



# Programming for FRC



## 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)

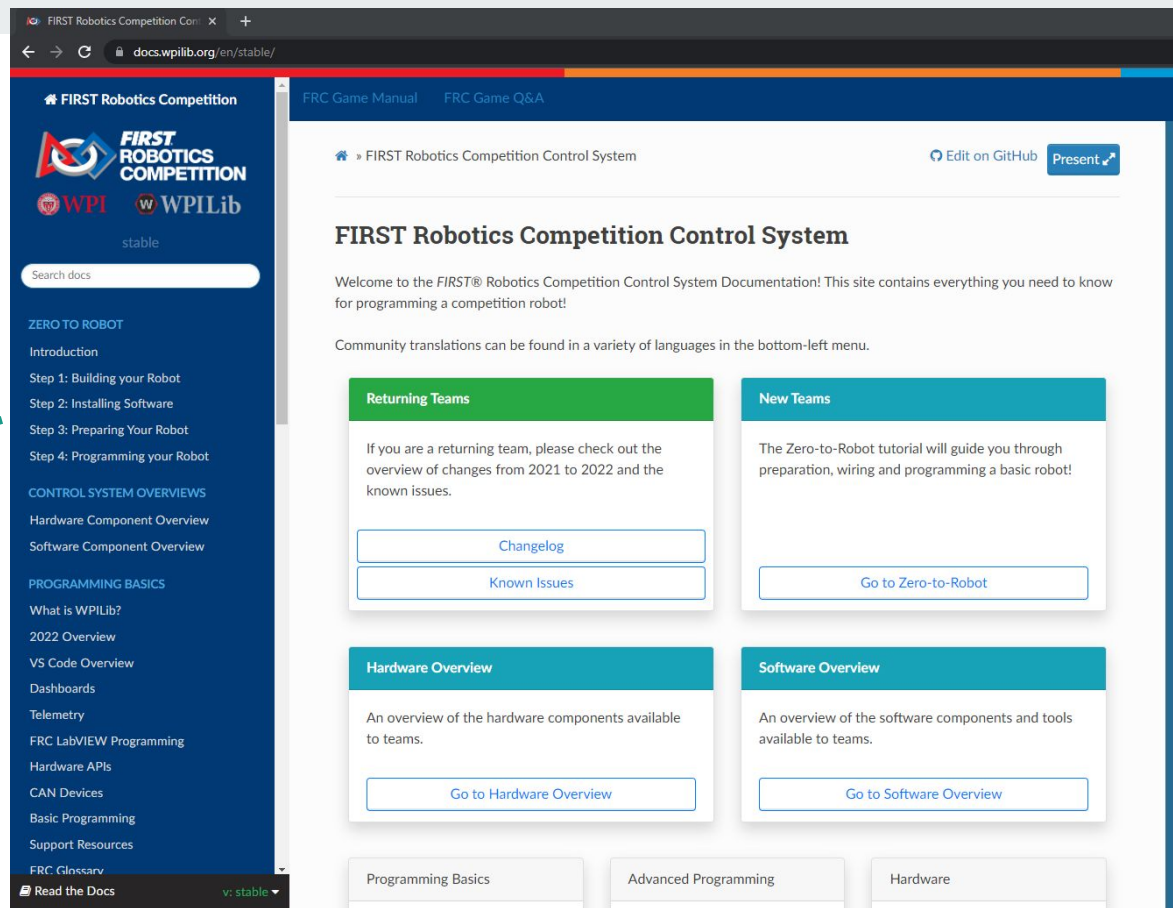
Installing the FRC Game Tools

WPILib Installation Guide

Next Steps

Step 3: Preparing Your Robot

Step 4: Programming your Robot



The screenshot shows the WPILib documentation website. The left sidebar contains a navigation menu with the following sections: **FIRST Robotics Competition** (with logos for FRC, WPI, and WPILib), **stable**, **Search docs**, **ZERO TO ROBOT** (with links for Introduction, Step 1: Building your Robot, Step 2: Installing Software, and Step 4: Programming your Robot), **CONTROL SYSTEM OVERVIEWS** (with links for Hardware Component Overview and Software Component Overview), **PROGRAMMING BASICS** (with links for What is WPILib?, 2022 Overview, VS Code Overview, Dashboards, Telemetry, FRC LabVIEW Programming, Hardware APIs, CAN Devices, Basic Programming, and Support Resources), and **FRC Glossary**. The main content area is titled **FIRST Robotics Competition Control System** and includes a welcome message, a link to **Edit on GitHub**, and a **Present** button. Below the welcome message, there are four featured sections: **Returning Teams** (with a **Changelog** button), **New Teams** (with a **Go to Zero-to-Robot** button), **Hardware Overview** (with a **Go to Hardware Overview** button), and **Software Overview** (with a **Go to Software Overview** button). At the bottom, there are three tabs: **Programming Basics**, **Advanced Programming**, and **Hardware**.

# CTR Electronics Documentation

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

The screenshot displays the CTR Electronics Documentation website. The left sidebar contains a navigation menu with the following items: Phoenix Software Reference Manual, Primer: CTRE CAN Devices, Primer: What is Phoenix Software, Do I need to install any of this?, Prepare your workstation computer, Before Installing Phoenix..., What to Download (and why), Workstation Installation, Post Installation Steps, FRC: VS Code C++/Java, FRC: Prepare NI roboRIO, Prepare Linux Robot Controller, Initial Hardware Testing, Bring Up: CAN Bus, Bring Up: CANivore, Bring Up: PCM, Bring Up: PDP, Bring Up: Pigeon IMU, Bring Up: Pigeon 2.0, Bring Up: CANifier, Bring Up: CANCoder, and Bring Up: CANDLE. The main content area is titled 'Option 1: Windows installer (strongly recommended)' and provides instructions for installing the Phoenix Framework on Windows. It includes a list of components that will be installed, such as the LabVIEW Phoenix API, C++/Java Phoenix API, Device Firmware Files, CTRE Support of RobotBuilder, and Phoenix Tuner. A warning box at the bottom states: 'Warning: If using Microsoft Edge, the browser may flag the download for security reasons. In the menu bar of the downloaded file, select "Keep".' The download link for the Phoenix Framework Software is highlighted with a green box, and the 'Keep' option in the download menu is also highlighted with a green box.

Prepare your workstation computer x +

docs.ctr-phoenix.com/en/stable/ch05\_PrepWorkstation.html

Phoenix

CTR ELECTRONICS

stable

Search docs

Phoenix Software Reference Manual  
Primer: CTRE CAN Devices  
Primer: What is Phoenix Software  
Do I need to install any of this?

Prepare your workstation computer

Before Installing Phoenix...  
What to Download (and why)  
Workstation Installation  
Post Installation Steps

FRC: VS Code C++/Java  
FRC: Prepare NI roboRIO  
Prepare Linux Robot Controller  
Initial Hardware Testing  
Bring Up: CAN Bus  
Bring Up: CANivore  
Bring Up: PCM  
Bring Up: PDP  
Bring Up: Pigeon IMU  
Bring Up: Pigeon 2.0  
Bring Up: CANifier  
Bring Up: CANCoder  
Bring Up: CANDLE

Read the Docs v: stable

### 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 tools. For older versions, see the archived versions [here](#).

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

FRC Teams looking for Documentation or writing code for PDP or PCM should see [here](#).

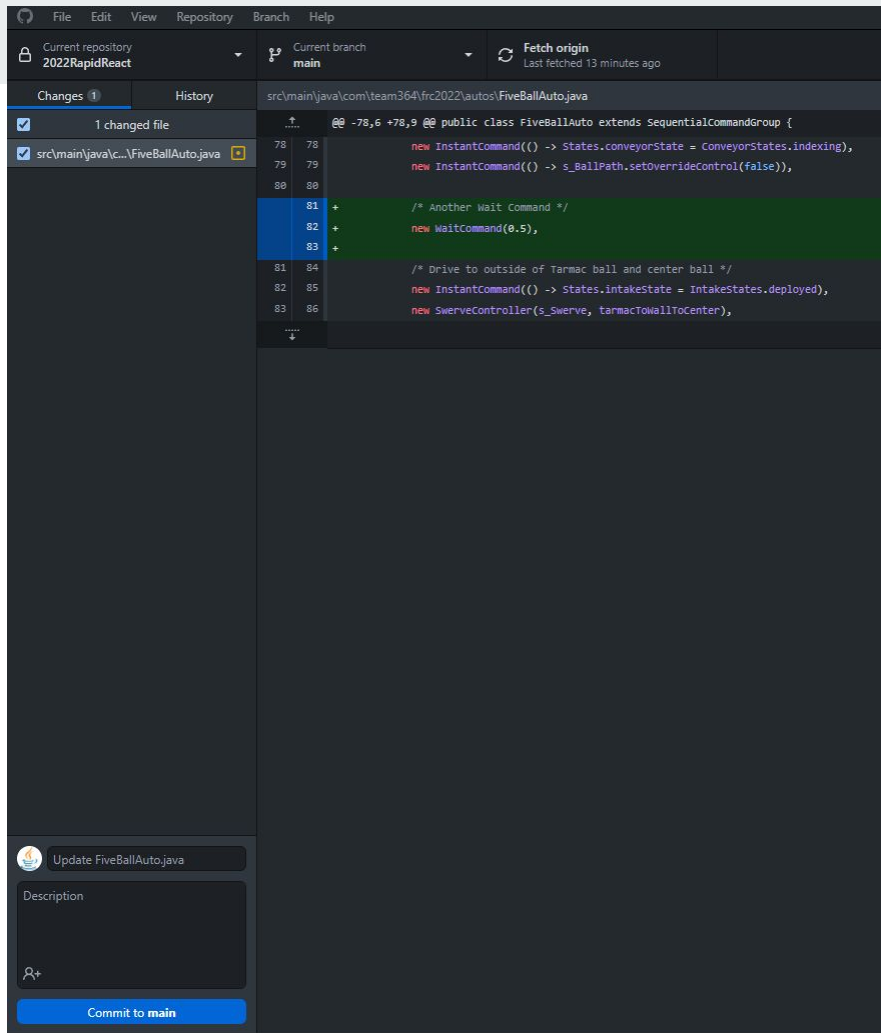
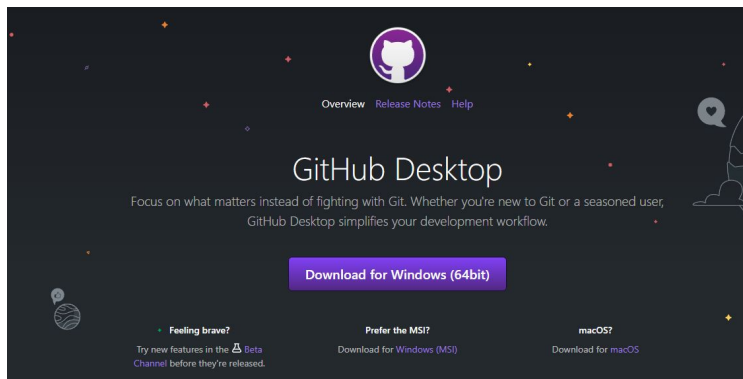
**Phoenix Framework Installer**

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

Downloads: [Windows 64-bit \(.exe\)](#) | [FRC Linux \(.zip\)](#) | [FRC macOS \(.zip\)](#)

Version 5.20.2.2 (2022.02.22)

# Github Desktop





## Brief Overview of Programming

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

```
2  //Comment Example
3
4  public int variableExample = 0;
5
6  public void functionExample(){
7
8  }
9
10 class ClassExample{
11
12     //Constructor
13     public ClassExample(){
14
15     }
```



## Brief Overview of Programming Logic

- If statements
- If else statements
- For loops

```
2  if(1 == 1){  
3      //Run if true  
4  }else{  
5      //Run if false  
6  }  
7  
8  for(int i = 0; i < 10; i++){  
9      //Runs code 10 times  
10 }
```

## Arm Class

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

```
Arm.java > Arm
1
2 class Arm{
3
4     //id is a variable with value 1
5     //The type of variable is int meaning an Integer
6     private int motorID = 1;
7
8     //Function called move that prints out "Motor 1 is moving."
9     public void moveMotor(){
10         System.out.println("Motor " + motorID + " is moving.");
11     }
12
13 }
14
```





## Variable Example

- `private` — (Modifier)
- `int` — (Type)
- `motorID` — (Name)
- `1` — (Value)

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



## Robot Class

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

```
Robot.java > ...
1
2 class Robot{
3
4     //motor is a variable
5     //The type of variable is Motor, a custom type made by the Motor.java file
6     private Arm arm;
7
8     //Constructor of Robot
9     //This is called when a Robot object is created
10    public Robot(){
11        arm = new Arm();
12    }
13
14    //A function to call the move function in the motor class
15    public void moveArm(){
16        arm.moveMotor();
17    }
18
19 }
20
```

## Main Class

```
Main.java > Main
1  public class Main {
2
3      Run | Debug
4      public static void main(String args[]){
5          //Create a variable called myRobot
6          //myRobot is a Robot object
7          Robot myRobot = new Robot();
8
9          //Call the function moveMotor from Robot class
10         myRobot.moveArm();
11     }
12 }
```



# Why Object Oriented

Why not just:

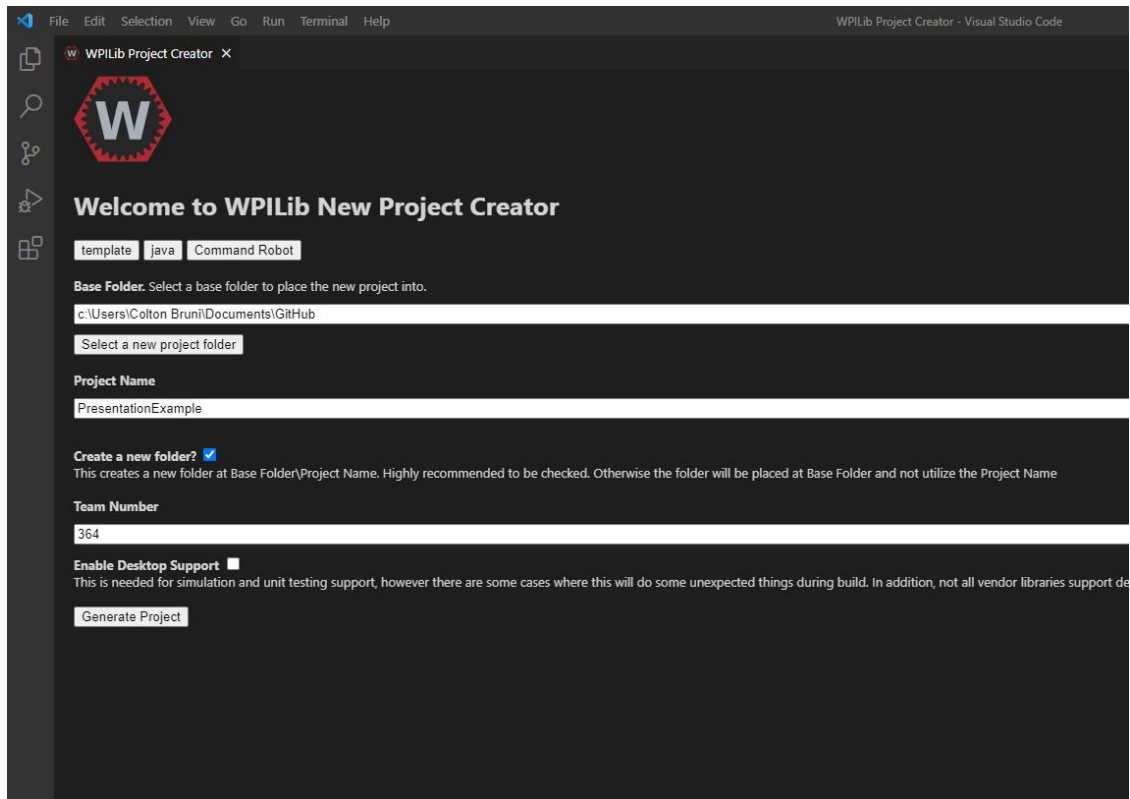
Arm.java  
moveMotor() → Robot.java  
moveArm() → Main.java  
myRobot.moveArm()

Main.java  
moveMotor()

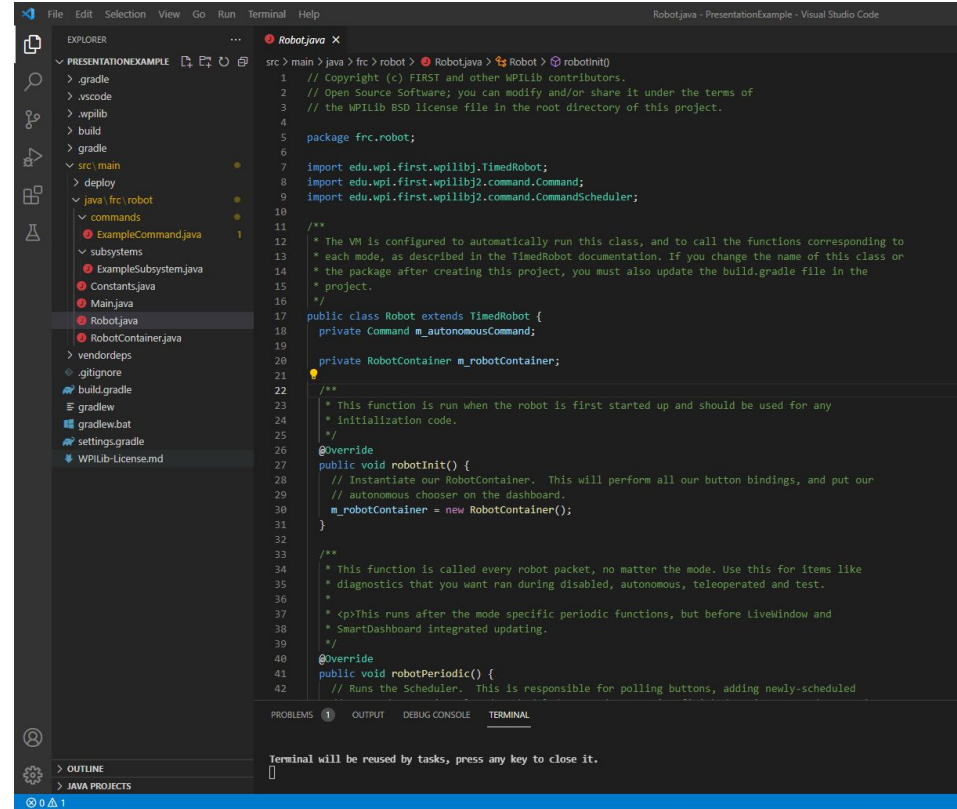
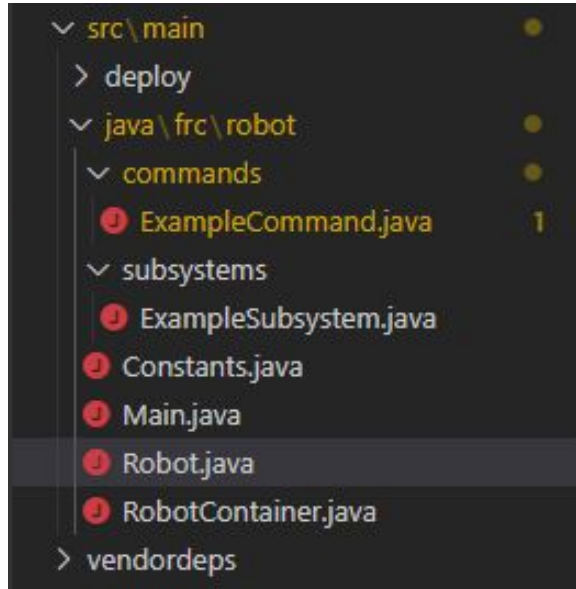
<u>Main.java</u>				
setOverrideControl()	isAllianceBall()	setPiston()	resetOdometry()	getRobotVelocity()
setLowerConveyorPower()	numberOfBalls()	getRPM()	getStates()	setPosition()
setUpperConveyorPower()	getAngle()	setRPM()	zeroModules()	checkSoftLimits()
setPooperPower()	setAngle()	getTargetRPM()	zeroGyro()	targetVisible()
getLowerSensor()	getTargetAngle()	drive()	setGyro()	getDegreePosition()
getUpperSensor()	resetHood()	setModuleStates()	getYaw()	getSoftLimited()
getPooperSensor()	setPower()	getAutoPose()	getRoll()	resetTurret()



# Getting Started



# Files and Classes



## Constants

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

```
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 edu.wpi.first.wpilibj.examples.armsbot;
6
7
8 public final class Constants {
9     public static final class DriveConstants {
10         public static final int leftMotorID = 0;
11         public static final int rightMotorID = 1;
12     }
13
14     public static final class ShooterConstants {
15         public static final int motorID = 4;
16     }
17
18     public static final class IntakeConstants {
19         public static final int motorID = 5;
20     }
21 }
```

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



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



# RobotContainer

- Initialize controllers, buttons, and subsystems
- Configure button bindings

```
22 public class RobotContainer {
23
24     //Controller
25     private final Joystick controller = new Joystick(0);
26
27     //Button
28     private final JoystickButton shoot = new JoystickButton(controller, 0);
29
30     //Subsystems
31     private final Shooter s_Shooter = new Shooter();
32     private final Drivetrain s_Drivetrain = new Drivetrain();
33
34     private final AutoCommand m_autoCommand = new AutoCommand(s_Shooter, s_Drivetrain);
35
36
37     public RobotContainer() {
38         s_Drivetrain.setDefaultCommand(new TankDriveCommand(s_Drivetrain, controller));
39         // Configure the button bindings
40         configureButtonBindings();
41     }
42
43     private void configureButtonBindings() {
44
45         shoot.whileHeld(new ShooterControl(s_Shooter, 1));
46
47     }
48
49     /**
50      * Use this to pass the autonomous command to the main {@link Robot} class.
51      *
52      * @return the command to run in autonomous
53      */
54     public Command getAutonomousCommand() {
55         return m_autoCommand;
56     }
57 }
58
```





# 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

```
5 package frc.robot.subsystems;
6
7 import com.ctre.phoenix.motorcontrol.ControlMode;
8 import com.ctre.phoenix.motorcontrol.can.TalonFX;
9
10 import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
11 import edu.wpi.first.wpilibj2.command.SubsystemBase;
12 import frc.robot.Constants;
13
14 public class Shooter extends SubsystemBase {
15
16     public TalonFX masterFx;
17     public TalonFX followerFx;
18
19     public Shooter() {
20         masterFx = new TalonFX(Constants.ShooterConstants.masterID);
21         masterFx.configFactoryDefault();
22         masterFx.setInverted(Constants.ShooterConstants.masterInvert);
23
24         followerFx = new TalonFX(Constants.ShooterConstants.followerID);
25         followerFx.configFactoryDefault();
26         followerFx.setInverted(Constants.ShooterConstants.followerInvert);
27         followerFx.follow(masterFx);
28     }
29
30     public void setPower(double power){
31         masterFx.set(ControlMode.PercentOutput, power);
32     }
33
34     @Override
35     public void periodic() {
36         SmartDashboard.putNumber("Velocity: ", masterFx.getSelectedSensorVelocity());
37     }
38
39 }
40
41
```



# Commands

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

```
5 package frc.robot.commands;
6
7 import frc.robot.subsystems.Shooter;
8 import edu.wpi.first.wpilibj2.command.CommandBase;
9
10 /** An example command that uses an example subsystem. */
11 public class ShooterControl extends CommandBase {
12
13     private final Shooter s_Shooter;
14
15     private double power;
16
17     public ShooterControl(Shooter s_Shooter, double power) {
18         this.s_Shooter = s_Shooter;
19         this.power = power;
20     }
21
22     @Override
23     public void initialize() {
24         addRequirements(s_Shooter);
25     }
26
27     @Override
28     public void execute() {
29         s_Shooter.setPower(power);
30     }
31
32     @Override
33     public void end(boolean interrupted){
34         s_Shooter.setPower(0);
35     }
36
37 }
38
```



# Commands (isFinished)

```
1 package frc.robot.commands;
2
3 import edu.wpi.first.wpilibj2.command.CommandBase;
4 import frc.robot.Constants;
5 import frc.robot.subsystems.Drivetrain;
6
7 public class DriveToDistance extends CommandBase{
8
9     private Drivetrain s_Drivetrain;
10    private double distance;
11
12    public DriveToDistance(Drivetrain s_Drivetrain, double distance){
13        this.s_Drivetrain = s_Drivetrain;
14        this.distance = distance;
15    }
16
17    @Override
18    public void execute() {
19        s_Drivetrain.setDistance(distance);
20    }
21
22
23    @Override
24    public void end(boolean interrupted) {
25        s_Drivetrain.tankDrive(0, 0);
26    }
27
28    @Override
29    public boolean isFinished() {
30        return Math.abs(s_Drivetrain.getLeftDistance() - distance) < s_Drivetrain.metersToFalcon(Constants.DriveConstants.targetDeadband);
31    }
32
33
34 }
35
```



## Default Commands

- Run continuously in teleop
- Set in RobotContainer

```
1 package frc.robot.commands;
2
3 import edu.wpi.first.wpilibj.Joystick;
4 import edu.wpi.first.wpilibj2.command.CommandBase;
5 import frc.robot.subsystems.Drivetrain;
6
7 public class TankDriveCommand extends CommandBase{
8
9     private Drivetrain s_Drivetrain;
10    private Joystick controller;
11
12    public TankDriveCommand(Drivetrain s_Drivetrain, Joystick controller){
13        this.s_Drivetrain = s_Drivetrain;
14        this.controller = controller;
15    }
16
17    @Override
18    public void execute() {
19        double left = controller.getRawAxis(1);
20        double right = controller.getRawAxis(3);
21        s_Drivetrain.tankDrive(left, right);
22    }
23
24 }
25
```

```
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

```
5 package frc.robot.commands;
6
7 import frc.robot.subsystems.Drivetrain;
8 import frc.robot.subsystems.Shooter;
9 import edu.wpi.first.wpilibj2.command.InstantCommand;
10 import edu.wpi.first.wpilibj2.command.SequentialCommandGroup;
11 import edu.wpi.first.wpilibj2.command.WaitCommand;
12
13 public class AutoCommand extends SequentialCommandGroup {
14
15     public AutoCommand(Shooter s_Shooter, Drivetrain s_Drivetrain) {
16
17         addCommands(
18             new InstantCommand(() -> s_Shooter.setPower(1)),
19             new WaitCommand(5),
20             new InstantCommand(() -> s_Shooter.setPower(0)),
21             new DriveToDistance(s_Drivetrain, 15)
22         );
23     }
24
25 }
26
```