2023 Software Workshop: Intro to Java and FRC Programming

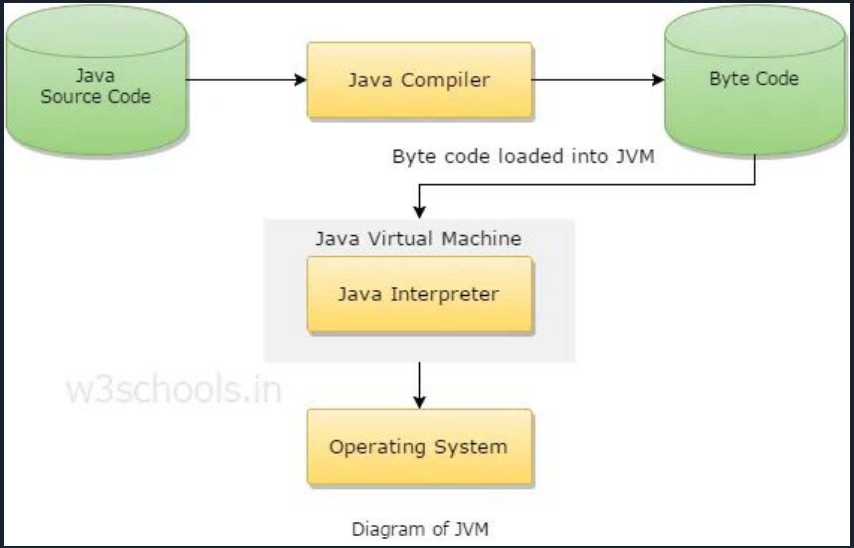
ROBOJACKETS

What is Java?

Java is a popular object oriented programming language.

It is platform independent and works on many different platforms (Mac, Windows, Linux etc…)

All Java programs are compiled into bytecode and passed into the Java Virtual Machine (JVM) which then interprets the bytecode into machine level language.



Object Oriented Programming (OOP) Basics

Java is object-oriented language

In OOP we create custom data types, called objects, which have whatever properties and behaviors you want them to have.

An object is an instance or a particular occurrence of a class.

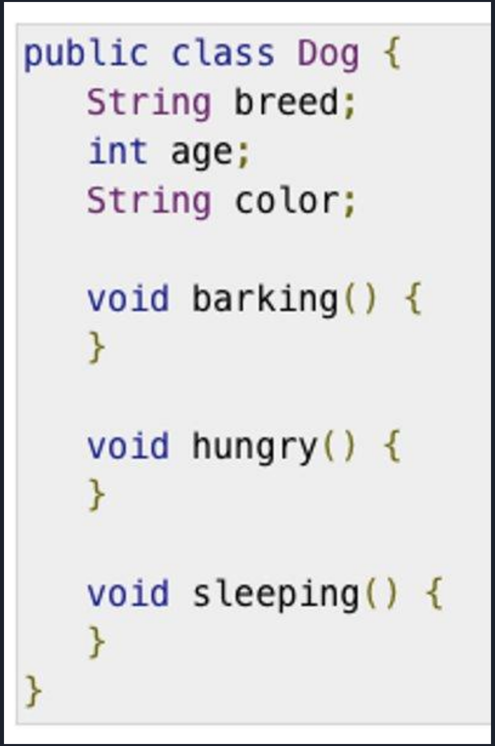
Example: Dog

Our Dog object has a breed, an age, and a color. These are attributes.

Our Dog can bark, be hungry, and fall asleep. These are its methods.

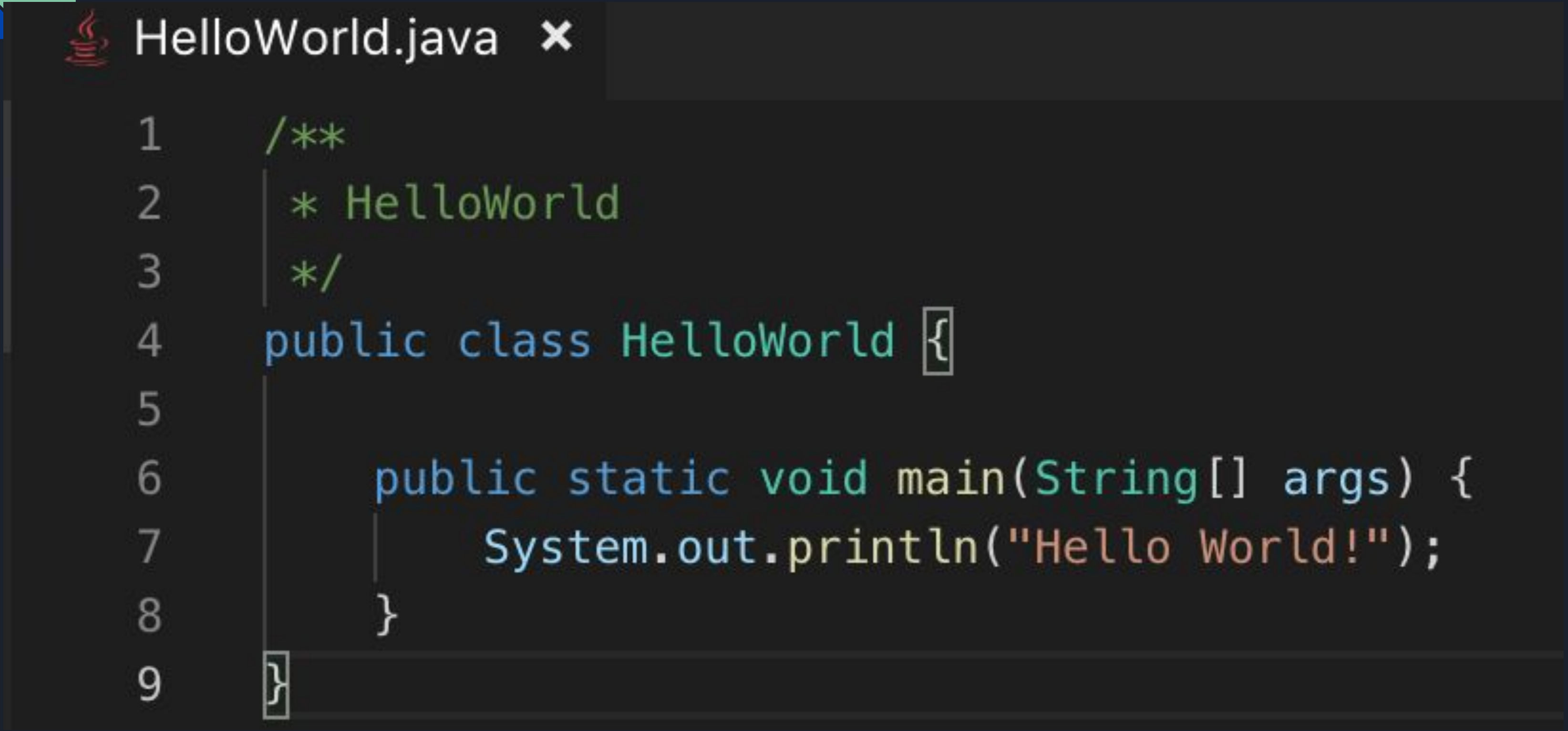
We can use Objects many ways when programming FRC Robots:

Command, Motors, Controllers, Sensors



```java
public class Dog {
    String breed;
    int age;
    String color;

    void barking() {
    }

    void hungry() {
    }

    void sleeping() {
    }
}
```

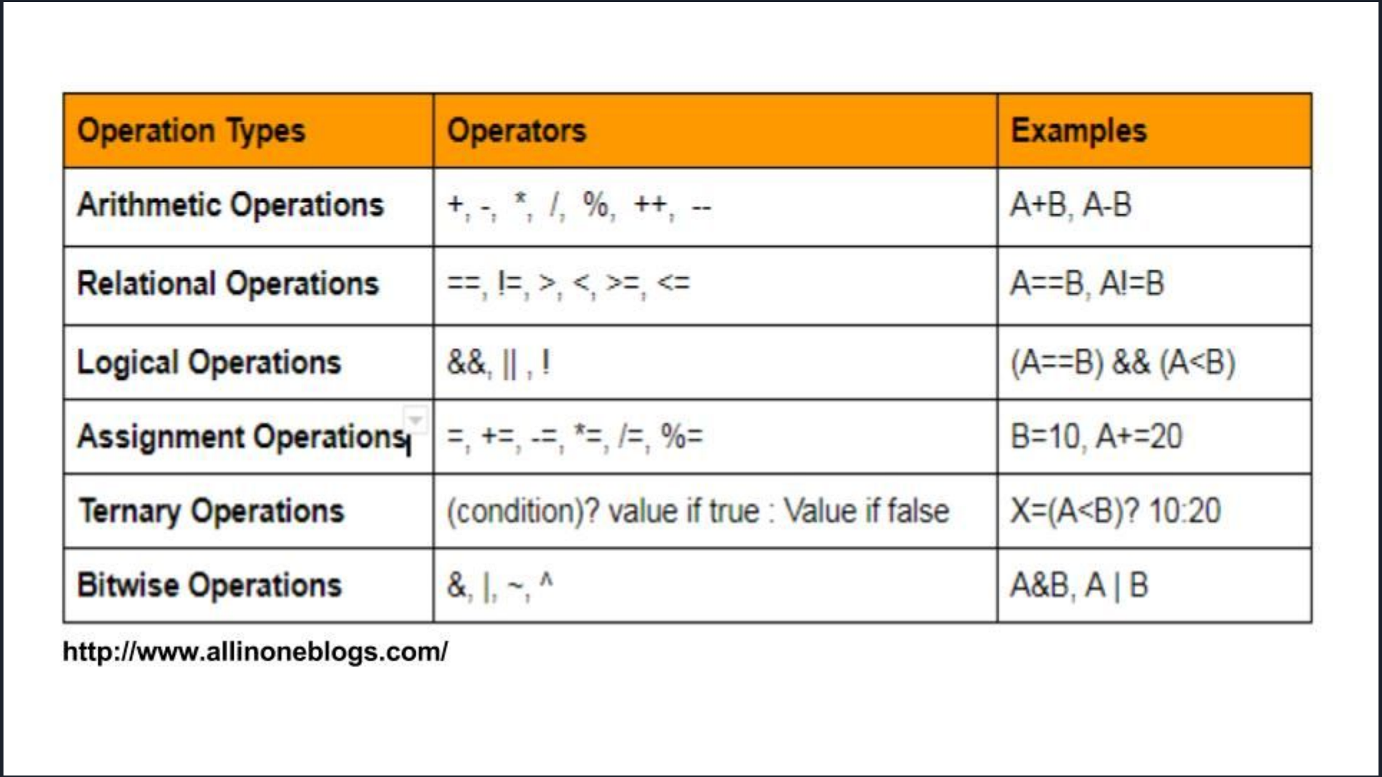
Hello World


HelloWorld.java

```java
/**
 * HelloWorld
 */
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Data Type	Default Value	Size	Range	Example
boolean	false	1 bit	true / false	boolean b = true;
char	‘\u0000’	2 bytes	0 to 65,535	char c = ‘a’;
byte	0	1 byte	-128 to 127	byte b = 10;
short	0	2 bytes	-32,768 to 32,767	short s = 11;
int	0	4 bytes	-2,147,483,648 to 2,147,483,647	int i = 10;
long	0L	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	long l = 1_00_000;
float	0.0f	4 bytes	1.40129846432481707e-45 to 3.40282346638528860e+38 (positive or negative)	float f = 10.3f;
double	0.0	8 bytes	4.94065645841246544e-324 to 1.79769313486623157 0e+308	double d = 11.124
String	null	n/a	n/a	String s = “Hello”;

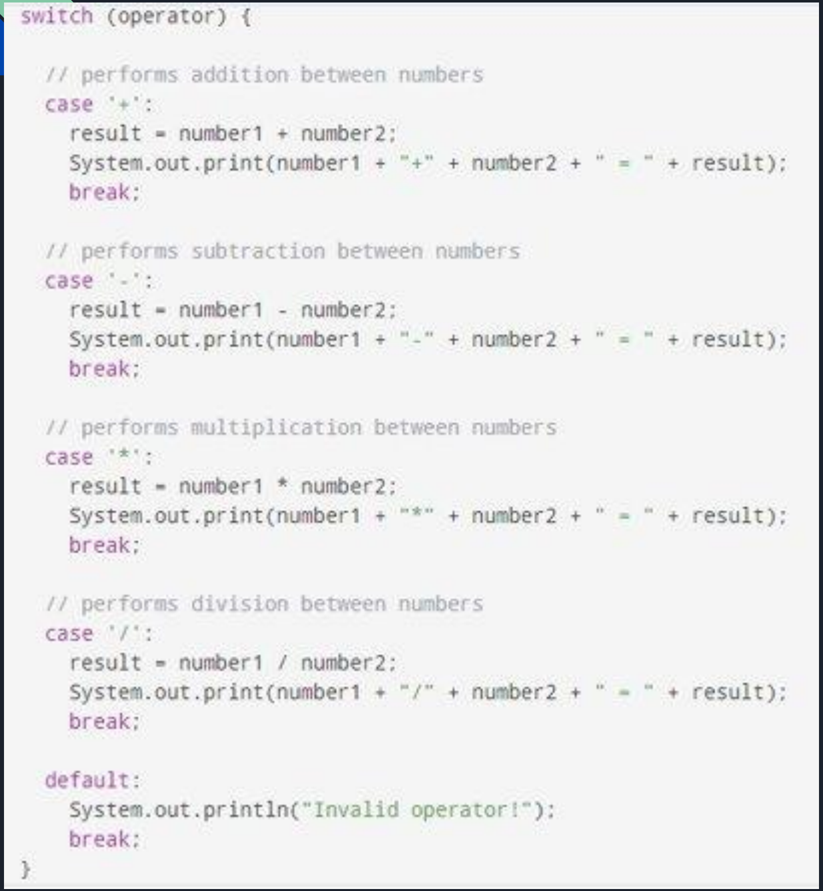
Operators in Java



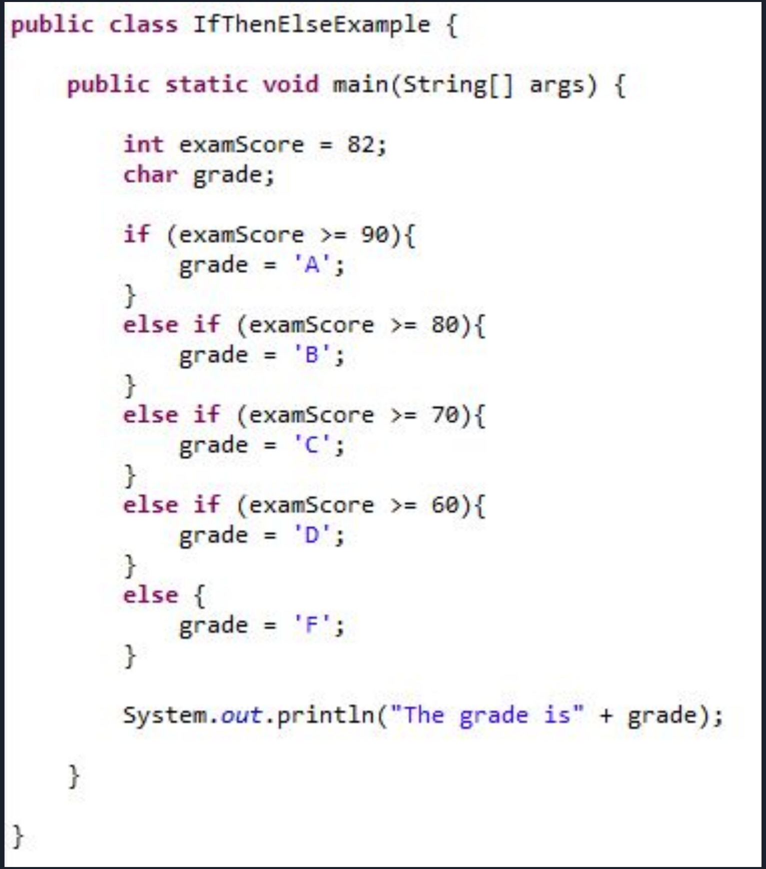
Operation Types	Operators	Examples
Arithmetic Operations	+, -, *, /, %, ++, --	A+B, A-B
Relational Operations	==, !=, >, <, >=, <=	A==B, A!=B
Logical Operations	&&, || , !	(A==B) && (A<B)
Assignment Operations	=, +=, -=, *=, /=, %=	B=10, A+=20
Ternary Operations	(condition)? value if true : Value if false	X=(A<B)? 10:20
Bitwise Operations	&, |, ~, ^	A&B, A | B

http://www.allinoneblogs.com/

Conditionals



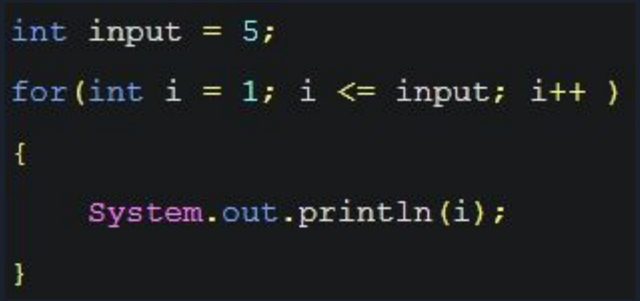
```
switch (operator) {

  // performs addition between numbers
  case '+':
    result = number1 + number2;
    System.out.print(number1 + "+" + number2 + " = " + result);
    break;

  // performs subtraction between numbers
  case '-':
    result = number1 - number2;
    System.out.print(number1 + "-" + number2 + " = " + result);
    break;

  // performs multiplication between numbers
  case '*':
    result = number1 * number2;
    System.out.print(number1 + "*" + number2 + " = " + result);
    break;

  // performs division between numbers
  case '/':
    result = number1 / number2;
    System.out.print(number1 + "/" + number2 + " = " + result);
    break;

  default:
    System.out.println("Invalid operator!");
    break;
}
```



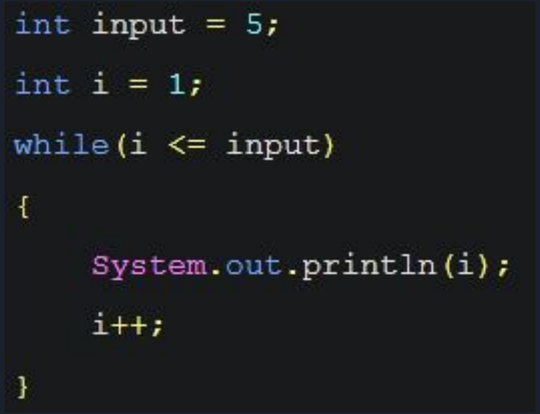
```
public class IfThenElseExample {

    public static void main(String[] args) {

        int examScore = 82;
        char grade;

        if (examScore >= 90){
            grade = 'A';
        }
        else if (examScore >= 80){
            grade = 'B';
        }
        else if (examScore >= 70){
            grade = 'C';
        }
        else if (examScore >= 60){
            grade = 'D';
        }
        else {
            grade = 'F';
        }

        System.out.println("The grade is" + grade);

    }

}
```

Loops

- While Loops
  - Loops that continue while a certain condition is true
  - Typically used when you want the loop to end on a certain condition and you are not sure how many iterations it will take to reach
- For Loops
  - Also continues while a certain condition is true
  - Typically used to loop a fixed number of times



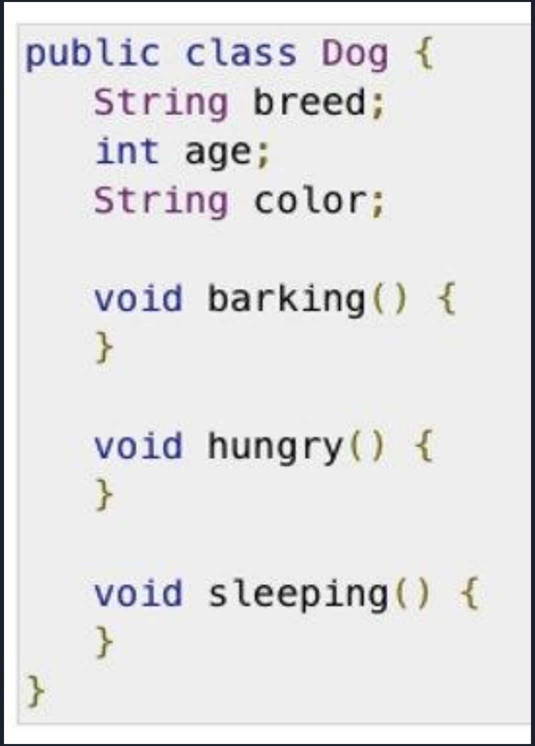
```java
int input = 5;
for(int i = 1; i <= input; i++ )
{
    System.out.println(i);
}
```



```java
int input = 5;
int i = 1;
while(i <= input)
{
    System.out.println(i);
    i++;
}
```

Classes

A class is a blueprint for an object. Take our earlier dog class for example, it demonstrates that our theoretical dog as several properties (breed, age, color) and actions (hungry, barking, sleeping), but in order to put it into fruition, we need to make on object.



```
public class Dog {
   String breed;
   int age;
   String color;

   void barking() {
   }

   void hungry() {
   }

   void sleeping() {
   }
}
```

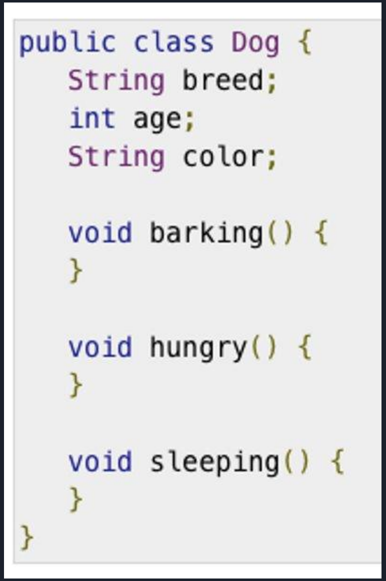
Objects

Objects are instances of classes. Classes act as a blueprint for an object, they define what an object can or cannot have / do.

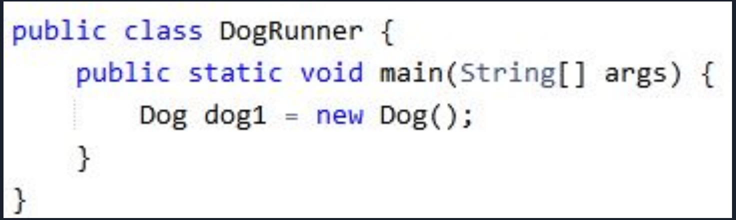
Imagine a factory that's producing toy dogs, the class to the right on top contains the properties and actions that this dog toy may or may not have. In this case an object would be one specific toy dog produced using that class in the top right.

Line 3 in the bottom example is a specific dog object called dog1.

Right now this dog does not have any of its properties set, how would we do this?



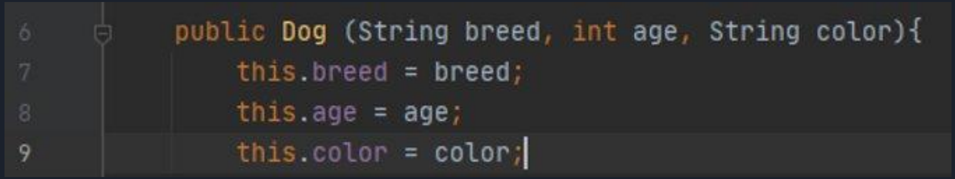
```
public class Dog {
   String breed;
   int age;
   String color;

   void barking() {
   }

   void hungry() {
   }

   void sleeping() {
   }
}
```



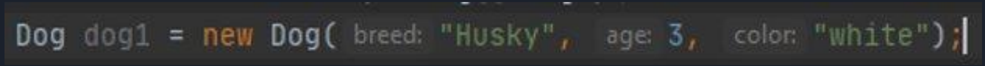
```
public class DogRunner {
    public static void main(String[] args) {
        Dog dog1 = new Dog();
    }
}
```

Constructors

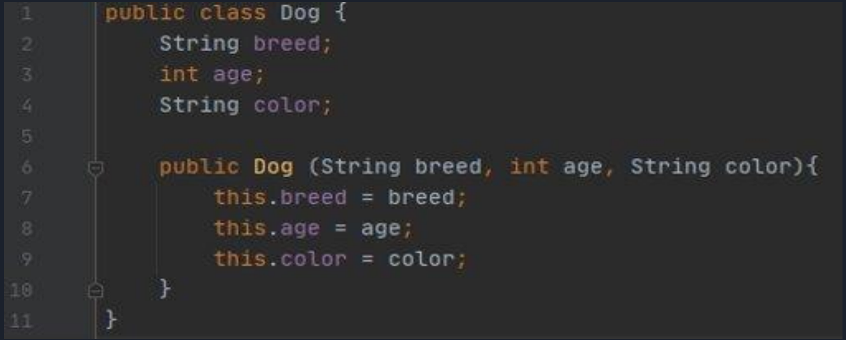
In order to create our objects with specific properties, we need to add something to our class called a constructor. Like the name sounds, this is a special method we would call whenever we need to construct or create a new instance of our class. See how this dog constructor related to its class.



```java
    public Dog (String breed, int age, String color){
        this.breed = breed;
        this.age = age;
        this.color = color;
```



```java
Dog dog1 = new Dog( breed: "Husky",  age: 3,  color: "white");
```



```java
public class Dog {
    String breed;
    int age;
    String color;

    public Dog (String breed, int age, String color){
        this.breed = breed;
        this.age = age;
        this.color = color;
    }
}
```

FRC Background

WPI Lib

- The WPI Robotics Library (WPILib) is the standard software library provided for teams to write code for their FRC® robots in either C++ or Java
- https://docs.wpilib.org/ is a fantastic resource for FRC teams which has sections from programming basics to software tools (such as driver station) and guides on advanced programming. (Vision Processing and Network Tables for example)
- WPI (Worcester Polytechnic Institute) gives out a modified version of VS Code to teams so that they can easily develop programs for their robots. Installing this version of VS Code is essential for programming your FRC Robot

Getting Started with WPI

1. Go to https://github.com/wpilibsuite/allwpilib/releases. Currently 2021.3.1, but 2022.1.1 should be out of beta soon
2. Find the most recent version and download it to your computer, making sure to select the right operating system and architecture. The installer should download as a disk image.
3. https://docs.wpilib.org/en/latest/docs/zero-to-robot/step-2/wpilib-setup.html provides an excellent guide for opening the disk image and stepping through in installer in all the popular OSes (Windows, Mac, Linux)
4. Now, your special version of VSCode should be equipped with the WPILib extension, which allows you to create a new project, build code, and deploy code.

Game Sections

Autonomous (15s)

No human input allowed

Drive team selects an autonomous command before the match

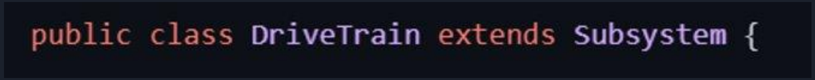
The selected command is the only thing that runs in the first 15 sec

Teleop (135s)

Create commands that can be called by pressing one or many buttons or by moving a joystick or some other toggle

During the last 20-30 seconds (ENDGAME) there’s often a high value objective like climbing a part of the field, or lifting your bot up

Subsystems



```java
public class DriveTrain extends Subsystem {
```

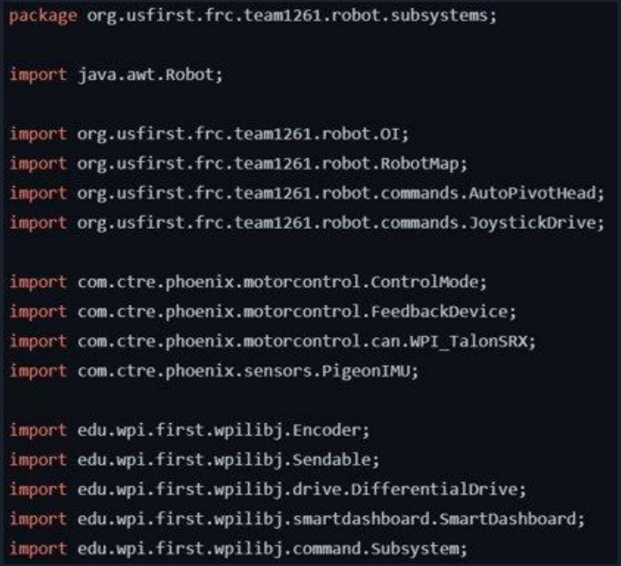
Building blocks of the robot

Declare the motors and sensors that will be used

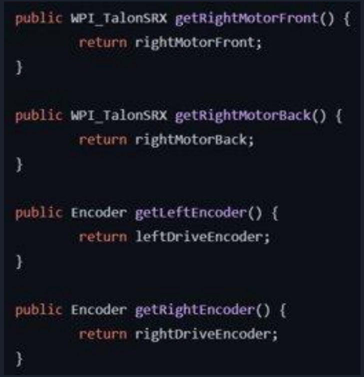
Create getters and setters to get the status of said motors and sensors

Add methods to make the motors and sensors do cool things

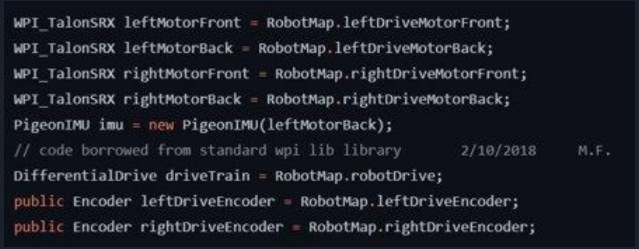
Add them to RobotMap



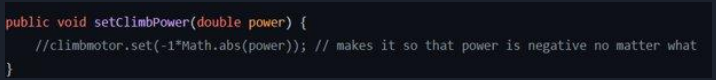
```java
package org.usfirst.frc.team1261.robot.subsystems;

import java.awt.Robot;

import org.usfirst.frc.team1261.robot.OI;
import org.usfirst.frc.team1261.robot.RobotMap;
import org.usfirst.frc.team1261.robot.commands.AutoPivotHead;
import org.usfirst.frc.team1261.robot.commands.JoystickDrive;

import com.ctre.phoenix.motorcontrol.ControlMode;
import com.ctre.phoenix.motorcontrol.FeedbackDevice;
import com.ctre.phoenix.motorcontrol.can.WPI_TalonSRX;
import com.ctre.phoenix.sensors.PigeonIMU;

import edu.wpi.first.wpilibj.Encoder;
import edu.wpi.first.wpilibj.Sendable;
import edu.wpi.first.wpilibj.drive.DifferentialDrive;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import edu.wpi.first.wpilibj.command.Subsystem;
```



```java
public WPI_TalonSRX getRightMotorFront() {
        return rightMotorFront;
}

public WPI_TalonSRX getRightMotorBack() {
        return rightMotorBack;
}

public Encoder getLeftEncoder() {
        return leftDriveEncoder;
}

public Encoder getRightEncoder() {
        return rightDriveEncoder;
}
```

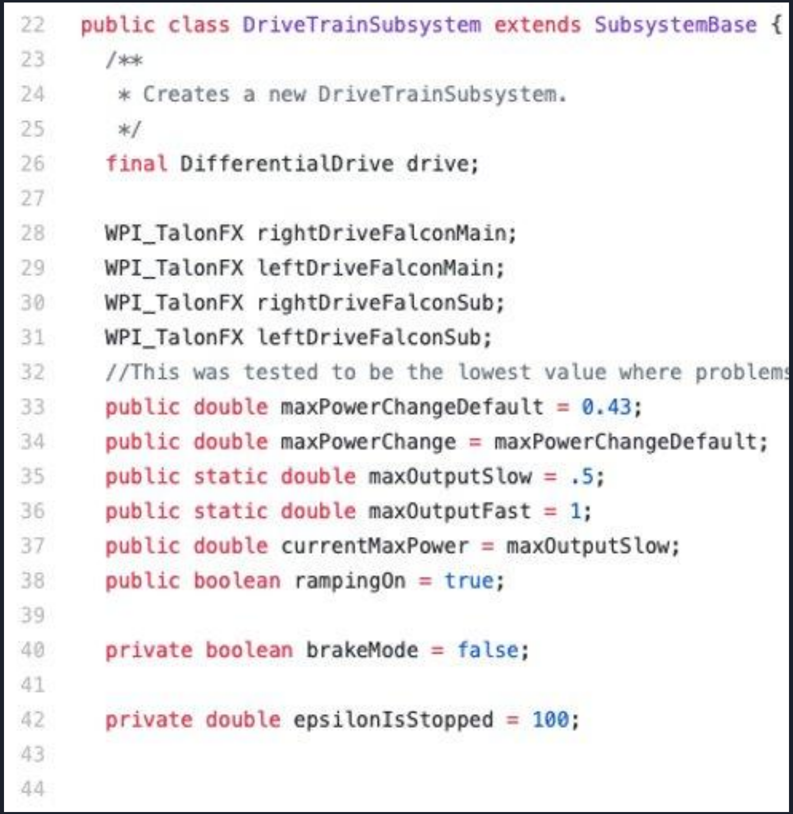


```java
WPI_TalonSRX leftMotorFront = RobotMap.leftDriveMotorFront;
WPI_TalonSRX leftMotorBack = RobotMap.leftDriveMotorBack;
WPI_TalonSRX rightMotorFront = RobotMap.rightDriveMotorFront;
WPI_TalonSRX rightMotorBack = RobotMap.rightDriveMotorBack;
PigeonIMU imu = new PigeonIMU(leftMotorBack);
// code borrowed from standard wpi lib library      2/10/2018     M.F.
DifferentialDrive driveTrain = RobotMap.robotDrive;
public Encoder leftDriveEncoder = RobotMap.leftDriveEncoder;
public Encoder rightDriveEncoder = RobotMap.rightDriveEncoder;
```

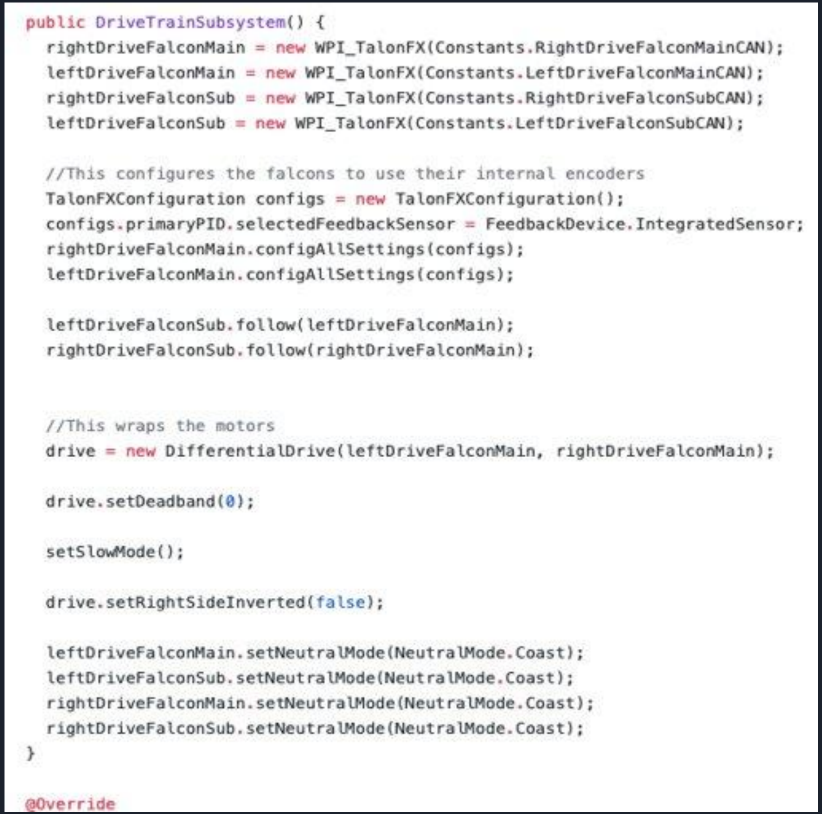


```java
public void setClimbPower(double power) {
    //climbmotor.set(-1*Math.abs(power)); // makes it so that power is negative no matter what
}
```

Drivetrain Subsystem Example

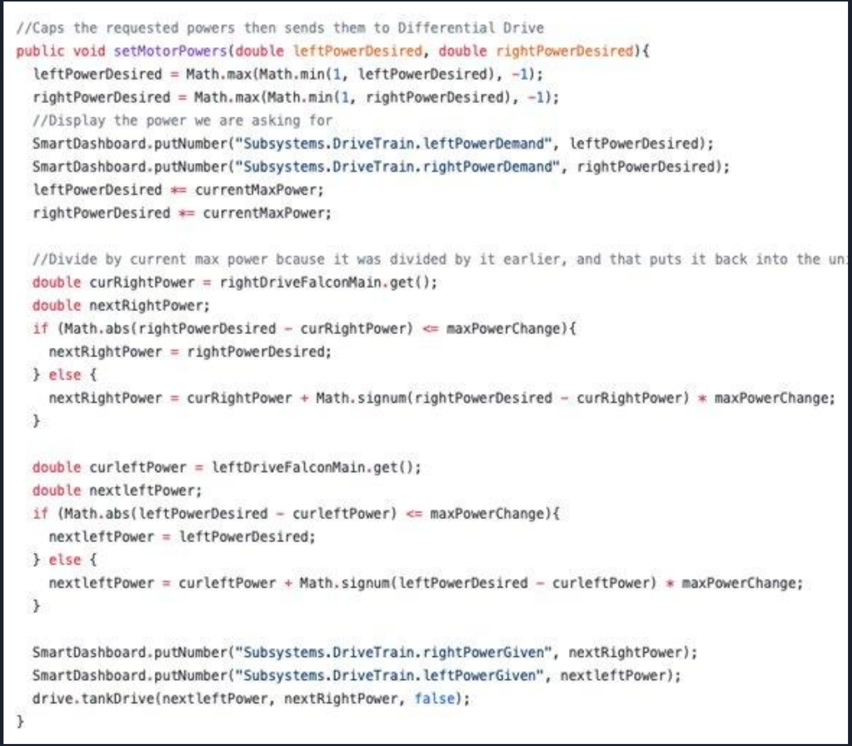


```java
public class DriveTrainSubsystem extends SubsystemBase {
  /**
   * Creates a new DriveTrainSubsystem.
   */
  final DifferentialDrive drive;

  WPI_TalonFX rightDriveFalconMain;
  WPI_TalonFX leftDriveFalconMain;
  WPI_TalonFX rightDriveFalconSub;
  WPI_TalonFX leftDriveFalconSub;
  //This was tested to be the lowest value where problems
  public double maxPowerChangeDefault = 0.43;
  public double maxPowerChange = maxPowerChangeDefault;
  public static double maxOutputSlow = .5;
  public static double maxOutputFast = 1;
  public double currentMaxPower = maxOutputSlow;
  public boolean rampingOn = true;

  private boolean brakeMode = false;

  private double epsilonIsStopped = 100;


```

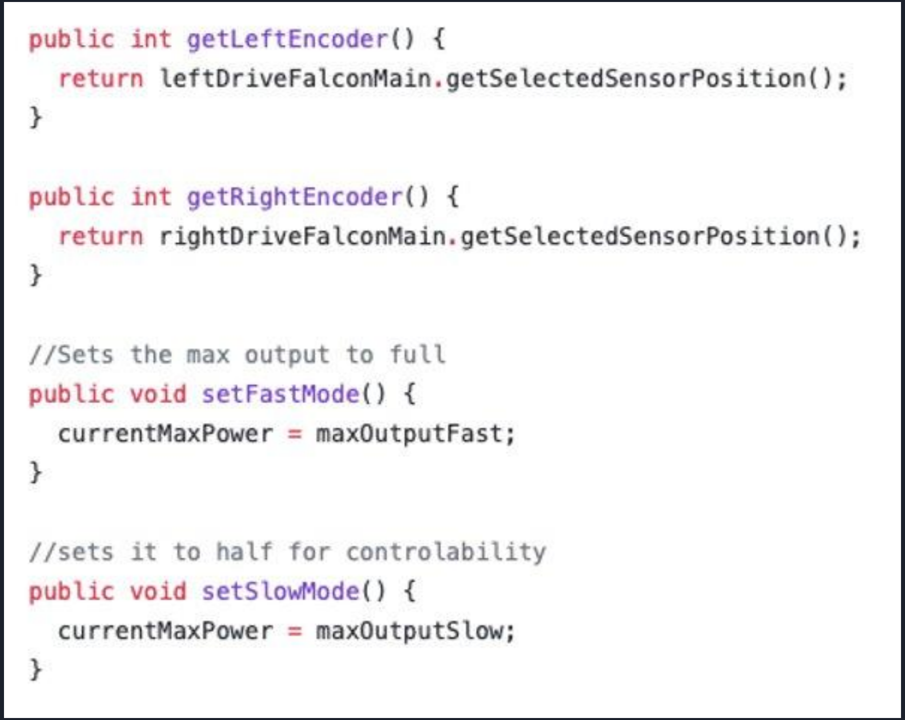


```java
public DriveTrainSubsystem() {
  rightDriveFalconMain = new WPI_TalonFX(Constants.RightDriveFalconMainCAN);
  leftDriveFalconMain = new WPI_TalonFX(Constants.LeftDriveFalconMainCAN);
  rightDriveFalconSub = new WPI_TalonFX(Constants.RightDriveFalconSubCAN);
  leftDriveFalconSub = new WPI_TalonFX(Constants.LeftDriveFalconSubCAN);

  //This configures the falcons to use their internal encoders
  TalonFXConfiguration configs = new TalonFXConfiguration();
  configs.primaryPID.selectedFeedbackSensor = FeedbackDevice.IntegratedSensor;
  rightDriveFalconMain.configAllSettings(configs);
  leftDriveFalconMain.configAllSettings(configs);

  leftDriveFalconSub.follow(leftDriveFalconMain);
  rightDriveFalconSub.follow(rightDriveFalconMain);


  //This wraps the motors
  drive = new DifferentialDrive(leftDriveFalconMain, rightDriveFalconMain);

  drive.setDeadband(0);

  setSlowMode();

  drive.setRightSideInverted(false);

  leftDriveFalconMain.setNeutralMode(NeutralMode.Coast);
  leftDriveFalconSub.setNeutralMode(NeutralMode.Coast);
  rightDriveFalconMain.setNeutralMode(NeutralMode.Coast);
  rightDriveFalconSub.setNeutralMode(NeutralMode.Coast);
}

@Override
```

More Drivetrain Subsystem



```java
//Caps the requested powers then sends them to Differential Drive
public void setMotorPowers(double leftPowerDesired, double rightPowerDesired){
  leftPowerDesired = Math.max(Math.min(1, leftPowerDesired), -1);
  rightPowerDesired = Math.max(Math.min(1, rightPowerDesired), -1);
  //Display the power we are asking for
  SmartDashboard.putNumber("Subsystems.DriveTrain.leftPowerDemand", leftPowerDesired);
  SmartDashboard.putNumber("Subsystems.DriveTrain.rightPowerDemand", rightPowerDesired);
  leftPowerDesired *= currentMaxPower;
  rightPowerDesired *= currentMaxPower;

  //Divide by current max power bcause it was divided by it earlier, and that puts it back into the un
  double curRightPower = rightDriveFalconMain.get();
  double nextRightPower;
  if (Math.abs(rightPowerDesired - curRightPower) <= maxPowerChange){
    nextRightPower = rightPowerDesired;
  } else {
    nextRightPower = curRightPower + Math.signum(rightPowerDesired - curRightPower) * maxPowerChange;
  }

  double curleftPower = leftDriveFalconMain.get();
  double nextleftPower;
  if (Math.abs(leftPowerDesired - curleftPower) <= maxPowerChange){
    nextleftPower = leftPowerDesired;
  } else {
    nextleftPower = curleftPower + Math.signum(leftPowerDesired - curleftPower) * maxPowerChange;
  }

  SmartDashboard.putNumber("Subsystems.DriveTrain.rightPowerGiven", nextRightPower);
  SmartDashboard.putNumber("Subsystems.DriveTrain.leftPowerGiven", nextleftPower);
  drive.tankDrive(nextleftPower, nextRightPower, false);
}
```



```java
public int getLeftEncoder() {
  return leftDriveFalconMain.getSelectedSensorPosition();
}

public int getRightEncoder() {
  return rightDriveFalconMain.getSelectedSensorPosition();
}

//Sets the max output to full
public void setFastMode() {
  currentMaxPower = maxOutputFast;
}

//sets it to half for controlability
public void setSlowMode() {
  currentMaxPower = maxOutputSlow;
}
```

Aside: Cool things to look into

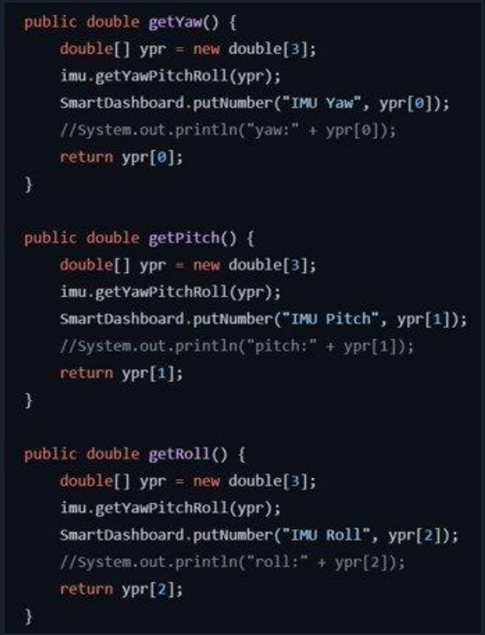
IMU

- Gives you roll, pitch, and yaw (your 3D orientation)

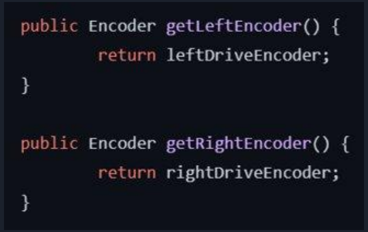
Encoders

- Stores distance / steps travel

Both of these are very helpful for determining your position when writing code for the autonomous portion.



```
public double getYaw() {
    double[] ypr = new double[3];
    imu.getYawPitchRoll(ypr);
    SmartDashboard.putNumber("IMU Yaw", ypr[0]);
    //System.out.println("yaw:" + ypr[0]);
    return ypr[0];
}

public double getPitch() {
    double[] ypr = new double[3];
    imu.getYawPitchRoll(ypr);
    SmartDashboard.putNumber("IMU Pitch", ypr[1]);
    //System.out.println("pitch:" + ypr[1]);
    return ypr[1];
}

public double getRoll() {
    double[] ypr = new double[3];
    imu.getYawPitchRoll(ypr);
    SmartDashboard.putNumber("IMU Roll", ypr[2]);
    //System.out.println("roll:" + ypr[2]);
    return ypr[2];
}
```



```
public Encoder getLeftEncoder() {
        return leftDriveEncoder;
}

public Encoder getRightEncoder() {
        return rightDriveEncoder;
}
```

PID (Proportion, Integral, Derivative)

The bottom equation may look scary, but trust me it's really not, and it is quite useful.

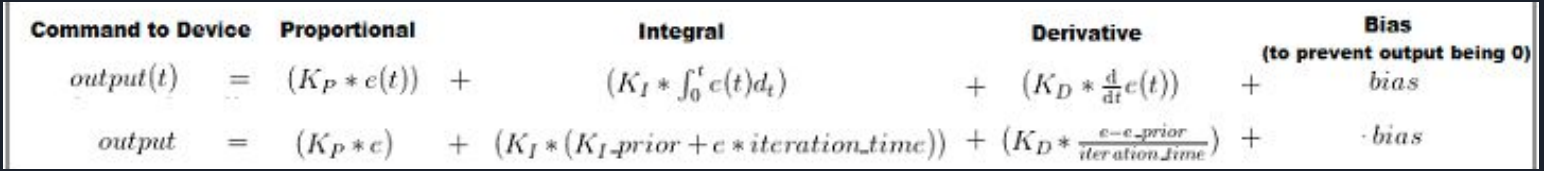
Sensors typically drift over time, so PID helps us get to our target speed / distance.

Its made up of three parts that work on the input from a certain sensor: Proportion, Integral, and Derivative. The hardest part of PID involves tuning the P, I and D scalars that end up being multiplied by their respective parts.

Helpful link for tuning, and more info:
http://robotsforroboticists.com/pid-control/

Pseudocode:

```
    error_prior = 0
    integral_prior = 0
    KP = Some value you need to come up (see tuning section below)
    KI = Some value you need to come up (see tuning section below)
    KD = Some value you need to come up (see tuning section below)
    bias = 0 (see below)

    while(1) {
        error = desired_value – actual_value
        integral = integral_prior + error * iteration_time
        derivative = (error – error_prior) / iteration_time
        output = KP*error + KI*integral + KD*derivative + bias
        error_prior = error
        integral_prior = integral
        sleep(iteration_time)
    }
```



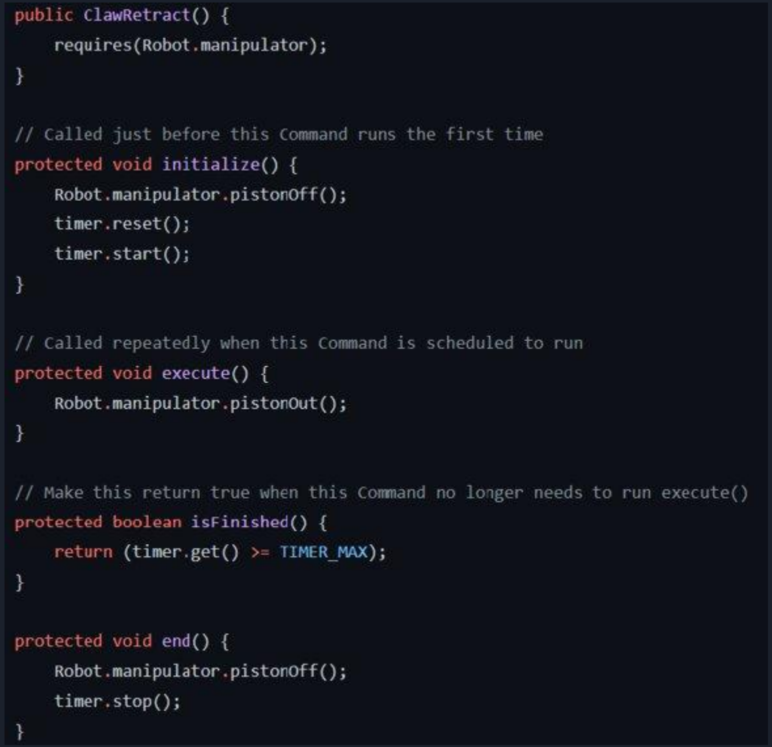
Command to Device		Proportional		Integral		Derivative		Bias (to prevent output being 0)
$output(t)$	$=$	$(K_P * e(t))$	$+$	$(K_I * \int_0^t e(t) d_t)$	$+$	$(K_D * \frac{d}{dt} e(t))$	$+$	$bias$
$output$	$=$	$(K_P * e)$	$+$	$(K_I * (K_I\_prior + e * iteration\_time))$	$+$	$(K_D * \frac{e - e\_prior}{iteration\_time})$	$+$	$bias$

Commands : Teleop

Call methods within various subsystems to make the bot do things!

These commands are mapped to controller buttons / joysticks

Multiple commands can be run at the same time



```java
public ClawRetract() {
    requires(Robot.manipulator);
}

// Called just before this Command runs the first time
protected void initialize() {
    Robot.manipulator.pistonOff();
    timer.reset();
    timer.start();
}

// Called repeatedly when this Command is scheduled to run
protected void execute() {
    Robot.manipulator.pistonOut();
}

// Make this return true when this Command no longer needs to run execute()
protected boolean isFinished() {
    return (timer.get() >= TIMER_MAX);
}

protected void end() {
    Robot.manipulator.pistonOff();
    timer.stop();
}
```

Commands : Autonomous

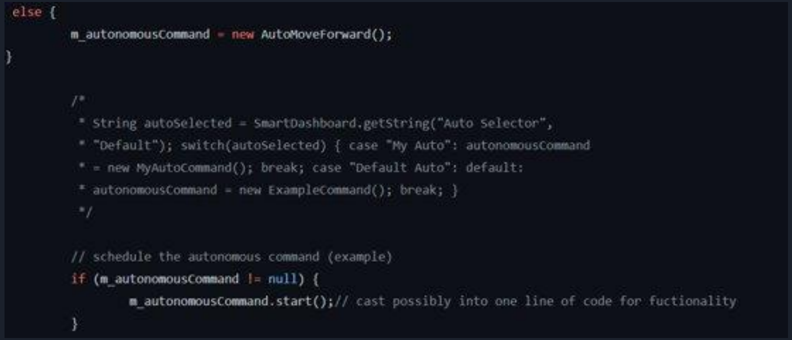
Still calling methods within various subsystems to control our bot

Mapped to Smart Dashboard inputs (used by Drive Team during a match)

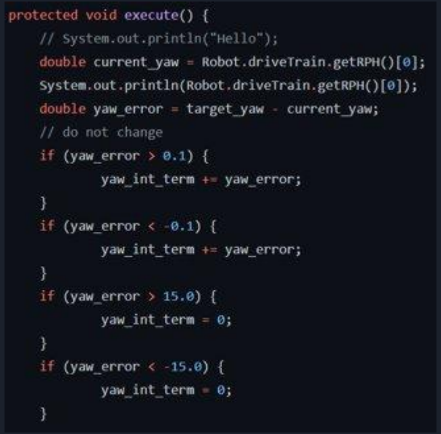
Can also be treated like teleop command but nothing else can run at the same time.

Make sure your Robot explicitly sets the autonomous command!

If this is null then none of your commands will execute :(



```java
else {
        m_autonomousCommand = new AutoMoveForward();
}


        /*
         * 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();// cast possibly into one line of code for fuctionality
        }
```



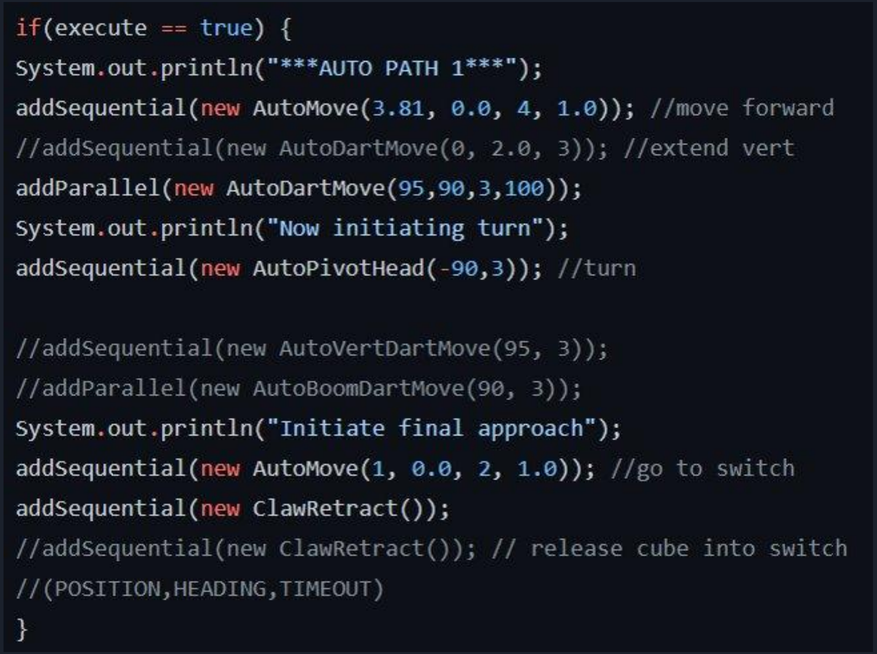
```java
protected void execute() {
    // System.out.println("Hello");
    double current_yaw = Robot.driveTrain.getRPH()[0];
    System.out.println(Robot.driveTrain.getRPH()[0]);
    double yaw_error = target_yaw - current_yaw;
    // do not change
    if (yaw_error > 0.1) {
            yaw_int_term += yaw_error;
    }
    if (yaw_error < -0.1) {
            yaw_int_term += yaw_error;
    }
    if (yaw_error > 15.0) {
            yaw_int_term = 0;
    }
    if (yaw_error < -15.0) {
            yaw_int_term = 0;
    }
```

Modular Autonomous Programs!

You can create complex paths or commands by adding sequential or parallel commands to run during the execution of a given command.

AddSequential - the command is run after the completion of the previous command added

AddParallel - the command is run at the same time as the previous command added



```
if(execute == true) {
System.out.println("***AUTO PATH 1***");
addSequential(new AutoMove(3.81, 0.0, 4, 1.0)); //move forward
//addSequential(new AutoDartMove(0, 2.0, 3)); //extend vert
addParallel(new AutoDartMove(95,90,3,100));
System.out.println("Now initiating turn");
addSequential(new AutoPivotHead(-90,3)); //turn

//addSequential(new AutoVertDartMove(95, 3));
//addParallel(new AutoBoomDartMove(90, 3));
System.out.println("Initiate final approach");
addSequential(new AutoMove(1, 0.0, 2, 1.0)); //go to switch
addSequential(new ClawRetract());
//addSequential(new ClawRetract()); // release cube into switch
//(POSITION,HEADING,TIMEOUT)
}
```

Helpful Links

WPILib: https://docs.wpilib.org/en/latest/index.html

WPI Java API: https://github.wpilib.org/allwpilib/docs/release/java/index.html

WPI C++ API: https://github.wpilib.org/allwpilib/docs/release/cpp/index.html

Chief Delphi (public FRC forum to ask questions): https://www.chiefdelphi.com

1261 codebase from 2018 (now public): https://github.com/RoboLions/frc2018-master

A Word of Advice

- Be sure to read the docs very carefully - WPI has a lot of information and goes into great detail which can be overwhelming at times.
- Look around for example templates in the project directories when creating a new robot. This makes the project a lot easier since you won’t have to write all of your code from scratch.
- Don’t forget to be creative. Think of unique ways to branch off templates and use different sensors and motors to fulfill objectives of the game.

Questions?