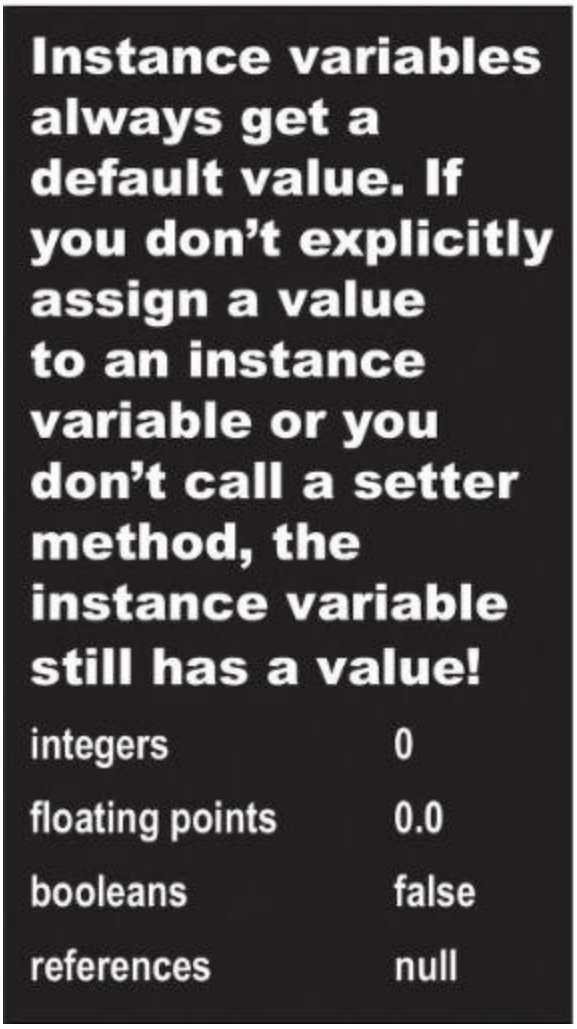
Chapter 4 important notes: How Objects Behave

- Remember that the class describes what an object knows and what an object does
- Every instance of particular class has the same methods but the methods can behave differently based on the value of the instance variables
- A caller passes arguments and a method takes parameters and the parameter is nothing more than a local variable
- You can get things back from a method too that's called return type and you must define the return type of each function when you define it if you do not want from it to return anything just make it void function otherwise make it return any data type you want
- You can send more than one thing to a method
- Java is passing by value that means it passes by copy and that's mean when you passing an argument the parameter takes copy of this argument not taking the original argument so if you change this parameter inside the method it will not change the original variable after the method finished
- You can return anything that can be implicitly promoted to that type. So, you can pass a byte where an int is expected. The caller won't care, because the byte fits just fine into the int the caller will use for assigning the result. You must use an explicit cast when the declared type is smaller than what you're trying to return
- Now we will start speaking about the encapsulation and how to apply it in our code and the first thing to make this is by using the access modifiers so Mark instance variables private , Mark getters and setters public
- The benefit of using getter and setters here one of things is if you do not make a setter for instance variable so any body in the code can change it cause it public and out another value not illegal like if you have a height and then you access it in somewhere in the code and you put its value with -1 so it is really something bad so otherwise if you will use setter so inside the setter code you can put your boundaries and that's very good benefit of encapsulation so you can add your checks and your boundaries that's fit in your variable okay and the second thing is that if there is anything changed you will just change inside the setters and getters you will not change the 100 dot operators that accessing the public instance variables cause as we know the change is the only thing is constant and what about if you maybe change your mind latter you will just change one place rather than changing multiple ones .



- and that's as we will know later cause of the default constructor that initialize them with their default values
- There is difference between instance and local variables the instance variables are declared inside the class but the local variables Is declared inside the methods and the instance variables if you use it before give it initialization the complier will do nothing cause it is already having default values rather than the local ones you can not use them before giving them a value in this case the complier will throw to you an error
- To compare 2 primitive things you can easily use == operator or to see if 2 references refer to the same object but use the equal () method to see if 2 different objects are equal.

- This is the summary of this chapter important notes :
  - Encapsulation gives you control over who changes the data in your class and how.
  - Make an instance variable private so it can't be changed by accessing the variable directly.
  - Create a public mutator method, e.g., a setter, to control how other code interacts with your data. For example, you can add validation code inside a setter to make sure the value isn't changed to something invalid.
  - Instance variables are assigned values by default, even if you don't set them explicitly.
  - Local variables, e.g., variables inside methods, are not assigned a value by default. You always need to initialize them.
  - Use == to check if two primitives are the same value.
  - Use == to check if two references are the same, i.e., two object variables are actually the same object.
  - Use .equals() to see if two objects are equivalent (but not necessarily the same object), e.g., to check if two String values contain the same characters.