* Method

Methods are pieces of **reusable code.** They take **parameters** as input and return a **value** as output. They can contain any number of instructions within. Therefore, a method is just an algorithm.

Why use methods?

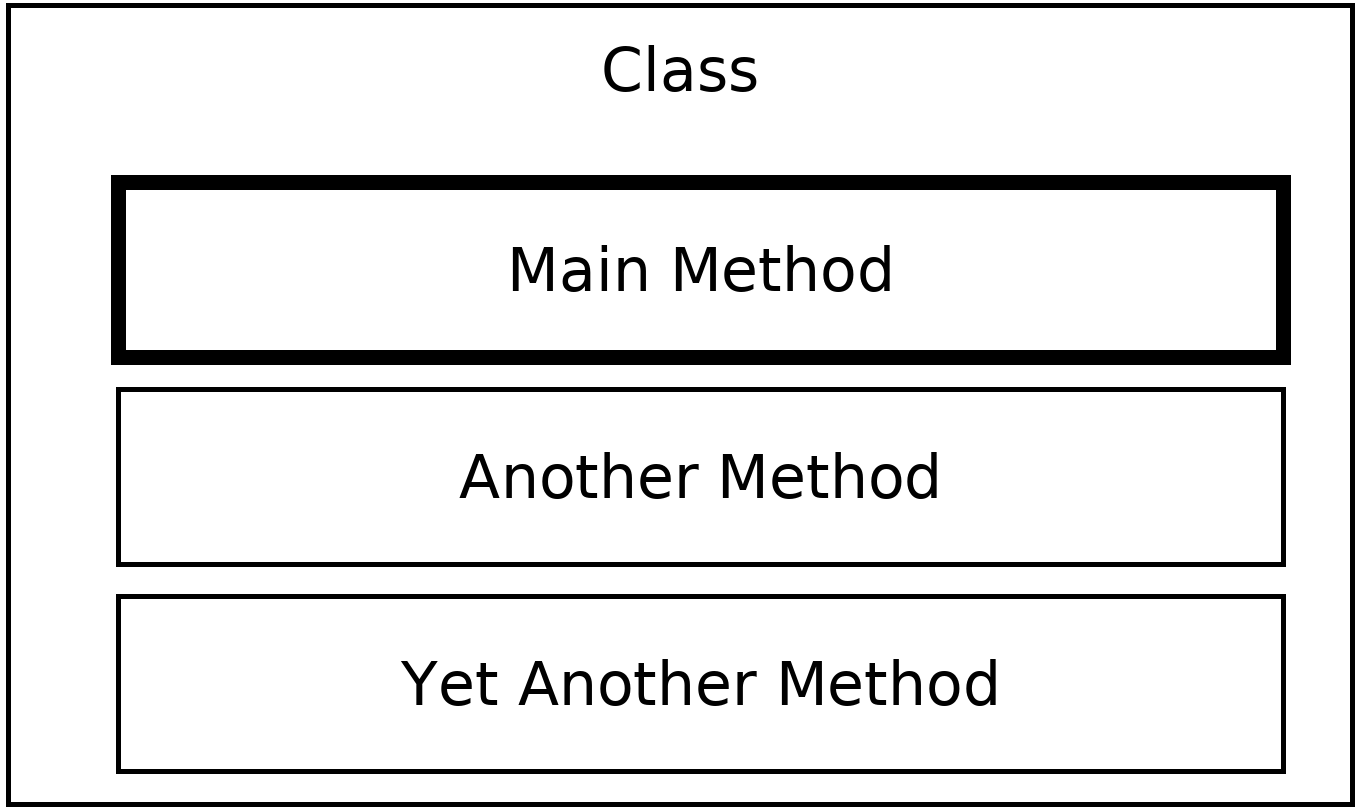
Makes it easy to divide our program into smaller algorithms.

Reusing the Math.sqrt method allows us to avoid re-writing it.

We hide all the unnecessary details.

Note that Java will automatically make sure that the call has the correct type.

For example, we can’t call the Math.sqrt() method and pass it a String

* Structure of a program

In our programs, we will have **class**es which contain many **method**s.

Within each of these methods, there are **instruction**s which may or may not be executed by Java.

Usually there is a **main method** in a class.

The class usually contains many other methods that we write.

IMPORTANT:

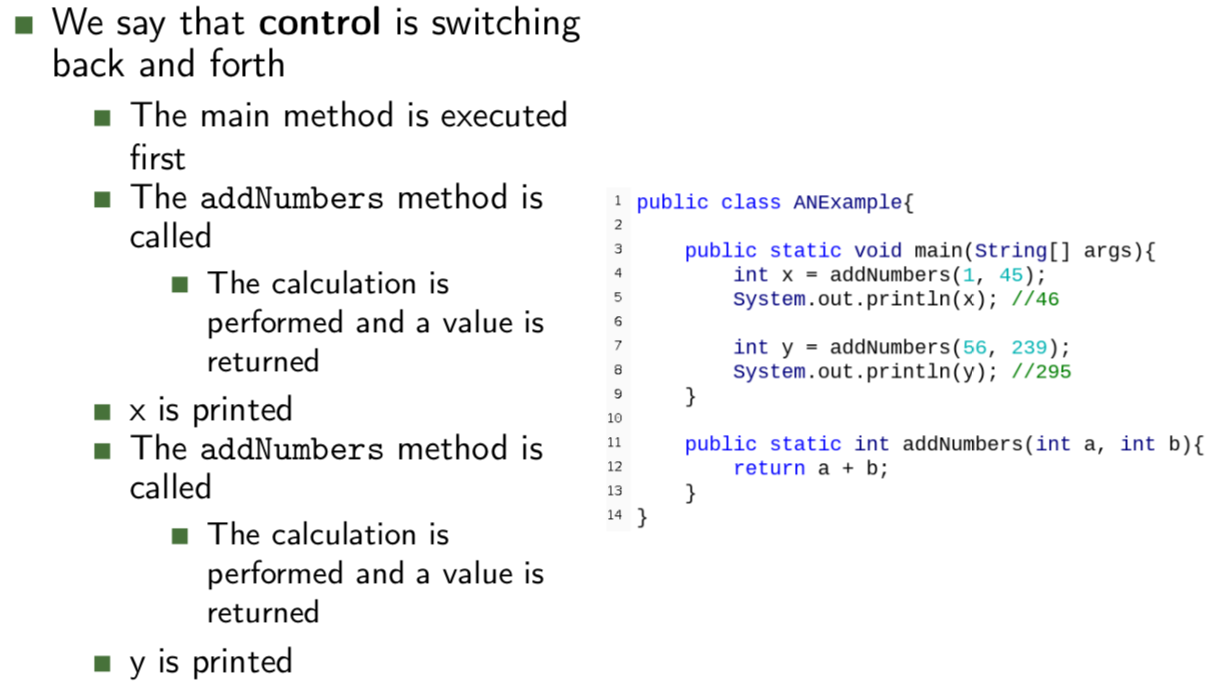
1. The main method can go above or below the other methods(it doesn’t matter)

2. When program runs, Java executes instructions starting with the main method

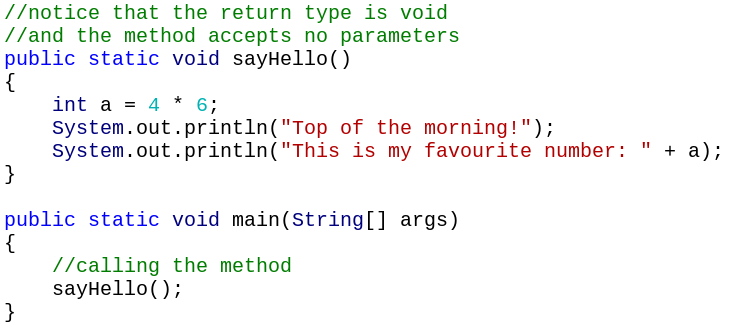
3. The instructions in other methods are not executed unless the

method is called

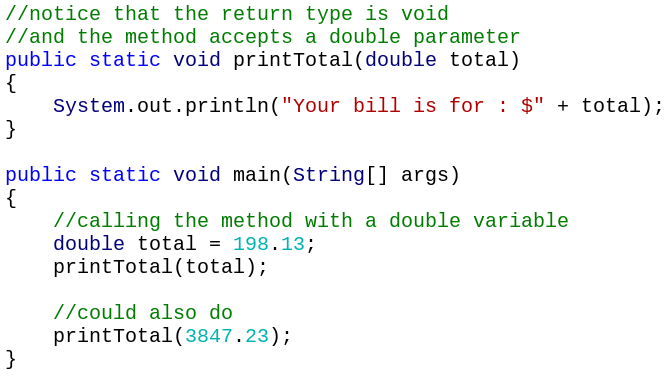
* Execution location



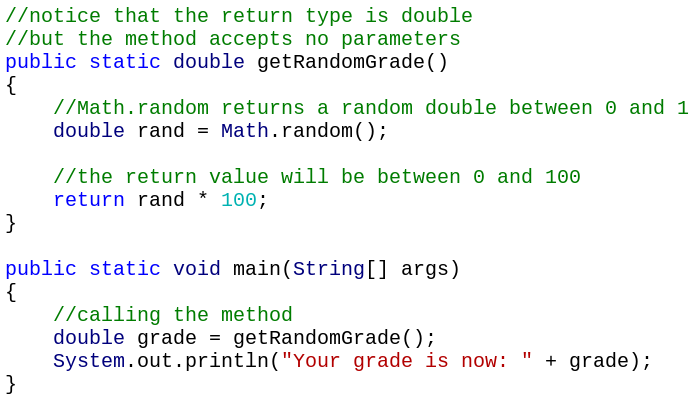
* Method types
* No input and no return value(May be useful for printing a welcome message)



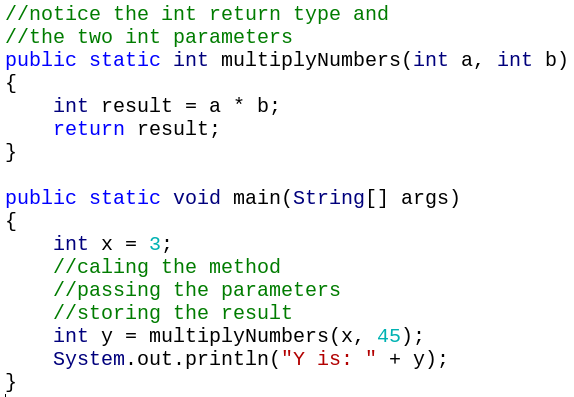
* Input but no return value (print out a total)



* No input, but a return value (less common)



* Input and a return value



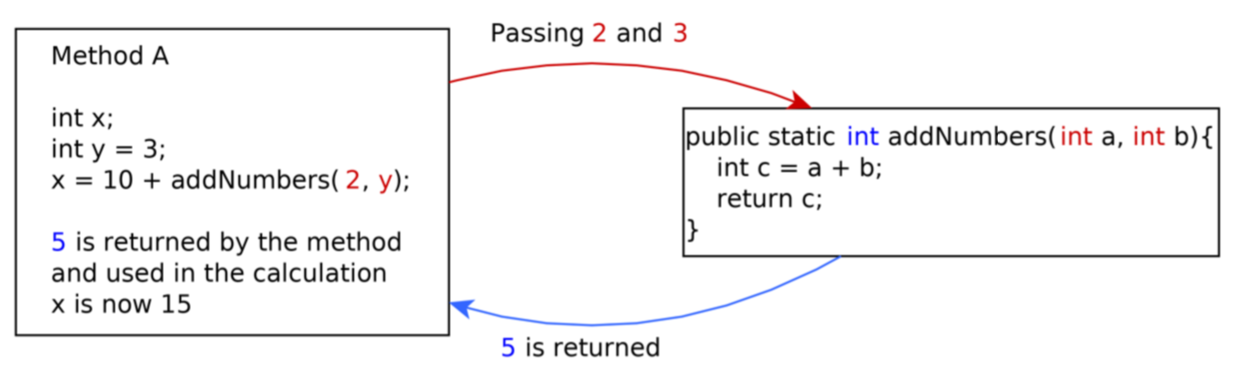
* Method Calls

We can **call** the method with or without passing parameters.

Methods can call themselves.

A value may or may not be returned.

* Note that printing something does NOT count as returning a value
* The **return** keyword returns a value



* Four Parts of a Method’s Description

The return type

The method name

The parameters the method accepts

A brief description or purpose (in comments to let others know what this method does)

* Example: addNumber Method

The four parts turn into the first line (**method header**) of the method:



Method name - addNumbers

Input - Two integers

Output - An integer

Purpose - To add the two input parameters.

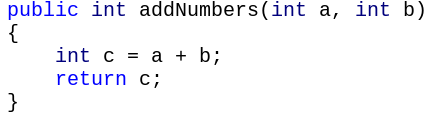
Why is it int a and int b?

Need to give the parameters names so you can refer to them in the method’s instructions. a and b are assigned values when the method is called

These variable names have **no connection** to variables in other methods

Variables only exist in the method they are declared. We call this the variable’s **scope**.

* Return Statement



The return statement says which variable to output

We can only output one variable, and the type must match the header.

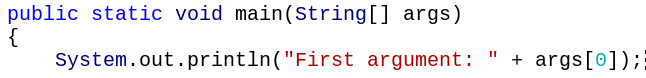
* Main method



This is a method that does not return anything (return type is **void**)

And has one parameter called **args** of type **String[]**

**String[] args** are parameters passed to the program. This is specially done when Java starts the program. This is why we need to have String[] args as the main method’s parameter



* System.out.println Method

This method is **called** on the System.out variable (this is a special variable), and does not return anything. Note that this method will accept ints, doubles, Strings, or booleans.

* Math method
* Math.sqrt(double a)-----a1/2

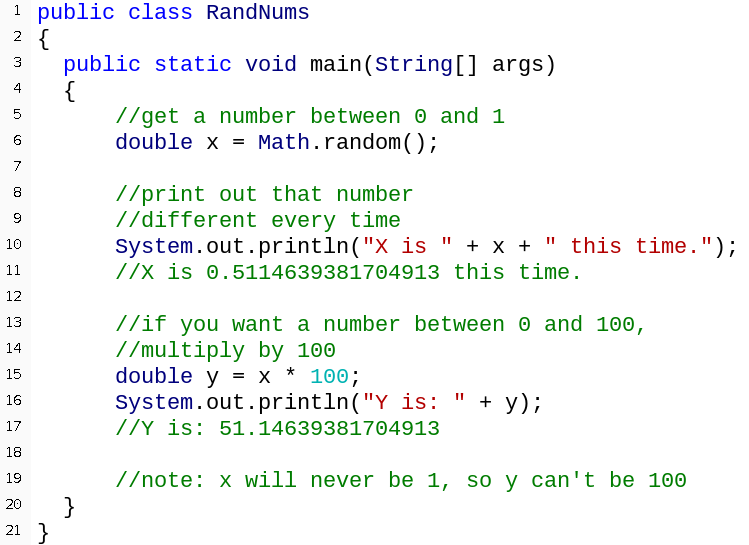
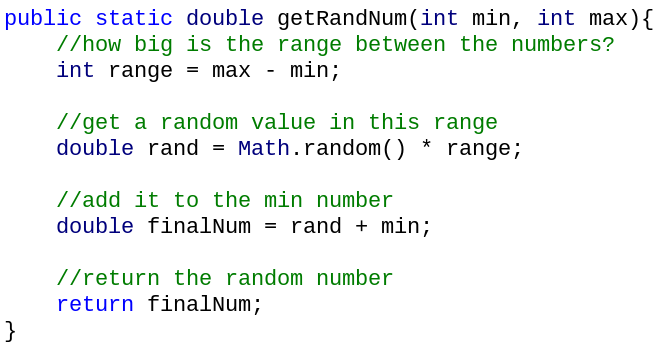
Returns the correctly rounded positive square root of a double value.

* Math.pow(double a, double b)-----ab

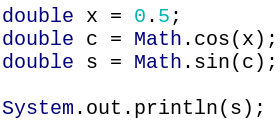
Returns the value of the first argument raised to the power of the second argument.

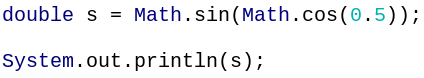
* Math.random()

Return any number between 0 and 0.99999… (but never be 1)



* Nested method calls



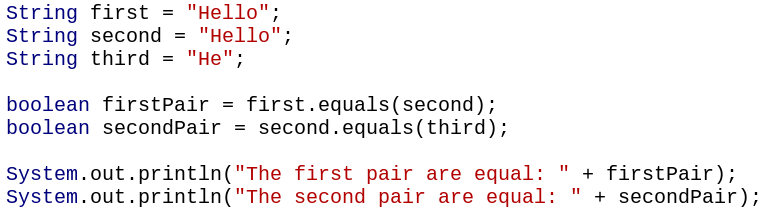


* .equals() Method

To test if the variable equals the String liberal.

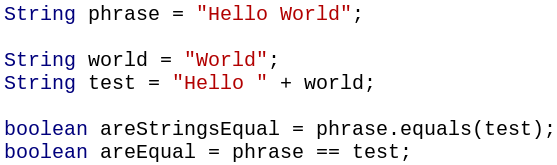
This method is **called** on a String variable, it accepts or is **passed** a String, and it **returns** a boolean value of true or false.

The variable’s name is followed by .equals() with the String literal inside the .equals() brackets.



We are **calling** the equals method on the first variable, and **passing** the second variable.

* == vs .equals()



The first boolean stores true while the second stores false.

Use the .equals() method for comparing Strings, == for everything else

* Strings into numbers

int x = Integer.parseInt(“123”);

double y = Double.parseDouble(“56.4”);

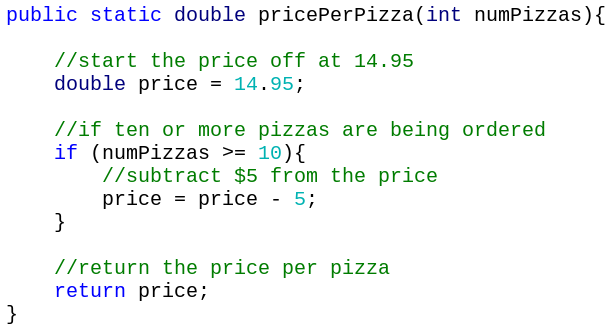
boolean z = Boolean.parseBoolean(“true”);

**An error is produced if the String doesn’t match the type**

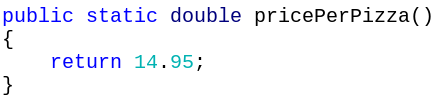
Integer.parseInt(“3.5”) doesn’t work

Neither does Double.parseDouble(“Hello”)

* If statement:

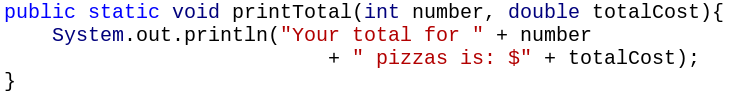
We have added an **if statement** to the method. If what’s inside the brackets (the **if condition** evaluates to true). Then the code inside the **if block** will be executed.

* EXAMPLE: Pizza Ordering
* pricePerPizza: the price per pizza



* printTotal: take the number of pizzas (int) and the total cost (double)

returns nothing (void)



* Main Method: get the price per pizza and perform the total cost calculation