**Java - WPILib: 02 - Subsystem**

**Lesson:**

A Subsystem is a component of a robot the completes a single function. Some examples of Subsystems would be a Drivetrain, Arm, Elevator, or Intake. Subsystems can have many different components inside them including motors, pneumatics, sensors, or servos. A Subsystem may be reliant on another Subsystem, such as an intake on an elevator.

In code, a Subsystem will be where all the component objects of the Subsystem are created and used. You should not use any component outside directly outside of a Subsystem (this we be explained further in Commands). A Subsystem is structured similarly to any ordinary Java Class. There is a Field, where all you component members are declared, a Constructor, where component members are initialized, and Methods, which control and use the component members to control the physical Subsystem on the Robot.

Subsystem files are located in the Subsystems Package inside source. One thing you will notice is that each of your Subsystems are a child class of the Subsystem class. This allows for all of your Subsystems to act in similar ways to each other as defined in the Subsystem class (this is simple Inheritance, if this is confusing you may want to go back to the Java Course and refresh your knowledge). To create a new Subsystem, right click on the Subsystem folder and select “Create a new class/command,” then select Subsystem from the drop down.

**Note:** When setting the state of different components it will make your life way easier if you try to only actually set the state in one place within the Subsystem. This is not required but will definitely make your Subsystem simpler to debug. Also, from now on you need to make sure you have the CTRE Pheonix library installed and added to the project. Follow this links below containing install and project setup instructions.

CTRE Phoenix Install:

<https://phoenix-documentation.readthedocs.io/en/latest/ch05_PrepWorkstation.html>

CTRE Phoenix Project Setup:

<https://phoenix-documentation.readthedocs.io/en/latest/ch05a_CppJava.html>

**Examples:**

**MyRobot – MySubsystem.java**

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

/\* Copyright (c) 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.subsystems;

import com.ctre.phoenix.motorcontrol.ControlMode;

import com.ctre.phoenix.motorcontrol.can.TalonSRX;

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

import frc.robot.RobotMap;

/\*

This is a Subsystem with one TalonSRX motor controller.

\*/

public class MySubsystem extends Subsystem {

  //Declare all components of Subsystem

  TalonSRX myMotor;

  //In Constructor, Initialize all components of Subsystem

  public MySubsystem() {

    //Initialize Talon to CAN Address.

    //It is good practice to use constants located in RobotMap

    myMotor = new TalonSRX(RobotMap.myMotorCanID);

  }

  //When this method is called the motor is run at full speed (1)

  public void RunMotorFullSpeed() {

    myMotor.set(ControlMode.PercentOutput, 1);

  }

  //When this method is called the motor is run at half speed (.5)

  public void RunMotorHalfSpeed() {

    myMotor.set(ControlMode.PercentOutput, .5);

  }

  //When this method is called the motor is stopped (0)

  public void StopMotor() {

    myMotor.set(ControlMode.PercentOutput, 0);

  }

  //When this method is called the motor is run at the input speed

  public void RunMotorAtSpeed(double speed) {

    myMotor.set(ControlMode.PercentOutput, speed);

  }

  @Override

  public void initDefaultCommand() {

    // Set the default command for a subsystem here.

    // setDefaultCommand(new MySpecialCommand());

    //We will cover this when we get to Commands

  }

}

**MyRobot – Robot.java**

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

/\* 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;

//Imports all of the Subsystems in the Subsystems package

import frc.robot.subsystems.\*;

/\*\*

 \* 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();

  //Create an instance of MySubsystem

  public static MySubsystem m\_mySubsystem = new MySubsystem();

  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() {

  }

}

**MyRobot – RobotMap.java**

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

/\* 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;

/\*\*

 \* The RobotMap is a mapping from the ports sensors and actuators are wired into

 \* to a variable name. This provides flexibility changing wiring, makes checking

 \* the wiring easier and significantly reduces the number of magic numbers

 \* floating around.

 \*/

public class RobotMap {

  // For example to map the left and right motors, you could define the

  // following variables to use with your drivetrain subsystem.

  // public static int leftMotor = 1;

  // public static int rightMotor = 2;

  // If you are using multiple modules, make sure to define both the port

  // number and the module. For example you with a rangefinder:

  // public static int rangefinderPort = 1;

  // public static int rangefinderModule = 1;

  //The CAN Address of TalonSRX myMotor

  public static final int myMotorCanID = 1;

}

**Note:** Try to build your code. To do this open the command palette and select “WPILib: Build Robot Code.” Also, make sure you don’t forget the imports.

**Exercises:**

1. Create a Robot project with a Subsystem that contains two TalonSRX motor controllers. There should be a method to control each TalonSRX independently with two given inputs and a method that controls each TalonSRX together with the same given input.

**Next Assignment Link:** <https://classroom.github.com/a/H8GqICsr>