OOPs (Object-Oriented Programming System)

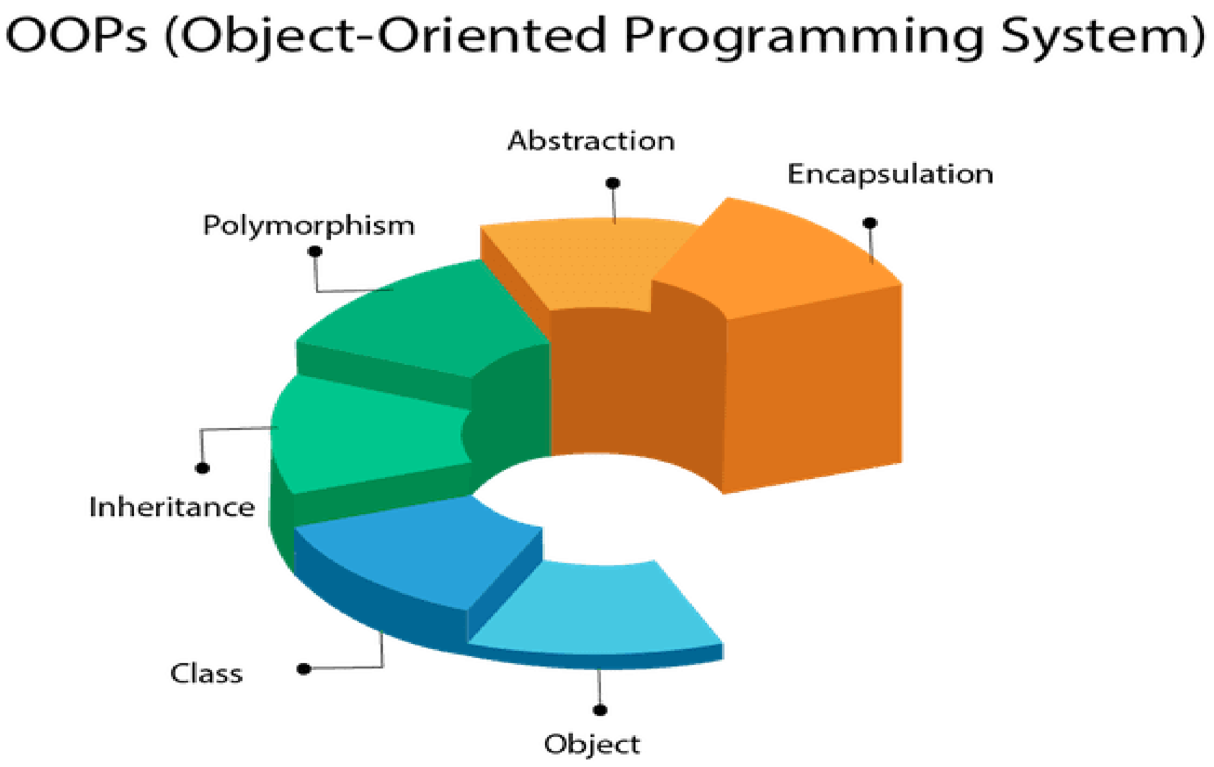
===

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology to design a program using classes and objects.

Pillars/feature of OOPs:-

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

===



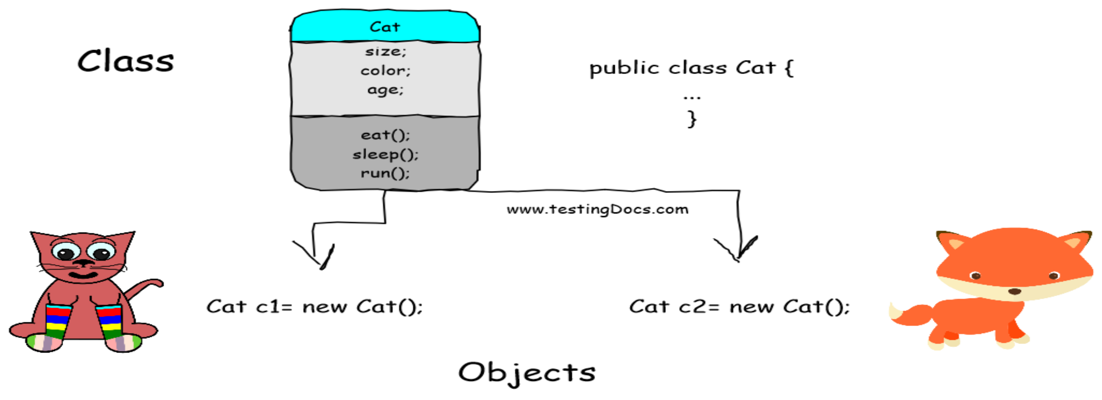
Object: -

Any entity which have state and behaviour is known as object. For example a car, book, clock, laptop, bike, etc.

- ❖ **State**: represents the data (value) of an object.
- ❖ **Behaviour**: represents the behaviour (functionality) of an object such as deposit, withdrawal, etc.

- ➢ An Object can be defined as an instance of a class, and there can be multiple instances of a class in a program.
- ➢ It can be physical or logical entity.
- ➢ An object contains an address and takes up some space in memory.

Example: A cat is an object as it has states like color, name etc. as well as behaviors like running, eating etc.



```
public class Cat {
...
}
```

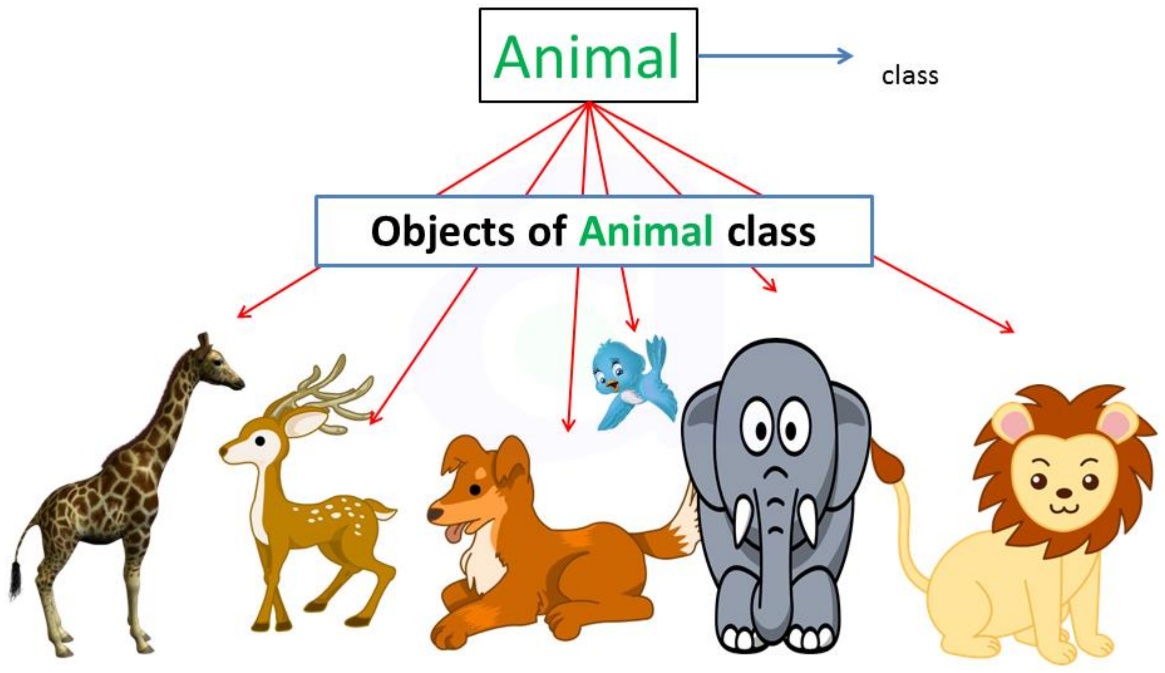
```
Cat c1= new Cat();
```

```
Cat c2= new Cat();
```

Class: -

- Class is collection of objects, methods, constructors, variables, statements, datatypes etc.
- Class is a blueprint or a set of instructions to build a specific type of object.
- Class in Java determines how an object will behave and what the object will contain.
- It is a logical entity, which doesn't consume any space.
- A class can also be defined as a blueprint or protocol from which objects are created.

Class and Object Relation -



==

Inheritance

==

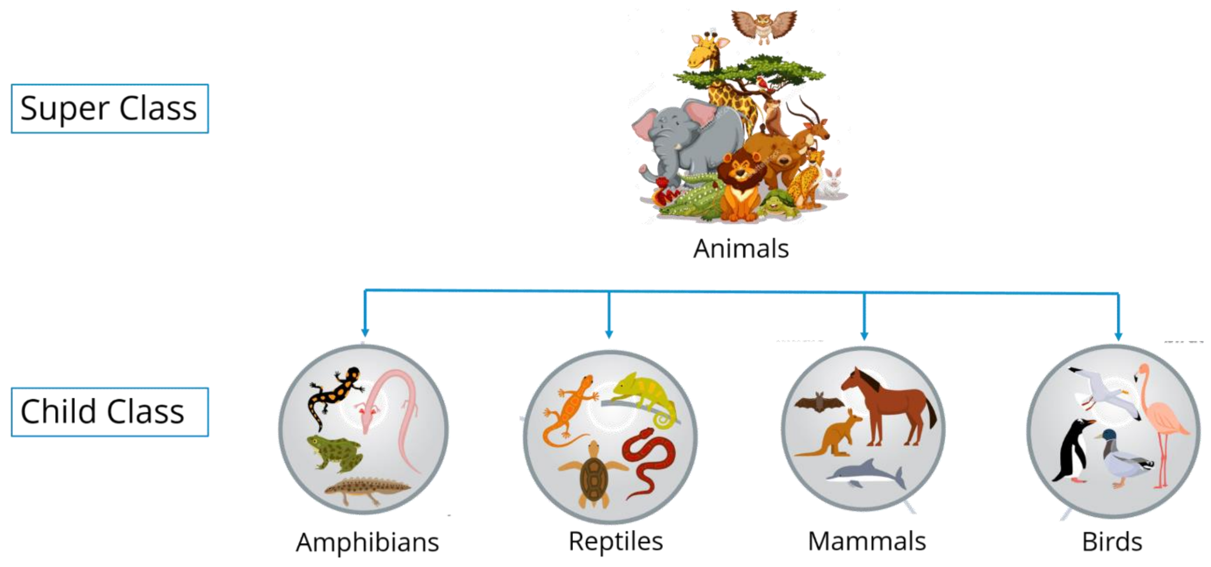
1. Inheritance in Java is a mechanism in which one class acquires all the properties and behaviours of another class with the help of extends keyword.
2. Inheritance is one of the most important Object-oriented programming language system.
3. Also, we can say that subclass can acquire the properties of superclass with the help of extends keyword
4. Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Why use inheritance in java

1. For Code Reusability.
2. For code optimization

Terms used in Inheritance

- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a *derived class, extended class, or child class*.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called *a base class or a parent class.*



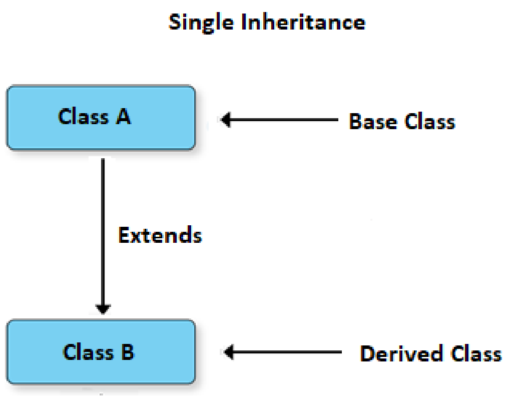
Types of Inheritance

1. Single Level Inheritance
2. Multi-Level Inheritance
3. Multiple Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance

==

Single Level Inheritance

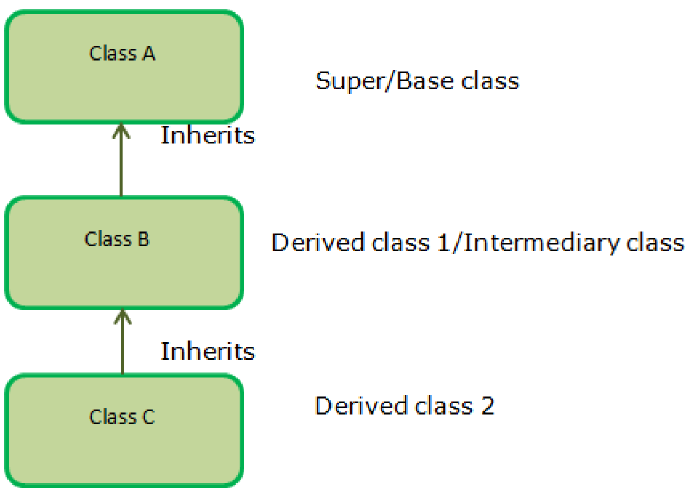
- The process of inheritance in which a class acquires/inherits the properties of another class by using extends keyword is called as Single Level Inheritance.
- It is an operation where inheritance takes place between two classes only to perform Single Level Inheritance



Eg. ClassB inherits classA.

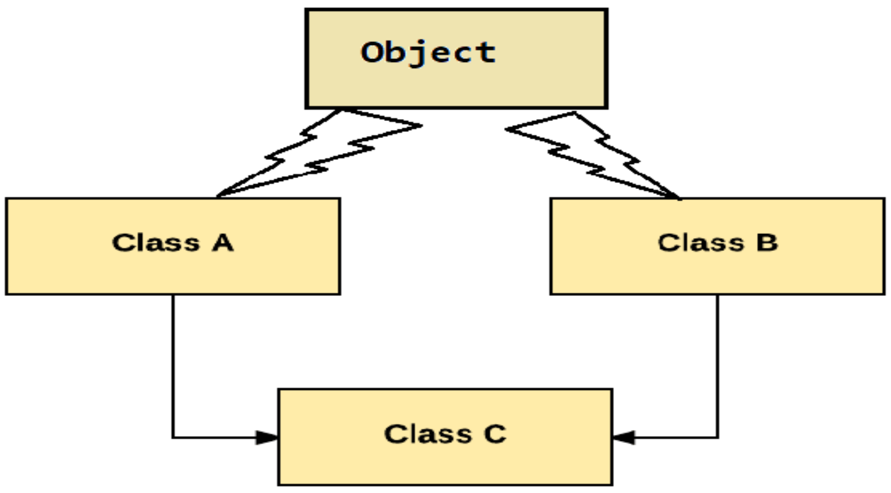
Multi-Level Inheritance

- Multilevel inheritance takes place between three or more classes.
- The process of inheritance in which one subclass acquires the properties of another superclass with the help of extends keyword and this phenomenon continues so that we called as "***Multi level Inheritance***"
- In short when there is a chain of inheritance it is also known as multilevel inheritance.



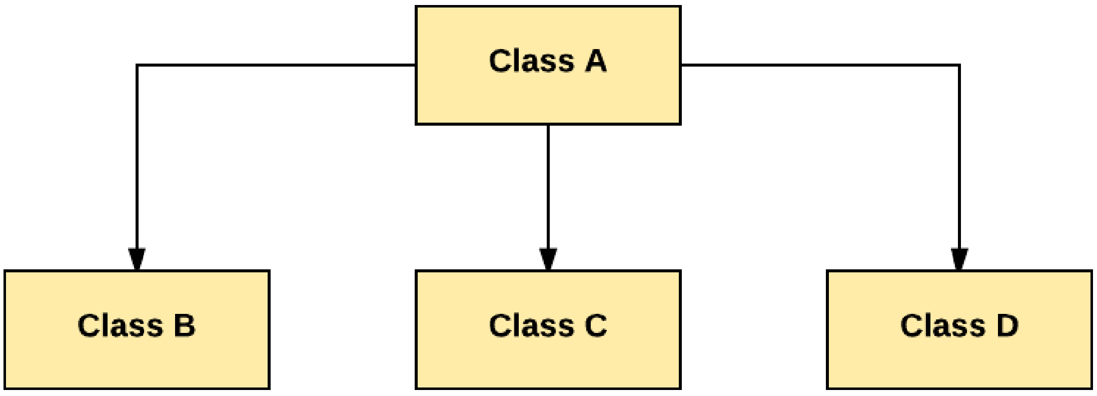
Multiple Inheritance

- The Process of inheritance in which one subclass acquires the properties of two or more super-classes at a same time with the help of extends keyword is called as Multiple Inheritance.
- Java doesn't support multiple inheritance because it results diamond ambiguity problem.
- It means when we try to inherits properties from more than two classes at the same time, the diamond like structure gets created in JVM and object variable gets confused for who need to inherited the property of superclass into subclass.
- Super Class of all the classes is object class so their diamond ambiguity forms
- To reduce the complexity of language multiple inheritance not supported by java, but we can achieve with the help of interface.



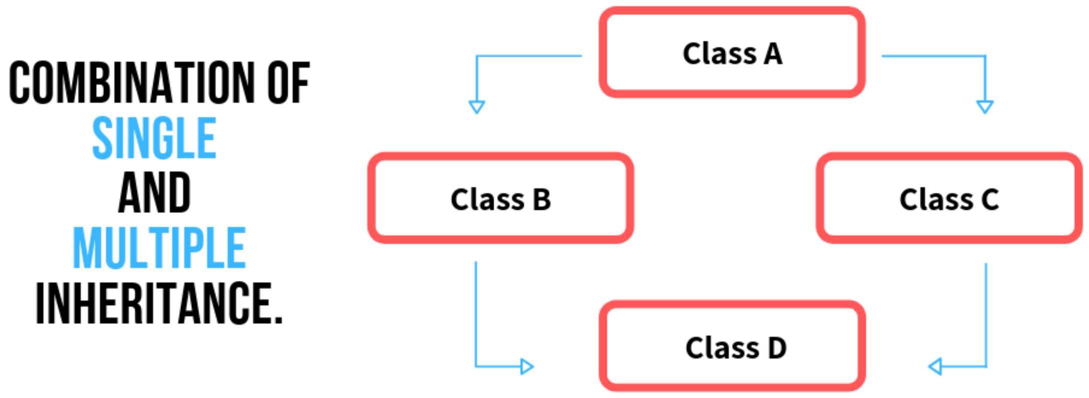
Hierarchical Inheritance

- The process of inheritance in which multiple subclasses acquires the properties one superclass with help of extends keyword is called as Hierarchical Inheritance.
- Hierarchical Inheritance takes place between one superclass and multiple subclasses.
- The two or more classes inherits a single class it is known as Hierarchical Inheritance

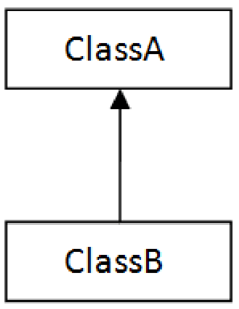


Hybrid Inheritance

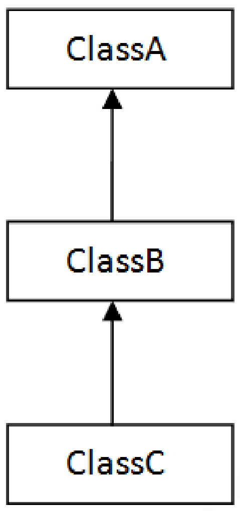
- Hybrid Inheritance is a combination of Inheritances. Since in Java Multiple Inheritance is not supported directly so hybrid inheritance is also not support but we can achieve Hybrid inheritance through Interfaces.



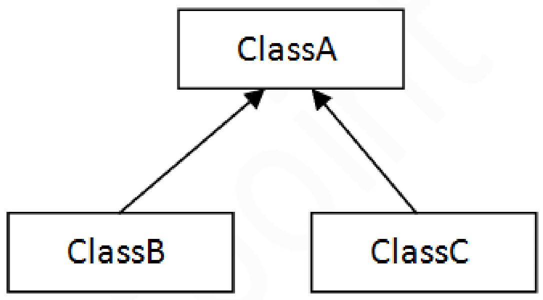
==



1) Single

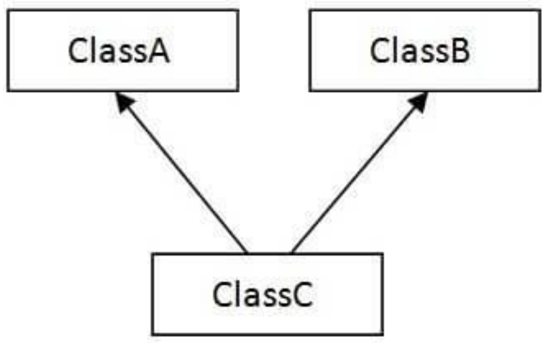


2) Multilevel

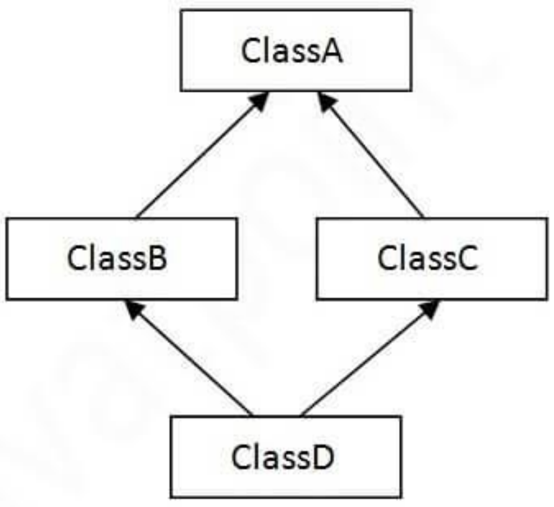


3) Hierarchical

Note: Multiple inheritance and Hybrid inheritance is not supported in Java through class.



4) Multiple



5) Hybrid

What is the difference between an object-oriented programming language and object-based programming language?

Object-based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object-based programming languages.

Who is the superclass of all the classes?

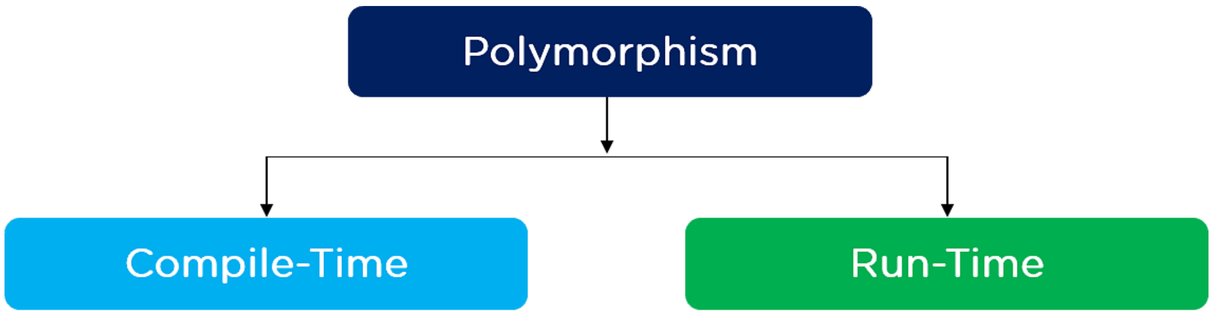
==

Polymorphism

==

- Polymorphism means “Many Forms”, is a concept by which we can perform a single action in different way.
- It occurs when we have many classes that are related to each other by inheritance.
- Polymorphism is derived form 2 Greek works: Ploy and morphs which means
  - “Ploy” means many
  - “morphs” mean forms
- One object shows different behaviour at different stages of life cycle as called as “Polymorphism”
- Eg. A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So, the same person possesses different behaviour in different situations. This is called polymorphism.

There are two types of polymorphism in JAVA



Compile Time Polymorphism

- The process of polymorphism in which method declaration is going to get bonded with its own definition during compile time based on arguments is called as *"Compile time polymorphism"*
- At compile-time, Java knows which method to call by checking method signature.
- Also called as *Static binding or Early binding*.
- As binding takes place during compilation time so it is known as "Early binding".
- Method overloading is an example of compile time polymorphism.

Run time polymorphism

- The process of polymorphism in which method declaration is going to get bonded with its own definition during run time or execution time based on object creation is called as *"runtime polymorphism"*.
- Also called as *Dynamic biding or Late binding.*
- As binding takes place during execution time so it is known as "Late binding"
- Method overriding is an example of run-time polymorphism.

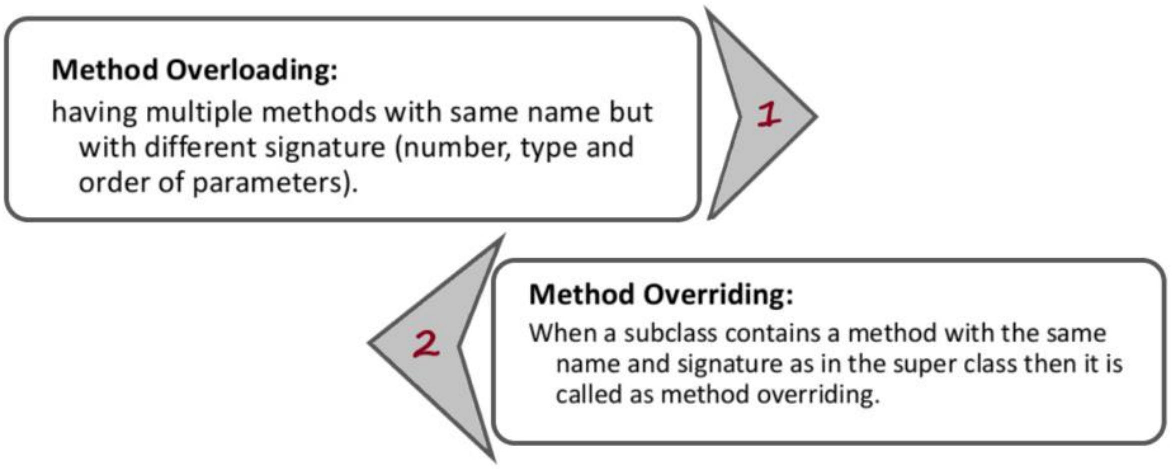
==

Method Overloading

- In a class we have multiple methods having same name but different in parameters, it is known as **Method Overloading**.
- If we have to perform only one operation, having same name of the methods increases the readability of the program.
- There are two ways to overload the method in java
  - By changing number of arguments
  - By changing the data type

Method Overriding

- If subclass (child class) have the same method as declared in the parent class, it is known as **method overriding**.
- If we have different classes, same method name, same argument having inheritance relationship between those classes.


Method Overloading:
having multiple methods with same name but with different signature (number, type and order of parameters).
1
2
Method Overriding:
When a subclass contains a method with the same name and signature as in the super class then it is called as method overriding.

- Eg. Suppose we have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs.

Difference between method overloading and method overriding

No.	Method Overloading	Method Overriding
1	It is used to increase the readability of the program.	It is used to provide the specific implementation of the method that is already provided by its super class.
2	It occurred within class.	It occurred in two classes that have IS-A (inheritance) relationship.
3	Parameter must be different.	Parameter must be same.
4	It is the example of compile time polymorphism.	It is the example of run time polymorphism.
5	In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter.	Return type must be same or covariant in method overriding.
6	Private and final methods can be overloaded.	Private and final methods can't be overridden.

Method Overloading

- **Compiler**
- ***Same name***
- **Same Class**
- **Different arguments**
  - **Number of Arguments**
  - **Sequence of Arguments**
  - **Types of Arguments**

Method Overriding

- ***JVM***
- ***Inheritance (IS-A Relationship)***
- ***Same name***
- ***Different Class***
- ***Same arguments***
  - ***Number of arguments***
  - ***Sequence of Arguments***
  - ***Types of Arguments***

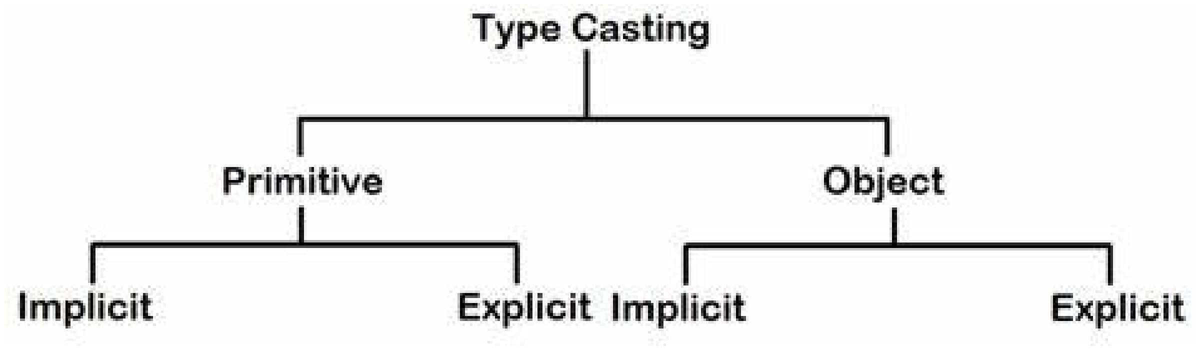
==

Casting / Conversions

==

- Converting one type of information into another type of information is called as casting.
- It is also called as promotion, conversion or type casting.

There are two types of casting



- ❖ Primitive Casting
- ❖ Object (non-primitive) Casting

Primitive Casting

- The process of converting one datatype information into another datatype information is known as Primitive Casting.

There are three subtypes of primitive casting

1. Implicit Casting/Widening Conversion
2. Explicit casting/ Narrowing Conversion
3. Boolean Casting

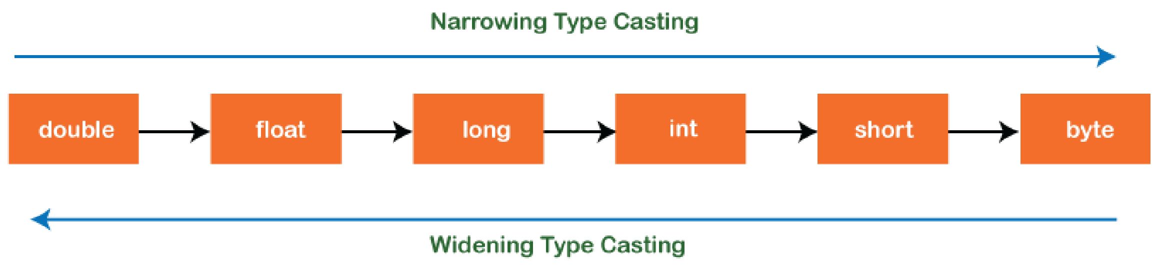
1. implicit Casting/Widening Conversion

- Converting lower datatype information into higher datatype information is known as Implicit Casting.
- It is also called as widening casting or automatic casting.
- It is done automatically. It is safe because there is no chance to lose data.
- It takes place when:
  - Both data types must be compatible with each other.

    - The target type must be larger than the source type.
- Implicit Casting format –
    - **byte** -> **short** -> **char** -> **int** -> **long** -> **float** -> **double**
- That's why we called as widening casting where memory size goes on increasing

2. Explicit casting/ manual casting / promotion

- Converting higher data type information into a lower datatype information is known Explicit casting.
- It is also called as narrowing casting or manual casting.
- It is done manually by the programmer, and data loss may be takes place
- If we do not perform casting then the compiler reports a compile-time error.
- Here largest type specifies the desired type to convert the specified value to .
    - **double** -> **float** -> **long** -> **int** -> **char** -> **short** -> **byte**
- Note -
    - Narrowing casting must be done manually by placing the type in () in front of the value



Type Casting in Java

The automatic conversion is done by the compiler and manual conversion performed by the programmer.

3. Boolean casting

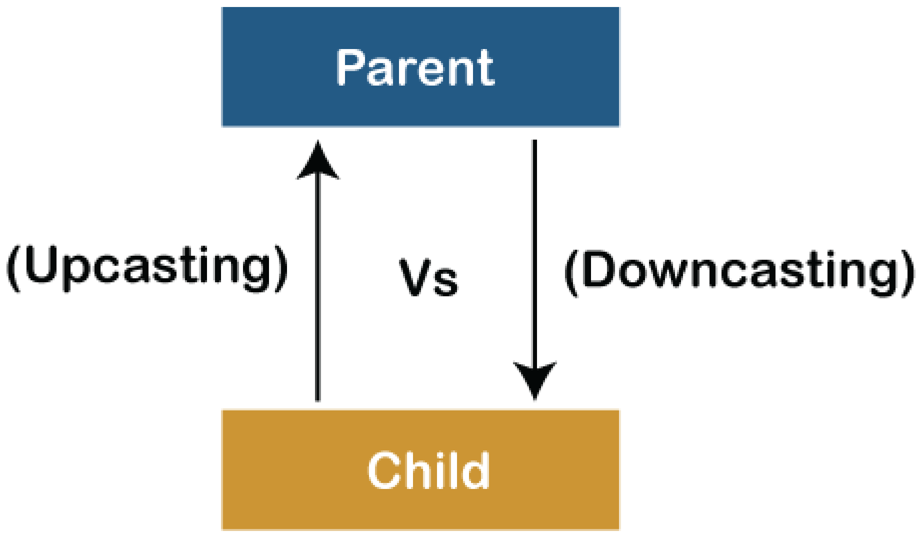
- It considers to be an incompatible casting type
- Boolean returns true or false so for converting true into false and false into true it is not possible so Java doesn't support boolean casting.
- it is considered to be as incompatible casting means we cannot convert true into false it is not possible in java.

Non-Primitive Casting / Object Typecasting

- ➢ The process of converting one type of class into another type of class is known as non-primitive casting.

There are two subtypes of non-primitive casting

1. Up casting
2. Down Casting



1. Up casting

- ➢ The process of assigning subclass property into superclass is known as Upcasting
- ➢ Upcasting is the typecasting **of a child object to a parent object**.
- ➢ Before performing upcasting it is mandatory to have inheritance relationship between classes. After performing inheritance properties which are present in super class that comes into subclass
- ➢ Upcasting can be done implicitly.
- ➢ Upcasting gives us the flexibility to access the parent class members but it is not possible to access all the child class members using this feature.

```
Parent p = new child();
```

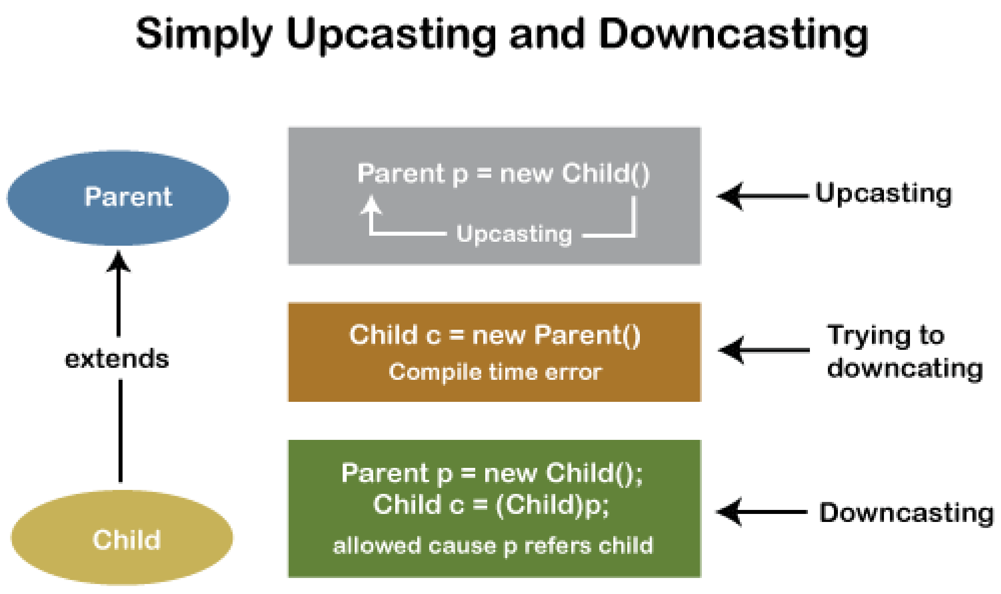
2. Down Casting

- Downcasting is the typecasting of a **parent object to a child object**.
- Downcasting cannot be implicit.
- before perform down casting perform upcasting operation first then perform down casting
- So, we have to cast child to parent forcefully which is known as down casting

Child c= (parent)p;

- So, Java doesn't support down casting/sampling
- Down casting has to be done externally and due to down casting a child object can acquire the properties of parent object

==



Example

Let us there is a parent class. There can be many children of a parent. Let's take one of the children into consideration. The child inherits the properties of the parent. Therefore, there is an "is-a" relationship between the child and parent.

Therefore, the child can be implicitly **upcasted** to the parent. However, a parent may or may not inherits the child's properties. However, we can forcefully cast a parent to a child which is known as **downcasting**.

After we define this type of casting explicitly, the compiler checks in the background if this type of casting is possible or not. If it’s not possible, the compiler throws a ClassCastException.

===

Access Modifiers in Java

==

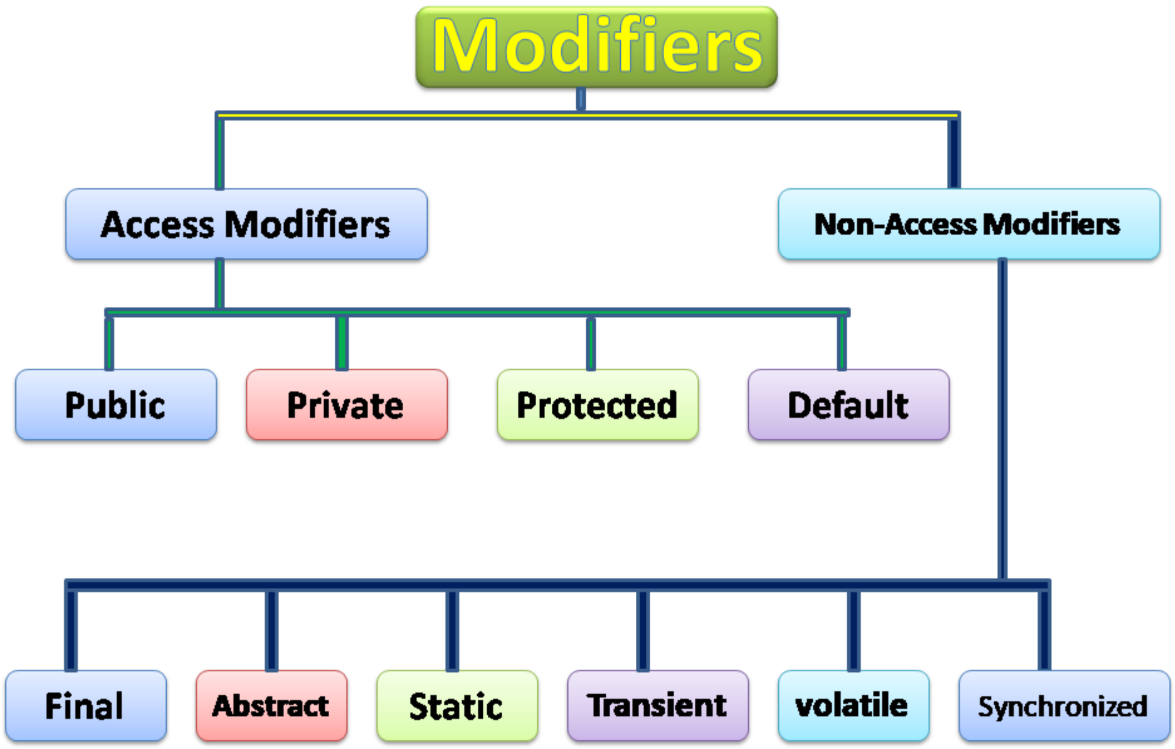
Modifiers

- ➢ It is used to set the access scope/level for classes, variables, methods and constructors.

There are two types of modifiers in Java:

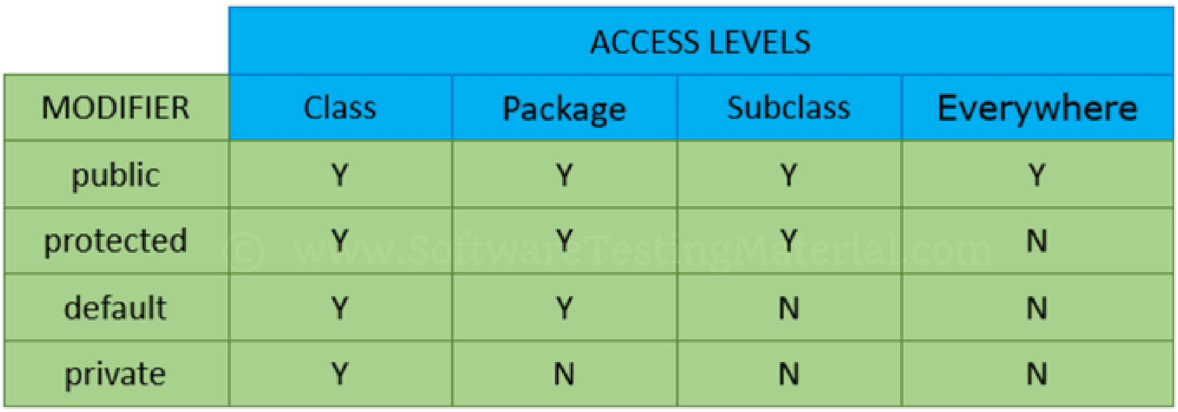
I. **Access Modifiers** - controls the access level

II. **Non-Access Modifiers** - do not control access level, but provides other functionality.



Access Modifiers

- ➢ The access modifiers in Java defines the accessibility or scope of a field such as method, constructor, variable or class.
- ➢ We can change the access level of fields, such as constructors, methods, and class by applying the access modifier on it.



	ACCESS LEVELS			
MODIFIER	Class	Package	Subclass	Everywhere
public	Y	Y	Y	Y
protected	Y	Y	Y	N
default	Y	Y	N	N
private	Y	N	N	N

==

1. **Private**
   - The access level of a private modifier is only within the class.
   - It cannot be accessed from outside the class.
   - Private Constructor - If you make any class constructor private, you cannot create the instance of that class from outside the class
2. **Default**
   - If we don't declare any modifier, it is treated as **default** by default.
   - The access level of a default modifier is only within the package. It cannot be accessed from outside the package.
   - It provides more accessibility than private. But it is more restrictive than protected, and public.
3. **Protected**
   - The access level of a protected modifier is within the package.
   - But if we use inheritance and make child class outside the package, it can be accessed from outside.
   - The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.
   - It provides more accessibility than the default modifier.
4. **Public**
   - The access level of a public modifier is everywhere.

- It has the widest scope among all other modifiers as it can be accessed from within the class, outside the class, within the package and outside the package.

Declaration Of Access modifers with each field				
Access Modifiers	default	public	protected	private
Class	Yes	Yes	No	No
Method	Yes	Yes	Yes	Yes
Variable	Yes	Yes	Yes	Yes
Constrcutor	Yes	Yes	Yes	Yes

==