FTC Programming in Java

Patrick R. Michaud
University of Texas at Dallas
Jonsson School of Engineering
patrick.michaud@utdallas.edu

Goals

Introduce FTC robot controller apps

Learn programming basics for FTC robots

Topics

Setup basics

Autonomous and teleop templates

Motor and servo control

Driving logic

Joystick buttons

Robot control system overview

Smartphone based

ZTE Speed
Motorola Moto G 2nd gen
Motorola Moto G 3rd gen
Motorola Moto G4 Play
Motorola Moto E4
Google Nexus 5
Samsung Galaxy S5

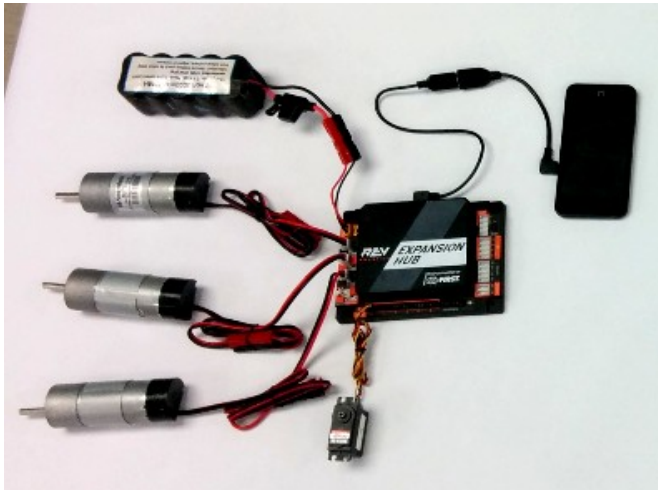
Controllers for motors, servos, sensors

Programming in Blocks, OnBotJava, or Android Studio (Java)

Robot control system basics

Two phones:

- Driver Station
- Robot Controller



Gamepads connect to Driver Station via USB OTG cable

Robot Controller phone connects to REV Expansion Hub via USB OTG cable

Motors, servos, and sensors connect to REV Expansion Hub

Setup: ftc_app SDK

Detailed instructions available on ftc_app wiki:

https://github.com/ftctechnh/ftc_app/wiki/

Setup: Configure phone

https://github.com/ftctechnh/ftc_app/wiki/Configuring-Your-Android-Devices

Programming process basics

All teams start with the same basic app framework, called “FtcRobotController”

Teams customize the app with “opmodes” for their robot's specific functionality

OpModes

Driver Station can place the Robot Controller app into one of several operational modes (“opmodes”)

Examples:

- TeleOp
- Autonomous 1
- Autonomous 2
- Stop Robot

Each “opmode” is a separate *class* instance in the Robot Controller App

Basic OpMode Java code

```
@TeleOp(name="MyTeleOp", group="Iterative Opmode")
// @Disabled
public class MyTeleOp extends OpMode {

    /* variables common to all methods */

    public void init() {
        /* code to run when opmode entered */
        /* set hardware variables */
    }

    public void loop() {
        /* code run repeatedly during operation
         * once every 25 milliseconds (nominal) */
    }

    public void stop() {
        /* cleanup to run when opmode disabled */
    }
}
```

Programming - motors

In variables section, declare a variable for each motor configured on the robot

```
private DcMotor leftDrive = null;
private DcMotor rightDrive = null;
```

Variable type

Variable name

In Java, type names normally begin with uppercase letters; variables, functions, and method calls start with lowercase letters.

In init() method, lookup components by name from controller configuration file

```
leftDrive = hardwareMap.get(DcMotor.class, “leftdrive”);
rightDrive = hardwareMap.get(DcMotor.class, “rightdrive”);
```

Java method calls

Java is an *object-oriented language*

Many operations are performed by making *method calls* using the dot notation

```
leftDrive.setDirection(DcMotor.Direction.REVERSE);
```

Which object we're working with

What we want it to do or get

Method calls can return a result

```
leftDrive = hardwareMap.get(DcMotor.class, “leftdrive”);
```

Programming - motors

To make a motor move, set its power to be between -1.0 and +1.0 in the loop() method:

```
leftDrive.setPower(0);      /* turn off motor */
rightDrive.setPower(1.0);   /* full power forward */
leftDrive.setPower(-0.5);   /* 50% power reverse */
```

The motor will continue running at its last commanded speed until given a new setPower value or the OpMode is cancelled

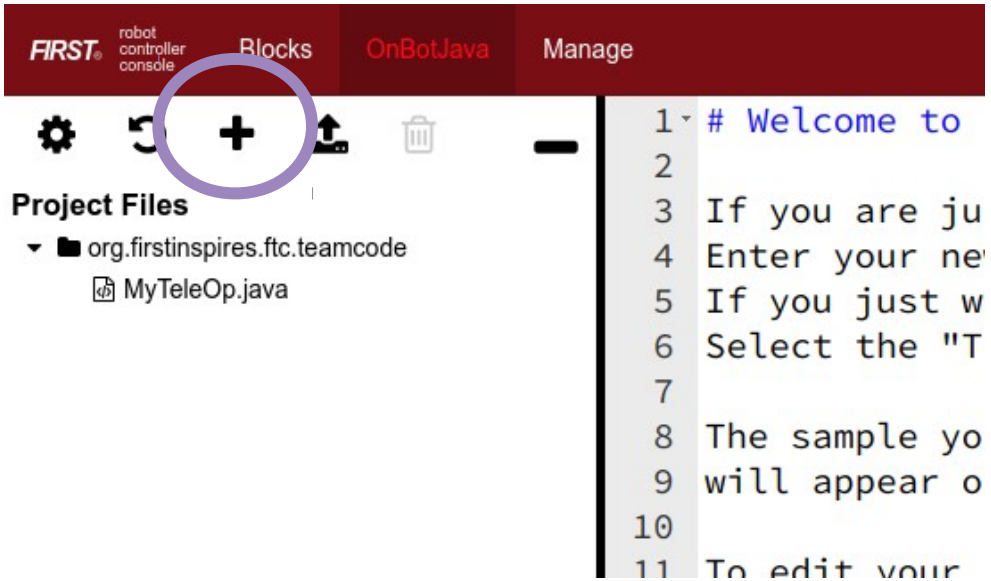
Create a new OpMode (OnBotJava)

Connect web browser to Program & Manage

http://192.168.49.1:8080

Select “OnBotJava” at the top of the screen

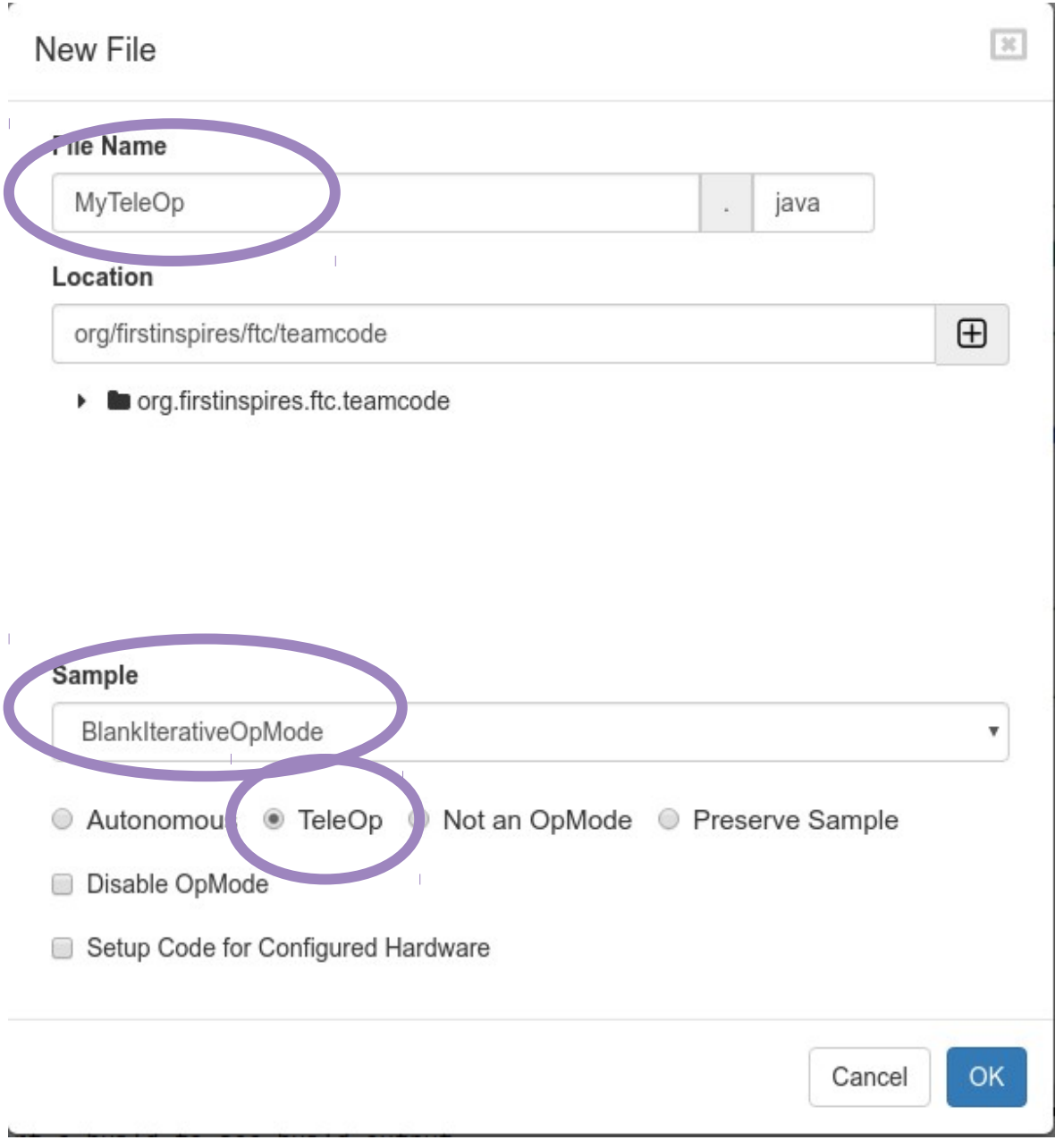
Click the “+” symbol in the left panel to create a new file.



1. Select a name for the opmode

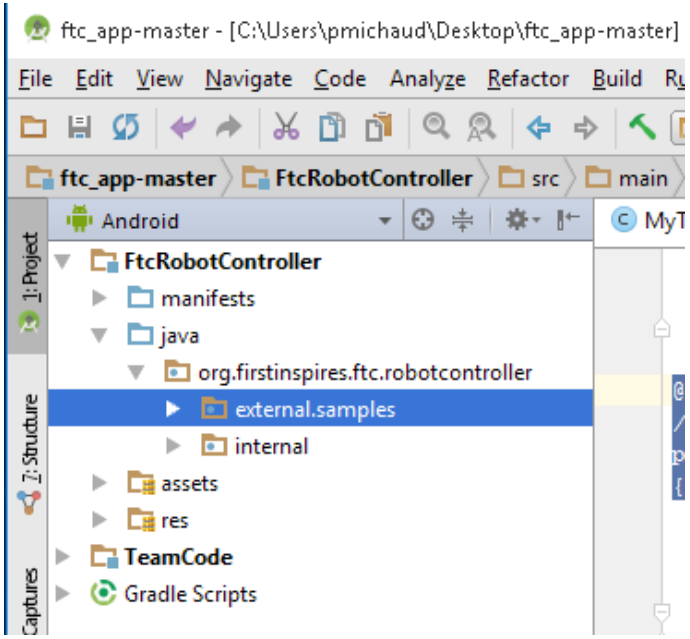
2. Select “BlankIterativeOpMode” as the template

3. Place it in the “TeleOp” menu of the Driver Station

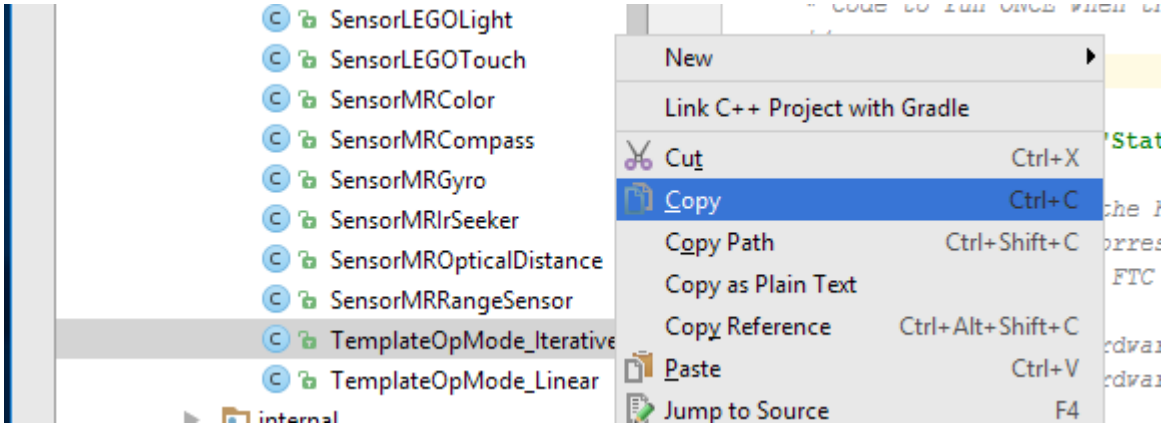


How to create a new OpMode (Android Studio)

Locate
“BasicOpMode_Iterative” or
“BasicOpMode_Linear”
in the external.samples folder

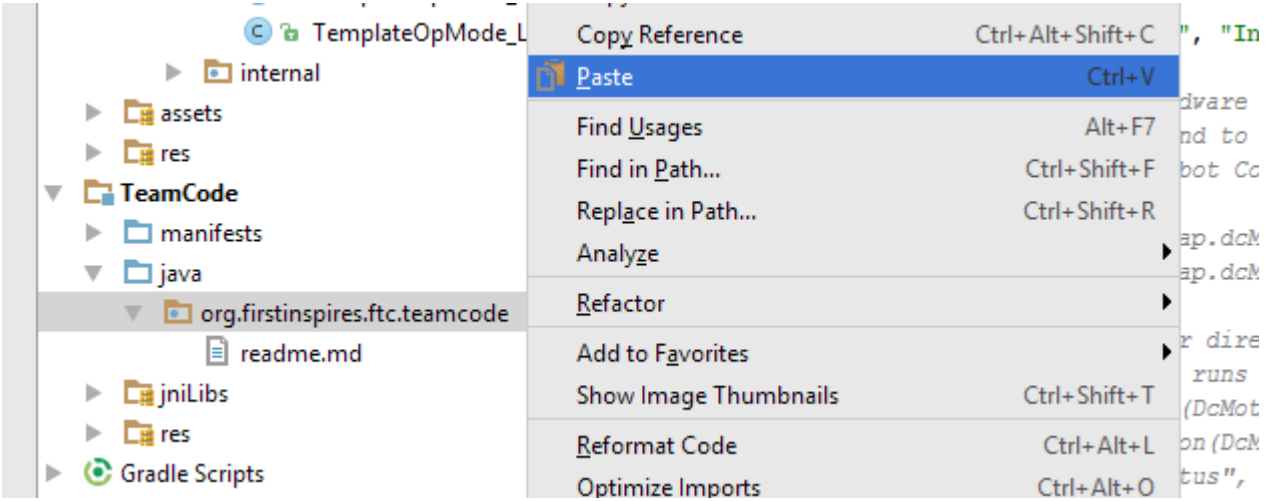


Right click and select “Copy”

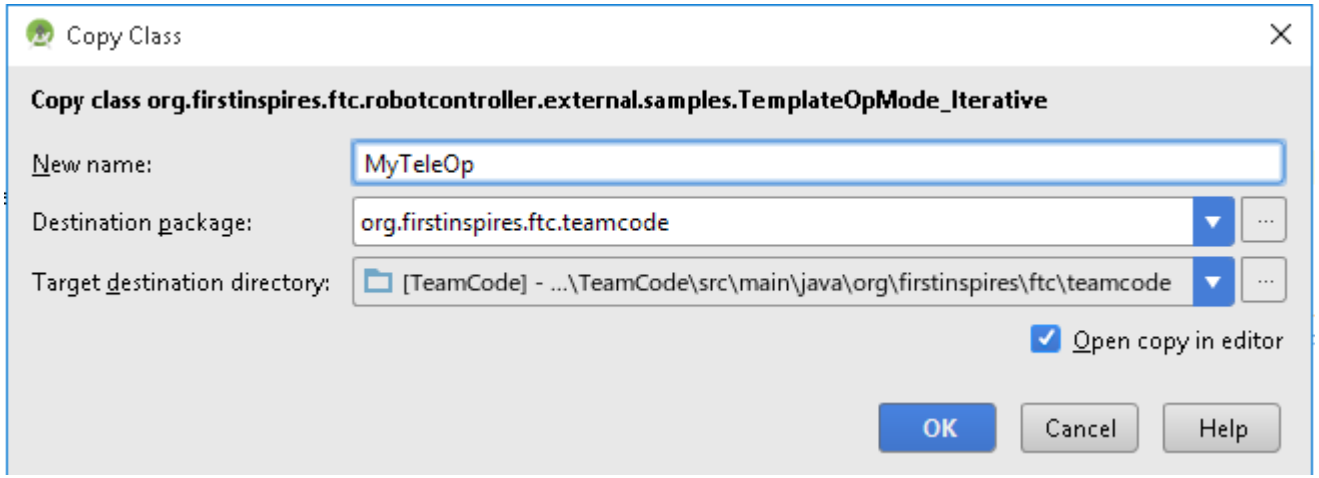


How to create a new OpMode (AS)

In the TeamCode folder, right click and select “Paste”



In the “Copy Class” dialog box, enter a name for the new OpMode



How to create a new OpMode (Android Studio)

The new OpMode is created:

```
@TeleOp(name="Template: Iterative OpMode", group="Iterative Opmode")
@Disabled
public class MyTeleOp extends OpMode
{
    /* Declare OpMode members. */
    private ElapsedTime runtime = new ElapsedTime();

    // private DcMotor leftMotor = null;
    // private DcMotor rightMotor = null;

    /*
```

Comment this line (add two slashes in front)
to enable the OpMode in the Driver Station menu

A simple “move forward” program

```
@TeleOp(name=”MyTeleOp”, group=”Iterative Opmode”)
// @Disabled
public class MyTeleOp extends OpMode {

    private DcMotor leftMotor = null;
    private DcMotor rightMotor = null;

    public MyTeleOp() { }

    public void init() {
        leftDrive  = hardware.get(DcMotor.class, “leftdrive”);
        rightDrive = hardware.get(DcMotor.class, “rightdrive”);
        leftDrive.setDirection(DcMotor.Direction.REVERSE);
        rightDrive.setDirection(DcMotor.Direction.FORWARD);
    }

    public void loop() {
        leftDrive.setPower(0.5);
        rightDrive.setPower(0.5);
    }

}
```

Teleop – joystick controls

During the “teleop” phase, drivers use gamepad controllers to command the robot

Driver 1

Driver 2

Teleop driving – tank controls

"Tank controls" allow each driving motor to be independently controlled

Driver 1

gamepad1.left_stick_y
(Left wheel)

gamepad1.right_stick_y
(Right wheel)

```
public void loop() {

    leftDrive.setPower( -gamepad1.left_stick_y );
    rightDrive.setPower( -gamepad1.right_stick_y );

}
```

Teleop driving – arcade controls

Steering control uses just one stick

Driver 1

gamepad1.left_stick_x
gamepad1.left_stick_y

```
public void loop() {

    double throttle = -gamepad1.left_stick_y;
    double turn     = gamepad1.left_stick_x;

    double leftspeed  = throttle – turn;
    double rightspeed = throttle + turn;

    leftDrive.setPower(leftspeed);
    rightDrive.setPower(rightspeed);

}
```

Teleop driving – arcade controls

```
public void loop() {

    double throttle = -gamepad1.left_stick_y;
    double turn     = gamepad1.left_stick_x;

    double leftspeed  = throttle – turn;
    double rightspeed = throttle + turn;

    leftDrive.setPower(leftspeed);
    rightDrive.setPower(rightspeed);

}
```

Robot will turn when left/right wheels are moving at different speeds

Y-axis is setting forward/backward speed of robot

X-axis is setting the difference between the two motors

Using joystick buttons

Each gamepad has 10 buttons available

Button can be read with “gamepad1.” and the name of the button

Using joystick buttons

Can use `if` statements to perform actions based on button setting:

```
private DcMotor arm = null;

public void init() {
    // ...
    arm = hardwareMap.get(DcMotor.class, “arm”);
    arm.setDirection(DcMotor.Direction.REVERSE);
}

public void loop() {

    // ...
    double armpower = 0;
    if (gamepad1.dpad_up) { armpower = 0.3; }
    arm.setPower(armpower);

}
```

Try: Add the “dpad_down” button to lower the arm

Dualing joysticks

Each robot is given two driver controllers

Many teams use one controller for driving, another for attachment control

Team 7172 prefers to duplicate all controls on both joysticks

Improving robot game scores

Use programming to automate as many of the robot operations as possible

“The less work the drivers have to do, the higher the scores”

- Automatic positioning of arms and lifts (“presets”)
- Limit switches
- Toggle items on and off
- Capturing game elements
- Driving / alignment / scoring

Controlling motor position with encoders

Motors have “encoders” that keep track of motor shaft position

The “setTargetPosition” command tells the motor to go to (and hold) a certain position

```
public void loop() {
    // ...
    arm.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    if (gamepad1.y) { arm.setTargetPosition(200); }
    arm.setPower(0.3);

}
```

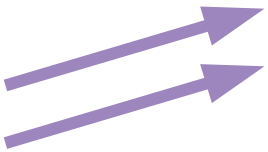
Try: Use button “x” to move the arm to position 80.

Controlling motor position with encoders

Set the motor's "zero position" in the init method:

```
public void init() {
    // ...
    arm = hardwareMap.get(DcMotor.class, "arm");
    arm.setDirection(DcMotorSimple.Direction.REVERSE);
    arm.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    arm.setTargetPosition(0);

}
```



Manually controlling arm using encoders

The highlighted code below manually lower the arm using the dpad_down button to adjust the target position downward by 10

```
arm.setMode(DcMotor.RunMode.RUN_TO_POSITION);
if (gamepad1.y) { arm.setTargetPosition(200); }
if (gamepad1.x) { arm.setTargetPosition(80); }

int armpos = arm.getCurrentPosition();
if (gamepad1.dpad_down) {
    arm.setTargetPosition(armpos - 10);
}

arm.setPower(0.4);
```

Coding style

Always keep opening and closing braces aligned

Indent + align code inside braces

Servo control

Servos are motors that can be told to move (and hold) a specific position

Servo positions are given as a number from 0 (fully counter-clockwise) to 1 (fully clockwise)

Servo control

To tell the servo to move to a position, use the `setPosition(...)` method:

```
claw.setPosition(0);    // fully counter-clockwise
claw.setPosition(1);    // fully clockwise
claw.setPosition(0.5);  // move to middle position
```

The servo will attempt to hold the requested position until told to move to a different one

General pattern for hardware devices

1. Name the device on the robot controller phone

2. Declare a variable in the OpMode

```
private Servo claw;
```

3. Look up the device via `hardwareMap` in `init()`

```
claw = hardwareMap.servo.get("claw");
```

4. Read/write values to the device in `loop()`

```
claw.setPosition(0.5);
```

Gamepad servo control

Make the servo controllable from the gamepad:

In the variables section, add a Servo variable:

```
Servo grab = null;
```

In the `init()` method, initialize from the hardware map:

```
grab = hardwareMap.servo.get("grab");
```

In the `loop()` method, control the arm from button A:

```
if (gamepad1.a) {
    grab.setPosition(0);
}
else {
    grab.setPosition(1);
}
```

Autonomous (Linear) OpModes

Autonomous programming

In the Autonomous portion of the game, there is no driver control (beyond pressing the stop button)

So, how do we get the robot to follow a sequence of commands?

Basic LinearOpMode Java code

```
@Autonomous(name="MyAutoOp", group="Autonomous")
// @Disabled
public class MyAutoOp extends OpMode {

    /* variables common to all methods */

    public void runOpMode() throws InterruptedException {
        /* set hardware variables */

        waitForStart();

        /* turn on motors */
        sleep(1000);   // drive for 1 second
        /* turn off motors */

    }

}
```

A simple “move forward” program autonomous

```
@Autonomous(name="MyAutoOp", group="Linear Opmode")
// @Disabled
public class MyAutoOp extends LinearOpMode {

    private DcMotor leftMotor = null;
    private DcMotor rightMotor = null;

    @Override
    public void runOpMode() throws InterruptedException {
        telemetry.addData("Status", "Initialized");
        telemetry.update();

        leftMotor  = hardwareMap.dcMotor.get("leftmotor");
        rightMotor = hardwareMap.dcMotor.get("rightmotor");
        rightMotor.setDirection(DcMotor.Direction.REVERSE);

        // Wait for the game to start (driver presses PLAY)
        waitForStart();

        if (opModeIsActive()) {
            leftMotor.setPower(0.5);
            rightMotor.setPower(0.5);
            sleep(2000);     // wait for two seconds
            leftMotor.setPower(0);
            rightMotor.setPower(0);
        }
    }
}
```

Multiple movements

```
public void runOpMode() throws InterruptedException {

    leftMotor  = hardwareMap.dcMotor.get("leftmotor");
    rightMotor = hardwareMap.dcMotor.get("rightmotor");
    rightMotor.setDirection(DcMotor.Direction.REVERSE);

    // Wait for the game to start (driver presses PLAY)
    waitForStart();

    // move forward for two seconds
    if (opModeIsActive()) {
        leftMotor.setPower(0.5);
        rightMotor.setPower(0.5);
        sleep(2000);     // wait for two seconds
        leftMotor.setPower(0);
        rightMotor.setPower(0);
    }

    // move backwards for one second
    if (opModeIsActive()) {
        leftMotor.setPower(-0.5);
        rightMotor.setPower(-0.5);
        sleep(1000);     // wait for one second
        leftMotor.setPower(0);
        rightMotor.setPower(0);
    }
}
```

Breaking code into functions

```
public void move(double lpower, double rpower, int msec)
        throws InterruptedException {
    if (opModeIsActive()) {
        leftMotor.setPower(lpower);
        rightMotor.setPower(rpower);
        sleep(msec);
        leftMotor.setPower(0);
        rightMotor.setPower(0);
    }
}


public void runOpMode() throws InterruptedException {
    leftMotor  = hardwareMap.dcMotor.get("leftmotor");
    rightMotor = hardwareMap.dcMotor.get("rightmotor");
    rightMotor.setDirection(DcMotor.Direction.REVERSE);

    // Wait for the game to start (driver presses PLAY)
    waitForStart();

    move(0.5, 0.5, 2000);    // move forward for two seconds
    move(-0.5, -0.5, 1000);  // move backwards for one second
    move(0.5, 0, 1500);      // turn to right for 1.5 seconds
}
```

Sensors

Hardware and Sensor types

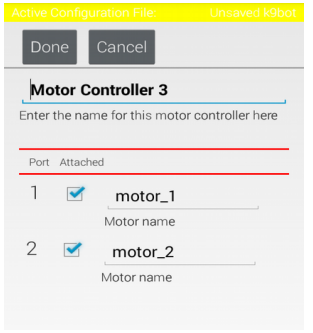
In general, everything we put on a robot has a corresponding class for programming in Java:

```
DcMotor
Servo


TouchSensor
GyroSensor
ColorSensor
UltrasonicSensor
```

General pattern for hardware devices

1. Name the device on the robot controller phone



2. Declare a variable in the OpMode

```
private DcMotor motor_1;
```

3. Look up the device via `hardwareMap` in `init()`

```
motor_1 = hardwareMap.dcMotor.get("motor_1");
```

4. Read/write values to the device in `loop()`

```
motor_1.setPower(0.5);
```

ftc_app Documentation

All of the sensor and hardware types are documented in *doc/javadoc/index.html* in the ftc_app directory.