Object Oriented Programming in Java

With Noah Orellana
And Luke Kastner

Universal Basics:

- Setup + Main Method
- Data types and Variables
- Boolean operators & If-else statements
- Basic Methods (you may know then as functions)
- Using (Built in) Classes/Objects
- Intro to writing classes
- Application (Bot)

Default Data Types

Primitive:

- Integers (int)

- Decimals (double)

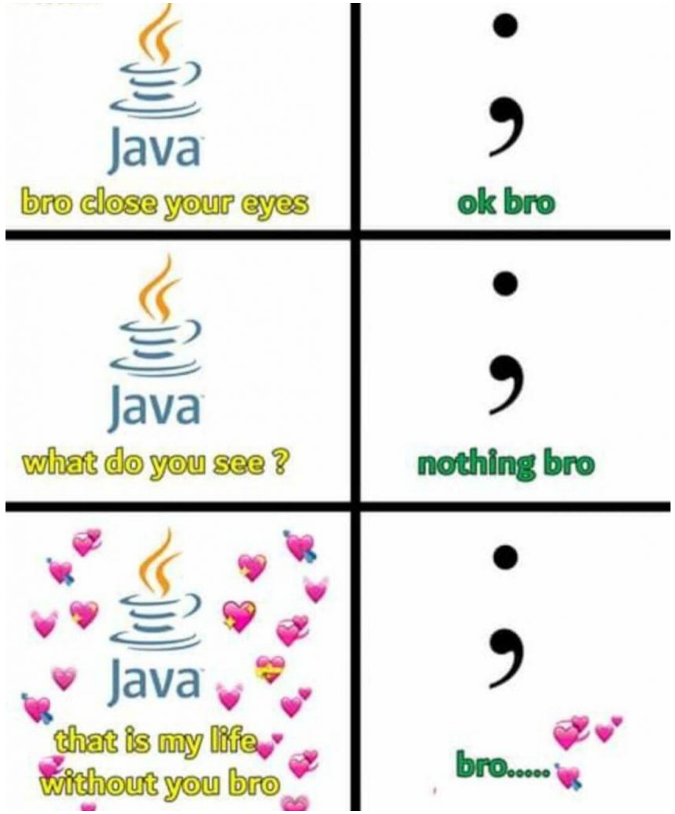
- Characters (char)

- True/False (boolean)

EXCEPTION:

- Strings! (String) [Notice the capital S]

- Strings are not a primitive type. Yes.. it's a little weird.



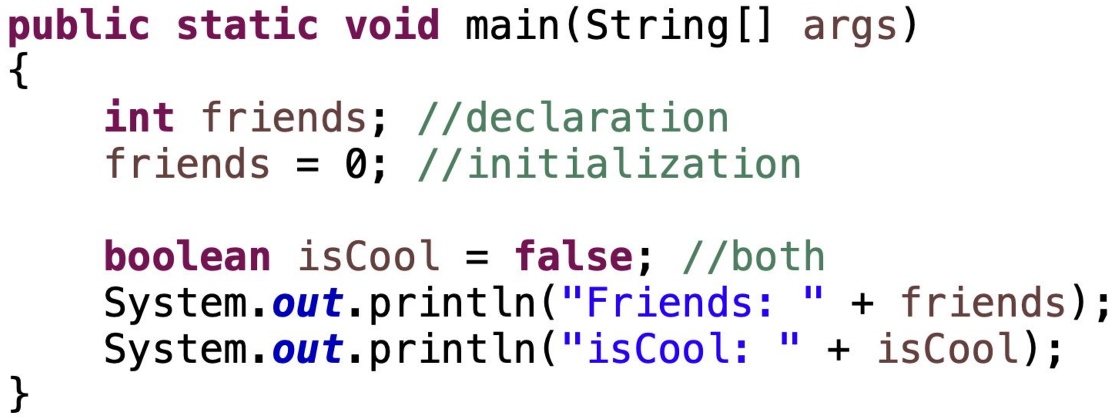
Variables!

- All variables must be declared with a data type.

- Only include when first declared.

- Can declare and Initialize (assign its value) in two statements or one!

- Like most other statements, a semicolon must follow



```
public static void main(String[] args)
{
    int friends; //declaration
    friends = 0; //initialization

    boolean isCool = false; //both
    System.out.println("Friends: " + friends);
    System.out.println("isCool: " + isCool);
}
```

Boolean operators

- And operator →&&

- Or operator → ||

- Not operator → !

- Equals (comparison) operator → ==

    - Note: Different from = (assignment) operator

    - Note: Only for primitive types

Warning:

Strings use .equals() to check equality. Do not use == !

```
public static void main(String[] args)
{
    boolean highMath = true;
    boolean highEnglish;
    highEnglish = true;

    boolean highTotal = highMath && highEnglish;
    System.out.println(highTotal);
}
```

If-else

- Must be contained in what is called a “block”

    - Unless only one statement

- A block is indicated by { }

- Similar to other languages.

    - Executes block (what is inside {}) if the statement

    - Notice no ; after a block or after the if statement

```
int age = 16;
if(age>65)
    System.out.println("you are old");
else if(age>19)
    System.out.println("you an adult");
else if(age>17)
{
    System.out.println("you are a teen");
    System.out.println("you are an adult");
}
else if(age>12)
    System.out.println("you are a teen");
else
    System.out.println("You are a kid");
```

Output:

```
you are a teen
```

Lists (ArrayList in java)

- Lists contain a **set** of data

- Create a list then add to it

- Basic list methods:

SKIP

  - .add(item) → adds to end of list

  - .add(index, item) → adds to specified index in list

  - .get(index) → returns item at this index

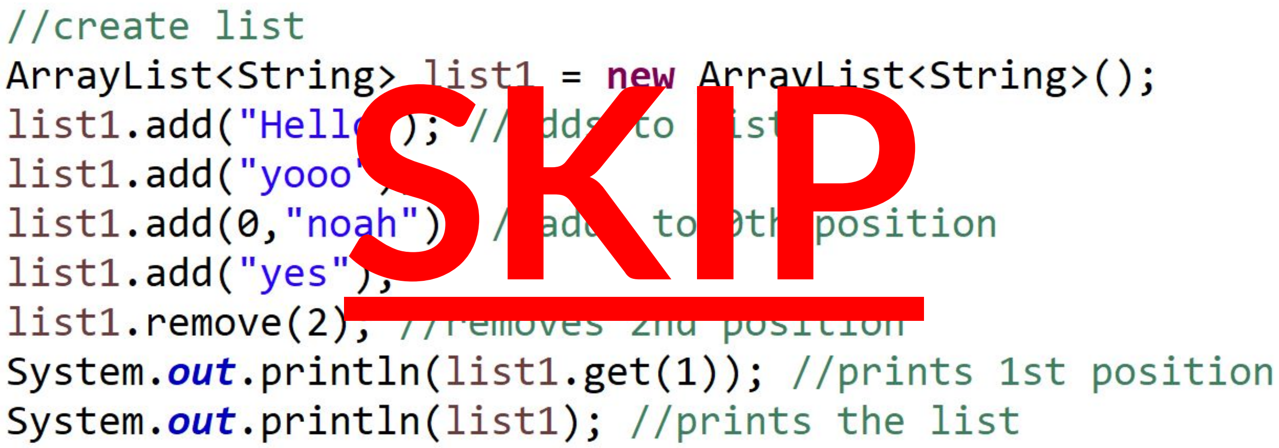
  - .remove(index) → removes item at this index

NOTE: Lists in java start at position 0

Create an ArrayList:

```
ArrayList<String> list1 = new ArrayList<String>();
```

List example



```
//create list
ArrayList<String> list1 = new ArrayList<String>();
list1.add("Hello"); //adds to list
list1.add("yooo");
list1.add(0,"noah"); //adds to 0th position
list1.add("yes");
list1.remove(2); //removes 2nd position
System.out.println(list1.get(1)); //prints 1st position
System.out.println(list1); //prints the list
```

SKIP

Output:

```
Yes
[noah, Hello, yes]
```

Loops

- Executes certain condition as shown in examples.

Output (while loop):

```
Num: 64

Num: 16

Num: 4
```

Output (for loop):

```
Hello, 0

Hello 1,

Hello 2,

Hello 3,

Hello, 4
```

```
int num = 64;
//while loops execute code blocks while condition is met

//so this loop repeats WHILE num > 1
while(num > 1)
{
    System.out.println("Num: " + num);
    num /= 4;  //divides num by 4
}
```

```
//for loops execute code in the block {} while
//the condition (in the middle) is met

//for loops are while loops, that keep track and modify
//a set variable (commonly i) each iteration (loop)

//so this loop
//repeats WHILE i is less than 5
for(int i = 0; i<5; i++) //i++ increments i by 1
{
    System.out.println("Hello, " + i);
    //prints out Hello, [current value of i]
    //then adds 1 to i
}
```

“Enhanced” For Loop

SKIP

- also called for-each loop

- Used to go through all elements in a list

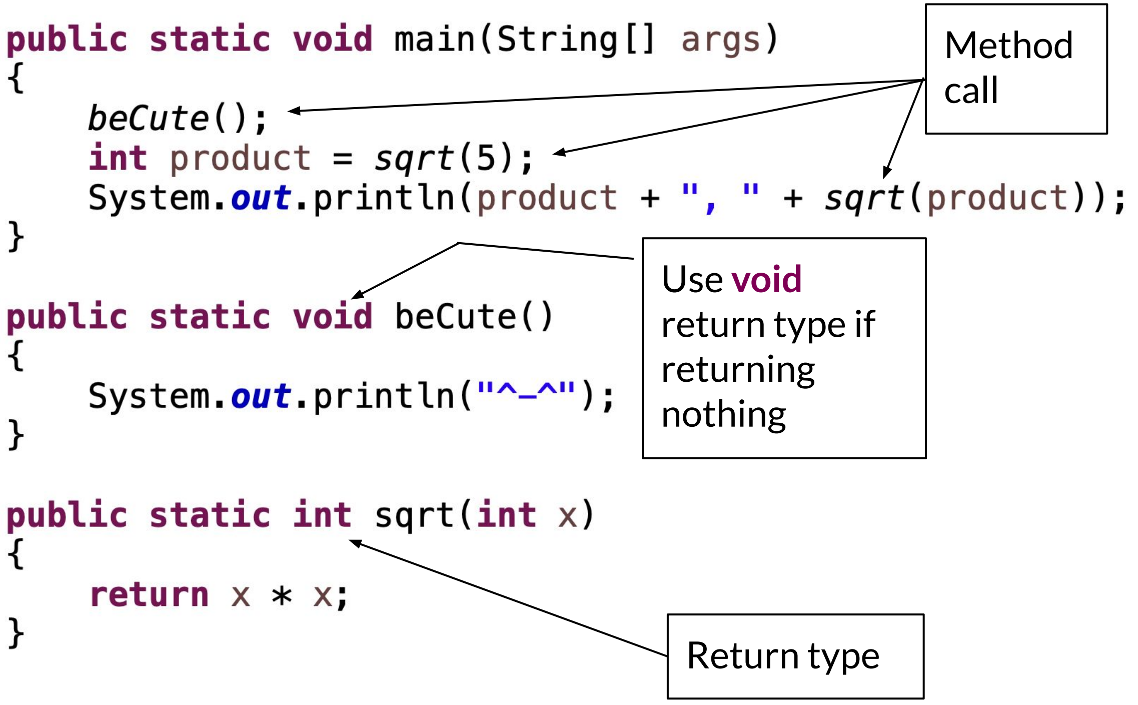
- think of the : as the word <u>in</u>

```
ArrayList<String> myList = new ArrayList<String>();
myList.add("hello");
myList.add("noah");
myList.add("how");
myList.add("are");
myList.add("you");

for(String word : myList)
{
    System.out.println(word);
}
```

“for each word in myList”

Methods

- Methods can be called to perform a function
- Always part of a class
- Contains return type (can only return one type)
- Methods must be surrounded by braces { }
- Methods are called by with the method name followed by (). If there are parameters, pass arguments into ()



```
public static void main(String[] args)
{
    beCute();
    int product = sqrt(5);
    System.out.println(product + ", " + sqrt(product));
}

public static void beCute()
{
    System.out.println("^_^");
}

public static int sqrt(int x)
{
    return x * x;
}
```

Classes

- Can be runnable (if they contain main method) **[Driver Class]**
    - We have already been using this
- May just contain useful methods (functions) **[Utility/Helper Class]**
- Can be used as a blueprint to create objects

SKIP

We will look at the last two now

Using built in classes with static methods and variables

- There are many built in Classes to java.
- You can use these throughout your code.
- What do you think System is? From `System.out.println("hi");`
  - It's a class!
- And of course, println() would be a method.
- Lets see some other examples!

SKIP

NOTE:

<u>Public/Private</u> indicates whether other classes have access to a field (variable) or method

Built in Class (static) examples:

```
//Math//
double radian = Math.PI/3; //public static variable from Math
double cool = Math.cos(radian); //cos() public static method from Math

System.out.println(cool); //who remembers trig?

//Integer//
int num = Integer.parseInt("5"); //takes a number as a string, turns into integer

//System//
String spacing = System.lineSeparator(); //gets line spacing
```

SKIP

From inside the class and out

Output:
8
true

SKIP

Class

```java
public class CoolMethods
{
    public static int power(int number, int power)
    {
        int newNum = 1;
        for(int i = 0; i<power; [illegible]) //whi[illegible] less t[illegible] pow[illegible]
        {
            newNum*=number;
        } //increments i by 1
        return newNum;
    }

    public static boolean startsWithN(String word)
    {
        word = word.toLowerCase(); //sets word to be lowercase
        if(word.substring(0,1).equals("n"))
            return true;
        return false;
    }
}
```

M[illegible]hod

Method call

```java
public class OtherClass
{

    public static void main(String[] args)
    {
        int num = CoolMethods.power(2, 3);
        System.out.println(num);

        String name = "Noah";
        System.out.println(CoolMethods.startsWithN(name));

    }

}
```

This would be an example of a “Utility Class”

NOTE: Calling the static method uses the class name

Objects

- Objects have unique data types
- Objects store information, and can perform tasks
    - Fields (variables) and (non-static) methods
- Every object belongs to a class.
    - The class is the data type.

Class name AND ()
Add arguments if necessary.
Looks like a method.

Variable name

```
Thingy gadget = new Thingy(5);
```

Data type (class)

new keyword
(use when creating new object)

Argument

Method name

Refer to object using variable name

```
gadget.doSomething();
```

Noun/Verbs

```
Person noah = new Person();
```

```
noah.speak();
```

Verb

Noun

We can say,
noah speaks

Example of a method that as not static.

Notice: method is being called from Object itself

Examples

Scanner object

```java
Scanner input = new Scanner(System.in);
System.out.print("Enter your name:");
String name = input.nextLine();
```

We take input this way

DecimalFormat object

```java
double amount = 5043.34;
DecimalFormat df = new DecimalFormat("$#,##0.00");
String money = df.format(amount);
```

Method of DecimalFormat class

Random object

```java
Random r = new Random();
int randomNumber = r.nextInt(6);
```

Method of Random class

Building our own class

```
public class OtherClass
{
    public static void main(String[] args)
    {
        Line arm = new Line(5);
        arm.increaseLength();
        arm.increaseLength(2);
        System.out.println(arm.getLength());
    }
}
```

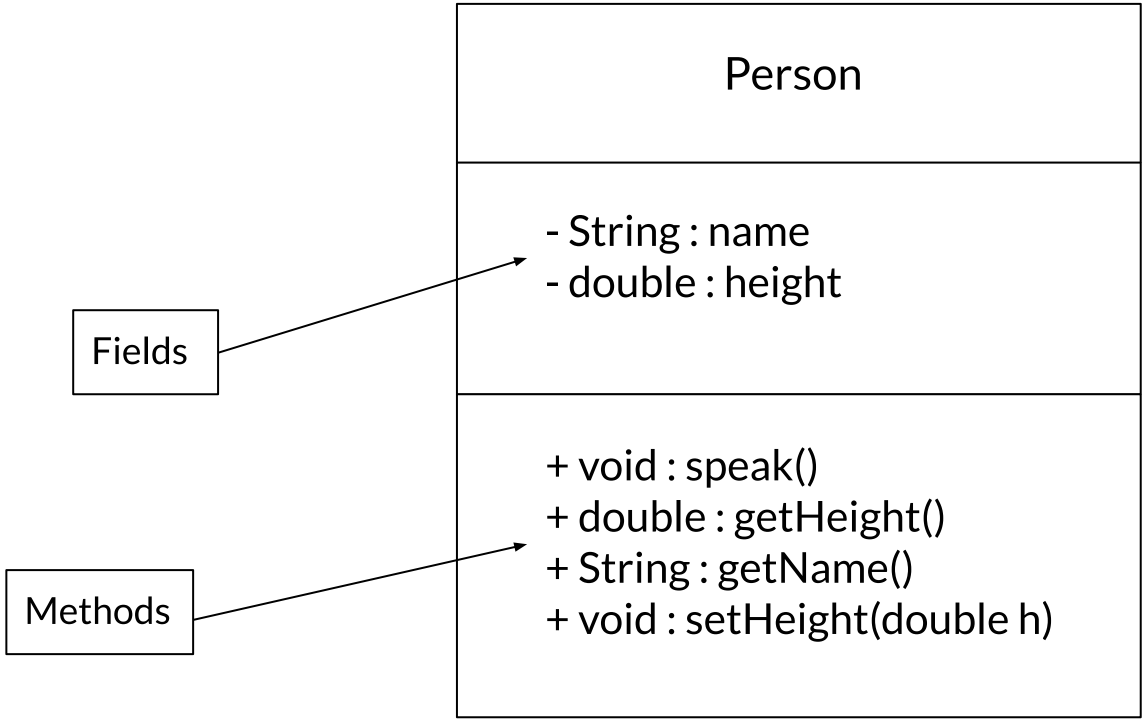
Output:

```
8
```

```
public class Line
{
    private int length; //field

    //constructor
    //called to create object
    public Line(int l)
    {
        length = l;
    }

    //returns length field
    // "getter"
    public int getLength()
    {
        return length;
    }

    //sets length field
    // "setter"
    public void setLength(int l)
    {
        length = l;
    }

    //this is a method
    public void increaseLength()
    {
        length++;
    }

    //overloaded method
    public void increaseLength(int num)
    {
        length += num;
    }

}
```

Guided Practice: Person Object

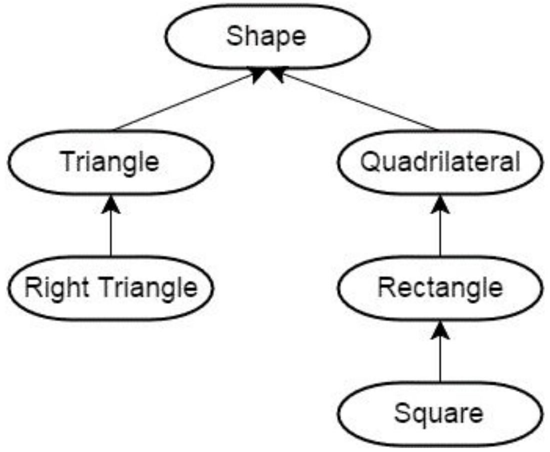
“UML”



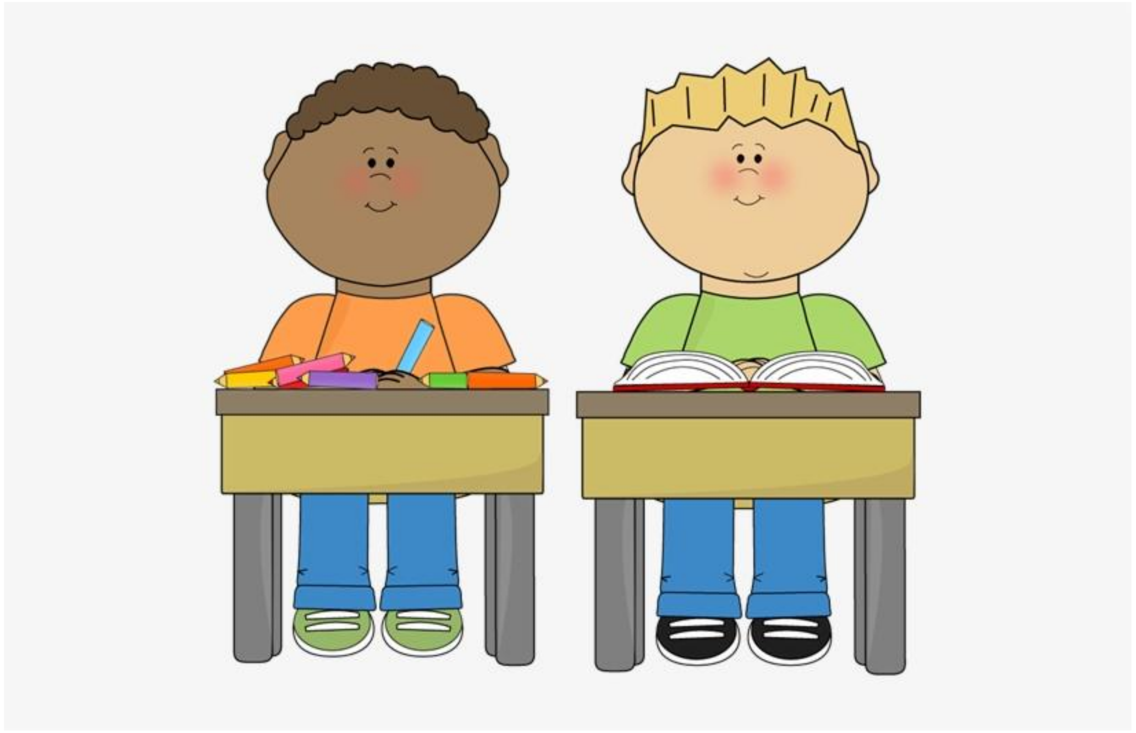
```
Person
- String : name
- double : height
+ void : speak()
+ double : getHeight()
+ String : getName()
+ void : setHeight(double h)
```

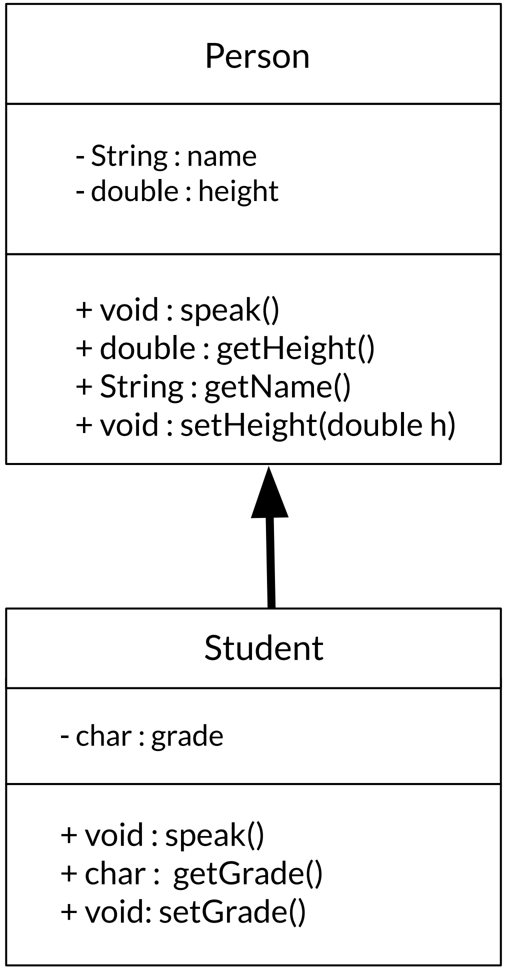
How to use inheritance

- Use <u>extends</u> keyword to inherit from another class
- <u>super</u> refers to the parent class
- child class inherits all parent class public methods/fields
- code reuse
    - Constructor: super(param, param);
    - Ex: super.method();



Guided Practice: Student class



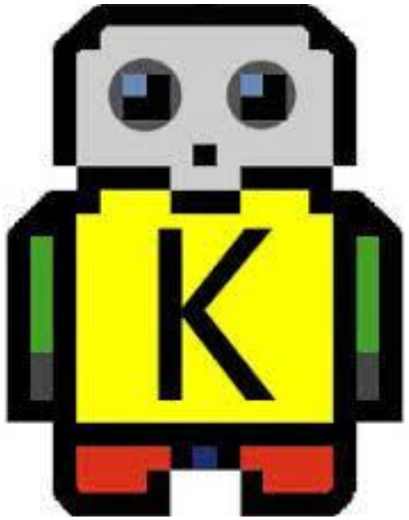


Now let's look at the Robot

- We will be using karelbot

- Lets try the default bot out

Goal: Build a better version of default Bot



Main functionality of our (given) Bot

- Constructor
    - Robot(x, y, Direction, beepers);
    - Is called when you make a new Robot
- move()
    - moves one space
- turnLeft()
    - -turns left
- putBeeper()
    - places a beeper
- getX()
- getY()

Let’s write some commands

Modifying the bot

- We don’t want to rewrite the whole bot
- We just want to add and modify existing code

So,

INHERITANCE TIME!

Let’s Write Our First Bot

- turnRight() - Have the robot turn right
- turnAround() - Have the robot turn around
- stepBack() - Have the robot move backwards once
- move(int steps) - Have the robot move forward a number of steps
- stepBack(int steps) - Have the robot move backwards a number of steps
- drawLine(int num) - have the robot draw a line of beepers that is num long

RainbowBot

- Changes beeper colors everytime it moves

Note: You can change color by doing World.

SKIP

MotherBot

- spawns new bot

- spawnBot() - adds it to list, returns a new robot

- shutDown() - shuts down all children

- moveAll() - moves all children one space

SKIP

LetterBot

- Make a bot that can draw a letter with beepers

-Pick a letter, and make a method for the bot to draw the letter in a 5x5 grid

Try creating a bot of your own

- Make sure it inherits from the a Bot class
- Ask if you need help
  - Ideas
  - Implementation