



The Hitchhikers | FRC Team 2059

FRC PROGRAMMING IN WPILIB

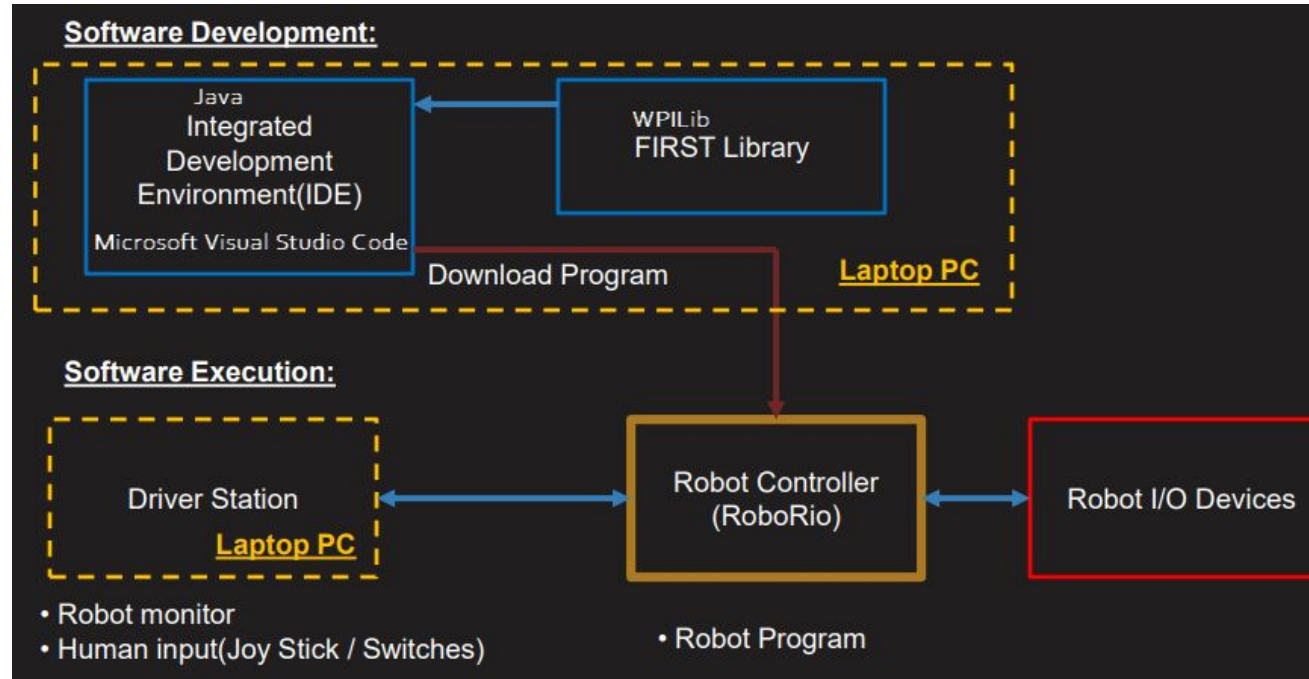
FRC Team 2059

By Mj

Co-Captain and Programming Lead

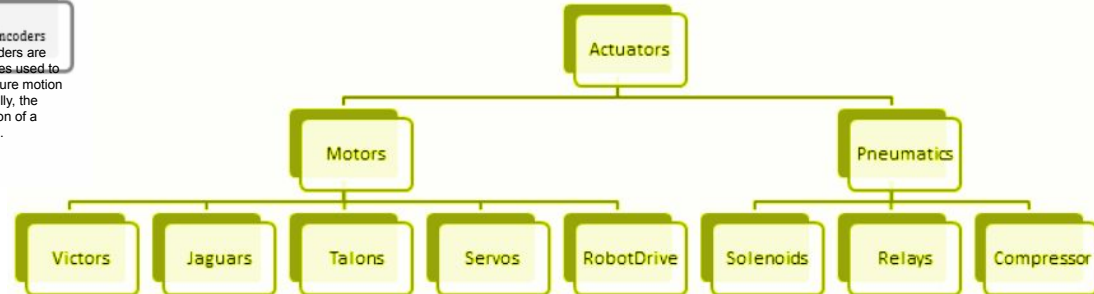
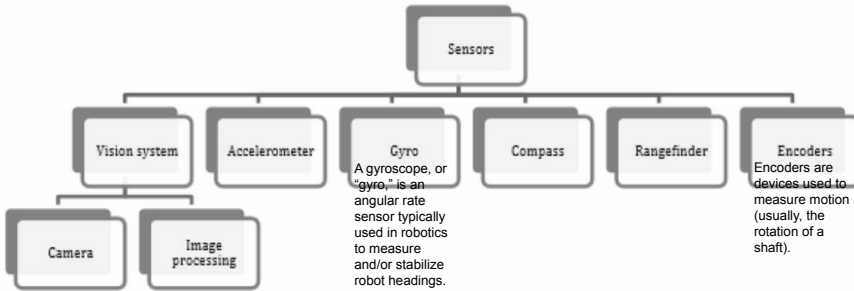
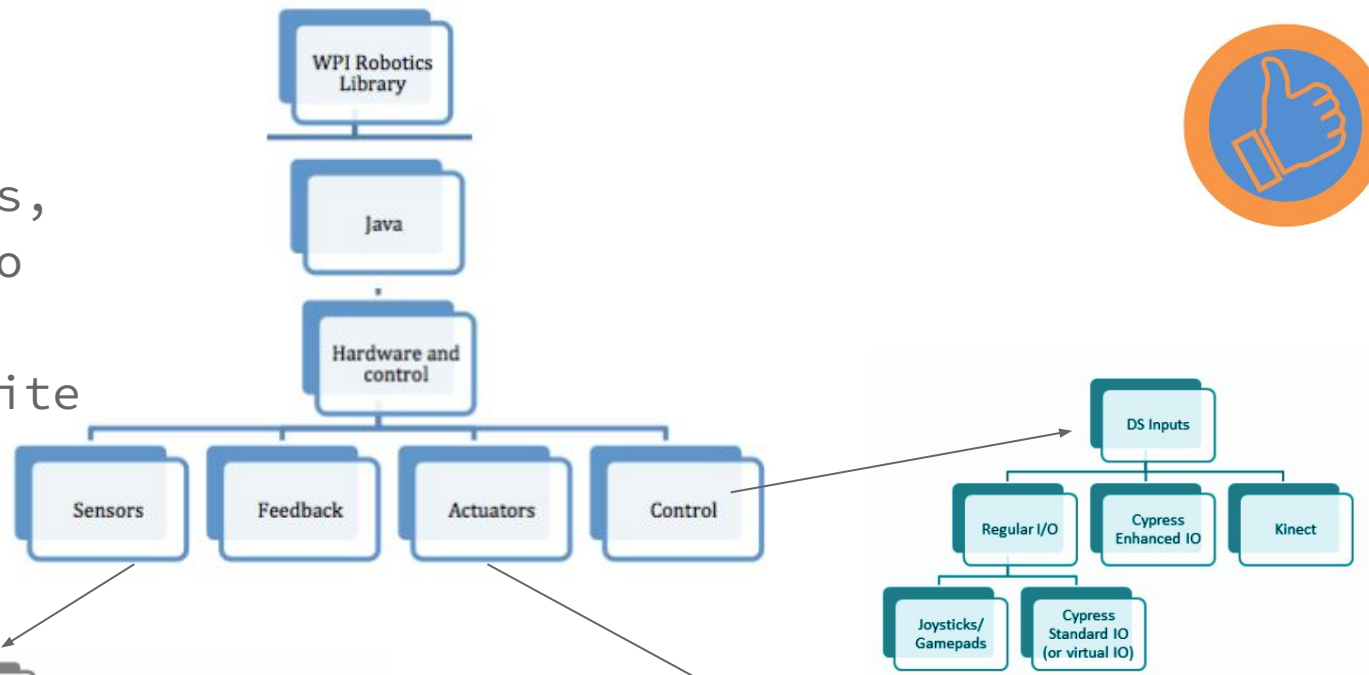
TECH STACK

WPI Library (abstracts hardware)
Java
MS Visual Studio IDE

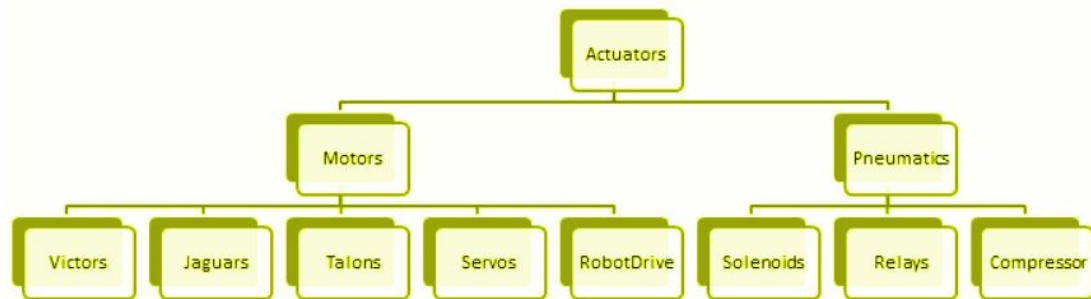


WPI LIBRARY

Gives you Sensors,
Motors classes to
work with so you
don't have to write
device drivers



MOTORS



Motor Controllers

Jaguar
VEX Robotics
15 KHz



Interface: PWM
CAN
(CAN = Controller Area Network)
WPILib Class: Jaguar

Talon
Cross the Road Electronics
15 KHz



Interface: PWM

WPILib Class: Talon

Victor 888/884
VEX Robotics
1 KHz



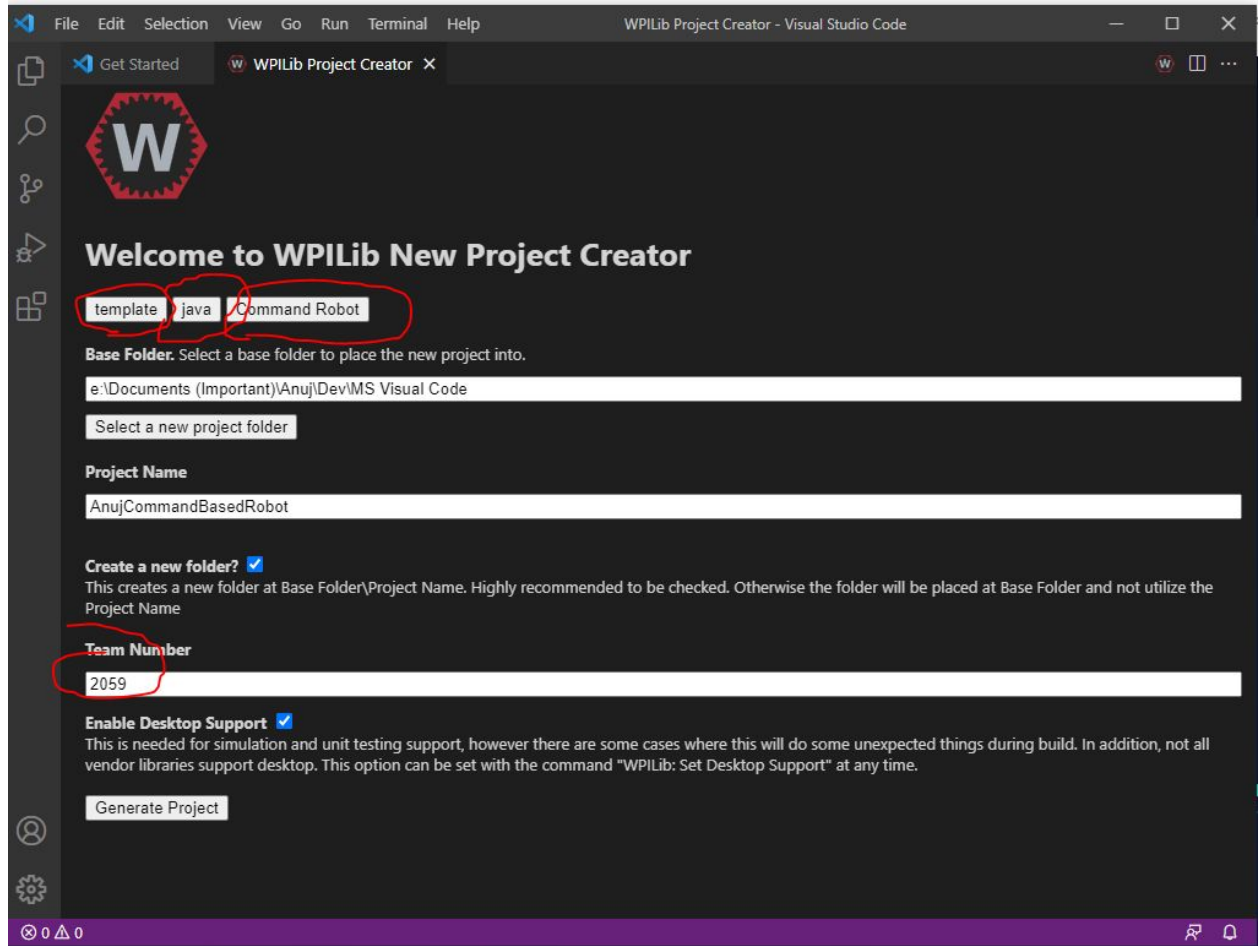
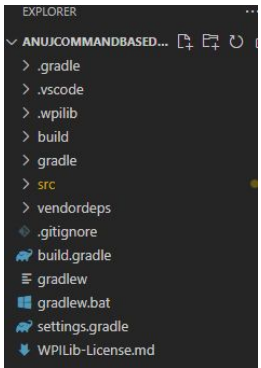
Interface: PWM

WPILib Class: Victor



INSTALLATION AND COMMAND BASED ROBOT (SORT OF HELLOWORLD.JAVA)

1. Use MS Visual Code IDE but distributed by WPI; [instructions](#)
 - a. Eclipse does not seem to be an IDE of choice [WPIlib in Eclipse](#)
 - b. If there are issues installing turn off the antivirus temporarily
2. Listen to YouTube video to learn [WPLIB in VS Code](#) and [how to use MS Visual Code](#)
3. Fire up MS Visual Code and create a sample project with “WPI Template” to create a “Command Based Robot” in “Java” with team 2059
4. This will create a gradle (build) file along with a lot of template code



COMMAND BASED ROBOT : AUTO GENERATED CODE



So far we have auto-generated code (6 classes) based on the template we had selected.

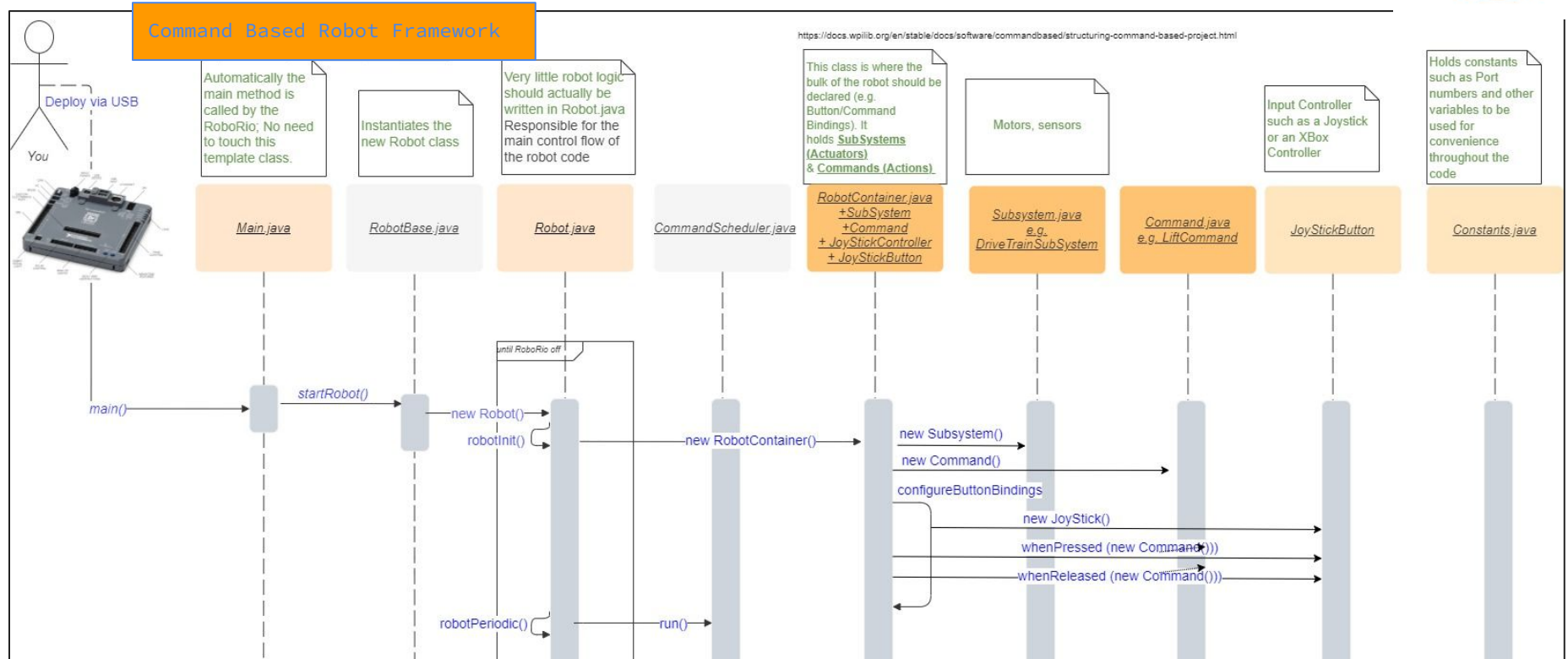
1. Robot.java
2. RobotContainer.java
3. Constants.java
4. Main.java
5. ExampleCommand
6. ExampleSubsystem

```
EXPLORER
ANUJCOMMANDBASED...
  > .gradle
  > .vscode
  > .wpilib
  > build
  > gradle
  > src\main
    > deploy
    > java\frc\robot
      > commands
      > ExampleCommand.java
      > subsystems
        > ExampleSubsystem.java
        > Constants.java
        > Main.java
        > Robot.java
        > RobotContainer.java
  > vendordeps
  > .gitignore
  > build.gradle
  > gradlew
  > gradlew.bat
  > settings.gradle
  > WPILib-License.md

src > main > java > frc > robot > subsystems > ExampleSubsystem.java > ExampleSubsystem > periodic()
1 // Copyright (c) FIRST and other WPILib contributors.
2 // Open Source Software; you can modify and/or share it under the terms of
3 // the WPILib BSD license file in the root directory of this project.
4
5 package frc.robot.subsystems;
6
7 import edu.wpi.first.wpilibj2.command.SubsystemBase;
8
9 public class ExampleSubsystem extends SubsystemBase {
10     /** Creates a new ExampleSubsystem. */
11     public ExampleSubsystem() {}
12
13     @Override
14     public void periodic() {
15         // This method will be called once per scheduler run
16     }
17
18     @Override
19     public void simulationPeriodic() {
20         // This method will be called once per scheduler run during simulation
21     }
22 }
23
```


HOW DOES IT WORK AT A HIGH LEVEL?

Who calls what and when? WIP...work in progress..



Use DRAW.IO for Sequence Diagrams and Learn about Sequence Diagrams at <https://createely.com/blog/diagrams/sequence-diagram-tutorial/>

Structuring a Command-Based Robot Project <https://docs.wpilib.org/en/stable/docs/software/commandbased/structuring-command-based-project.html>

MAIN.JAVA



- Main entry point or the class
- **Its main method is called by RoboRio at start automatically; You never call it directly.**
- The main method **instantiates your Robot.java** by calling its constructor

```
public final class Main {  
    private Main() {}  
  
    /**  
     * Main initialization function. Do not perform any initialization here.  
     *  
     * <p>If you change your main robot class, change the parameter type.  
     */  
    Run | Debug  
    public static void main(String... args) {  
        RobotBase.startRobot(Robot::new);  
    }  
}
```


ROBOT.JAVA: UNDERSTANDING GENERATED CODE



Primary Class that gets instantiated.

Two types of functions

- Init()
- Periodic()

```
> .gradle
> .vscode
> .wpilib
> build
> gradle
> src\main
> deploy
  > java\src\robot
    > commands
    > subsystems
    ● Constants.java
    ● Main.java
    ● Robot.java
    ● RobotContainer.java
  > OUTLINE
    {} frc.robot
    > Robot
      ● autonomousInit() :void
      ● autonomousPeriodic() :void
      ● disabledInit() :void
      ● disabledPeriodic() :void
      ● m_autonomousCommand
      ● m_robotContainer
      ● robotInit() :void
      ● robotPeriodic() :void
      ● teleopInit() :void
      ● teleopPeriodic() :void
      ● testInit() :void
      ● testPeriodic() :void

7  import edu.wpi.first.wpilibj.TimedRobot;
8  import edu.wpi.first.wpilibj2.command.Command;
9  import edu.wpi.first.wpilibj2.command.CommandScheduler;
10
11
12  /**
13   * The VM is configured to automatically run this class, and to call the functions corresponding to
14   * each mode, as described in the TimedRobot documentation. If you change the name of this class or
15   * the package after creating this project, you must also update the build.gradle file in the
16   * project.
17   */
18  public class Robot extends TimedRobot {
19      private Command m_autonomousCommand;
20
21      private RobotContainer m_robotContainer;
22
23      /**
24       * This function is run when the robot is first started up and should be used for any
25       * initialization code.
26       */
27      @Override
28      public void robotInit() {
29          // Instantiate our RobotContainer. This will perform all our button bindings, and put our
30          // autonomous chooser on the dashboard.
31          m_robotContainer = new RobotContainer();
32      }
33
34      /**
35       * This function is called every robot packet, no matter the mode. Use this for items like
36       * diagnostics that you want ran during disabled, autonomous, teleoperated and test.
37       *
38       * <p>This runs after the mode specific periodic functions, but before LiveWindow and
39       * SmartDashboard integrated updating.
40       */
41      @Override
42      public void robotPeriodic() {
43          // Runs the Scheduler. This is responsible for polling buttons, adding newly-scheduled
44          // commands, running already-scheduled commands, removing finished or interrupted commands,
45          // and running subsystem periodic() methods. This must be called from the robot's periodic
46          // block in order for anything in the Command-based framework to work.
47          CommandScheduler.getInstance().run();
48      }
49  }
```



ROBOT.JAVA: TWO TYPES OF FUNCTIONS

INIT()

Initializing called once when the motor is started

```
public class Robot extends TimedRobot {  
    /**  
     * This function is run when the robot is first  
     * for any initialization code.  
     */  
    @Override  
    public void robotInit() {  
    }  
  
    @Override  
    public void autonomousInit() {  
    }  
  
    @Override  
    public void autonomousPeriodic() {  
    }  
  
    @Override  
    public void teleopInit() {  
    }  
  
    @Override  
    public void teleopPeriodic() {  
    }  
  
    @Override  
    public void testInit() {  
    }  
  
    @Override  
    public void testPeriodic() {  
    }  
}
```

PERIODIC()

are called repeatedly every 0.02 seconds TO updated the commands running on the robot

```
public class Robot extends TimedRobot {  
    /**  
     * This function is run when the robot is first  
     * for any initialization code.  
     */  
    @Override  
    public void robotInit() {  
    }  
  
    @Override  
    public void autonomousInit() {  
    }  
  
    @Override  
    public void autonomousPeriodic() {  
    }  
  
    @Override  
    public void teleopInit() {  
    }  
  
    @Override  
    public void teleopPeriodic() {  
    }  
  
    @Override  
    public void testInit() {  
    }  
  
    @Override  
    public void testPeriodic() {  
    }  
}
```

ROBOT.JAVA: DEPENDING ON THE MODE DIFFERENT METHODS ARE CALLED



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

```
@Override  
public void autonomousInit() {  
}  
  
@Override  
public void autonomousPeriodic() {  
}
```

```
@Override  
public void teleopInit() {  
}  
  
@Override  
public void teleopPeriodic() {  
}
```

```
@Override  
public void testInit() {  
}  
  
@Override  
public void testPeriodic() {  
}
```

Autonomous Mode

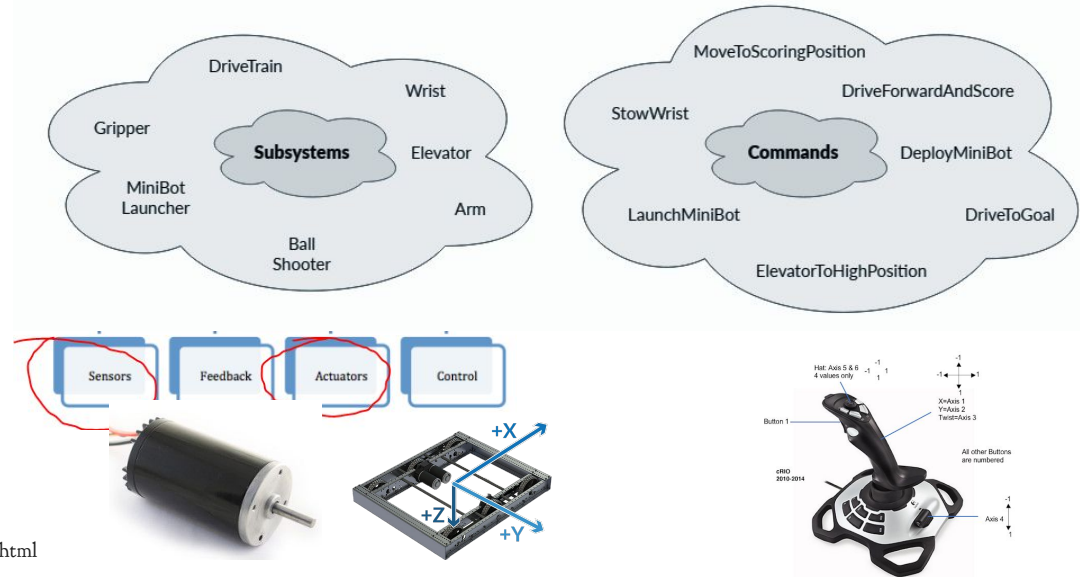
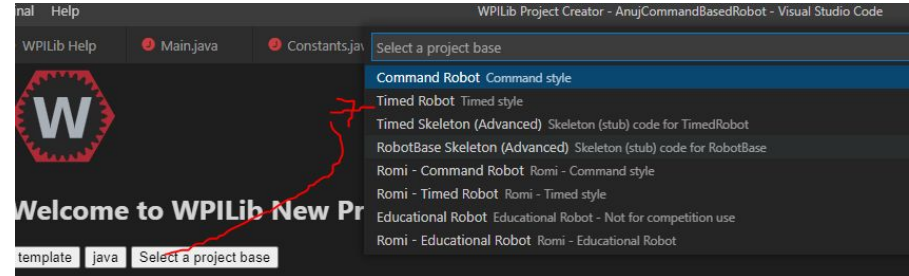
Tele-operated Mode

Test Mode

WHAT IS A "COMMAND-BASED" ROBOT VS. TIMED ROBOT OR ITERATIVE



- Input (Joystick) and Output (Motor)
- Essentially a design pattern (boilerplate framework/code) that reduces the amount of code one has to write to get a robot to do something.
- This kind of robot is composed of two parts, viz., *Command* and *Subsystem*
- You **bind** Commands in your code to an Input Controller (e.g. Joystick button)
- You focus on "command" (actions) and "subsystem" (sensors or a motors) objects.
 - Subsystems represent the actual lower level hardware such as different types of Actuators (e.g. motors, pneumatic) and Sensors in your code. A subsystem can be a group of motors or sensors or just one motor or a sensor
 - Commands represent actions or behavior one wants to send to a subsystem. An action is either starting (initializing), executing, ending, or idle.



SUBSYSTEM (THINK HARDWARE SENSOR OR A MOTOR OR A COMPOSITION OF THESE)

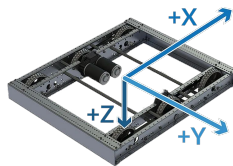
- You write **methods** to control hardware or read sensor values. E.g. `grabHatch()`
- **`periodic()`** is run once per run of the scheduler
- Works through the **CommandScheduler**
- Can set **default background commands that are run when nothing else is scheduled**. E.g. keeping an arm held at a setpoint.

```
exampleSubsystem.setDefaultCommand(exampleCommand);
```

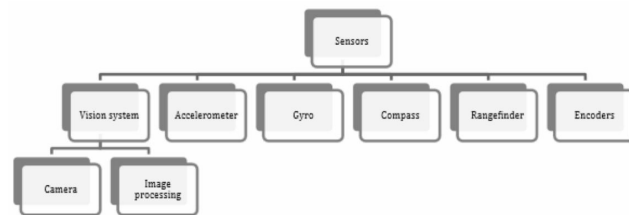
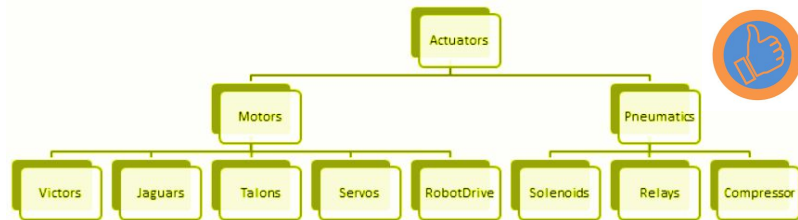
- Is **initialized** via the Robot class methods (that are suffixed with **`init()`**)

```
public class ExampleSubsystem extends SubsystemBase {  
    /** Creates a new ExampleSubsystem. */  
    public ExampleSubsystem() {}  
  
    @Override  
    public void periodic() {  
        // This method will be called once per scheduler run  
    }  
  
    @Override  
    public void simulationPeriodic() {  
        // This method will be called once per scheduler run during simulation  
    }  
}
```

Subsystems <https://docs.wpilib.org/en/stable/docs/software/commandbased/subsystems.html>



E.g. A Drivetrain subsystem will consist of a left Motor and a right Motor, along with the method `setSpeed(double s)`



```
/** A hatch mechanism actuated by a single {@Link D  
public class HatchSubsystem extends SubsystemBase {  
    private final DoubleSolenoid m_hatchSolenoid =  
        new DoubleSolenoid(  
            PneumaticsModuleType.CTREPCM,  
            HatchConstants.kHatchSolenoidPorts[0],  
            HatchConstants.kHatchSolenoidPorts[1]);  
  
    /** Grabs the hatch. */  
    public void grabHatch() {  
        m_hatchSolenoid.set(kForward);  
    }  
  
    /** Releases the hatch. */  
    public void releaseHatch() {  
        m_hatchSolenoid.set(kReverse);  
    }  
}
```


COMMAND (THINK ACTION ON A MOTOR OR SENSOR OR ON A GROUP/SUBSYSTEM OF THESE)



```
/** An example command that uses an example subsystem. */
public class ExampleCommand extends CommandBase {
    @SuppressWarnings({"PMD.UnusedPrivateField", "PMD.SingularField"})
    //private final ExampleSubsystem m_subsystem;

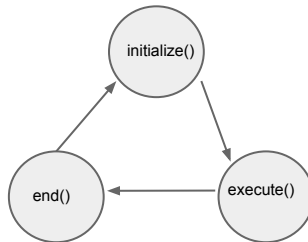
    /**
     * Creates a new ExampleCommand.
     *
     * @param subsystem The subsystem used by this command.
     */
    public ExampleCommand(ExampleSubsystem subsystem) {
        // m_subsystem = subsystem;
        // Use addRequirements() here to declare subsystem dependencies.
        addRequirements(subsystem);
    }

    // Called when the command is initially scheduled.
    @Override
    public void initialize() {}

    // Called every time the scheduler runs while the command is scheduled.
    @Override
    public void execute() {}

    // Called once the command ends or is interrupted.
    @Override
    public void end(boolean interrupted) {}

    // Returns true when the command should end.
    @Override
    public boolean isFinished() {
        return false;
    }
}
```



- A 3 state machine
- **Knows about a SubSystem (takes it in constructor)**
- You fill in the templated methods
- Calls Subsystem methods
- The CommandScheduler will not schedule more than one Command for a SubSystem at a time.



```
public class DefaultDrive extends CommandBase {
    private final DriveSubsystem m_drive;
    private final DoubleSupplier m_forward;
    private final DoubleSupplier m_rotation;

    /**
     * Creates a new DefaultDrive.
     *
     * @param subsystem The drive subsystem this command will run on.
     * @param forward The control input for driving forwards/backwards
     * @param rotation The control input for turning
     */
    public DefaultDrive(DriveSubsystem subsystem, DoubleSupplier forward, DoubleSupplier rotation) {
        m_drive = subsystem;
        m_forward = forward;
        m_rotation = rotation;
        addRequirements(m_drive);
    }

    @Override
    public void execute() {
        m_drive.arcadeDrive(m_forward.getAsDouble(), m_rotation.getAsDouble());
    }
}
```

COMMANDGROUPS

(THINK MULTIPLE COMMANDS GROUPED TOGETHER FOR EASE OF USE)



1. Working with 3 State commands can get cumbersome
2. A grouping of multiple commands to reduce complexity in programming
3. Takes in one or more Subsystems in constructor
4. Four types of templated groupings
 - SequentialCommandGroup
 - ParallelCommandGroup
 - ParallelRaceGroup
 - ParallelDeadlineGroup

```
/** A complex auto command that drives forward, releases a hatch, and then drives backward. */
public class ComplexAuto extends SequentialCommandGroup {
    /**
     * Creates a new ComplexAuto.
     *
     * @param drive The drive subsystem this command will run on
     * @param hatch The hatch subsystem this command will run on
     */
    public ComplexAuto(DriveSubsystem drive, HatchSubsystem hatch) {
        addCommands(
            // Drive forward the specified distance
            new DriveDistance(
                AutoConstants.kAutoDriveDistanceInches, AutoConstants.kAutoDriveSpeed, drive),

            // Release the hatch
            new ReleaseHatch(hatch),

            // Drive backward the specified distance
            new DriveDistance(
                AutoConstants.kAutoBackupDistanceInches, -AutoConstants.kAutoDriveSpeed, drive));
    }
}
```


JOYSTICK BUTTON/INPUT TRIGGER: BINDING TO A COMMAND



- How do you run a command if not autonomous mode?
- E.g. a button press by a human.
- Solution: bind the command to this triggering event
- Instantiate the actual hardware class that initiates input/trigger (e.g. XBOXController)
- Bindings only need to be declared once, ideally some time during robot initialization. The library handles everything else.



In RobotContainer.java

```
// Creates a joystick on port 1
Joystick exampleStick = new Joystick(1);

// Creates a new JoystickButton object for button 1 on exampleStick
JoystickButton exampleButton = new JoystickButton(exampleStick, 1);

// Binds an ExampleCommand to be scheduled when the trigger of the example joystick is pressed
exampleButton.whenPressed(new ExampleCommand());

// Binds a BarCommand to be scheduled when that same button is released
exampleButton.whenReleased(new BarCommand());
```

COMMANDSCHEDULER

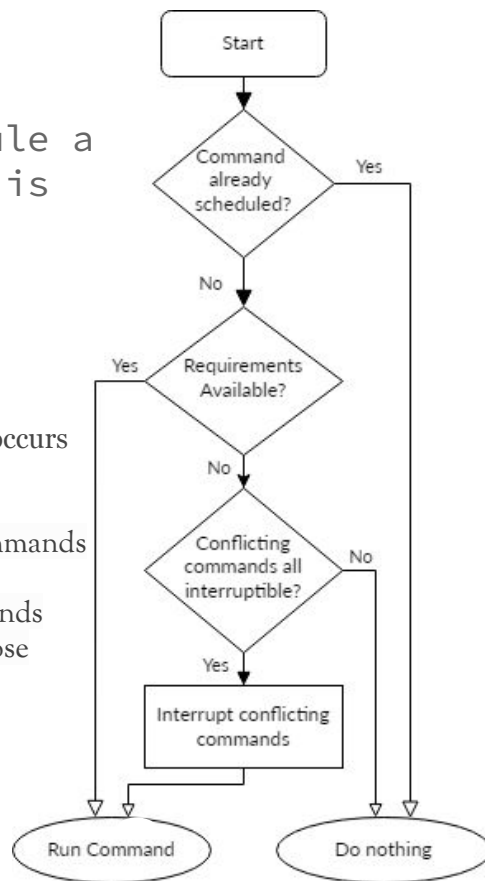
- You call its **schedule()** to schedule a command; its initialize() method is called after getting added
- Runs the actual commands automatically through the **run()** method

One call to run() is one iteration.

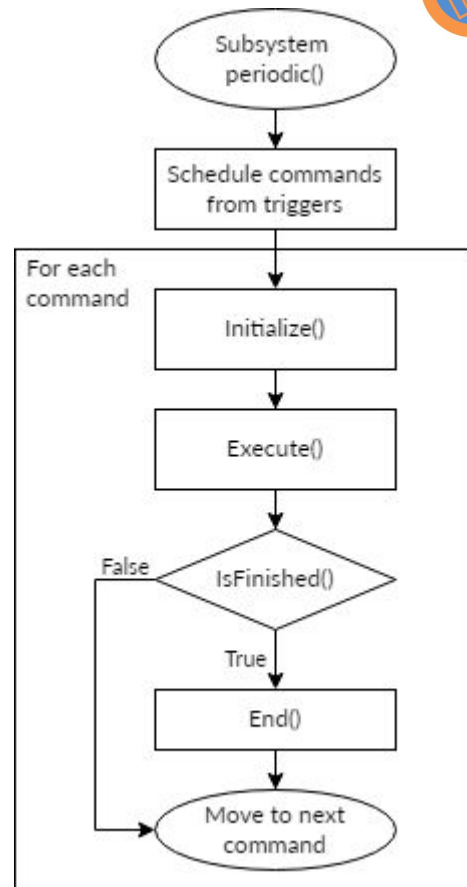
Each iteration (ordinarily once per 20ms), the scheduler execution occurs in the following order:

- call Subsystem periodic() methods
 - poll the state of all registered triggers/buttons, get new commands
 - schedule new commands for execution
 - run the command bodies of all currently scheduled commands
 - check end conditions on scheduled commands and end those
- commands that have finished or are interrupted
 - CommandScheduler.getInstance()
 - You never call its methods except to start
(CommandScheduler.getInstance()).run()
it from your Robot's
robotPeriodic()

The **schedule()** Method



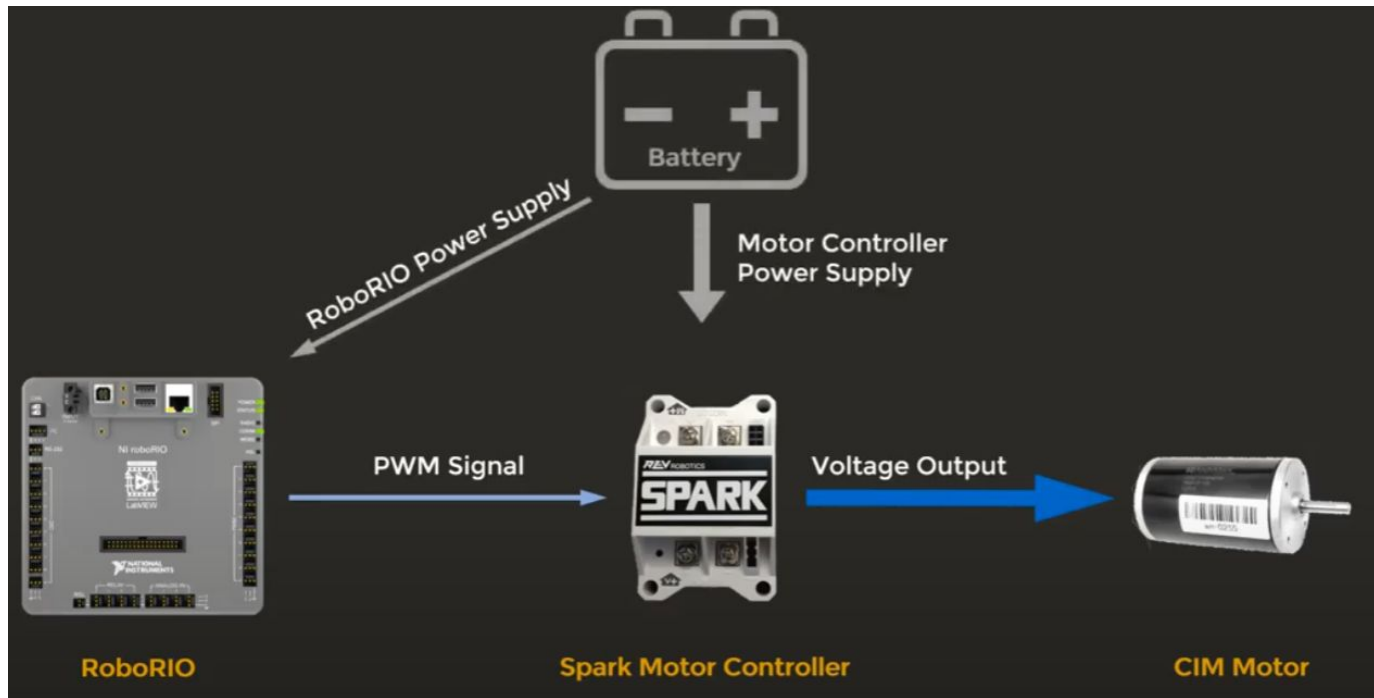
The **run()** Method



TO CONTROL THE ROBOT MOTOR YOU NEED A MOTOR CONTROLLER

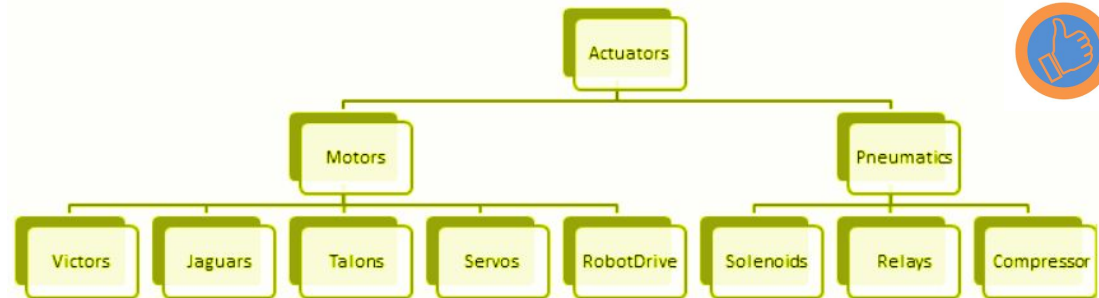


1. From Java code (burned in RoboRio) you can **send a PWM signal** to the Motor Controller telling it **how fast to spin a motor**
2. The motor controller will take the signal into a voltage output to the motor



WHAT IS A MOTOR

- A type of an Actuator class in WPILib



Motor Controllers



Jaguar
VEX Robotics
15 KHz

Interface: PWM
CAN
(CAN = Controller Area Network)
WPILib Class: Jaguar



Talon
Cross the Road Electronics
15 KHz

Interface: PWM

WPILib Class: Talon



Victor 888/884
VEX Robotics
1 KHz

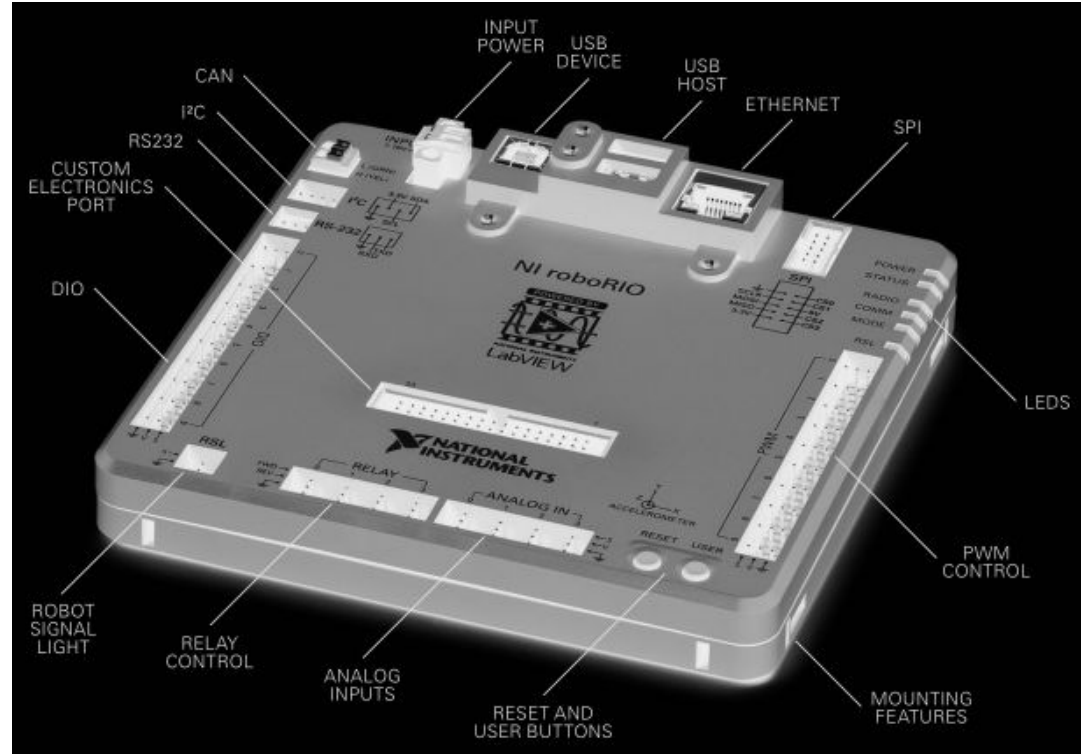
Interface: PWM

WPILib Class: Victor

WHAT IS ROBORIO

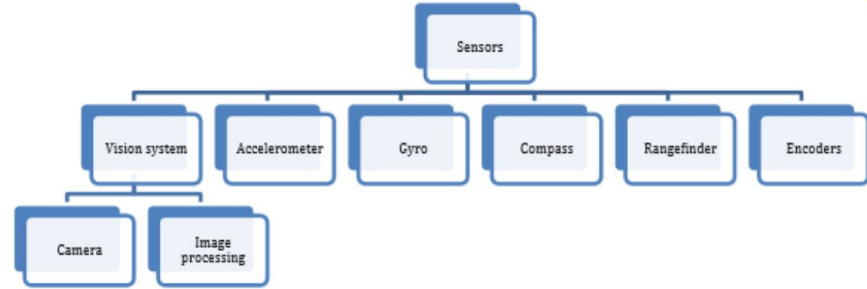


- The brain of the robot where the code will be deployed and run
- Headless, no GUI, need only a Jar
- has many ports that are used in coding and for connecting to different devices
 - USB - to deploy code
 - CAN - used to control not only the pneumatics, but we use it for motor controllers that support CAN. CAN is the easiest way to wire things vs. using PWM, although at times we may use PWM too.
 - PWM - for Motor Controllers
 - DIO - to connect to sensors
 - MXP - for board expansion
 - SPI - for a gyro meter



WHAT IS A SENSOR

- Many types such as
 - Camera
 - Encoder
- To help robot understand its location and physical space around it



JAVA CODING CONVENTIONS



JAVA CODING CONVENTIONS



1. Use JavaDoc Documentation comments (`/** * */`) to help you think about what the method does and help others understand what it is supposed to do without reading the code. A method encapsulates a behavior that one should be able to describe without needing to read the code.
2. Use Block comments (`/* * */`) and single line comments (`/* */`), or end of line comments (`/**`) to describe a variable or other parts of code
3. Do not use underscores in class or variable names
4. Once done coding, beautify your code
5. A variable name should be descriptive and
 - a. start with a lowercase letter following camel case. E.g. `myDriveTrainSubsystem`
6. A class name should be descriptive and
 - a. start with a capital letter following camel case. `MyDriveTrainSubsystem`
 - b. should include the name of the command or subsystem it is extending to help with self-documentation (E.g. `TeleopCommand`)

JAVA CODING CONVENTIONS...



A good practice would be to specify in Constants.java ports as constants. E.g.

```
Constants {  
  
public static final int LEFT_MOTOR_PORT = 1;  
  
public static final int RIGHT_MOTOR_PORT = 1;  
  
public static final int JOYSTICK_PORT = 1;  
  
public static final double PI = 3.14159;  
  
}
```

Then you can access the variable in your code as Constants.LEFT_MOTOR_PORT

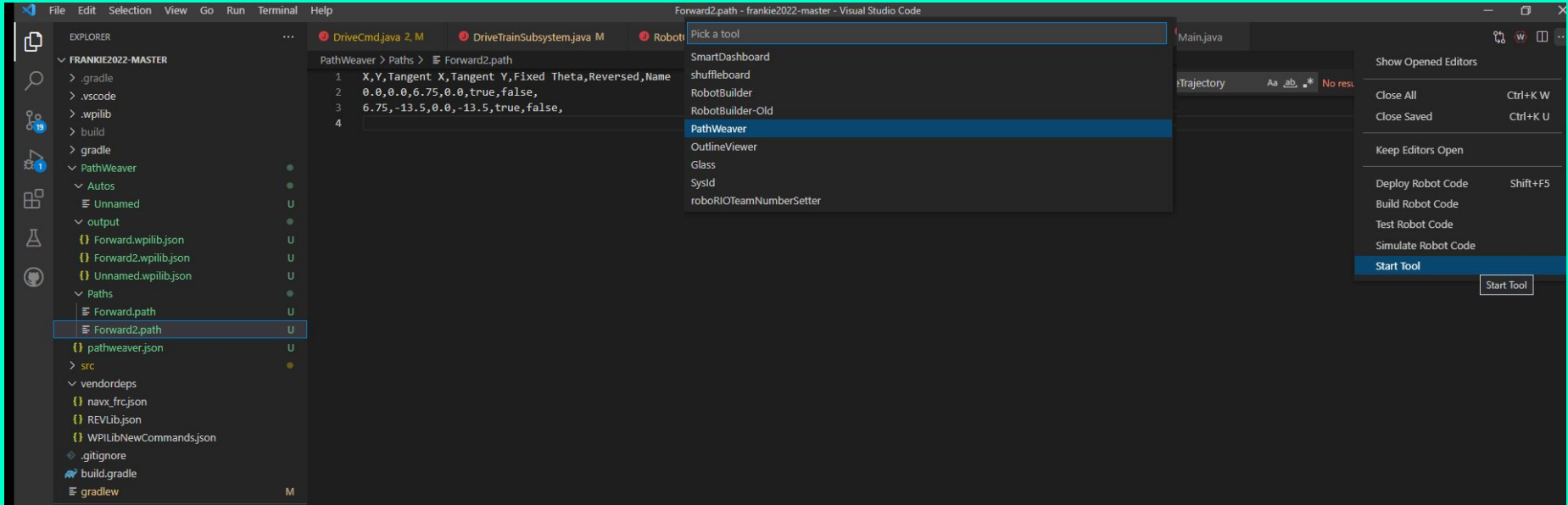
Note the upper case and underscores to define constants as static finals ; this is an exception to the regular nomenclature in java for naming variables/class

Also note that making it public will allow you to access the variable without getters/setters; again an exception to accessing the variables via methods as these are constants.

Two good programming resource Java for WPILib can be the following;; Keep in mind it might be a little outdated (2018)

1. <https://frc6506.github.io/docs/Documents/Tome%20of%20Secrets.pdf>
2. <https://buildmedia.readthedocs.org/media/pdf/frc-pdr/latest/frc-pdr.pdf>

VISUAL TOOLS FROM MICROSOFT VISUAL CODE

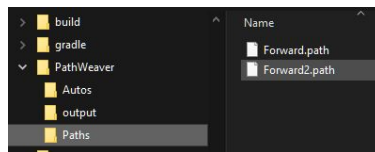
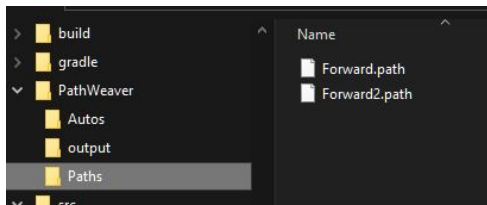
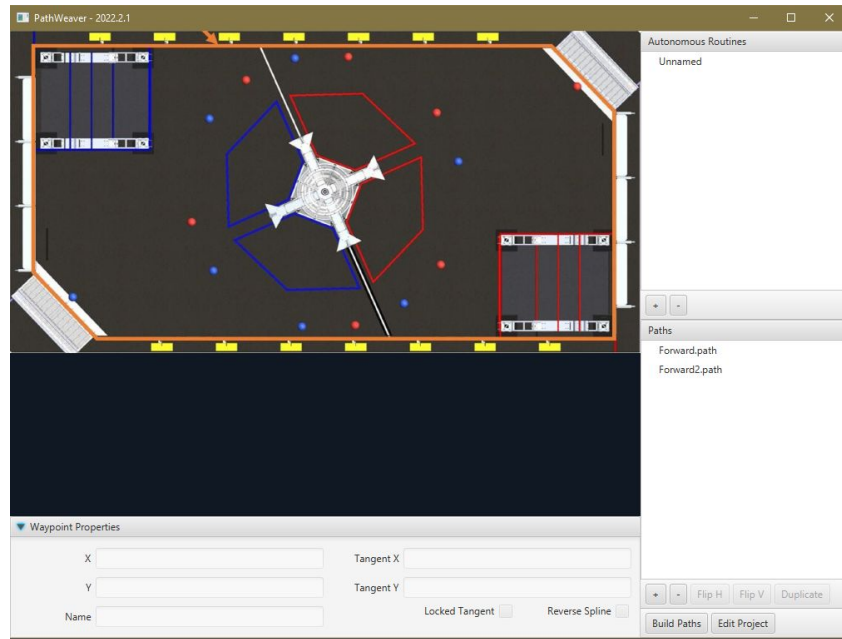
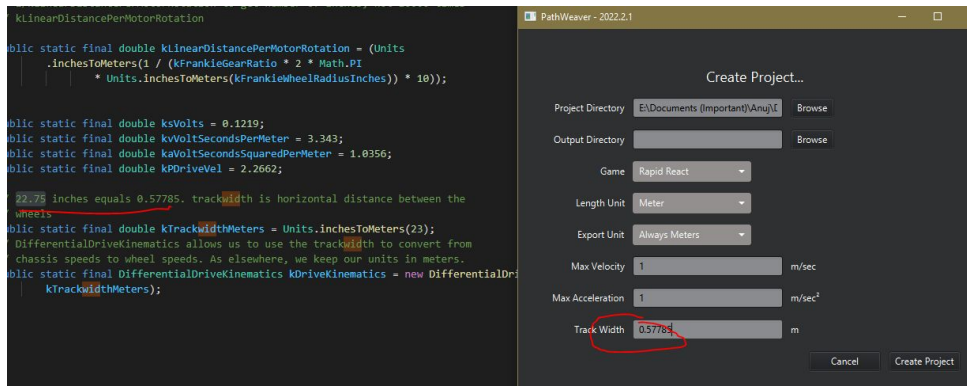


PathWeaver to visualize trajectories
RobotBuilder to build a robot and export code
RobotSimulator to simulate robot code

PATHWEAVER TOOL AND DECLARATIVE TRAJECTORY



- PathWeaver tool/gui places `wpilib.json` files in `src/main/deploy/paths` which will automatically be placed on the roboRIO file system in `/home/lvuser/deploy/paths` and can be accessed using `getDeployDirectory`
- <https://docs.wpilib.org/en/stable/docs/software/pathplanning/pathweaver/creating-pathweaver-project.html>.



ROBOT SIMULATION TOOL



- Used for Autonomous PID tuning

