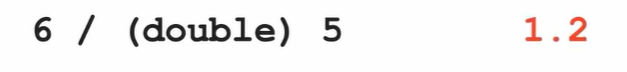
**Casting and Ranging of Variables**

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

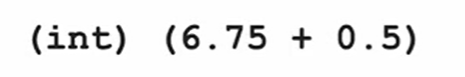
* (int)
* (double)



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



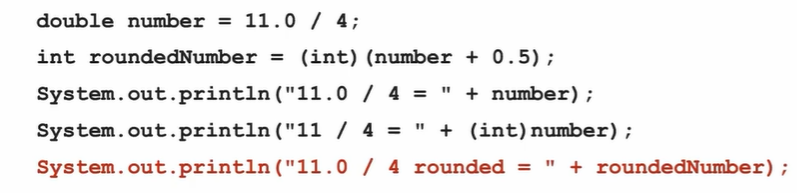
Integer 6 divided by double 5 results in the double 1.2



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.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Primitives:**

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

**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.

**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

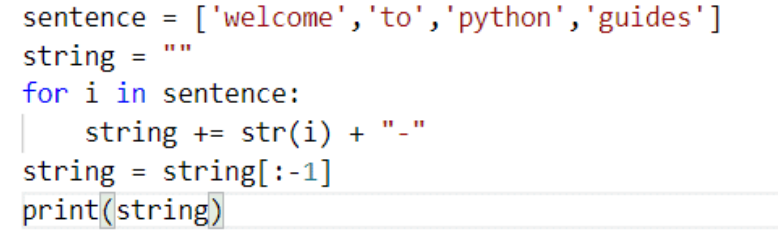
**final** int DAYS\_IN\_WEEK;

(we use all caps and underscores while naming)

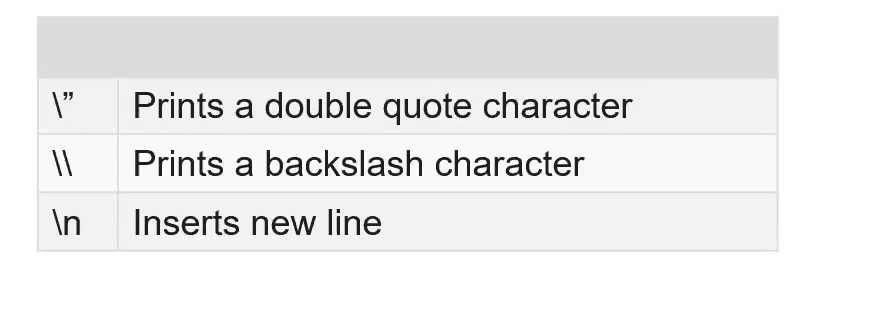
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

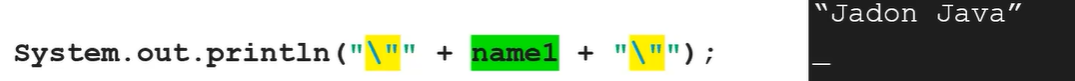
**Concatincation**

a series of interconnected things or events.



+= works with strings too, it will just concatinate





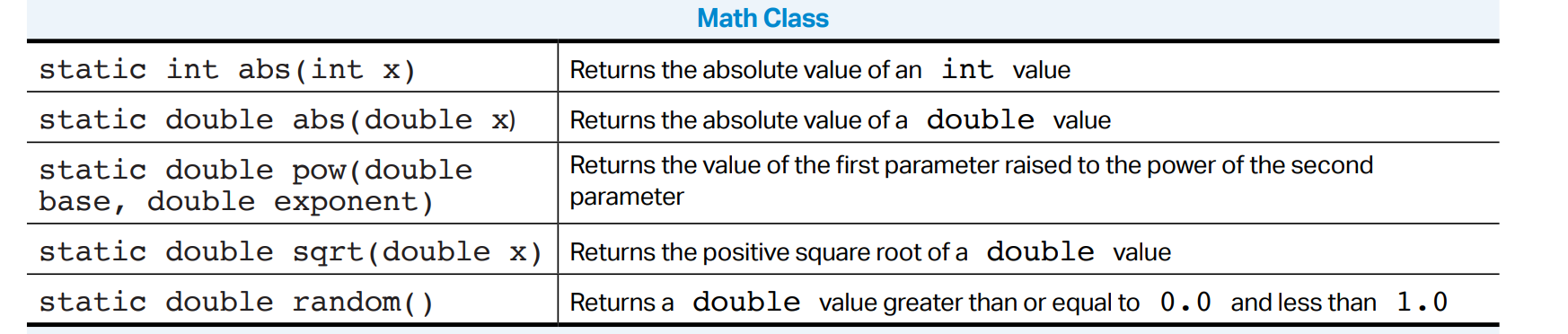
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**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)

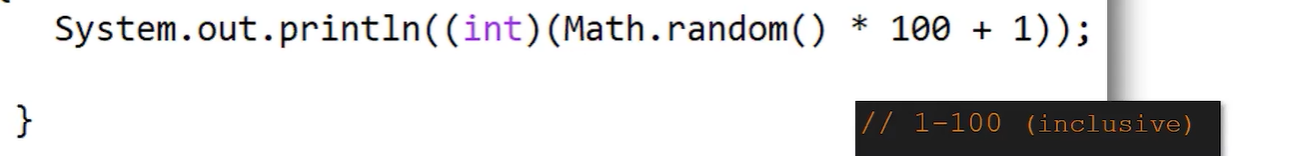
Notice how everything in the table is primitive

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



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Compound) Boolean Expressions**

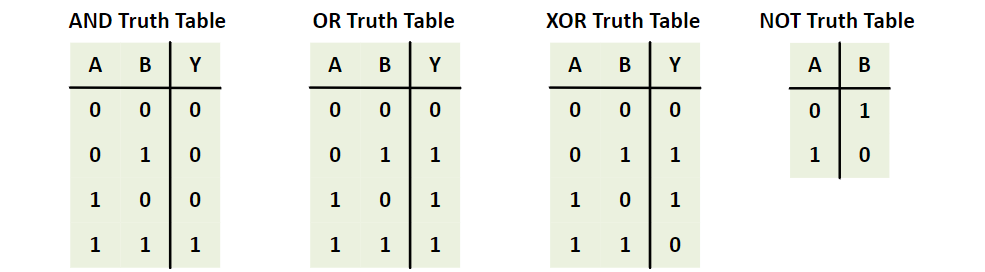
* Booleans store a true/false value (can only be one of these)
* Booleans can be generated using comparison expressions (equal/==, greater than/>, less than/<, etc.)
* If statements take in a boolean or boolean expression and run if the expression evaluates to "true"
* Else & Else if statements can be 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



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Truth Tables**

* Can be used to see the values of boolean expressions

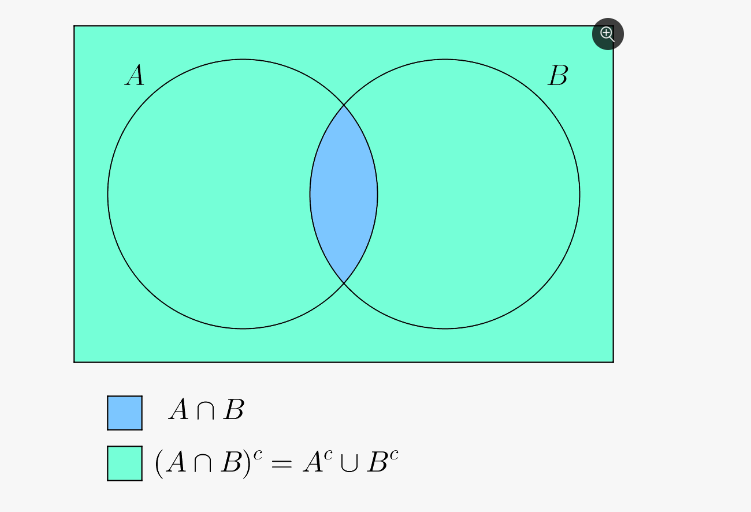


\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**De Morgan’s Law**

De Morgan's first law states that the complement of the union of two sets A and B is equal to the intersection of the complement of the sets A and B.

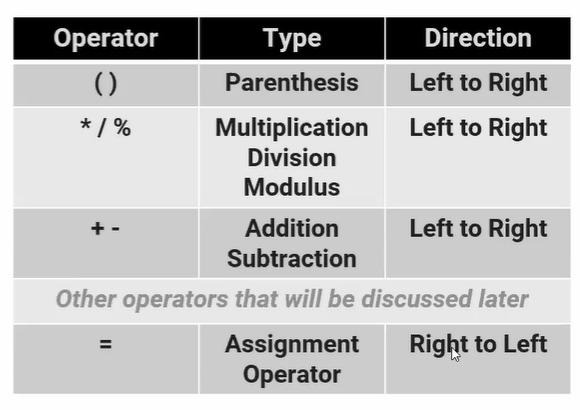
* *Not* (A *and* B) is the same as *Not* A *or* *Not* B.
* *Not* (A *or* B) is the same as *Not* A *and* *Not* B.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Comparing Numbers with Operations**

**=** 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.



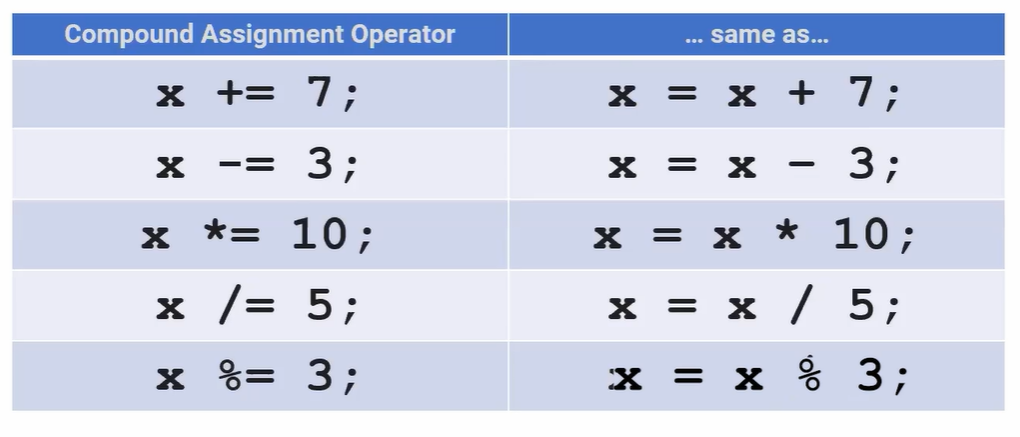
Assignment is the last thing that happens; once the right side is fully evaluated, then it is assigned to the left side.

Works right to left

**Compound Assignment operator**

x += 7

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



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Comparing Strings**

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.

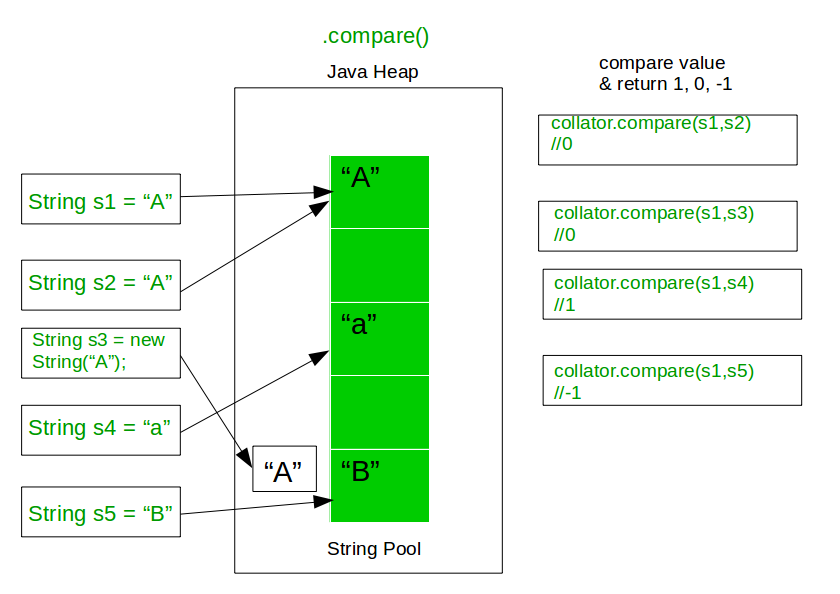
Similarly, we have a literal, which will also literally print out what you tell it. However, a literal is an integer (integer literal)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

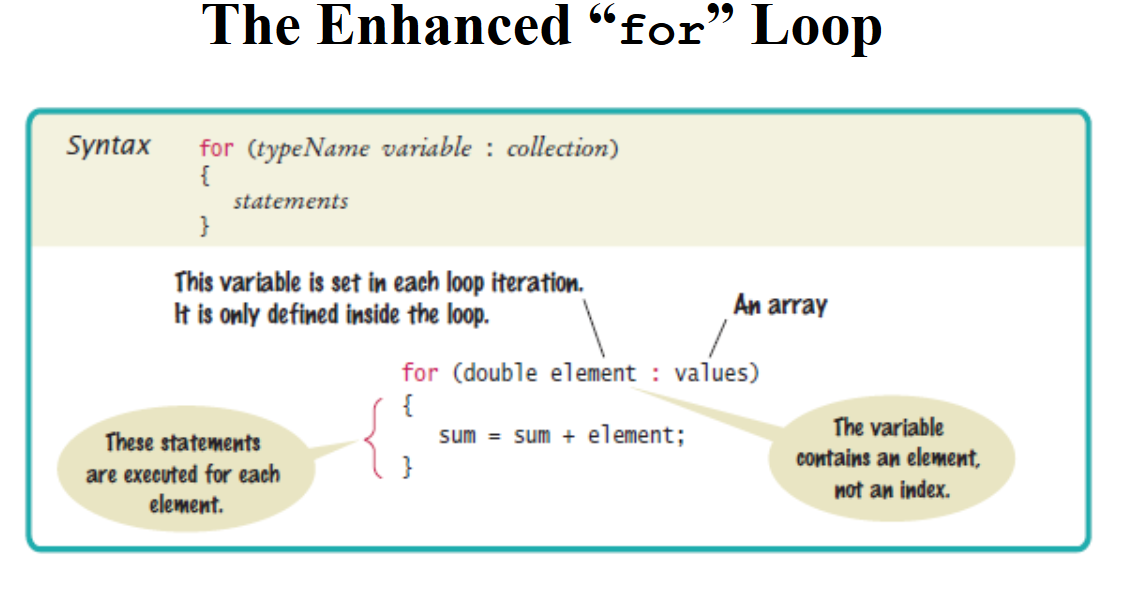
Comparing Objects

Unlike numbers and strings, we compare objects with ".equals()" rather than "==" operator

The "==" operator compares the memory location rather than the contents of the object

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

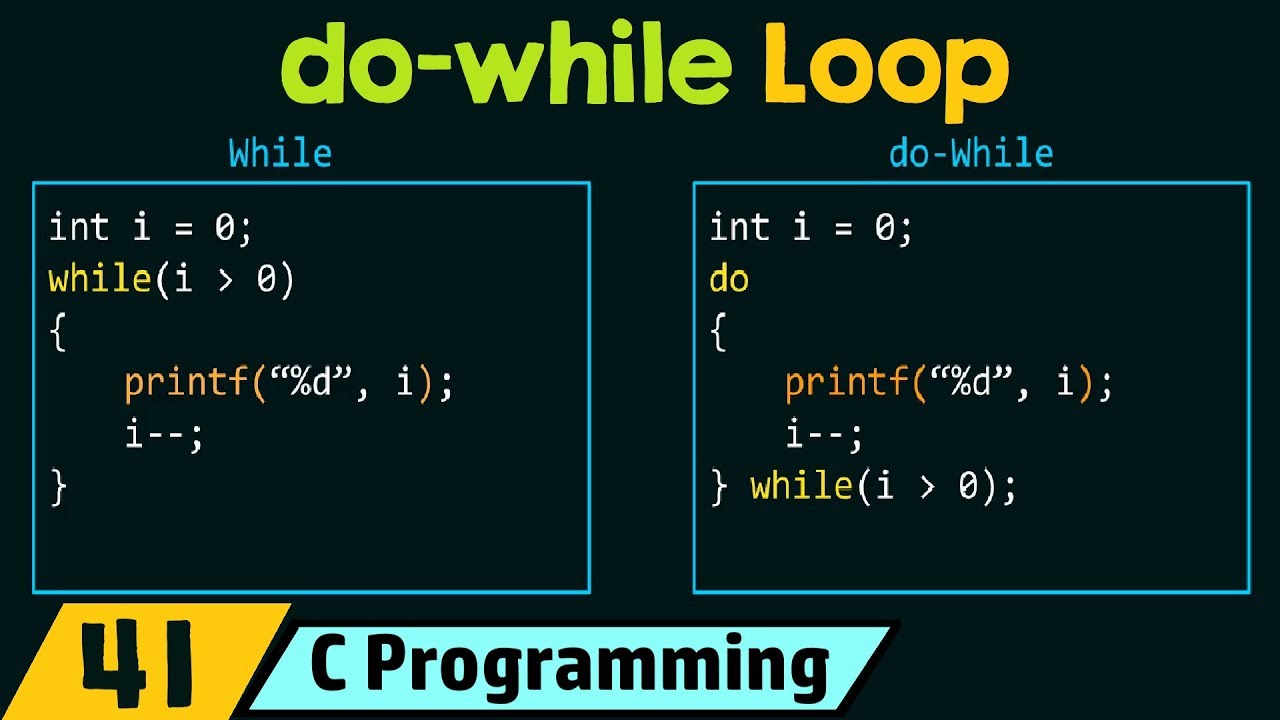
**For Loops (enhanced)**

* For loops can be used to iterate through an index, and modify it in different ways in the for loop declaration
* The enhanced for loop is exclusively used for iterating fully through an iterable (such as array)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**While Loops**

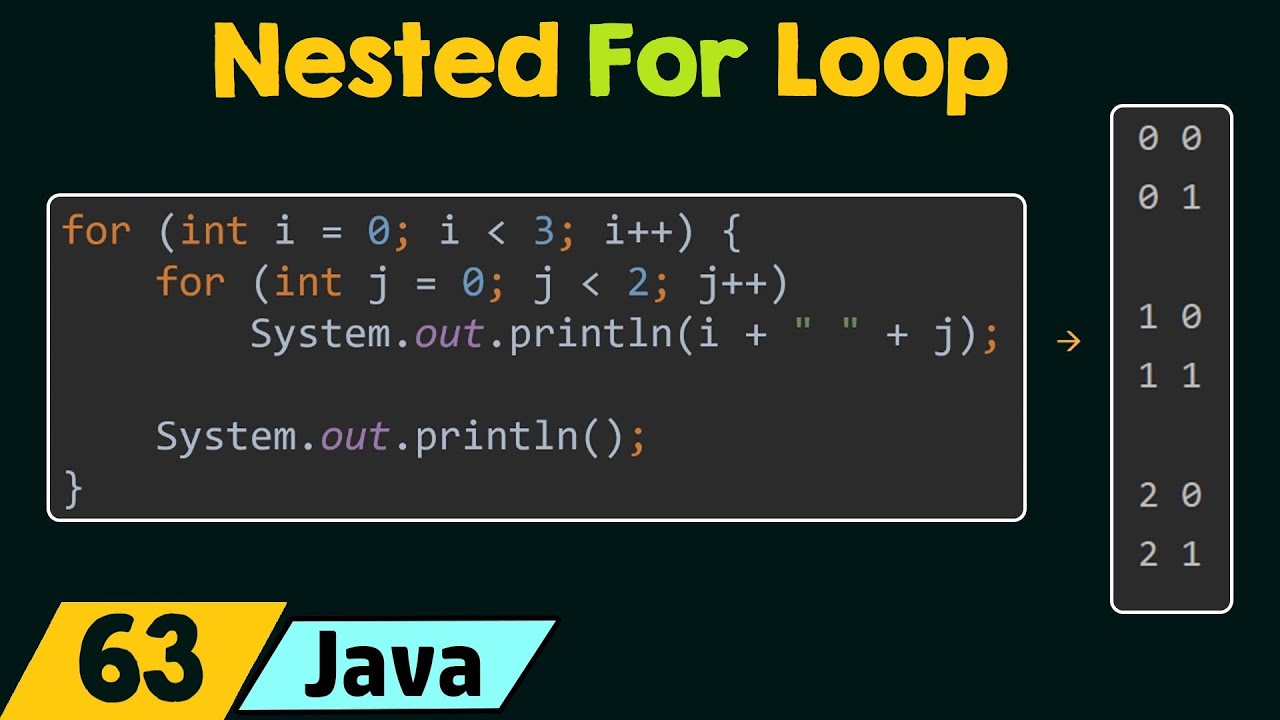
* While loops run while a condition is true, the condition is checked before each iteration of the code is run
* Do while loops also run while a condition is true, but the condition is checked AFTER each iteration of the code is run
  + This means that no matter what the do block runs at least once before the condition is checked

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Nested Loops**

* Loops can be used inside each other for better iteration
* Especially useful for 2D arrays

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Creating 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

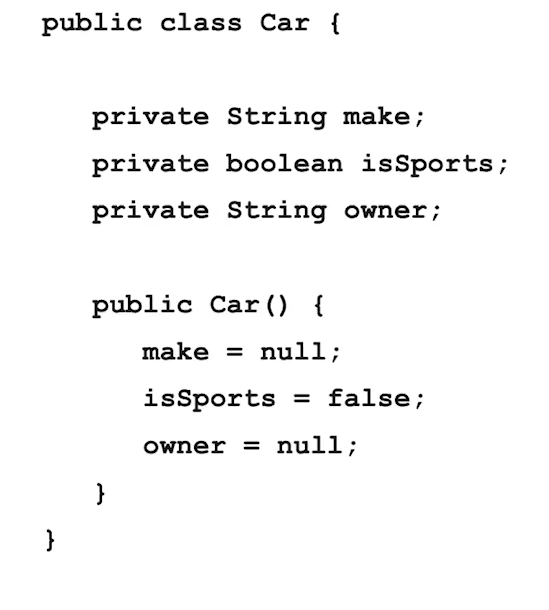
**Instantiation**

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.

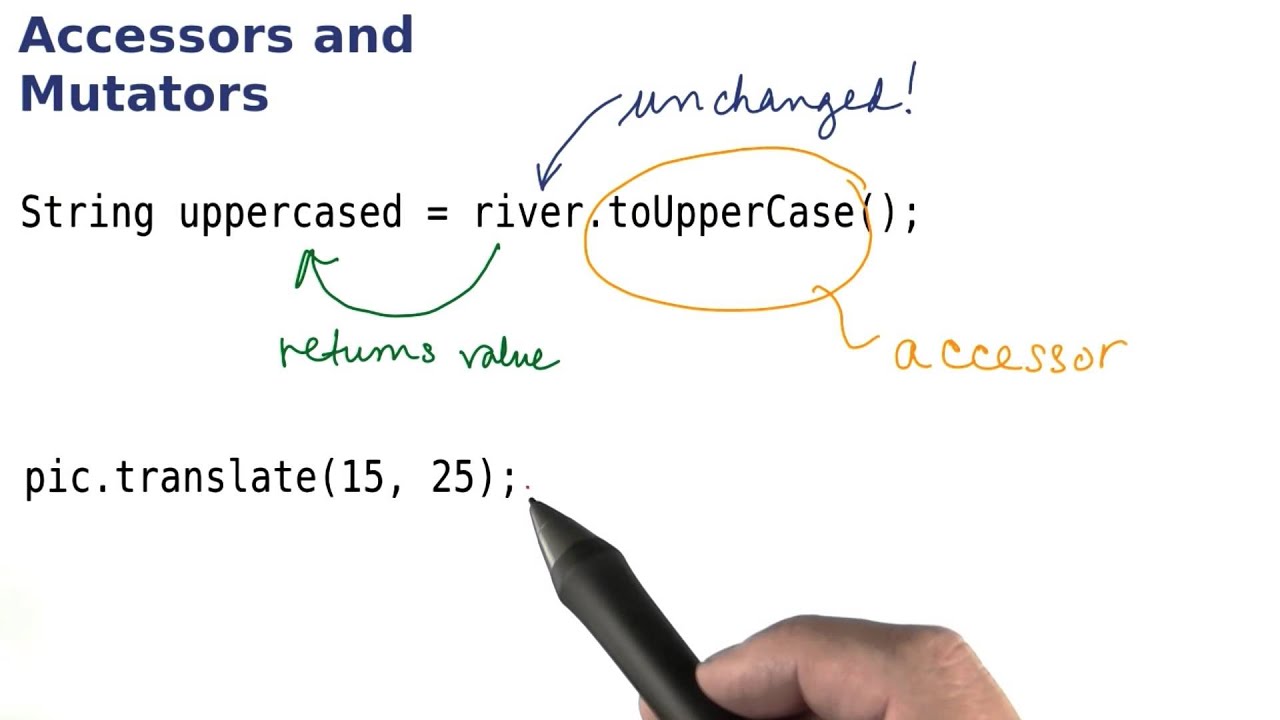
Below are three different constructors. The first one has no formal parameters because it is just a basic default employee. When you make an object using this, the values will be all the default values. The second two can be used to create actual employees with personalized data:

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

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:



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Accessor Methods**

**Can be null as well**

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.

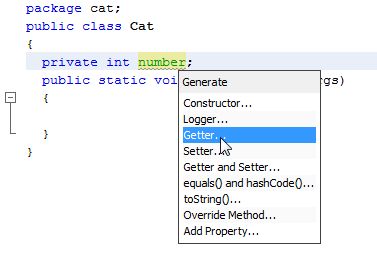
If you run a method on this object, since it is null, it isn’t an object, and it wouldn’t have any methods or constructors. So, if you run a method on it, it will return NullPointerExecption.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Mutator Methods**

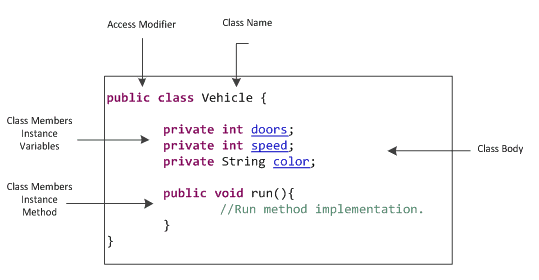
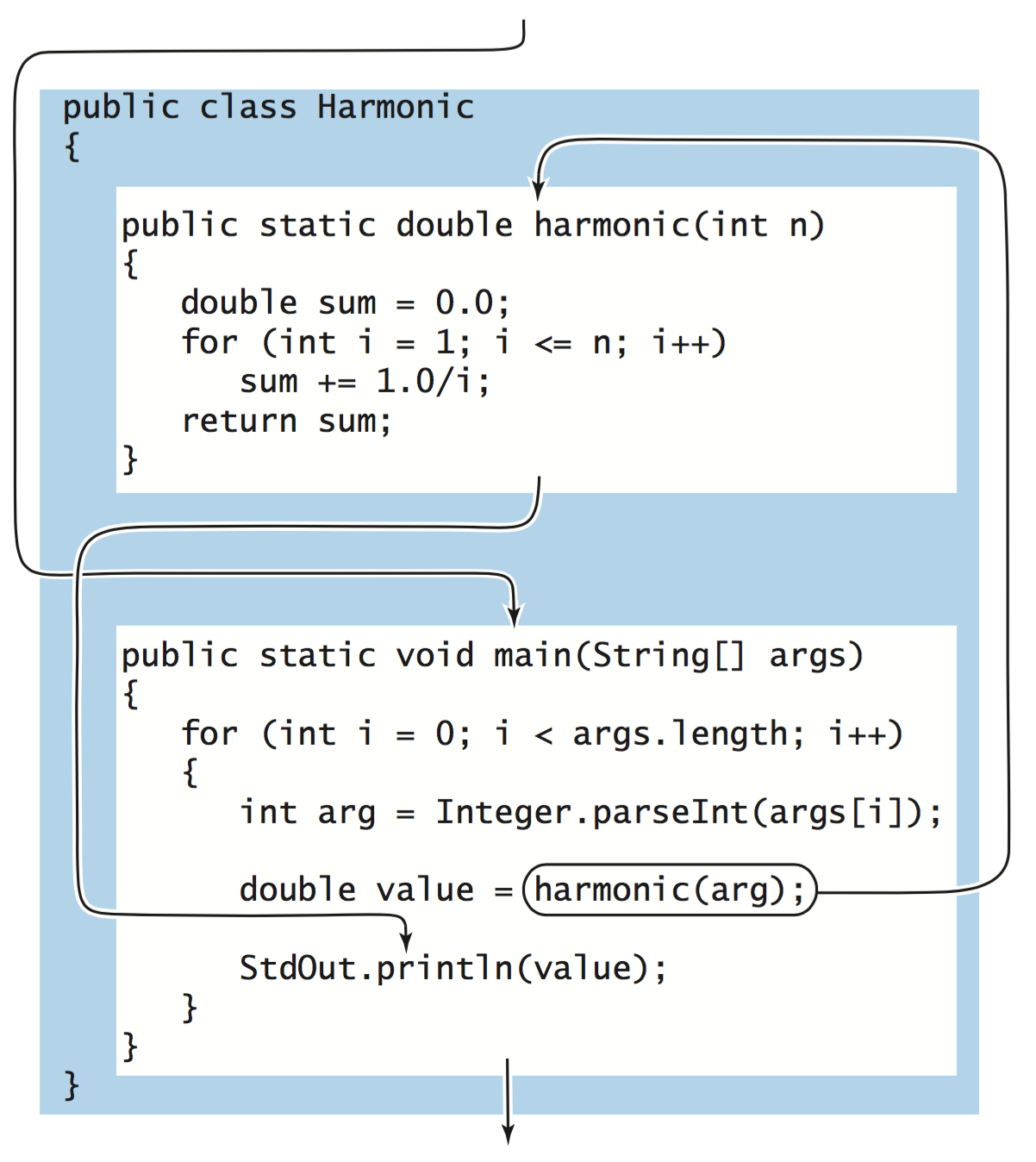
**Void method**:

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:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Static vs. Class**

A static method in Java is **a method that is part of a class rather than an instance of that class**. Every instance of a class has access to the method. Static methods have access to class variables (static variables) without using the class's object (instance). Only static data may be accessed by a static method.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Diff types of Classes**

Static methods can be called by any object

Public method: can perform this.name

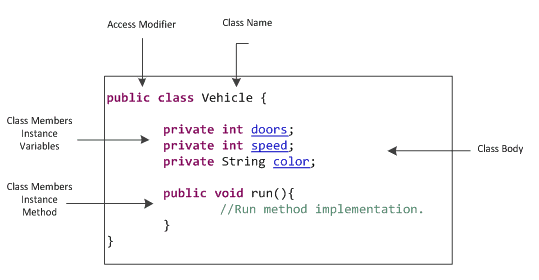
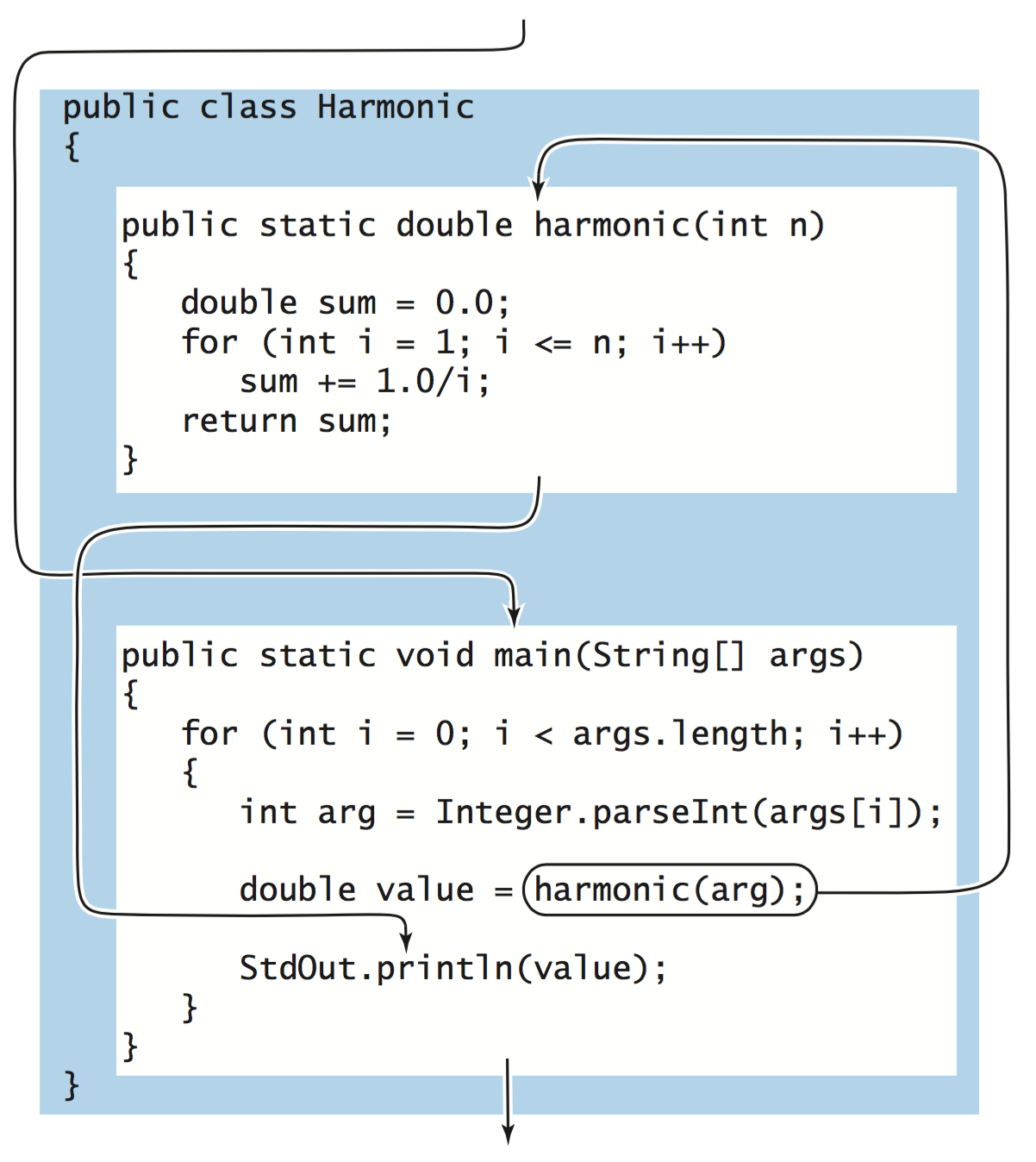
Private: restriction on what part of the code can access.

Protected: just like package-private (default visibility), except that it also can be accessed from subclasses.

****

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Static vs. Class Mathods**

A static method in Java is **a method that is part of a class rather than an instance of that class**. Every instance of a class has access to the method. Static methods have access to class variables (static variables) without using the class's object (instance). Only static data may be accessed by a static method.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**This Keyword:**

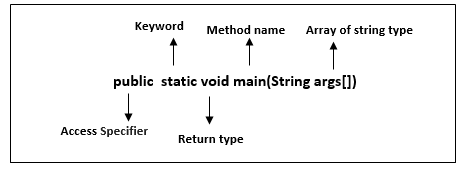
The this keyword refers to the current object in a method or constructor. The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Main and Tester Methods**

The Java main method is usually the first method you learn about when you start programming in Java because its the entry point for executing a Java program. The main method can contain code to execute or call other methods, and it can be placed in any class that's part of a program.



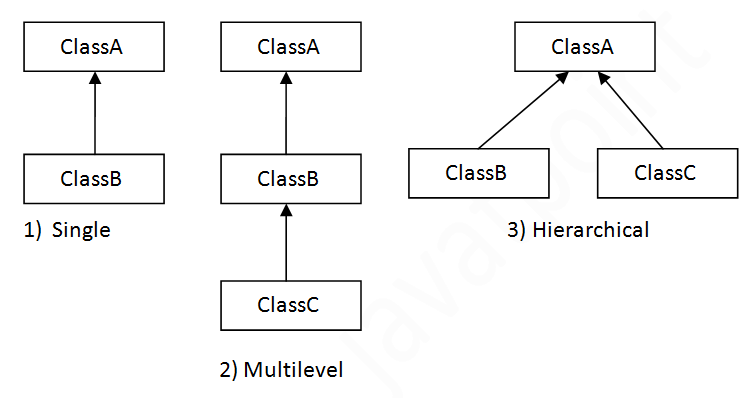
**Non-Void methods**

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.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

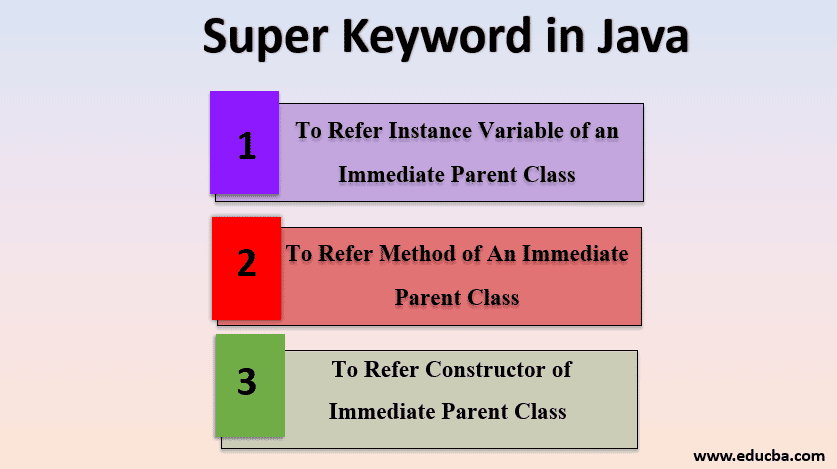
**Inheritance**

Inheritance in Java is the method to create a hierarchy between classes by inheriting from other classes. Java Inheritance is transitive - so if Sedan extends Car and Car extends Vehicle, then Sedan is also inherited from the Vehicle class. The Vehicle becomes the superclass of both Car and Sedan



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

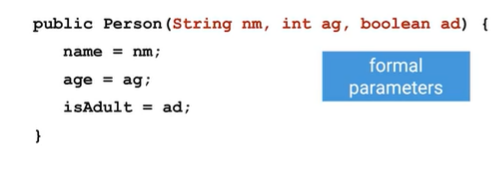
* **subclass** (child) - the class that inherits from another class
* **Super -** is used to call superclass methods, and to access the superclass constructor. The most common use of the super keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Overloading with Constructors and Methods**

Constructors - initialize the attributes for an object



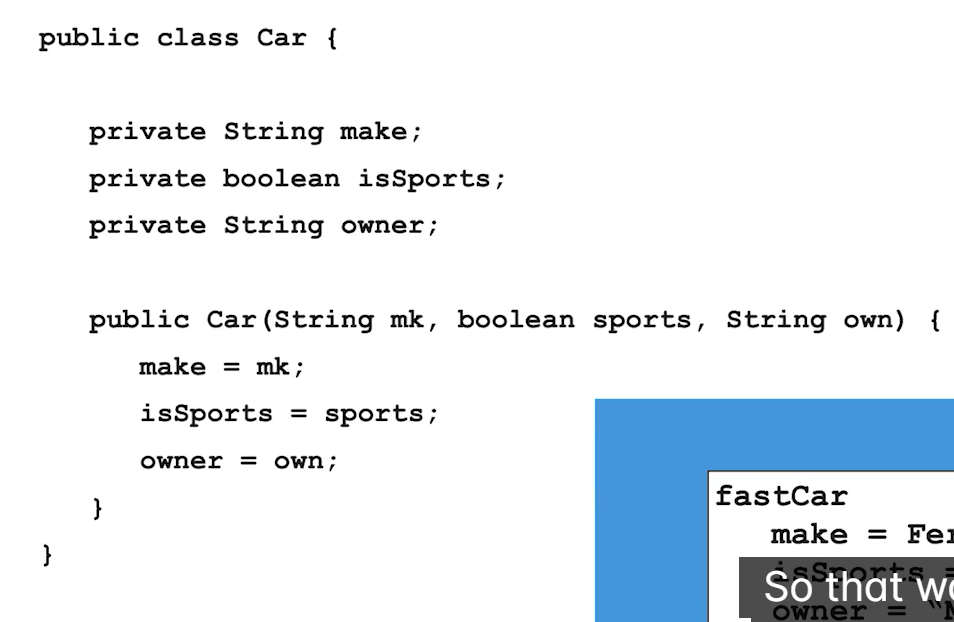
Constructors start with public, and have the same name as the class. In this case, these constructors are for the class Person.

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

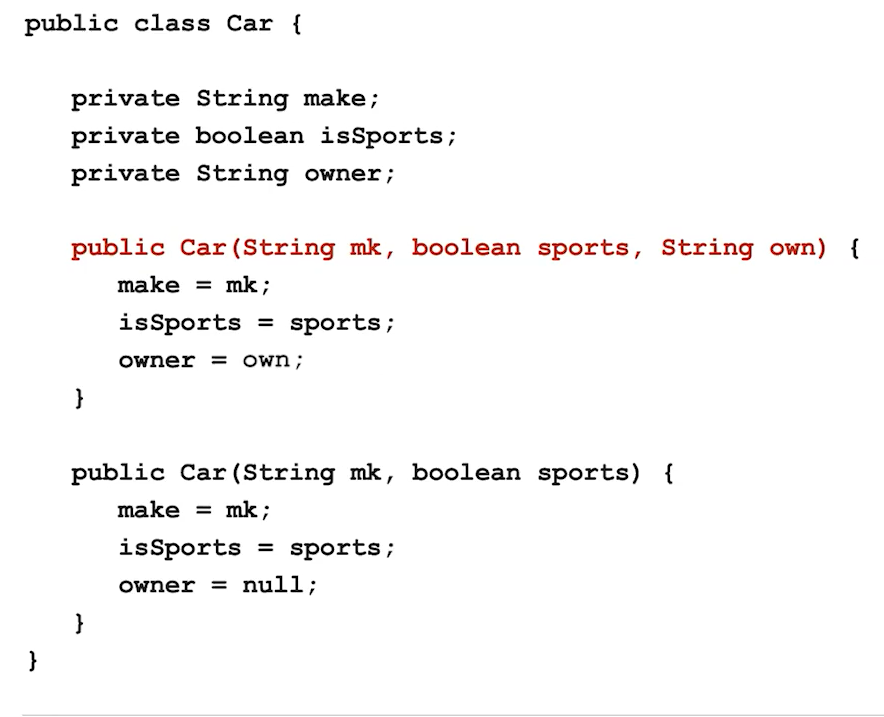
When creating an object, you would have to define those constructors - actual parameters:



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



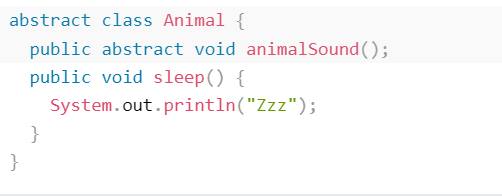
Can have something like this. This is an example of multiple constructors:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

**Abstract method**: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

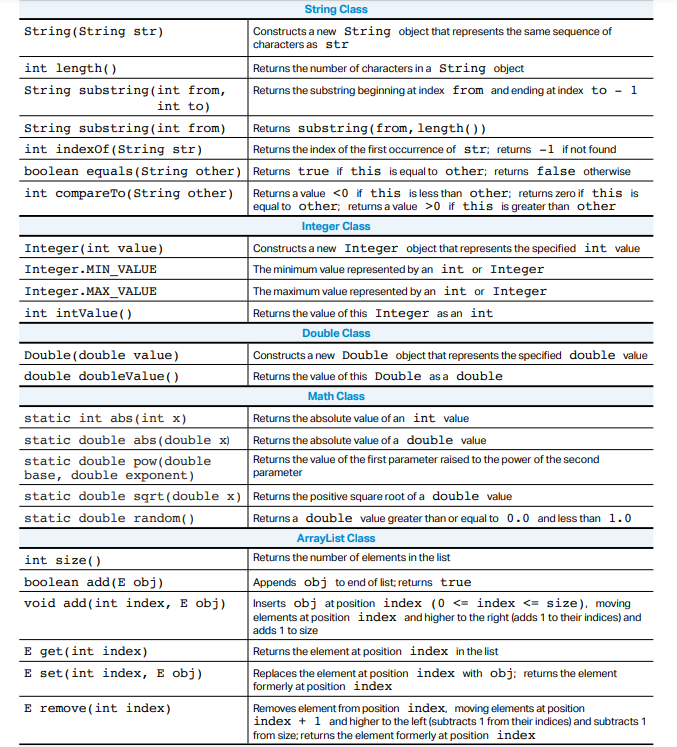


\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Standard Methods**

A method is a block of code that can be called from another location in the program or class. It is used to reduce the repetition of multiple lines of code.

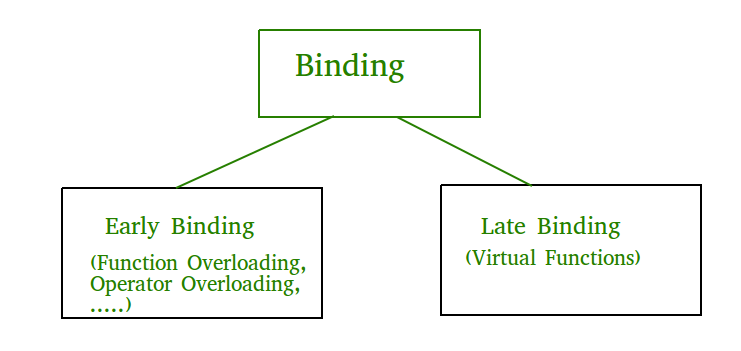
For the AP Exam, need to know these:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

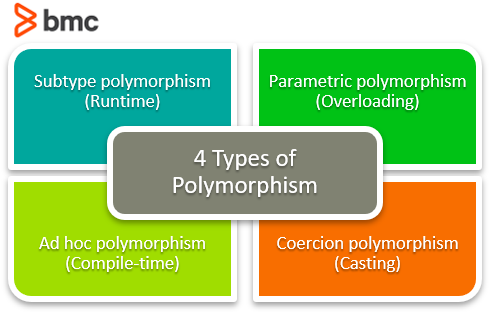
**Late Bounding of Object**

An object is late bound when it is assigned to a variable declared to be of type Object. Objects of this type can hold references to any object, but lack many of the advantages of early-bound objects.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Polymorphism -** is a Greek word that means "many-shaped" and it has two distinct aspects: At run time, objects of a derived class may be treated as objects of a base class in places such as method parameters and collections or arrays.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Big O Notation** - is a tool used to describe the time complexity of algorithms. It calculates the time taken to run an algorithm as the input grows. In other words, it calculates the worst-case time complexity of an algorithm. Big O Notation in Data Structure describes the upper bound of an algorithm's runtime.

