

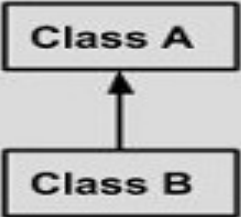
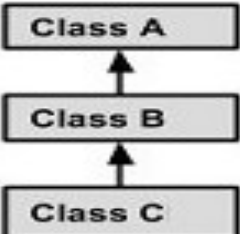
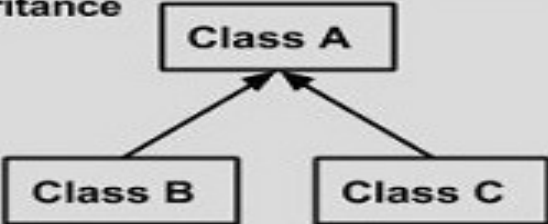
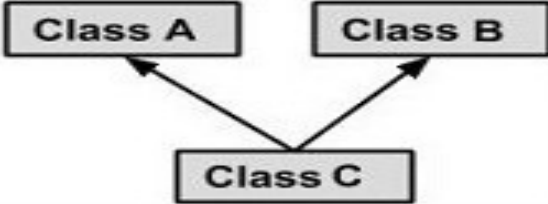
Inheritance



Inheritance

- The mechanism of deriving (acquire properties) of an existing class into a new class.
- Super class / base class/ parent class
- Sub class. / derived class / child class
- Java does not support multiple inheritance directly but used implements interface to provide support for it.

Java : Types of Inheritance

Single Inheritance	 <pre>graph BT; B[Class B] --> A[Class A]</pre>	<pre>public class A { } public class B extends A { }</pre>
Multi Level Inheritance	 <pre>graph BT; C[Class C] --> B[Class B]; B --> A[Class A]</pre>	<pre>public class A {} public class B extends A {.....} public class C extends B {.....}</pre>
Hierarchical Inheritance	 <pre>graph BT; B[Class B] --> A[Class A]; C[Class C] --> A</pre>	<pre>public class A {} public class B extends A {.....} public class C extends A {.....}</pre>
Multiple Inheritance	 <pre>graph BT; C[Class C] --> A[Class A]; C --> B[Class B]</pre>	<pre>public class A {} public class B {.....} public class C extends A,B { } // Java does not support mutiple Inheritance</pre>

Inheritance : Some Terminologies

Super Class: The class whose features are inherited is known as super class(or a base class or a parent class).

Sub Class: The class that inherits the other class is known as sub class(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.

Inheritance : Some Terminologies

Reusability: Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class..

```
class Animal
```

```
{  int noOfLegs; void eat()
```

```
{  System.out.println("eating...");
```

```
}
```

```
}
```

```
class Dog extends Animal
```

```
{  Dog(int no OfLegs)
```

```
{    this.noOfLegs= noOfLegs;
```

```
}
```

```
void bark()
```

```
{  System.out.println("barking..."+"With no Legs..."  
    "+noOfLegs);
```

```
}
```

```
}
```

```
class TestInheritance{  
    public static void main(String args[]){  
        Dog d=new Dog(4);  
        d.bark();  
        d.eat();  
    }  
}
```

Method Overriding

When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to **override** the method in the super-class.

Compare Method overloading and Overriding.

Static and Dynamic Binding

Binding

Association of object with the method is known as binding. There are two types of binding:

Static or Early Binding that happens at compile time and done by compiler.

Dynamic or Late Binding that happens at runtime.

Static and Dynamic Binding

```
void method( )  
{  
  
}
```

-

```
Obj.method(); //Method  
call
```

Static and Dynamic Binding

Static Binding: The binding which can be resolved at compile time by compiler is known as **static or early binding**.

Binding of all the method and fields that can not be overridden (static, private and final) is done at compile-time .

Static and Dynamic Binding

```
Static Binding class A {  
static void print()  
{    System.out.println("print in superclass.");  
}}  
class B extends A { static void print()  
{    System.out.println("print in subclass.");  
}}
```

```
class StaticBinding  
{    public static void main(String[] args)  
    {  
        A.print();  
        B.print();  
    }  
}
```

Static and Dynamic Binding

Static Binding :

Static binding is better performance wise (no extra overhead is required).

Compiler knows that all such methods cannot be overridden and will always be accessed by object of local class or by class it self Hence compiler doesn't have any difficulty in resolving method call.

Static and Dynamic Binding

Dynamic or late Binding: Binding that happen at run time.

In Dynamic binding compiler doesn't decide the method to be called.

Method overriding is example where dynamic binding happens.

Dyamic Method dispatch or Run time polymorphism

- When call or binding to polymorphic methods resolve at run time it is called **Dyamic Method dispatch or Run time polymorphism.**

Run Time polymorphisam is achived through run time or late or dynamic binding.

Dynamic Method dispatch or Run time polymorphism

Method overriding is one of the ways in which Java supports Runtime Polymorphism.

Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

Dynamic Method dispatch or Run time polymorphism

Implement :

When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs.

Thus, this determination is made at run time.

```
b.method();
```

```
b=new BaseClass(); b.method();
```

BaseClass b,

sub class object

```
b=new DerivedClass(); // Super class ref variable referring to
```

Dynamic Method dispatch or Run time polymorphism

Implement :

```
BaseClass b;  
b=new DerivedClass(); // Super class ref variable referring to  
  
b.method();  
b=new BaseClass();  
b.method();
```

Dynamic Method dispatch or Run time polymorphism

Implement :

At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed

A superclass reference variable can refer to a subclass object. This is known as upcasting. Java uses this fact to resolve calls to overridden methods at run time.

Invoke Base class constructor

Use **super** keyword .

Conditions for using of super keyword. Within

- subclass constructor.
- First statement within subclass constructor.
- The parameter in the super call must match the order and type of the variable declared in the super class constructor.

Super

The super keyword in Java is a reference variable which is used to refer immediate parent class object.

- Whenever the instance of subclass is created, an instance of parent class is created implicitly which is referred by super reference variable.

Super

Usage of Java super Keyword

- super can be used to refer immediate parent class instance variable.
- super can be used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

Why multiple inheritance is not supported in java?

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

- Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Why multiple inheritance is not supported in java?

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

(No sytnatx available for multiple inheritance)