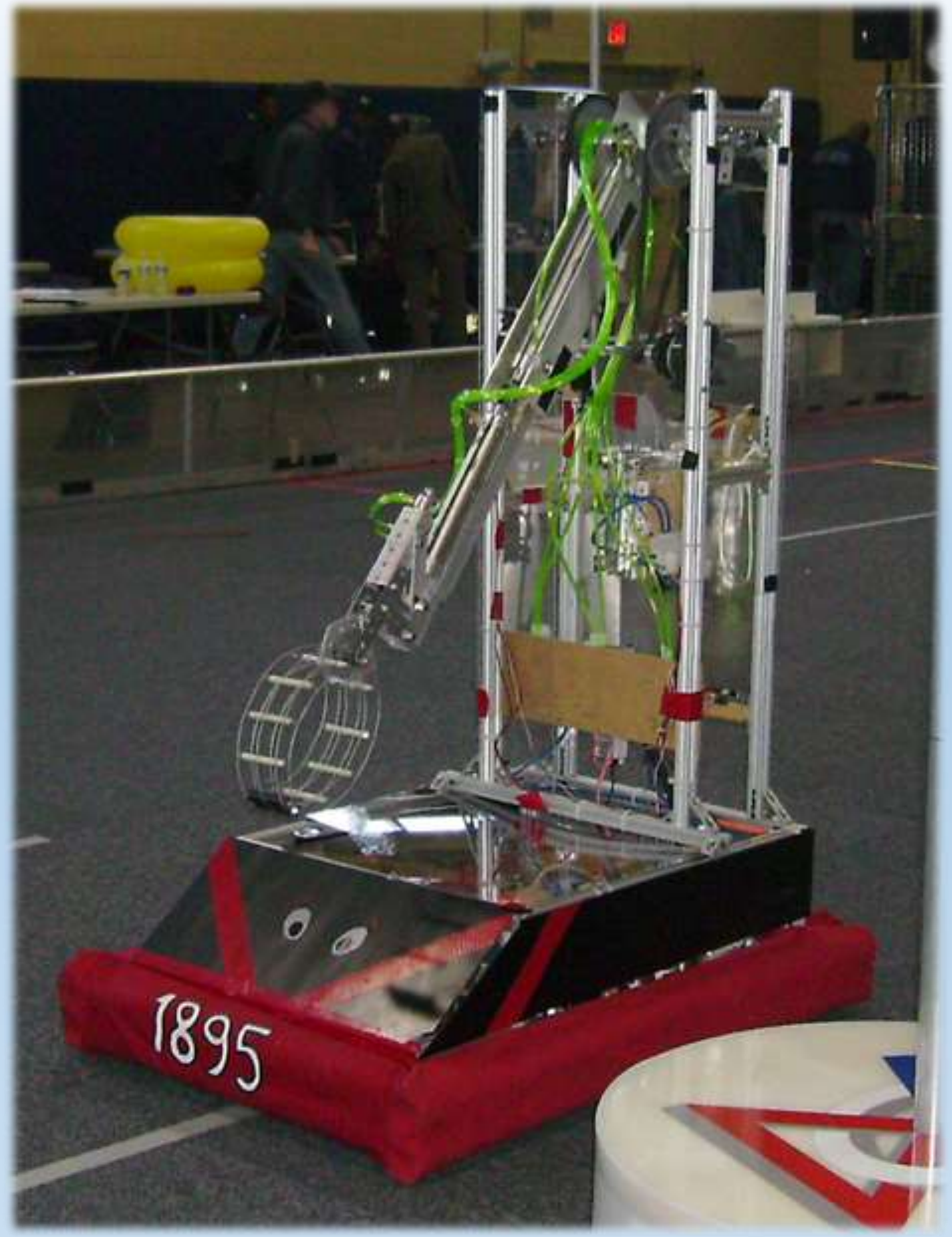
Types of forces & movements:

- Linear movement –Motor/chain/track, pneumatics
- Rotational movement – Motors / gears

For every automated action/motion, a feedback sensor is required

- Types of Feedback Sensors:
  - Limit / position switch, Gyro, Accelerometer, potentiometer, shaft encoders, Range finders, camera


1895


1895

Design the Robot - Sensors

For every automated action/motion, a feedback sensor is required

For example, to have the robot turn 90 degrees, your can’t rely on turn rate over for a fixed amount of time. Need to use a Gyro!

Why? (Battery voltage fades, wheels slip, …)

For every automated action/motion, a feedback sensor is required

Design the Robot - Sensors for Feedback

Sensor	Usage
Limit / position switches	End of travel for arm, elevator, kicker, …
Gyro	Heading to assist in autonomously driving a straight line
Accelerometer	Tilt of robot
Potentiometer	Rotational Angle
Shaft encoders	Rotations of a shaft (translating into height elevator)
Range finders	Distance to wall
Camera	Target tracking, Assist driver with aligning

For every automated action/motion, a feedback sensor is required

Software Design

Think about what to include in the subsystems

Drivetrain should include the sensors to measure distance, heading and range to target

This allows high level commands to be given to the subsystem

- For example, Move forward 2 feet, turn 90 degrees

Object Oriented Design:
Details of the Object implementation are kept within the subsystem

Software Design – Subsystem Requirements

DriveTrain:

- Requirements:
  - Move forward, backward, and rotate
  - Maximum forward speed: 2 feet/Second (Fast)
  - Simple to build, light weight
  - Drive in a straight line in autonomous mode
  - Can stop at specified distance to wall
  - Can turn 90 degrees on command
  - Can move forward 4 feet on command
  - Drive with single joystick
- Implementation:
  - Tank Drive : 4 Motors
  - Sensors: Shaft Encoder, Gyro and Range finder

Software Design – Subsystem Requirements

Manipulator / Arm:

- Requirements:
  - Move left and right, slow and precise
  - Move to any position with buttons
  - Move to center position with button

- Implementation:
  - Strong slow Motor
  - CAN bus controller
  - Sensor that provides feedback: Potentiometer

Software Design

Document each subsystem methods and commands

- Subsystems
  - Drive Train – 4 motor
    - Commands:
      - Stop, manual drive with joystick
      - Drive forward until 5 feet from wall
      - Turn Right 90 degrees, Turn Left 90 degrees
      - Move forward x feet
  - Arm/Manipulator
    - Commands:
      - Stop, manual drive with left and right buttons
      - Move to setpoint
  - Operator Interface
    - Single Joystick: X and Y axis
    - 3 Buttons: Move arm left, Move arm to Center, Move arm to right

Software Development

Setup

Software Development Setup

One Time Configuration of Eclipse

- Initial Configurations (One Time)
  - Set Team Number
    - Eclipse => Windows => Preferences => WPI Lib Preferences = Team Number
  - Configure Eclipse to sync with RobotBuilder
    - Updates in RobotBuilder are automatically added to the Eclipse Project
    - Eclipse => Windows => Preferences => General => Workspace = Enable Refresh using Native hooks or Polling
  - Display Console Window
    - Eclipse => Window => Show View => Other ... => General => RioLog
  - Create Workspace in Eclipse to hold Robot Project
    - Eclipse => File => New => Project… => WPILib Robot Java Development => Example Robot Java Project => Getting Started with Java => Getting Started => Finish

Resource:

https://wpilib.screenstepslive.com/s/4485

Detailed Process:

https://wpilib.screenstepslive.com/s/4485/m/13809/l/145307-creating-your-benchtop-test-program

Step 2: Build Robot in RobotBuilder

Start and Initialize RobotBuilder Project

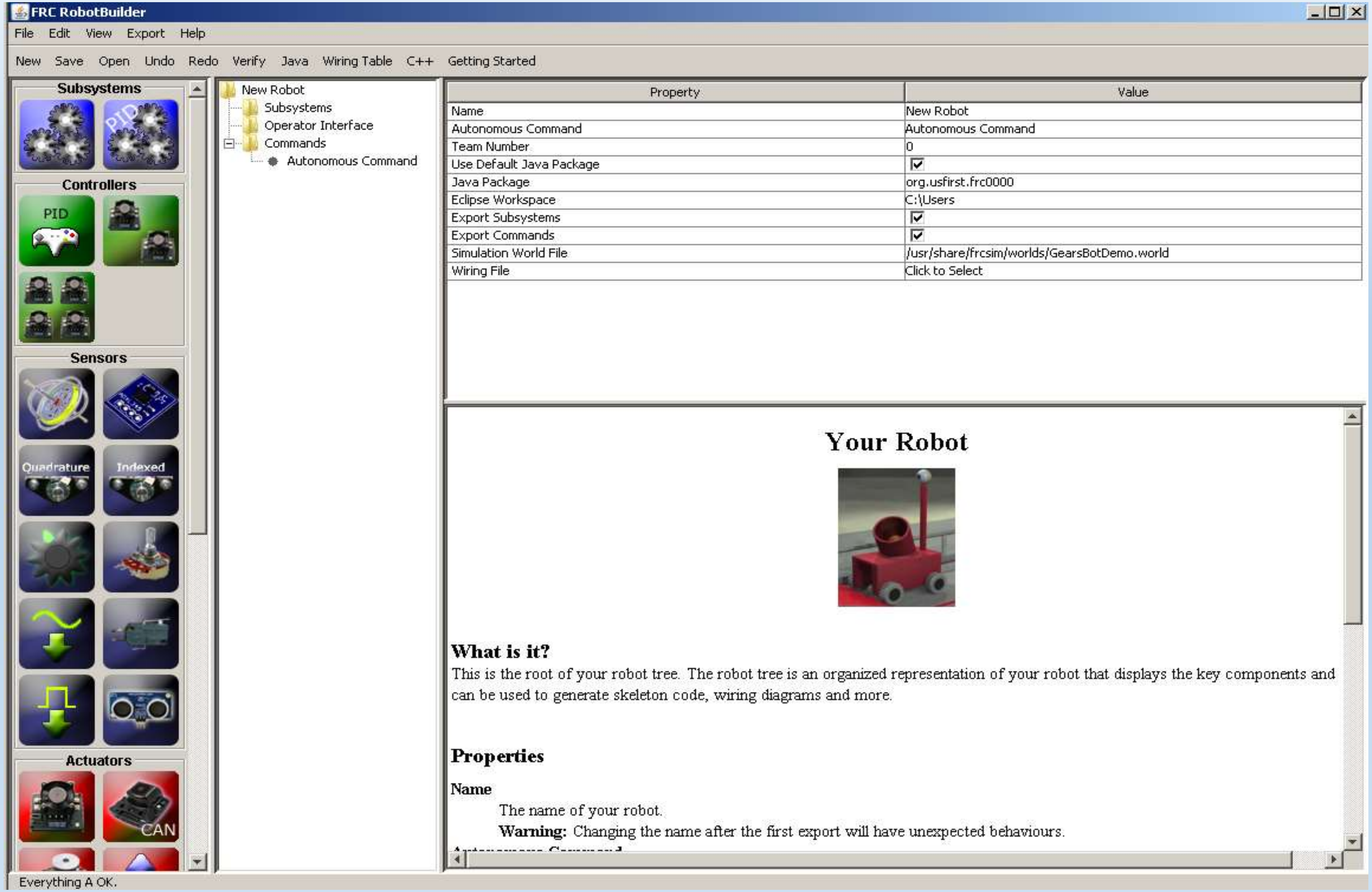
- Eclipse => WIPLib => Run RobotBuilder

- On Creation of new Robot Project, set:
  - Project Name: e.g.: RobotOne
  - Team Number: e.g.: 1895
- In Robot Project, set:
  - Java Package: e.g.: “Team1895.RobotOne”
  - Eclipse Workspace: e.g.: “C:_Robotics\2016\Eclipse_Projects”
    - See current workspace: Eclipse => File => Switch Workspace => Other
  - Wiring File: e.g.: “C:_Robotics\2016\Eclipse_Projects\RobotOneWires”

- Save the RobotBuilder Project file
  - Select RobotBuilder => Save As => “C:_Robotics\2016\Eclipse_Projects\RobotOne”

A package is a namespace that organizes a set of related classes and interfaces. Conceptually you can think of packages as being similar to different folders on your computer.

Robot Builder – First startup



Property	Value
Name	New Robot
Autonomous Command	Autonomous Command
Team Number	0
Use Default Java Package	☑
Java Package	org.usfirst.frc0000
Eclipse Workspace	C:\Users
Export Subsystems	☑
Export Commands	☑
Simulation World File	/usr/share/frcsim/worlds/GearsBotDemo.world
Wiring File	Click to Select

Your Robot

What is it?

This is the root of your robot tree. The robot tree is an organized representation of your robot that displays the key components and can be used to generate skeleton code, wiring diagrams and more.

Properties

Name

The name of your robot.

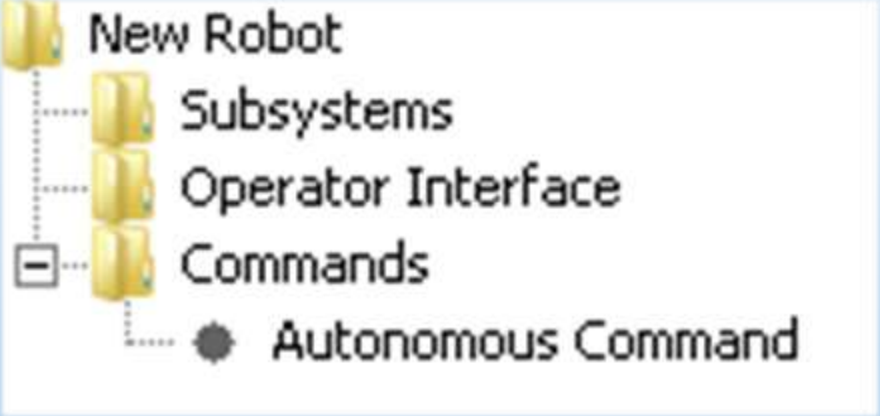
Warning: Changing the name after the first export will have unexpected behaviours.

Autonomous Command

Everything A OK.

Build Robot in RobotBuilder

- Create
  - Subsystems
  - Commands for subsystems
  - Operator Interface
    - Add a JoyStick
    - Add a Button on the JoyStick
  - Associate Buttons with Commands



New Robot
- Subsystems
- Operator Interface
- Commands
  - Autonomous Command

Software Design *(Reminder from the design phase)*

Document each subsystem methods and commands

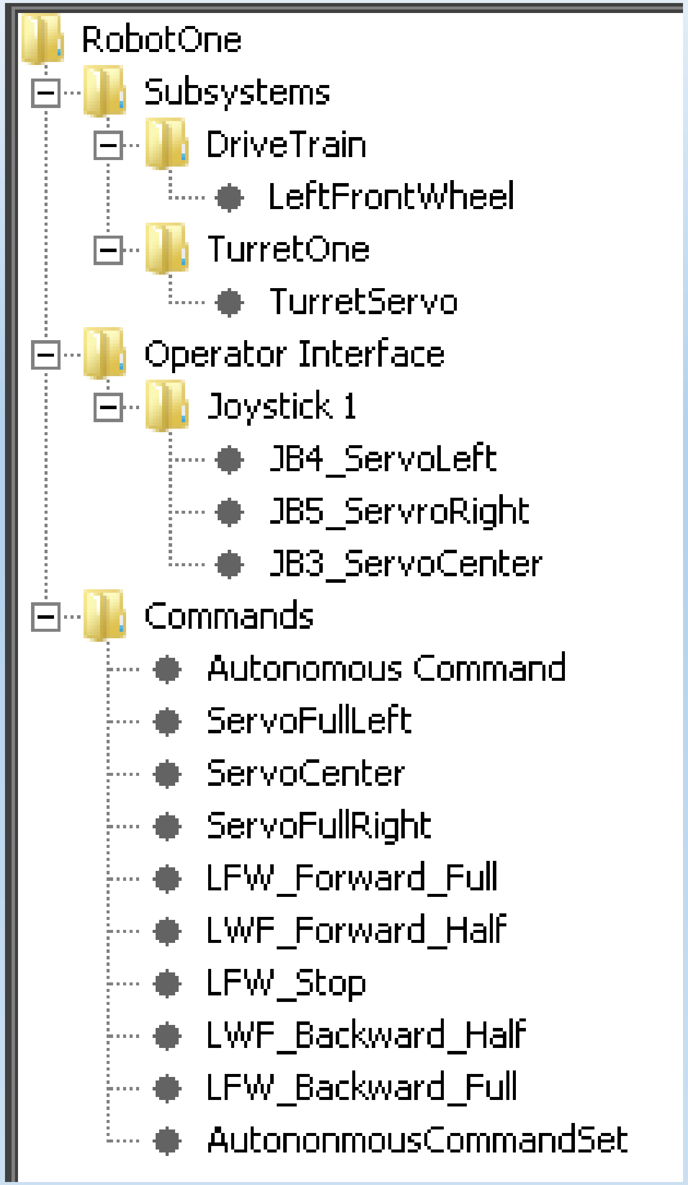
- Subsystems
  - Drive Train – 4 motor
    - Commands:
      - Stop, manual drive with joystick
      - Drive forward until 5 feet from wall
      - Turn Right 90 degrees, Turn Left 90 degrees
  - Arm/Manipulator
    - Commands:
      - Stop, manual drive with left and right buttons
      - Move to setpoint
  - Operator Interface
    - Single Joystick to drive the robot using the X and Y axis
    - 3 Buttons: Move arm left, Move arm to Center, Move arm to right

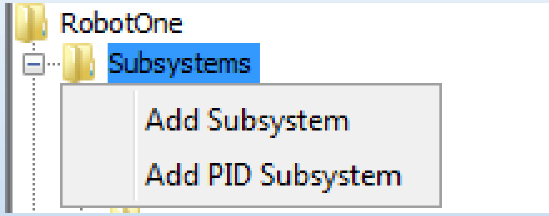
Adding Subsystem and Commands to the Robot in RobotBuilder

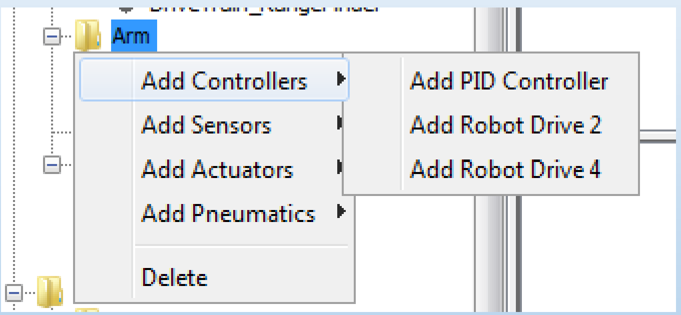
Two methods to add elements

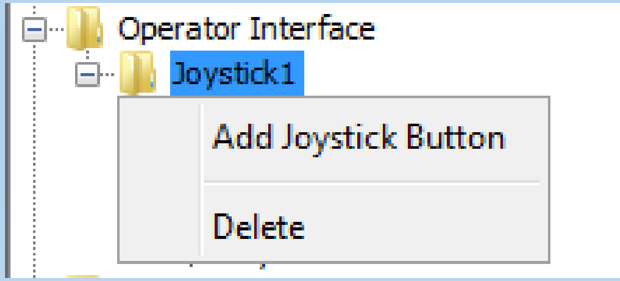
- Drop and Drag
- Right-click and select

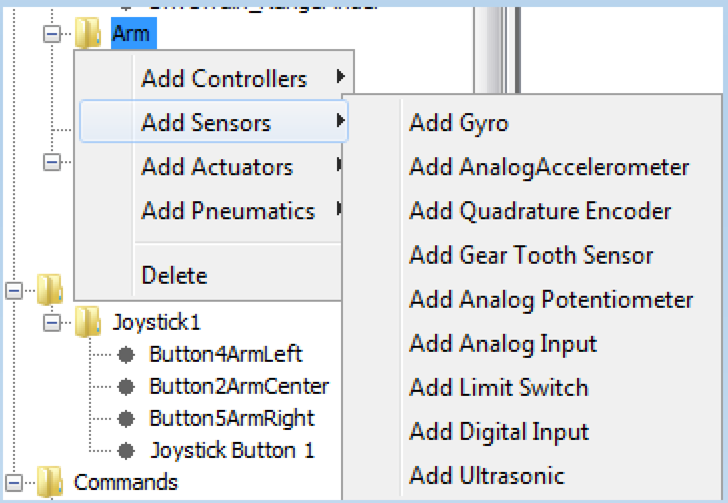
Add subsystems, Joysticks, and sensors by right clicking and selecting

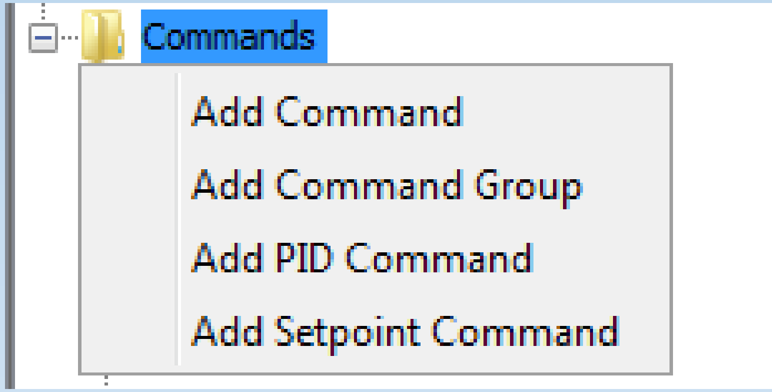


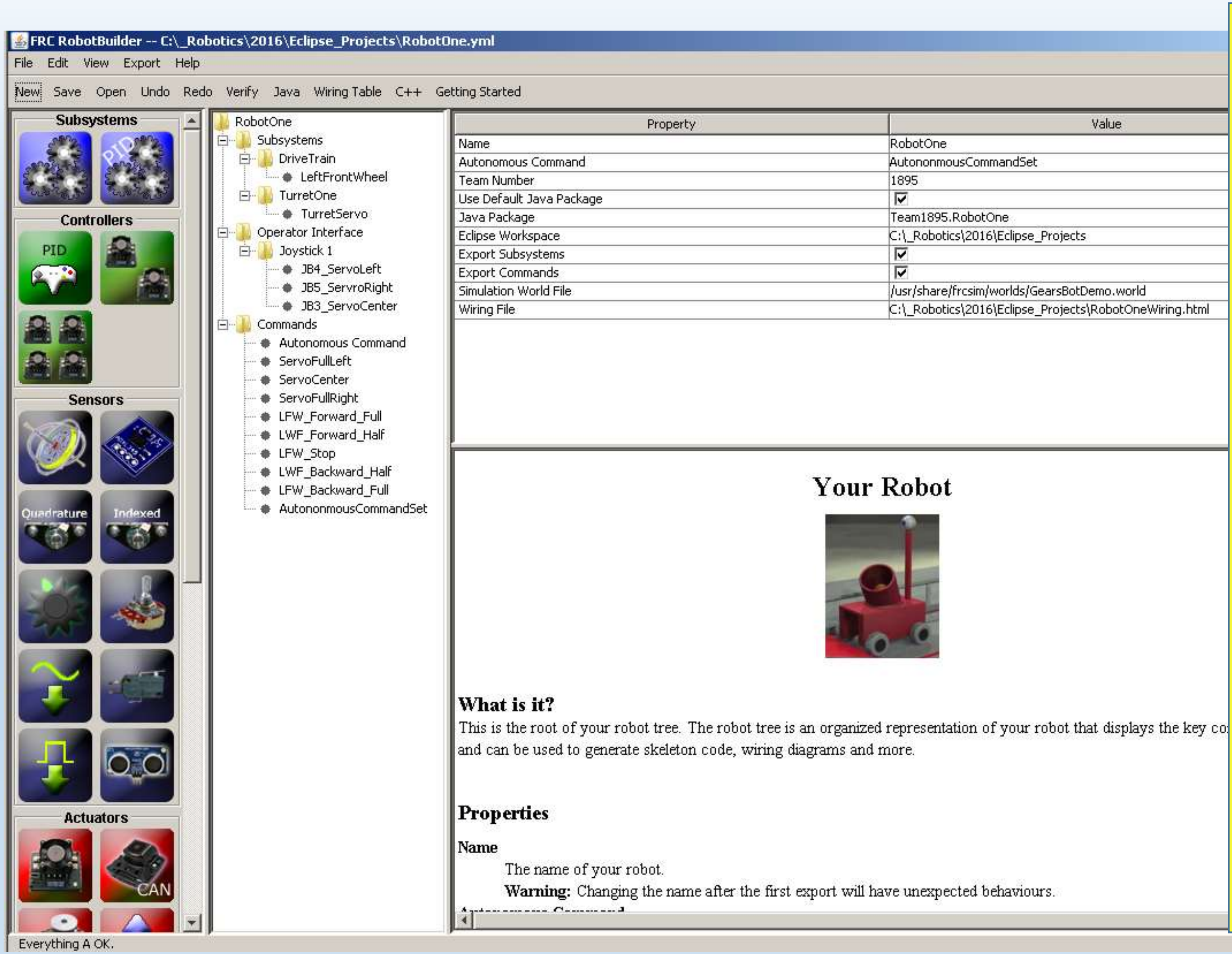












FRC RobotBuilder -- C:_Robotics\2016\Eclipse_Projects\RobotOne.yml
File Edit View Export Help
New Save Open Undo Redo Verify Java Wiring Table C++ Getting Started
Subsystems
PID
Controllers
PID
Sensors
Quadrature
Indexed
Actuators
CAN

- RobotOne
  - Subsystems
    - DriveTrain
      - LeftFrontWheel
    - TurretOne
      - TurretServo
  - Operator Interface
    - Joystick 1
      - JB4_ServoLeft
      - JB5_ServroRight
      - JB3_ServoCenter
  - Commands
    - Autonomous Command
    - ServoFullLeft
    - ServoCenter
    - ServoFullRight
    - LFW_Forward_Full
    - LWF_Forward_Half
    - LFW_Stop
    - LWF_Backward_Half
    - LFW_Backward_Full
    - AutononmousCommandSet

Property	Value
Name	RobotOne
Autonomous Command	AutononmousCommandSet
Team Number	1895
Use Default Java Package	☑
Java Package	Team1895.RobotOne
Eclipse Workspace	C:_Robotics\2016\Eclipse_Projects
Export Subsystems	☑
Export Commands	☑
Simulation World File	/usr/share/frcsim/worlds/GearsBotDemo.world
Wiring File	C:_Robotics\2016\Eclipse_Projects\RobotOneWiring.html

Your Robot

What is it?

This is the root of your robot tree. The robot tree is an organized representation of your robot that displays the key co and can be used to generate skeleton code, wiring diagrams and more.

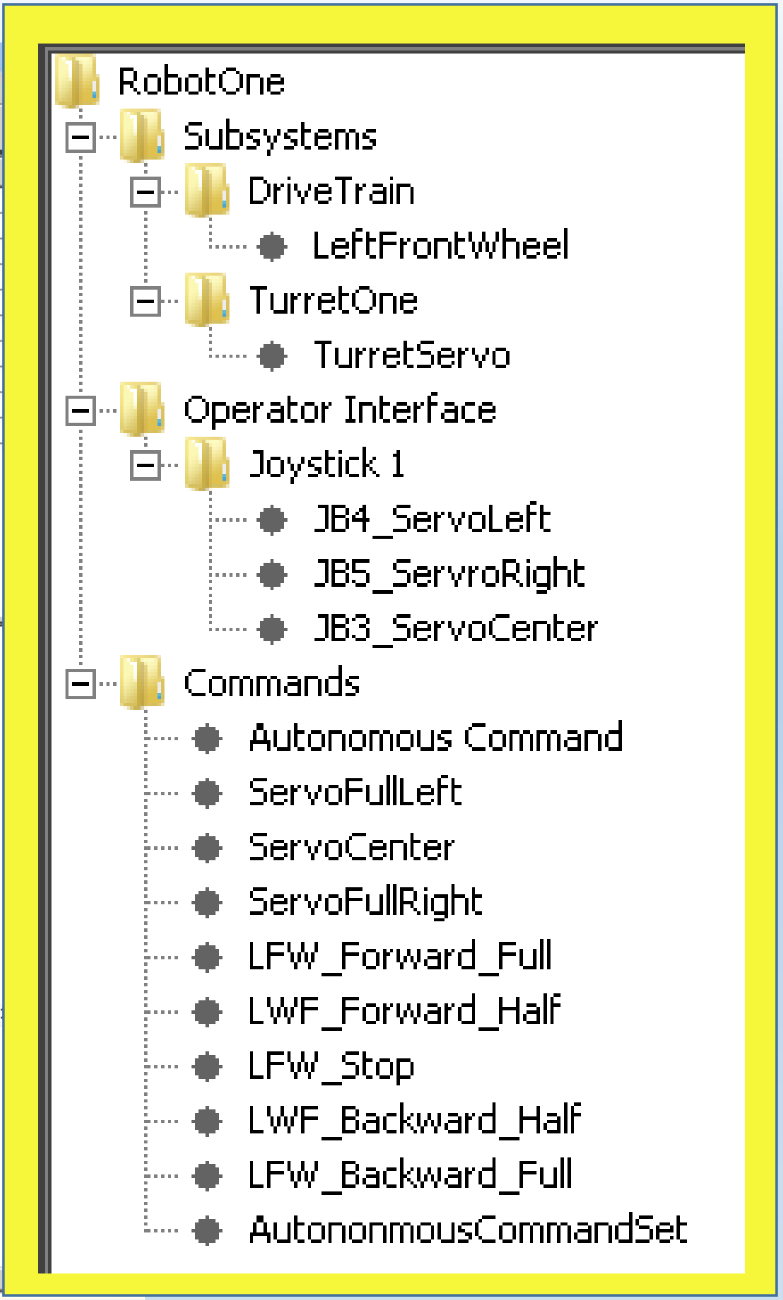
Properties

Name

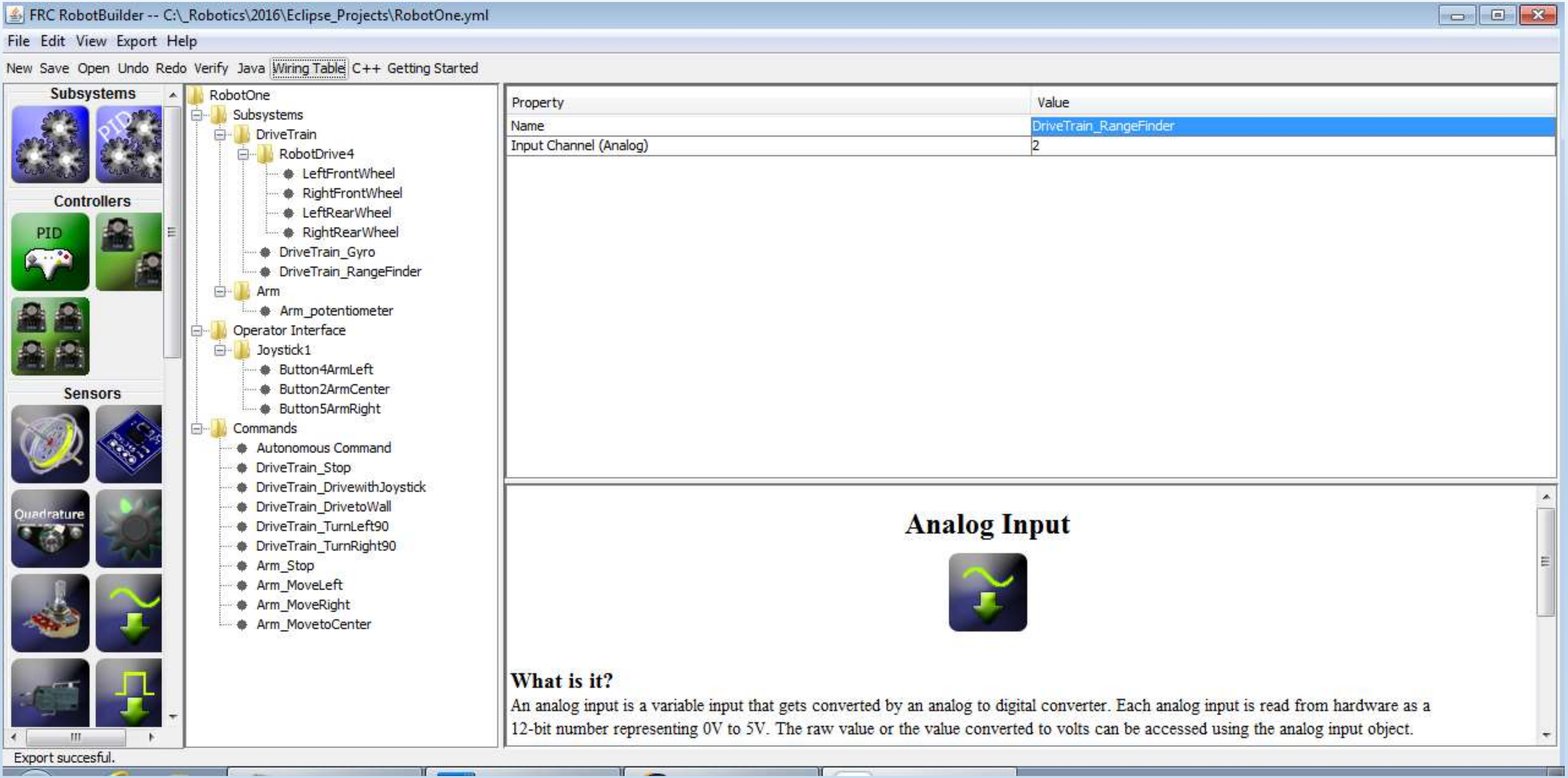
The name of your robot.

Warning: Changing the name after the first export will have unexpected behaviours.

Everything A OK.



- RobotOne
  - Subsystems
    - DriveTrain
      - LeftFrontWheel
    - TurretOne
      - TurretServo
  - Operator Interface
    - Joystick 1
      - JB4_ServoLeft
      - JB5_ServroRight
      - JB3_ServoCenter
  - Commands
    - Autonomous Command
    - ServoFullLeft
    - ServoCenter
    - ServoFullRight
    - LFW_Forward_Full
    - LWF_Forward_Half
    - LFW_Stop
    - LWF_Backward_Half
    - LFW_Backward_Full
    - AutononmousCommandSet


FRC RobotBuilder -- C:_Robotics\2016\Eclipse_Projects\RobotOne.yml
File Edit View Export Help
New Save Open Undo Redo Verify Java Wiring Table C++ Getting Started
Subsystems
PID
Controllers
PID
Sensors
Quadrature

```
RobotOne
    Subsystems
        DriveTrain
            RobotDrive4
                LeftFrontWheel
                RightFrontWheel
                LeftRearWheel
                RightRearWheel
            DriveTrain_Gyro
            DriveTrain_RangeFinder
        Arm
            Arm_potentiometer
    Operator Interface
        Joystick1
            Button4ArmLeft
            Button2ArmCenter
            Button5ArmRight
    Commands
        Autonomous Command
        DriveTrain_Stop
        DriveTrain_DrivewithJoystick
        DriveTrain_DrivetoWall
        DriveTrain_TurnLeft90
        DriveTrain_TurnRight90
        Arm_Stop
        Arm_MoveLeft
        Arm_MoveRight
        Arm_MovetoCenter
```

Property	Value
Name	DriveTrain_RangeFinder
Input Channel (Analog)	2

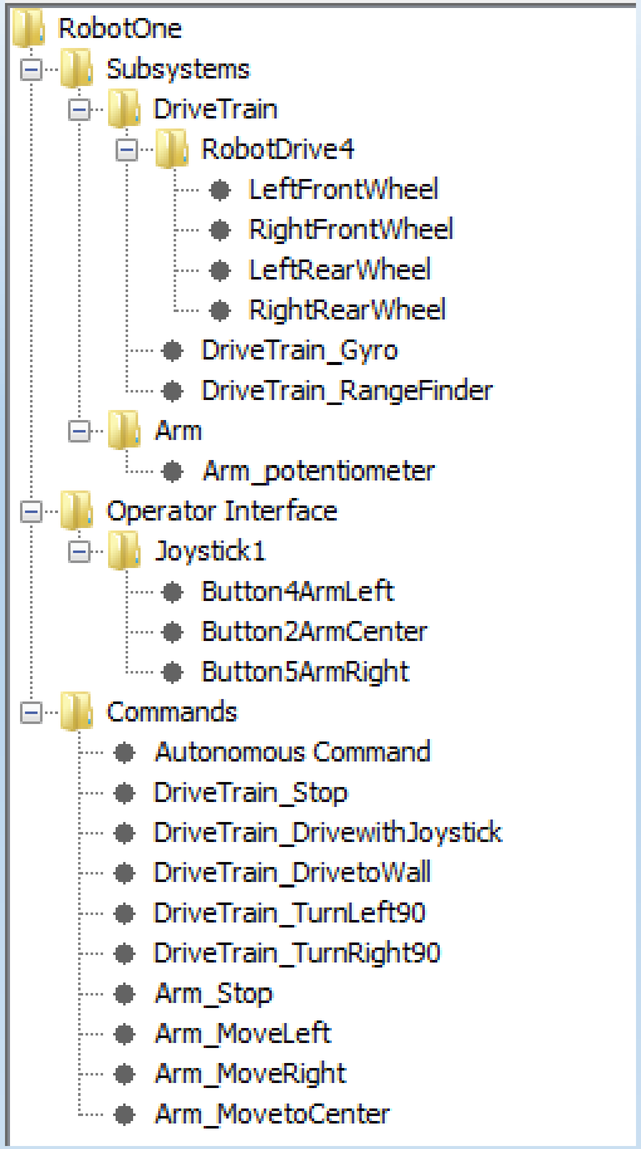
Analog Input

What is it?

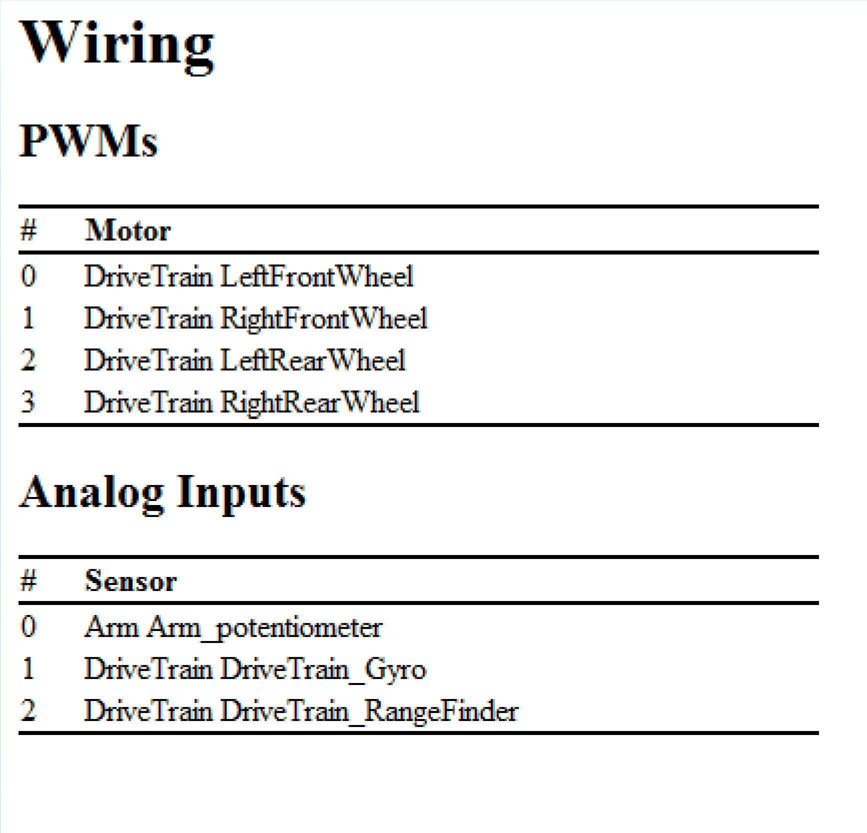
An analog input is a variable input that gets converted by an analog to digital converter. Each analog input is read from hardware as a 12-bit number representing 0V to 5V. The raw value or the value converted to volts can be accessed using the analog input object.

Export succesful.

Robot Builder – Explorer and Wiring Diagram



- RobotOne
  - Subsystems
    - DriveTrain
      - RobotDrive4
        - LeftFrontWheel
        - RightFrontWheel
        - LeftRearWheel
        - RightRearWheel
      - DriveTrain_Gyro
      - DriveTrain_RangeFinder
    - Arm
      - Arm_potentiometer
  - Operator Interface
    - Joystick1
      - Button4ArmLeft
      - Button2ArmCenter
      - Button5ArmRight
  - Commands
    - Autonomous Command
    - DriveTrain_Stop
    - DriveTrain_DrivewithJoystick
    - DriveTrain_DrivetoWall
    - DriveTrain_TurnLeft90
    - DriveTrain_TurnRight90
    - Arm_Stop
    - Arm_MoveLeft
    - Arm_MoveRight
    - Arm_MovetoCenter



Wiring

PWMs

#	Motor
0	DriveTrain LeftFrontWheel
1	DriveTrain RightFrontWheel
2	DriveTrain LeftRearWheel
3	DriveTrain RightRearWheel

Analog Inputs

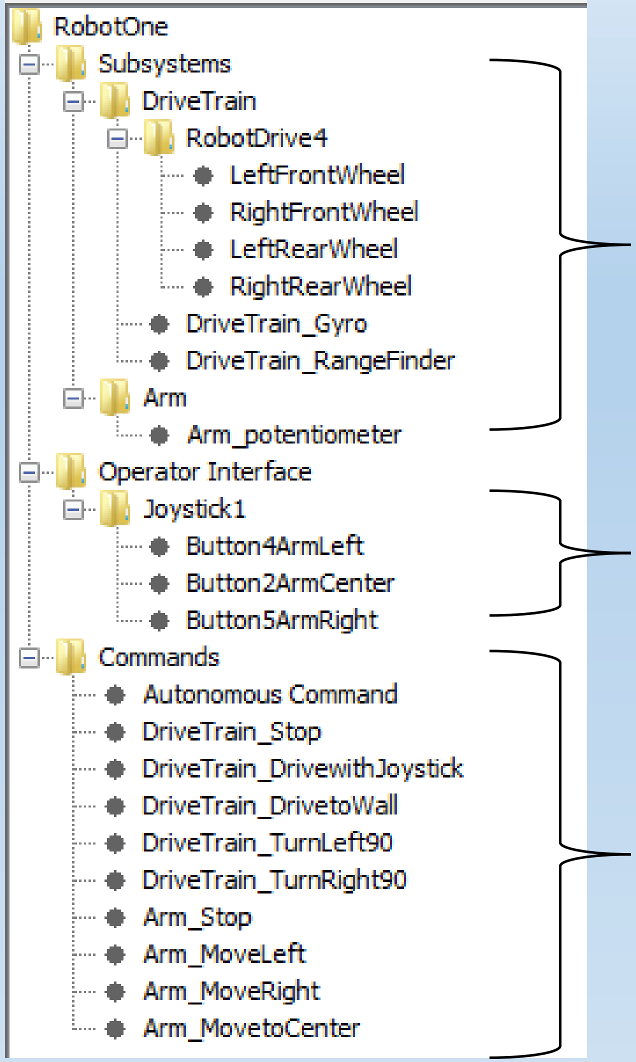
#	Sensor
0	Arm Arm_potentiometer
1	DriveTrain DriveTrain_Gyro
2	DriveTrain DriveTrain_RangeFinder

Step 3: Import RobotBuilder into Eclipse

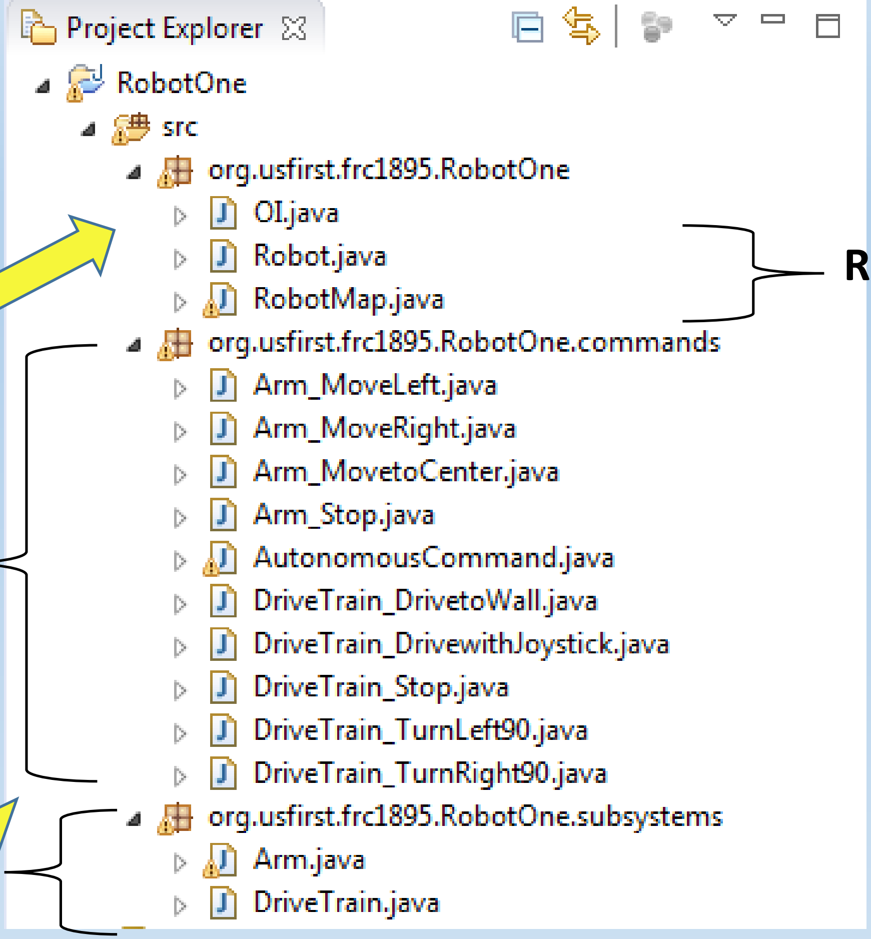
- Save the RobotBuilder Project
  - Select **RobotBuilder** => Save As => “C:_Robotics\2016\Eclipse_Projects\RobotOne”
- Export the Java Code
  - Select **RobotBuilder** => Export => Java Or “Java” on the Menu
- Import the Java Project into Eclipse
  - Select **Eclipse** => File => Import => General => Existing Projects into Workspace
  - Browse to project folder: e.g.: “C:_Robotics\2016\Eclipse_Projects\RobotOne”

RobotBuilder Project

Resulting JAVA Project

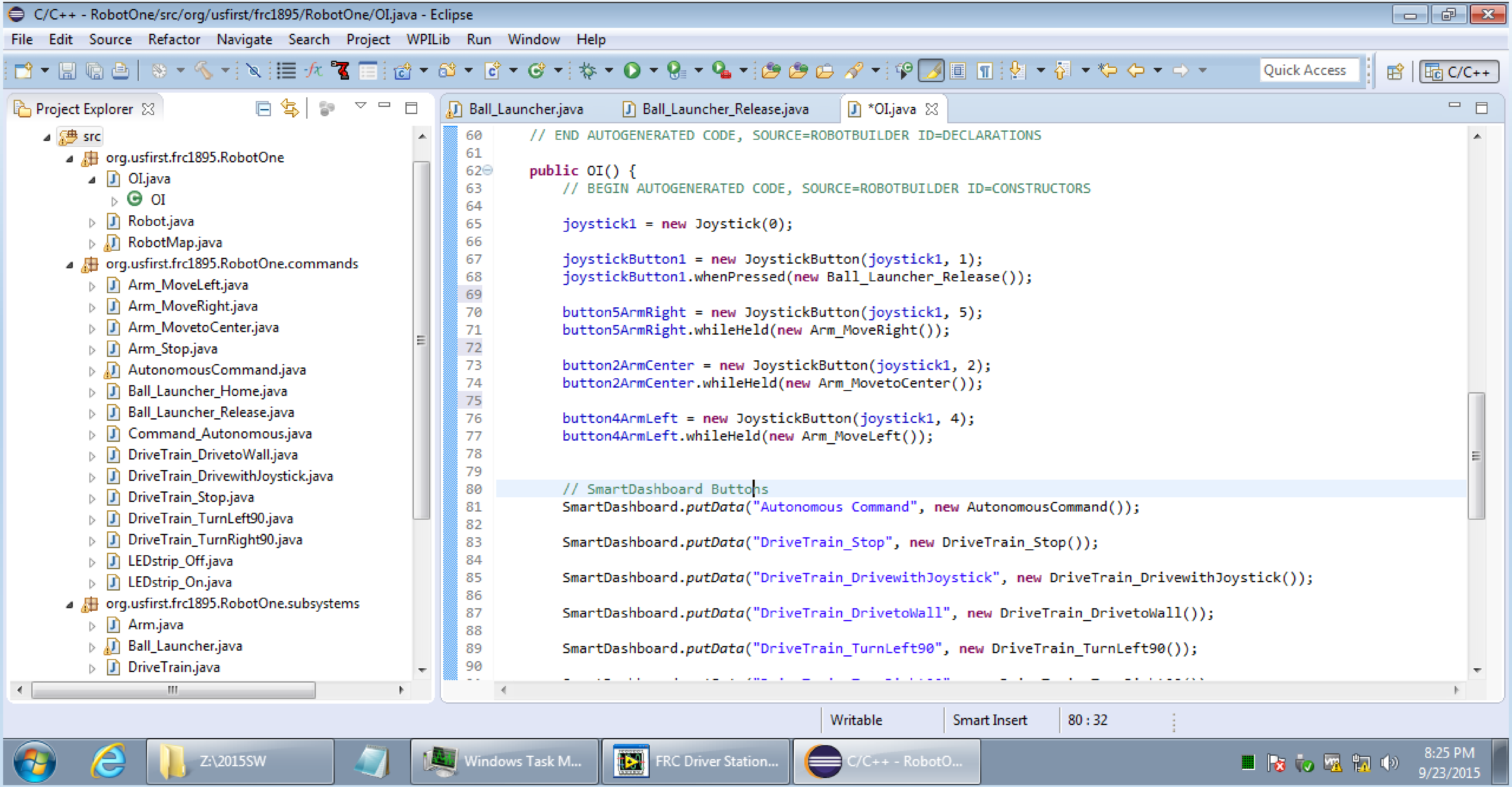


- RobotOne
  - Subsystems
    - DriveTrain
      - RobotDrive4
        - LeftFrontWheel
        - RightFrontWheel
        - LeftRearWheel
        - RightRearWheel
      - DriveTrain_Gyro
      - DriveTrain_RangeFinder
    - Arm
      - Arm_potentiometer
  - Operator Interface
    - Joystick1
      - Button4ArmLeft
      - Button2ArmCenter
      - Button5ArmRight
  - Commands
    - Autonomous Command
    - DriveTrain_Stop
    - DriveTrain_DrivewithJoystick
    - DriveTrain_DrivetoWall
    - DriveTrain_TurnLeft90
    - DriveTrain_TurnRight90
    - Arm_Stop
    - Arm_MoveLeft
    - Arm_MoveRight
    - Arm_MovetoCenter



- RobotOne
  - src
    - org.usfirst.frc1895.RobotOne
      - OI.java
      - Robot.java
      - RobotMap.java
    - org.usfirst.frc1895.RobotOne.commands
      - Arm_MoveLeft.java
      - Arm_MoveRight.java
      - Arm_MovetoCenter.java
      - Arm_Stop.java
      - AutonomousCommand.java
      - DriveTrain_DrivetoWall.java
      - DriveTrain_DrivewithJoystick.java
      - DriveTrain_Stop.java
      - DriveTrain_TurnLeft90.java
      - DriveTrain_TurnRight90.java
    - org.usfirst.frc1895.RobotOne.subsystems
      - Arm.java
      - DriveTrain.java

Robot Overhead



```java
    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS

    public OI() {
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS

        joystick1 = new Joystick(0);

        joystickButton1 = new JoystickButton(joystick1, 1);
        joystickButton1.whenPressed(new Ball_Launcher_Release());

        button5ArmRight = new JoystickButton(joystick1, 5);
        button5ArmRight.whileHeld(new Arm_MoveRight());

        button2ArmCenter = new JoystickButton(joystick1, 2);
        button2ArmCenter.whileHeld(new Arm_MovetoCenter());

        button4ArmLeft = new JoystickButton(joystick1, 4);
        button4ArmLeft.whileHeld(new Arm_MoveLeft());


        // SmartDashboard Buttons
        SmartDashboard.putData("Autonomous Command", new AutonomousCommand());

        SmartDashboard.putData("DriveTrain_Stop", new DriveTrain_Stop());

        SmartDashboard.putData("DriveTrain_DrivewithJoystick", new DriveTrain_DrivewithJoystick());

        SmartDashboard.putData("DriveTrain_DrivetoWall", new DriveTrain_DrivetoWall());

        SmartDashboard.putData("DriveTrain_TurnLeft90", new DriveTrain_TurnLeft90());

```

Step 4: Finish the Software in Eclipse

Detailed Coding Begins ……….

At this point you have a lot of code that does **nothing**!

Need to tie inputs to outputs using methods

Need to create Methods() within the Subsystems

Commands call Methods ()

User Interface calls Commands

A "Command" object Tells a "Subsystem" object to do something by way of "Methods"

Enter the FRC IterativeRobot Class

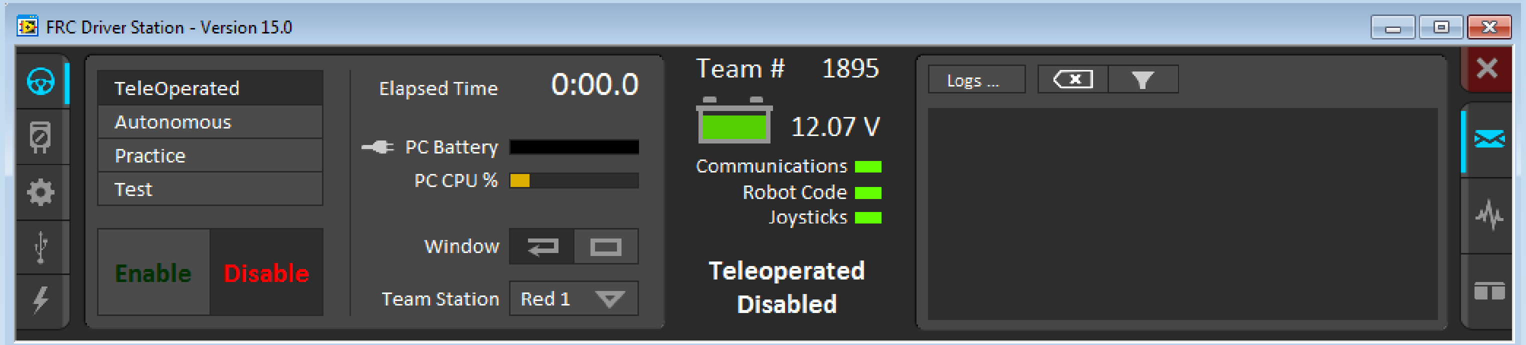
RobotBuilder creates an framework or template based on the IterativeRobot

Supports:

- Robot software initialization
- Autonomous mode
- TeleOperate mode

During development, the operating mode is selected by the **DriversStation**

During Competition, the mode is controlled by the Field Management System



FRC Driver Station - Version 15.0

Elapsed Time 0:00.0

Team # 1895

12.07 V

Team Station Red 1

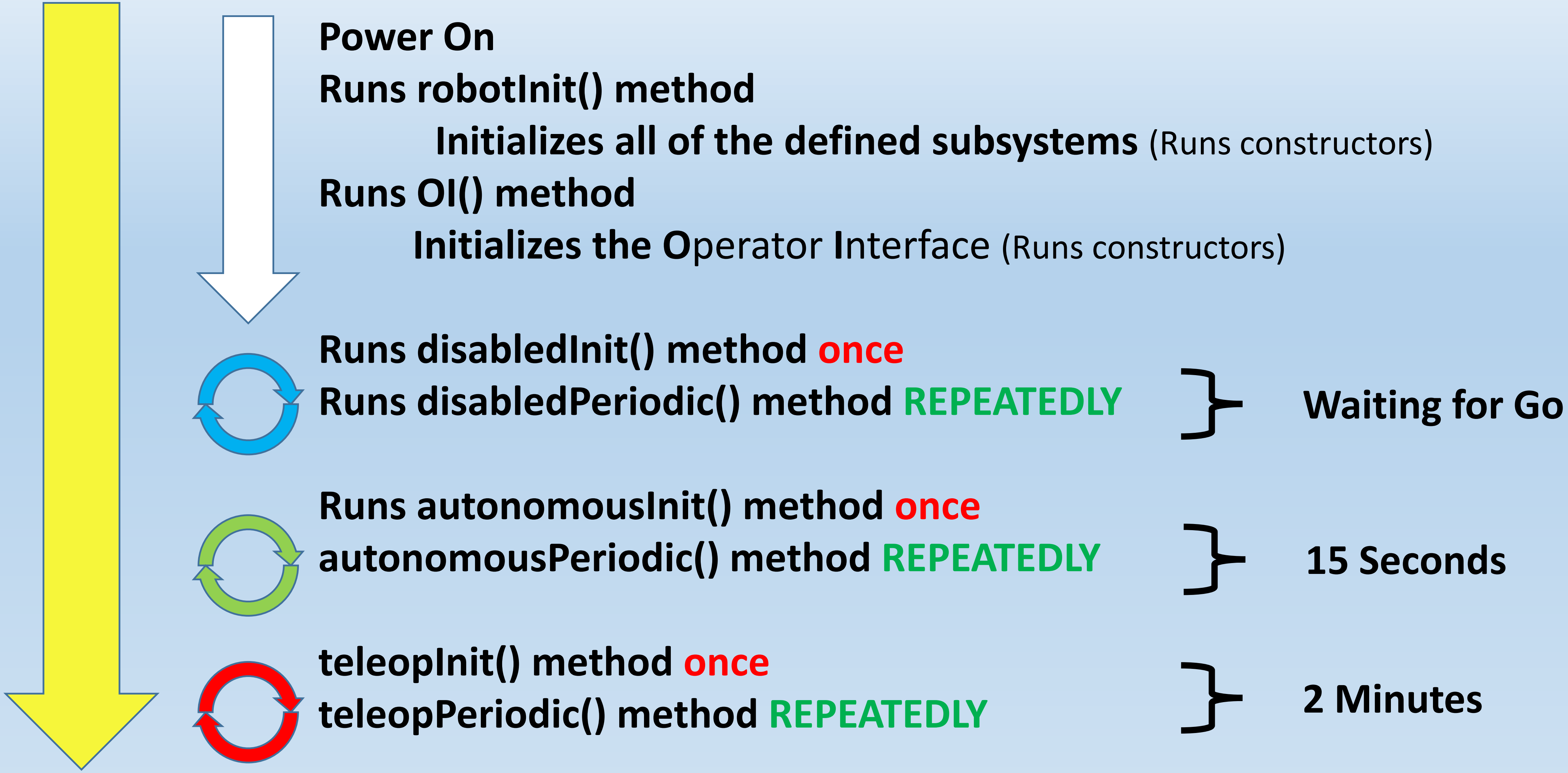
Communications 🟩

Robot Code 🟩

Joysticks 🟩

Teleoperated Disabled

High level structure of the IterativeRobot



Power On

Runs robotInit() method

Initializes all of the defined subsystems (Runs constructors)

Runs OI() method

Initializes the Operator Interface (Runs constructors)

Runs disabledInit() method once

Runs disabledPeriodic() method REPEATEDLY

} **Waiting for Go**

Runs autonomousInit() method once

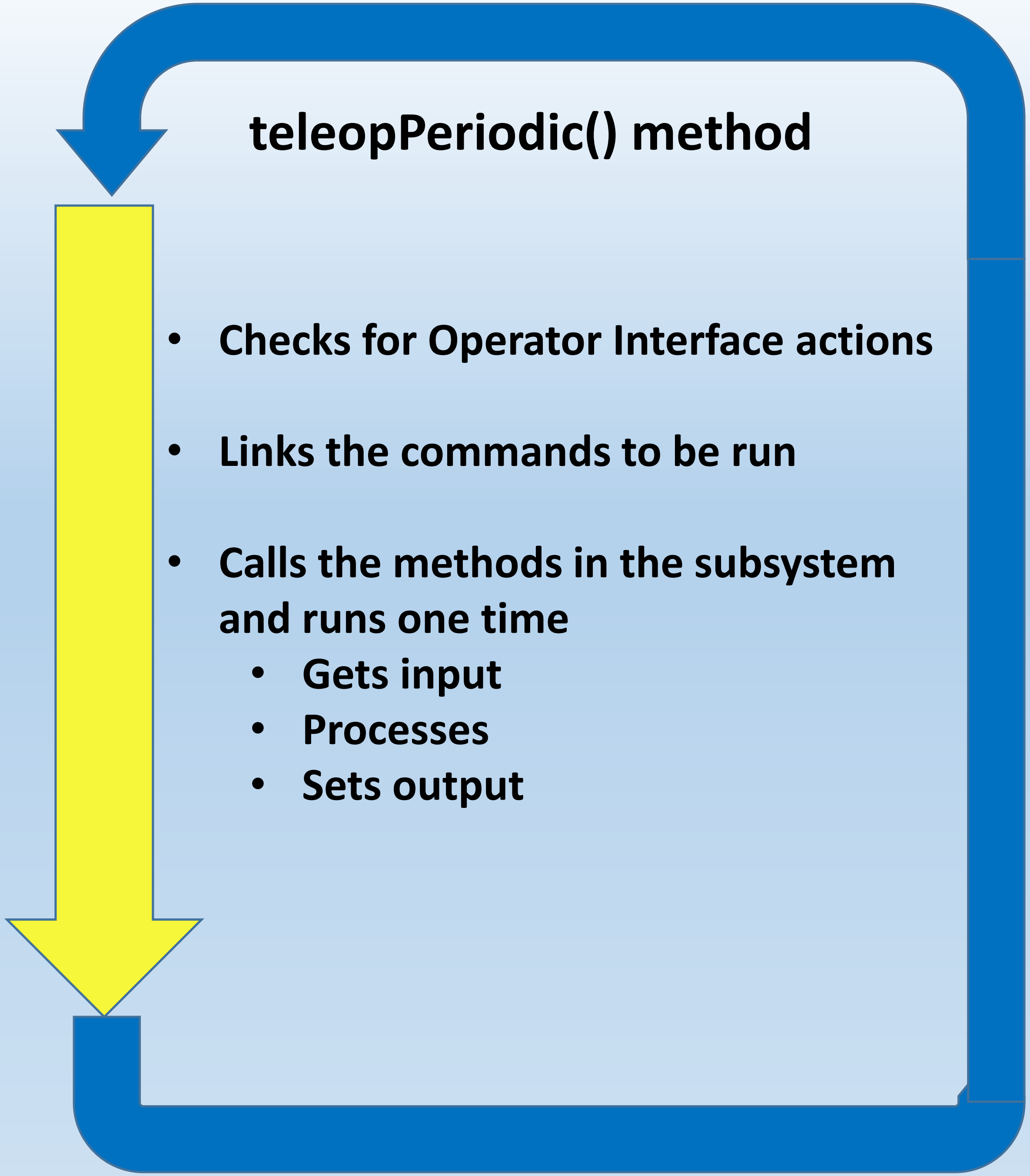
autonomousPeriodic() method REPEATEDLY

} **15 Seconds**

teleopInit() method once

teleopPeriodic() method REPEATEDLY

} **2 Minutes**



teleopPeriodic() method

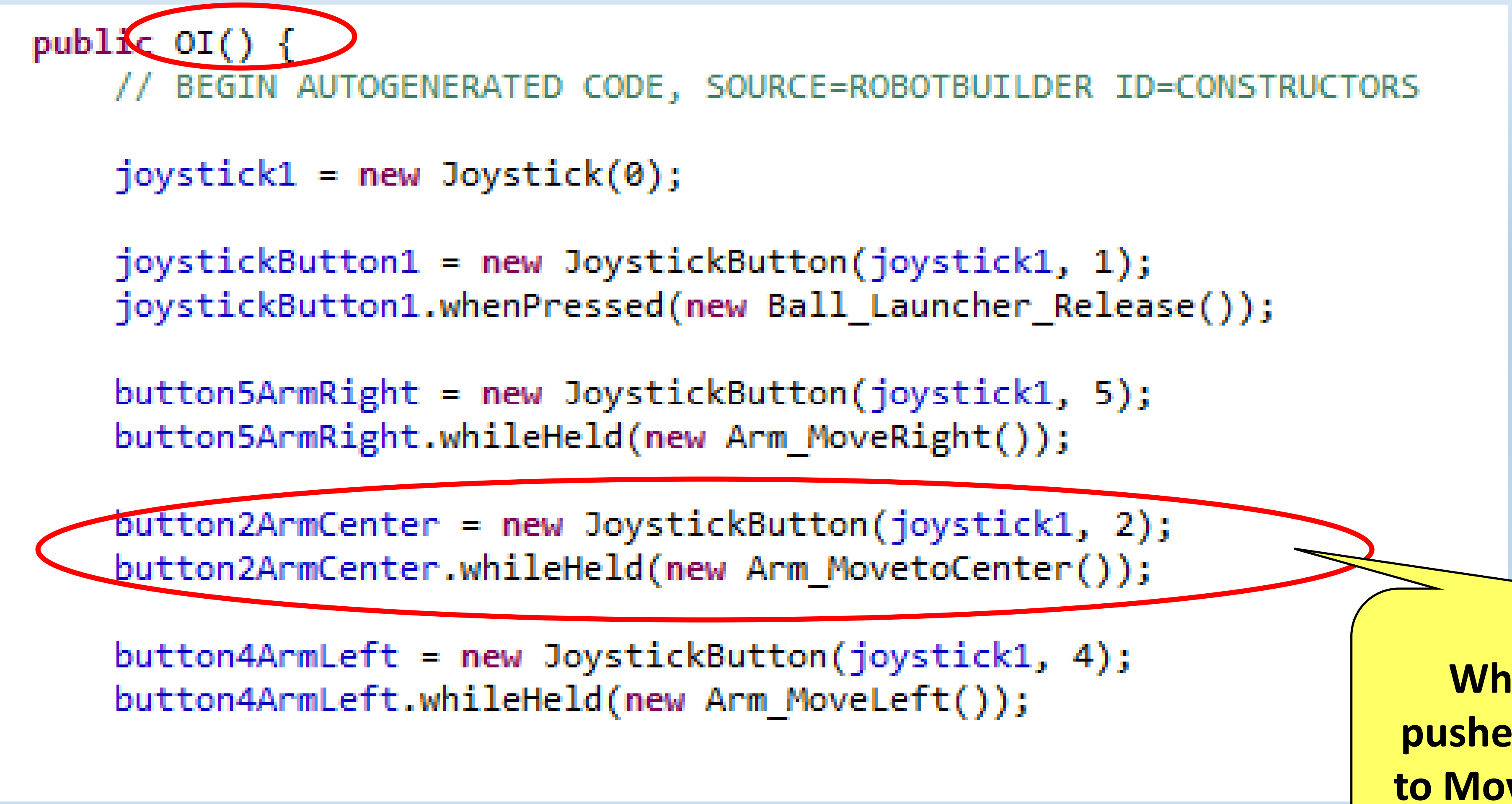
- Checks for Operator Interface actions
- Links the commands to be run
- Calls the methods in the subsystem and runs one time
  - Gets input
  - Processes
  - Sets output

- Code run in continuous loop for autonomousPeriodic() or teleopPeriodic()
- Loop is call 50 times per second (Wow!)

Simple Example of a Control Flow

- JoyStick button initiates a command
- Command calls a subsystem method
- Subsystem method takes action

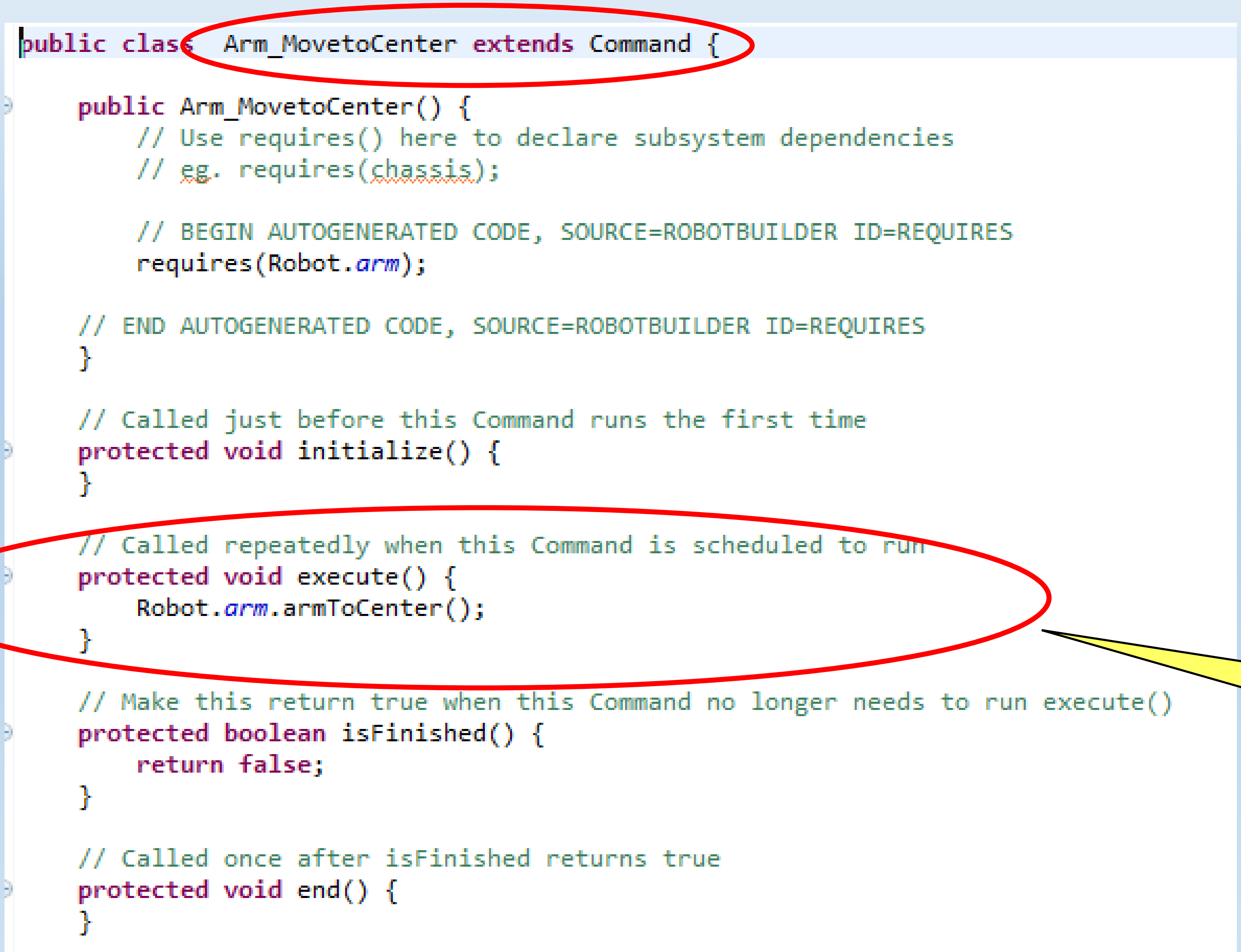
Simple Example – Operator Interface Calls the Command



```
public OI() {
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS

    joystick1 = new Joystick(0);

    joystickButton1 = new JoystickButton(joystick1, 1);
    joystickButton1.whenPressed(new Ball_Launcher_Release());

    button5ArmRight = new JoystickButton(joystick1, 5);
    button5ArmRight.whileHeld(new Arm_MoveRight());

    button2ArmCenter = new JoystickButton(joystick1, 2);
    button2ArmCenter.whileHeld(new Arm_MovetoCenter());

    button4ArmLeft = new JoystickButton(joystick1, 4);
    button4ArmLeft.whileHeld(new Arm_MoveLeft());
```

When Button 2 is pushed, the command to Move the Arm is run

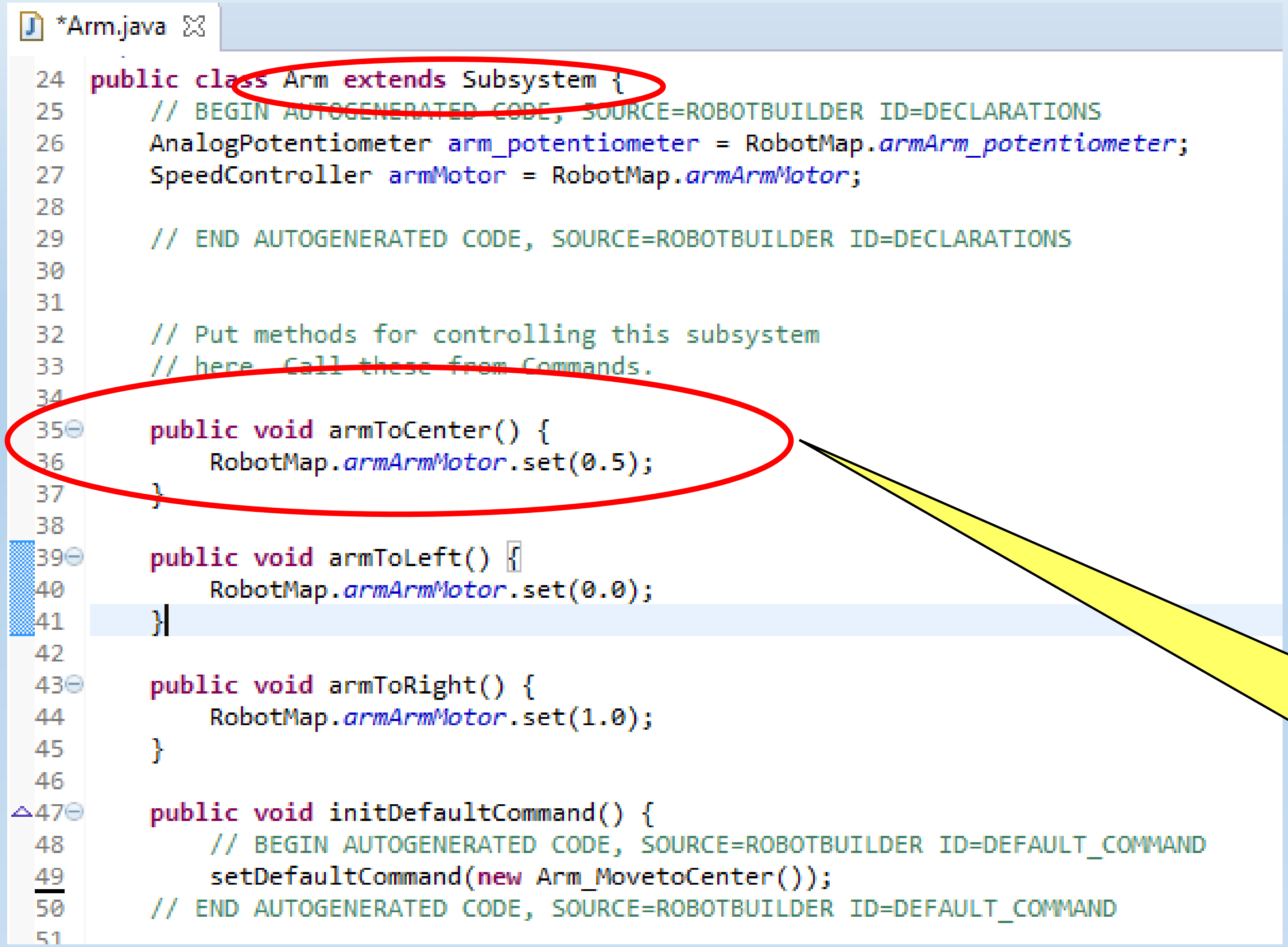
Simple Example – Commands Calls the Subsystem Method



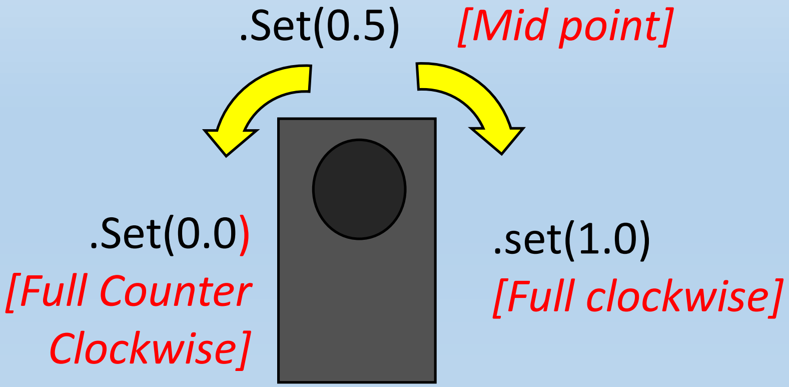
```
public class  Arm_MovetoCenter extends Command {

    public Arm_MovetoCenter() {
        // Use requires() here to declare subsystem dependencies
        // eg. requires(chassis);

        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
        requires(Robot.arm);

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
    }

    // Called just before this Command runs the first time
    protected void initialize() {
    }

    // Called repeatedly when this Command is scheduled to run
    protected void execute() {
        Robot.arm.armToCenter();
    }

    // Make this return true when this Command no longer needs to run execute()
    protected boolean isFinished() {
        return false;
    }

    // Called once after isFinished returns true
    protected void end() {
    }
```

Commands call Subsystem Methods

Simple Example –Subsystem Method controls Hardware



```
public class Arm extends Subsystem {
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
    AnalogPotentiometer arm_potentiometer = RobotMap.armArm_potentiometer;
    SpeedController armMotor = RobotMap.armArmMotor;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS


    // Put methods for controlling this subsystem
    // here. Call these from Commands.

    public void armToCenter() {
        RobotMap.armArmMotor.set(0.5);
    }

    public void armToLeft() {
        RobotMap.armArmMotor.set(0.0);
    }

    public void armToRight() {
        RobotMap.armArmMotor.set(1.0);
    }

    public void initDefaultCommand() {
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT_COMMAND
        setDefaultCommand(new Arm_MovetoCenter());
    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT_COMMAND
```



.Set(0.5) *[Mid point]*

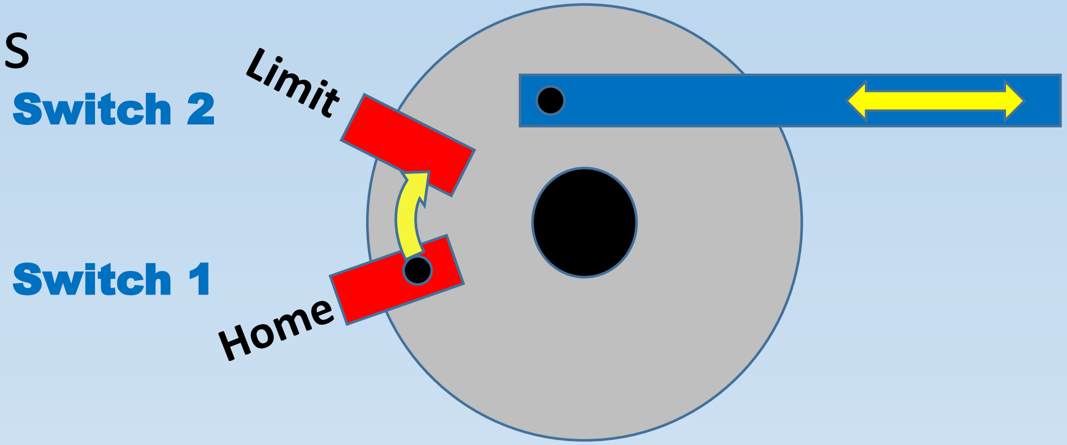
.Set(0.0) *[Full Counter Clockwise]*

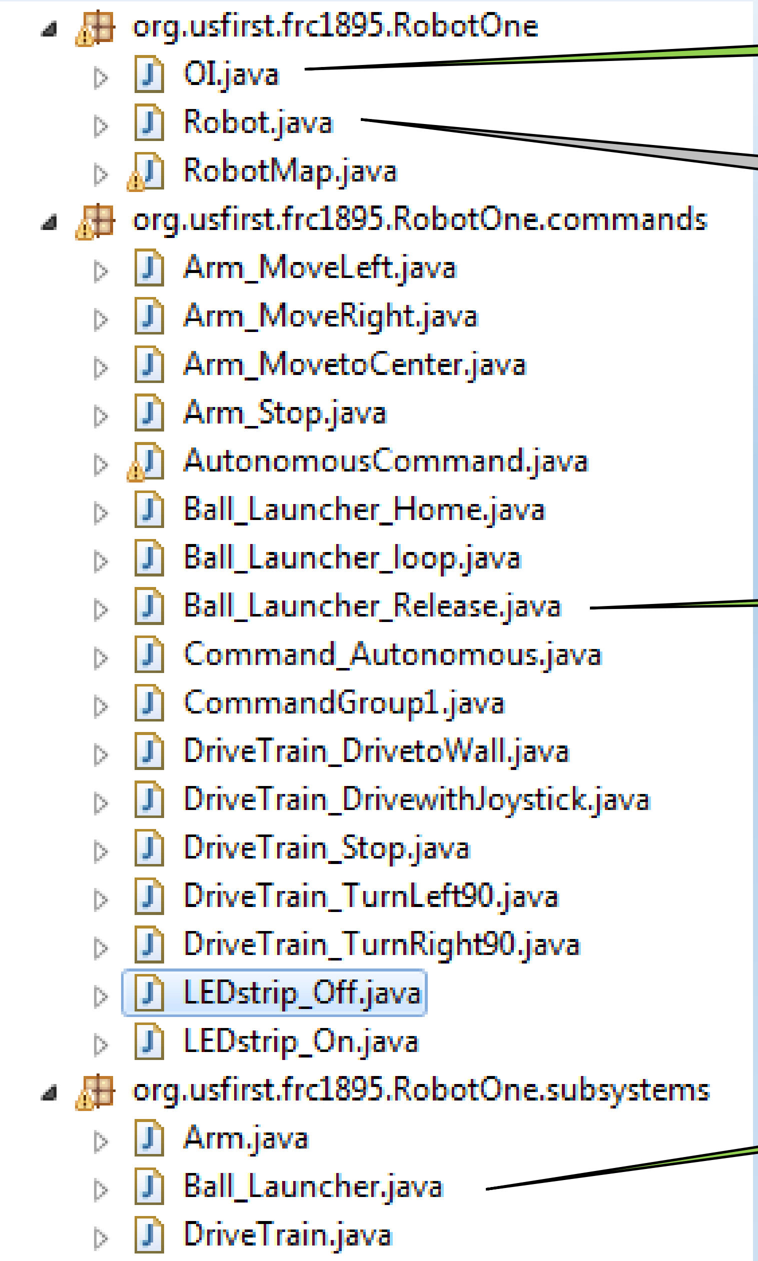
.set(1.0) *[Full clockwise]*

Methods Control the hardware with primitive commands

Another Example:
Add Methods – Release a ball held by a Trigger

- Pull the Trigger and release the Ball
- Create a method called **ReleaseBall()**
  - When called:
    - Start motor turning clockwise at 25% speed
    - Stop the motor when switch 2 closes
    - Reverse motor, turn counter-clockwise at 25%
    - Stop the motor when switch 1 closes





- org.usfirst.frc1895.RobotOne
  - OI.java
  - Robot.java
  - RobotMap.java
- org.usfirst.frc1895.RobotOne.commands
  - Arm_MoveLeft.java
  - Arm_MoveRight.java
  - Arm_MovetoCenter.java
  - Arm_Stop.java
  - AutonomousCommand.java
  - Ball_Launcher_Home.java
  - Ball_Launcher_loop.java
  - Ball_Launcher_Release.java
  - Command_Autonomous.java
  - CommandGroup1.java
  - DriveTrain_DrivetoWall.java
  - DriveTrain_DrivewithJoystick.java
  - DriveTrain_Stop.java
  - DriveTrain_TurnLeft90.java
  - DriveTrain_TurnRight90.java
  - LEDstrip_Off.java
  - LEDstrip_On.java
- org.usfirst.frc1895.RobotOne.subsystems
  - Arm.java
  - Ball_Launcher.java
  - DriveTrain.java

Operator Interface

Pull a trigger

Robot Top file

The Operator Interface Calls Commands

Commands

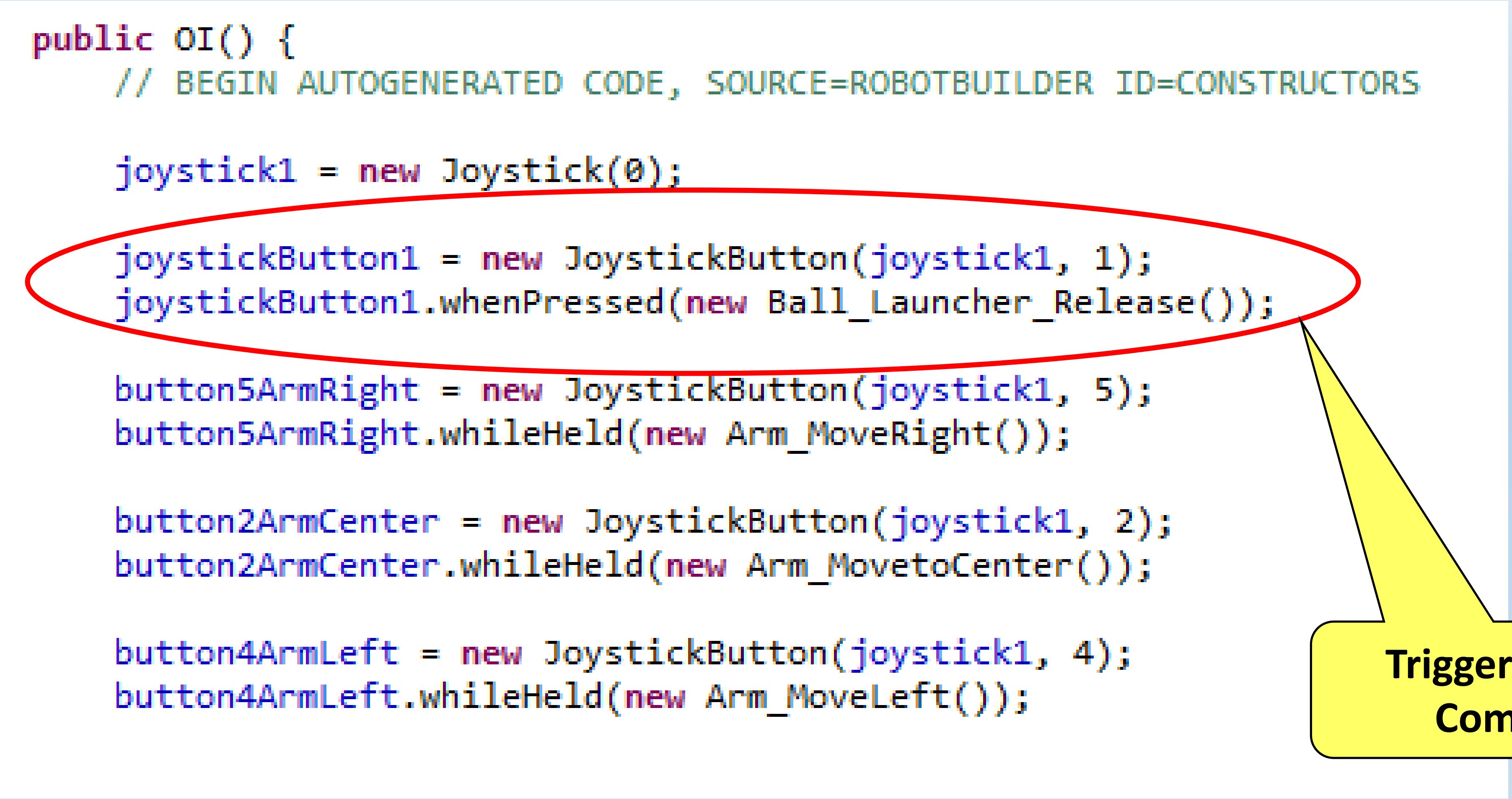
Fire the Ball

The Commands Calls Methods within the Subsystems

Subsystems

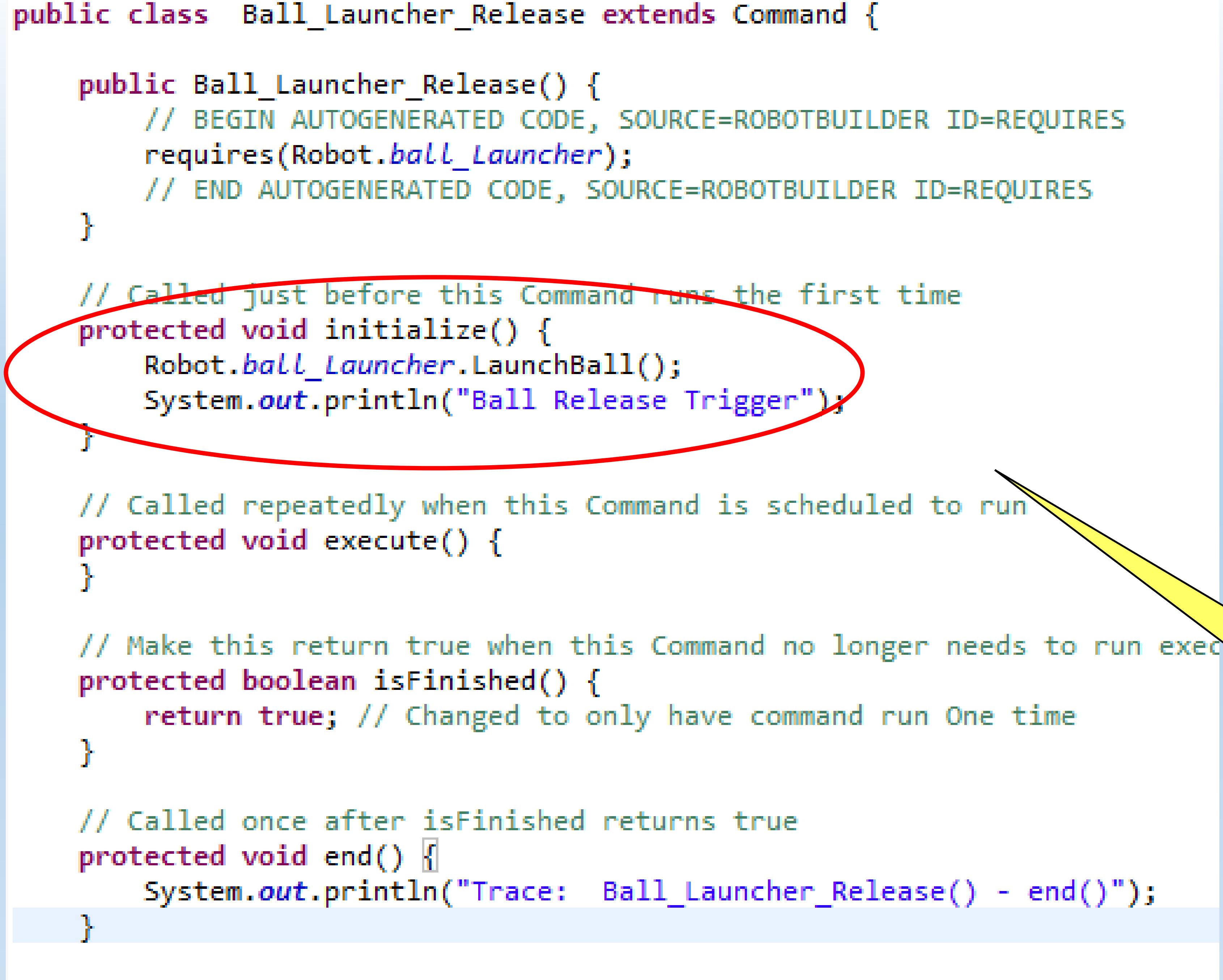
ReleaseBall()

OI calls a command



```
public OI() {
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS

    joystick1 = new Joystick(0);

    joystickButton1 = new JoystickButton(joystick1, 1);
    joystickButton1.whenPressed(new Ball_Launcher_Release());

    button5ArmRight = new JoystickButton(joystick1, 5);
    button5ArmRight.whileHeld(new Arm_MoveRight());

    button2ArmCenter = new JoystickButton(joystick1, 2);
    button2ArmCenter.whileHeld(new Arm_MovetoCenter());

    button4ArmLeft = new JoystickButton(joystick1, 4);
    button4ArmLeft.whileHeld(new Arm_MoveLeft());
```

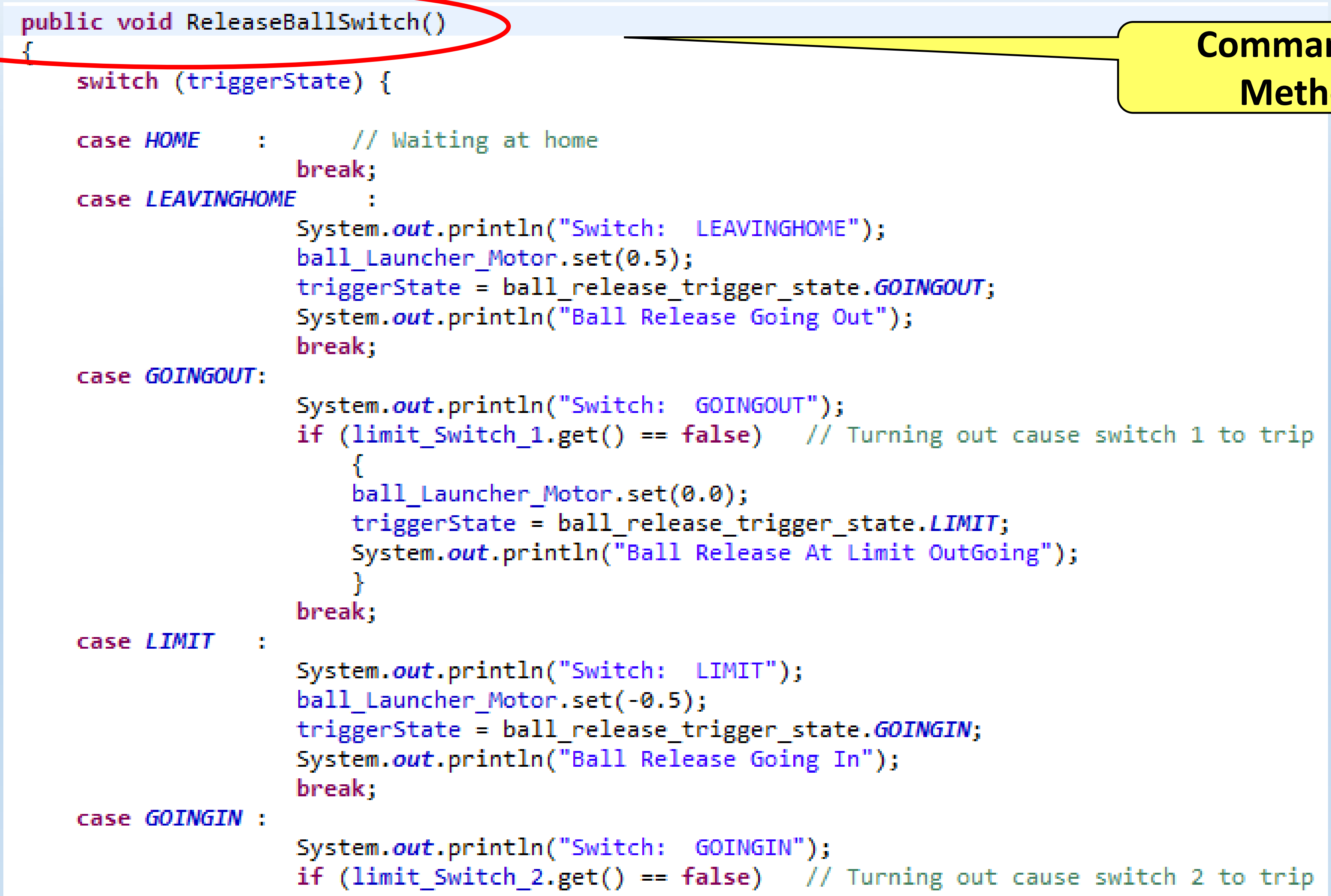
Trigger calls the Command



```
public class  Ball_Launcher_Release extends Command {

    public Ball_Launcher_Release() {
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
        requires(Robot.ball_Launcher);
        // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
    }

    // Called just before this Command runs the first time
    protected void initialize() {
        Robot.ball_Launcher.LaunchBall();
        System.out.println("Ball Release Trigger");
    }

    // Called repeatedly when this Command is scheduled to run
    protected void execute() {
    }

    // Make this return true when this Command no longer needs to run exec
    protected boolean isFinished() {
        return true; // Changed to only have command run One time
    }

    // Called once after isFinished returns true
    protected void end() {
        System.out.println("Trace:  Ball_Launcher_Release() - end()");
    }
```

Update Commands with subsystem Methods

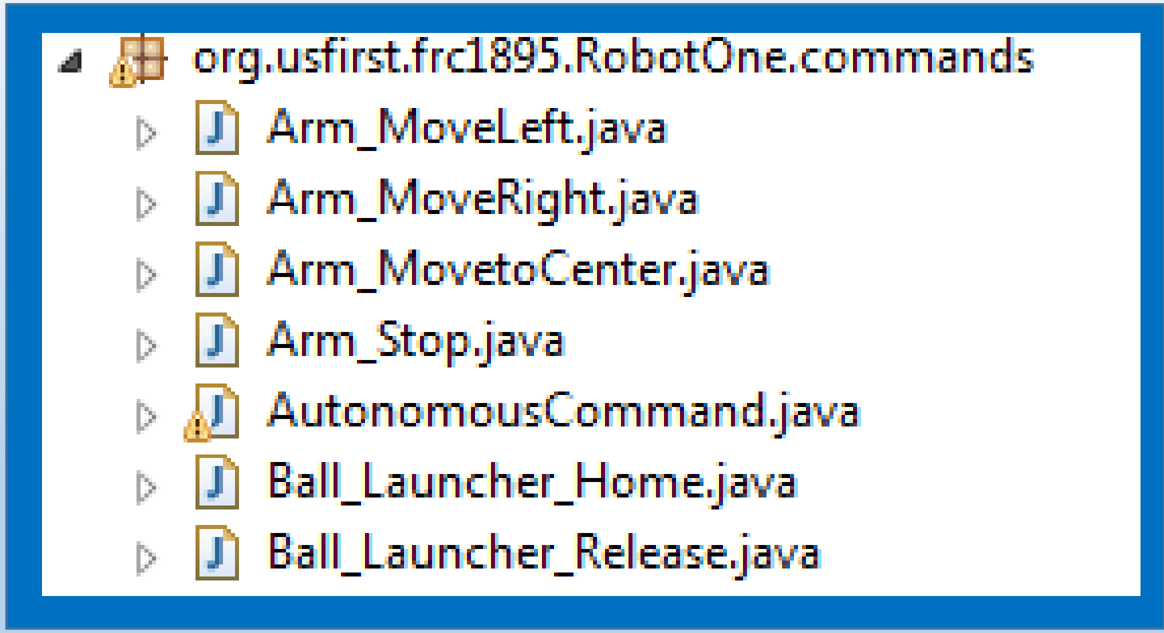
Methods to Release a ball held by a Trigger



```
public void ReleaseBallSwitch()
{
    switch (triggerState) {

    case HOME    :      // Waiting at home
                    break;
    case LEAVINGHOME    :
                    System.out.println("Switch:  LEAVINGHOME");
                    ball_Launcher_Motor.set(0.5);
                    triggerState = ball_release_trigger_state.GOINGOUT;
                    System.out.println("Ball Release Going Out");
                    break;
    case GOINGOUT:
                    System.out.println("Switch:  GOINGOUT");
                    if (limit_Switch_1.get() == false)   // Turning out cause switch 1 to trip
                        {
                        ball_Launcher_Motor.set(0.0);
                        triggerState = ball_release_trigger_state.LIMIT;
                        System.out.println("Ball Release At Limit OutGoing");
                        }
                    break;
    case LIMIT   :
                    System.out.println("Switch:  LIMIT");
                    ball_Launcher_Motor.set(-0.5);
                    triggerState = ball_release_trigger_state.GOINGIN;
                    System.out.println("Ball Release Going In");
                    break;
    case GOINGIN :
                    System.out.println("Switch:  GOINGIN");
                    if (limit_Switch_2.get() == false)   // Turning out cause switch 2 to trip
```

Command call Methods

Options for Commands



org.usfirst.frc1895.RobotOne.commands
- Arm_MoveLeft.java
- Arm_MoveRight.java
- Arm_MovetoCenter.java
- Arm_Stop.java
- AutonomousCommand.java
- Ball_Launcher_Home.java
- Ball_Launcher_Release.java

The Command based approach provides a great deal of flexibility

When a command is called you can call methods:

- Call a subsystem method one time at the beginning
- Repeatedly
- Call a subsystem method one time at the end of a command

Options for Commands

```
public class  Arm_MovetoCenter extends Command {
public Arm_MovetoCenter() {
requires(Robot.arm);
}
    // Called just before this Command runs the first time
    protected void initialize() {
    }
    // Called repeatedly when this Command is scheduled to run
    protected void execute() {
    }
    // Make this return true when this Command no longer needs to run execute()
    protected boolean isFinished() {
        return false;
    }
    // Called once after isFinished returns true
    protected void end() {
    }
    // Called when another command which requires one or more of the same
    // subsystems is scheduled to run
    protected void interrupted() {
    }
}
```

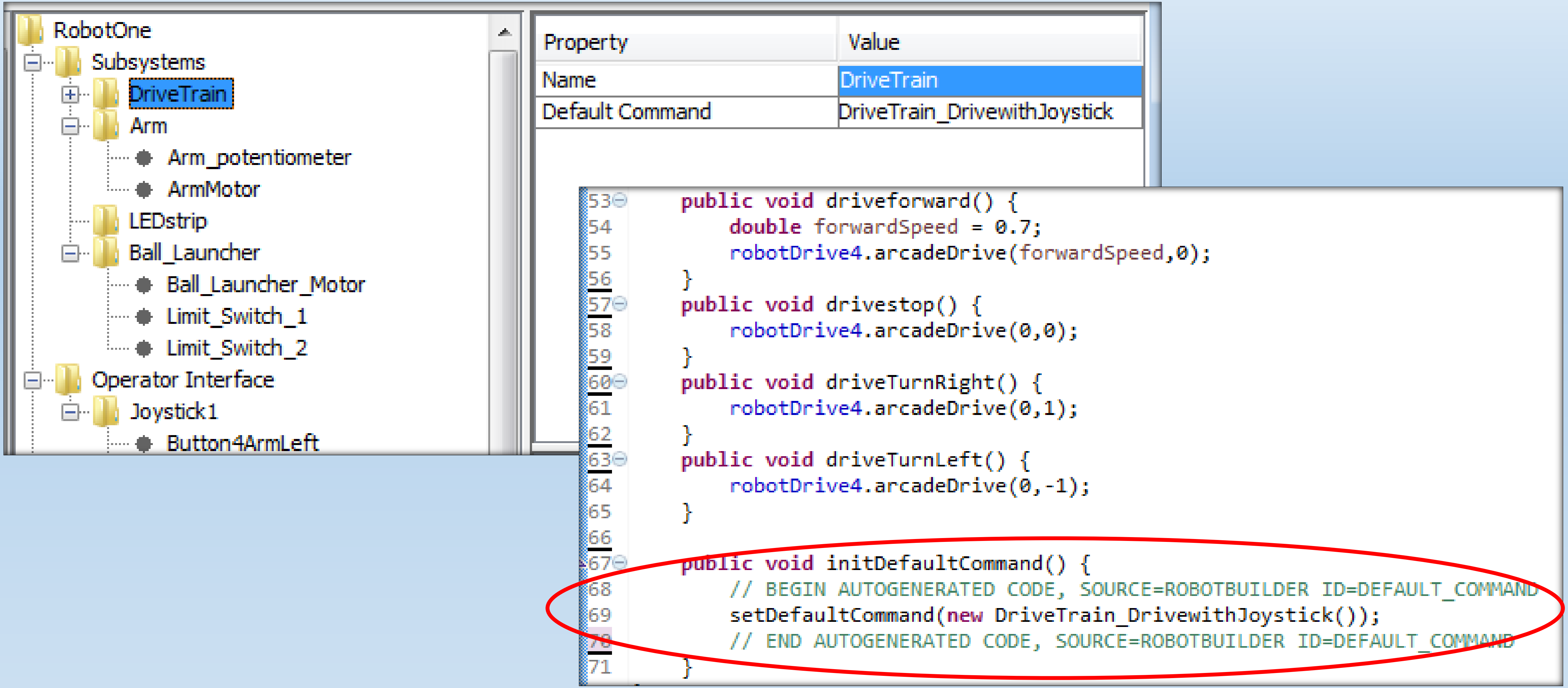
Default Command

- The command performed when no other commands are given to a subsystem

- Each subsystem may, but is not required to, have a default command which is scheduled whenever the subsystem is idle

- The most common example of a default command is a command for the drivetrain that implements the normal joystick control.

Default Command



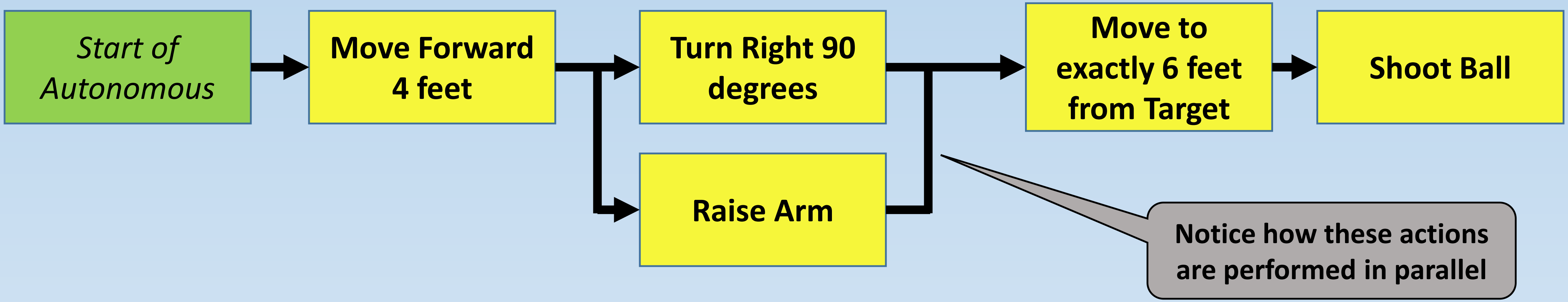
Property	Value
Name	DriveTrain
Default Command	DriveTrain_DrivewithJoystick

```
    public void driveforward() {
        double forwardSpeed = 0.7;
        robotDrive4.arcadeDrive(forwardSpeed,0);
    }
    public void drivestop() {
        robotDrive4.arcadeDrive(0,0);
    }
    public void driveTurnRight() {
        robotDrive4.arcadeDrive(0,1);
    }
    public void driveTurnLeft() {
        robotDrive4.arcadeDrive(0,-1);
    }

    public void initDefaultCommand() {
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT_COMMAND
        setDefaultCommand(new DriveTrain_DrivewithJoystick());
        // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT_COMMAND
    }
```

Autonomous Command

Create an **Autonomous Command Group**

- Tells the Robot what to do when in Autonomous Mode
- Consists of a sequence of other command
- Performed in series or parallel

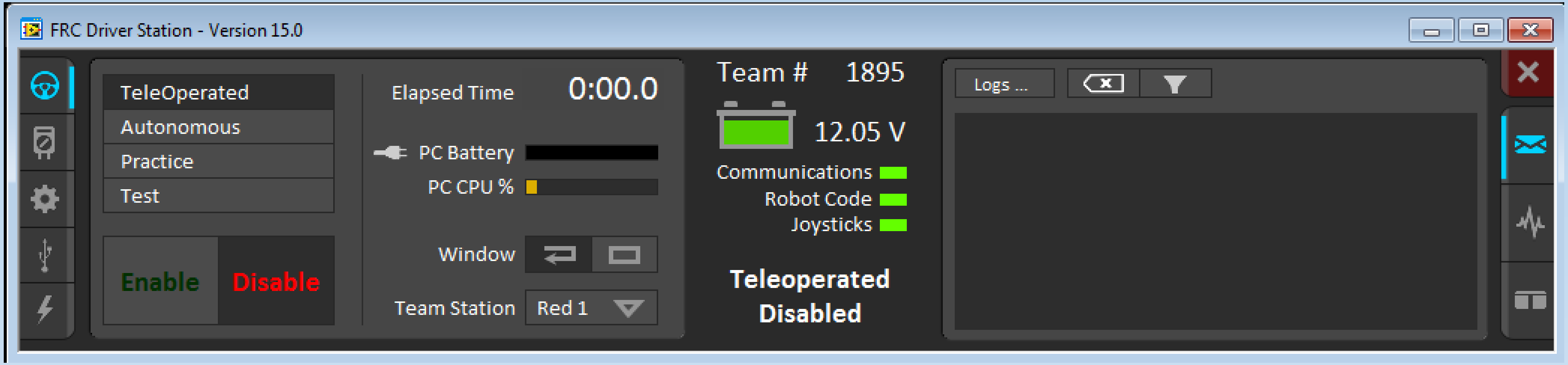


Notice how these actions are performed in parallel

Go!

- Compile => Deploy => Execute

Screen Capture of Eclipse
Start Dashboard
Enable Smartdashboard
Compile and Download
Enable and test

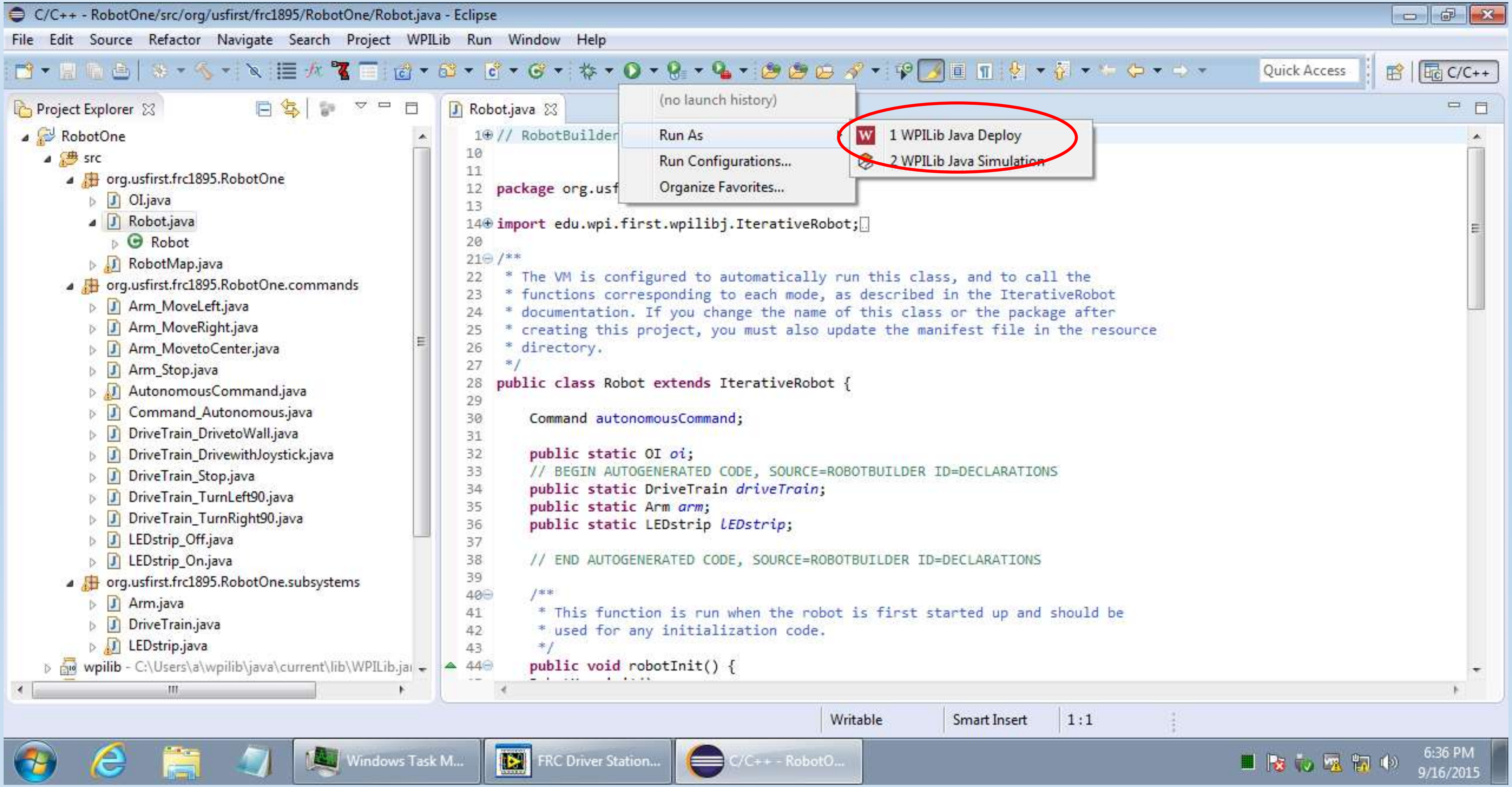


FRC Driver Station - Version 15.0

Elapsed Time 0:00.0
Team Station Red 1
Team # 1895
12.05 V
Communications 🟩
Robot Code 🟩
Joysticks 🟩

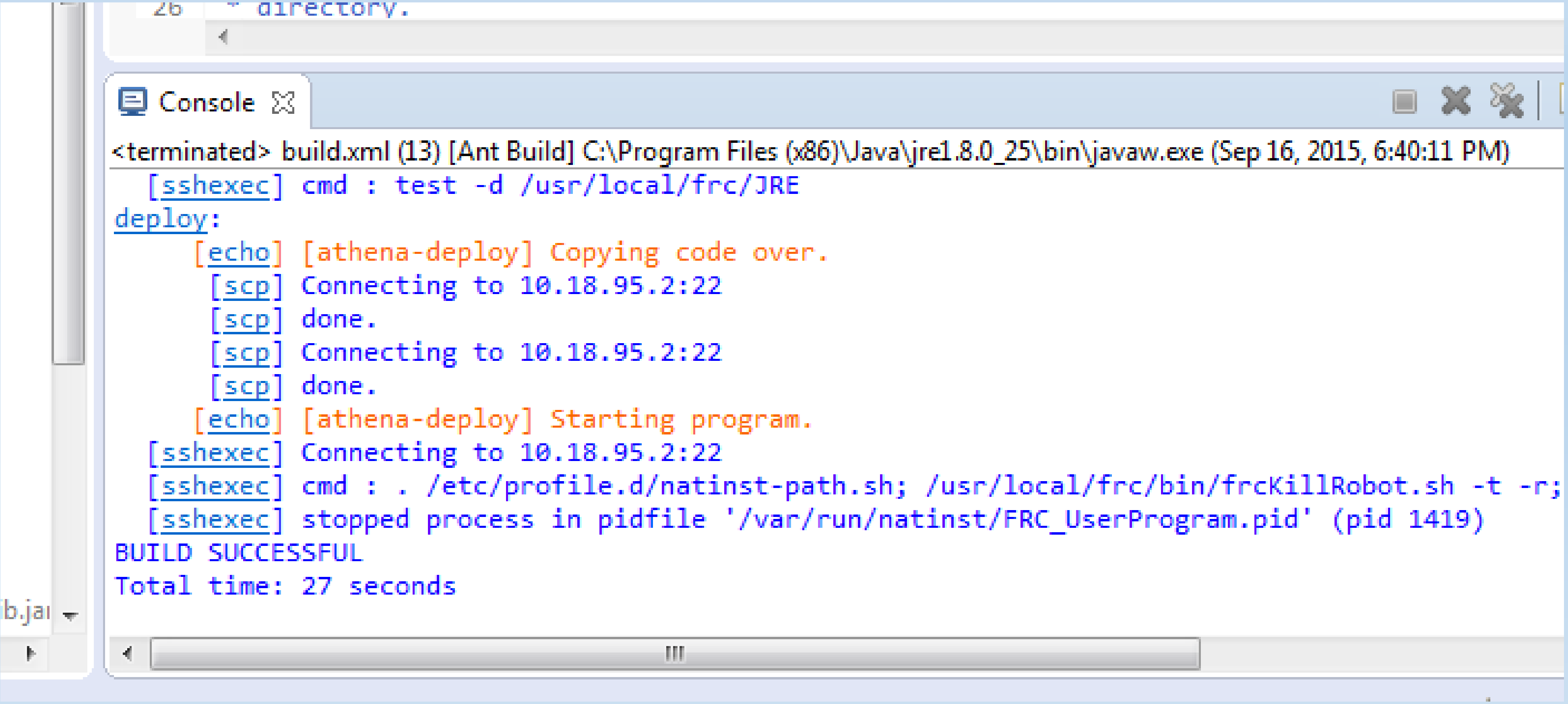
Teleoperated
Disabled

Compile and Deploy



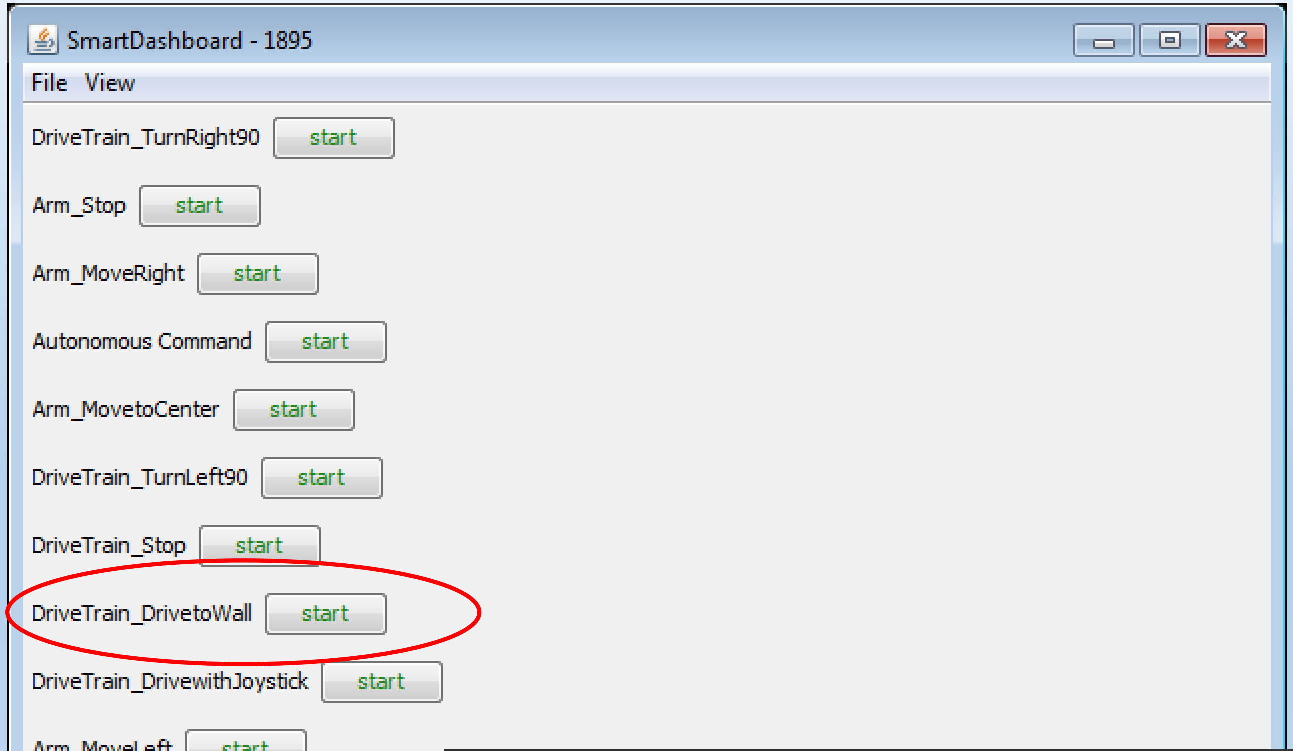
```
// RobotBuilder


package org.usf

import edu.wpi.first.wpilibj.IterativeRobot;

/**
 * The VM is configured to automatically run this class, and to call the
 * functions corresponding to each mode, as described in the IterativeRobot
 * documentation. If you change the name of this class or the package after
 * creating this project, you must also update the manifest file in the resource
 * directory.
 */
public class Robot extends IterativeRobot {

    Command autonomousCommand;

    public static OI oi;
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
    public static DriveTrain driveTrain;
    public static Arm arm;
    public static LEDstrip lEDstrip;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS

    /**
     * This function is run when the robot is first started up and should be
     * used for any initialization code.
     */
    public void robotInit() {
```

Console Display of Successful Deployment



```
<terminated> build.xml (13) [Ant Build] C:\Program Files (x86)\Java\jre1.8.0_25\bin\javaw.exe (Sep 16, 2015, 6:40:11 PM)
  [sshexec] cmd : test -d /usr/local/frc/JRE
deploy:
     [echo] [athena-deploy] Copying code over.
      [scp] Connecting to 10.18.95.2:22
      [scp] done.
      [scp] Connecting to 10.18.95.2:22
      [scp] done.
     [echo] [athena-deploy] Starting program.
  [sshexec] Connecting to 10.18.95.2:22
  [sshexec] cmd : . /etc/profile.d/natinst-path.sh; /usr/local/frc/bin/frcKillRobot.sh -t -r;
  [sshexec] stopped process in pidfile '/var/run/natinst/FRC_UserProgram.pid' (pid 1419)
BUILD SUCCESSFUL
Total time: 27 seconds
```

Smart Dashboard



SmartDashboard - 1895

File View

DriveTrain_TurnRight90 start

Arm_Stop start

Arm_MoveRight start

Autonomous Command start

Arm_MovetoCenter start

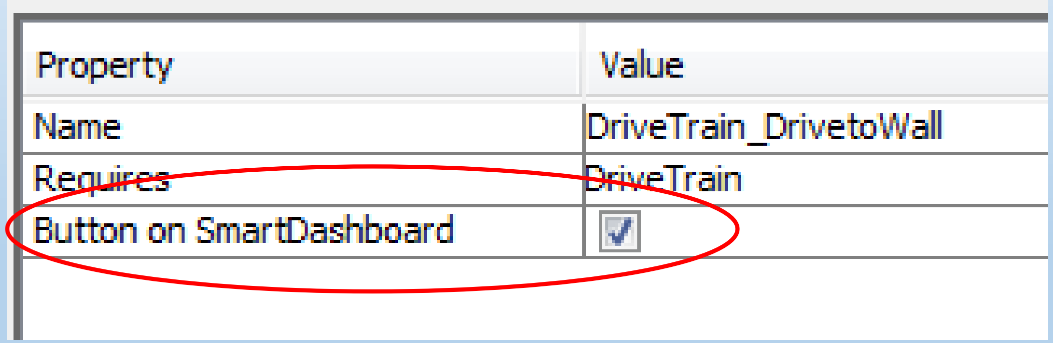
DriveTrain_TurnLeft90 start

DriveTrain_Stop start

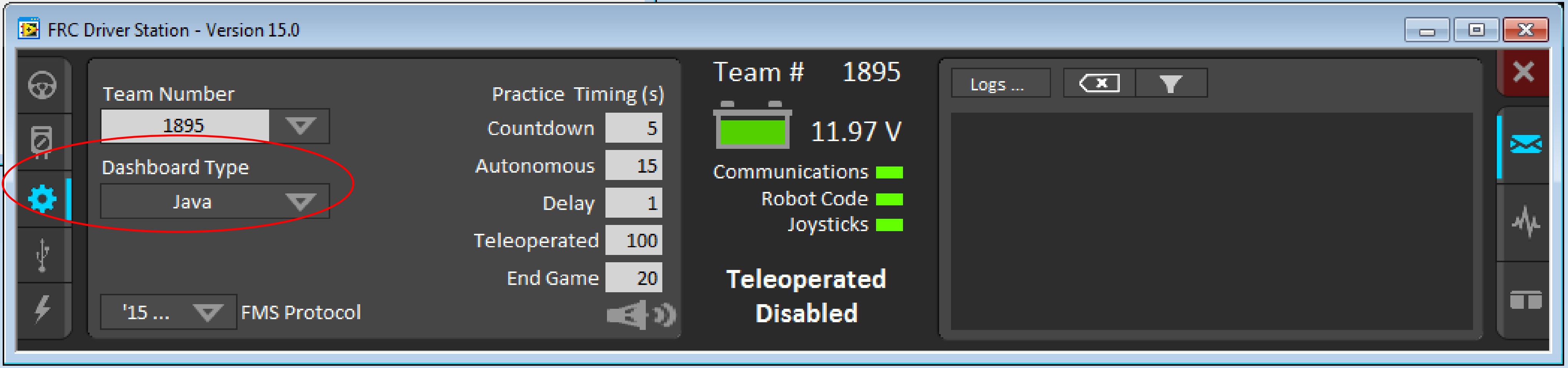
DriveTrain_DrivetoWall start

DriveTrain_DrivewithJoystick start

Arm_MoveLeft start



Property	Value
Name	DriveTrain_DrivetoWall
Requires	DriveTrain
Button on SmartDashboard	☑



FRC Driver Station - Version 15.0

Team Number: 1895

Dashboard Type: Java

'15 ... FMS Protocol

Practice Timing (s)

Countdown	5
Autonomous	15
Delay	1
Teleoperated	100
End Game	20

Team # 1895

11.97 V

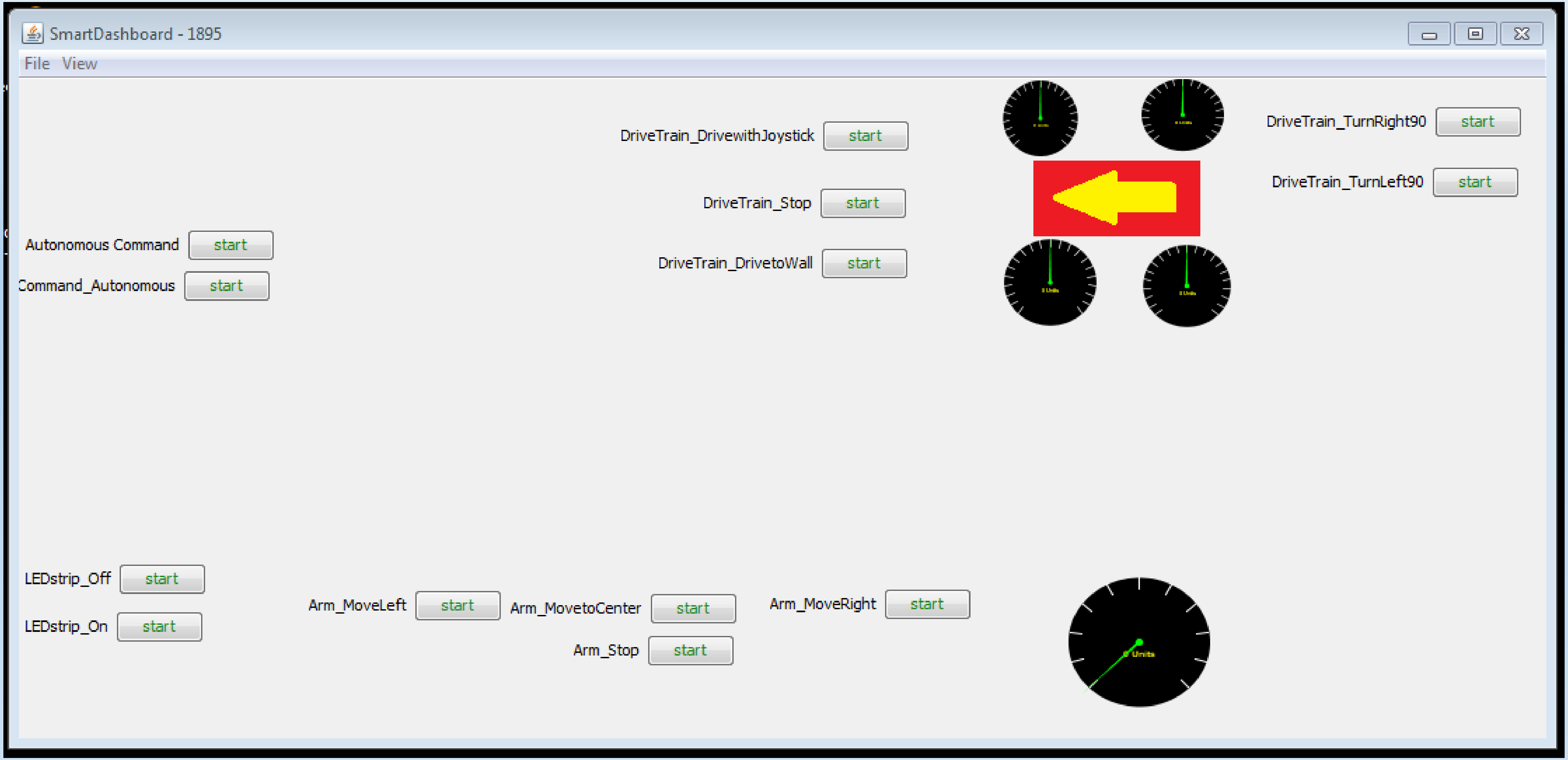
Communications 🟩

Robot Code 🟩

Joysticks 🟩

Teleoperated Disabled

Start the Smart Dashboard

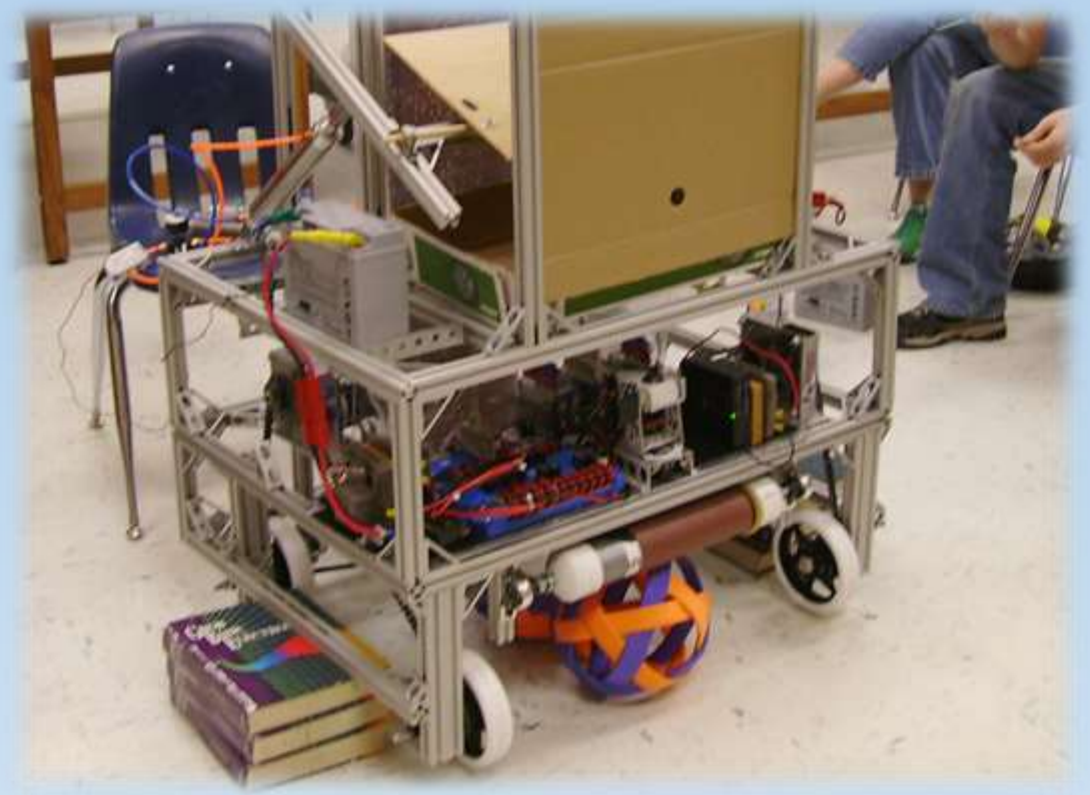


Initial Checkout!

Think Safety!

- Lift Robot wheels off of ground to prevent sudden unplanned movement
- Keep hands, fingers and hair away from Robot

- Test one function at a time
- Observe and think
- Take Notes



Updating the Code to Solve Problems

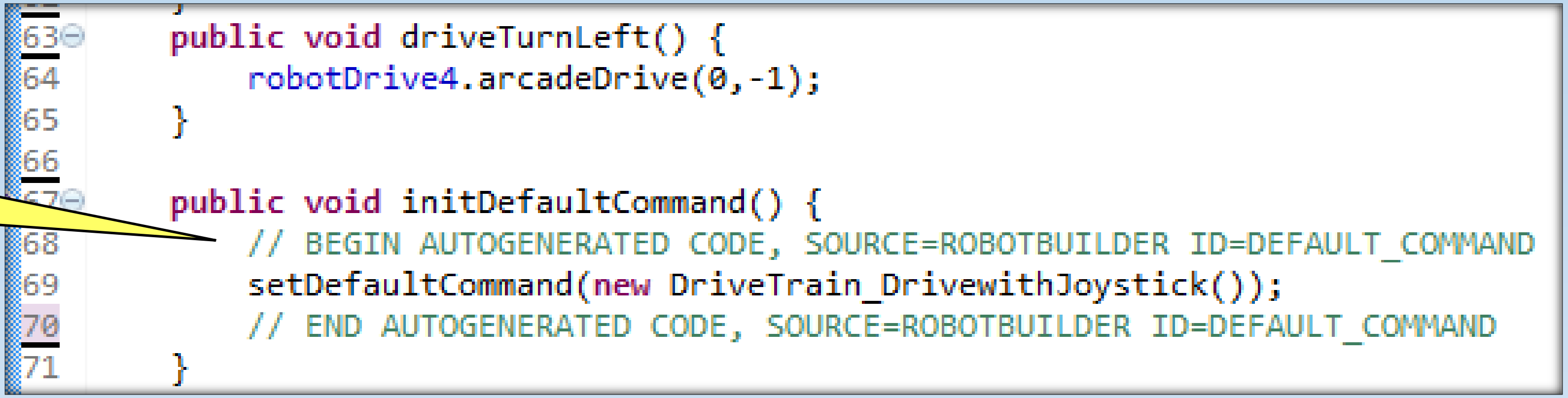
Code update can be performed either in Eclipse or RobotBuilder

- In Eclipse, edit code directly.   Always add comments

In RobotBuilder:

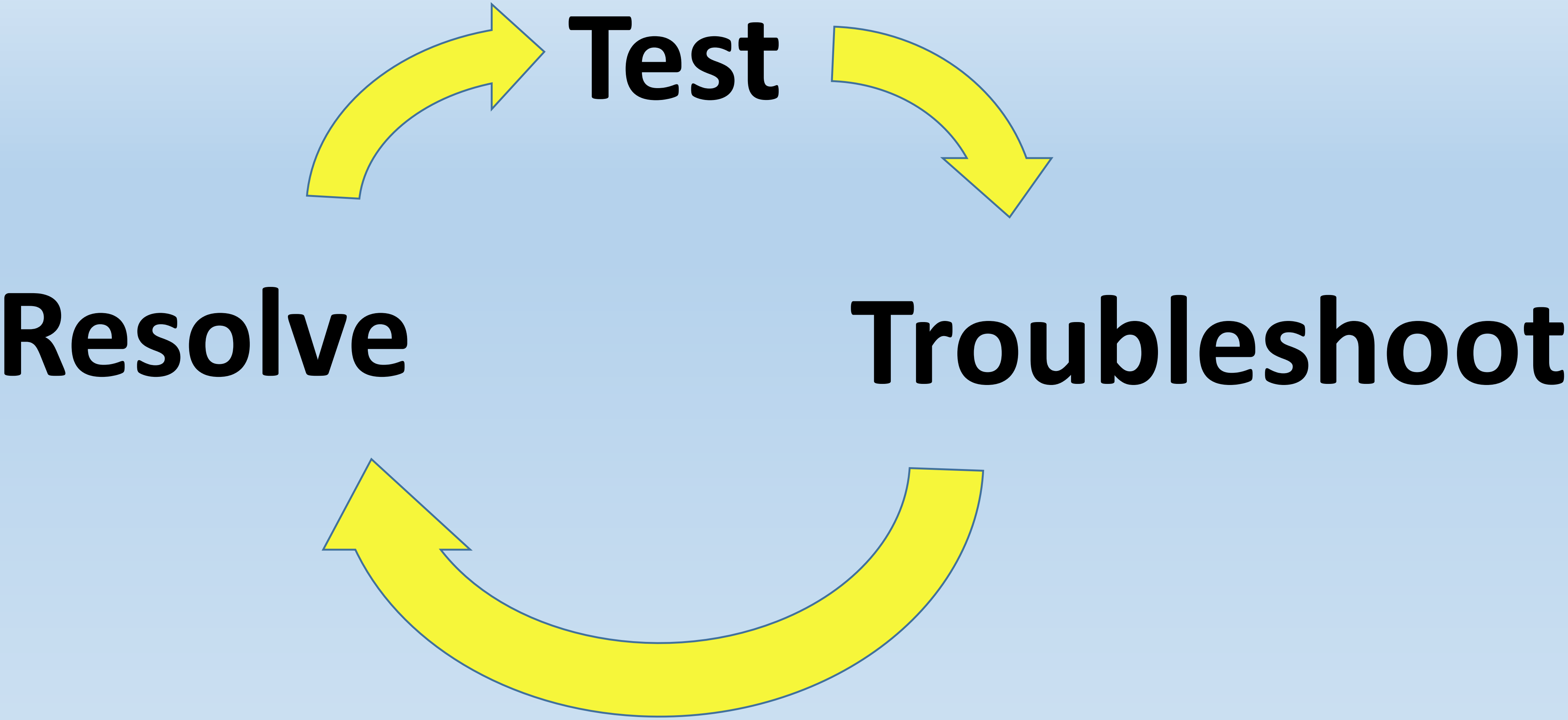
- Re-Open RobotBuilder, Make the required updates, then export
- In Eclipse, Refresh the project by selecting **"F5"**
- The robotBuilder updates will be shown, User provided code remains

Don't edit code between the RobotBuilder comments



```
    public void driveTurnLeft() {
        robotDrive4.arcadeDrive(0,-1);
    }

    public void initDefaultCommand() {
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT_COMMAND
        setDefaultCommand(new DriveTrain_DrivewithJoystick());
        // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT_COMMAND
    }
```

Step 5: Test, Troubleshoot, Resolve
Iterative Process…



Test

Troubleshoot

Resolve

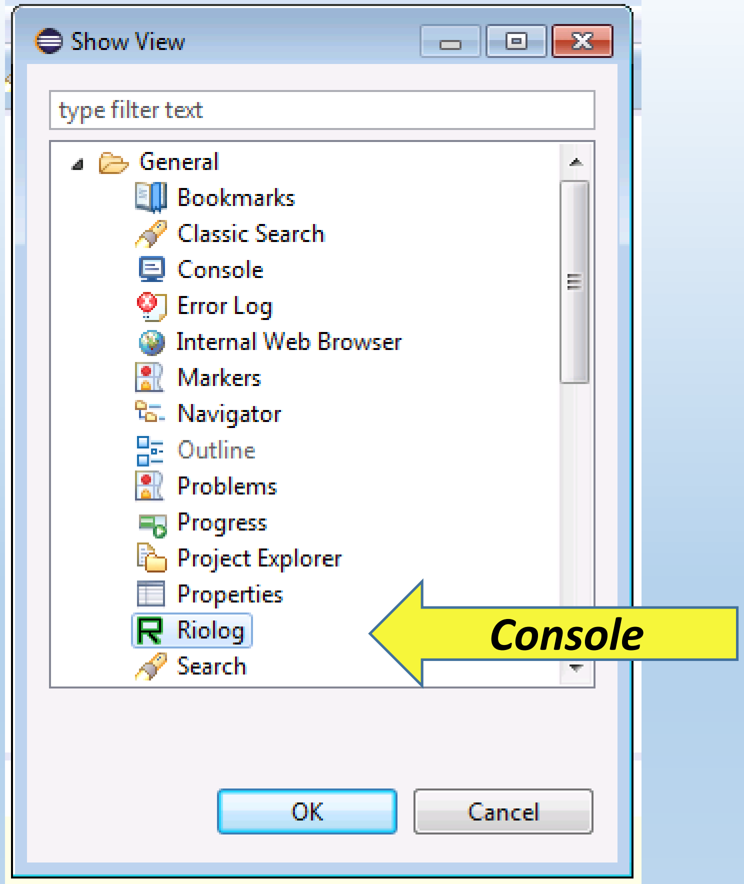
Troubleshooting Methods:

1) Print statements to the System Console (Riolog)

```
System.out.println("Ball Release Going Out");
```

2) Smart Dashboard

3) Many other approaches …



Console

?

λ Lambda Corps

FIRST Robotics Team # 1895

Manassas, VA
Osbourn High school

Resources

Guidance

https://wpilib.screenstepslive.com/s/4485

WPI Library

http://first.wpi.edu/FRC/roborio/stable/docs/java

http://first.wpi.edu/FRC/roborio/stable/docs/java/classedu_1_1wpi_1_1first_1_1wpilibj_1_1command_1_1WaitCommand.html

Eclipse:

http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.cdt.doc.user%2Ftasks%2Fcdt_t_comment_out.htm

FRC Roborio eclipse plugins zip file

http://first.wpi.edu/FRC/roborio/zipfile/

WPI ThinkTank

http://thinktank.wpi.edu/Portal

Eclipse Issues

- Unable to find a javac compiler; com.sun.tools.javac.Main is not on the classpath. Perhaps JAVA_HOME does not point to the JDK.
  - For eclipse
  - Right Click build.xml ---> Build path ---> configure buildpath ---> select libraries tab
  - click "Add library" ---> double click on [jre system library] ---> environments ---> installed jres ---> Add ---> standard vm
  - click on directory ---> Browse upto jdk [C:\Program Files\Java\jdk1.7.0_01]
  - finish
  - change the selection jre to jdk ---> click ok
- Import Errors
  - Eclipse => Source => Organize Imports
- Download errors due to Network Name Resolution
  - Right Click “Build.xml”,

Set the "JAVA_HOME“ Environment Variable

Instructions

1. Click "start“
2. Right click "computer“
3. Click "properties“
4. Click "advanced system settings“
5. Click "environment variables“
6. If "JAVA_HOME" is in the system variables then go to verify else click "new"
   "variable name" = "JAVA_HOME"
   "variable value" = the location of your Java JDK it is close to "C:\Program Files\Java\jdk1.8.0_25“
7. Click "ok"

Verify

1. Open a command window and enter: **`set | find "JAVA_HOME"`**
2. Should display something like "C:\Program Files\Java\jdk1.8.0_25“
3. In the command Window, enter: **`dir "C:\Program Files\Java\jdk1.8.0_25`**”
4. Should display the contents of the JAVA folder

Eclipse Networking Issues

- Download errors due to Network Name Resolution
  - Right Click “Build.xml”,

Terminology

Motor = Creates continuous rotational motion.
Speed of rotation controlled by a PWM interface
value of +1 to -1

Servo = Creates a limited rotational motion
Angle of output limited to +/- 100 degrees

Angle controlled by a PWM interface value of 0 to 1

LED = Light Emitting Diode – Lights up when voltage applied

Limit Switch = Provides an electrical connection when lever arm depressed.