OOPS concepts:

Example : Mobile - Samsung

* Nokia
* iPhone

Object:

Any real world entity which can have some characteristics or which can perform some work is called as Object. This object is also called as an instance i.e. - a copy of entity in programming language.

Class:

A Class is a plan which describes the object. We call it as a blue print of how the object should be represented. Mainly a class would consist of a name, attributes & operations. Considering the above example, A Mobile can be a class which has some attributes like Profile Type, IMEI Number, Processor, and some more.) & operations like Dial, Receive & SendMessage.

OOPS Concepts are mainly 4 :

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

Abstraction:

* Hidding unnecessary data from the users details,is called abstraction.
* It is process of **hiding the implementation details** and showing only the functionality.
* Abstraction in java is achieved by using interface and abstract class. Interface give 100% abstraction and abstract class give 0-100% abstraction.

Real time example :

1. Dialing a number call some method internally which concatenate the numbers and displays it on screen but what is it doing we don’t know.
2. Clicking on green button actual send signals to calling person’s mobile but we are unaware of how it is doing.

**When do you use abstraction?**  
When you know something needs to be there but not sure how exactly it should look like.

**Summary:**  
-    Use abstraction if you know something needs to be in class but implementation of that varies.  
-     In Java you **cannot create instance of abstract class**, its compiler error.  
-    abstract is a keyword in java.  
-    A class automatically becomes abstract class when any of its method declared as abstract.  
-     abstract method doesn't have method body.  
-    **Variable cannot be made abstract**, its only behavior or methods which would be abstract.  
-    If a class extends an abstract class or interface it has to provide implementation to all its abstract method to be a concrete class. Alternatively this class can also be abstract.

Encapsulation:

* Encapsulation is process of **wrapping code** and data together into a single unit.
* We can create fully encapsulated class by making the entire **data member will be private** and create getter, setter method to access that data member.
* POJO class is the best example of fully encapsulated class.
* Encapsulation is also known as "**Data hiding**" because they are protecting data which is prone to change.
* Both Abstraction & Encapsulation works hand in hand because Abstraction says what details to be made visible & Encapsulation provides the level of access right to that visible details.

Polymorphism :

Single Form behaving differently in different Situations.  
Example:-  
Person  
Person in Home act is husband/son,  
in Office acts Employer.  
in Public Good Cityzen.

Java has excellent support of polymorphism in terms of

1. **Method Overloading** (Static Binding or compile time polymorphism)
2. **Method Overriding** (Dynamic Binding or Runtime Polymorphism)

Method overloading:

* If a class have multiple methods by same name but different parameter is known as **Method Overloading**.

**Advantage of Method Overloading**

Method overloading increases the readability of program.

Method overriding:

* Having the same method in the subclass as declare in parent class is known as **method overriding**.
* object type (NOT reference variable type) determines which overriden method will be used at runtime
* abstract methods MUST be overridden
* final methods CANNOT be overridden
* static methods CANNOT be overridden
* constructors CANNOT be overridden

**Advantages of Method Overriding:**

* Time to invest method signature is reduced
* Different functionality in both super class and sub class by sharing same signature
* The functionality can be enhanced
* The behavior can be replaced in the sub class

The SOLID principles:

are the most talked about in the Enterprise Java world. These principles deal with cohesion, coupling, inheritance, abstract types in the Object Oriented Analysis and Design (OOAD)

The five principles are also asked about in Java or JEE interview. SOLID is an acronym for 5 principles of the Java language which are listed here:

1) S – Single Responsibility Principle: 2) O – Open Closed Principle:

3) L – Liskov Substitution Principle: 4) I – Interface Segregation Principle:

5) D – Dependency Inversion Principle:

Inheritence

**class** SuperClass {

**public** **void** printName(){sysout("Super");}

}

**class** SubClass **extends** SuperClass {

**public** **void** printName(){sysout("Sub");}

}

**class** MainClass {

**public** **static** **void** main(String[] args) {

SuperClass A = **new** SuperClass();

SubClass B = **new** SubClass();

SuperClass C = **new** SubClass();

//SubClass D = (SubClass) **new** SuperClass();

A.printName(); 🡪 Super

B.printName(); 🡪 Sub

C.printName(); 🡪 Sub

}

}

* Upcasting is possible - SuperClass C = **new** SubClass();

C is a reference of SuperClass type and refers to an object of SubClass. In Java, functions are virtual by default. So the run time polymorphism happens and derived fun() is called.

* Downcasting is not possible – prompts to cast to sub class - SubClass D = (SubClass) **new** SuperClass(); in runtime java.lang.ClassCastException occurs.
* Final methods cannot be inherited. - throws a compile time error.

**class** SuperClass {

**final** **public** **void** printName(){sysout("Super");}

}

* Static methods also cannot be inherited and runtime polymorphism doesn’t happen

**class** SuperClass/SubClass {

**static** **public** **void** printName(){sysout("Super");}

}

A.printName(); 🡪 Super

B.printName(); 🡪 Sub

C.printName(); 🡪 Super

* Private methods are final.
* Protected members are accessible within a package and inherited classes outside the package.
* We cannot override private methods.
* In Java all classes inherit from the Object class directly or indirectly. The Object class is root of all classes.
* Multiple inheritance is not allowed in Java.
* Unlike C++, there is nothing like type of inheritance in Java where we can specify whether the inheritance is protected, public or private.
* o starts referring to the same object as referred by y

**public** **static** **void** DoPrint(SuperClass o) {

o.printName();

}

*DoPrint*(A); 🡪 Super

*DoPrint*(B); 🡪 Sub

*DoPrint*(C); 🡪 Sub

* public methods cannot be overridden to a private method in Subclass – throws a compile time error

It is compiler error to give more restrictive access to a derived class function which overrides a base class function.

**class** SubClass {

**private** **void** printName(){sysout("Super");}

}

* super.super.printName(); - throws compile time error

SuperiorClass 🡪 SuperClass 🡪 SubClass

**class** SubClass {

**private** **void** printName(){super.printName(); sysout("Super");}

}

B.printName(); 🡪 Super Sub

* Variables invoked are referenced from the reference type and not object type

SuperClass 🡪 int i = 10;

SubClass 🡪 int i = 20;

System.***out***.println(A.i); 🡪 10

System.***out***.println(B.i); 🡪 20

System.***out***.println(C.i); 🡪 10

* Static blocks/constructors/initialization block will get executed from parent to child
* If all three are present – static block of parent and child class gets executed, followed by initialization block & constructor of parent and child