Inheritance

Parent – child

Child class acquires all the properties and behavior of parent class

1. Reusability
2. Runtime polymorphism (overriding )

Class sub/child extends super/parent{

}

------- > extends

Single -> one class inherits from another

Multilevel -> chain of inheritance

**public** **class** hello {

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

cat d = **new** cat();

d.a();

d.b();

d.c();

}

}

**class** animals{

**void** a() {

System.***out***.println("hello");

}

}

**class** dog **extends** animals{

**void** b() {

System.***out***.println("hello world");

}

}

**class** cat **extends** dog{

**void** c() {

System.***out***.println("hello world njkhj");

}

}

Hierarchical - > two or more classes depending on one single class

**public** **class** hello {

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

cat d = **new** cat();

dog e = **new** dog();

d.a();

e.b();

e.a();

d.c();

}

}

**class** animals{

**void** a() {

System.***out***.println("hello");

}

}

**class** dog **extends** animals{

**void** b() {

System.***out***.println("hello world");

}

}

**class** cat **extends** animals{

**void** c() {

System.***out***.println("hello world njkhj");

}

}

Multiple - > is not possible (reduces the time complexity )

Abstraction

Hiding the implementation and showing the functionality

Abstract / concrete

Ab – void a(int a, int b);

Co - void a(int a, int b){

//sum a+b

}

Class ---- > combination of only concrete methods

Abstract class -- > both combination of abstract and concrete method

**public** **class** hello **extends** animals {

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

animals a = **new** hello();

a.a();

a.sum(10,12);

}

**void** a() {

System.***out***.println(“hello”);

}

}

**abstract** **class** animals{

**abstract** **void** a();

**void** sum(**int** a, **int** b) {

System.***out***.println(a+b);

}

}

Interface -- > combination of only abstract methods

Achieving abstraction

-------> abstract class (0 --- 100)

----🡪 interface (100%)

Points to Remember

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have [constructors](https://www.javatpoint.com/java-constructor) and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.