1. Object-oriented programming?

* Inheritance
  + It is the mechanism in java by which one class is allowed to inherit the features (fields and methods ) of another class.
  + SuperClass, SubClass – can add their own fields and methods in addition to superclass fields and methods. and Reusability.
* Encapsulation
* Polymorphism
  + Polymorphism is considered an important feature of OOPS
  + Allows to define one interface and have multiple implementations
  + There are 2 types
    - Compile Time polymorphism
    - Run time polymorphism
  + Method Overloading - compile-time polymorphism:
    - Multiple functions with the same name but different parameters
    - Function can be overloaded in a number of arguments or change in the type of arguments.
  + Method Overriding – run time
    - Known as dynamic method dispatch.
    - Derived class a definition for one of the member functions in the base class.
    - Base function is said to be overridden.
    - Only the methods can be overridden and not the variables.
* Data abstraction

– defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details.

It also helps in classifying or grouping the objects.

* Inheritance and polymorphism are used together in java to achieve fast performance and readability of code.

1. Why is finally bloc k used for ?

* Finally is a keyword.
* It is intended to guarantee the execution of code .
* Get executed regardless of whether exception is thrown or caught.
* Can be used without catch block
* To execute cleanup code like closing connections, closing files or freeing up threads.
* As alternate we can use try-with-resources instead of finally block
* Finally executed if there was no exception /exception
* Try🡪 finally🡪 and throws exception in try block
* Try🡪 catch🡪 finally
* Try🡪 finally🡪 then return in try will be ignored
* Try🡪 finally🡪 then return in catch is ignored
* In case of System.exit(1)🡪 finally block is aborted.
* In case of Runtime.getRuntime().halt(1) 🡪 finally block is aborted
* Return statement in finally block ignores an uncaught exception.

<https://medium.com/@ashaynayak50/and-equals-in-java-interview-question-8f5ffd88d811>

What is the difference between **==** and **.equals()**?

It’s simple. One is an operator and the other one is a method. What’s the big deal?

Just FYI, the .equals() method is present in Object class which is by default a parent of every class you make or use.

Whenever you search its difference on the internet, you will generally find that == operator compares reference i.e. address whereas .equals() method compares content i.e. value. It is wrong. If you don’t believe then copy the below code and run it.

class Person{  
   
 String name;  
   
 Person(String name){  
 this.name = name;  
 }  
   
}  
class Main  
{  
 public static void main (String[] args) throws java.lang.Exception  
 {  
 Person p1 = new Person("ashay");  
 Person p2 = new Person("ashay");  
   
 System.out.println(p1==p2);  
 System.out.println(p1.equals(p2));  
 }  
}Output:  
false  
false

As per the difference mentioned above, the first print statement should print **false** (which it is doing) because it compares the address and here we have created two objects so two different addresses, hence **false**.

But what about the second print statement? As the .equals() method compares the value then it should print **true** because both objects have the same value “ashay”.

Oh no! What’s happening then? Wait, I will explain but before that please see the below code and its output.

class Main  
{  
 public static void main (String[] args) throws java.lang.Exception  
 {  
   
 String s1 = new String("ashay");  
 String s2 = new String("ashay");  
   
 System.out.println(s1==s2);  
 System.out.println(s1.equals(s2));  
 }  
}Output:  
false  
true

Here, the output seems justifiable as it is in sync with the discussion we did i.e. == compares address and .equals() compares value.

Lemme explain to you.

First of all, the == operator and .equals() method both work in the same way i.e. both compare the address only. So output we got from the first code snippet is correct.

In case of the second code snippet, we are using String object and String class has overridden the .equals() method and they override in a way that it should compare value. Originally, .equals() is present in Object class and String class by default extend Object class so String class can override the .equals() method of an object class. In the first code snippet, we used Person class and we didn’t override the .equals() method so it used the default .equals() method of Object class, and thus you got **false** as an output despite having the same value.

During Interviews, if they ask you the difference then please explain to them what I have explained. Please don’t directly state that == compares address and .equals() method compares value as it is not true in every case.

They can also give you a code snippet and ask for the output then please remember what I have explained.

I hope it is clear now.

1. Overriding equals only

* Casues the two equal instance to have unequal hashcodes.
* Emp1 will be placed in one bucket and emp2 will be palced in once bucket

Overiding only hashcode

* Thy will has in the same bucket as thy prduce same has code .
* If quals not overridden. When emp2 is added in set and iternate it
* Emp2.equals(emp1) will be false.

<https://medium.com/@pulapaphani/why-overriding-equals-and-hashcode-in-java-b5f869f985d2>

Only Override HashCode, **Use the default Equals**: Only the references to the same object will return true.

In other words, those objects you expected to be equal will not be equal by calling the equals method.

Only Override Equals, Use the default HashCode: There might be duplicates in the HashMap or HashSet