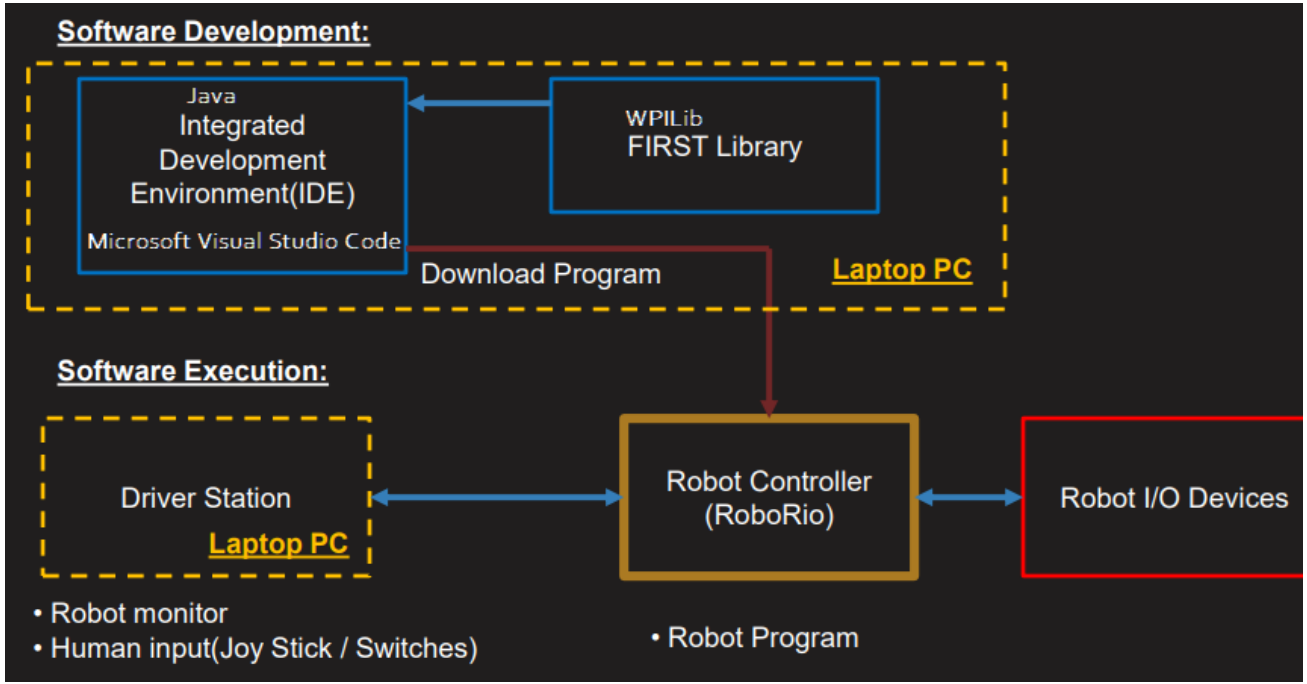
The Hitchhikers | FRC Team 2059

FRC Programming in WPILib

FRC Team 2059
By Mj
Co-Captain and Programming Lead

Tech Stack

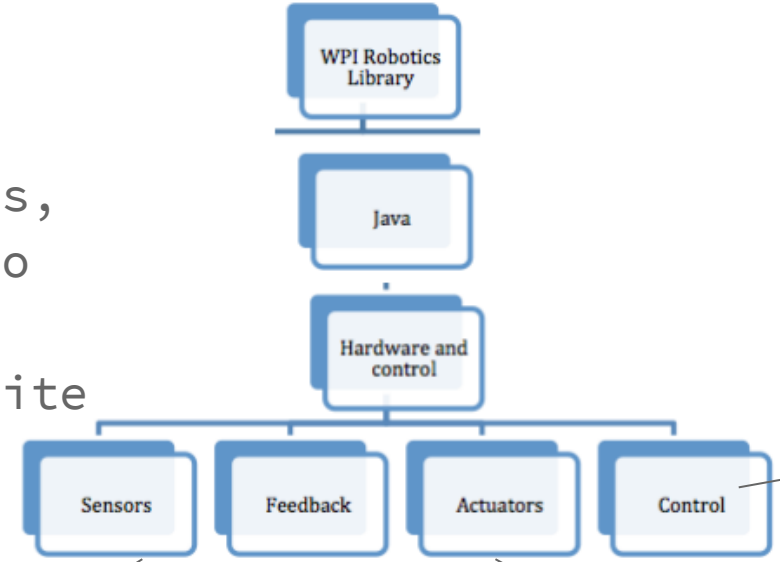
WPI Library (abstracts hardware)
Java
MS Visual Studio IDE

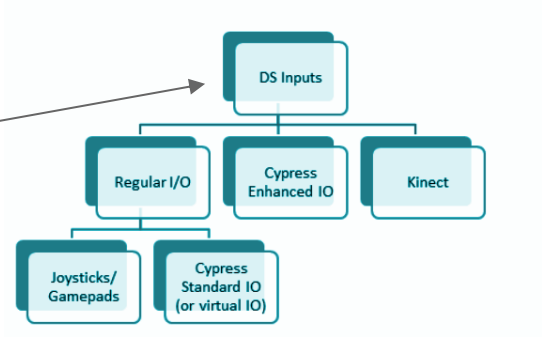


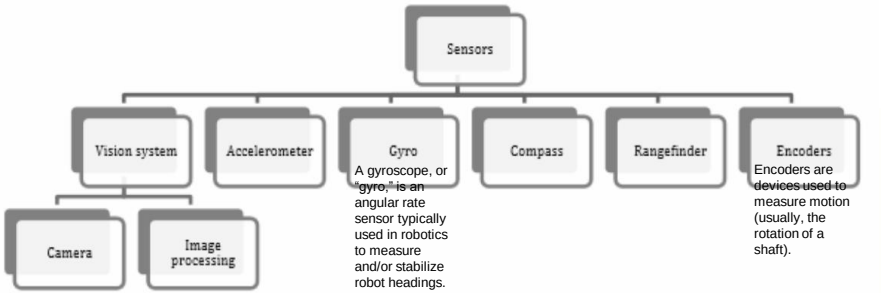
http://cougartech.org/resources/Training/3-Engineering/CTSoft1-FRC%20Soft%20Design%20Process%20and%20Prog%20Structure.pdf

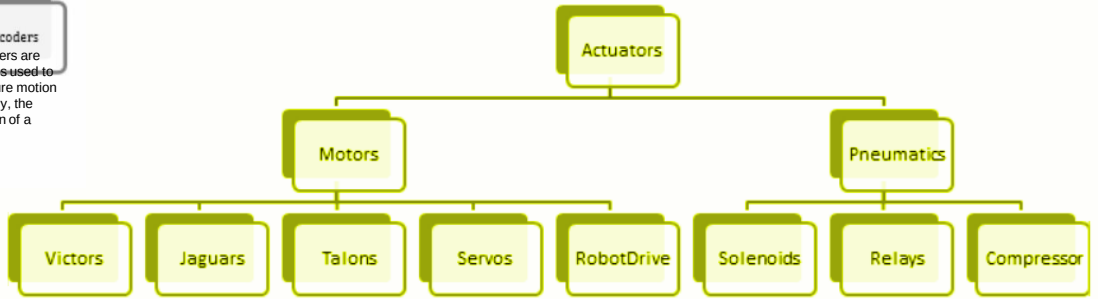
WPI Library

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



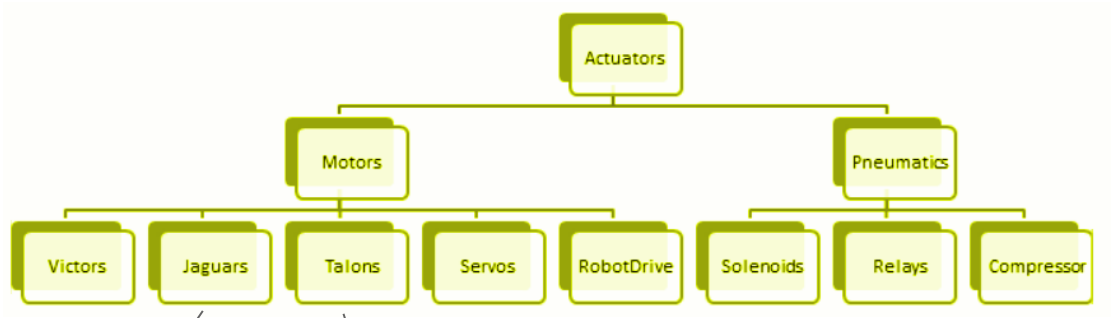






https://stemrobotics.cs.pdx.edu/sites/default/files/WPILib_programming.pdf

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

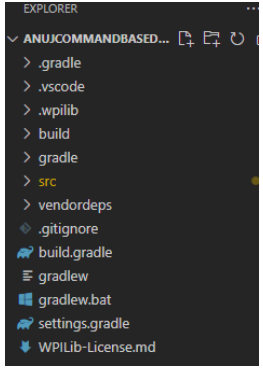
Victor 888/884
VEX Robotics
1 Khz

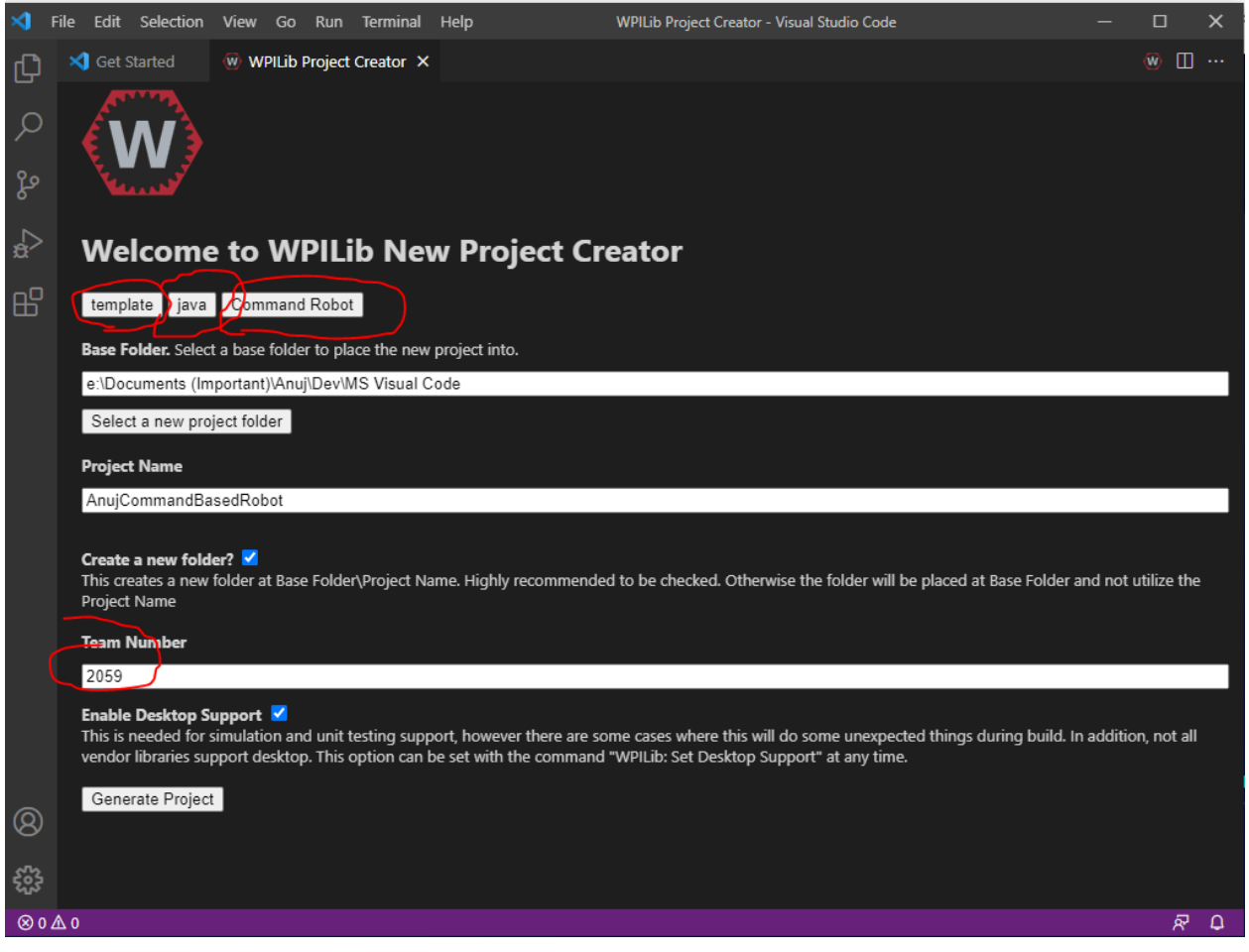
Interface: PWM

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



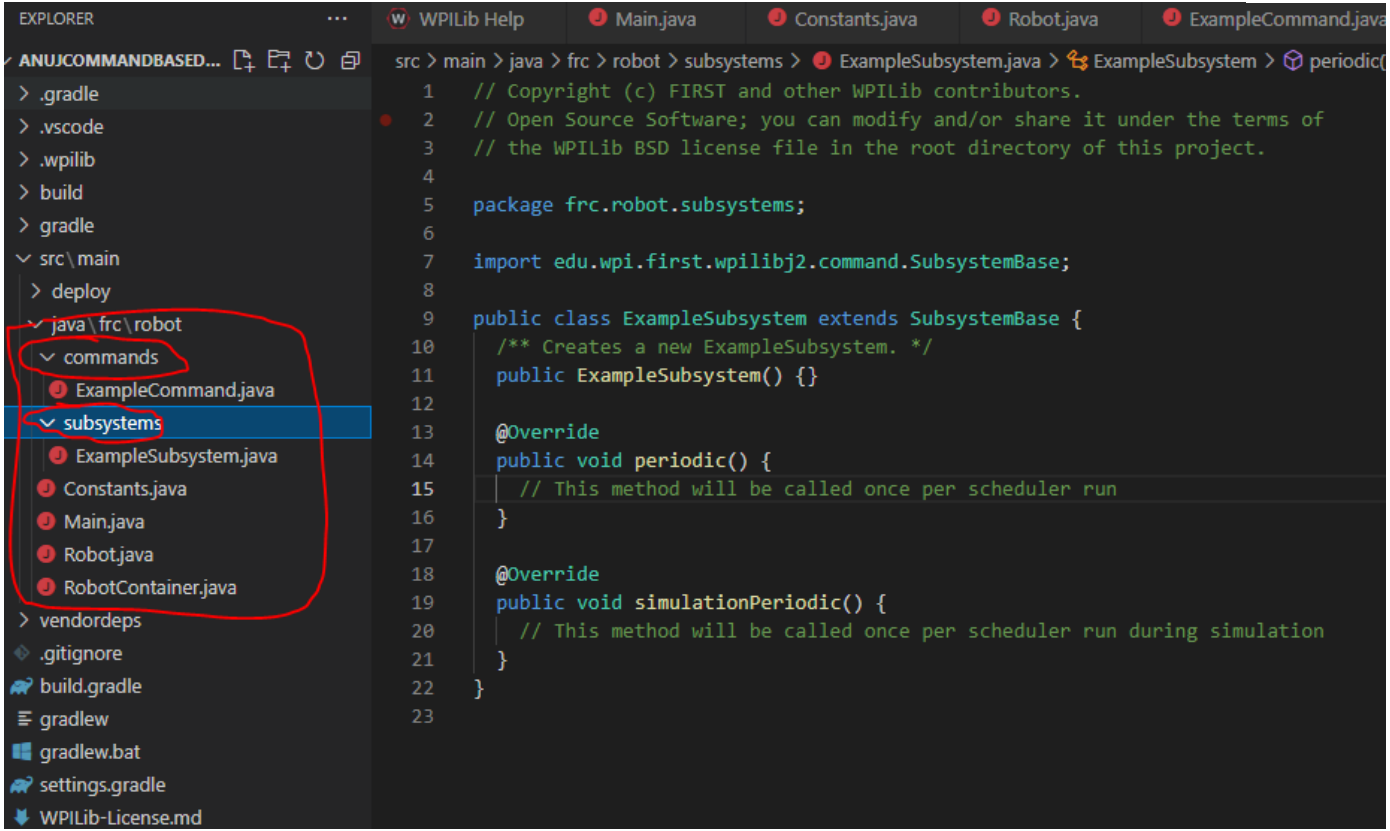


Command Based Robot : Auto Generated Code

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

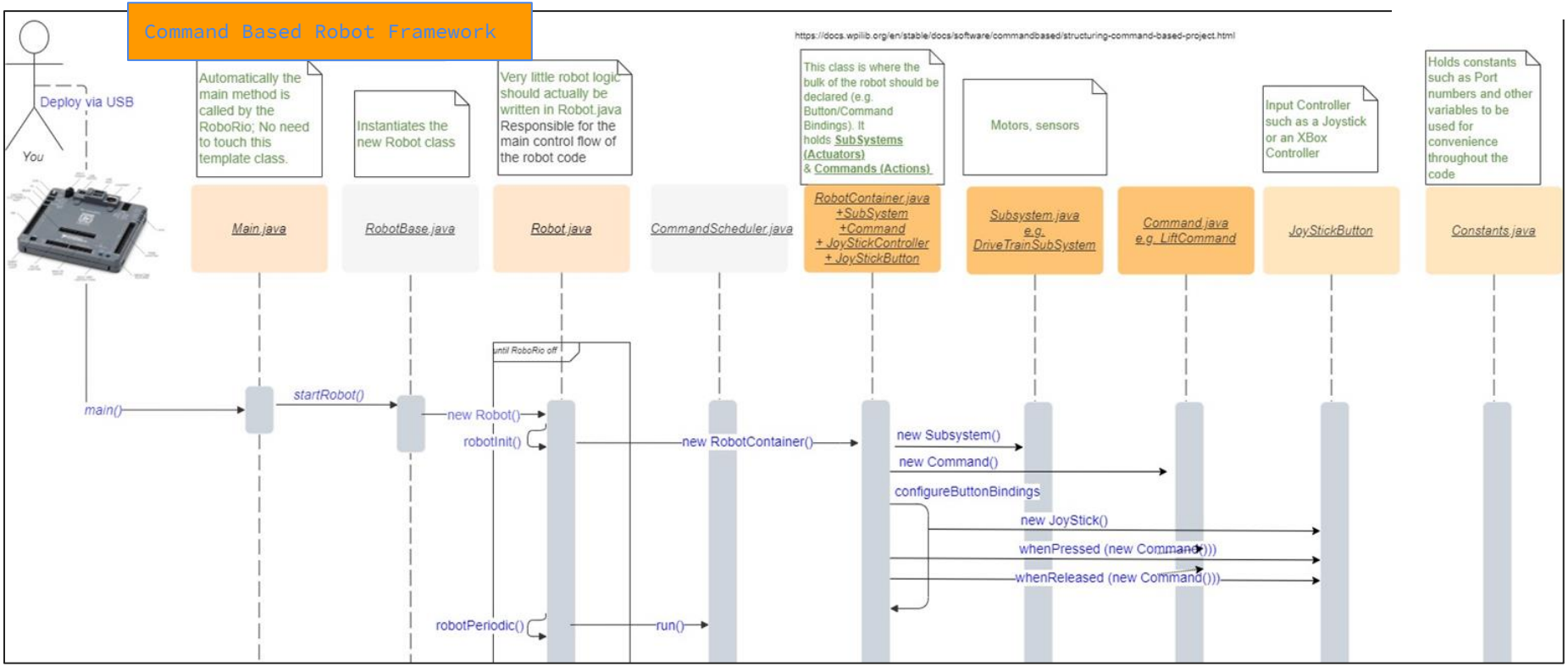
1. Robot.java
2. RobotContainer.java
3. Constants.java
4. Main.java

1. ExampleCommand
2. ExampleSubsystem



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://creately.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

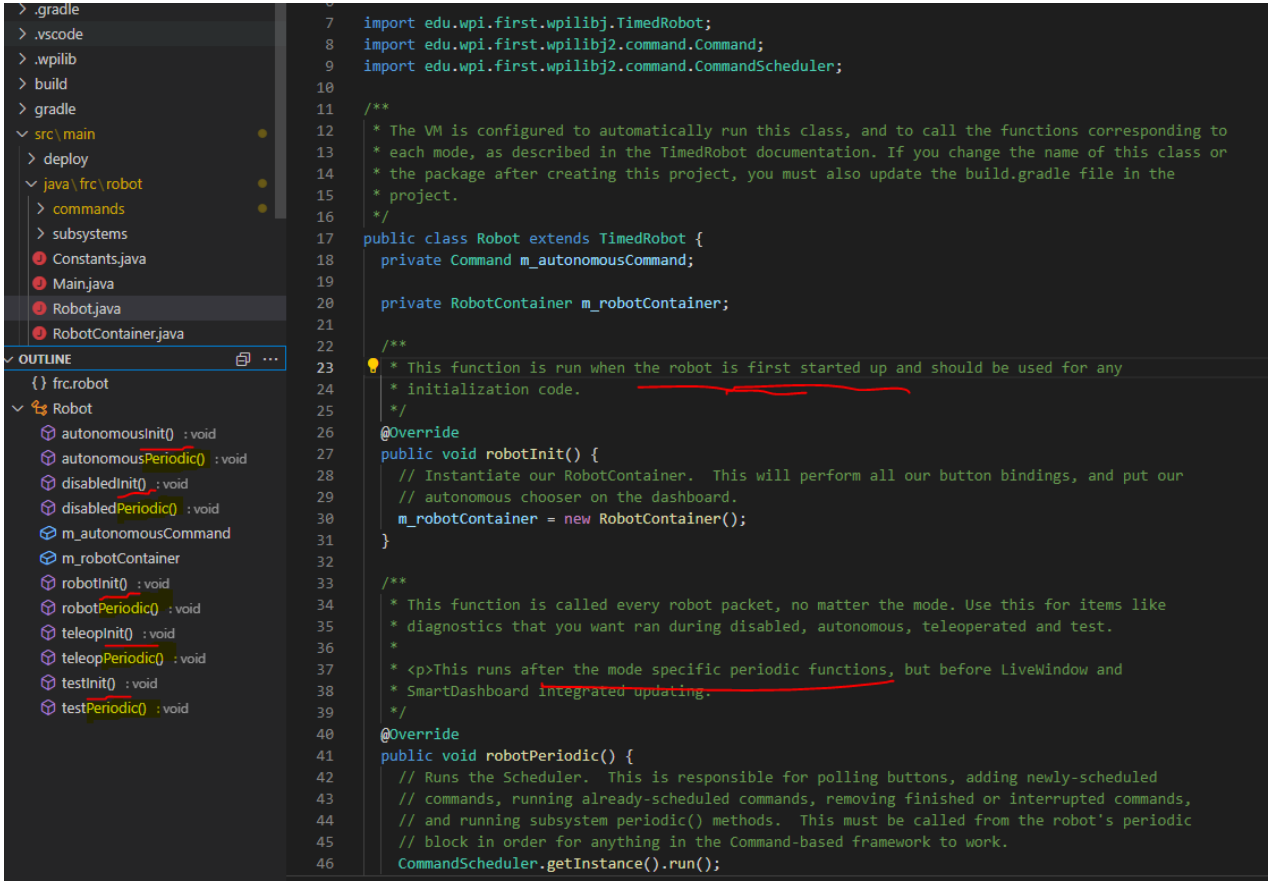
```java
public final class Main {
  private Main() {}

  /**
   * 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()



Robot.java: two types of functions

init()

Initializing called once when the motor is started

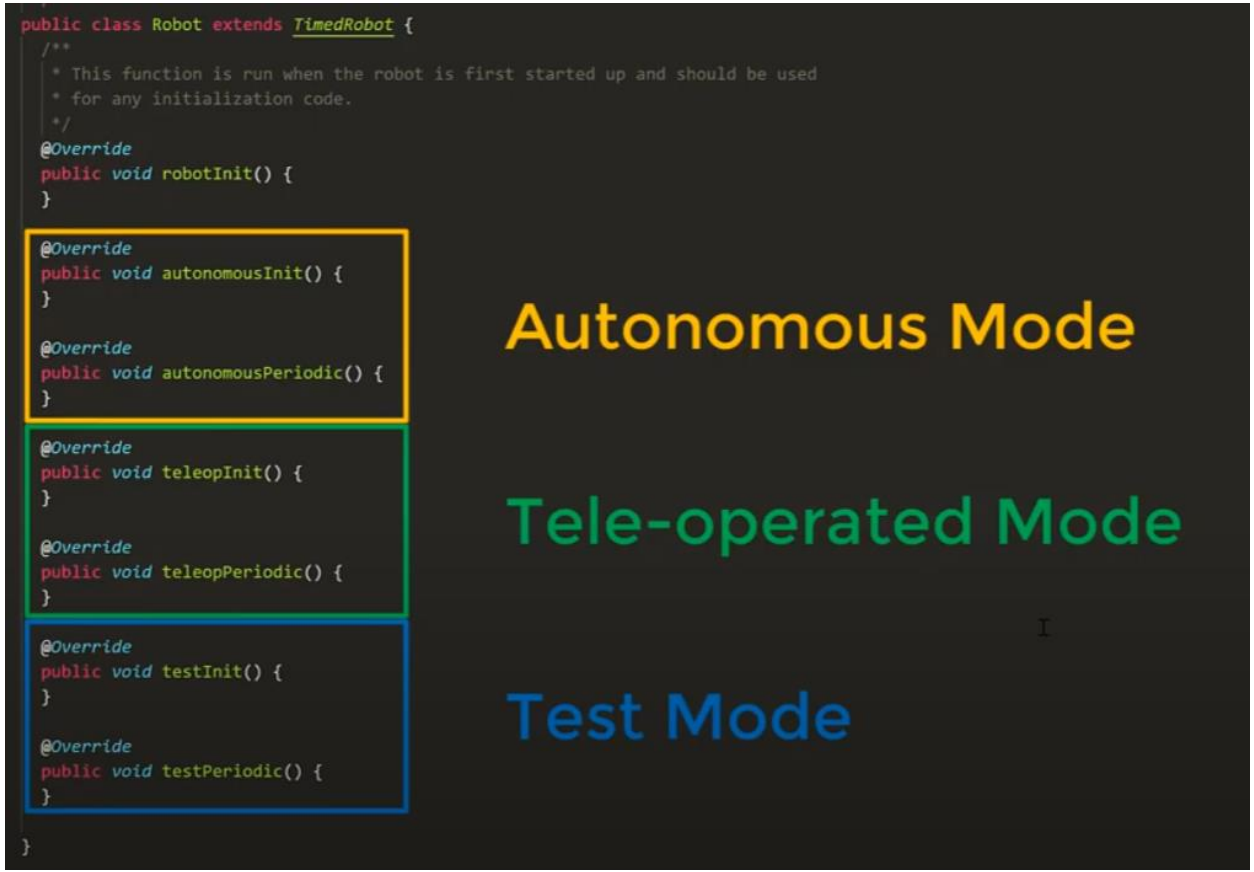
```
public class Robot extends TimedRobot {
  /**
   * This function is run when the robot is fi
   * for any initialization code.
   */
  @Override
  public void robotInit() {
  }

  @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

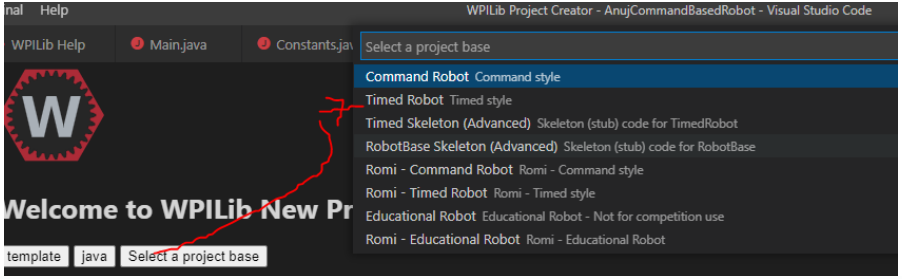
```
*/
public class Robot extends TimedRobot
  /**
   * This function is run when the rob
   * for any initialization code.
   */
  @Override
  public void robotInit() {
  }

  @Override
  public void autonomousInit() {
  }

  @Override
  public void autonomousPeriodic() {
  }

  @Override
  public void teleopInit() {
  }

  @Override
  public void teleopPeriodic() {
  }

  @Override
  public void testInit() {
  }

  @Override
  public void testPeriodic() {
  }

}
```

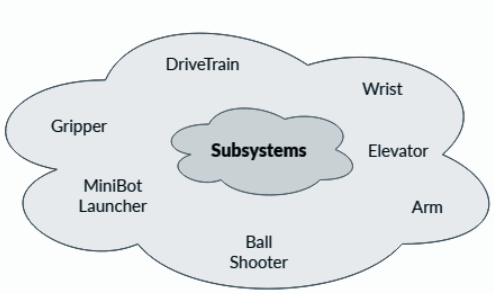
Robot.java: Depending on the Mode different methods are called

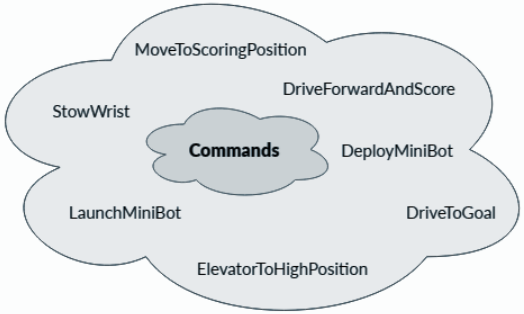


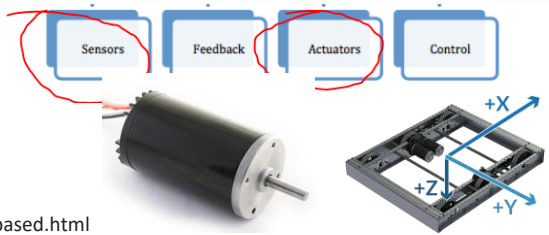
What Is a "Command-Based" Robot vs. Timed Robot or Iterative

- 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.









https://docs.wpilib.org/en/stable/docs/software/commandbased/what-is-command-based.html

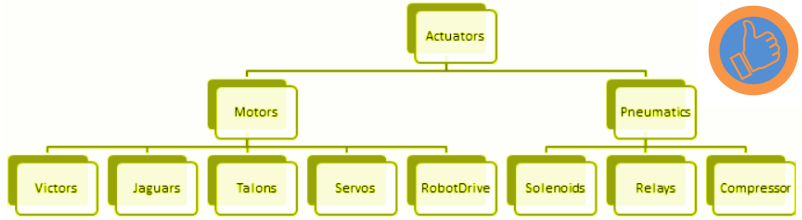
Subsystem (think hardware sensor or a motor or a composition of these)

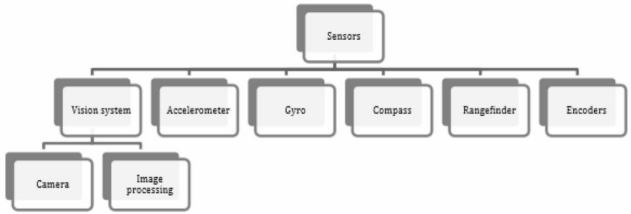
- **You write methods** to control hardware or read sensor values. E.g. grabHatch()
- **periodic()** is run once per run of the scheduler
- Works through the **CommandScheduler**
- Can set **default background commands that are run when nothing else is scheduled.** E.g. keeping an arm held at a setpoint.

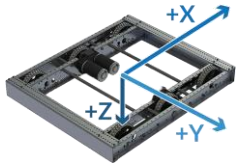
```
exampleSubsystem.setDefaultCommand(exampleCommand);
```

- 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
  }
}
```





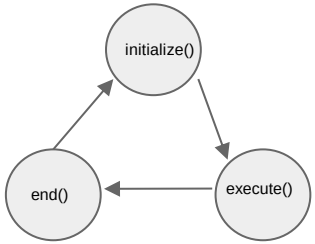


E.g. A Drivetrain subsystem will consist of a left Motor and a right Motor, along with the method setSpeed(doubles)

```
/** A hatch mechanism actuated by a single {@link Do
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);
  }
}
```

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

Command (think action on a motor or sensor or on a group/subsystem of these)



- 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.

```java
/** 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;
  }
}
```


```java
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 wil 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());
  }
}
```

Commands https://docs.wpilib.org/en/stable/docs/software/commandbased/commands.html

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));
  }
}
```

Command Groups https://docs.wpilib.org/en/stable/docs/software/commandbased/command-groups.html

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());
```

Command Bindings to Triggers https://docs.wpilib.org/en/stable/docs/software/commandbased/binding-commands-to-triggers.html

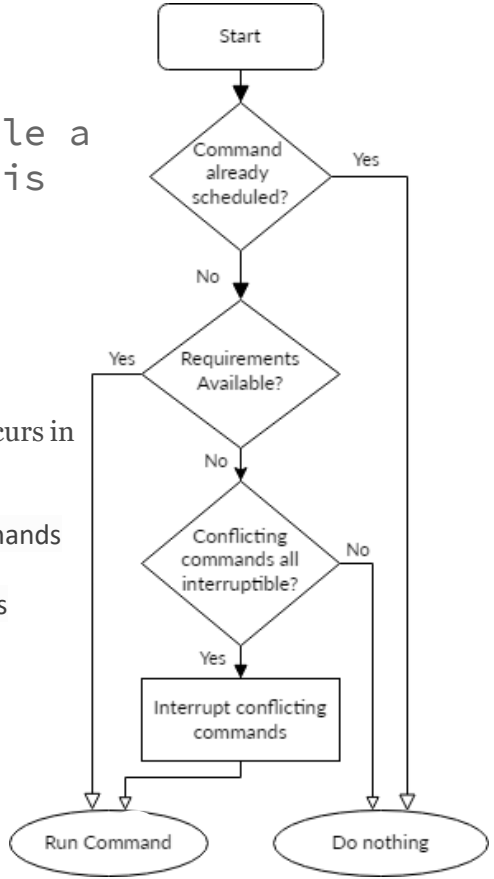
CommandScheduler

- 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

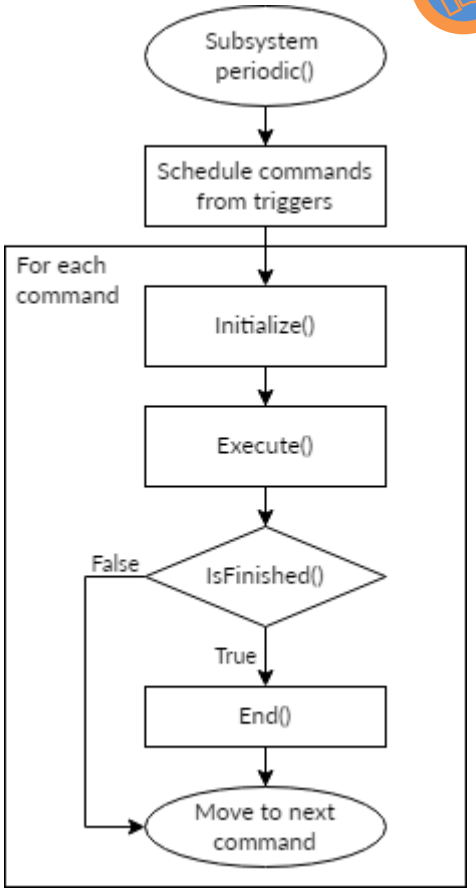
One call to run() is one iteration.
Each iteration (ordinarily once per 20ms), the scheduler execution occurs in the following order:

1. call Subsystem periodic() methods
2. poll the state of all registered triggers/buttons, get new commands
3. schedule new commands for execution
4. run the command bodies of all currently scheduled commands
5. 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



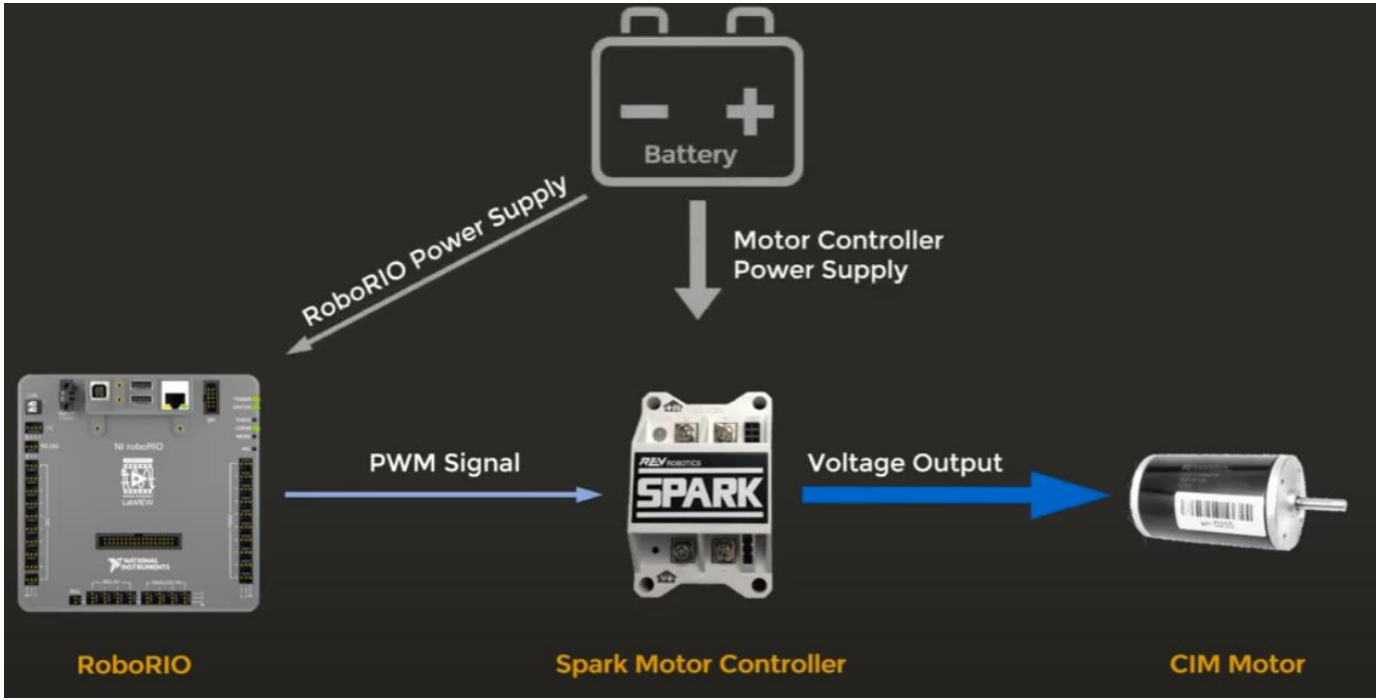
The run() Method



Command Groups https://docs.wpilib.org/en/stable/docs/software/commandbased/command-scheduler.html
https://first.wpi.edu/wpilib/allwpilib/docs/release/java/edu/wpi/first/wpilibj2/command/CommandScheduler.html#run()

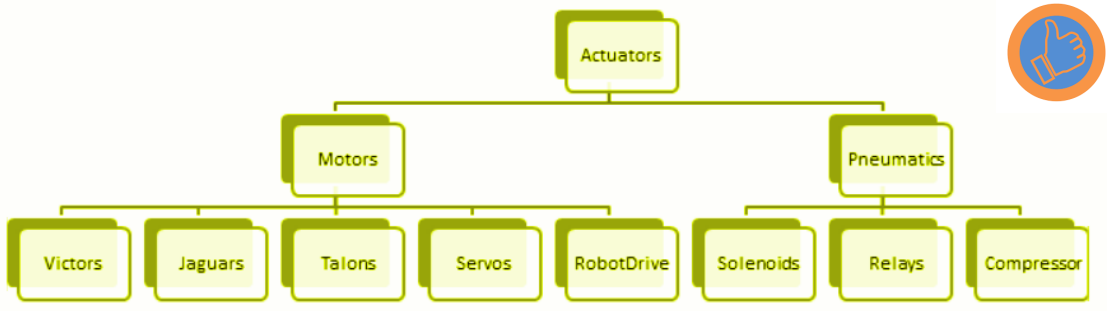
To control the Robot motor you need a Motor Controller

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

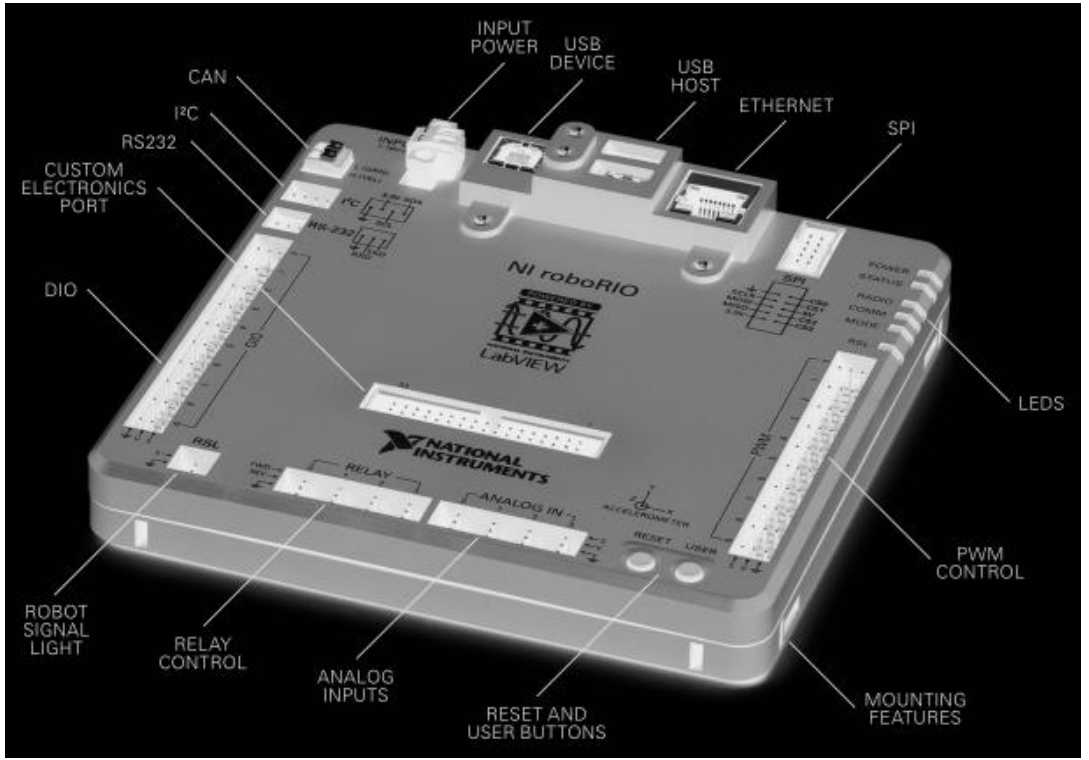
Interface: PWM
WPILib Class: Talon

Victor 888/884
VEX Robotics
1 Khz

Interface: PWM
WPILib Class: Victor

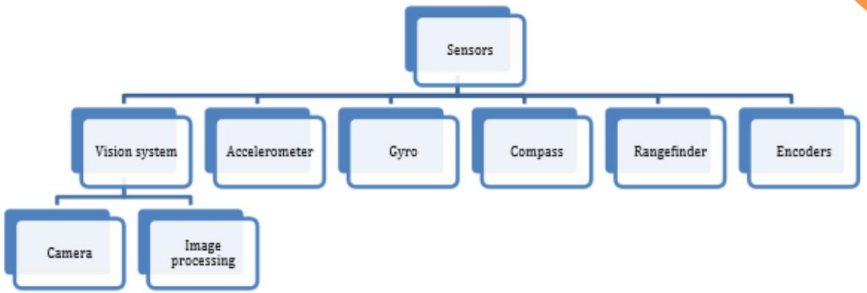
What is RoboRIO

- 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

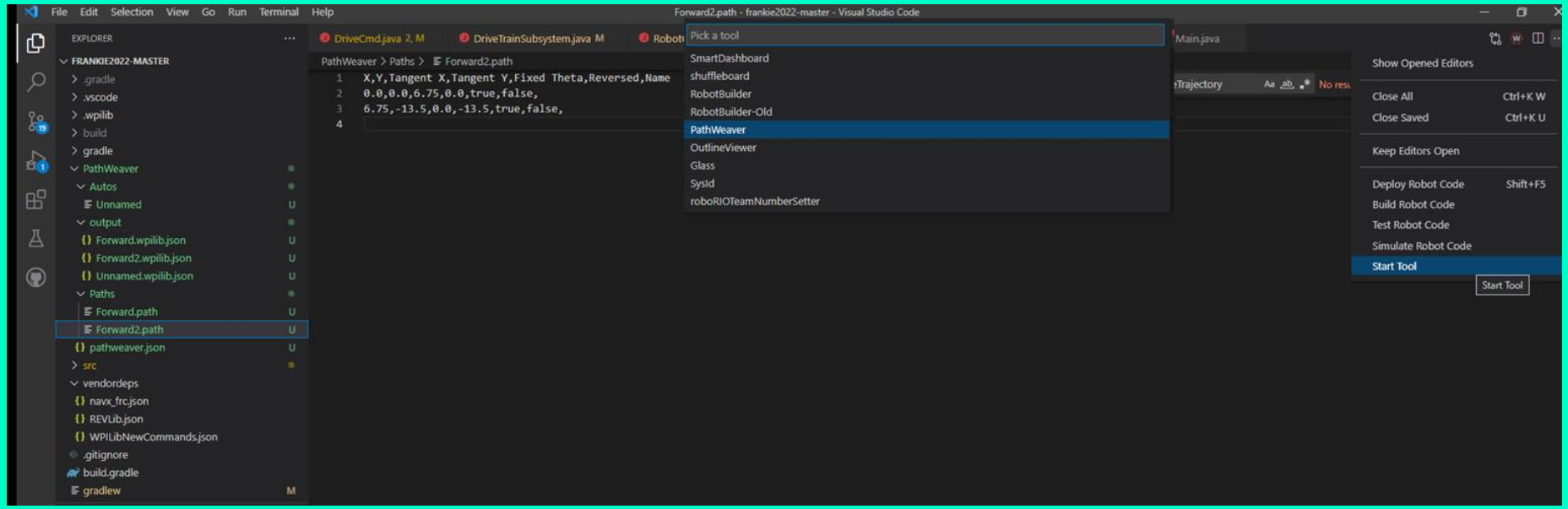
Note the upper case and underscores to define constants as static finals ; this is an exception to the regular nomenclature in java for naming variables/class

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 folowing;; Keep in mind it might be a little outdated (2018)

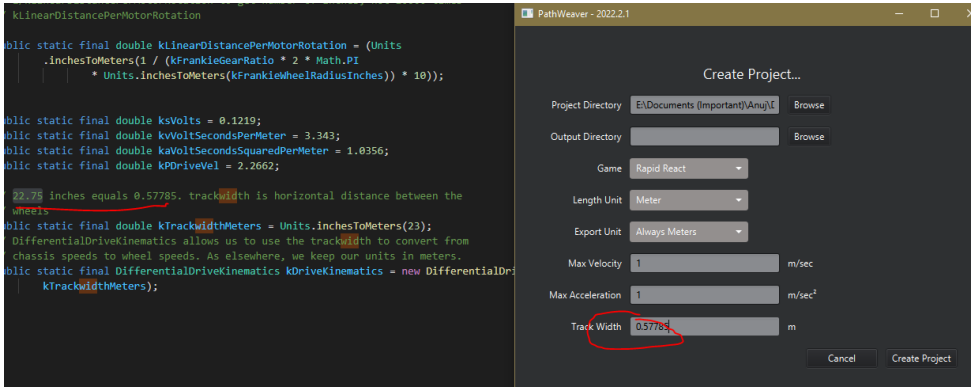
1. https://frc6506.github.io/docs/Documents/Tome%20of%20Secrets.pdf
2. https://buildmedia.readthedocs.org/media/pdf/frc-pdr/latest/frc-pdr.pdf

Visual Tools from Microsoft Visual Code

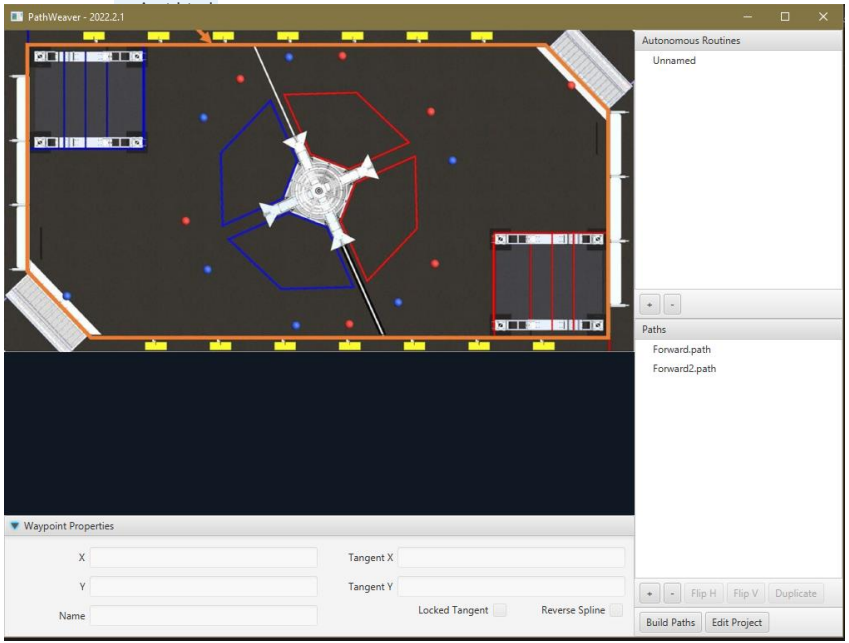


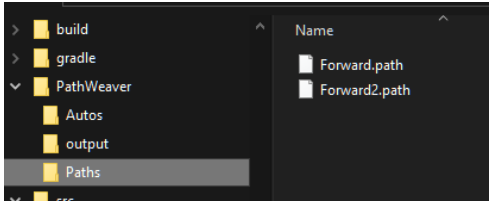
PathWeaver to visualize trajectories
RobotBuilder to build a robot and export code
RobotSimulator to simulate robot code

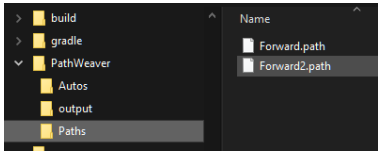
PathWeaver tool and declarative trajectory



- 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-







Robot Simulation tool

- Used for Autonomous PID tuning