There are three main features of OOPS.

**1)** Encapsulation **-Java beans with private fields and public methods**

**2)** Inheritance

**3)** Polymorphism

4) ABSTRACTION

5) Association

6) Aggregation

7) Composition

**Encapsulation:**

Encapsulation means putting together all the variables (instance variables) and the methods into a single unit called Class. It also means hiding data and methods within an Object. Encapsulation provides the security that keeps data and methods safe from in.

Class is an example of encapsulation.

**Inheritance**

the ability to create classes that share the attributes and methods of existing classes, but with more specific features. Inheritance is mainly used for code reusability. So you are making use of already written class and further extending on that. That why we discussed about the code reusability the concept. In general one line definition we can tell that deriving a new class from existing class.

Extends is the key word which is used to accomplish this in the java.

**Polymorphism**

Method overloading : (static polymorphism –compile time)

**Method overloading** is defining several methods in the same class, that accept different numbers and types of parameters. In this case, the actual method called is decided at compile-time, based on the number and types of arguments. For instance, the method System.out.println() is overloaded, so that you can pass ints as well as Strings, and it will call a different version of the method.

Method overriding : (dynamic polymorphism –run time)

For example, the standard Java class java.util.LinkedHashSet extends java.util.HashSet. The method add() is overridden in LinkedHashSet. If you have a variable that is of type HashSet, and you call its add() method, it will call the appropriate implementation of add(), based on whether it is a HashSet or a LinkedHashSet. This is called polymorphism.

**ABASTRACTION – abstract classes and interfaces**

**Abstraction vs Encapsulation – Java OOPS**

Abstraction and Encapsulation in Java are two important [Object oriented programming concept](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html) and they are completely different to each other. Only similarity between Abstraction and Encapsulation is that they are OOPS concept, other than that they mean two different things.

**Abstraction represent taking out the behavior from How exactly its implemented, one example of** [**abstraction in Java**](http://javarevisited.blogspot.sg/2010/10/abstraction-in-java.html)**is interface**

while **Encapsulation** means hiding details of implementation from outside world so that when things change no body gets affected. One example of [Encapsulation in Java](http://javarevisited.blogspot.sg/2012/03/what-is-encapsulation-in-java-and-oops.html) is private methods; clients don't care about it, You can change, amend or even remove that method  if that method is not encapsulated and it were public all your clients would have been affected. Apart from this main difference in behavior here are couple of more *differences between Abstraction and Encapsulation in Java*.

1. First difference between Abstraction and Encapsulation is that, Abstraction is implemented in Java using [interface](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html) and abstract class while Encapsulation is implemented using [private](http://javarevisited.blogspot.sg/2012/03/private-in-java-why-should-you-always.html), package-private and protected access modifier.
2. Encapsulation is also called data hiding.

**Abstraction** is using extension points to let the choice be deferred to a different part of which exact code is run. That choice could be made elsewhere in your program, in another program, or dynamically at runtime.

There are also strong benefits to abstraction:

* When you change your code that implements an abstraction, the user of the abstraction doesn't have to change their code. As long as the abstraction doesn't change, the users won't have to change their code.
* When you write code that uses an abstraction, you can write code once that will be reusable against any new code that implements that abstraction. You can write less code to do more.

Rules of Method Overloading and Overriding in Java

<http://java67.blogspot.com/2012/09/what-is-rules-of-overloading-and-overriding-in-java.html>

Rule of overloading a method in Java

Here is the list of rule which needs to be followed to overload a method in Java :

1) First and foremost rule to overload a method in Java is to change [method signature](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html). method signature is made of number of arguments, type of arguments and order of arguments if they are of different types.  
  
2) return type of method is not part of method signature, so just changing the return type will not overload method in Java.  see [What is method overloading in Java](http://java67.blogspot.sg/2012/08/what-is-method-overloading-in-java-example.html) for code example of these rules,

**Overriding Rules in Java**  
Overriding is completely different than overloading and so it's rules are also different. For terminology, original method is known as overridden method and new method is known as overriding method. **Following rules must be followed to correctly override a method in Java :**  
1) A method can only be written in Sub class, not in same class.  
  
2) Overriding method cannot throw [checked Exception](http://javarevisited.blogspot.sg/2011/12/checked-vs-unchecked-exception-in-java.html) which is higher in hierarchy, than checked Exception thrown by overridden method. For example if overridden method throws IOException or [ClassNotfoundException](http://javarevisited.blogspot.sg/2011/08/classnotfoundexception-in-java-example.html), which are checked Exception, than overriding method can not throw java.lang.Exception because it comes higher in type hierarchy (it's super class of IOExceptionand ClassNotFoundExcepiton).  
  
3) overriding method can not reduce access of overridden method. It means if overridden method is defined as public than overriding method can not be protected or package private.  
  
4) overriding method can increase access of overridden method. this is opposite of earlier rule, according to this if overridden method is declared as protected than overriding method can be protected or public.  
  
5) [private](http://java67.blogspot.sg/2012/08/can-we-override-private-method-in-java.html), [static](http://java67.blogspot.sg/2012/08/can-we-override-static-method-in-java.html) and [final method](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) can  not be overridden in Java.

That’s all on **Rules of method overloading and overriding in Java**, Its very important to remember these rule to correctly overload and override any method in Java. Also remember to use @Override annotation to accidentally overloading a method, instead of overriding it.

### Que) Why multiple inheritance is not supported in java?

* To reduce the complexity and simplify the language, multiple inheritance is not supported in java. For example:

1. **class** A{
2. **void** msg(){System.out.println("Hello");}
3. }
5. **class** B{
6. **void** msg(){System.out.println("Welcome");}
7. }
9. **class** C **extends** A,B{//suppose if it were
11. Public Static **void** main(String args[]){
12. C obj=**new** C();
13. obj.msg();//Now which msg() method would be invoked?
14. }
15. }

### Composition

[Composition](https://www.infoworld.com/article/3029325/application-development/exploring-association-aggregation-and-composition-in-oop.html)is a stricter form of aggregation. It occurs when the two classes you associate are mutually dependent on each other and can’t exist without each other. For example, take a Car and an Engine class. A Car cannot run without an Engine, while an Engine also can’t function without being built into a Car. This kind of relationship between objects is also called a PART-OF relationship.

Composition in Java:

* A restricted form of aggregation
* Represents a PART-OF relationship between two classes
* Both classes are dependent on each other
* If one class ceases to exist, the other can’t survive alone

### Aggregation

[Aggregation](https://beginnersbook.com/2013/05/aggregation/) is a narrower kind of association. It occurs when there’s a one-way (HAS-A) relationship between the two classes you associate through their objects. For example, every Passenger has a Car but a Car doesn’t necessarily have a Passenger. When you declare the Passenger class, you can create a field of the Car type that shows which car the passenger belongs to. Then, when you instantiate a new Passenger object, you can access the data stored in the related Car as well.

**Aggregation in Java:**

* One-directional association.
* Represents a HAS-A relationship between two classes.
* Only one class is dependent on the other.