**String Literals**

System.out.print() only prints the thing that you give it

System.out.println() prints what you want it to, and creates a new line

A string literal is when you say

String test = “Hello”;

This will literally print what you want it to. String literals are stored in a JVM (java virtual machine). It is almost like a global bank.

A string object however, looks like this:

String s = new String(“Hello”);

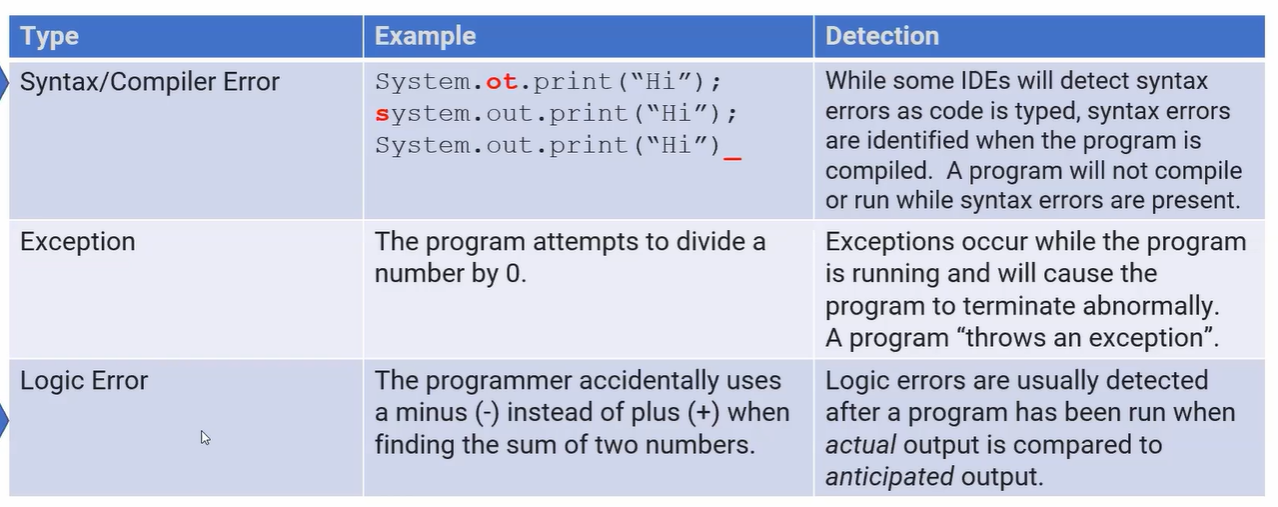
Both are an object of the String class, but the first is just a shorthand notation of the second.

If you compare a string literal and a string object, you will not get the same result because, even though they have the same content, they are completely different entities in themselves.

Similarly, we have a literal, which will also literally print out what you tell it. However, a literal is an integer (integer literal)

**Types of Errors**

These are three types of errors.



**Primitive:**

Doubles - 64 bits

Integers - 32 bits - use int when declaring, not Integer. Integer is non-primitive

Booleans - 1 bit

String is a non-primitive data type

Non-primitive or **Wrapper classes** data types use methods to perform actions

**Naming variables:**

Can’t start with a digit

Can’t use Java reserved terms

**Key word: final**

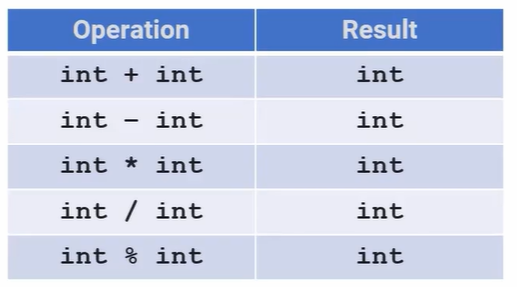
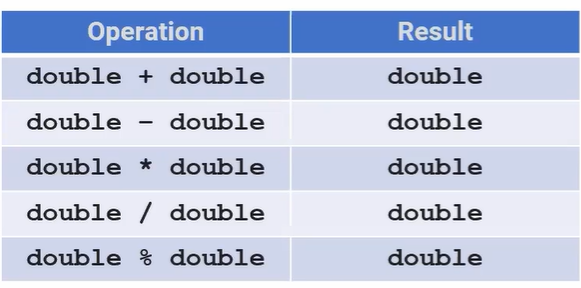
If we want to declare a variable that cannot be changed once we give it a value, a constant.

Use the keyword final before its declaration

**final** int DAYS\_IN\_WEEK;

(we use all caps and underscores while naming)

**Expressions and Assignment statements**



The result will always take the largest data type of the two

Modulo returns the remainder

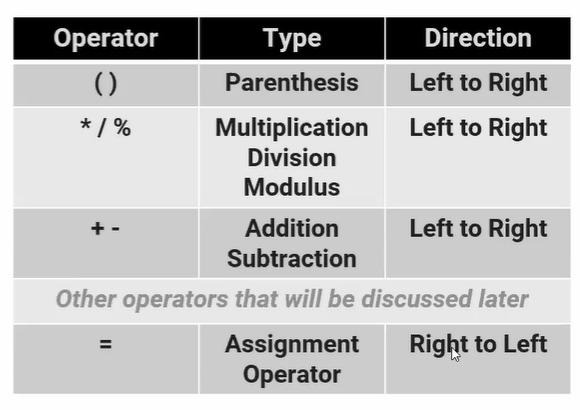
**Compound Expressions**

Pemmdas still applies (extra m because of modulo = multiply = division)

Arithmetic exception is when something is not defined, or not executable. For example, division by 0 is undefined. Brings everything to a halt.

**Assignment Operator**

**=** is used to initialize and assign the value on the right to the value on the left. The two sides are not interchangeable. Left is usually a variable.



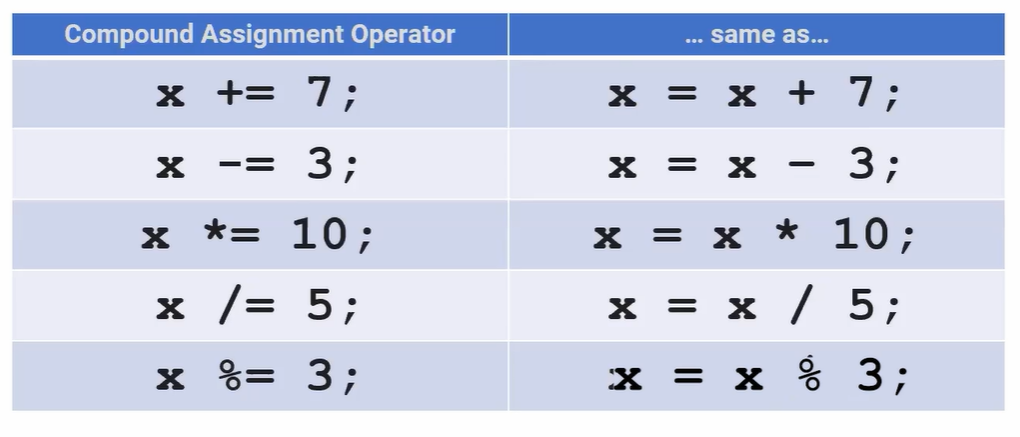
Assignment is the last thing that happens; once the right side is fully evaluated, then it is assigned to the left side.

Works right to left

**Compound Assignment operator**

x += 7

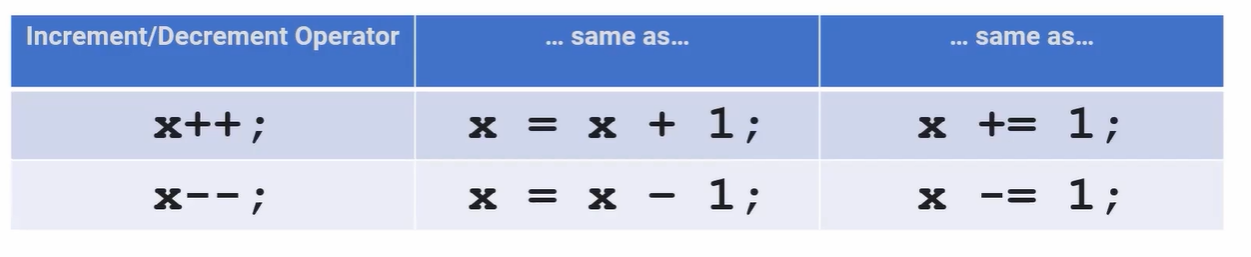
Will take x, add seven, and assign that new value to x.



**Increment Decrement Operators**

x++

Takes x, adds 1, and then assigns it back to x.



**Looking at code line by line (practice)**

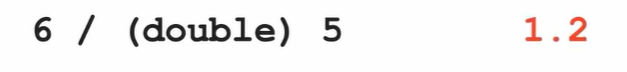
**Casting and Ranging of Variables**

Casting - change one data type to another. Two casting operators:

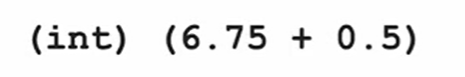
* (int)
* (double)



Divides the double 6 and integer 4 and results in the double 2.75.



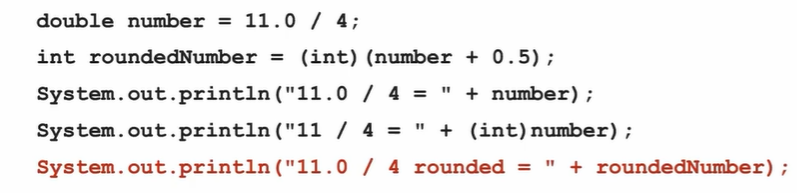
Integer 6 divided by double 5 results in the double 1.2



The result should be the double 7.25 but we cast it to the integer 7.

When you have negative numbers, like -2.5, and you cast it to an int, it becomes -2. Truncate

Code to **round/truncate** a number:



**Casting and Ranges of Variables**

Max values of an integer

That number is 232

The negative one is one larger because 0 is included.

Integers can be represented from -231 through (231-1) which would be 232 numbers total. We do minus one on the last number because 0 is included.

Integer.MAX\_VALUE and Integer.MIN\_VALUE are in the Integer class.

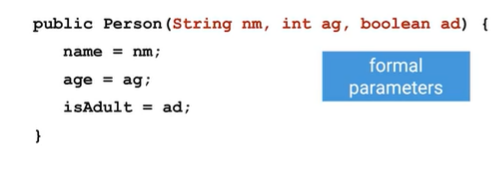
If you were to do the max number plus one, java will return the min value. Same with the min, if you did min number minus one, java will return the max number. This is because it is looping through a circle. This is an overflow error.

**Classes and Objects**

* A class is a blueprint for creating objects with the same behavior and defined attributes
* An object is a specific entity made from a class that you can manipulate in your programs
* Objects are instances of classes with variables used to name them
* Each object has behaviors and attributes that are defined by the class that was used to create it.
* Each object is individual, changing one object doesn’t affect the others

**Instantiation**

Constructors - initialize the attributes for an object



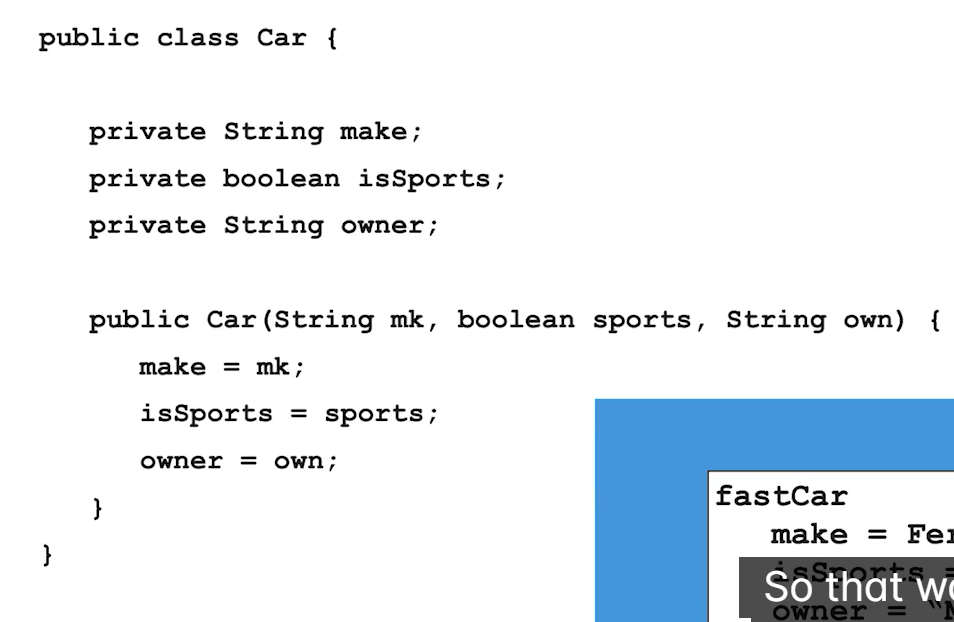
Constructors start with public, and have the same name as the class. In this case, these constructors are for the class Person.

The stuff in red is the formal parameters are creating the variables

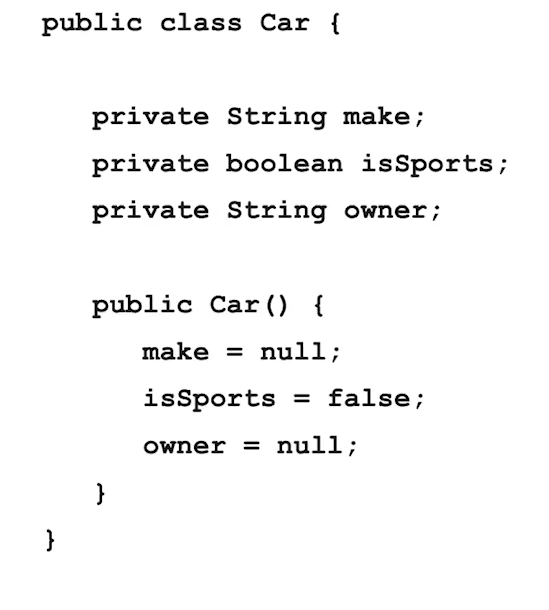
When creating an object, you would have to define those constructors - actual parameters:

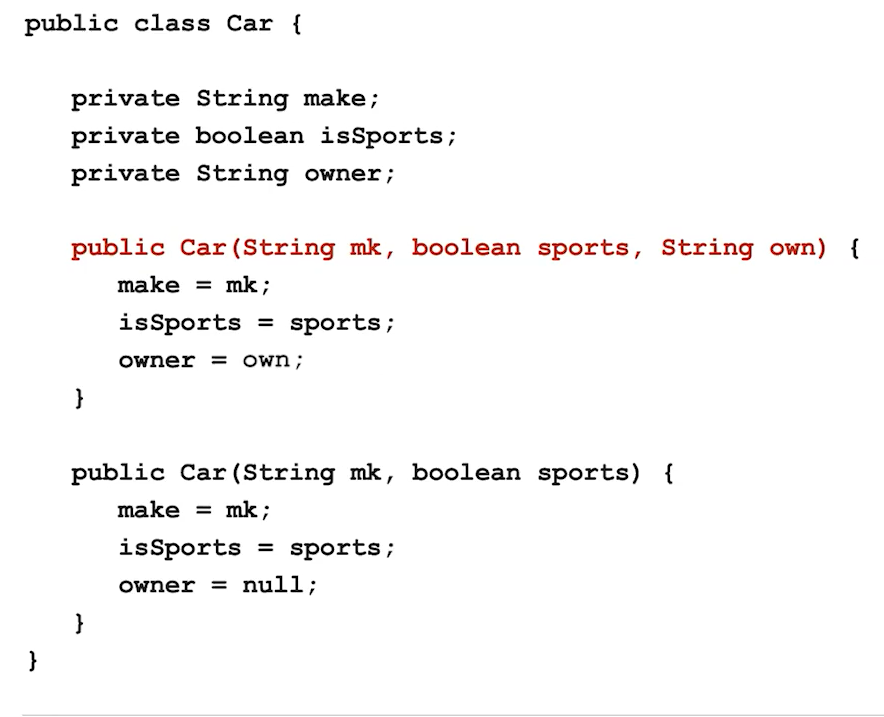


We can have more than one constructor for an object. This is called overloading the constructor.



The no-argument constructor has no parameters and sets the instance variables for the object to default values. This will set the constructor to whatever the default value is for that data type. For example:

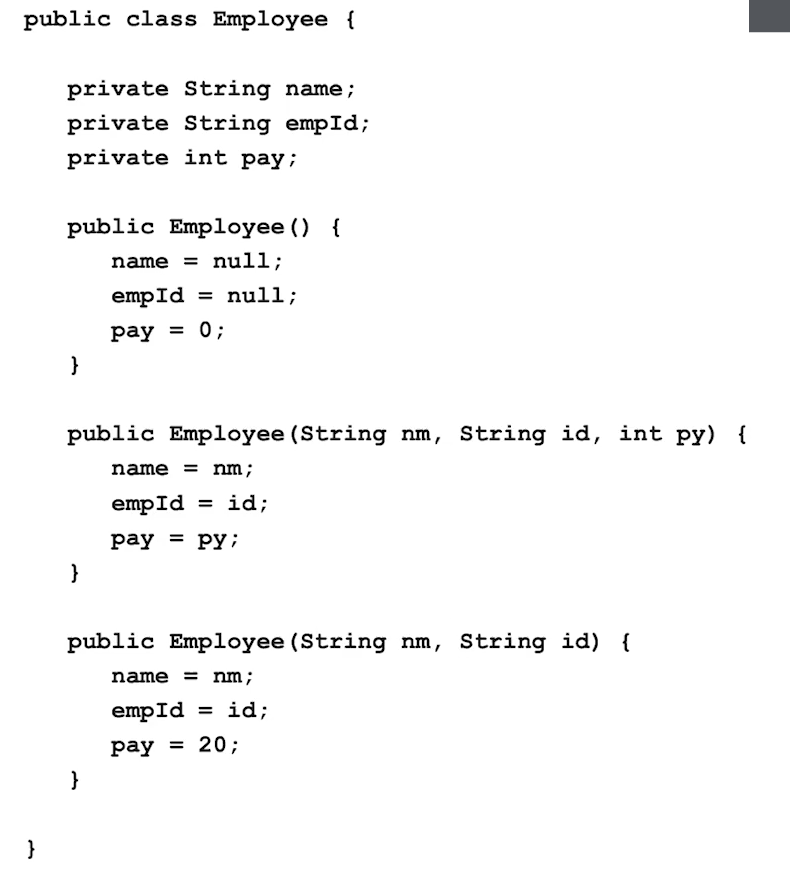


Can have something like this. This is an example of multiple constructors:

**Instantiation**

Basically when you create a class, you have to create the variables, which are the class attributes. And then in constructors, you set the value of those variables (attributes) to something. So you define them.

Below are three different constructors. The first one has no formal parameters because it is just a basic default employee. When you make an object using this, the values will be all the default values. The second two can be used to create actual employees with personalized data:



**Null**

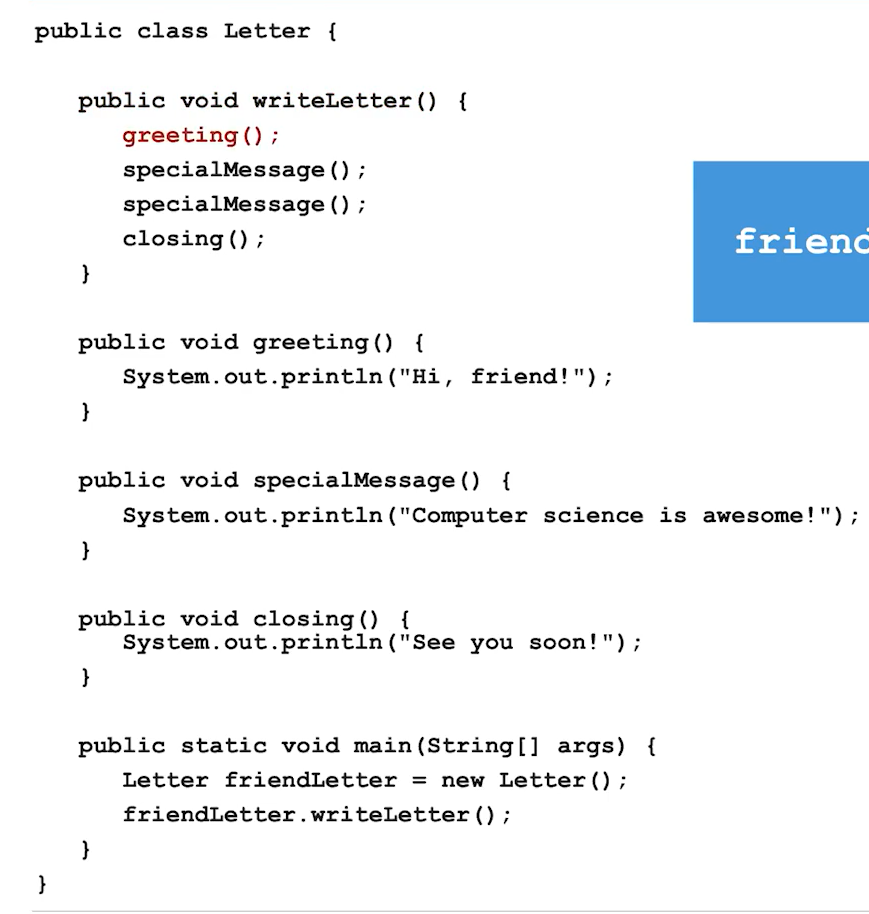
If we create an object and set it to null, without using any “new” keywords or referencing any constructor, it will quite literally be set to nothing. It is still taking up storage in memory but it will not contain anything.

It creates a reference to an object that takes up memory but is not exactly an object.

If you run a method on this object, since it is null, it isn’t an object, and it wouldn’t have any methods or constructors. So, if you run a method on it, it will return NullPointerExecption.

**Void method**

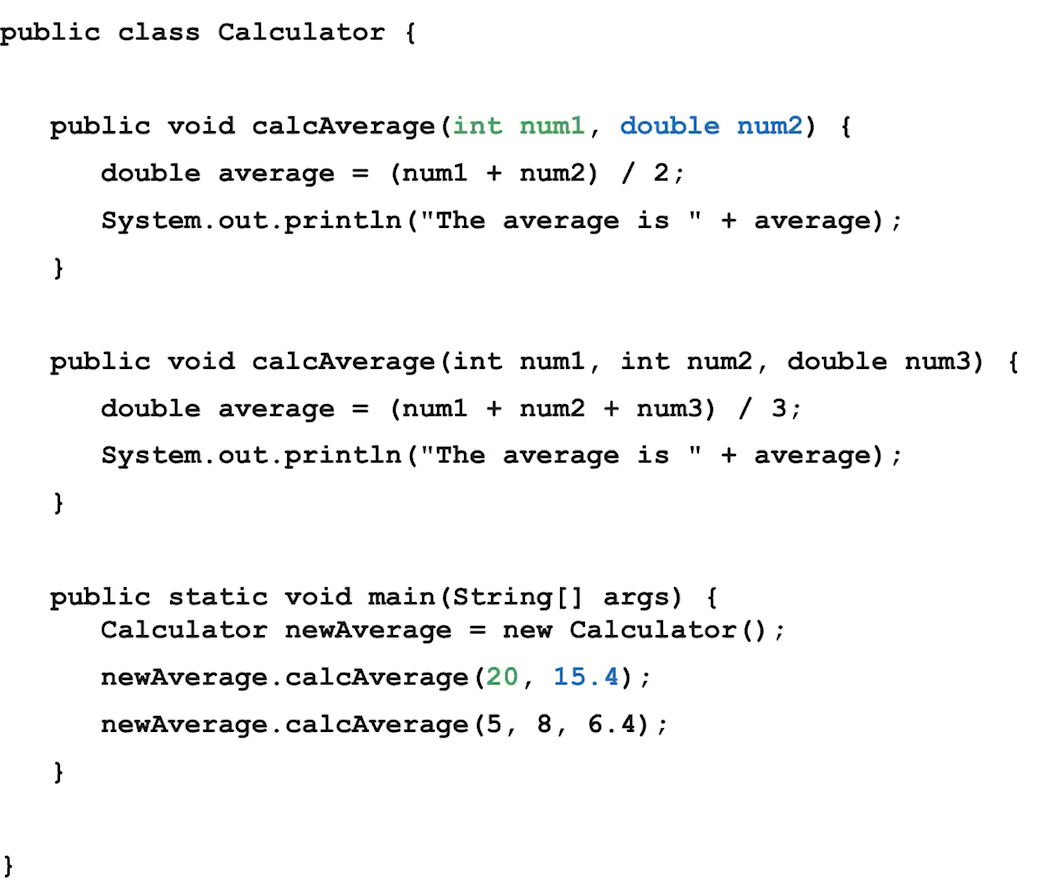
Methods define the behaviors for all objects of a class and consist of a set of instructions for executing the behavior. Void methods don't return anything. Example:



**Void method with parameters**

These are methods that take in some parameter, or input but doesn’t return anything

Overloading: Having multiple methods with the same name, but different parameters.



Static methods can be called by any object

Public method: can do this.name

Private: restriction on who can access.

**Non-Void methods**

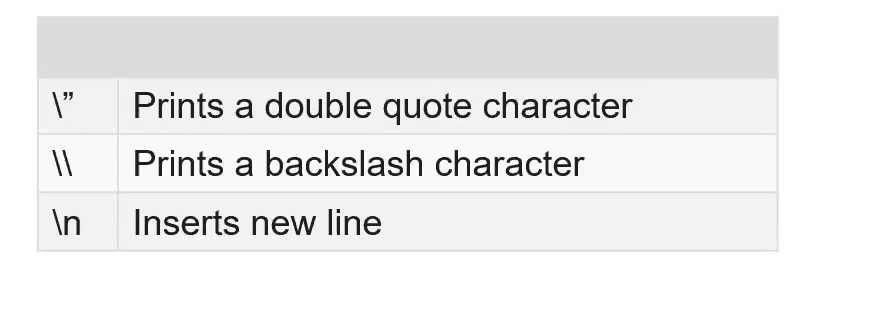
The value that is returned from a method, you need to specify the type that is being returned in the method signature. Let’s say you return something to main, main now has access to that returned value. Remember, return does not mean print. You would need to specifically say print the return value if you want it to be displayed.

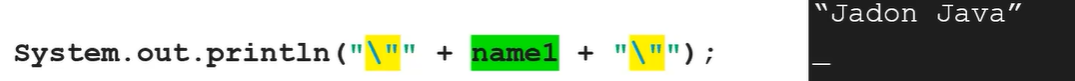
**String Class**

Strings are immutable - unable to be changed

Methods that act upon string objects do not change the state of the defined string object.

+= works with strings too, it will just concatinate





<https://apstudents.collegeboard.org/ap/pdf/ap-computer-science-a-java-quick-reference_0.pdf>

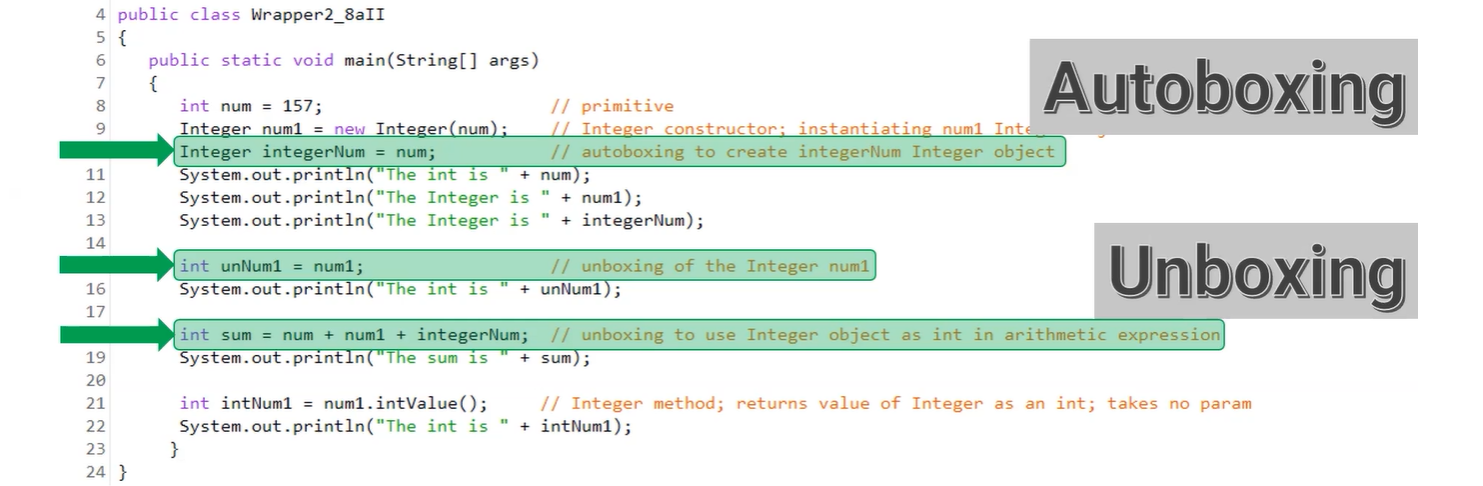
**Autoboxing/Unboxing**

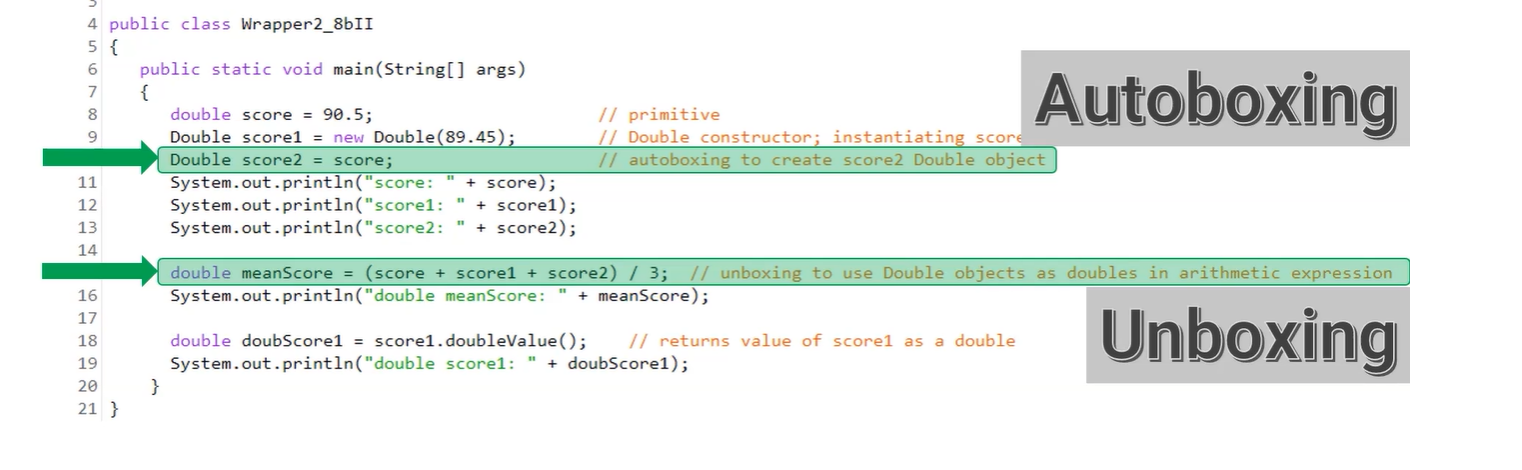
Integer is a wrapper class, and int is a primitive

Autoboxing (done by the compiler): turning the primitive type into their associated wrapper class

Unboxing (done by the compiler): turning the wrapper class into their associated primitive type

* Primitive data types only store a single value (int, double, boolean)
* Reference types only stores references (strings, arraylists)
* Wrapper classes are used to convert from primitive to reference types



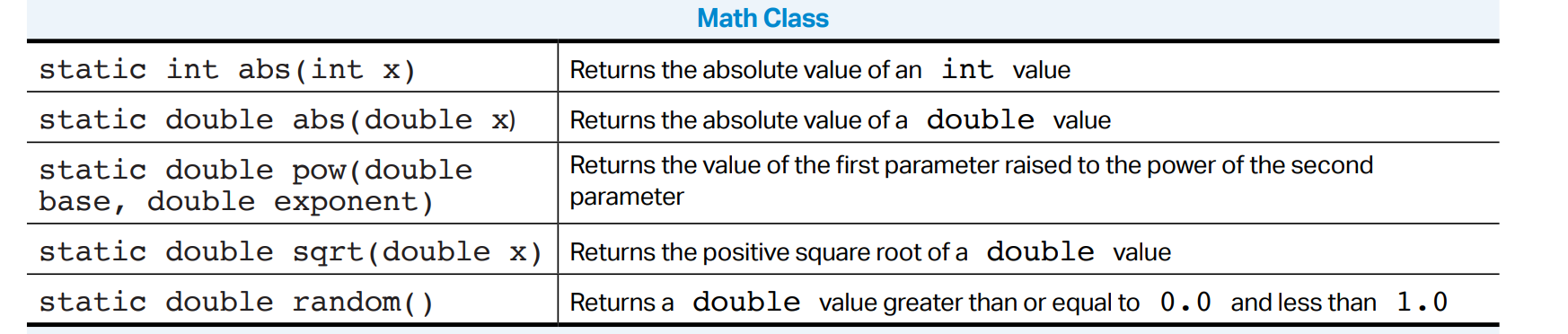


**Math class**

Math class methods are static so you don’t create Math objects to invoke the method; directly invoke the methods on the class (Math.method\_name)

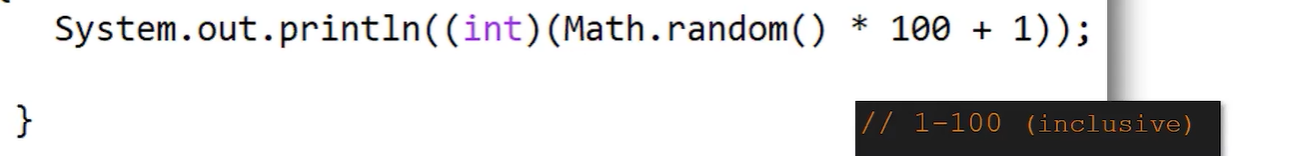
Notice how everything in the table is primitive

Math.sqrt(double x) only takes nonnegative numbers

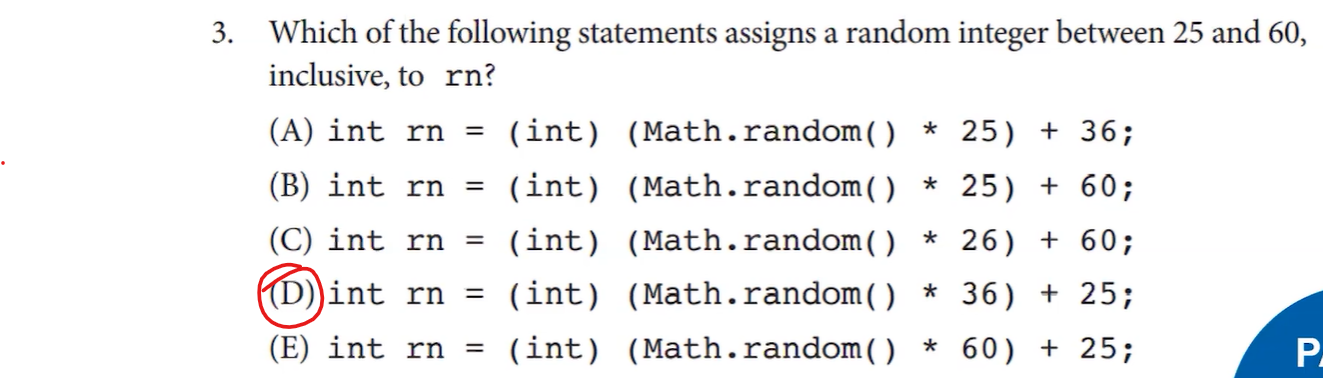


Since Math.random() obviously returns double, we can cast it into an int

* If you multiply Math.random by number N, then you will get a random number from 0 until N
* If you add a number to the Math.random, then that will be the minimum starting number



QUIZ:



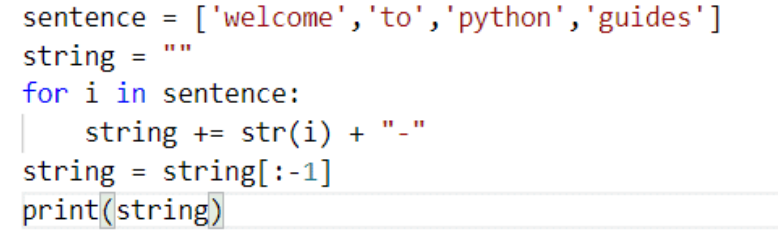
**(Compound) Boolean Expressions**

* Booleans store a true/false value (can only be one of these)
* Booleans can be generated using comparison expressions (equal/==, greater than/>, less than/<, etc.)
* If statements take in a boolean or boolean expression and run if the expression evaluates to "true"
* Else & Else if statements can be used in conjunction with if statements to run code if the if statement evaluates to false
* Using combinations of boolean operators, you can make compound boolean expressions
* Operators that can be used include and (&&), or (||), not (!), as well as parenthesis for grouping purposes



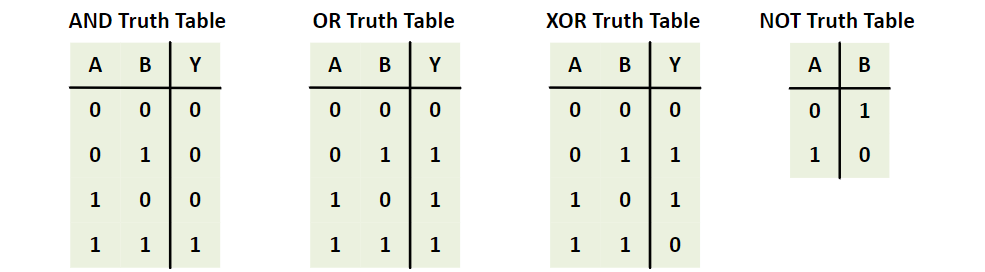
**Concatincation**

a series of interconnected things or events.



**Truth Tables**

* Can be used to see the values of boolean expressions



**While Loop**

Loops to reduce the code

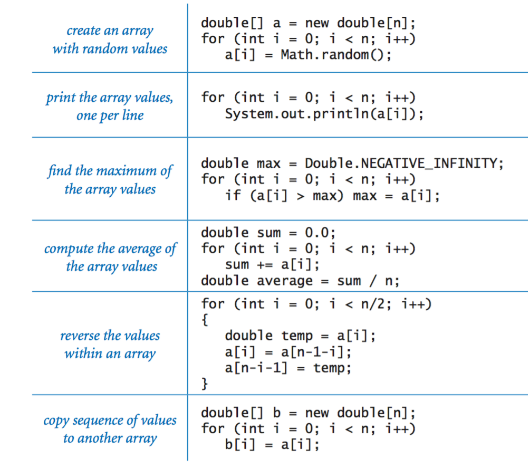
Writing code, and making it faster, instead of writing a tedious amount of code.

While loop: while something is true, keeps repeating the segment inside it. Once it is no longer true, the loop exists.

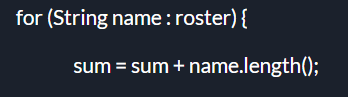
For Loop: will repeat the loop for a certain amount of time.

Recursion Loop: instead of any conditional, you repeat the code with a loop by calling on the function itself.

Nested iteration: technique; loops within loops. May be used to loop through a 2D array.



**Array Lists**

* Traversing through as array list can be done with loops and recursion loops
* Enhanced for-loop
  + 
* Uses of ArrayList is finding the minimum and maximum of an array list