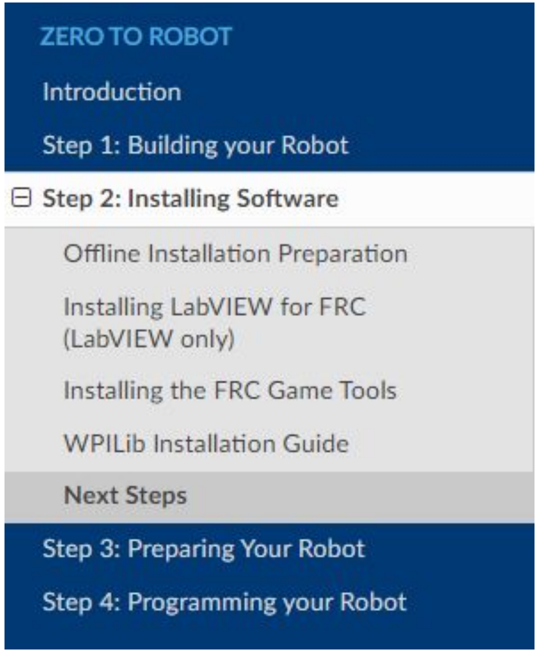
Programming for FRC

colton.bruni@gmail.com
228-297-0422

Preview and General Information

- Java
- wpilib - very important resource
    - Wiring
    - Code
    - Examples
- CTR Electronics
- Github Desktop

Wpilib docs



ZERO TO ROBOT

Introduction

Step 1: Building your Robot

⊟ Step 2: Installing Software

Offline Installation Preparation

Installing LabVIEW for FRC (LabVIEW only)

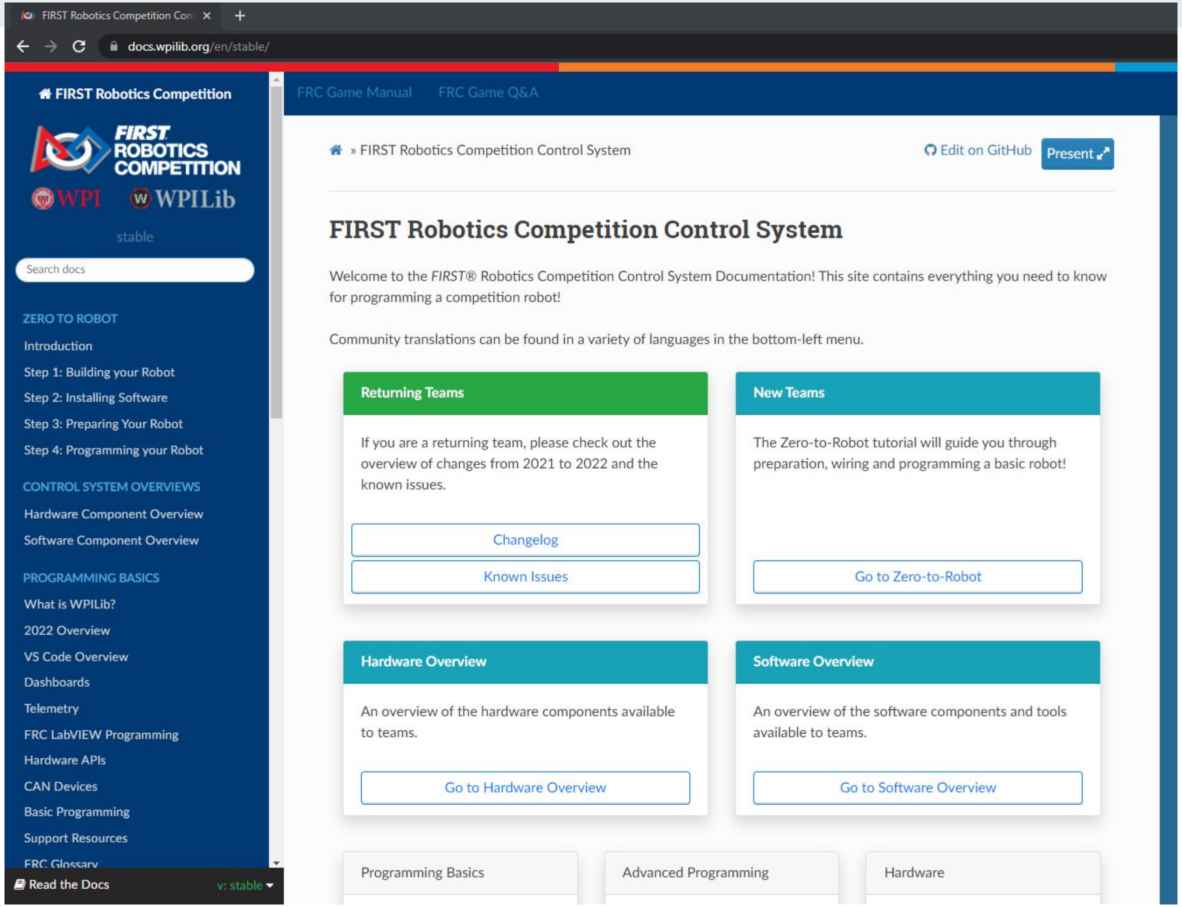
Installing the FRC Game Tools

WPILib Installation Guide

Next Steps

Step 3: Preparing Your Robot

Step 4: Programming your Robot



FIRST Robotics Competition Con

docs.wpilib.org/en/stable/

FIRST Robotics Competition

stable

Search docs

ZERO TO ROBOT

Introduction

Step 1: Building your Robot

Step 2: Installing Software

Step 3: Preparing Your Robot

Step 4: Programming your Robot

CONTROL SYSTEM OVERVIEWS

Hardware Component Overview

Software Component Overview

PROGRAMMING BASICS

What is WPILib?

2022 Overview

VS Code Overview

Dashboards

Telemetry

FRC LabVIEW Programming

Hardware APIs

CAN Devices

Basic Programming

Support Resources

FRC Glossary

Read the Docs v: stable

FRC Game Manual FRC Game Q&A

» FIRST Robotics Competition Control System

Edit on GitHub Present

FIRST Robotics Competition Control System

Welcome to the *FIRST*® Robotics Competition Control System Documentation! This site contains everything you need to know for programming a competition robot!

Community translations can be found in a variety of languages in the bottom-left menu.

Returning Teams

If you are a returning team, please check out the overview of changes from 2021 to 2022 and the known issues.

Changelog

Known Issues

New Teams

The Zero-to-Robot tutorial will guide you through preparation, wiring and programming a basic robot!

Go to Zero-to-Robot

Hardware Overview

An overview of the hardware components available to teams.

Go to Hardware Overview

Software Overview

An overview of the software components and tools available to teams.

Go to Software Overview

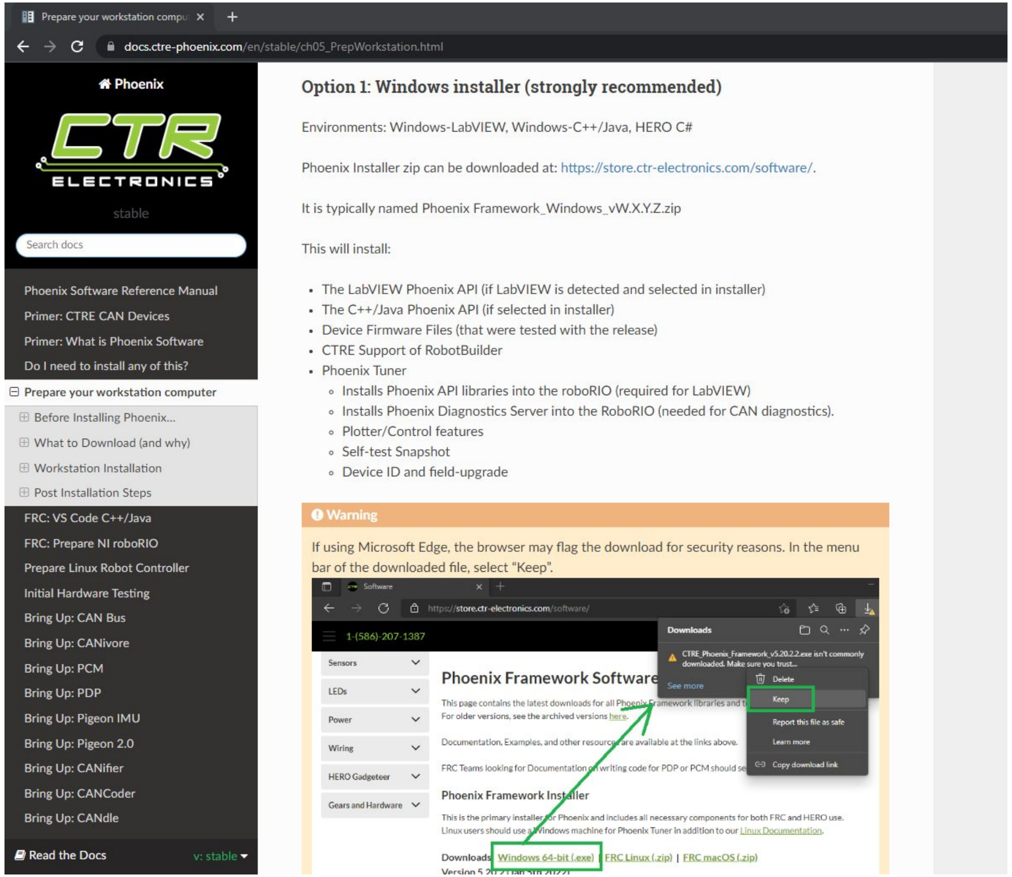
Programming Basics

Advanced Programming

Hardware

CTR Electronics Documentation

- Important tools like Phoenix tuner
- Vendor file for code for CTRE Products



Option 1: Windows installer (strongly recommended)

Environments: Windows-LabVIEW, Windows-C++/Java, HERO C#

Phoenix Installer zip can be downloaded at: https://store.ctr-electronics.com/software/.

It is typically named Phoenix Framework_Windows_vW.X.Y.Z.zip

This will install:

- The LabVIEW Phoenix API (if LabVIEW is detected and selected in installer)
- The C++/Java Phoenix API (if selected in installer)
- Device Firmware Files (that were tested with the release)
- CTRE Support of RobotBuilder
- Phoenix Tuner
  - Installs Phoenix API libraries into the roboRIO (required for LabVIEW)
  - Installs Phoenix Diagnostics Server into the RoboRIO (needed for CAN diagnostics).
  - Plotter/Control features
  - Self-test Snapshot
  - Device ID and field-upgrade

Warning

If using Microsoft Edge, the browser may flag the download for security reasons. In the menu bar of the downloaded file, select "Keep".

Phoenix Framework Software

This page contains the latest downloads for all Phoenix Framework libraries and t
For older versions, see the archived versions here.

Documentation, Examples, and other resources are available at the links above.

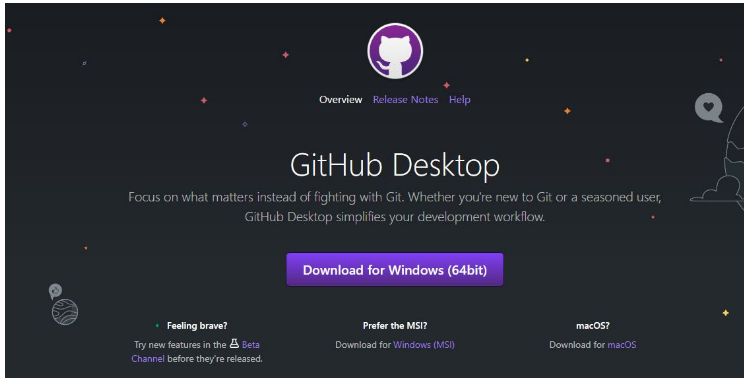
FRC Teams looking for Documentation on writing code for PDP or PCM should se

Phoenix Framework Installer

This is the primary installer for Phoenix and includes all necessary components for both FRC and HERO use. Linux users should use a Windows machine for Phoenix Tuner in addition to our Linux Documentation.

Downloads Windows 64-bit (.exe) | FRC Linux (.zip) | FRC macOS (.zip)

Github Desktop



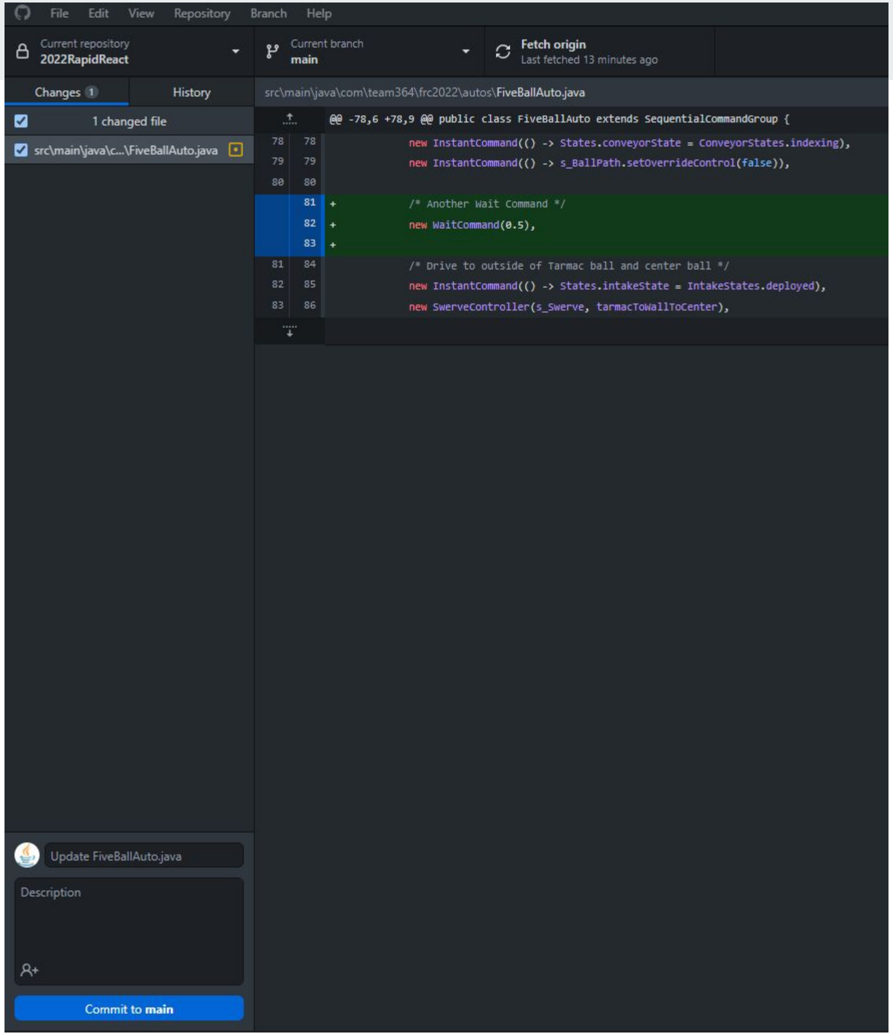
GitHub Desktop

Focus on what matters instead of fighting with Git. Whether you're new to Git or a seasoned user, GitHub Desktop simplifies your development workflow.

Feeling brave?
Try new features in the Beta Channel before they're released.

Prefer the MSI?
Download for Windows (MSI)

macOS?
Download for macOS



Current repository: 2022RapidReact
Current branch: main
Last fetched 13 minutes ago

1 changed file
src\main\java\c...\FiveBallAuto.java

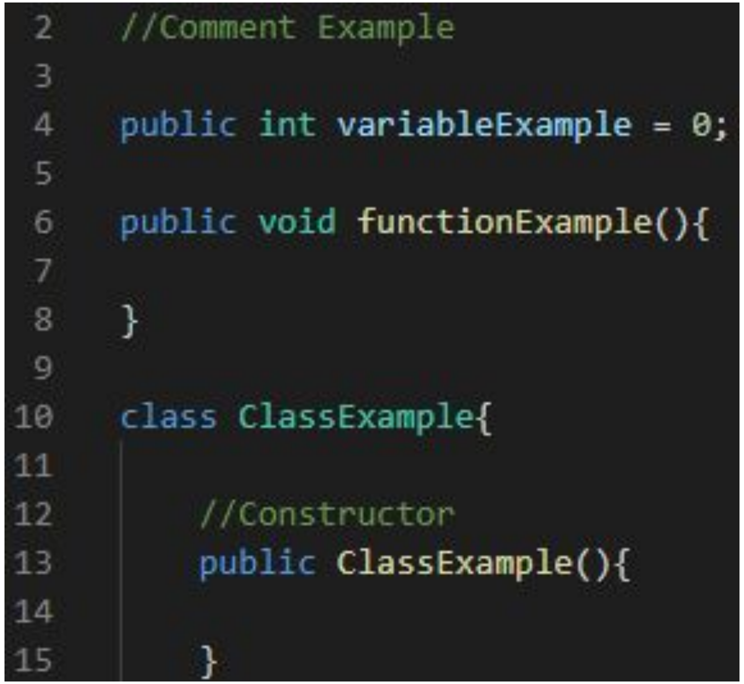
src\main\java\com\team364\frc2022\autos\FiveBallAuto.java

```
@@ -78,6 +78,9 @@ public class FiveBallAuto extends SequentialCommandGroup {
                new InstantCommand(() -> States.conveyorState = ConveyorStates.indexing),
                new InstantCommand(() -> s_BallPath.setOverrideControl(false)),

+               /* Another Wait Command */
+               new WaitCommand(0.5),
+
                /* Drive to outside of Tarmac ball and center ball */
                new InstantCommand(() -> States.intakeState = IntakeStates.deployed),
                new SwerveController(s_Swerve, tarmacToWallToCenter),
```

Update FiveBallAuto.java
Description

Brief Overview of Programming

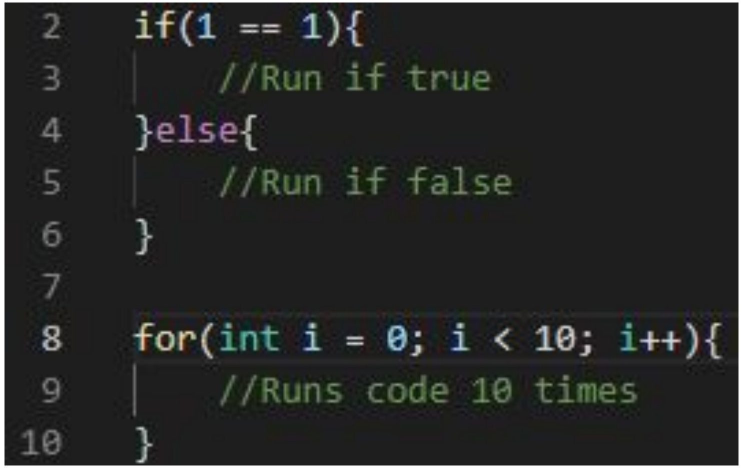
- Comments
- Modifiers, Type, Name, Value
- Variables
- Functions
- Classes
- Constructors
- Objects



```
//Comment Example

public int variableExample = 0;

public void functionExample(){

}

class ClassExample{

    //Constructor
    public ClassExample(){

    }
```

Brief Overview of Programming Logic

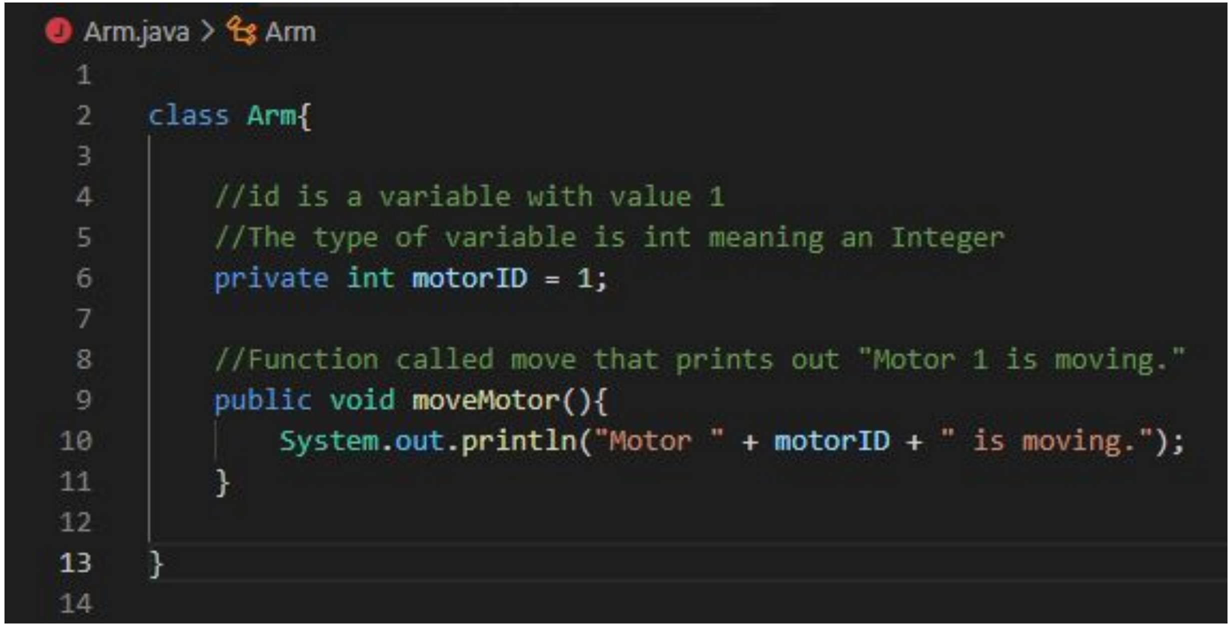
- If statements
- If else statements
- For loops



```
if(1 == 1){
    //Run if true
}else{
    //Run if false
}

for(int i = 0; i < 10; i++){
    //Runs code 10 times
}
```

Arm Class

- Arm – (Class)
- motorID – (Variable)
- moveMotor() – (Function)


Arm.java > Arm

```java

class Arm{

    //id is a variable with value 1
    //The type of variable is int meaning an Integer
    private int motorID = 1;

    //Function called move that prints out "Motor 1 is moving."
    public void moveMotor(){
        System.out.println("Motor " + motorID + " is moving.");
    }

}

```

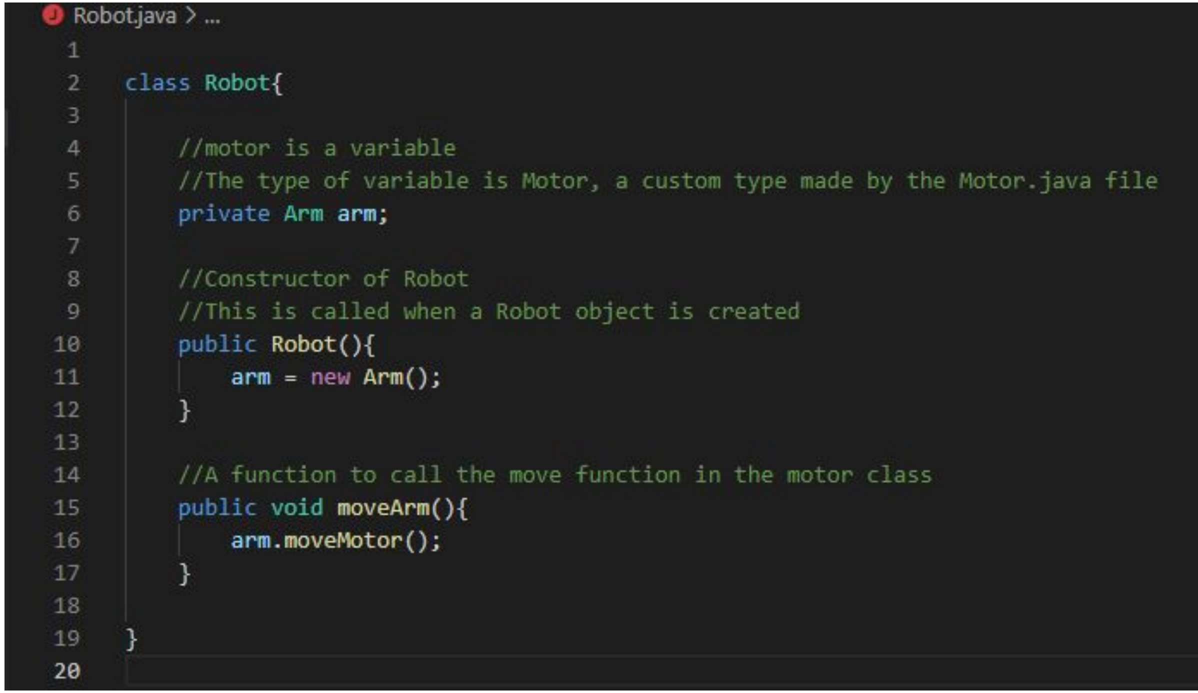
Variable Example

- private – (Modifier)
- int – (Type)
- motorID – (Name)
- 1 – (Value)

```
private int id = 1;
```

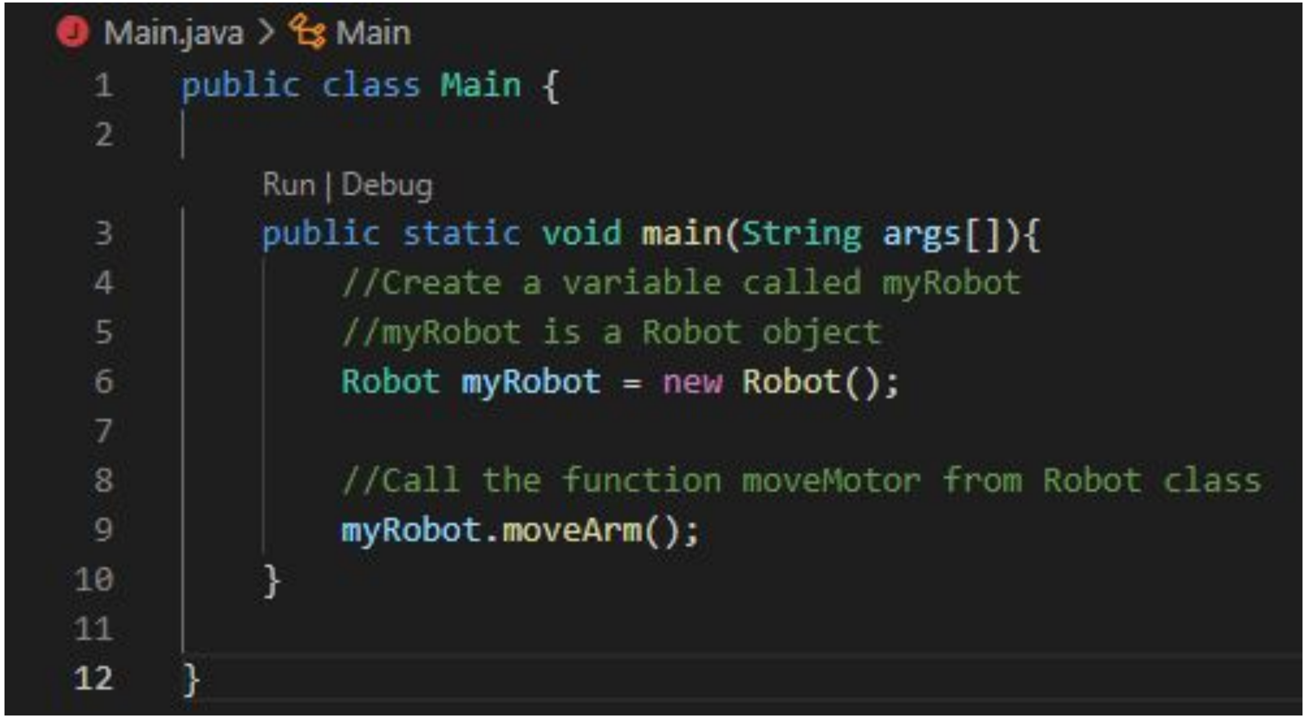
Robot Class

- Robot — (Class)
- arm — (Variable)
- Robot() — (Constructor)
- moveArm() — (Function)



```java

class Robot{

    //motor is a variable
    //The type of variable is Motor, a custom type made by the Motor.java file
    private Arm arm;

    //Constructor of Robot
    //This is called when a Robot object is created
    public Robot(){
        arm = new Arm();
    }

    //A function to call the move function in the motor class
    public void moveArm(){
        arm.moveMotor();
    }

}

```

Main Class



```java
public class Main {

    public static void main(String args[]){
        //Create a variable called myRobot
        //myRobot is a Robot object
        Robot myRobot = new Robot();

        //Call the function moveMotor from Robot class
        myRobot.moveArm();
    }

}
```

Why Object Oriented

$$\begin{tikzcd}
\underline{\text{Arm.java}} & \underline{\text{Robot.java}} & \underline{\text{Main.java}} \\
\text{moveMotor()} \arrow[r] & \text{moveArm()} \arrow[r] & \text{myRobot.moveArm()}
\end{tikzcd}$$

Why not just:

<u>Main.java</u>
moveMotor()

<u>Main.java</u>

setOverrideControl()
setLowerConveyorPower()
setUpperConveyorPower()
setPooperPower()
getLowerSensor()
getUpperSensor()
getPooperSensor()

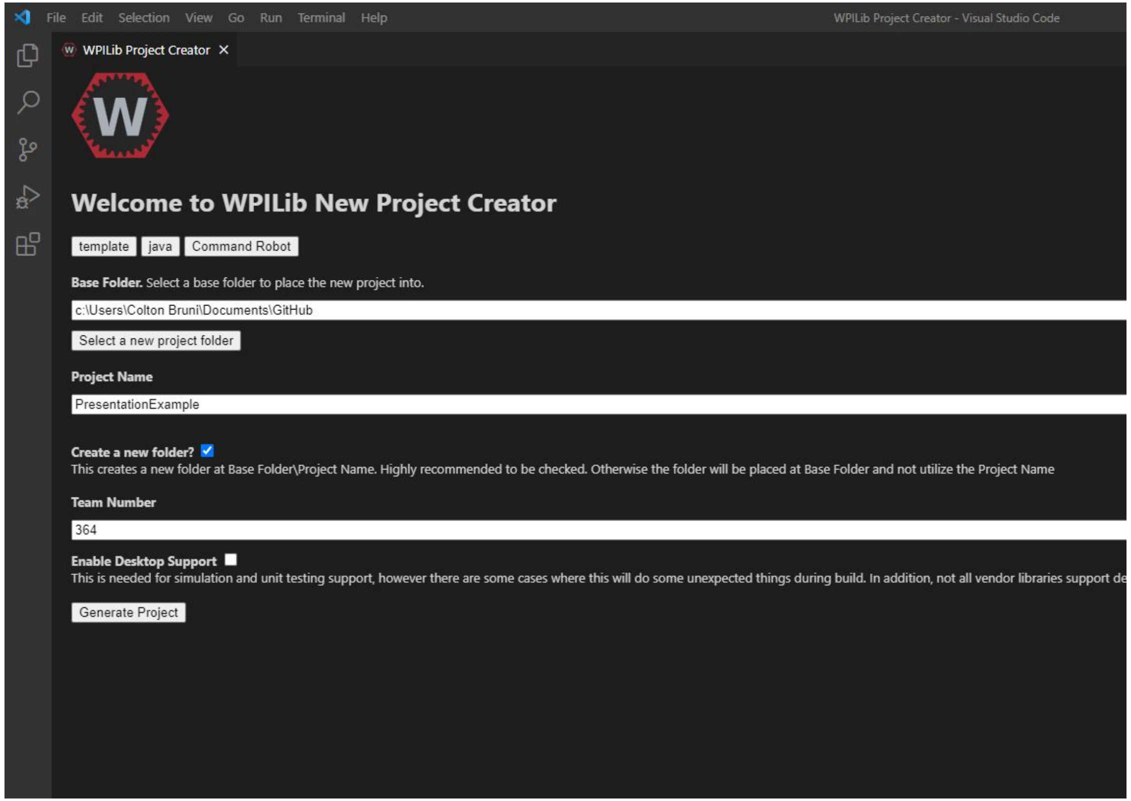
isAllianceBall()
numberOfBalls()
getAngle()
setAngle()
getTargetAngle()
resetHood()
setPower()

setPiston()
getRPM()
setRPM()
getTargetRPM()
drive()
setModuleStates()
getAutoPose()

resetOdometry()
getStates()
zeroModules()
zeroGyro()
setGyro()
getYaw()
getRoll()

getRobotVelocity()
setPosition()
checkSoftLimits()
targetVisible()
getDegreePosition()
getSoftLimited()
resetTurret()

Getting Started


File Edit Selection View Go Run Terminal Help
WPILib Project Creator - Visual Studio Code
WPILib Project Creator

Welcome to WPILib New Project Creator

template | java | Command Robot

Base Folder. Select a base folder to place the new project into.

c:\Users\Colton Bruni\Documents\GitHub

Select a new project folder

Project Name

PresentationExample

Create a new folder? ✓
This creates a new folder at Base Folder\Project Name. Highly recommended to be checked. Otherwise the folder will be placed at Base Folder and not utilize the Project Name

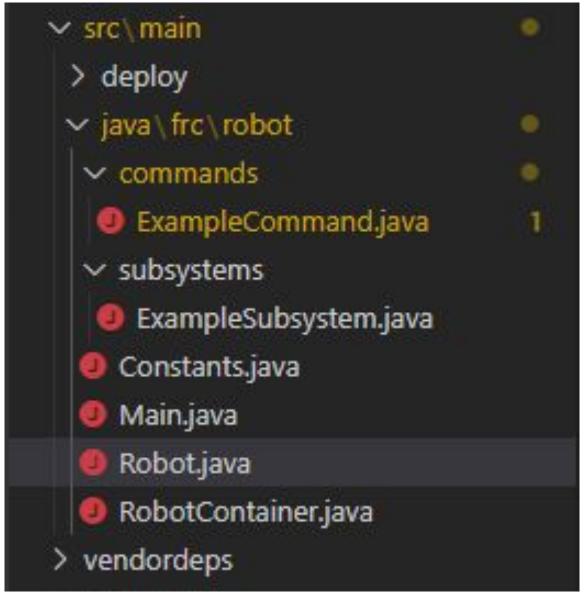
Team Number

364

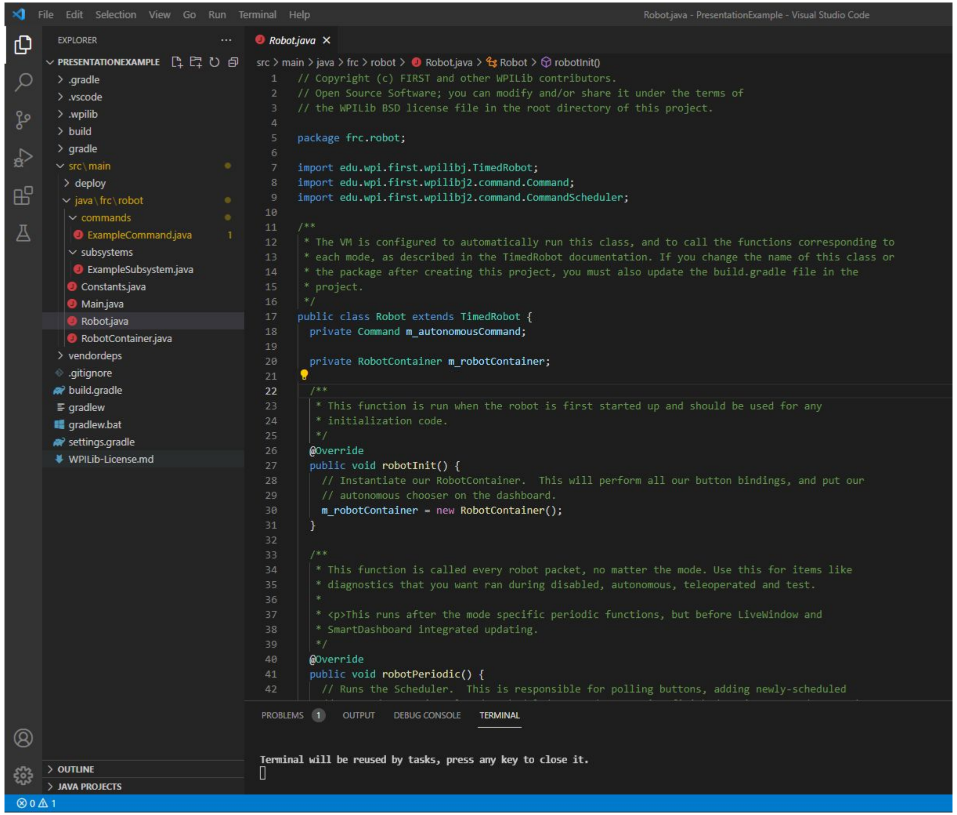
Enable Desktop Support ☐
This is needed for simulation and unit testing support, however there are some cases where this will do some unexpected things during build. In addition, not all vendor libraries support de

Generate Project

Files and Classes



```
src\main
  deploy
  java\frc\robot
    commands
      ExampleCommand.java    1
    subsystems
      ExampleSubsystem.java
    Constants.java
    Main.java
    Robot.java
    RobotContainer.java
  vendordeps
```


File Edit Selection View Go Run Terminal Help
Robot.java - PresentationExample - Visual Studio Code
EXPLORER
PRESENTATIONEXAMPLE
Robot.java
src > main > java > frc > robot > Robot.java > Robot > robotInit()
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
OUTLINE
JAVA PROJECTS
0 1

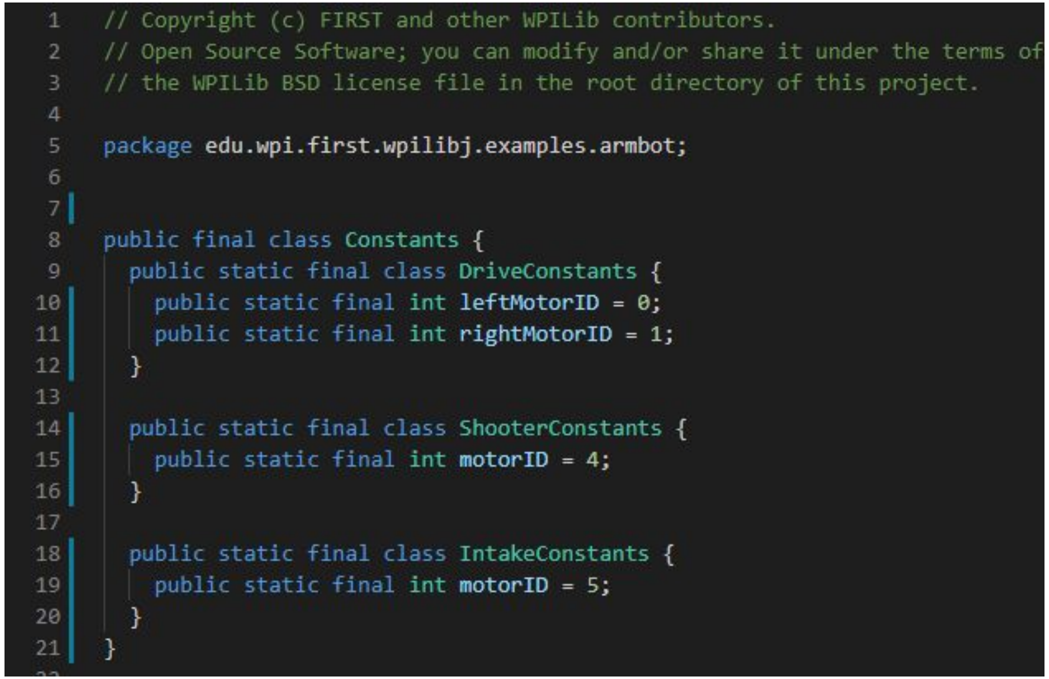
```
PRESENTATIONEXAMPLE
  .gradle
  .vscode
  .wpilib
  build
  gradle
  src\main
    deploy
    java\frc\robot
      commands
        ExampleCommand.java   1
      subsystems
        ExampleSubsystem.java
      Constants.java
      Main.java
      Robot.java
      RobotContainer.java
  vendordeps
  .gitignore
  build.gradle
  gradlew
  gradlew.bat
  settings.gradle
  WPILib-License.md
```

```java
// Copyright (c) FIRST and other WPILib contributors.
// Open Source Software; you can modify and/or share it under the terms of
// the WPILib BSD license file in the root directory of this project.

package frc.robot;

import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.wpilibj2.command.Command;
import edu.wpi.first.wpilibj2.command.CommandScheduler;

/**
 * The VM is configured to automatically run this class, and to call the functions corresponding to
 * each mode, as described in the TimedRobot documentation. If you change the name of this class or
 * the package after creating this project, you must also update the build.gradle file in the
 * project.
 */
public class Robot extends TimedRobot {
  private Command m_autonomousCommand;

  private RobotContainer m_robotContainer;

  /**
   * This function is run when the robot is first started up and should be used for any
   * initialization code.
   */
  @Override
  public void robotInit() {
    // Instantiate our RobotContainer.  This will perform all our button bindings, and put our
    // autonomous chooser on the dashboard.
    m_robotContainer = new RobotContainer();
  }

  /**
   * This function is called every robot packet, no matter the mode. Use this for items like
   * diagnostics that you want ran during disabled, autonomous, teleoperated and test.
   *
   * <p>This runs after the mode specific periodic functions, but before LiveWindow and
   * SmartDashboard integrated updating.
   */
  @Override
  public void robotPeriodic() {
    // Runs the Scheduler.  This is responsible for polling buttons, adding newly-scheduled
```

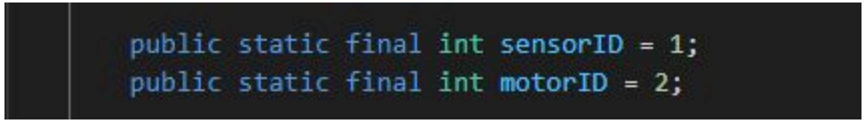
```
Terminal will be reused by tasks, press any key to close it.
```

Constants

- Hold variables
  - Should be final
- Use classes to separate subsystems

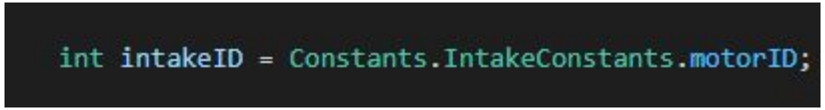


```java
// Copyright (c) FIRST and other WPILib contributors.
// Open Source Software; you can modify and/or share it under the terms of
// the WPILib BSD license file in the root directory of this project.

package edu.wpi.first.wpilibj.examples.armbot;


public final class Constants {
  public static final class DriveConstants {
    public static final int leftMotorID = 0;
    public static final int rightMotorID = 1;
  }

  public static final class ShooterConstants {
    public static final int motorID = 4;
  }

  public static final class IntakeConstants {
    public static final int motorID = 5;
  }
}
```



```java
    public static final int sensorID = 1;
    public static final int motorID = 2;
```

→

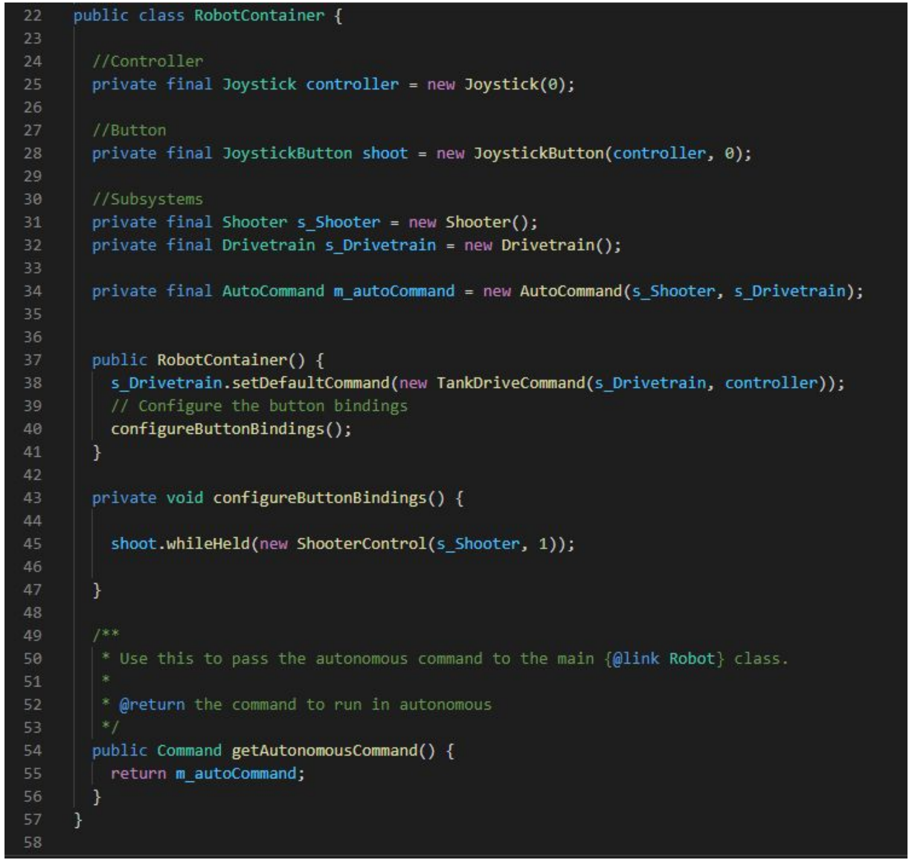


```java
int intakeID = Constants.IntakeConstants.motorID;
```

RobotContainer

- Initialize controllers, buttons, and subsystems

- Configure button bindings



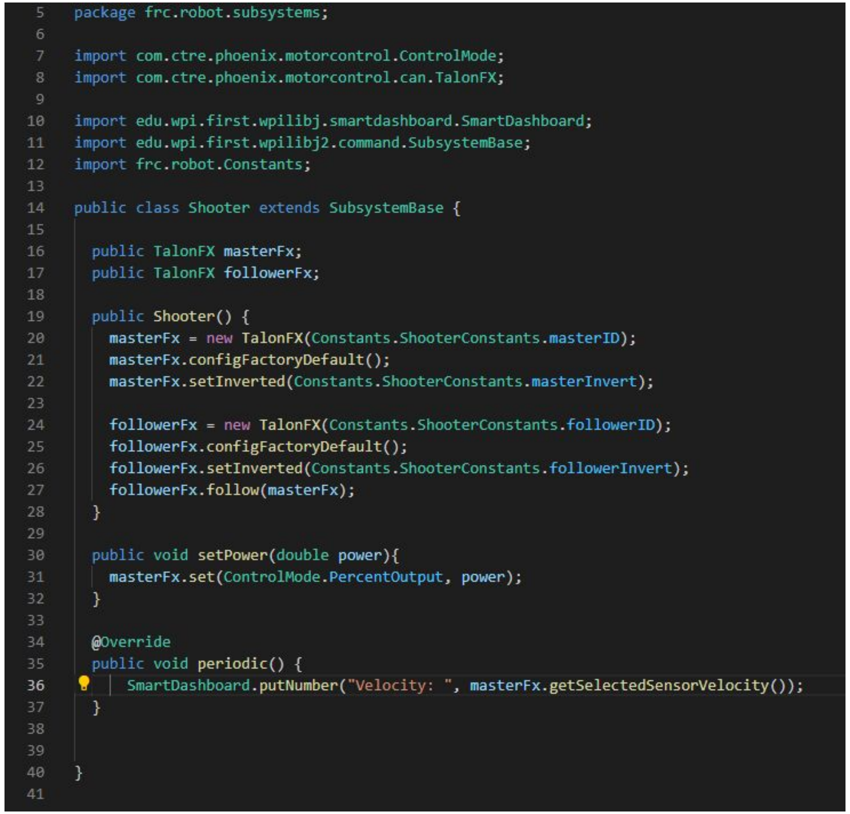
```java
public class RobotContainer {

  //Controller
  private final Joystick controller = new Joystick(0);

  //Button
  private final JoystickButton shoot = new JoystickButton(controller, 0);

  //Subsystems
  private final Shooter s_Shooter = new Shooter();
  private final Drivetrain s_Drivetrain = new Drivetrain();

  private final AutoCommand m_autoCommand = new AutoCommand(s_Shooter, s_Drivetrain);


  public RobotContainer() {
    s_Drivetrain.setDefaultCommand(new TankDriveCommand(s_Drivetrain, controller));
    // Configure the button bindings
    configureButtonBindings();
  }

  private void configureButtonBindings() {

    shoot.whileHeld(new ShooterControl(s_Shooter, 1));

  }

  /**
   * Use this to pass the autonomous command to the main {@link Robot} class.
   *
   * @return the command to run in autonomous
   */
  public Command getAutonomousCommand() {
    return m_autoCommand;
  }
}
```

Robot

- Periodic and Init functions
- Typically do not need to run code, except for telemetry

Subsystems

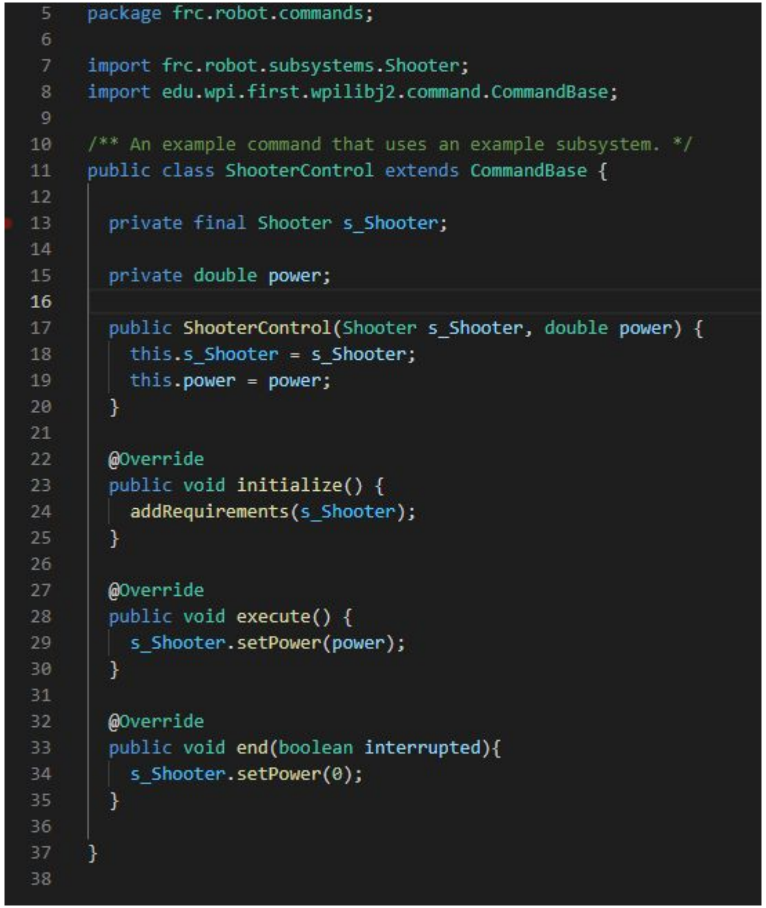
- Extends SubsystemBase
- Typically composed of:
    - Constructor
    - periodic()
    - custom functions
- Contains variables for motors
- Constructor initializes motors



```java
package frc.robot.subsystems;

import com.ctre.phoenix.motorcontrol.ControlMode;
import com.ctre.phoenix.motorcontrol.can.TalonFX;

import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import edu.wpi.first.wpilibj2.command.SubsystemBase;
import frc.robot.Constants;

public class Shooter extends SubsystemBase {

  public TalonFX masterFx;
  public TalonFX followerFx;

  public Shooter() {
    masterFx = new TalonFX(Constants.ShooterConstants.masterID);
    masterFx.configFactoryDefault();
    masterFx.setInverted(Constants.ShooterConstants.masterInvert);

    followerFx = new TalonFX(Constants.ShooterConstants.followerID);
    followerFx.configFactoryDefault();
    followerFx.setInverted(Constants.ShooterConstants.followerInvert);
    followerFx.follow(masterFx);
  }

  public void setPower(double power){
    masterFx.set(ControlMode.PercentOutput, power);
  }

  @Override
  public void periodic() {
    SmartDashboard.putNumber("Velocity: ", masterFx.getSelectedSensorVelocity());
  }


}
```

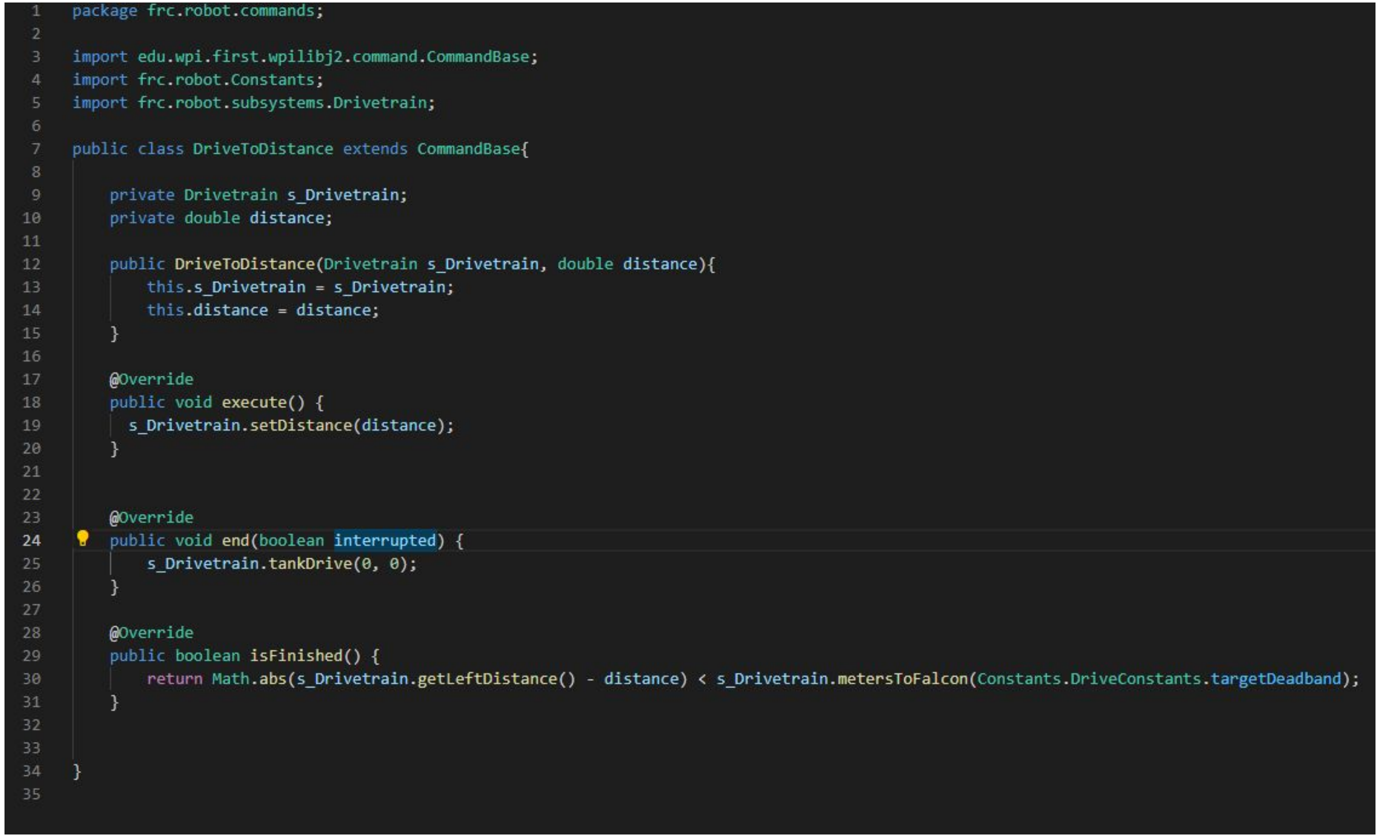
Commands

- Extends CommandBase
- Typically composed of:
  - initialize()
  - execute()
  - end()
  - isFinished() *



```java
package frc.robot.commands;

import frc.robot.subsystems.Shooter;
import edu.wpi.first.wpilibj2.command.CommandBase;

/** An example command that uses an example subsystem. */
public class ShooterControl extends CommandBase {

  private final Shooter s_Shooter;

  private double power;

  public ShooterControl(Shooter s_Shooter, double power) {
    this.s_Shooter = s_Shooter;
    this.power = power;
  }

  @Override
  public void initialize() {
    addRequirements(s_Shooter);
  }

  @Override
  public void execute() {
    s_Shooter.setPower(power);
  }

  @Override
  public void end(boolean interrupted){
    s_Shooter.setPower(0);
  }

}
```

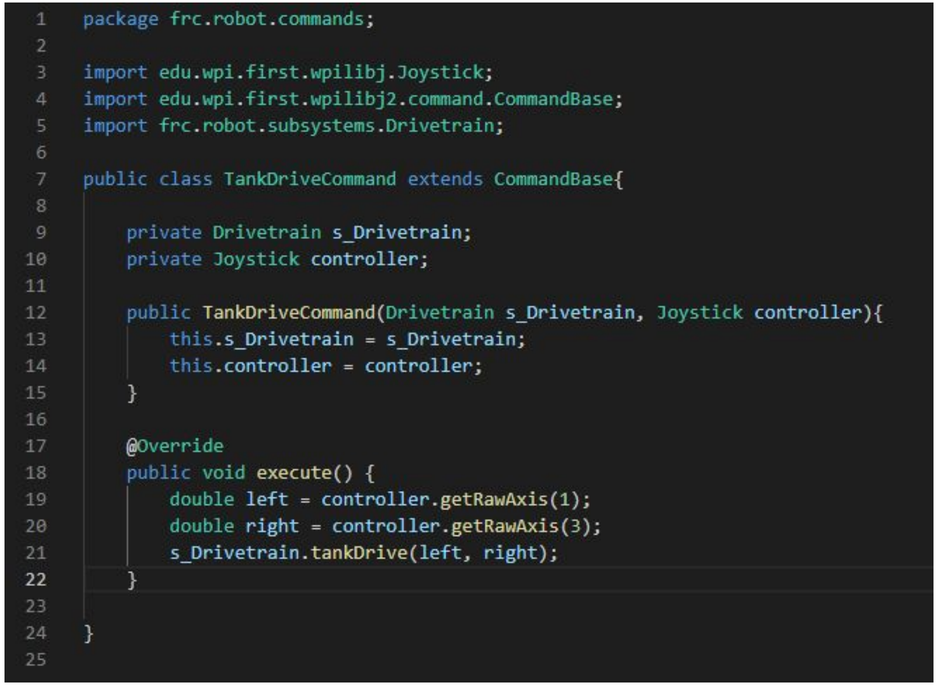
Commands (isFinished)



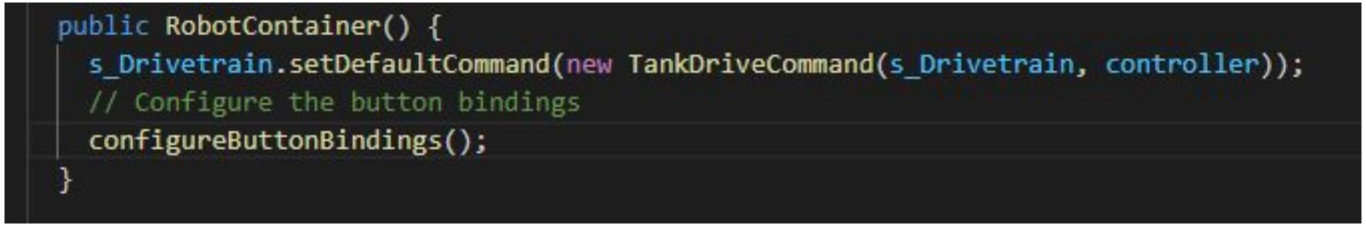
```java
package frc.robot.commands;

import edu.wpi.first.wpilibj2.command.CommandBase;
import frc.robot.Constants;
import frc.robot.subsystems.Drivetrain;

public class DriveToDistance extends CommandBase{

    private Drivetrain s_Drivetrain;
    private double distance;

    public DriveToDistance(Drivetrain s_Drivetrain, double distance){
        this.s_Drivetrain = s_Drivetrain;
        this.distance = distance;
    }

    @Override
    public void execute() {
      s_Drivetrain.setDistance(distance);
    }


    @Override
    public void end(boolean interrupted) {
        s_Drivetrain.tankDrive(0, 0);
    }

    @Override
    public boolean isFinished() {
        return Math.abs(s_Drivetrain.getLeftDistance() - distance) < s_Drivetrain.metersToFalcon(Constants.DriveConstants.targetDeadband);
    }


}
```

Default Commands

- Run continuously in teleop
- Set in RobotContainer



```java
package frc.robot.commands;

import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj2.command.CommandBase;
import frc.robot.subsystems.Drivetrain;

public class TankDriveCommand extends CommandBase{

    private Drivetrain s_Drivetrain;
    private Joystick controller;

    public TankDriveCommand(Drivetrain s_Drivetrain, Joystick controller){
        this.s_Drivetrain = s_Drivetrain;
        this.controller = controller;
    }

    @Override
    public void execute() {
        double left = controller.getRawAxis(1);
        double right = controller.getRawAxis(3);
        s_Drivetrain.tankDrive(left, right);
    }

}
```



```java
public RobotContainer() {
  s_Drivetrain.setDefaultCommand(new TankDriveCommand(s_Drivetrain, controller));
  // Configure the button bindings
  configureButtonBindings();
}
```

Structure

Subsystems

- Drivetrain
  - tankdrive()
  - setDistance()
  - getLeftDistance()
- Shooter
  - setPower

Commands

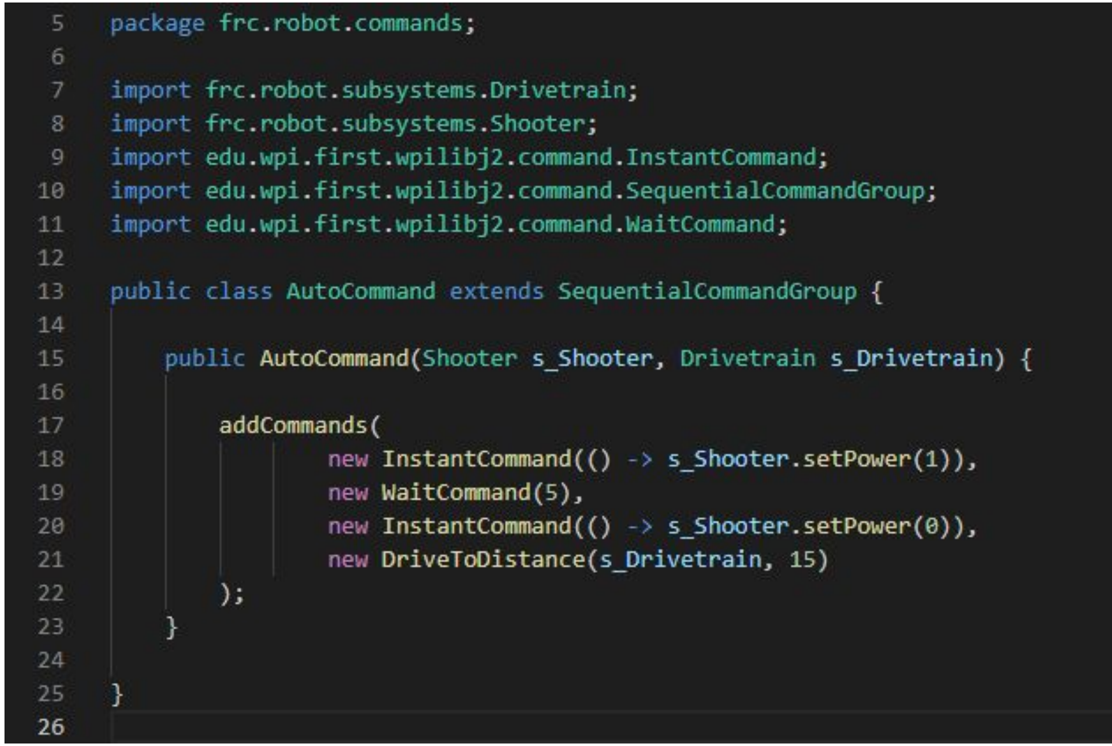
- TankDriveCommand
- DriveToDistance
- ShooterControl

RobotContainer

- Default command
  - TankDriveCommand (uses joystick axis to drive)
- shoot button
  - ShooterControl (uses joystick button to turn on shooter)

Autonomous

- Extends SequentialCommandGroup
- Wpilib Commands
- Functions as InstantCommands
- Call other commands from the program



```java
package frc.robot.commands;

import frc.robot.subsystems.Drivetrain;
import frc.robot.subsystems.Shooter;
import edu.wpi.first.wpilibj2.command.InstantCommand;
import edu.wpi.first.wpilibj2.command.SequentialCommandGroup;
import edu.wpi.first.wpilibj2.command.WaitCommand;

public class AutoCommand extends SequentialCommandGroup {

    public AutoCommand(Shooter s_Shooter, Drivetrain s_Drivetrain) {

        addCommands(
                new InstantCommand(() -> s_Shooter.setPower(1)),
                new WaitCommand(5),
                new InstantCommand(() -> s_Shooter.setPower(0)),
                new DriveToDistance(s_Drivetrain, 15)
        );
    }

}

```