**Java - WPILib: 03 - Commands**

**Lesson:**

Commands are the way you tell the Robot to do something in the Command Based model (That is why it is in the name). A Command can be simple and only control one subsystem or it can be very complicated and control multiple subsystems working together to complete a task. Commands are run in a few ways. They can be linked to a button on a joystick, set as the default command for a subsystem (This means that the command is always running for that Subsystem during TeleOp), or it can be added directly to the command scheduler.

Just like with Subsystems, Commands are child classes of the Command class. This means they all run in a similar way to each other. To create a new Command simply right click on the Command folder, select “Create a new class/command,” and select Command from the drop down menu.

Commands are made up of a few parts. First there is the Constructer. This is where you will get the Subsystems you need access to from the Robot class. To do this use a require statement. Next, there is the Initialization. This is where you initialize any member variable for the Command, set the states of any state machines, or set call methods from the Subsystem to set the components to the correct state. The Execute method is where the code you want to run for the Command is placed. This can be as simple as running a motor at a constant speed or extremely complicated with state machines and PID control of the components. The Execute method will run once each clock cycle until IsFinished returns TRUE. The IsFinished method is where you set the conditions for when the Command should finish. This should produce a Boolean value of TRUE when you want the Command to End. Finally, there are the End and Interrupt methods. The End method is where you put the code for what to do at the End of the Command and the Interrupt method is where you put code for if the Command is interrupted.

**CommandGroup**

A CommandGroup is essentially a list of commands. You can make multiple Commands run in series, in parallel, or a combination of the two. You can also set time delays between Commands. CommandGroups are used the same way as any other Command. To create a CommandGroup use the same process as creating a Command but select CommandGroup instead.

**Examples:**

**MyRobot – MySubsystem**

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

/\* 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;

import frc.robot.commands.\*;

/\*\*

 \* This Subsystem has two TalonSRX motor controller

 \*/

public class MySubsystem extends Subsystem {

  // Put methods for controlling this subsystem

  // here. Call these from Commands.

  //Declare Components

  TalonSRX motor1, motor2;

  //Constructor

  public MySubsystem() {

    //Initialize Components

    motor1 = new TalonSRX(RobotMap.motor1CanID);

    motor2 = new TalonSRX(RobotMap.motor2CanID);

  }

  public void RunMotor1(double speed) {

    motor1.set(ControlMode.PercentOutput, speed);

  }

  public void RunMotor2(double speed) {

    motor2.set(ControlMode.PercentOutput, speed);

  }

  @Override

  public void initDefaultCommand() {

    // Set the default command for a subsystem here.

    setDefaultCommand(new StopMotors());

  }

}

**MyRobot – RunMotor1HalfSpeed**

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

/\* 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.commands;

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

import frc.robot.Robot;

public class RunMotor1HalfSpeed extends Command {

  public RunMotor1HalfSpeed() {

    // Use requires() here to declare subsystem dependencies

    // eg. requires(chassis);

    requires(Robot.m\_mySubsystem);

  }

  // 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\_mySubsystem.RunMotor1(.5);

  }

  // 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() {

    Robot.m\_mySubsystem.RunMotor1(0);

  }

}

**MyRobot – RunMotor2FullSpeed**

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

/\* 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.commands;

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

import frc.robot.Robot;

public class RunMotor2FullSpeed extends Command {

  public RunMotor2FullSpeed() {

    // Use requires() here to declare subsystem dependencies

    // eg. requires(chassis);

    requires(Robot.m\_mySubsystem);

  }

  // 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\_mySubsystem.RunMotor2(1);

  }

  // 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() {

    Robot.m\_mySubsystem.RunMotor2(0);

  }

}

**MyRobot – StopRobot**

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

/\* 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.commands;

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

import frc.robot.Robot;

public class StopMotors extends Command {

  public StopMotors() {

    // Use requires() here to declare subsystem dependencies

    // eg. requires(chassis);

    requires(Robot.m\_mySubsystem);

  }

  // 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\_mySubsystem.RunMotor1(0);

    Robot.m\_mySubsystem.RunMotor2(0);

  }

  // 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() {

  }

}

**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.\*;

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 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 MyCommandGroup());

    // 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 {

  // 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;

  //CAN IDs for TalonSRX

  public static final int motor1CanID = 1;

  public static final int motor2CanID = 2;

}

**MyRobot – MyCommandGroup**

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

/\* 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.commands;

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

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

public class MyCommandGroup extends CommandGroup {

  /\*\*

   \* Add your docs here.

   \*/

  public MyCommandGroup() {

    // Add Commands here:

    // e.g. addSequential(new Command1());

    // addSequential(new Command2());

    // these will run in order.

    // To run multiple commands at the same time,

    // use addParallel()

    // e.g. addParallel(new Command1());

    // addSequential(new Command2());

    // Command1 and Command2 will run in parallel.

    // A command group will require all of the subsystems that each member

    // would require.

    // e.g. if Command1 requires chassis, and Command2 requires arm,

    // a CommandGroup containing them would require both the chassis and the

    // arm.

    //RunMotor1HalfSpeed and RunMotor2FullSpeed will run at the same time

    addParallel(new RunMotor1HalfSpeed());

    addSequential(new RunMotor2FullSpeed());

    //Puts a Time delay between Commands

    addSequential(new WaitCommand(100));

    //StopMotors will run after the previous finish

    addSequential(new StopMotors());

  }

}

**Exercises:**

1. Create a new Robot project. Create a Subsystem with three TalonSRX motors, each with a method to control their speed. Then create three commands, one to control the speed of each motor based on a Command Argument. Then, create a Command Group that runs the two of the motors at different speeds for 20 seconds, stops them, then runs the third motor for 10 seconds, finally stop the motor. Make the Command Group the default autonomous command and then try your code on a robot (Deploy and Run instructions are in the OI assignment).

**Note:** DO NOT RUN CODE ON ROBOT UNLESS AN EXPERIENCED PERSON HAS CHECKED THE CODE AND ROBOT CORRECTNESS.

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