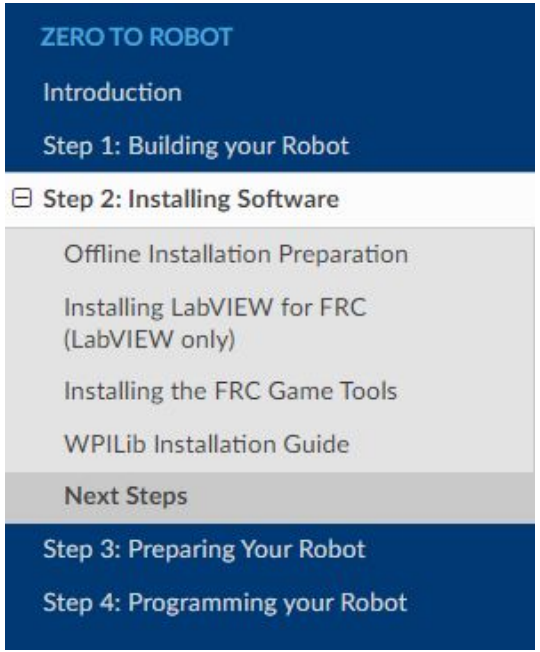
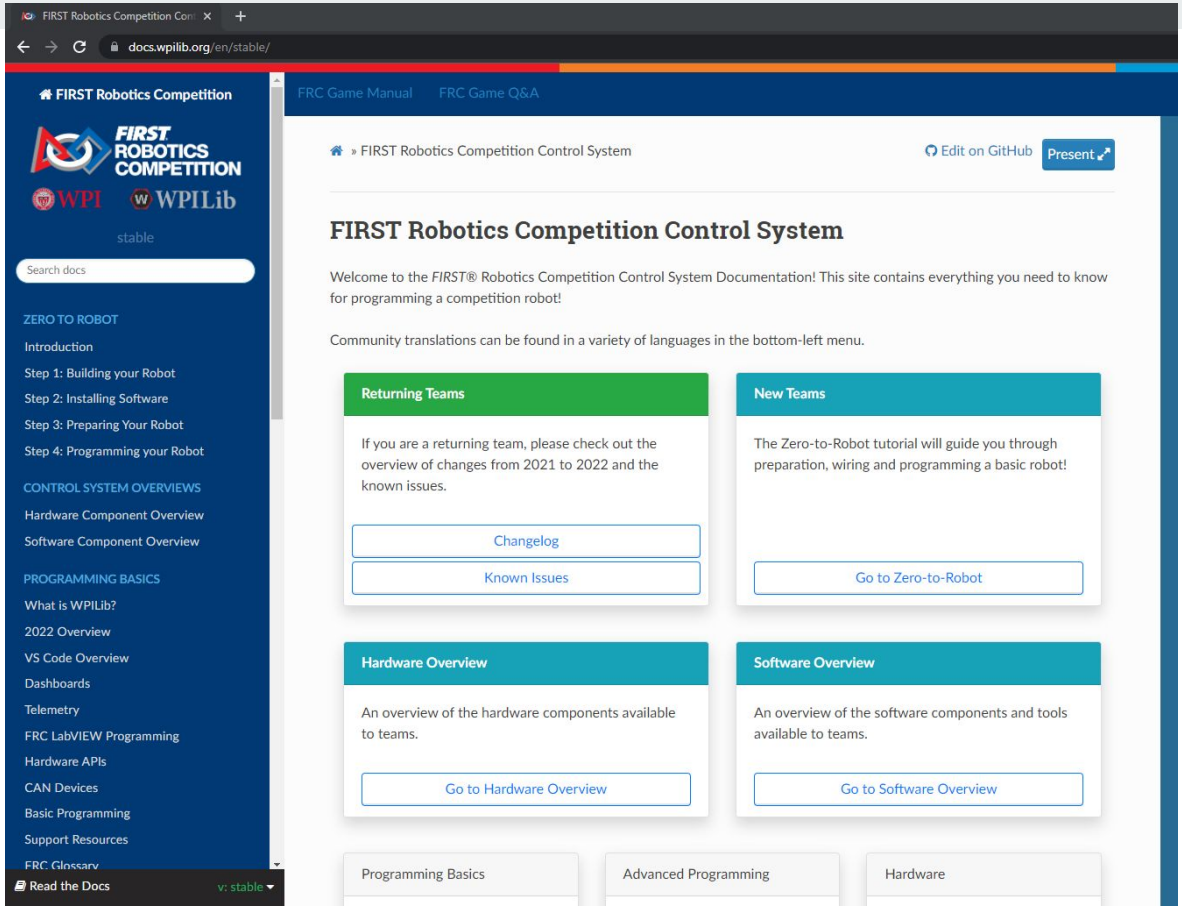
Programming for FRC

Preview and General Information

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

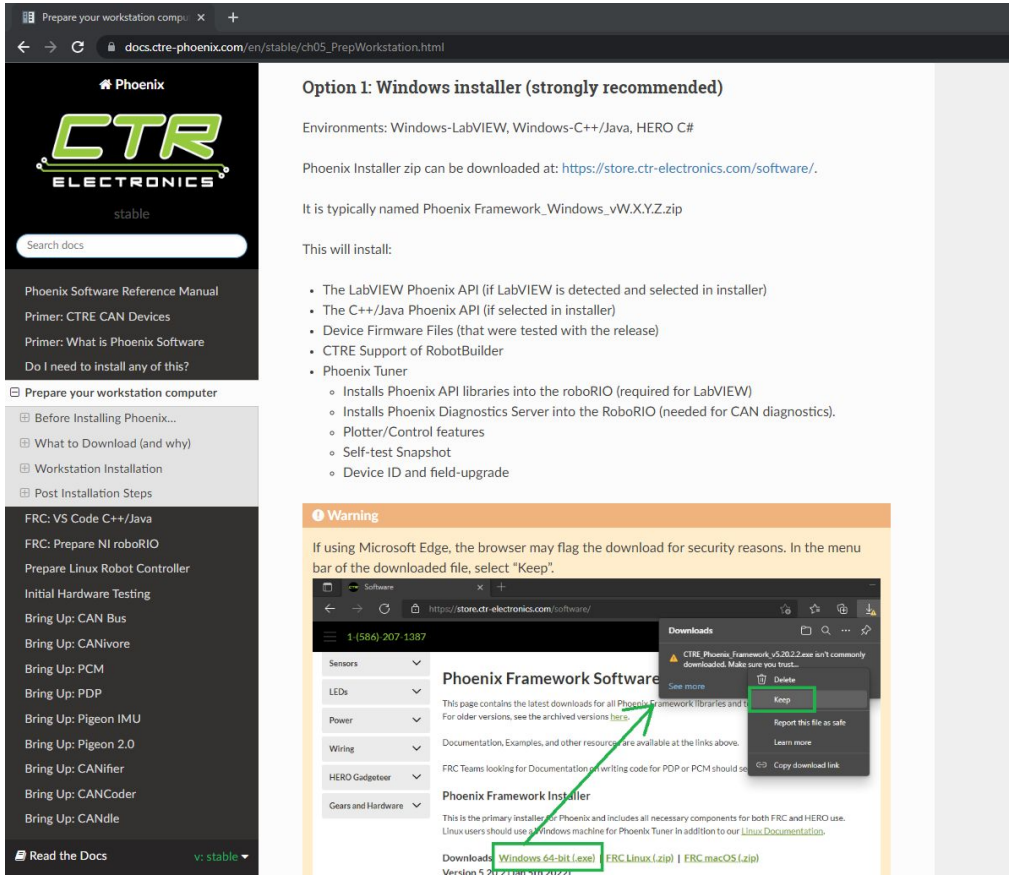
Wpilib docs


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

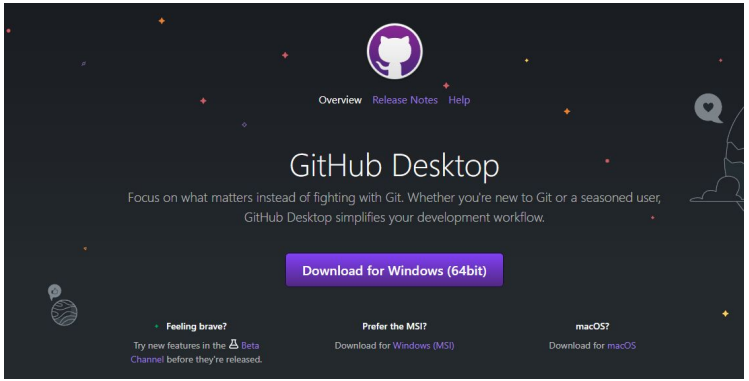

docs.wpilib.org/en/stable/
FIRST Robotics Competition
stable
Search docs
ZERO TO ROBOT
Introduction
Step 1: Building your Robot
Step 2: Installing Software
Step 3: Preparing Your Robot
Step 4: Programming your Robot
CONTROL SYSTEM OVERVIEWS
Hardware Component Overview
Software Component Overview
PROGRAMMING 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

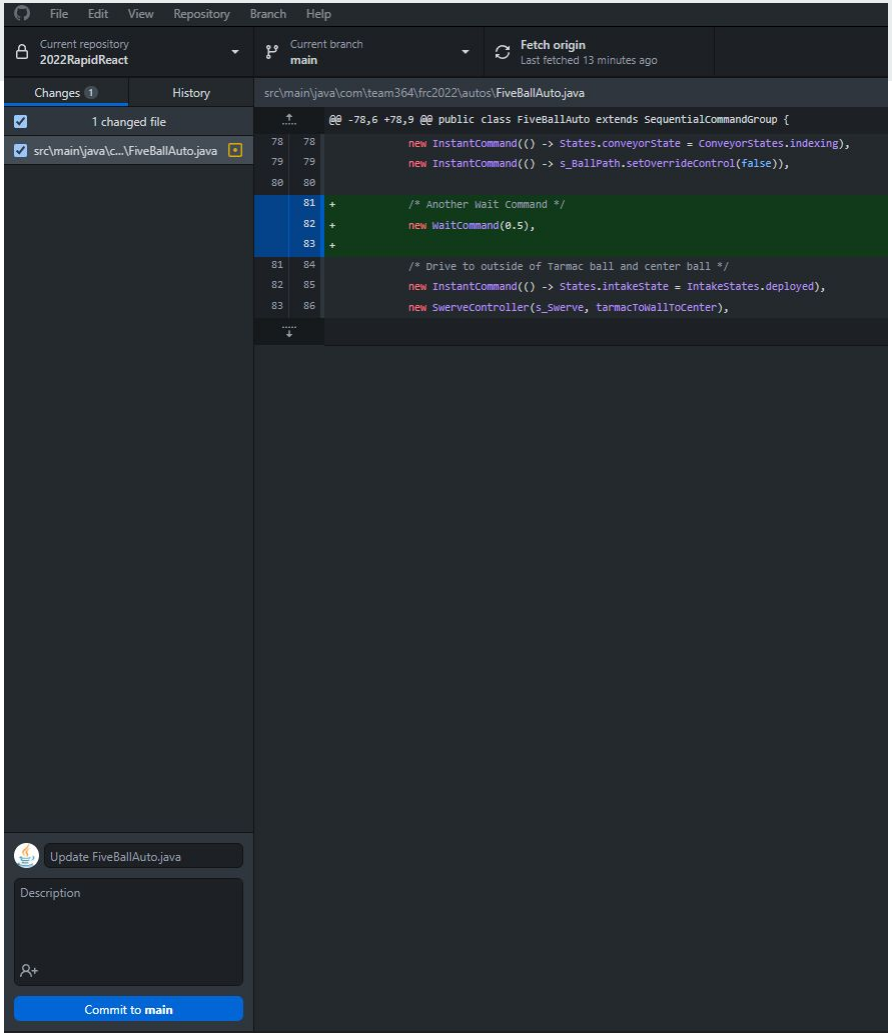
CTR Electronics Documentation

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



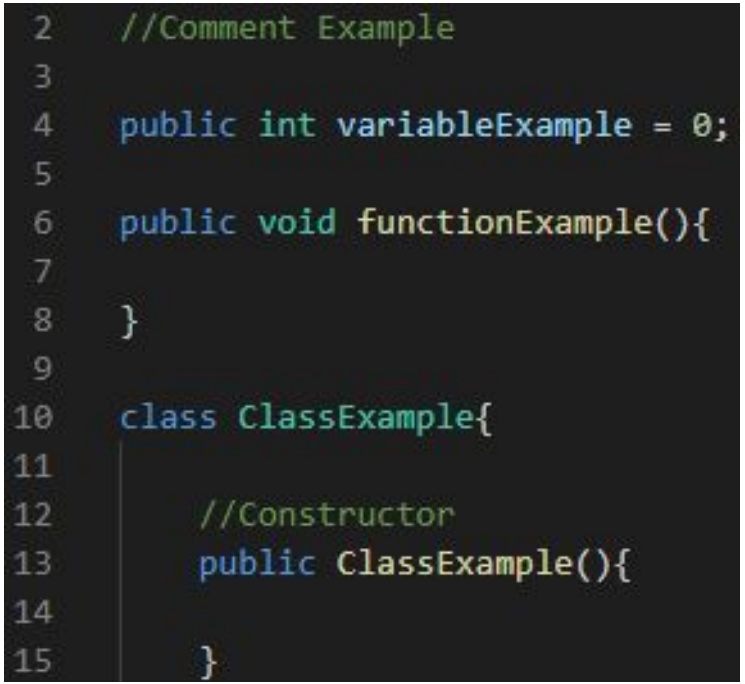
Github Desktop





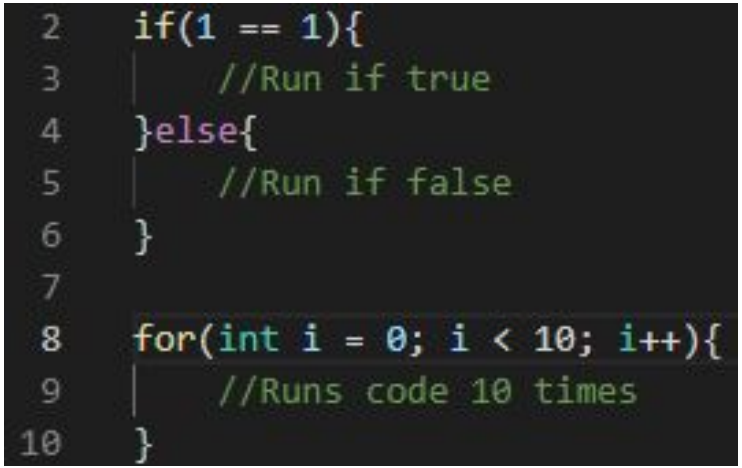
Brief Overview of Programming

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



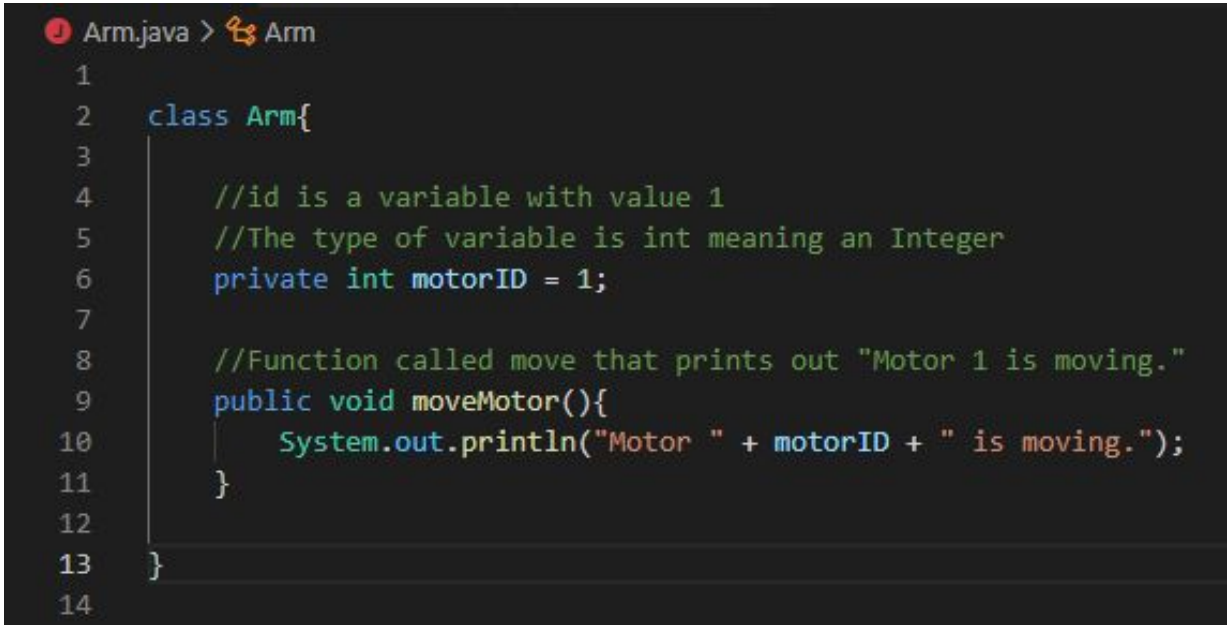
Brief Overview of Programming Logic

- If statements
- If else statements
- For loops



Arm Class

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



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 > ...

```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

Main.java > Main

```java
public class Main {

    Run | Debug
    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

Arm.java
moveMotor()

Robot.java
moveArm()

Main.java
myRobot.moveArm()

Why not just:

Main.java
moveMotor()

Main.java
setOverrideControl()
setLowerConveyorPower()
setUpperConveyorPower()
setPooperPower()
getLowerSensor()
getUpperSensor()
getPooperSensor()

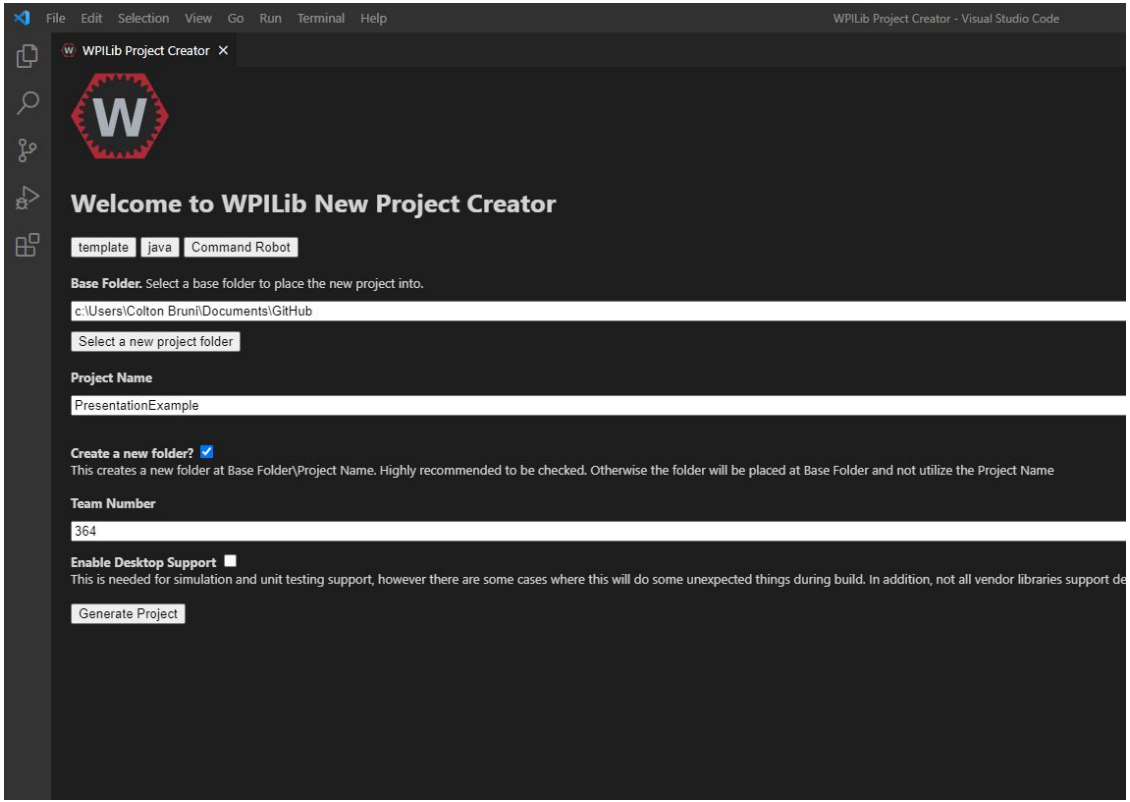
isAllianceBall()
numberOfBalls()
getAngle()
setAngle()
getTargetAngle()
resetHood()
setPower()

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

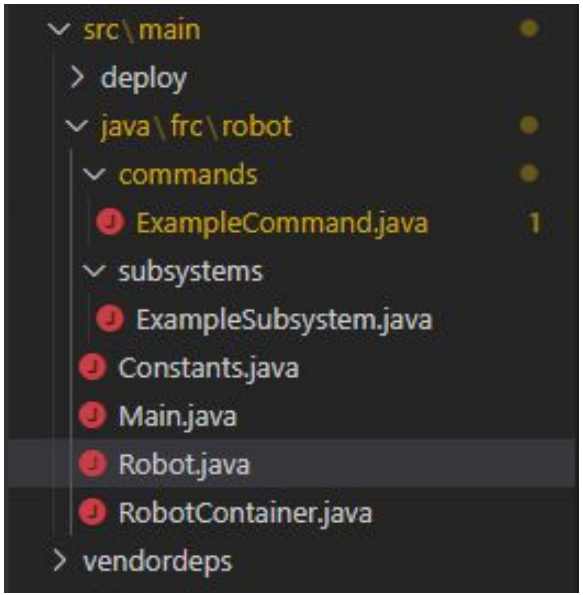
resetOdometry()
getStates()
zeroModules()
zeroGyro()
setGyro()
getYaw()
getRoll()

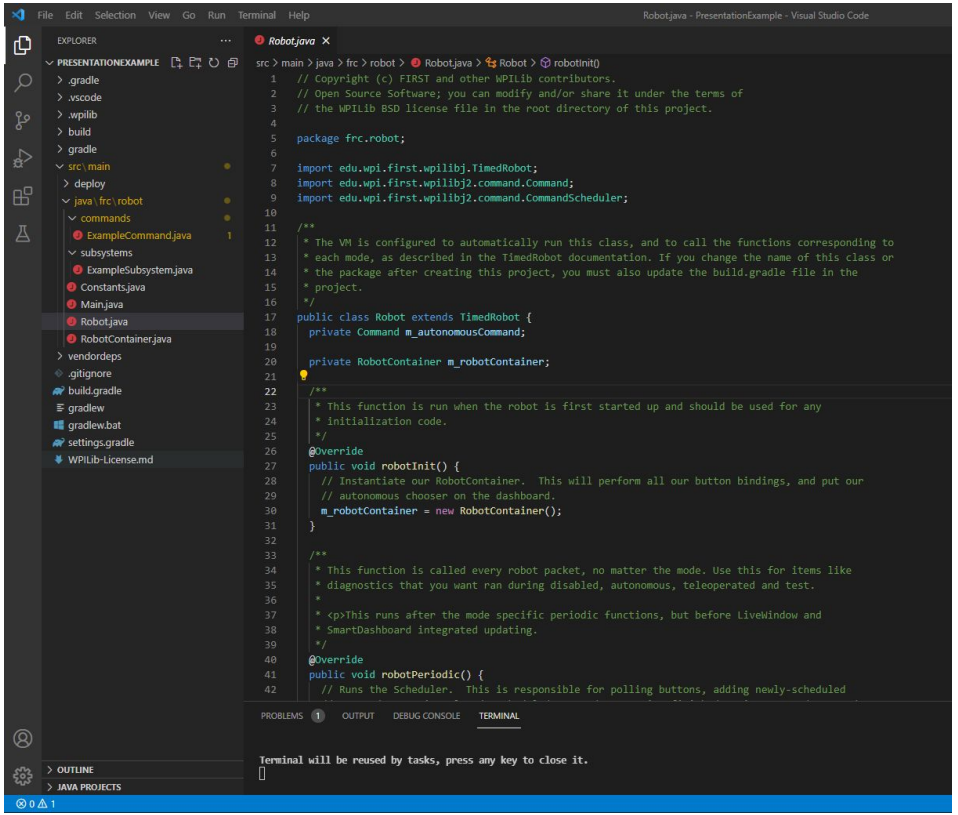
getRobotVelocity()
setPosition()
checkSoftLimits()
targetVisible()
getDegreePosition()
getSoftLimited()
resetTurret()

Getting Started



Files and Classes





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

```
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
```