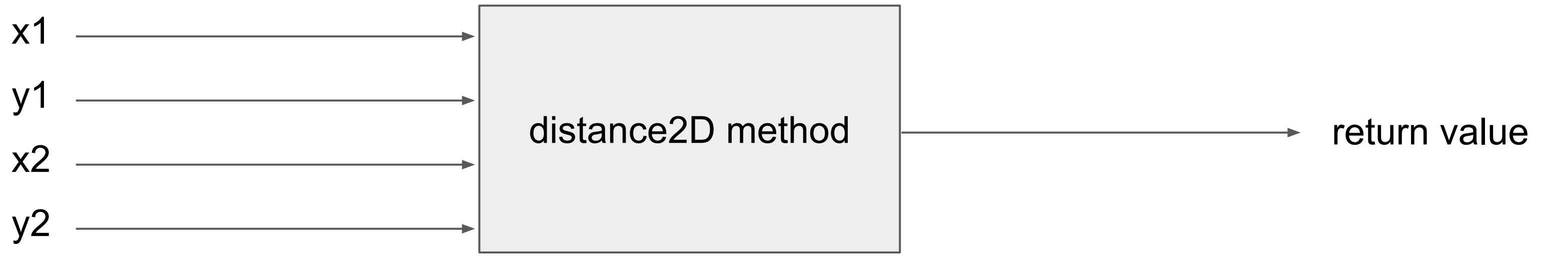
2.5: Calling Methods That Return Values

Return Value

Methods can take **inputs** (“arguments” or “parameters”), and they can also spit out a single **output** (“return value”)

```
public double distance2D(double x1, double y1, double x2, double y2)
```



Methods are like functions… what is the difference between a function and a method?

Methods vs. Functions

A method represents an action supported by some class of object in Object-Oriented Programming (OOP). Java is an OOP language.

Some programming languages have functions. Some have methods. Some have functions and methods!

In Java, there are only methods, but you can use methods to model functions like mathematical functions (Math.sin, Math.cos are static methods of the Math class)

Return Value

Methods that don’t **return** anything have a **void** return type

```
public void printGreeting(String name) {
    System.out.println("Hello " + name + "!");
}
```

Return Value

- The **type** of the return value must match what is declared in the method declaration
- Right:

```
public int getNumberTimesThree(int value) {
    return 3 * value;
}
```

- Wrong (why?):

```
public int getNumberTimesThree(int value) {
    return 3.0 * value;
}
```

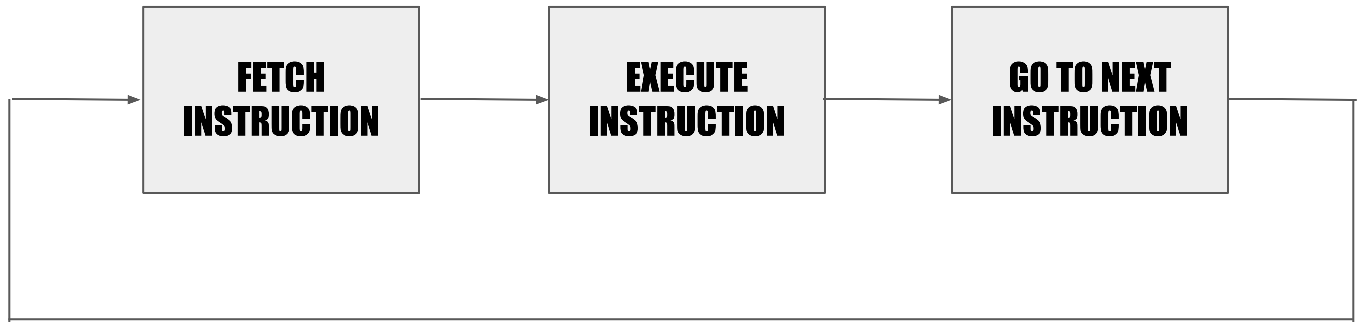
- Q: Java only lets you return one value from a method. How might you return multiple pieces of data at once?

The return statement

- The **return** statement specifies the return value of a method.
- The return statement is one of Java's **control flow** statements. What do you think that means?

The return statement and control flow

Basic program of the Java virtual machine (and any CPU, really)



Control flow statements may jump to somewhere else other than the next instruction.

The return statement exits your method immediately with the specified return value; all other statements are skipped.

The return statement in void methods

If your method is other than void return type, it MUST use return

You can use return in a void method too, though. Just don't specify any return value, since the method has none:

```
return;
```

Why do you think you might use return in a void method?

Getter and Setter Methods

In Java, you'll commonly find that classes declare `getXYZ` and `setXYZ` methods for their properties (instance variables).

```
public class TurtleTestGetSet
{
  public static void main(String[] args)
  {
      World world = new World(300,300);
      Turtle yertle = new Turtle(world);
      System.out.println("Yertle's width is: " + yertle.getWidth()); // Yertle's width is: 15
(this is the default width)
      yertle.setWidth(200);
      yertle.setHeight(200);
      System.out.println("Yertle's width is: " + yertle.getWidth()); // Yertle's width is: 200
(this is the width after we've set it to 200 2 lines above
      yertle.turnRight();
      world.show(true);
  }
}
```

This is considered a **best practice**. Q: Why do you think that is?

toString Methods

- In Java, all objects can be represented in String form by defining a **toString** method.
- This can be useful for programmers to get a visual or textual representation of an otherwise abstract object.
- How can we make the `toString` method to the right more descriptive?
- What gets printed if you don't define a `toString` method?

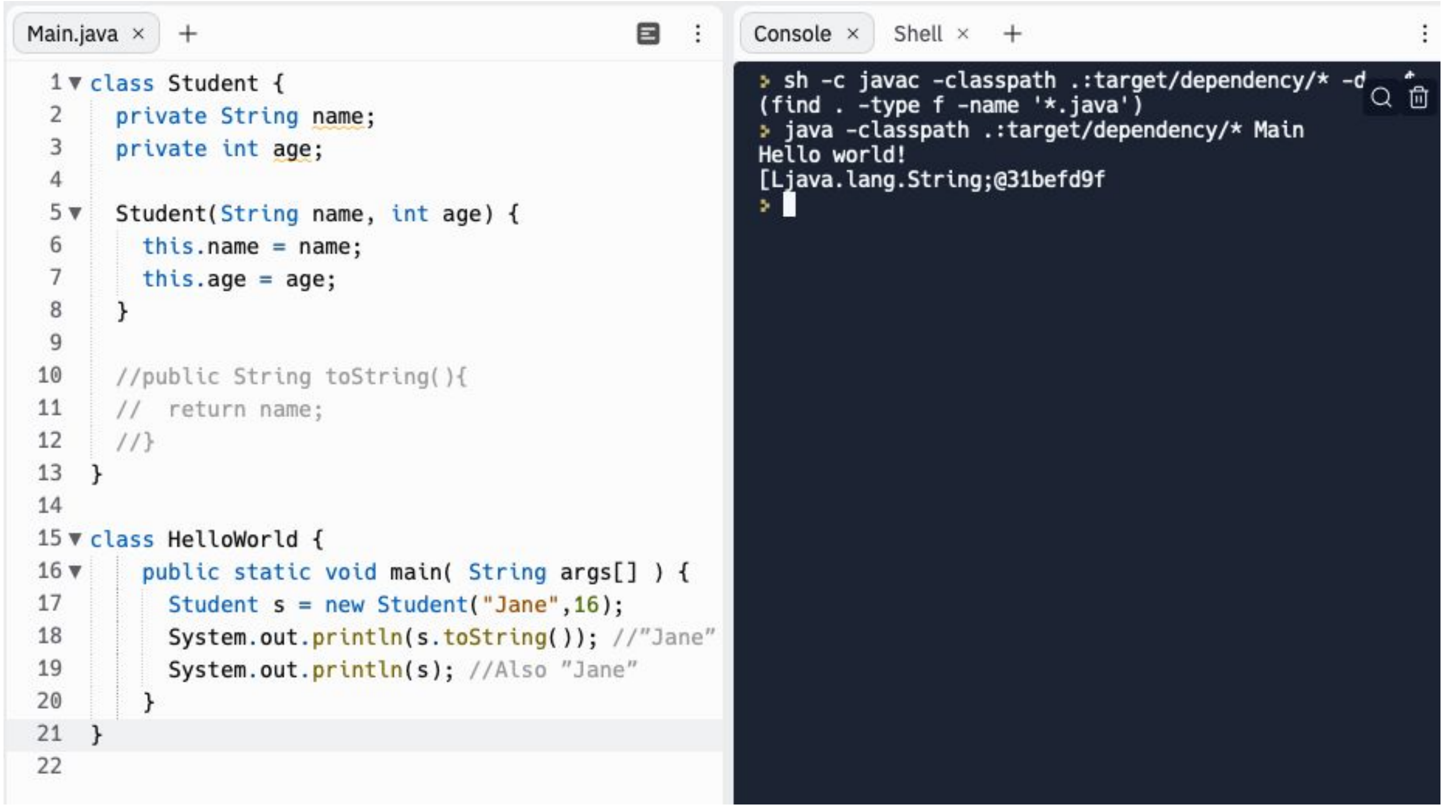
```
class Student {
  private String name;
  private int age;

  Student(String name, int age) {
    this.name = name;
    this.age = age;
  }

  public String toString(){
    return name;
  }
}

class HelloWorld {
    public static void main( String args[] ) {
      Student s = new Student("Jane",16);
      System.out.println(s.toString()); //”Jane”
      System.out.println(s); //Also ”Jane”
    }
}
```

Object.toString



Main.java

```java
class Student {
  private String name;
  private int age;

  Student(String name, int age) {
    this.name = name;
    this.age = age;
  }

  //public String toString(){
  //  return name;
  //}
}

class HelloWorld {
   public static void main( String args[] ) {
     Student s = new Student("Jane",16);
     System.out.println(s.toString()); //"Jane"
     System.out.println(s); //Also "Jane"
   }
}
```

Console

```
› sh -c javac -classpath .:target/dependency/* -d
(find . -type f -name '*.java')
› java -classpath .:target/dependency/* Main
Hello world!
[Ljava.lang.String;@31befd9f
›
```

this??

- The code on the right says `this.name`… what do you think `this` is?

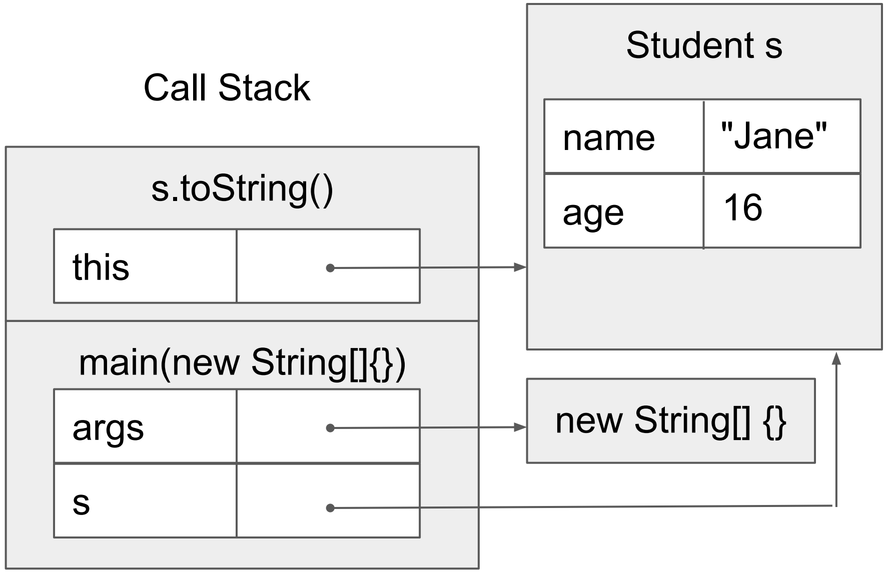
```
class Student {
  private String name;
  private int age;

  Student(String name, int age) {
    this.name = name;
    this.age = age;
  }

  public String toString(){
    return name;
  }
}

class HelloWorld {
    public static void main( String args[] ) {
      Student s = new Student("Jane",16);
      System.out.println(s.toString()); //”Jane”
      System.out.println(s); //Also ”Jane”
    }
}
```

`this` is a reference to the current object instance



```
class Student {
  private String name;
  private int age;

  Student(String name, int age) {
    this.name = name;
    this.age = age;
  }

  public String toString(){
    return name;
  }
}

class HelloWorld {
    public static void main( String args[] ) {
      Student s = new Student("Jane",16);
      System.out.println(s.toString()); //"Jane"
      System.out.println(s); //Also "Jane"
    }
}
```

VERY common pattern.

Q: What's the difference here between name and this.name?

Q: What do you think the next thing on the call stack will be on top of main?

Exercise: What's the output of this program?

```java
class Liquid
{
    private int boilingPoint = 100;
    private int freezingPoint = 0;
    private double currentTemp;
    private String name;

    public Liquid(String name) {
        currentTemp = 50;
        this.name = name;
    }

    public void lowerTemp() {
        currentTemp -= 10;
    }

    public double getTemp() {
        return currentTemp;
    }

    public boolean isFrozen() {
        return this.currentTemp <= this.freezingPoint;
    }

    public String toString() {
        return "Liquid | Name: " + this.name + " | Temp : " + this.currentTemp;
    }
}
```

```java
public class HelloWorld{

    public static void main(String[] args){
        Liquid myLiquid = new Liquid("Water");
        System.out.println(myLiquid);
        myLiquid.lowerTemp();
        System.out.println(myLiquid);
        System.out.println(myLiquid.isFrozen());
        myLiquid.lowerTemp();
        myLiquid.lowerTemp();
        myLiquid.lowerTemp();
        myLiquid.lowerTemp();
        System.out.println(myLiquid.toString());
        System.out.println(myLiquid.isFrozen());
    }
}
```

Your Turn!

http://tpcg.io/CBv7Yr6c

http://tpcg.io/CBv7Yr6c

1. Change the toString so that it displays whether or not the liquid is Frozen
2. Similar to lowerTemp, write a raiseTemp method that increases the temperature of the liquid by 10
3. Add code to your HelloWorld program so that you get the liquid to no longer be frozen
4. Implement a method isBoiling() that tells us whether or not the liquid is boiling
5. Change your toString to just tell us the liquid name and whether or not it’s boiling
6. Change your HelloWorld program to that you get the liquid to boil. Print out the object once it’s boiling!