FTC JAVA PROGRAMMING BASICS

FRUITPORT TECHNO TROJANS FTC WORKSHOP

FTC JAVA PROGRAMMING BASICS

FRUITPORT TECHNO TROJANS FTC WORKSHOP

PRESENTED BY:

Lawrence Welty

FTC JAVA PROGRAMMING BASICS

FRUITPORT TECHNO TROJANS FTC WORKSHOP

PRESENTED BY:

Lawrence Welty

Imperial Dade - Current

- Senior Data Engineer (current)
  - Enterprise Data Warehouse
    - Bringing business data from 70+ branches together
    - Using Azure Cloud Services and Resources

Nichols – a division of Imperial Dade

- Senior System Analyst (3 years)
  - System Integration, Design, Architecture
  - Microsoft 365

FTC JAVA PROGRAMMING BASICS

FRUITPORT TECHNO TROJANS FTC WORKSHOP

PRESENTED BY:

Lawrence Welty

Imperial Dade - Current

- Senior Data Engineer (current)
  - Enterprise Data Warehouse
    - Bringing business data from 70+ branches together
    - Using Azure Cloud Services and Resources

Nichols – a division of Imperial Dade

- Senior System Analyst (3 years)
  - System Integration, Design, Architecture
  - Microsoft 365

Holland Freight – YRC Freight (Yellow / YRCW)

- Senior System Analyst (11 years)

FTC JAVA PROGRAMMING BASICS

FRUITPORT TECHNO TROJANS FTC WORKSHOP

PRESENTED BY:

Lawrence Welty

Imperial Dade - Current

- Senior Data Engineer (current)
  - Enterprise Data Warehouse
    - Bringing business data from 70+ branches together
    - Using Azure Cloud Services and Resources

Nichols – a division of Imperial Dade

- Senior System Analyst (3 years)
  - System Integration, Design, Architecture
  - Microsoft 365

Holland Freight – YRC Freight (Yellow / YRCW)

- Senior System Analyst (11 years)

Herman Miller

- Mainframe Database Administrator (4 years)
- Application Developer / System Analyst (7 years)

FTC JAVA PROGRAMMING BASICS

FRUITPORT TECHNO TROJANS FTC WORKSHOP

PRESENTED BY:

Lawrence Welty

FIRST Robotics Competition - FRC

- 4th Year as Programming Mentor/Coach

Advancements:

- Migrate from LabView to Java Programming
- Convert drivetrain from mecanum drive to swerve drive

- Event Volunteer Roles:
  - Control System Analyst – CSA

FTC JAVA PROGRAMMING BASICS

FRUITPORT TECHNO TROJANS FTC WORKSHOP

PRESENTED BY:

Lawrence Welty

FIRST ROBOTICS COMPETITION

WPI WPILib

FIRST TECH CHALLENGE

FIRST Robotics Competition - FRC

- 4th Year as Programming Mentor/Coach

  Advancements:

  - Migrate from LabView to Java Programming
  - Convert drivetrain from mecanum drive to swerve drive

- Event Volunteer Roles:
  - Control System Analyst – CSA

FIRST Tech Challenge - FTC

- 2nd Year as Programming Mentor/Coach

  Advancements:

  - Migrate from Blocks to Java Programming with Green team
  - Incorporate field-centric control on mecanum drivetrain

- Event Volunteer Roles:
  - Control System Analyst – CSA
  - Scorekeeper

FTC JAVA PROGRAMMING BASICS

FRUITPORT TECHNO TROJANS FTC WORKSHOP

PRESENTED BY:

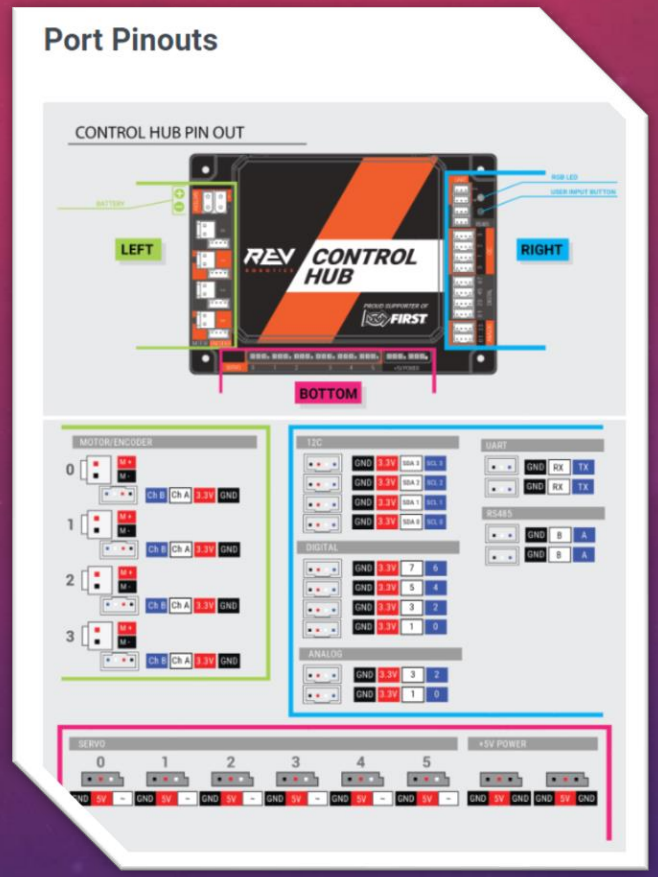
Lawrence Welty

Agenda:

- Robot Basics
- Programming Tools
- Java Framework – FtcRobotController
- OpMode Structure – Teleop
- OpMode Structure – Autonomous
- Resources
- Questions

ROBOT BASICS

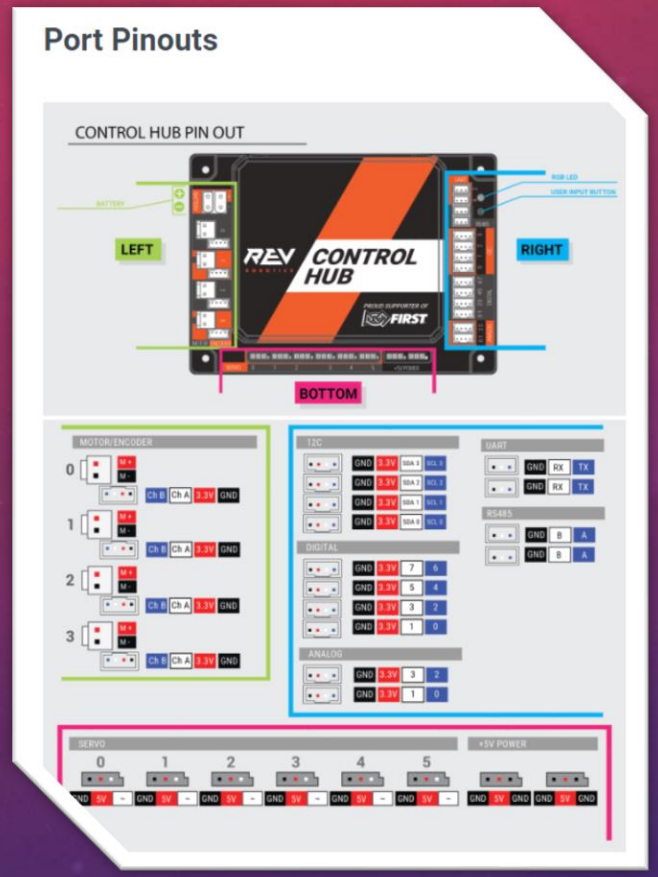
Rev Robotics Control Hub



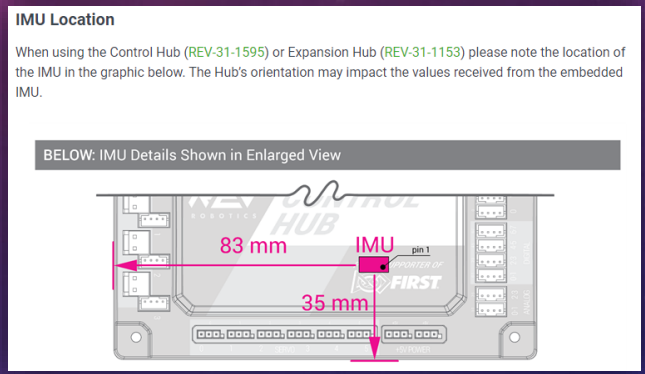
FIRST TECH CHALLENGE

ROBOT BASICS

Rev Robotics Control Hub



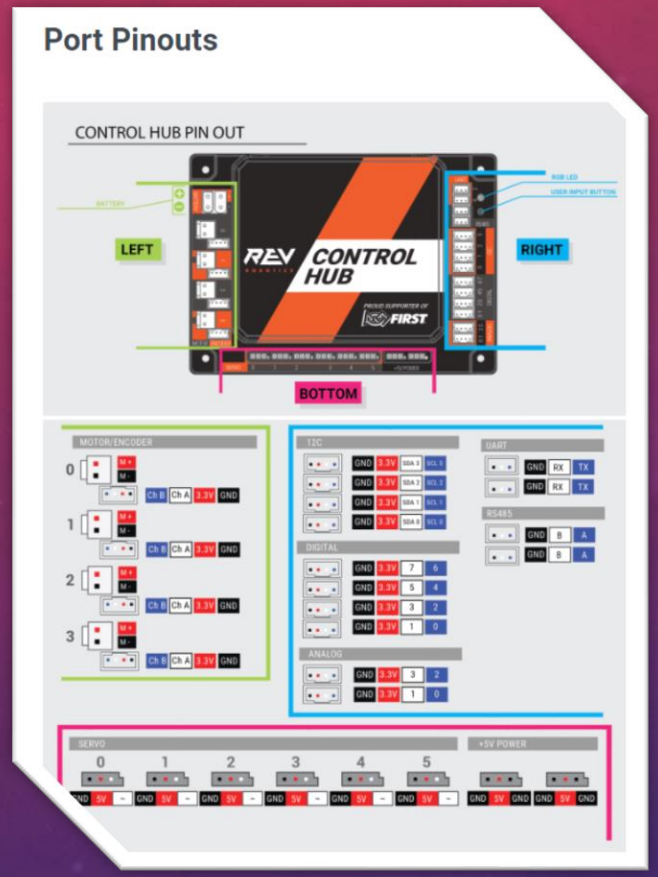
IMU – Inertial Measurement Unit



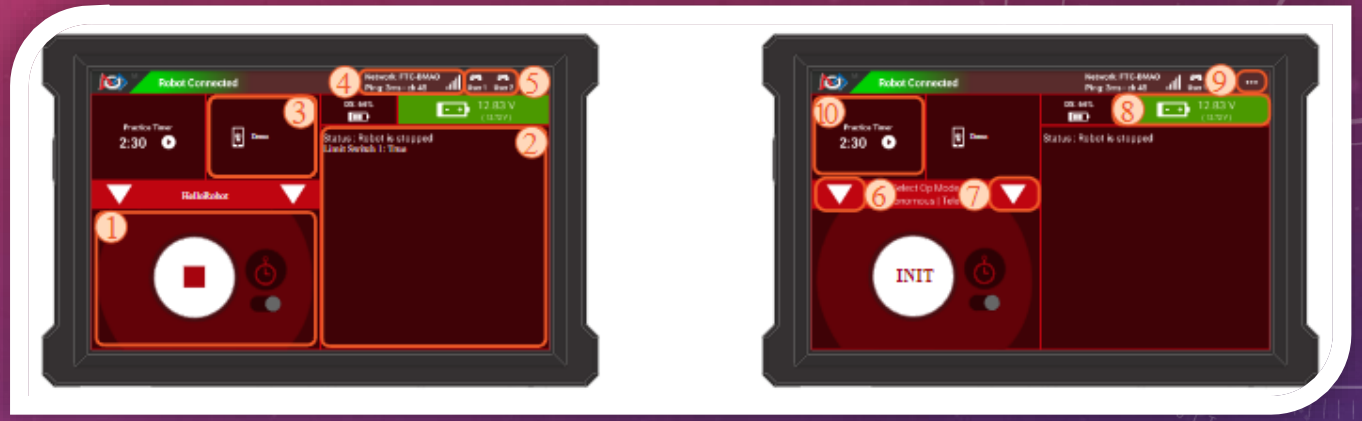
FIRST TECH CHALLENGE

ROBOT BASICS

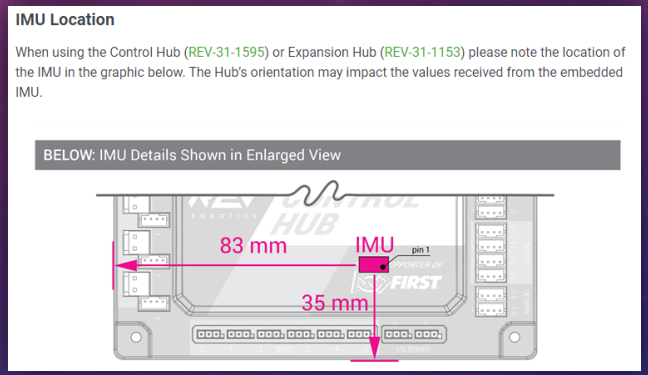
Rev Robotics Control Hub



Rev Robotics Driver Hub



IMU – Inertial Measurement Unit



FIRST TECH CHALLENGE

PROGRAMMING TOOLS

Required Software / Tools

Development Environment

Android Studio

FTCRobotController – repository on Github

FIRST TECH CHALLENGE

PROGRAMMING TOOLS

Required Software / Tools

Development Environment

Android Studio

FTCRobotController – repository on Github

Optional Software / Tools

Source Code Management

Git-Scm

Github Desktop Client

FIRST TECH CHALLENGE

PROGRAMMING TOOLS

Required Software / Tools

Development Environment

Android Studio

FTCRobotController – repository on Github

Optional Software / Tools

Source Code Management

Git-Scm

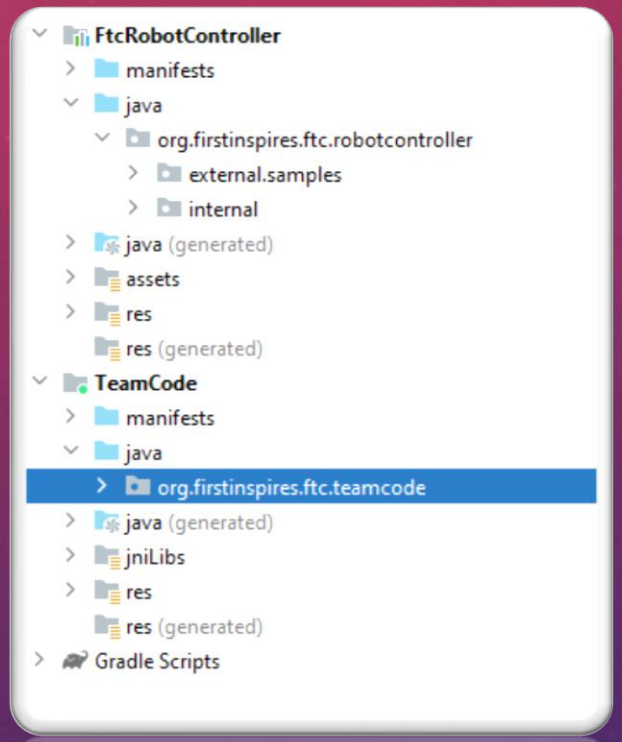
Github Desktop Client

Other (honorable mention)

FIRST TECH CHALLENGE

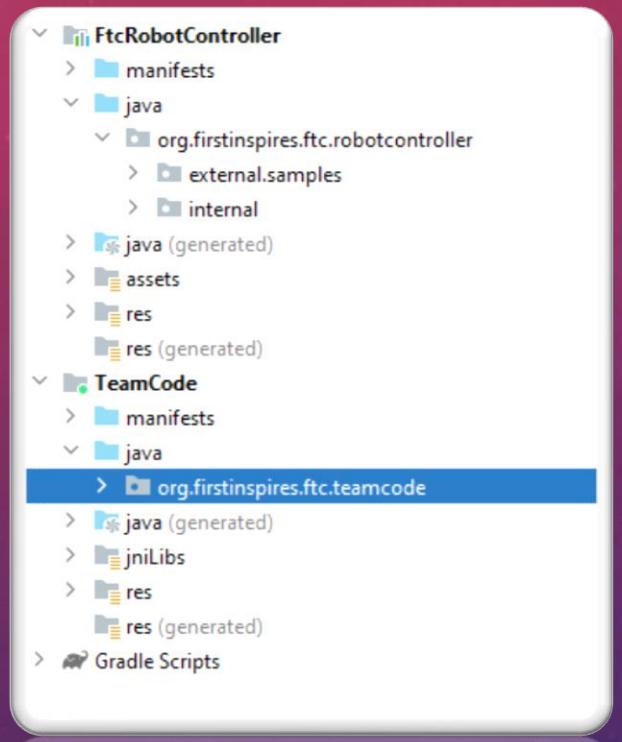
JAVA FRAMEWORK – FTCROBOTCONTROLLER

What is the FTCRobotController?



FIRST TECH CHALLENGE

JAVA FRAMEWORK – FTCROBOTCONTROLLER

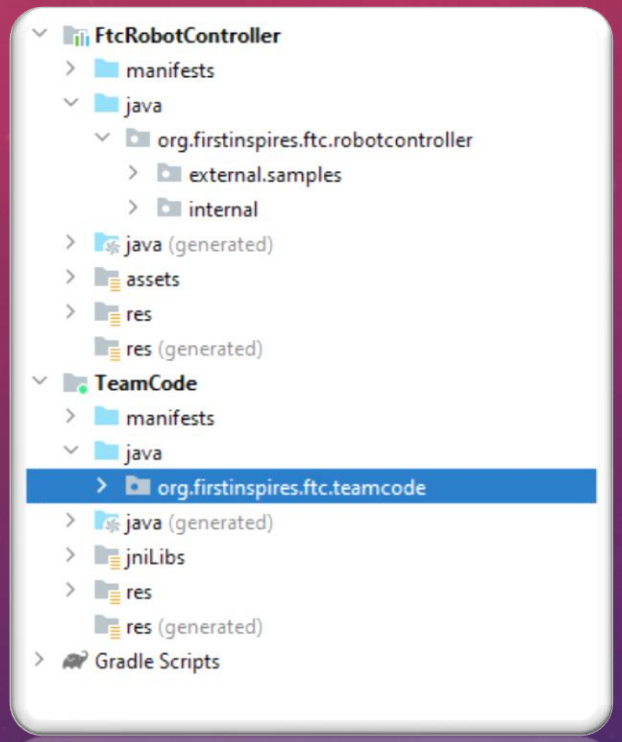


What is the FTCRobotController?

- Android Studio Project obtained from a public repository hosted on GitHub
  - Can download the source as a zip archive
  - Can clone the repository (will need knowledge of how git repository's function)

FIRST TECH CHALLENGE

JAVA FRAMEWORK – FTCROBOTCONTROLLER



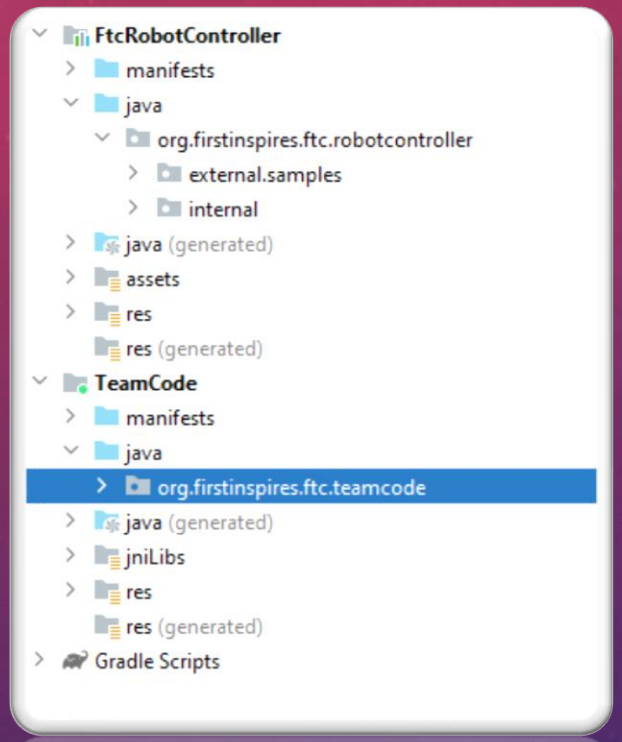
What is the FTCRobotController?

- Android Studio Project obtained from a public repository hosted on GitHub
  - Can download the source as a zip archive
  - Can clone the repository (will need knowledge of how git repository's function)

- Used as the base project for your own FTC Team Competition program
  - Your program code to reside in the TeamCode folder
  - TeamCode package: org.firstinspires.ftc.teamcode

FIRST TECH CHALLENGE

JAVA FRAMEWORK – FTCROBOTCONTROLLER



What is the FTCRobotController?

- Android Studio Project obtained from a public repository hosted on GitHub
  - Can download the source as a zip archive
  - Can clone the repository (will need knowledge of how git repository's function)

- Used as the base project for your own FTC Team Competition program
  - Your program code to reside in the TeamCode folder
  - TeamCode package: org.firstinspires.ftc.teamcode

- Contains sample modules designed to help you implement various components
  - Samples located in the FtcRobotController folder
  - package: org.firstinspires.ftc.robotcontroller.external.samples

FIRST TECH CHALLENGE

OPMODE STRUCTURE - TELEOP

Teleop Program Module

```java
@TeleOp(name="Basic: Omni Linear OpMode", group="Linear Opmode")
@Disabled
public class BasicOmniOpMode_Linear extends LinearOpMode {

    // Declare OpMode members for each of the 4 motors.
    2 usages
    private ElapsedTime runtime = new ElapsedTime();
    3 usages
    private DcMotor leftFrontDrive = null;
    3 usages
    private DcMotor leftBackDrive = null;
    3 usages
    private DcMotor rightFrontDrive = null;
    3 usages
    private DcMotor rightBackDrive = null;

    @Override
    public void runOpMode() {

        // Initialize the hardware variables. Note that the strings used here must correspond
        // to the names assigned during the robot configuration step on the DS or RC devices.
        leftFrontDrive  = hardwareMap.get(DcMotor.class,  deviceName: "left_front_drive");
        leftBackDrive  = hardwareMap.get(DcMotor.class,  deviceName: "left_back_drive");
        rightFrontDrive = hardwareMap.get(DcMotor.class,  deviceName: "right_front_drive");
        rightBackDrive = hardwareMap.get(DcMotor.class,  deviceName: "right_back_drive");

        //...
        leftFrontDrive.setDirection(DcMotor.Direction.REVERSE);
        leftBackDrive.setDirection(DcMotor.Direction.REVERSE);
        rightFrontDrive.setDirection(DcMotor.Direction.FORWARD);
        rightBackDrive.setDirection(DcMotor.Direction.FORWARD);

        // Wait for the game to start (driver presses PLAY)
        telemetry.addData( caption: "Status",  value: "Initialized");
        telemetry.update();

        waitForStart();
        runtime.reset();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {
```

```java
            double max;

            // POV Mode uses left joystick to go forward & strafe, and right joystick to rotate.
            double axial   = -gamepad1.left_stick_y;  // Note: pushing stick forward gives negative
            double lateral =  gamepad1.left_stick_x;
            double yaw     =  gamepad1.right_stick_x;

            // Combine the joystick requests for each axis-motion to determine each wheel's power.
            // Set up a variable for each drive wheel to save the power level for telemetry.
            double leftFrontPower  = axial + lateral + yaw;
            double rightFrontPower = axial - lateral - yaw;
            double leftBackPower   = axial - lateral + yaw;
            double rightBackPower  = axial + lateral - yaw;

            // Normalize the values so no wheel power exceeds 100%
            // This ensures that the robot maintains the desired motion.
            max = Math.max(Math.abs(leftFrontPower), Math.abs(rightFrontPower));
            max = Math.max(max, Math.abs(leftBackPower));
            max = Math.max(max, Math.abs(rightBackPower));

            if (max > 1.0) {
                leftFrontPower  /= max;
                rightFrontPower /= max;
                leftBackPower   /= max;
                rightBackPower  /= max;
            }

            //...

            /*...*/

            // Send calculated power to wheels
            leftFrontDrive.setPower(leftFrontPower);
            rightFrontDrive.setPower(rightFrontPower);
            leftBackDrive.setPower(leftBackPower);
            rightBackDrive.setPower(rightBackPower);

            // Show the elapsed game time and wheel power.
            telemetry.addData( caption: "Status",  value: "Run Time: " + runtime.toString());
            telemetry.addData( caption: "Front left/Right",  format: "%4.2f, %4.2f", leftFrontPower, right
            telemetry.addData( caption: "Back  left/Right",  format: "%4.2f, %4.2f", leftBackPower, rightB
            telemetry.update();
        }
    }}
```

FIRST TECH CHALLENGE

OPMODE STRUCTURE - TELEOP

Teleop Program Module

```
@TeleOp(name="Basic: Omni Linear OpMode", group="Linear Opmode")
@Disabled
```

```
@TeleOp(name="Basic: Omni Linear OpMode", group="Linear Opmode")
@Disabled
public class BasicOmniOpMode_Linear extends LinearOpMode {

    // Declare OpMode members for each of the 4 motors.
    2 usages
    private ElapsedTime runtime = new ElapsedTime();
    3 usages
    private DcMotor leftFrontDrive = null;
    3 usages
    private DcMotor leftBackDrive = null;
    3 usages
    private DcMotor rightFrontDrive = null;
    3 usages
    private DcMotor rightBackDrive = null;

    @Override
    public void runOpMode() {

        // Initialize the hardware variables. Note that the strings used here must correspond
        // to the names assigned during the robot configuration step on the DS or RC devices.
        leftFrontDrive  = hardwareMap.get(DcMotor.class,  deviceName: "left_front_drive");
        leftBackDrive   = hardwareMap.get(DcMotor.class,  deviceName: "left_back_drive");
        rightFrontDrive = hardwareMap.get(DcMotor.class,  deviceName: "right_front_drive");
        rightBackDrive = hardwareMap.get(DcMotor.class,  deviceName: "right_back_drive");

        //...
        leftFrontDrive.setDirection(DcMotor.Direction.REVERSE);
        leftBackDrive.setDirection(DcMotor.Direction.REVERSE);
        rightFrontDrive.setDirection(DcMotor.Direction.FORWARD);
        rightBackDrive.setDirection(DcMotor.Direction.FORWARD);

        // Wait for the game to start (driver presses PLAY)
        telemetry.addData( caption: "Status",  value: "Initialized");
        telemetry.update();

        waitForStart();
        runtime.reset();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {
```

```
            double max;

            // POV Mode uses left joystick to go forward & strafe, and right joystick to rotate.
            double axial   = -gamepad1.left_stick_y;  // Note: pushing stick forward gives negative
            double lateral =  gamepad1.left_stick_x;
            double yaw     =  gamepad1.right_stick_x;

            // Combine the joystick requests for each axis-motion to determine each wheel's power.
            // Set up a variable for each drive wheel to save the power level for telemetry.
            double leftFrontPower  = axial + lateral + yaw;
            double rightFrontPower = axial - lateral - yaw;
            double leftBackPower   = axial - lateral + yaw;
            double rightBackPower  = axial + lateral - yaw;

            // Normalize the values so no wheel power exceeds 100%
            // This ensures that the robot maintains the desired motion.
            max = Math.max(Math.abs(leftFrontPower), Math.abs(rightFrontPower));
            max = Math.max(max, Math.abs(leftBackPower));
            max = Math.max(max, Math.abs(rightBackPower));

            if (max > 1.0) {
                leftFrontPower  /= max;
                rightFrontPower /= max;
                leftBackPower   /= max;
                rightBackPower  /= max;
            }

            //...

            /*...*/

            // Send calculated power to wheels
            leftFrontDrive.setPower(leftFrontPower);
            rightFrontDrive.setPower(rightFrontPower);
            leftBackDrive.setPower(leftBackPower);
            rightBackDrive.setPower(rightBackPower);

            // Show the elapsed game time and wheel power.
            telemetry.addData( caption: "Status",  value: "Run Time: " + runtime.toString());
            telemetry.addData( caption: "Front left/Right",  format: "%4.2f, %4.2f", leftFrontPower, right
            telemetry.addData( caption: "Back  left/Right",  format: "%4.2f, %4.2f", leftBackPower, rightB
            telemetry.update();
        }
    }}
```

FIRST TECH CHALLENGE

OPMODE STRUCTURE - TELEOP

Teleop Program Module

```
@TeleOp(name="Basic: Omni Linear OpMode", group="Linear Opmode")
@Disabled
```

```
@Override
public void runOpMode() {
```

```
@TeleOp(name="Basic: Omni Linear OpMode", group="Linear Opmode")
@Disabled
public class BasicOmniOpMode_Linear extends LinearOpMode {

    // Declare OpMode members for each of the 4 motors.
    2 usages
    private ElapsedTime runtime = new ElapsedTime();
    3 usages
    private DcMotor leftFrontDrive = null;
    3 usages
    private DcMotor leftBackDrive = null;
    3 usages
    private DcMotor rightFrontDrive = null;
    3 usages
    private DcMotor rightBackDrive = null;

    @Override
    public void runOpMode() {

        // Initialize the hardware variables. Note that the strings used here must correspond
        // to the names assigned during the robot configuration step on the DS or RC devices.
        leftFrontDrive  = hardwareMap.get(DcMotor.class,  deviceName: "left_front_drive");
        leftBackDrive   = hardwareMap.get(DcMotor.class,  deviceName: "left_back_drive");
        rightFrontDrive = hardwareMap.get(DcMotor.class,  deviceName: "right_front_drive");
        rightBackDrive = hardwareMap.get(DcMotor.class,  deviceName: "right_back_drive");

        //...
        leftFrontDrive.setDirection(DcMotor.Direction.REVERSE);
        leftBackDrive.setDirection(DcMotor.Direction.REVERSE);
        rightFrontDrive.setDirection(DcMotor.Direction.FORWARD);
        rightBackDrive.setDirection(DcMotor.Direction.FORWARD);

        // Wait for the game to start (driver presses PLAY)
        telemetry.addData( caption: "Status",  value: "Initialized");
        telemetry.update();

        waitForStart();
        runtime.reset();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {
```

```
        double max;

        // POV Mode uses left joystick to go forward & strafe, and right joystick to rotate.
        double axial   = -gamepad1.left_stick_y;  // Note: pushing stick forward gives negative
        double lateral =  gamepad1.left_stick_x;
        double yaw     =  gamepad1.right_stick_x;

        // Combine the joystick requests for each axis-motion to determine each wheel's power.
        // Set up a variable for each drive wheel to save the power level for telemetry.
        double leftFrontPower  = axial + lateral + yaw;
        double rightFrontPower = axial - lateral - yaw;
        double leftBackPower   = axial - lateral + yaw;
        double rightBackPower  = axial + lateral - yaw;

        // Normalize the values so no wheel power exceeds 100%
        // This ensures that the robot maintains the desired motion.
        max = Math.max(Math.abs(leftFrontPower), Math.abs(rightFrontPower));
        max = Math.max(max, Math.abs(leftBackPower));
        max = Math.max(max, Math.abs(rightBackPower));

        if (max > 1.0) {
            leftFrontPower  /= max;
            rightFrontPower /= max;
            leftBackPower   /= max;
            rightBackPower  /= max;
        }

        //...

        /*...*/

        // Send calculated power to wheels
        leftFrontDrive.setPower(leftFrontPower);
        rightFrontDrive.setPower(rightFrontPower);
        leftBackDrive.setPower(leftBackPower);
        rightBackDrive.setPower(rightBackPower);

        // Show the elapsed game time and wheel power.
        telemetry.addData( caption: "Status",  value: "Run Time: " + runtime.toString());
        telemetry.addData( caption: "Front left/Right",  format: "%4.2f, %4.2f", leftFrontPower, right
        telemetry.addData( caption: "Back  left/Right",  format: "%4.2f, %4.2f", leftBackPower, rightB
        telemetry.update();
    }
}}
```

FIRST TECH CHALLENGE

OPMODE STRUCTURE - TELEOP

Teleop Program Module

```
@TeleOp(name="Basic: Omni Linear OpMode", group="Linear Opmode")
@Disabled
```

```
@Override
public void runOpMode() {
```

```
    waitForStart();
    runtime.reset();
```

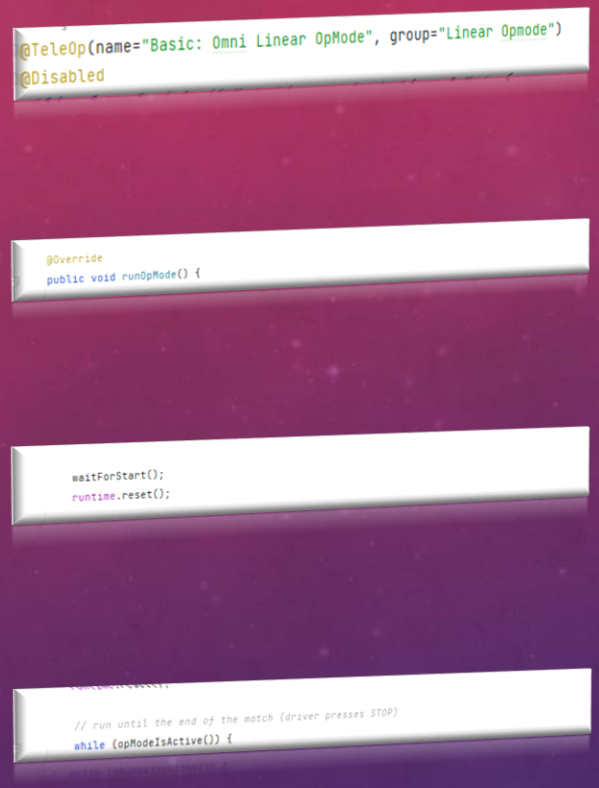
```
@TeleOp(name="Basic: Omni Linear OpMode", group="Linear Opmode")
@Disabled
public class BasicOmniOpMode_Linear extends LinearOpMode {

    // Declare OpMode members for each of the 4 motors.
    2 usages
    private ElapsedTime runtime = new ElapsedTime();
    3 usages
    private DcMotor leftFrontDrive = null;
    3 usages
    private DcMotor leftBackDrive = null;
    3 usages
    private DcMotor rightFrontDrive = null;
    3 usages
    private DcMotor rightBackDrive = null;

    @Override
    public void runOpMode() {

        // Initialize the hardware variables. Note that the strings used here must correspond
        // to the names assigned during the robot configuration step on the DS or RC devices.
        leftFrontDrive  = hardwareMap.get(DcMotor.class, deviceName: "left_front_drive");
        leftBackDrive  = hardwareMap.get(DcMotor.class, deviceName: "left_back_drive");
        rightFrontDrive = hardwareMap.get(DcMotor.class, deviceName: "right_front_drive");
        rightBackDrive = hardwareMap.get(DcMotor.class, deviceName: "right_back_drive");

        //...
        leftFrontDrive.setDirection(DcMotor.Direction.REVERSE);
        leftBackDrive.setDirection(DcMotor.Direction.REVERSE);
        rightFrontDrive.setDirection(DcMotor.Direction.FORWARD);
        rightBackDrive.setDirection(DcMotor.Direction.FORWARD);

        // Wait for the game to start (driver presses PLAY)
        telemetry.addData( caption: "Status", value: "Initialized");
        telemetry.update();

        waitForStart();
        runtime.reset();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {
```

```
            double max;

            // POV Mode uses left joystick to go forward & strafe, and right joystick to rotate.
            double axial   = -gamepad1.left_stick_y;  // Note: pushing stick forward gives negative
            double lateral =  gamepad1.left_stick_x;
            double yaw     =  gamepad1.right_stick_x;

            // Combine the joystick requests for each axis-motion to determine each wheel's power.
            // Set up a variable for each drive wheel to save the power level for telemetry.
            double leftFrontPower  = axial + lateral + yaw;
            double rightFrontPower = axial - lateral - yaw;
            double leftBackPower   = axial - lateral + yaw;
            double rightBackPower  = axial + lateral - yaw;

            // Normalize the values so no wheel power exceeds 100%
            // This ensures that the robot maintains the desired motion.
            max = Math.max(Math.abs(leftFrontPower), Math.abs(rightFrontPower));
            max = Math.max(max, Math.abs(leftBackPower));
            max = Math.max(max, Math.abs(rightBackPower));

            if (max > 1.0) {
                leftFrontPower  /= max;
                rightFrontPower /= max;
                leftBackPower   /= max;
                rightBackPower  /= max;
            }

            //...

            /*...*/

            // Send calculated power to wheels
            leftFrontDrive.setPower(leftFrontPower);
            rightFrontDrive.setPower(rightFrontPower);
            leftBackDrive.setPower(leftBackPower);
            rightBackDrive.setPower(rightBackPower);

            // Show the elapsed game time and wheel power.
            telemetry.addData( caption: "Status", value: "Run Time: " + runtime.toString());
            telemetry.addData( caption: "Front left/Right", format: "%4.2f, %4.2f", leftFrontPower, right
            telemetry.addData( caption: "Back  left/Right", format: "%4.2f, %4.2f", leftBackPower, rightB
            telemetry.update();
        }
    }}
```

FIRST TECH CHALLENGE

OPMODE STRUCTURE - TELEOP

Teleop Program Module




```java
@TeleOp(name="Basic: Omni Linear OpMode", group="Linear Opmode")
@Disabled
public class BasicOmniOpMode_Linear extends LinearOpMode {

    // Declare OpMode members for each of the 4 motors.
    2 usages
    private ElapsedTime runtime = new ElapsedTime();
    3 usages
    private DcMotor leftFrontDrive = null;
    3 usages
    private DcMotor leftBackDrive = null;
    3 usages
    private DcMotor rightFrontDrive = null;
    3 usages
    private DcMotor rightBackDrive = null;

    @Override
    public void runOpMode() {

        // Initialize the hardware variables. Note that the strings used here must correspond
        // to the names assigned during the robot configuration step on the DS or RC devices.
        leftFrontDrive  = hardwareMap.get(DcMotor.class,  deviceName: "left_front_drive");
        leftBackDrive  = hardwareMap.get(DcMotor.class,  deviceName: "left_back_drive");
        rightFrontDrive = hardwareMap.get(DcMotor.class,  deviceName: "right_front_drive");
        rightBackDrive = hardwareMap.get(DcMotor.class,  deviceName: "right_back_drive");

        //...
        leftFrontDrive.setDirection(DcMotor.Direction.REVERSE);
        leftBackDrive.setDirection(DcMotor.Direction.REVERSE);
        rightFrontDrive.setDirection(DcMotor.Direction.FORWARD);
        rightBackDrive.setDirection(DcMotor.Direction.FORWARD);

        // Wait for the game to start (driver presses PLAY)
        telemetry.addData( caption: "Status",  value: "Initialized");
        telemetry.update();

        waitForStart();
        runtime.reset();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {
```

```java
            double max;

            // POV Mode uses left joystick to go forward & strafe, and right joystick to rotate.
            double axial   = -gamepad1.left_stick_y;  // Note: pushing stick forward gives negative
            double lateral =  gamepad1.left_stick_x;
            double yaw     =  gamepad1.right_stick_x;

            // Combine the joystick requests for each axis-motion to determine each wheel's power.
            // Set up a variable for each drive wheel to save the power level for telemetry.
            double leftFrontPower  = axial + lateral + yaw;
            double rightFrontPower = axial - lateral - yaw;
            double leftBackPower   = axial - lateral + yaw;
            double rightBackPower  = axial + lateral - yaw;

            // Normalize the values so no wheel power exceeds 100%
            // This ensures that the robot maintains the desired motion.
            max = Math.max(Math.abs(leftFrontPower), Math.abs(rightFrontPower));
            max = Math.max(max, Math.abs(leftBackPower));
            max = Math.max(max, Math.abs(rightBackPower));

            if (max > 1.0) {
                leftFrontPower  /= max;
                rightFrontPower /= max;
                leftBackPower   /= max;
                rightBackPower  /= max;
            }

            //...

            /*...*/

            // Send calculated power to wheels
            leftFrontDrive.setPower(leftFrontPower);
            rightFrontDrive.setPower(rightFrontPower);
            leftBackDrive.setPower(leftBackPower);
            rightBackDrive.setPower(rightBackPower);

            // Show the elapsed game time and wheel power.
            telemetry.addData( caption: "Status",  value: "Run Time: " + runtime.toString());
            telemetry.addData( caption: "Front left/Right",  format: "%4.2f, %4.2f", leftFrontPower, right
            telemetry.addData( caption: "Back  left/Right",  format: "%4.2f, %4.2f", leftBackPower, rightB
            telemetry.update();
        }
    }}
```

FIRST TECH CHALLENGE

OPMODE STRUCTURE - AUTONOMOUS

Autonomous Program Module

```java
@Autonomous(name="Robot: Auto Drive To Line", group="Robot")
@Disabled
public class RobotAutoDriveToLine_Linear extends LinearOpMode {

    /* Declare OpMode members. */
    4 usages
    private DcMotor         leftDrive   = null;
    4 usages
    private DcMotor         rightDrive  = null;

    /** The colorSensor field will contain a reference to our color sensor hardware object */
    5 usages
    NormalizedColorSensor colorSensor;

    1 usage
    static final double     WHITE_THRESHOLD = 0.5;  // spans between 0.0 - 1.0 from dark to light
    2 usages
    static final double     APPROACH_SPEED  = 0.25;

    @Override
    public void runOpMode() {

        // Initialize the drive system variables.
        leftDrive  = hardwareMap.get(DcMotor.class, deviceName: "left_drive");
        rightDrive = hardwareMap.get(DcMotor.class, deviceName: "right_drive");

        //...
        leftDrive.setDirection(DcMotor.Direction.REVERSE);
        rightDrive.setDirection(DcMotor.Direction.FORWARD);

        //...
        colorSensor = hardwareMap.get(NormalizedColorSensor.class, deviceName: "sensor_color");

        // If necessary, turn ON the white LED (if there is no LED switch on the sensor)
        if (colorSensor instanceof SwitchableLight) {
            ((SwitchableLight)colorSensor).enableLight(true);
        }

        //...
        colorSensor.setGain(15);

        //...
        while (opModeInInit()) {

            // Send telemetry message to signify robot waiting;
            telemetry.addData( caption: "Status", value: "Ready to drive to white line.");    //

            // Display the light level while we are waiting to start
            getBrightness();
        }

        // Start the robot moving forward, and then begin looking for a white line.
        leftDrive.setPower(APPROACH_SPEED);
        rightDrive.setPower(APPROACH_SPEED);

        // run until the white line is seen OR the driver presses STOP;
        while (opModeIsActive() && (getBrightness() < WHITE_THRESHOLD)) {
            sleep( milliseconds: 5);
        }

        // Stop all motors
        leftDrive.setPower(0);
        rightDrive.setPower(0);
    }

    // to obtain reflected light, read the normalized values from the color sensor.  Return the Alph
    2 usages
    double getBrightness() {
        NormalizedRGBA colors = colorSensor.getNormalizedColors();
        telemetry.addData( caption: "Light Level (0 to 1)",  format: "%4.2f", colors.alpha);
        telemetry.update();

        return colors.alpha;
    }
}
```

FIRST TECH CHALLENGE

OPMODE STRUCTURE - AUTONOMOUS

Autonomous Program Module

```
@Autonomous(name="Robot: Auto Drive To Line", group="Robot")
@Disabled
```

```
@Autonomous(name="Robot: Auto Drive To Line", group="Robot")
@Disabled
public class RobotAutoDriveToLine_Linear extends LinearOpMode {

    /* Declare OpMode members. */
    4 usages
    private DcMotor         leftDrive   = null;
    4 usages
    private DcMotor         rightDrive  = null;

    /** The colorSensor field will contain a reference to our color sensor hardware object */
    5 usages
    NormalizedColorSensor colorSensor;

    1 usage
    static final double     WHITE_THRESHOLD = 0.5;  // spans between 0.0 - 1.0 from dark to light
    2 usages
    static final double     APPROACH_SPEED  = 0.25;

    @Override
    public void runOpMode() {

        // Initialize the drive system variables.
        leftDrive  = hardwareMap.get(DcMotor.class, deviceName: "left_drive");
        rightDrive = hardwareMap.get(DcMotor.class, deviceName: "right_drive");

        //...
        leftDrive.setDirection(DcMotor.Direction.REVERSE);
        rightDrive.setDirection(DcMotor.Direction.FORWARD);

        //...
        colorSensor = hardwareMap.get(NormalizedColorSensor.class, deviceName: "sensor_color");

        // If necessary, turn ON the white LED (if there is no LED switch on the sensor)
        if (colorSensor instanceof SwitchableLight) {
            ((SwitchableLight)colorSensor).enableLight(true);
        }

        //...
        colorSensor.setGain(15);

        //...
        while (opModeInInit()) {
```

```
            // Send telemetry message to signify robot waiting;
            telemetry.addData( caption: "Status", value: "Ready to drive to white line.");    //

            // Display the light level while we are waiting to start
            getBrightness();
        }

        // Start the robot moving forward, and then begin looking for a white line.
        leftDrive.setPower(APPROACH_SPEED);
        rightDrive.setPower(APPROACH_SPEED);

        // run until the white line is seen OR the driver presses STOP;
        while (opModeIsActive() && (getBrightness() < WHITE_THRESHOLD)) {
            sleep( milliseconds: 5);
        }

        // Stop all motors
        leftDrive.setPower(0);
        rightDrive.setPower(0);
    }

    // to obtain reflected light, read the normalized values from the color sensor.  Return the Alph
    2 usages
    double getBrightness() {
        NormalizedRGBA colors = colorSensor.getNormalizedColors();
        telemetry.addData( caption: "Light Level (0 to 1)",  format: "%4.2f", colors.alpha);
        telemetry.update();

        return colors.alpha;
    }
}
```

FIRST TECH CHALLENGE

OPMODE STRUCTURE - AUTONOMOUS

Autonomous Program Module

```
@Autonomous(name="Robot: Auto Drive To Line", group="Robot")
@Disabled
```

```
    @Override
    public void runOpMode() {
```

```
@Autonomous(name="Robot: Auto Drive To Line", group="Robot")
@Disabled
public class RobotAutoDriveToLine_Linear extends LinearOpMode {

    /* Declare OpMode members. */
    4 usages
    private DcMotor         leftDrive   = null;
    4 usages
    private DcMotor         rightDrive  = null;

    /** The colorSensor field will contain a reference to our color sensor hardware object */
    5 usages
    NormalizedColorSensor colorSensor;

    1 usage
    static final double     WHITE_THRESHOLD = 0.5;  // spans between 0.0 - 1.0 from dark to light
    2 usages
    static final double     APPROACH_SPEED  = 0.25;

    @Override
    public void runOpMode() {

        // Initialize the drive system variables.
        leftDrive  = hardwareMap.get(DcMotor.class, deviceName: "left_drive");
        rightDrive = hardwareMap.get(DcMotor.class, deviceName: "right_drive");

        //...
        leftDrive.setDirection(DcMotor.Direction.REVERSE);
        rightDrive.setDirection(DcMotor.Direction.FORWARD);

        //...
        colorSensor = hardwareMap.get(NormalizedColorSensor.class, deviceName: "sensor_color");

        // If necessary, turn ON the white LED (if there is no LED switch on the sensor)
        if (colorSensor instanceof SwitchableLight) {
            ((SwitchableLight)colorSensor).enableLight(true);
        }

        //...
        colorSensor.setGain(15);

        //...
        while (opModeInInit()) {
```

```
            // Send telemetry message to signify robot waiting;
            telemetry.addData( caption: "Status", value: "Ready to drive to white line.");    //

            // Display the light level while we are waiting to start
            getBrightness();
        }

        // Start the robot moving forward, and then begin looking for a white line.
        leftDrive.setPower(APPROACH_SPEED);
        rightDrive.setPower(APPROACH_SPEED);

        // run until the white line is seen OR the driver presses STOP;
        while (opModeIsActive() && (getBrightness() < WHITE_THRESHOLD)) {
            sleep( milliseconds: 5);
        }

        // Stop all motors
        leftDrive.setPower(0);
        rightDrive.setPower(0);
    }

    // to obtain reflected light, read the normalized values from the color sensor.  Return the Alph
    2 usages
    double getBrightness() {
        NormalizedRGBA colors = colorSensor.getNormalizedColors();
        telemetry.addData( caption: "Light Level (0 to 1)",  format: "%4.2f", colors.alpha);
        telemetry.update();

        return colors.alpha;
    }
}
```

FIRST TECH CHALLENGE

OPMODE STRUCTURE - AUTONOMOUS

Autonomous Program Module

```
@Autonomous(name="Robot: Auto Drive To Line", group="Robot")
@Disabled
```

```
@Override
public void runOpMode() {
```

```
//...
while (opModeInInit()) {
```

```
@Autonomous(name="Robot: Auto Drive To Line", group="Robot")
@Disabled
public class RobotAutoDriveToLine_Linear extends LinearOpMode {

    /* Declare OpMode members. */
    4 usages
    private DcMotor         leftDrive   = null;
    4 usages
    private DcMotor         rightDrive  = null;

    /** The colorSensor field will contain a reference to our color sensor hardware object */
    5 usages
    NormalizedColorSensor colorSensor;

    1 usage
    static final double     WHITE_THRESHOLD = 0.5;  // spans between 0.0 - 1.0 from dark to light
    2 usages
    static final double     APPROACH_SPEED  = 0.25;

    @Override
    public void runOpMode() {

        // Initialize the drive system variables.
        leftDrive  = hardwareMap.get(DcMotor.class,  deviceName: "left_drive");
        rightDrive = hardwareMap.get(DcMotor.class,  deviceName: "right_drive");

        //...
        leftDrive.setDirection(DcMotor.Direction.REVERSE);
        rightDrive.setDirection(DcMotor.Direction.FORWARD);

        //...
        colorSensor = hardwareMap.get(NormalizedColorSensor.class,  deviceName: "sensor_color");

        // If necessary, turn ON the white LED (if there is no LED switch on the sensor)
        if (colorSensor instanceof SwitchableLight) {
            ((SwitchableLight)colorSensor).enableLight(true);
        }

        //...
        colorSensor.setGain(15);

        //...
        while (opModeInInit()) {
```

```
            // Send telemetry message to signify robot waiting;
            telemetry.addData( caption: "Status",  value: "Ready to drive to white line.");    //

            // Display the light level while we are waiting to start
            getBrightness();
        }

        // Start the robot moving forward, and then begin looking for a white line.
        leftDrive.setPower(APPROACH_SPEED);
        rightDrive.setPower(APPROACH_SPEED);

        // run until the white line is seen OR the driver presses STOP;
        while (opModeIsActive() && (getBrightness() < WHITE_THRESHOLD)) {
            sleep( milliseconds: 5);
        }

        // Stop all motors
        leftDrive.setPower(0);
        rightDrive.setPower(0);
    }

    // to obtain reflected light, read the normalized values from the color sensor.  Return the Alph
    2 usages
    double getBrightness() {
        NormalizedRGBA colors = colorSensor.getNormalizedColors();
        telemetry.addData( caption: "Light Level (0 to 1)",  format: "%4.2f", colors.alpha);
        telemetry.update();

        return colors.alpha;
    }
}
```

FIRST TECH CHALLENGE

OPMODE STRUCTURE - AUTONOMOUS

Autonomous Program Module

```
@Autonomous(name="Robot: Auto Drive To Line", group="Robot")
@Disabled
```

```
@Override
public void runOpMode() {
```

```
//...
while (opModeInInit()) {
```

```
// run until the white line is seen OR the driver presses STOP;
while (opModeIsActive() && (getBrightness() < WHITE_THRESHOLD)) {
    sleep( milliseconds: 5);
}
```

```
@Autonomous(name="Robot: Auto Drive To Line", group="Robot")
@Disabled
public class RobotAutoDriveToLine_Linear extends LinearOpMode {

    /* Declare OpMode members. */
    4 usages
    private DcMotor         leftDrive   = null;
    4 usages
    private DcMotor         rightDrive  = null;

    /** The colorSensor field will contain a reference to our color sensor hardware object */
    5 usages
    NormalizedColorSensor colorSensor;

    1 usage
    static final double     WHITE_THRESHOLD = 0.5;  // spans between 0.0 - 1.0 from dark to light
    2 usages
    static final double     APPROACH_SPEED  = 0.25;

    @Override
    public void runOpMode() {

        // Initialize the drive system variables.
        leftDrive  = hardwareMap.get(DcMotor.class,  deviceName: "left_drive");
        rightDrive = hardwareMap.get(DcMotor.class,  deviceName: "right_drive");

        //...
        leftDrive.setDirection(DcMotor.Direction.REVERSE);
        rightDrive.setDirection(DcMotor.Direction.FORWARD);

        //...
        colorSensor = hardwareMap.get(NormalizedColorSensor.class,  deviceName: "sensor_color");

        // If necessary, turn ON the white LED (if there is no LED switch on the sensor)
        if (colorSensor instanceof SwitchableLight) {
            ((SwitchableLight)colorSensor).enableLight(true);
        }

        //...
        colorSensor.setGain(15);

        //...
        while (opModeInInit()) {
```

```
            // Send telemetry message to signify robot waiting;
            telemetry.addData( caption: "Status",  value: "Ready to drive to white line.");    //

            // Display the light level while we are waiting to start
            getBrightness();
        }

        // Start the robot moving forward, and then begin looking for a white line.
        leftDrive.setPower(APPROACH_SPEED);
        rightDrive.setPower(APPROACH_SPEED);

        // run until the white line is seen OR the driver presses STOP;
        while (opModeIsActive() && (getBrightness() < WHITE_THRESHOLD)) {
            sleep( milliseconds: 5);
        }

        // Stop all motors
        leftDrive.setPower(0);
        rightDrive.setPower(0);
    }

    // to obtain reflected light, read the normalized values from the color sensor.  Return the Alph
    2 usages
    double getBrightness() {
        NormalizedRGBA colors = colorSensor.getNormalizedColors();
        telemetry.addData( caption: "Light Level (0 to 1)",  format: "%4.2f", colors.alpha);
        telemetry.update();

        return colors.alpha;
    }
}
```

FIRST TECH CHALLENGE

OPMODE STRUCTURE - AUTONOMOUS

Autonomous Program Module

```
@Autonomous(name="Robot: Auto Drive To Line", group="Robot")
@Disabled
```

```
@Override
public void runOpMode() {
```

```
//...
while (opModeInInit()) {
```

```
// run until the white line is seen OR the driver presses STOP;
while (opModeIsActive() && (getBrightness() < WHITE_THRESHOLD)) {
    sleep( milliseconds: 5);
}
```

```
// to obtain reflected light, read the normalized values from the color sensor.  Return the Alp
2 usages
double getBrightness() {
    NormalizedRGBA colors = colorSensor.getNormalizedColors();
    telemetry.addData( caption: "Light Level (0 to 1)",  format: "%4.2f", colors.alpha);
    telemetry.update();

    return colors.alpha;
}
```

```
@Autonomous(name="Robot: Auto Drive To Line", group="Robot")
@Disabled
public class RobotAutoDriveToLine_Linear extends LinearOpMode {

    /* Declare OpMode members. */
    4 usages
    private DcMotor         leftDrive   = null;
    4 usages
    private DcMotor         rightDrive  = null;

    /** The colorSensor field will contain a reference to our color sensor hardware object */
    5 usages
    NormalizedColorSensor colorSensor;

    1 usage
    static final double     WHITE_THRESHOLD = 0.5;  // spans between 0.0 - 1.0 from dark to light
    2 usages
    static final double     APPROACH_SPEED  = 0.25;

    @Override
    public void runOpMode() {

        // Initialize the drive system variables.
        leftDrive  = hardwareMap.get(DcMotor.class,  deviceName: "left_drive");
        rightDrive = hardwareMap.get(DcMotor.class,  deviceName: "right_drive");

        //...
        leftDrive.setDirection(DcMotor.Direction.REVERSE);
        rightDrive.setDirection(DcMotor.Direction.FORWARD);

        //...
        colorSensor = hardwareMap.get(NormalizedColorSensor.class,  deviceName: "sensor_color");

        // If necessary, turn ON the white LED (if there is no LED switch on the sensor)
        if (colorSensor instanceof SwitchableLight) {
            ((SwitchableLight)colorSensor).enableLight(true);
        }

        //...
        colorSensor.setGain(15);

        //...
        while (opModeInInit()) {
```

```
            // Send telemetry message to signify robot waiting;
            telemetry.addData( caption: "Status",  value: "Ready to drive to white line.");    //

            // Display the light level while we are waiting to start
            getBrightness();
        }

        // Start the robot moving forward, and then begin looking for a white line.
        leftDrive.setPower(APPROACH_SPEED);
        rightDrive.setPower(APPROACH_SPEED);

        // run until the white line is seen OR the driver presses STOP;
        while (opModeIsActive() && (getBrightness() < WHITE_THRESHOLD)) {
            sleep( milliseconds: 5);
        }

        // Stop all motors
        leftDrive.setPower(0);
        rightDrive.setPower(0);
    }

    // to obtain reflected light, read the normalized values from the color sensor.  Return the Alph
    2 usages
    double getBrightness() {
        NormalizedRGBA colors = colorSensor.getNormalizedColors();
        telemetry.addData( caption: "Light Level (0 to 1)",  format: "%4.2f", colors.alpha);
        telemetry.update();

        return colors.alpha;
    }
}
```

FIRST TECH CHALLENGE

RESOURCES

[FIRST Tech Challenge documentation — FIRST Tech Challenge Docs 0.2 documentation (firstinspires.org)]

Development Environment

- Android Studio - https://developer.android.com/studio
- FTCRobotController – https://github.com/FIRST-Tech-Challenge/FtcRobotController
- Git-Scm - https://git-scm.com/
- Github Desktop Client - https://desktop.github.com/

[2023-2024 FTC Resources | fruitportrobotics]

FIRST
TECH
CHALLENGE

FTC JAVA PROGRAMMING BASICS

FRUITPORT TECHNO TROJANS FTC WORKSHOP

PRESENTED BY:
Lawrence Welty

THANK YOU!

QUESTIONS AND ANSWERS

FIRST TECH CHALLENGE