**Java-WPILib: 00 - Index**

**00-Introduction**: <https://classroom.github.com/a/VuJ9pOyi>

**01-Robot**: <https://classroom.github.com/a/m9QoR6k8>

**02-Subsystem**: <https://classroom.github.com/a/6qQxQqxj>

**03-Commands**: <https://classroom.github.com/a/H8GqICsr>

**04-OI**: <https://classroom.github.com/a/xXLphMLD>

**05-Autonomous**: <https://classroom.github.com/a/xyNcot-J>

**06-Motor**: <https://classroom.github.com/a/Hnq9L7Sw>

**07-Pneumatics**: <https://classroom.github.com/a/j77U4Ruj>

**08-Sensors**: <https://classroom.github.com/a/u6n03I40>

**??-PID**:

**??-Magic**:

**??-Pathfinder**:

**??-Vision**:

If you are looking for the Java-Java course, the first assignment is at the following link:

<https://classroom.github.com/a/DK8GoKs5>

**Note:** It is recommended that you pass one of the two WPILib Entrance Tests before starting this course. A pass is determined by a Mentor of your team. These tests assignments are linked below:

WPILib Entrance Test: <https://classroom.github.com/a/GgILpR-T>

WPILib Entrance Test Version 2: <https://classroom.github.com/a/3KhvpetM>

**Java-WPILib: 00 - Introduction**

**Lesson:**

**Introduction**

In this course we will go over how to program for FRC in Java. This course is meant to go with programming in Java, and therefore assumes a knowledge of the material from that course.

\*\*If you have already installed Visual Studio Code and Git, skip to **FRC Update Suite**\*\*

**Integrated Development Environment (IDE)**

For this workshop we will be using Visual Studio Code as our IDE. An IDE is a program where you can write code, run code, and debug code. They often include features that help make the user a more productive programmer such as auto-completion. To install Visual Studio Code follow the link below and complete the steps listed.

Visual Studio Code Installation Guide: <https://wpilib.screenstepslive.com/s/currentCS/m/java/l/1027503-installing-c-and-java-development-tools-for-frc>

**Git**

**Git** is a version control software. This means that it keeps track of different versions of the files kept inside. This allows you to go back to older versions, branch off to allow multiple people to work on the same project at the same time, and store projects online so they can be accessed anywhere. To install Git on your computer follow the link below and follow the instructions.

Git Install: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

**NOTE: If you are on Windows I recommend choosing the GitHub Desktop Client**

**FRC Update Suite**

In addition to the IDE and Git, to develop software for FRC you need to install the FRC Update Suite. This install includes the Driver Station and other FRC Utilities.

Note: The FRC Update Suite can only be installed on Windows. If you are on another operating system you can still program a robot, you just will not be able to control it. To go around this you can make a Windows Virtual Machine. Instructions for this can be found below:

*Coming Soon!*

**GitHub Classroom**

For this course we will be using GitHub Classroom to provide you with the materials required to complete each worksheet. To sign up for the GitHub Classroom for this class use the link below.

GitHub Classroom Link: <https://classroom.github.com/a/VuJ9pOyi>

By using this link you will be added to the classroom and be given the first assignment **repository** (this document is the first assignment).

**How to Properly Complete this Class**

To succeed in learning the WPILib from this course you should complete each assignment in the way described below.

**Step 1: Clone the Git Repository**

To do this go into your Git terminal (on MacOS and Linux this is the Terminal, on Windows this is the GitHub Desktop Client). In the terminal, navigate to the directory where you would like to put your repository. This is done by using the following command:

cd *DIRECTORY-PATH*

The *DIRECTORY-PATH* is the location in your file system where you want to put the repository. How to do this differs depending on which operating system you are using:

MacOS:

Enter the desired folder. Then click on the settings tool (the gear) on the top of the window. Then select “Copy *FolderName* as Pathname.” This copies the location to your clipboard.

Windows:

Click on the folder name near the top and copy the destination path that appears.

Once you are in the correct directory run the command:

git clone *REPOSITORY-URL*

To get the *REPOSITORY-URL* go to your repository on GitHub and click on “Clone or download” and then copy the URL.

If this is the first time you are using Git it will then prompt you to log-in to your GitHub account.

Once this is done you can use the “cd” command to navigate to the repository directory by using the repository name as the *DIRECTORY-PATH*. Inside this directory is where all your projects for the current worksheet should go. Inside the folder you will also find a copy of the worksheet for that assignment, a folder with all the Example projects, and a folder with the Solutions of the Exercises, as well as any other materials provided to complete it.

**Step 2: Read the Worksheet**

Each worksheet is split up into three sections: the Lesson, Examples, and Exercises. You should first read the Lesson. Then go through the Examples. When going through the examples it may be beneficial to try and run them. They are located in the Examples folder in your repository. Finally, read the Exercises and think through how to solve them.

**Step 3: Complete the Exercises**

To complete the exercises, you will need to make new projects for each exercise. To do this, open Visual Studio Code and hit the settings icon (the gear) on the bottom left of the window. Then click on the “Command Palette…” and notice that a search bar pops up. In this bar search for “WPILib: Create a new Project” and then select it. In the window that pops up, make sure to select **Template**, **Java**, and **Command Robot**.Finally follow the instructions, making sure to place the project in the repository directory.

**Template, Build, Deploy**

Once you have a project you will notice a window on the left. This is you Explorer window, where you can navigate your projects files. Expand the folder “src” and you will see a number of files inside as well as two folders. The folders are the Commands and Subsystems folders. Inside each of them is where your Commands and Subsystems will go. To build the project use the shortcut CONTROL+SHIFT+B (On MacOS replace CONTROL with COMMAND). To deploy your code to a robot you need to be connected to a robot. This is either via Wifi, Ethernet, or USB. To deploy, open the Command Palette and search for “WPILib: Deploy Robot Code.”

**Commit Your Code to Git**

When you are done with your project you should upload it to GitHub. To do this open a terminal window in Visual Studio Code by selecting “Terminal” at the top and then “New Terminal.” This terminal will open on the bottom of the window and is already in your project directory. To update your git repository and add or update your files, follow the list of commands below in **bold**.

**git add -A**

This command adds all modified files to be staged to be committed

**git commit -m “*YOUR-COMMIT-MESSAGE*”**

This command commits the staged files

**git push**

This command pushes all new commits to GitHub

*YOUR-COMMIT-MESSAGE* should be replaced with the message you want to put with your current commit.

After doing this, if you go back to your GitHub page online you should see that the repository has been updated to include your new project.

**Step 4: Check your Work**

Once you have completed the exercises you should check your work against the provided Solutions. Your code does not need to match exactly to the solution, but it will likely be similar. It is important to not look at the solutions until you have completed the exercise. The goal of this course is to learn Java, not learn how to copy Java code. If you cheat and look at the solution before completing the exercises you are only harming yourself.

**Step 5: Go on to the next Assignment**

The link to the next assignment on GitHub classroom is located at the bottom of each assignment worksheet.

**Step 6: HAVE FUN!**

Remember, the WPILib is a large library with a huge amount of outside knowledge required to fully understand each part. It will take time to learn how to properly use it. So just remember, go at your own pace, make sure you understand everything before going on, and if you ever need help make sure to ask questions to your peers, mentors, teachers, dog, cat, robot offender, Chief Delphi, or the always helpful Google.

**Examples:**

Here is an example of a template Robot class

**Template**

/\*----------------------------------------------------------------------------\*/

/\* Copyright (c) 2017-2018 FIRST. All Rights Reserved.                        \*/

/\* Open Source Software - may be modified and shared by FRC teams. The code   \*/

/\* must be accompanied by the FIRST BSD license file in the root directory of \*/

/\* the project.                                                               \*/

/\*----------------------------------------------------------------------------\*/

package frc.robot;

import edu.wpi.first.wpilibj.TimedRobot;

import edu.wpi.first.wpilibj.command.Command;

import edu.wpi.first.wpilibj.command.Scheduler;

import edu.wpi.first.wpilibj.smartdashboard.SendableChooser;

import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;

import frc.robot.commands.ExampleCommand;

import frc.robot.subsystems.ExampleSubsystem;

/\*\*

 \* 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 {

  public static ExampleSubsystem m\_subsystem = new ExampleSubsystem();

  public static OI m\_oi;

  Command m\_autonomousCommand;

  SendableChooser<Command> m\_chooser = new SendableChooser<>();

  /\*\*

   \* This function is run when the robot is first started up and should be

   \* used for any initialization code.

   \*/

  @Override

  public void robotInit() {

    m\_oi = new OI();

    m\_chooser.setDefaultOption("Default Auto", new ExampleCommand());

    // chooser.addOption("My Auto", new MyAutoCommand());

    SmartDashboard.putData("Auto mode", m\_chooser);

  }

  /\*\*

   \* 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() {

  }

  /\*\*

   \* This function is called once each time the robot enters Disabled mode.

   \* You can use it to reset any subsystem information you want to clear when

   \* the robot is disabled.

   \*/

  @Override

  public void disabledInit() {

  }

  @Override

  public void disabledPeriodic() {

    Scheduler.getInstance().run();

  }

  /\*\*

   \* This autonomous (along with the chooser code above) shows how to select

   \* between different autonomous modes using the dashboard. The sendable

   \* chooser code works with the Java SmartDashboard. If you prefer the

   \* LabVIEW Dashboard, remove all of the chooser code and uncomment the

   \* getString code to get the auto name from the text box below the Gyro

   \*

   \* <p>You can add additional auto modes by adding additional commands to the

   \* chooser code above (like the commented example) or additional comparisons

   \* to the switch structure below with additional strings & commands.

   \*/

  @Override

  public void autonomousInit() {

    m\_autonomousCommand = m\_chooser.getSelected();

    /\*

     \* String autoSelected = SmartDashboard.getString("Auto Selector",

     \* "Default"); switch(autoSelected) { case "My Auto": autonomousCommand

     \* = new MyAutoCommand(); break; case "Default Auto": default:

     \* autonomousCommand = new ExampleCommand(); break; }

     \*/

    // schedule the autonomous command (example)

    if (m\_autonomousCommand != null) {

      m\_autonomousCommand.start();

    }

  }

  /\*\*

   \* This function is called periodically during autonomous.

   \*/

  @Override

  public void autonomousPeriodic() {

    Scheduler.getInstance().run();

  }

  @Override

  public void teleopInit() {

    // This makes sure that the autonomous stops running when

    // teleop starts running. If you want the autonomous to

    // continue until interrupted by another command, remove

    // this line or comment it out.

    if (m\_autonomousCommand != null) {

      m\_autonomousCommand.cancel();

    }

  }

  /\*\*

   \* This function is called periodically during operator control.

   \*/

  @Override

  public void teleopPeriodic() {

    Scheduler.getInstance().run();

  }

  /\*\*

   \* This function is called periodically during test mode.

   \*/

  @Override

  public void testPeriodic() {

  }

}

**Exercises:**

1. Create a new project

**Next Assignment Link:** <https://classroom.github.com/a/m9QoR6k8>