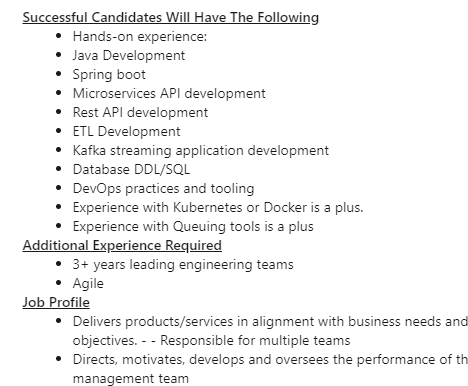
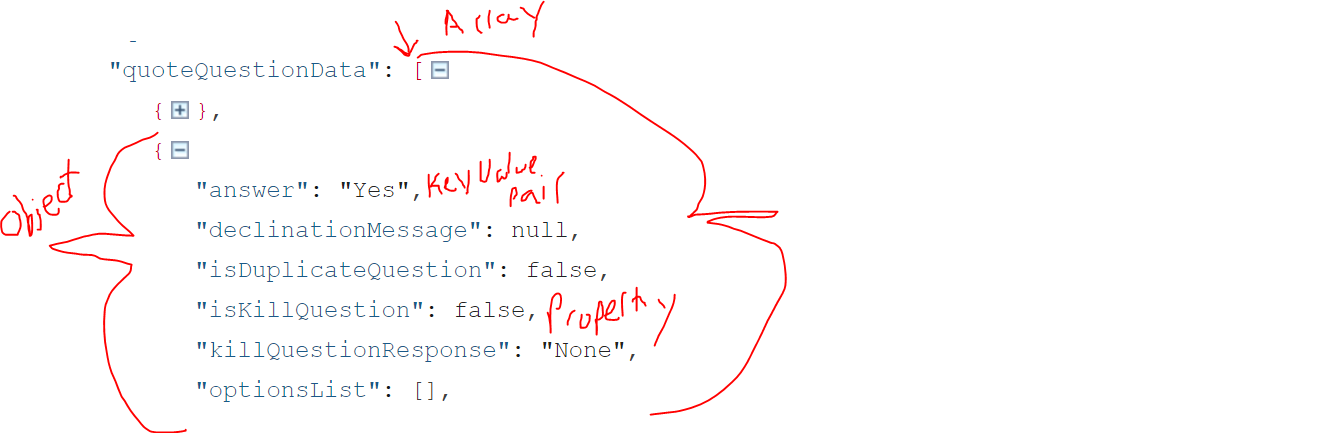
## PNC Job overview



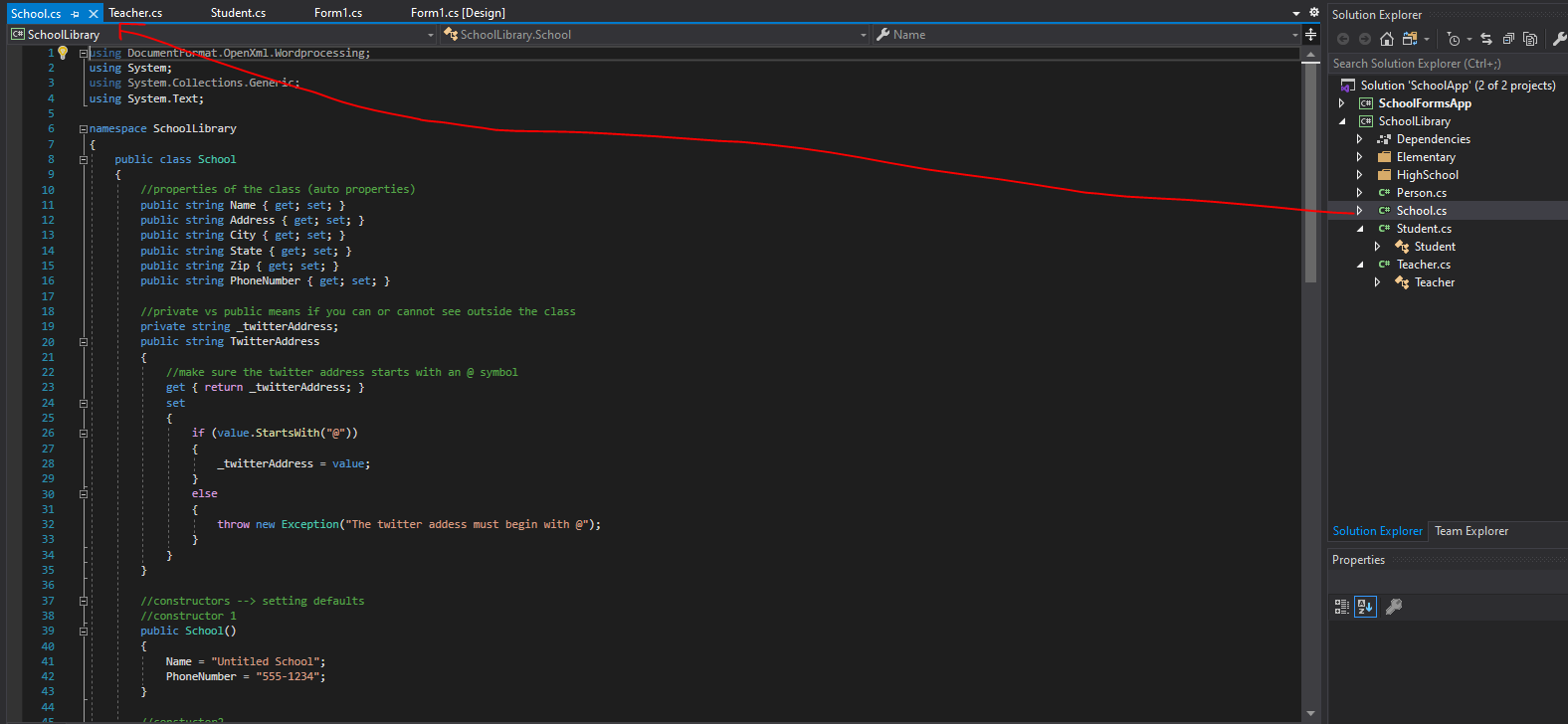
## JSON

### The JSON format

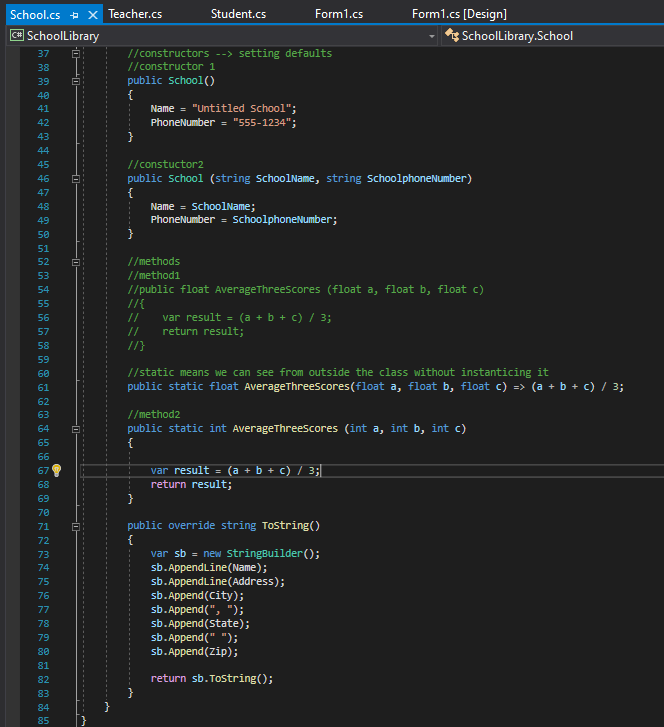
There are just a few rules that you need to remember:

* Objects are encapsulated within opening and closing brackets { }
* An empty object can be represented by { }
* Arrays are encapsulated within opening and closing square brackets [ ]
* An empty array can be represented by [ ]
* A member is represented by a key-value pair
* The key of a member should be contained in double quotes. (JavaScript does not require this. JavaScript and some parsers will tolerate single-quotes)
* Each member should have a unique key within an object structure
* The value of a member must be contained in double quotes if it's a string (JavaScript and some parsers will tolerates single-quotes)
* Boolean values are represented using the true or false literals in lower case
* Number values are represented using double-precision floating-point format. Scientific notation is supported
* "Offensive" characters in a string need to be escaped using the backslash character
* Null values are represented by the null literal in lower case
* Other object types, such as dates, are not properly supported and should be converted to strings. It becomes the responsibility of the parser/client to manage this
* Each member of an object or each array value must be followed by a comma if it's not the last one
* The common extension for json files is '.json'

## C#





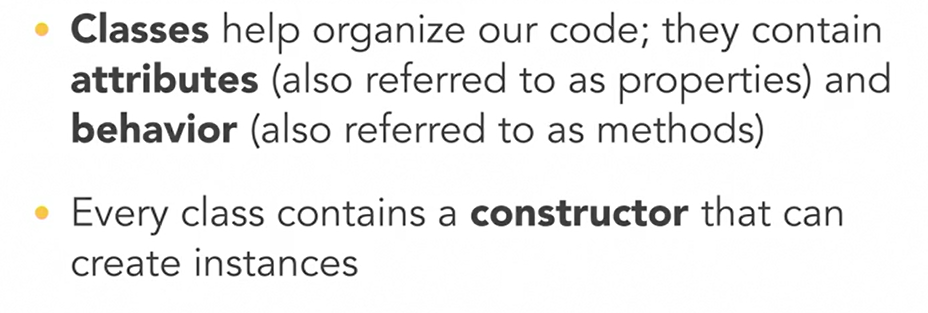


## Python

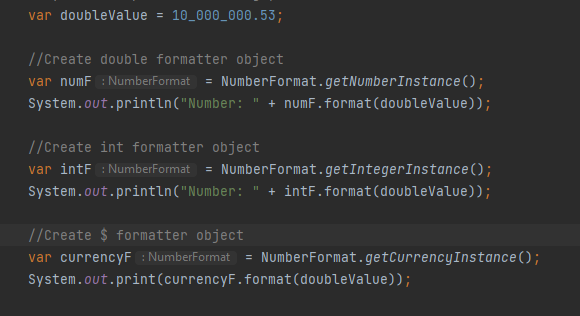


## Java

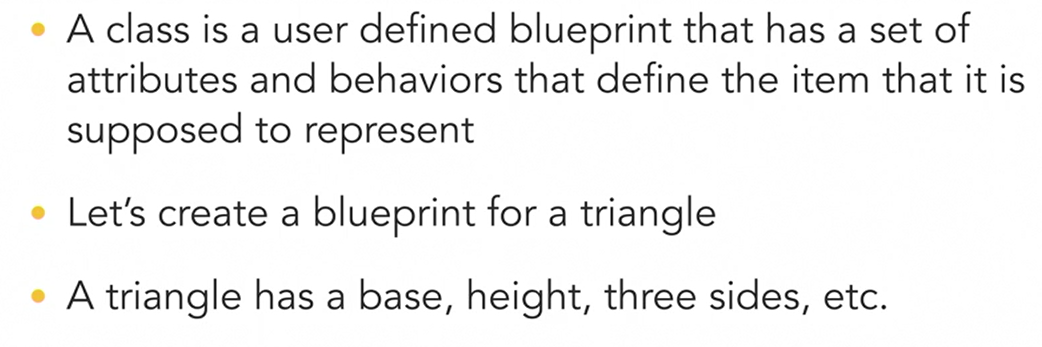
Primitive data types = int, double, char, boolean, short, long, float

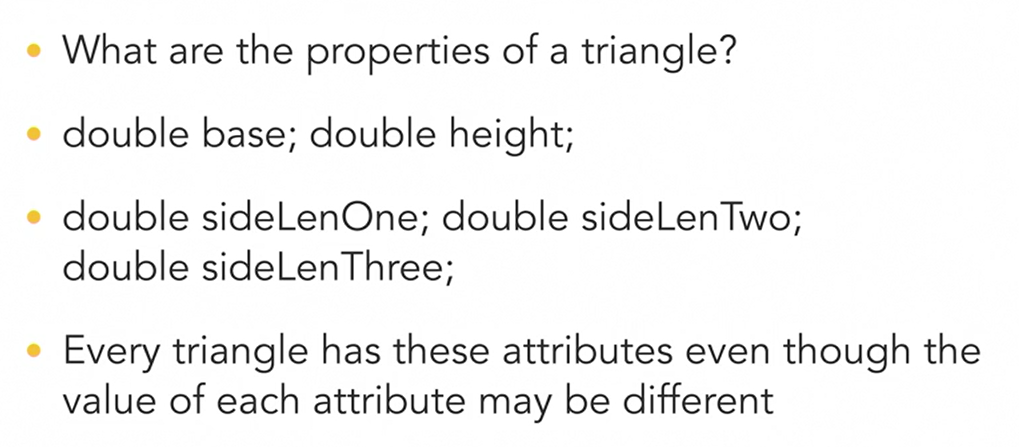


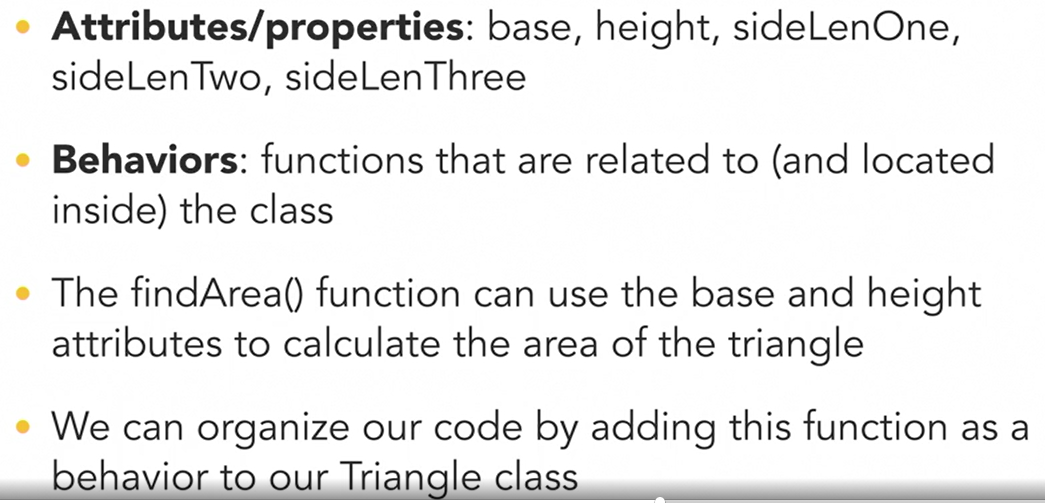
### Format data types

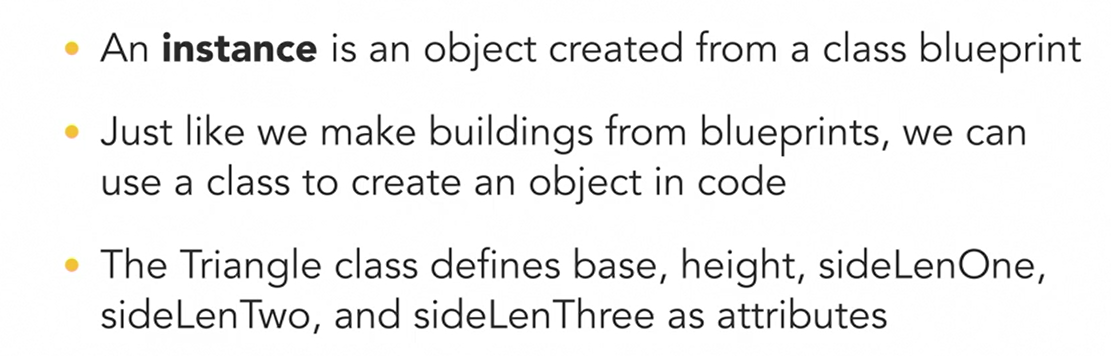


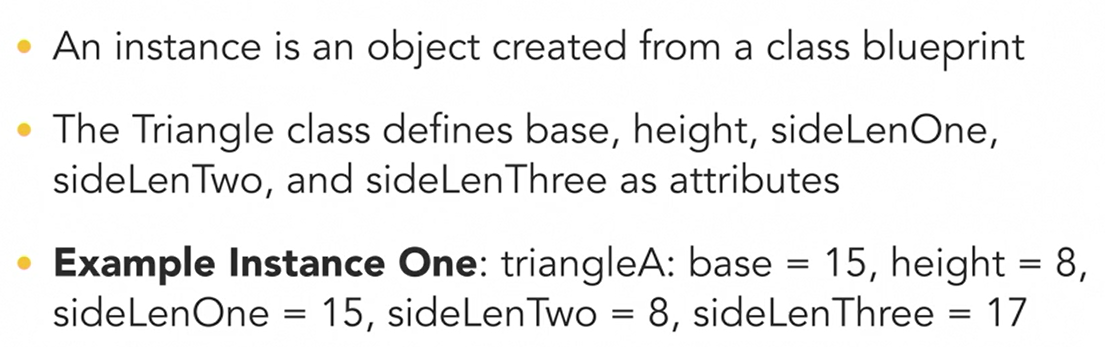
### Classes:



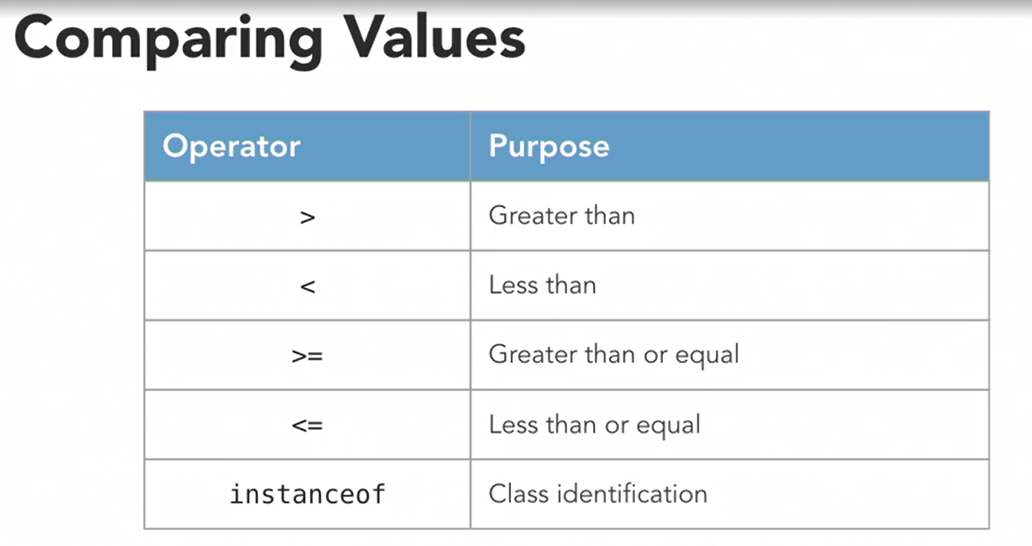


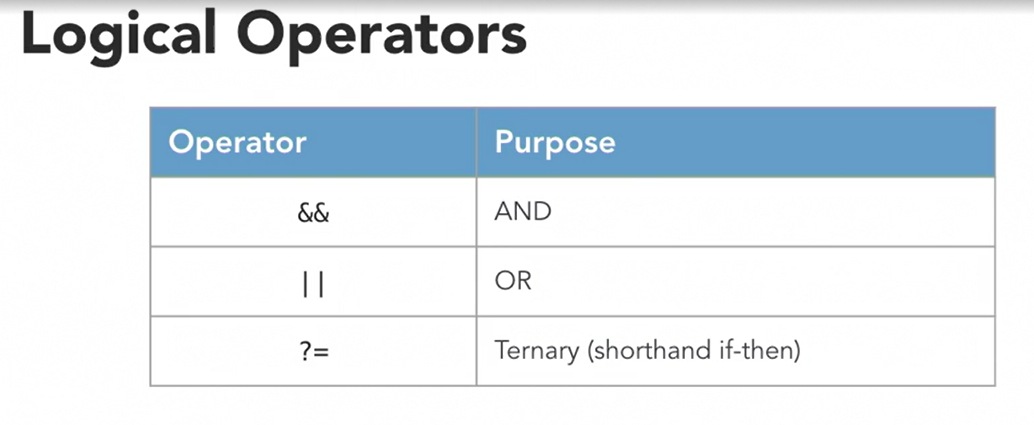




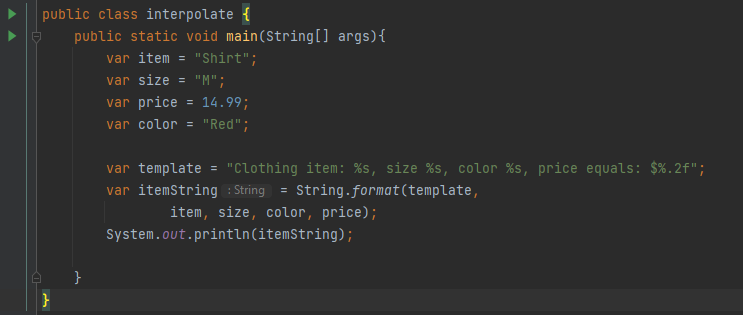


### Operators



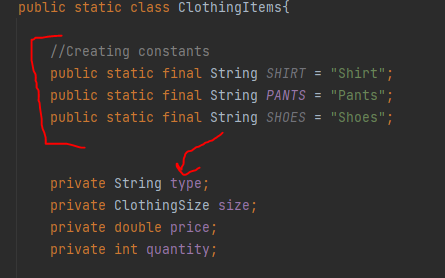


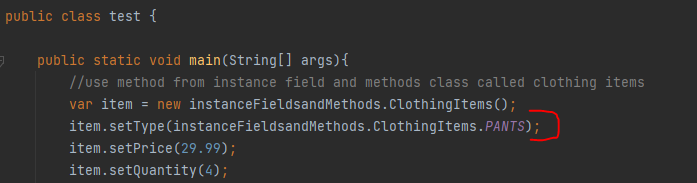
#### Workings with Strings and Interpolate Strings (index, trim, equals, len)

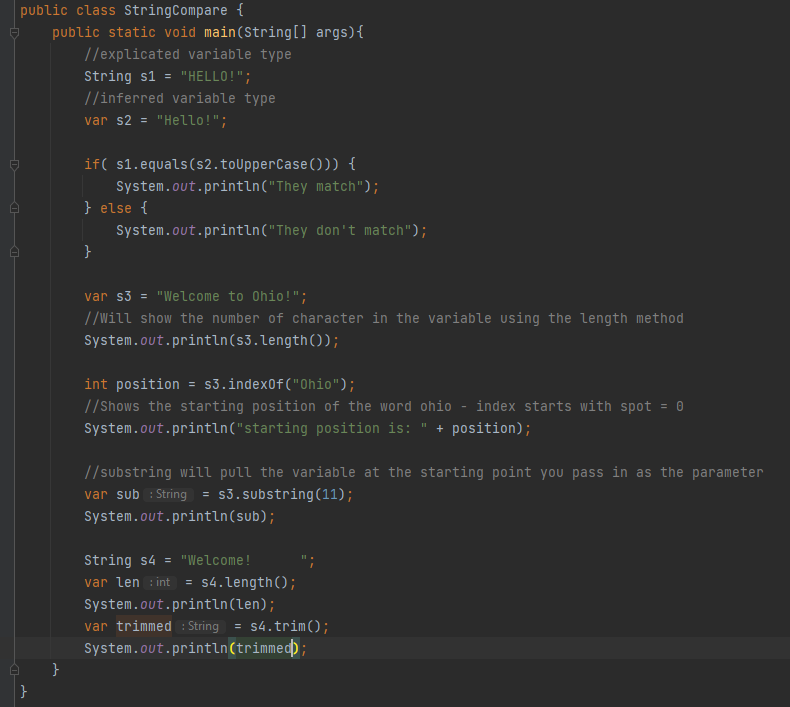


### Constants

Will allow you set options to pass in strings when using the method so that there is no mistyping



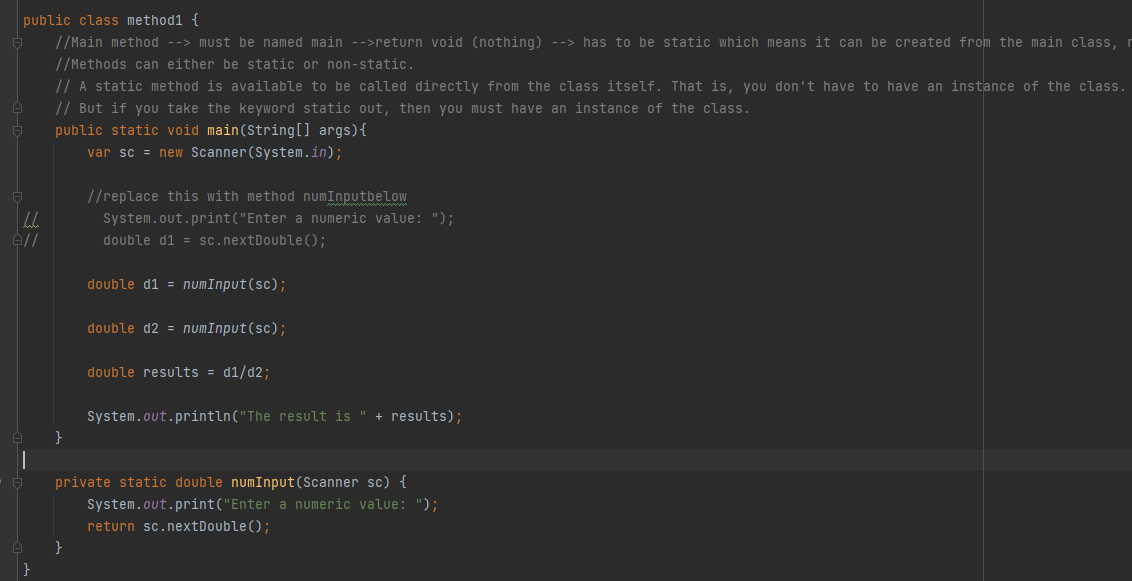


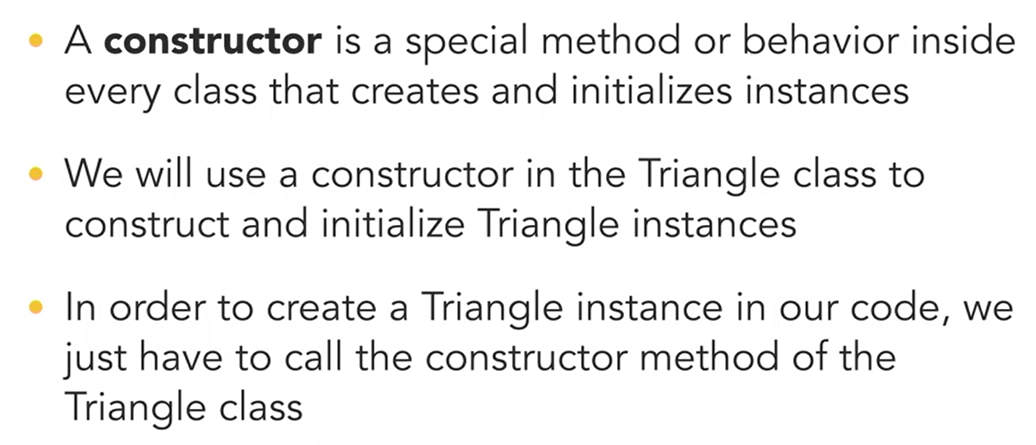


### Constructor

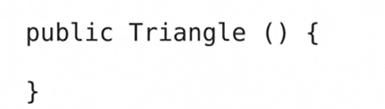
Also a method…Methods always have () at the end

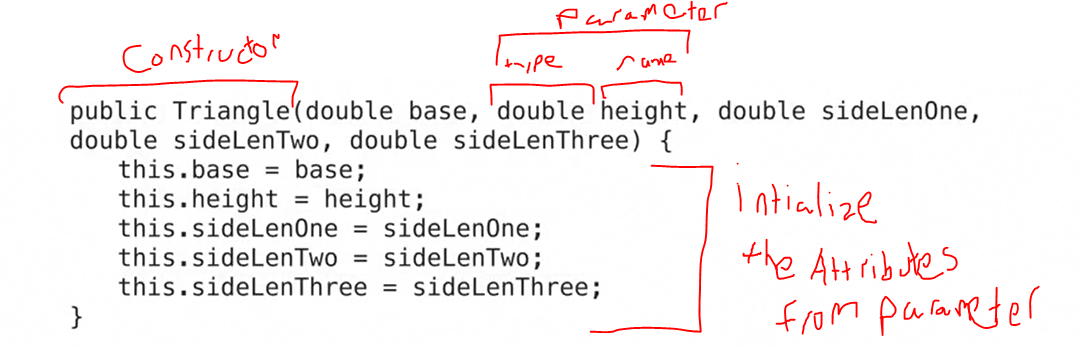
Name of the structure is always the same name of the class



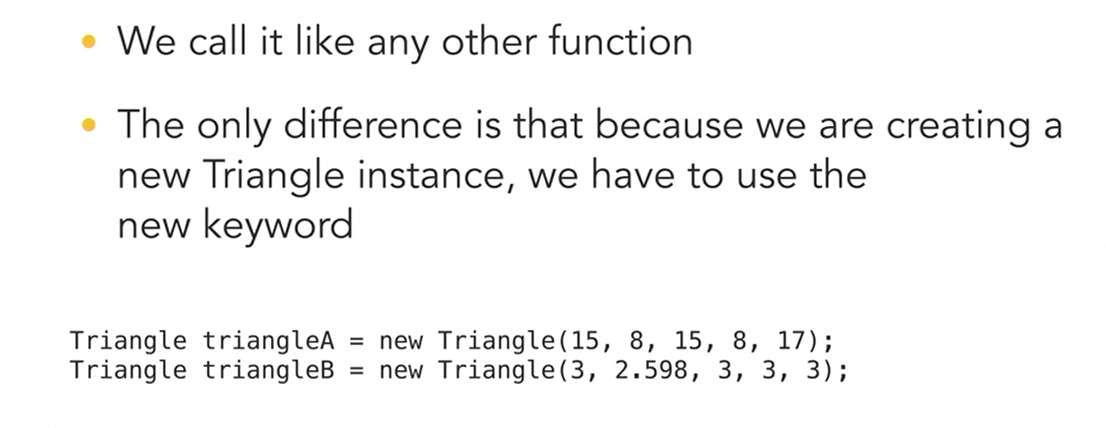


If the class was public class Triangle 🡪 constructor below





Call it twice (2 instances):



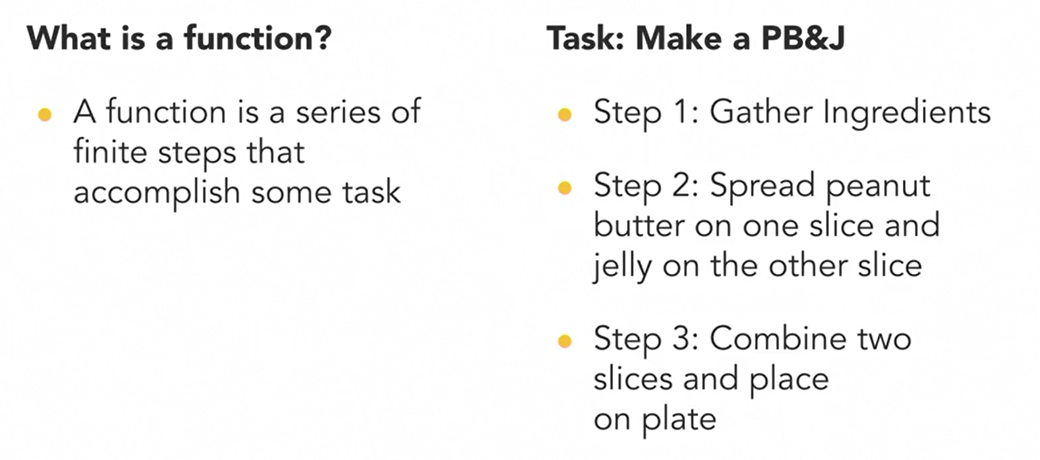
Triangle is now basically a datatype we’re using to create those variables

### Functions/methods:

Method is just another name for a behavior or function that belongs to a class

Organize your code in a meaningful way

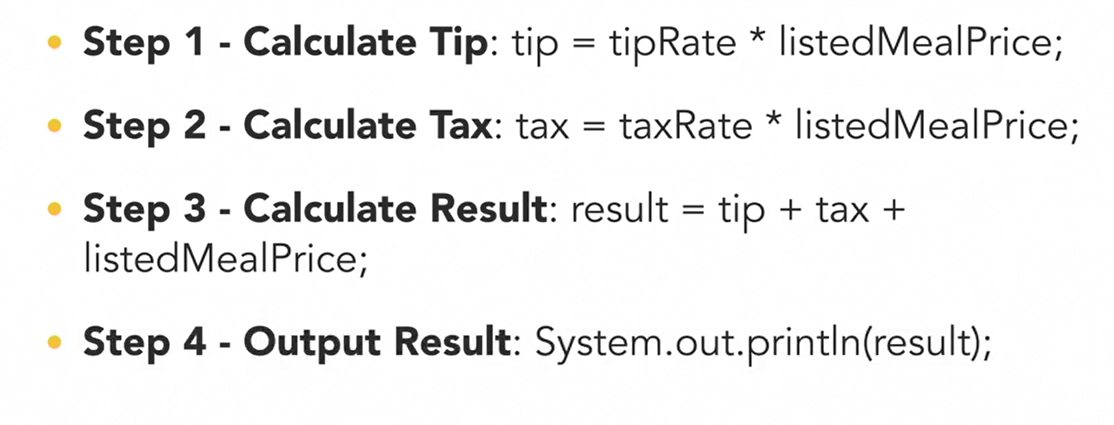
Easier to make change

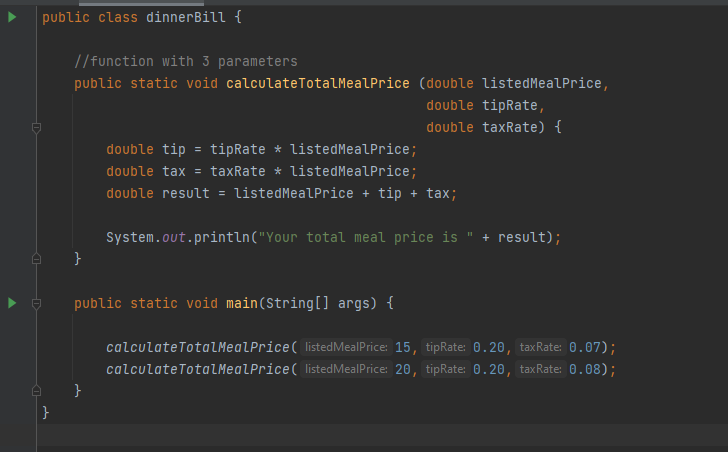


Function = makePBAndJSandwich();

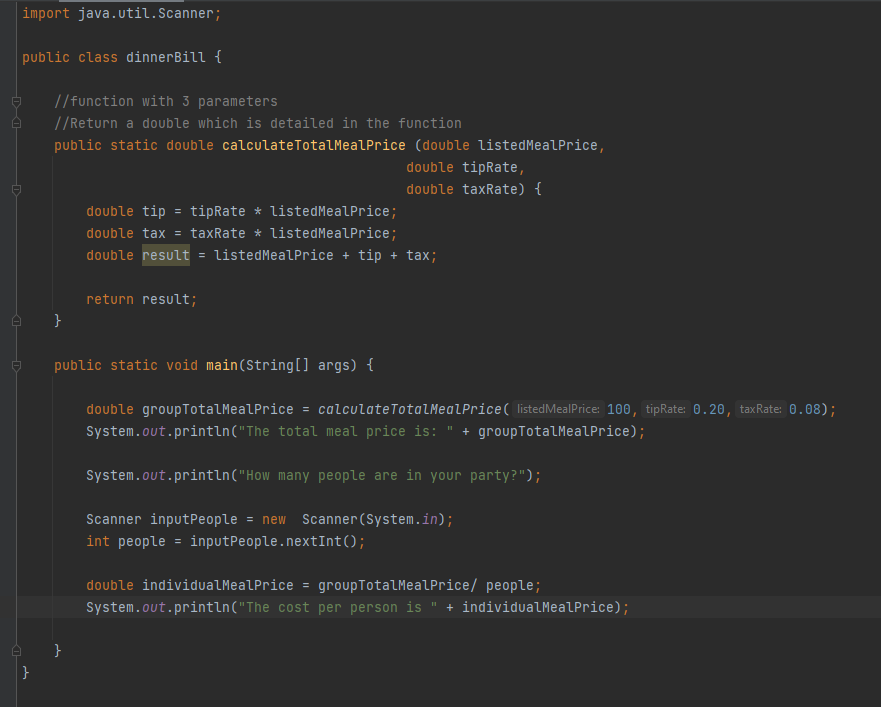
Logic = I don’t need to tell someone how to make PBJ every time. I can write the steps for how to do it…then I can call the person/function to do it

Parameters in a function:



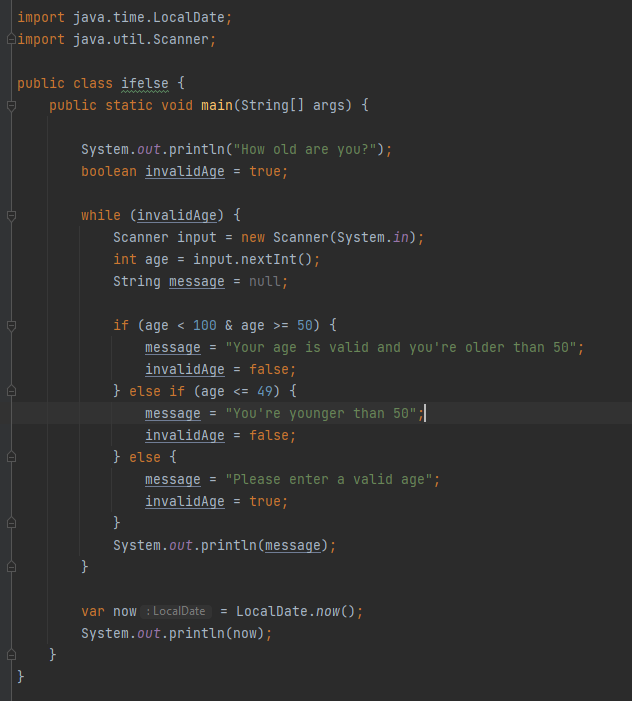


Function with parameters and a return:

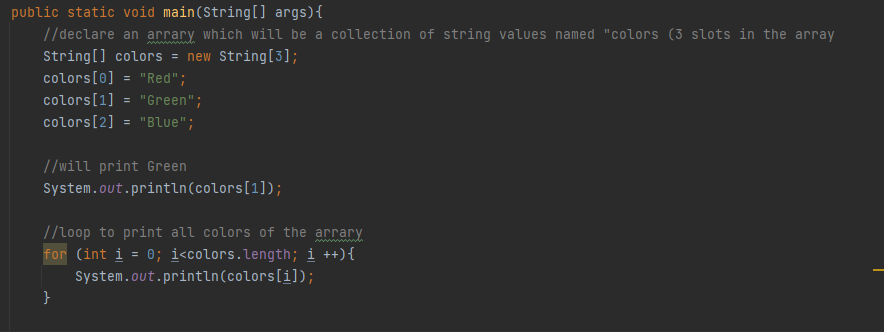




### While loop with if statement to print out message



While loop again with an array and printing the array

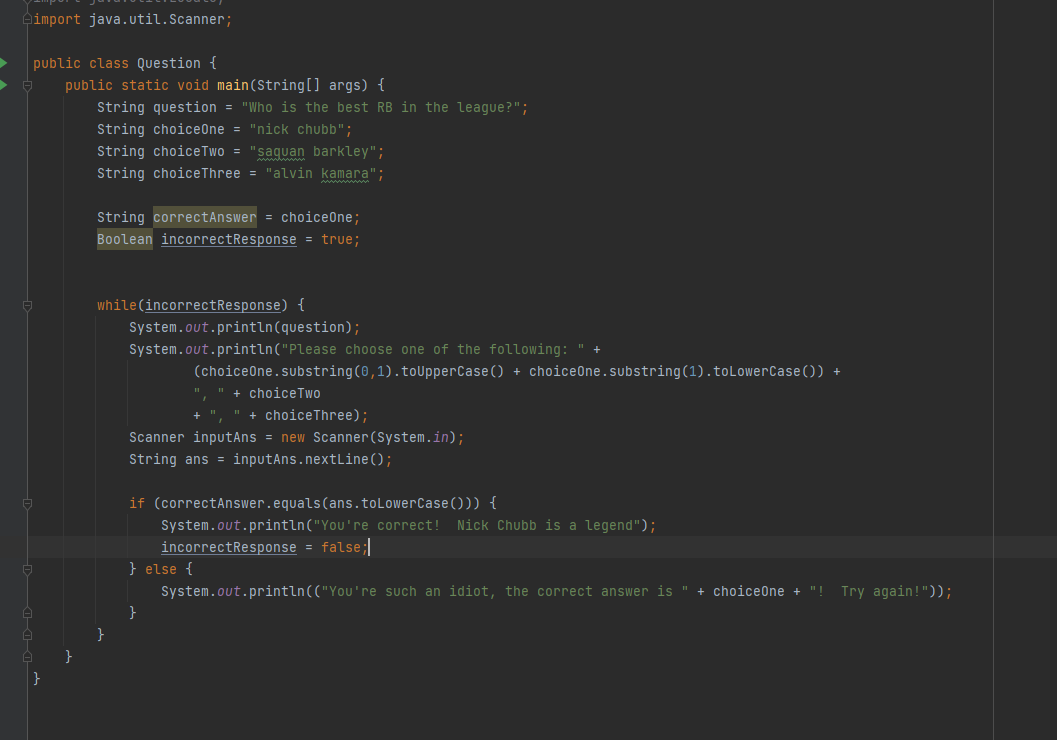




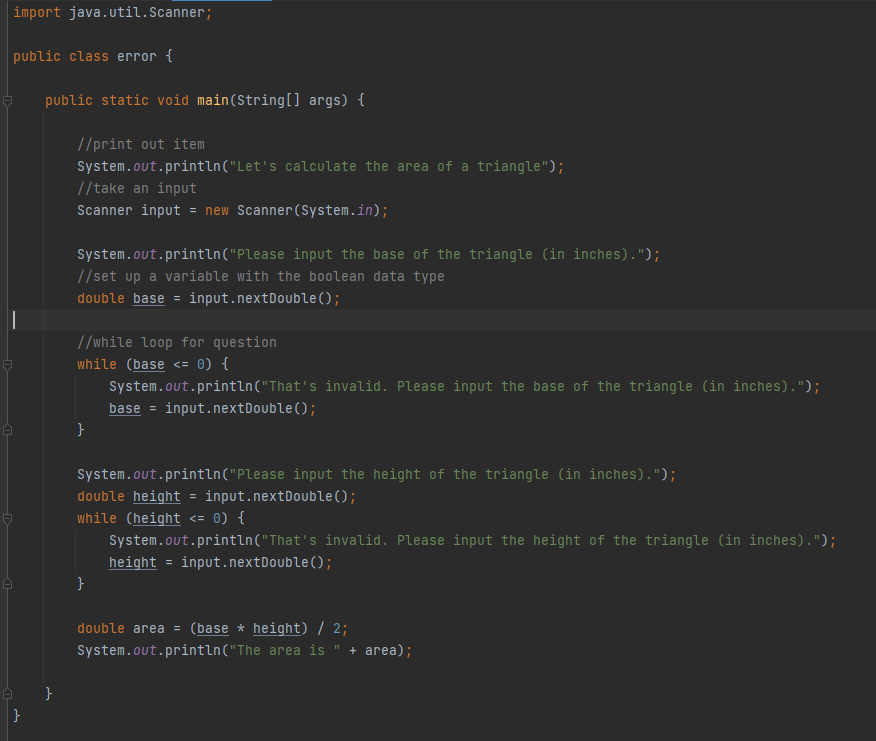
### Mathematical comparisons



### Examples 1

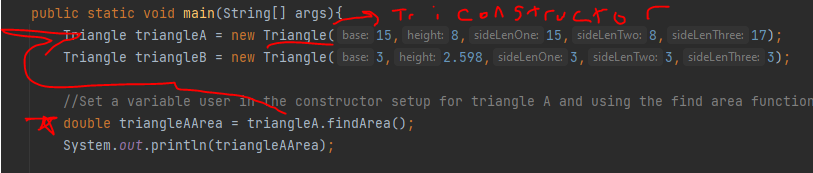


### Example 2



Static vs non-static variables

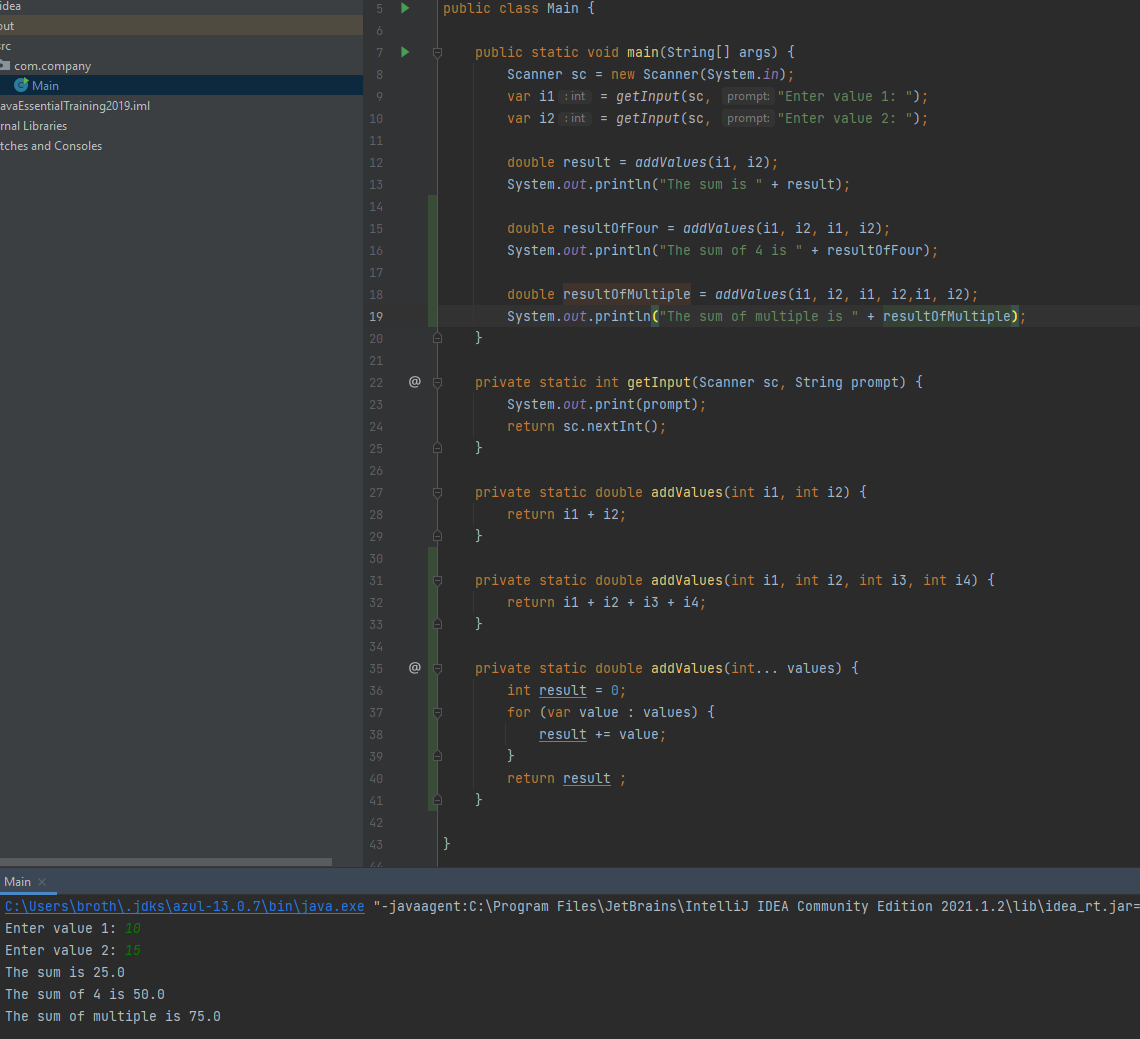
Non-static variable are values which are NOT the same each time – their values will chance depending on the instance



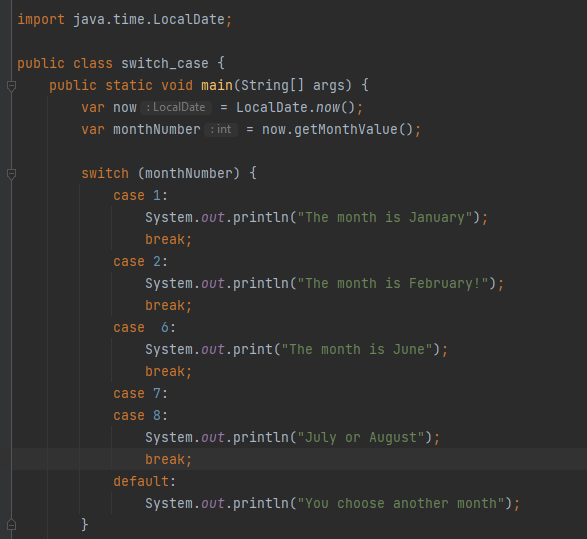
Static variable – do not change per instance. They hold a value for the class to use.

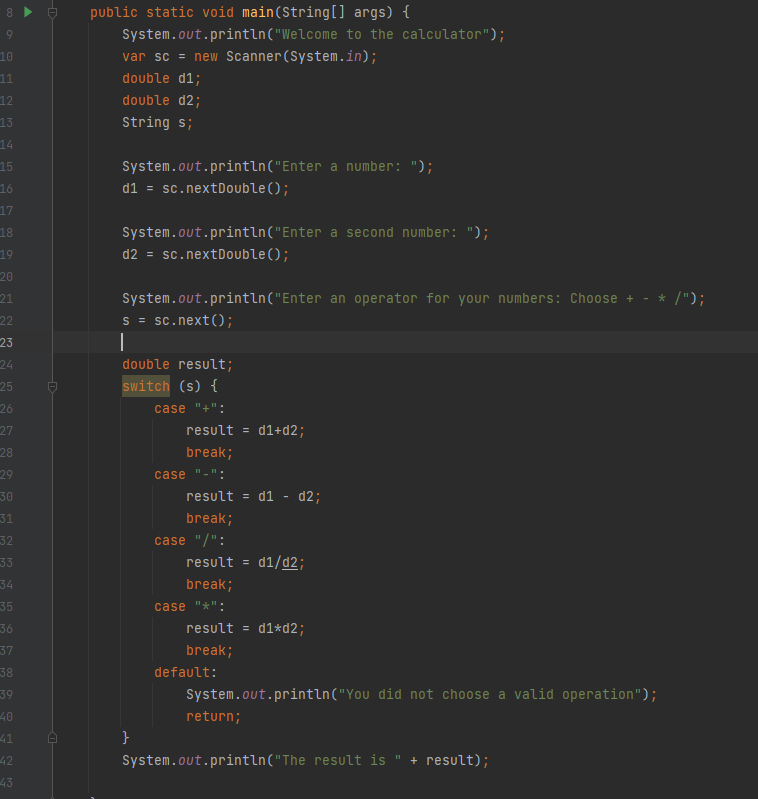
int Num1 = 20;

example 3

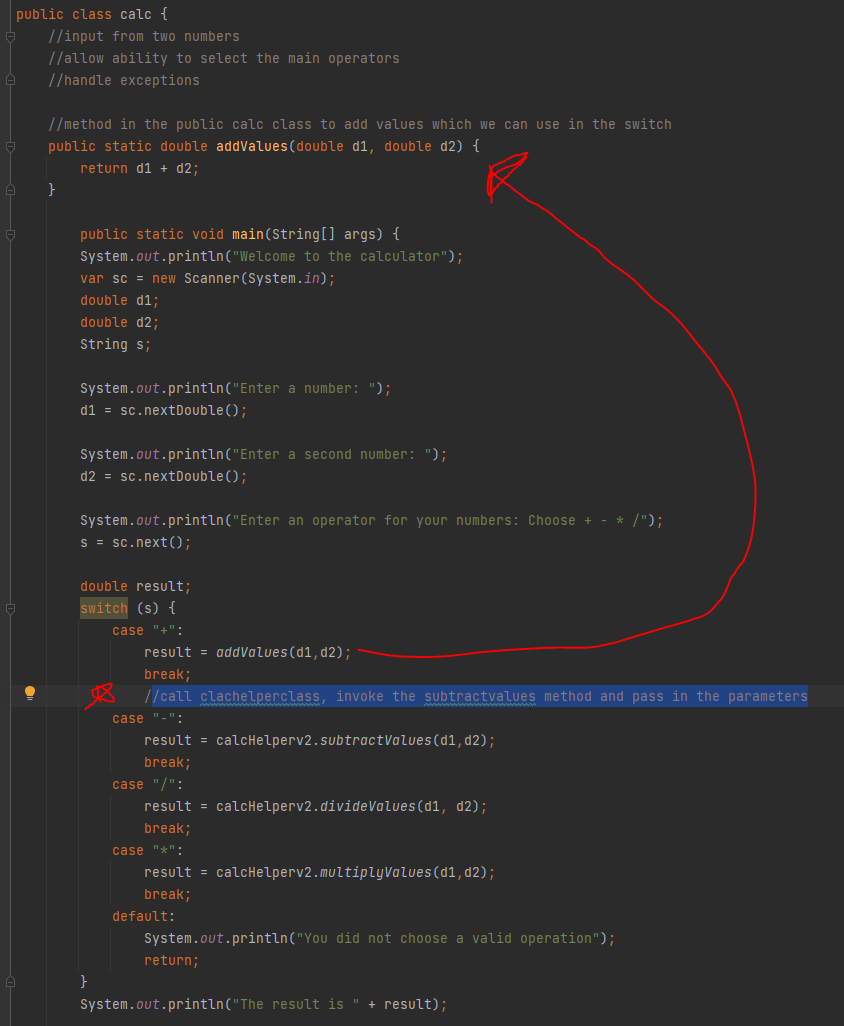


### Switch expression

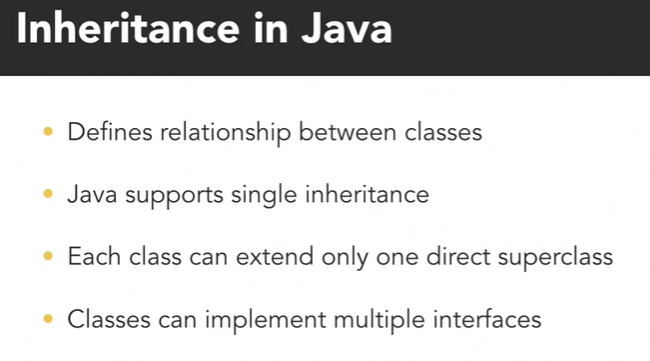


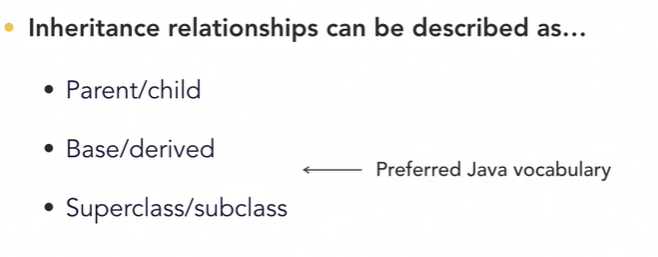


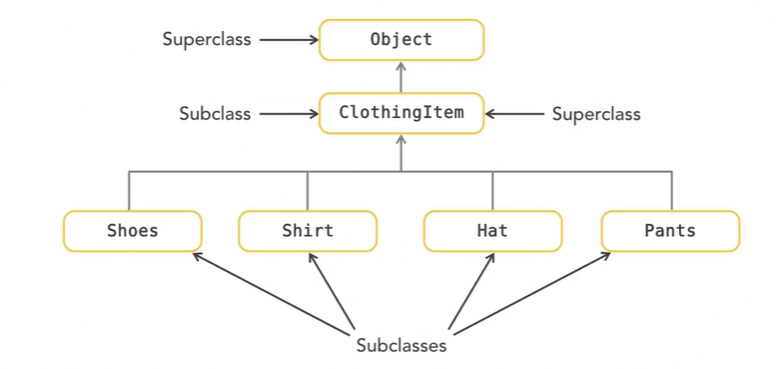
#### Call clachelperclass, invoke the subtractvalues method and pass in the parameters



### Inheritance







## Branching

* Start a new user story and branch off release 🡪 Name Branch “Story\_xzy”
* Develop Branch – merge all changes during the initial development to develop
  + (Story\_xzy to develop)
* IT and UAT Signoff – when go to staging, merge all of the changes into the release branch
* Deploy the release branch to production
* Merge all new commits from release to main/master branch for post production
  + Then merge all of the changes to develop after the push – not necessary for functionality. But helps with commit history
  + 