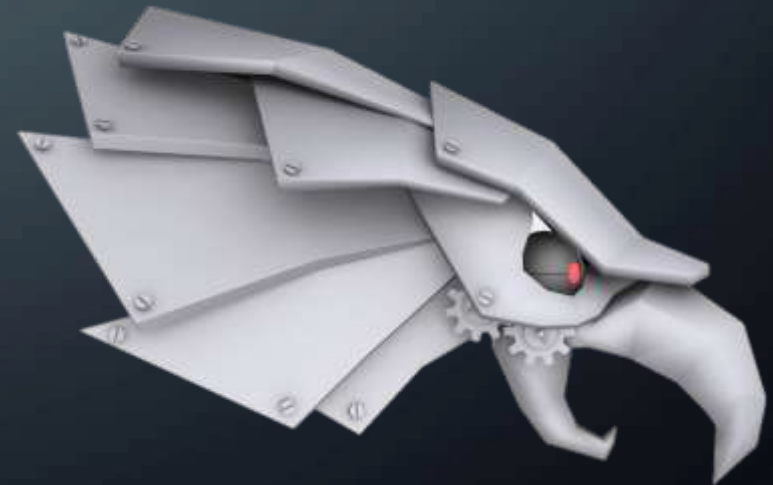


PROGRAMMING WITH AN DROID

EAGLE ROBOTICS TEAM 7373


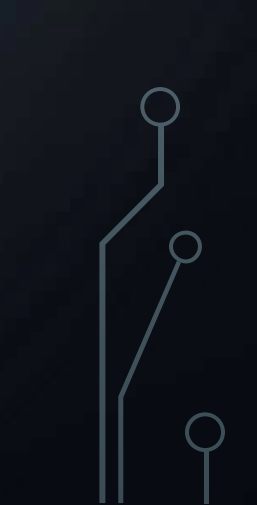


DISCLAIMER

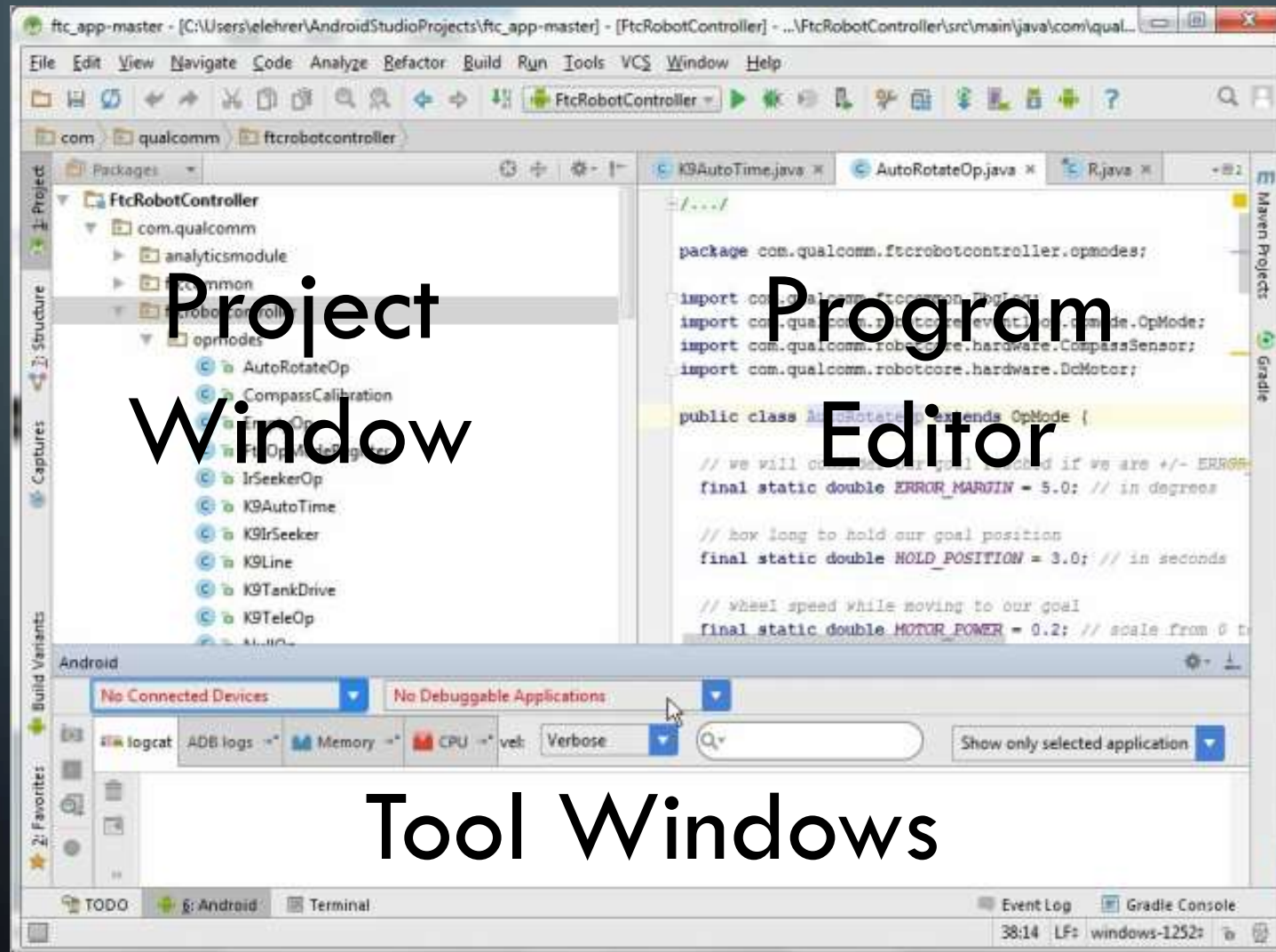
- This is only an overview
- We cannot cover every aspect of Android Studio
- If you have questions, contact us using the information provided in your handout or check out our code library: eaglerobotics.net/code



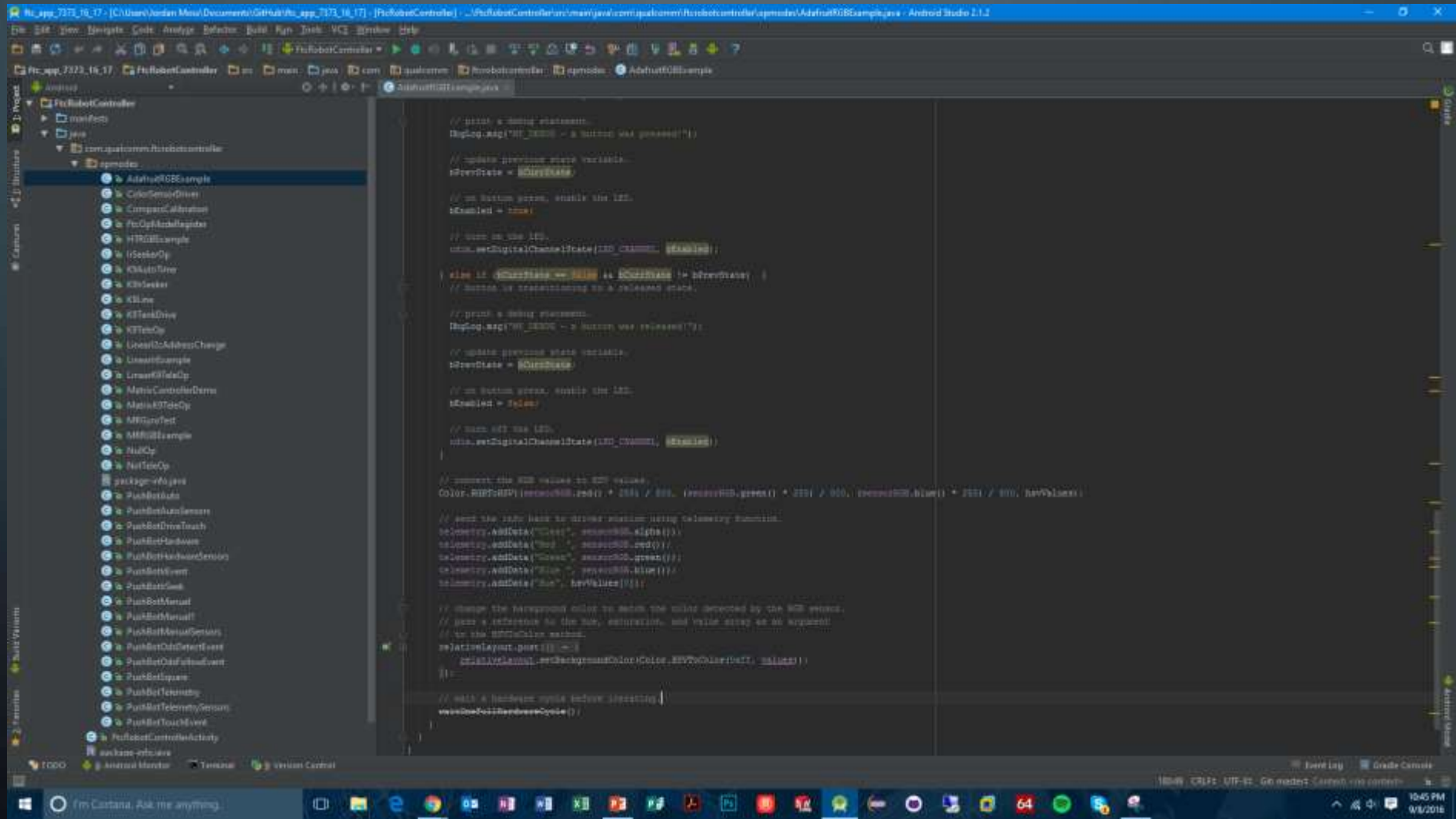
SYSTEM.OUT.PRINTLN("WELCOME!");

- What tools will you need to code in Android Studio?
 - JDK 8
 - Android Studio 2.1
 - The latest version of the ftc_app from GitHub
- 
- 

ANDROID STUDIO INTERFACE



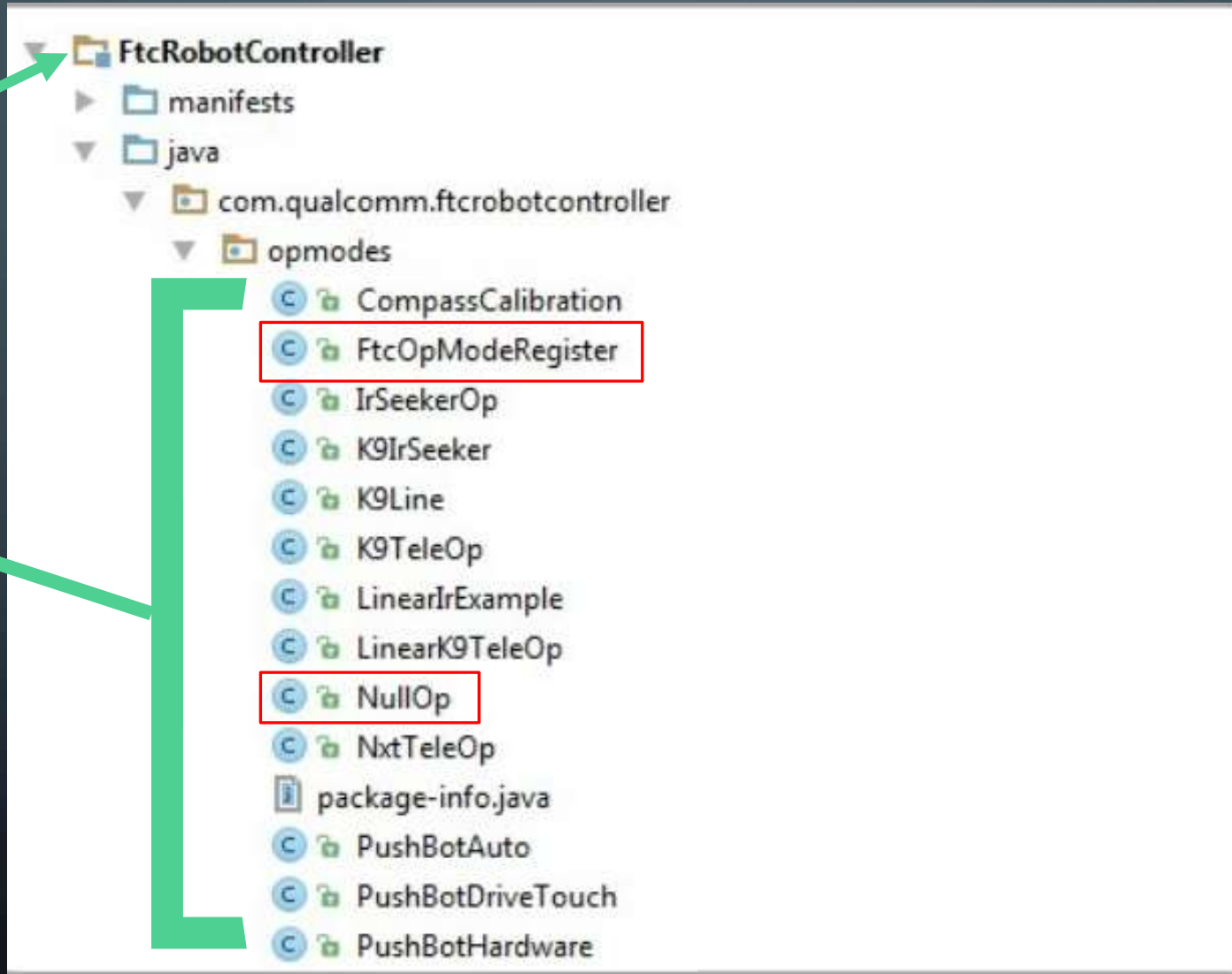
ANDROID STUDIO INTERFACE



NAVIGATING PROJECTS

Package

Classes



EDITING OPMODES

```
PushBotAuto.java x PushBotDriveTouch.java x PushBotHardware.java x PushBotIrSeek.java x

//-----
//
// PushBotAuto
//
//
// **
 * Extends the PushBotTelemetry and PushBotHardware classes to provide a basic
 * autonomous operational mode for the Push Bot.
 *
 * @author SSI Robotics
 * @version 2015-08-01-06-01
 */
public class PushBotAuto extends PushBotTelemetry
{
    //-----
    //
    // v_state
    //
    //-----
    // This class member remembers which state is currently active. When the
    // start method is called, the state will be initialized (0). When the loop
    // starts, the state will change from initialize to state_1. When state_1
    // actions are complete, the state will change to state_2. This implements
    // a state machine for the loop method.
    //-----
    int v_state = 0;
```

REGISTERING OPMODES

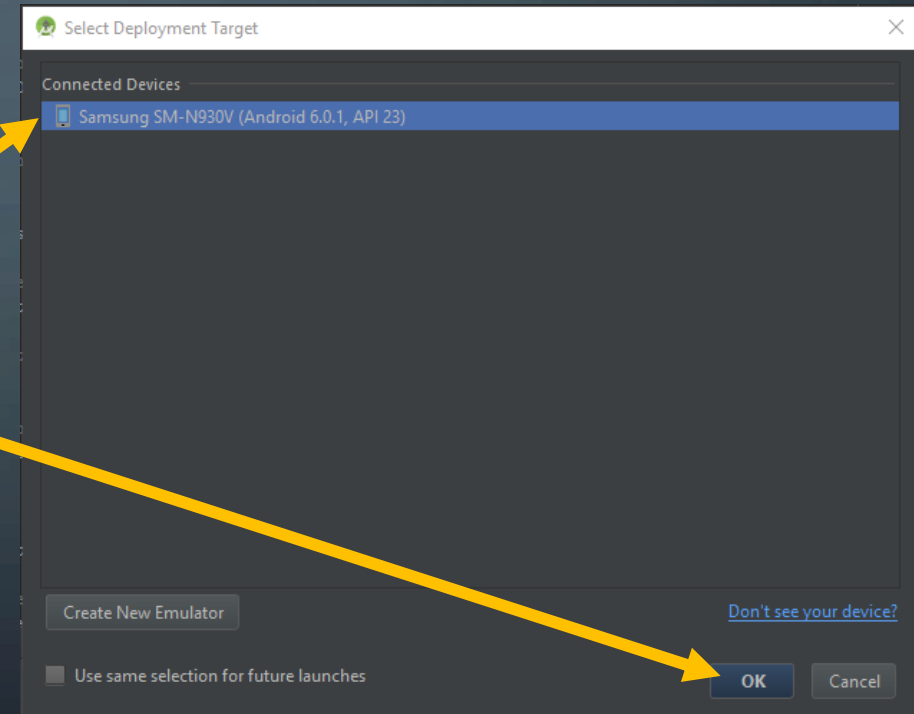
- In order to use an opmode, it must be enabled
- In order to do this you only need to comment out one line of code above the class header
- You can also change the labels that describe your opmode to keep things more organized
 - They can be organized by Autonomous/Teleop, Group, and Name

```
@Autonomous(name = "Concept: NullOp", group = "Concept")  
//@Disabled  
public class ConceptNullOp extends OpMode {
```

UPLOADING THE PROJECT



- Plug in Android Robot Controller
- Press Play Button in Menu
- Select Android Device to Deploy to
- Press OK
- The App Should Now be on your Android Robot Controller



BASIC JAVA SYNTAX

- Almost All Statements will end with a ;
 - i.e. `System.out.println("Hello World");`
- Every method or class will be surrounded by braces

- i.e.

```
public void method ()  
{  
    //Code goes here...  
}
```

BASIC JAVA SYNTAX

- Comments can be written using `//`
 - These are ignored by the compiler but help to document and organize code
 - i.e.

```
//set drive motor power to 1  
driveMotor.setPower(1);
```
 - These can also be quite helpful for testing small sections of the code at a time
- Comments can span multiple lines if the following from is used
 - ```
/*
* put an asterisk on every line
* then you can comment
* on multiple lines
*/
```

# JAVA VARIABLE TYPES

- Double, float, and int
  - These are the most commonly used number type in programming FTC robots
- boolean
  - This is a value that stores TRUE or FALSE
  - This is used for logic statements such as While, For, If, etc.
- Enumerated types
  - These are a specific type of variable that can hold explicit values like Go, Stop, Start...
  - This is not the same as a string

# JAVA LOGIC OPERTATORS

- && - And
- || - Or
- ! - Note
- == - Equals
- > - Greater Than
- < - Less Than
- >= - Greater Than or Equal To
- <= Less Than or Equal To
- != - Does Not Equal

# JAVA ITERATIVE AND LOGIC STATEMENTS

- `while(conditions) { ... }`
- `for(int i = 0; i <= 100; i++) { ... }`
- `if(condition) { ... } else { ... }`
- `switch(case){ case 1: ... break; case 2: ... break; default: ... break;`

# OPMODEISACTIVE() METHOD

- `opModelsActive();`
  - Returns a Boolean
- This method is very important to be able to stop your robot at any time even when you are running a loop
  - `while( opModelsActive() ) { ... }`

# BASIC FORM OF A PROGRAM

- There are 2 basic forms any robot program will take
  - Iterative
    - Init/start Method
      - Starts when init button is hit in driver station
      - Runs code once then waits for start to move to run the start and then loop method
    - loop Method
      - Runs repeatedly until stopped by the driver station
  - Linear
    - runOpMode Method
      - Runs through only once until stop button is pressed or timer runs out
      - waitForStart method is used to separate the initialization of the robot from the start of the program

# INITIALIZING THE ROBOT

- Create objects Servo and Motor
  - i.e.  
Motor leftdrive;  
Servo arm;
- Reference the variable to the hardware map
  - leftdrive = hardwareMap.dcMotor.get("motor name");  
arm = hardwareMap.servo.get("servo name");
  - Fill in motor name or servo name for the name used on the robot controller hardware map
    - Keep these names simple and consistent yet descriptive

# RUNNING THE ROBOT

- Use either object `gamepad1` or `gamepad2` to get input

| <b>gamepad1.</b>                                 | <b>Output</b> | <b>Range</b>  |
|--------------------------------------------------|---------------|---------------|
| <code>*_bumper</code>                            | Boolean       | True or False |
| <code>*_trigger</code>                           | Float         | [0,1]         |
| <code>*_stick_y</code> or <code>*_stick_x</code> | Float         | [-1,1]        |
| <code>dpad_*</code>                              | Boolean       | True or False |
| <code>a, b, x, y</code>                          | Boolean       | True or False |

- Fill in `*` for direction, Left, Right

# RUNNING THE ROBOT: MOTORS

- Running a motor
  - `motor.setPower(float);`
  - Fill in your motor variable for motor name and drive power value for float
  - This drive power can be from -1 to 1
    - If a value outside the bounds is sent, a null pointer error will be returned
    - To prevent this clip the range of the input into the motor
      - `power = Range.clip(power, -1, 1);`
  - Use the gamepad joystick to control your power variable
    - `float power = gamepad1.left_joystick_y;`  
`power = Range.clip(power, -1, 1);`  
`motor.setPower(power);`

# RUNNING THE ROBOT: SERVOS

- Running a servo
  - `servo.setPosition(float)`
  - Fill in servo variable name for servo and position value for float
  - The position can be from 0 to 1
    - The same range clipping technique can be used for servos
    - `position = Range.clip(position, 0, 1);`
  - Running a servo...
    - ```
if(gamepad1.dpad_up) {  
    servo.setPosition(1);  
} else {  
    servo.setPosition(0);  
}
```

A BASIC TELEOP

```
public void init()
{
    Servo arm;
    Motor leftDrive;
    Motor rightDrive;
    arm = hardwareMap.servo.get("arm");
    leftdrive = hardwareMap.motor.get("leftdrive");
    rightDrive = hardwareMap.motor.get("rightdrive");
}
```

A BASIC TELEOP

```
public void loop()
{
    //get boolean values from dpad and change position of servo
    if(gamepad1.dpad_up) {
        servo.setPosition(1);
    } else if(gamepad1.dpad_down){
        servo.setPosition(0);
    }

    //get power from left joystick and run motor
    float leftPower = gamepad1.left_joystick_y;
    power = Range.clip(leftPower, -1, 1);
    leftDrive.setPower(leftPower);

    float rightPower = gamepad1.right_joystick_y;
    power = Range.clip(leftPower, -1, 1);
    rightDrive.setPower(rightPower);
}
```

QUESTIONS?

- www.eaglerobotics.net/code
- Email us at team7373robotics@gmail.com or kk4jrq@gmail.com

