**Java - WPILib: 04 - OI**

**Lesson:**

The OI (Operator Interface) is where commands are linked to actions on the Joystick. Usually this is done with just Joystick Buttons, since the other controls on a Joystick are easier to implement in other ways, but can also use any other form of controller.

**Button**

There are three general types of Button Events: whenPressed, whileHeld, and whenReleased. Each of these are as the name describes. The whenPressed method of a button takes in a command and will add that command to the scheduler when the Button instance is pressed and run it until it isFinished. WhileHeld will add the command to the scheduler when the Button instance is held and then interrupt the command when the Button is released. WhenReleased will add the command to the scheduler when the Button is released and run it until it isFinished.

The command passed in can be a single command or a command group, since command groups act as if they are one command. One thing to remember is it is good practice to only assign one command to each Button Event.

**Examples:**

**MyRobot - OI**

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

/\* 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.Joystick;

import edu.wpi.first.wpilibj.buttons.JoystickButton;

import frc.robot.commands.MyCommand;

/\*\*

\* This class is the glue that binds the controls on the physical operator

\* interface to the commands and command groups that allow control of the robot.

\*/

public class OI {

//// CREATING BUTTONS

// One type of button is a joystick button which is any button on a

//// joystick.

// You create one by telling it which joystick it's on and which button

// number it is.

// Joystick stick = new Joystick(port);

// Button button = new JoystickButton(stick, buttonNumber);

// There are a few additional built in buttons you can use. Additionally,

// by subclassing Button you can create custom triggers and bind those to

// commands the same as any other Button.

//// TRIGGERING COMMANDS WITH BUTTONS

// Once you have a button, it's trivial to bind it to a button in one of

// three ways:

// Start the command when the button is pressed and let it run the command

// until it is finished as determined by it's isFinished method.

// button.whenPressed(new ExampleCommand());

// Run the command while the button is being held down and interrupt it once

// the button is released.

// button.whileHeld(new ExampleCommand());

// Start the command when the button is released and let it run the command

// until it is finished as determined by it's isFinished method.

// button.whenReleased(new ExampleCommand());

//Declare Buttons

public Joystick m\_joystick; //This is public for reasons we will get to later

private JoystickButton m\_runMotorButton;

private JoystickButton m\_stopMotorButton;

//Initialize Button to Command

public OI() {

m\_joystick = new Joystick(RobotMap.joystickUSBID);

m\_runMotorButton = new JoystickButton(m\_joystick, 1); //Button A

m\_stopMotorButton = new JoystickButton(m\_joystick, 2); //Button B

//Button A will cause motor to run, Button B will cause motor to stop

m\_runMotorButton.whenPressed(new MyCommand(1));

m\_stopMotorButton.whenPressed(new MyCommand(0));

}

}

**MyRobot – Robot**

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

/\* 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.MyCommand;

import frc.robot.subsystems.MySubsystem;

/\*\*

\* 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 MySubsystem m\_subsystem = 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 MyCommand(0));

// 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**

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

/\* 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 {

//The CAN ID of TalonSRX motor controllers

public static final int motorCanID = 0;

public static final int joystickUSBID = 0;

}

**MyRobot – MySubsystem**

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

/\* 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.subsystems;

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

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

import com.ctre.phoenix.motorcontrol.ControlMode;

import frc.robot.RobotMap;

/\*\*

\* MySubsystem with 1 TalonSRX motor controller

\*/

public class MySubsystem extends Subsystem {

//Declare Components

private TalonSRX motor;

//Initialize Components

public MySubsystem() {

motor = new TalonSRX(RobotMap.motorCanID);

}

//Run the motor at speed

public void runMotor(double speed) {

motor.set(ControlMode.PercentOutput, speed);

}

@Override

public void initDefaultCommand() {

// Set the default command for a subsystem here.

// setDefaultCommand(new MySpecialCommand());

}

}

**MyRobot – MyCommand**

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

/\* 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.commands;

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

import frc.robot.Robot;

/\*\*

\* An example command. You can replace me with your own command.

\*/

public class MyCommand extends Command {

private double m\_speed;

public MyCommand(double speed) {

// Use requires() here to declare subsystem dependencies

requires(Robot.m\_subsystem);

m\_speed = speed;

}

// Called just before this Command runs the first time

@Override

protected void initialize() {

}

// Called repeatedly when this Command is scheduled to run

@Override

protected void execute() {

Robot.m\_subsystem.runMotor(m\_speed);

}

// Make this return true when this Command no longer needs to run execute()

@Override

protected boolean isFinished() {

return true;

}

// Called once after isFinished returns true

@Override

protected void end() {

}

// Called when another command which requires one or more of the same

// subsystems is scheduled to run

@Override

protected void interrupted() {

}

}

**Exercises:**

1. Create a Robot with a Subsystem containing one TalonSRX and a method to control it through a variable, one Command to run the Subsystem at a input speed, and then map a joystick button to run the motor at full speed whenPressed, a different button to run the motor at half speed whileHeld and to stop whenReleased.

**Next Assignment Link:** <https://classroom.github.com/a/xyNcot-J>