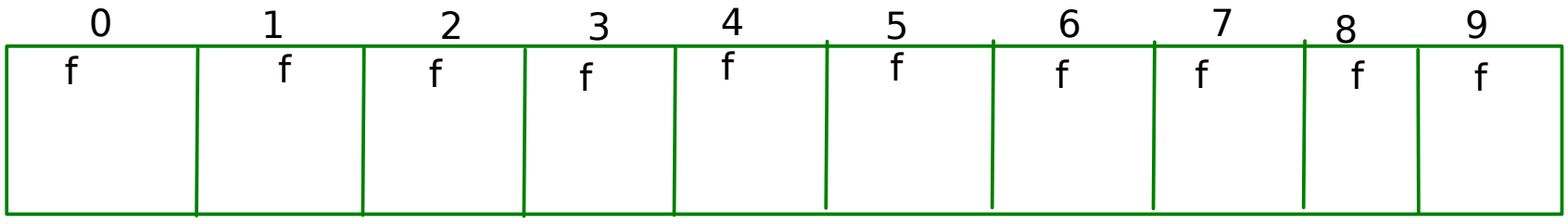



```
boolean arr[]= new boolean[10];
int economy = 0;
int first_class = 5;


 firstClass(){
 if(first_class<10){
 arr[first_class]=true;
 first_class++;
 }else
 {
 sysout("Do you want to book in economy")
 sysout("press 0 to cancel 1 to confirm")
 int choice = sc.nextInt();
 if(choice ==1)
        economyClass();
  }

  }
```

```
if(economy<5){
arr[economy]=true;
economy++;
}else
{
sysout("Do you want to book in first_class")
sysout("press 0 to cancel 1 to confirm")
int choice = sc.nextInt();
if(choice ==1)
        firstClass();
}
```

```
 Arrays -> Helper/Utility class
 java.util
```

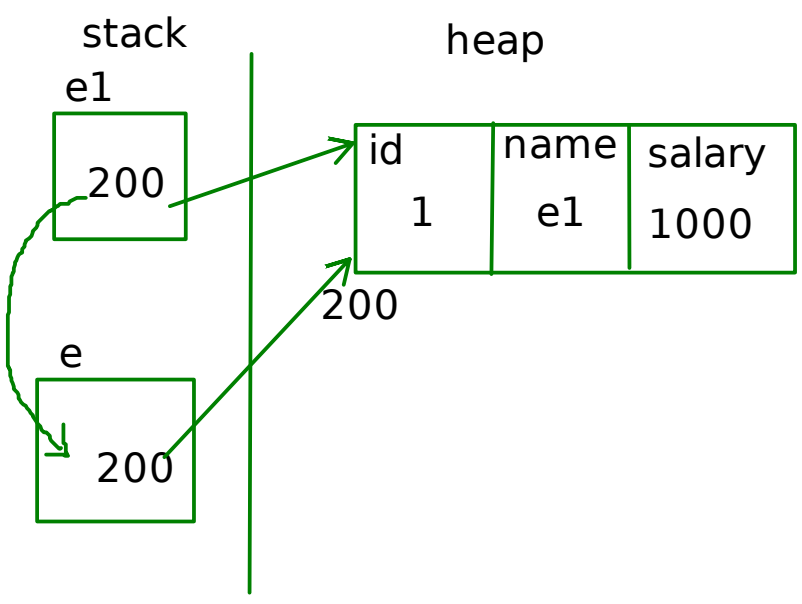
```
toString(arr);
sort(arr);
binarySearch(arr,key);
```

```
swap(int n2, int n2){

}
```

```
int n1=10, int n2=20
swap(n1,n2); // pass by value
```

```
 Employee e1 = new Employee();
 Employee e2 = e1;
```

```
acceptEmployee(Employee e){
e.empid = 1;
e.salary = 10000;
}
```




```
main(){
Employee e1 = new Employee;
acceptEmployee(e1); // pass by Reference

}
```

```
Singleton Design Pattern
class Test{
static Test tref =null;
private Test(){
}

public static Test getInstance(){
        if(tref == null)
                tref = new Test();
return tref;
}
}
```

```
class Test{
int num1 = 10;
static int num2 = 20;

static{
//num2=20;
}

{
num1=100;
}

Test(){
num1=1000;
}
}
```

```
static void method()
        - no this reference
        - should be called on classname
        -  cannot access nonstatic fields
```

OOP

has-a

Association
eg->
Human has-a Heart
Room has-a Window
Employee has-a Doj
Employee has-a Car

is-a

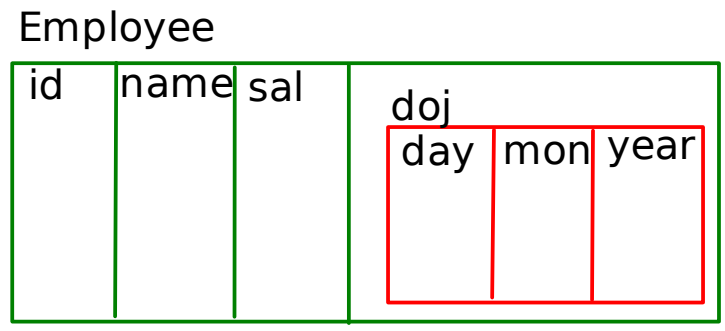
inheritance

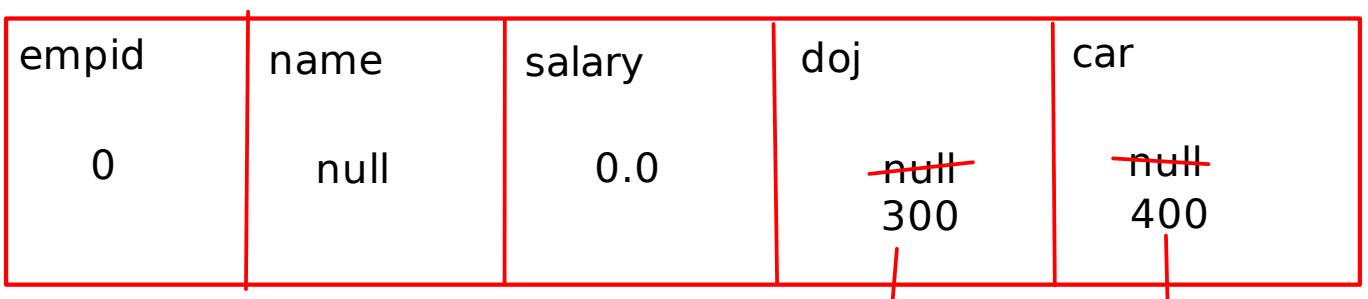
Types of Association
1. Composition
   - If entities are tightly coupled we use composition
2. Aggegration
   - If entities are loosely coupled we use aggegration

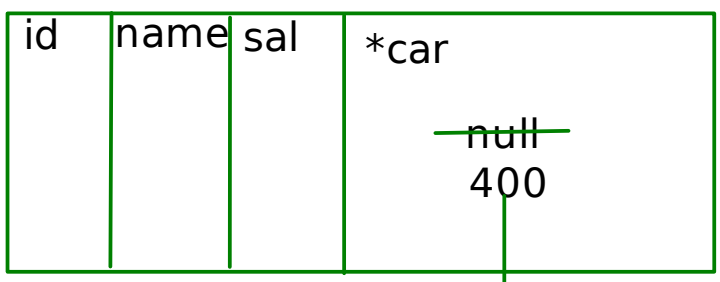
```
// CPP
class Employee{
Date doj; // Object
}
```

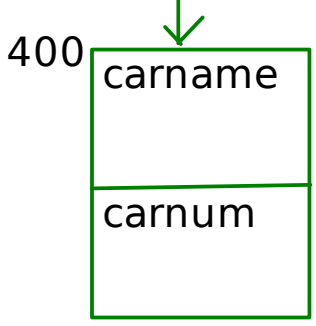
```
// Java
class Employee{
Date doj; // reference
Car car; // reference
}
```

new Employee();

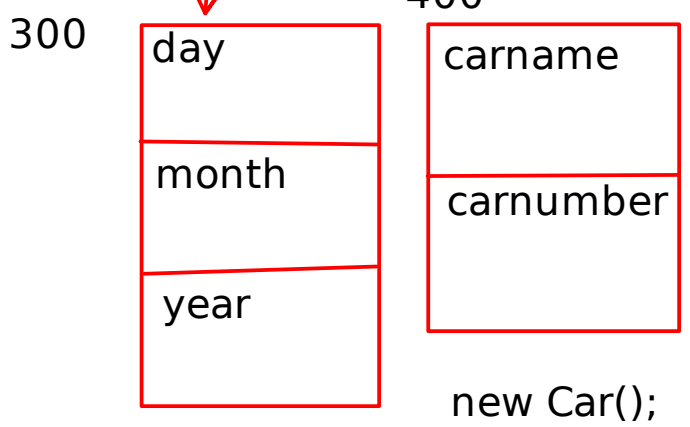








delete

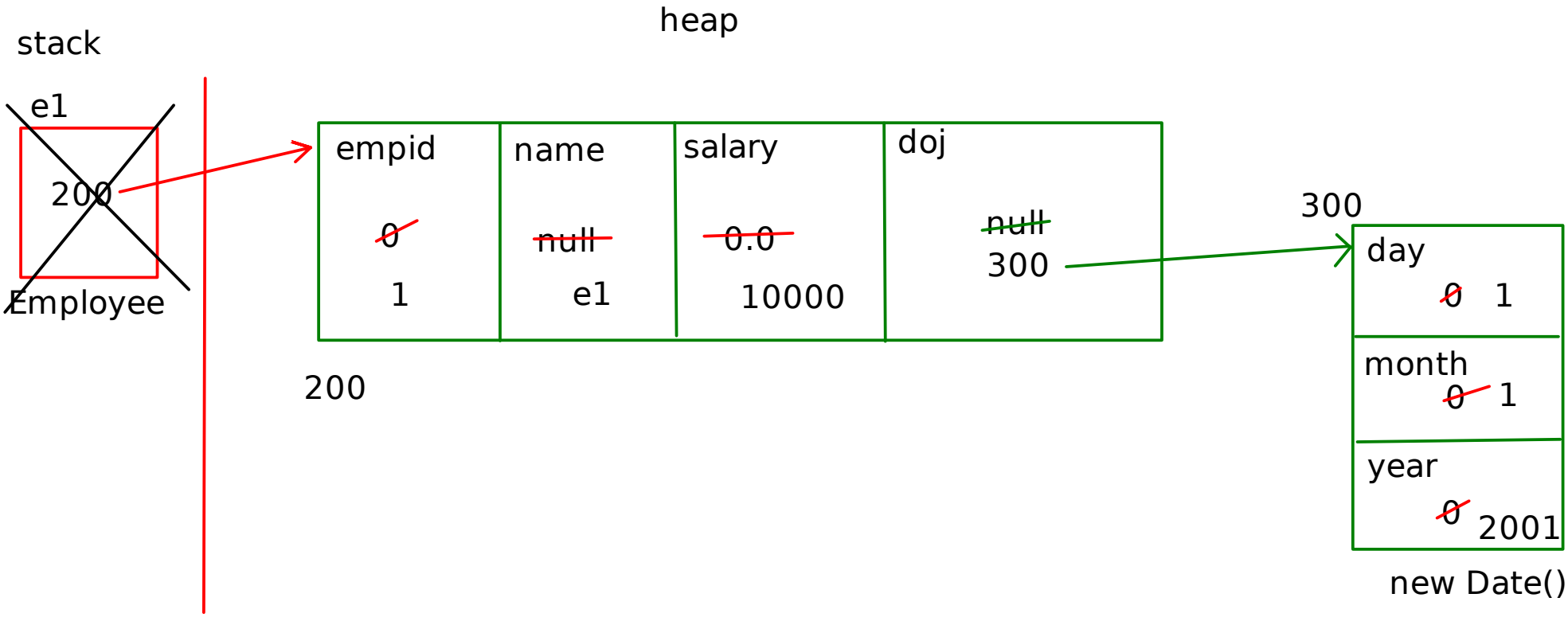


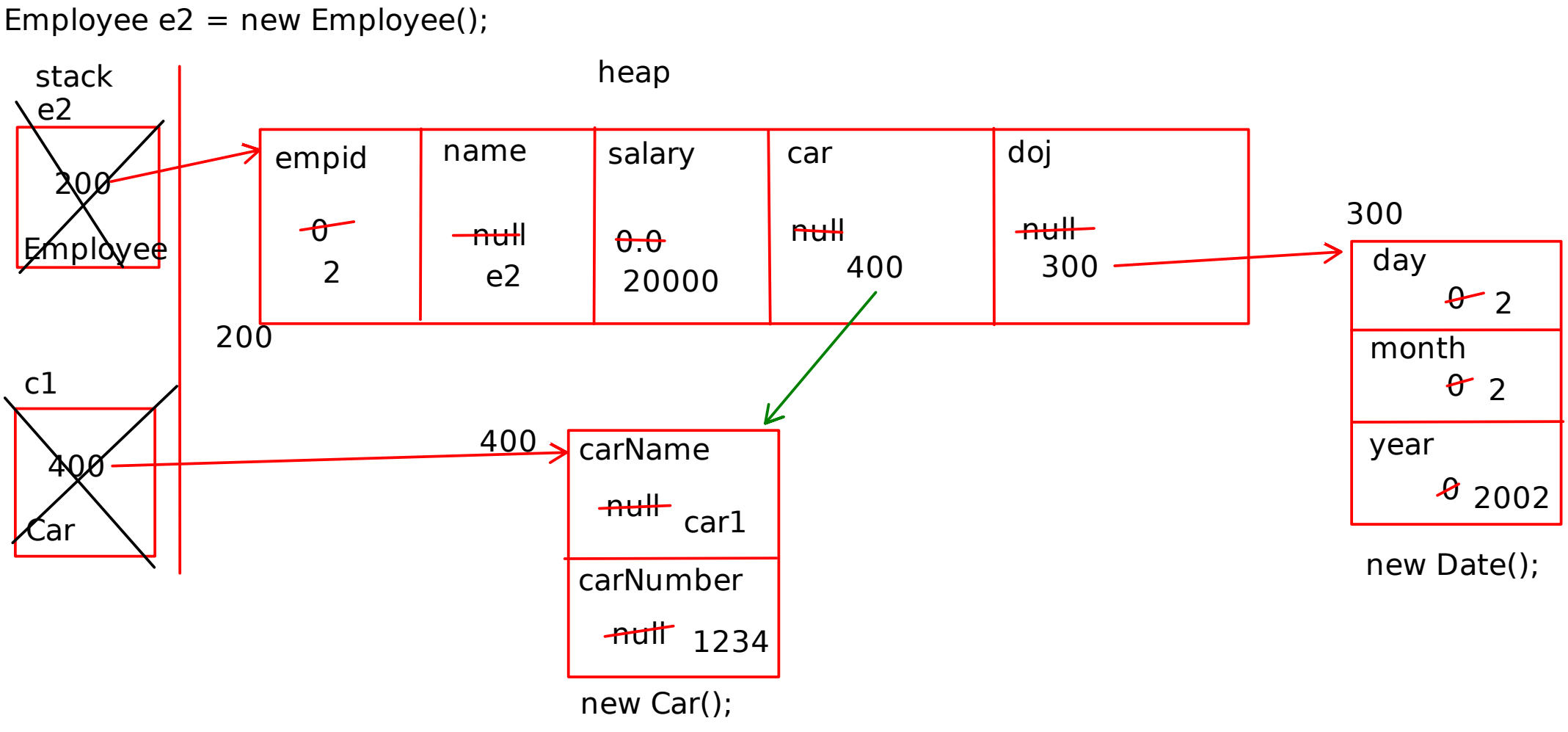
new Date();

new Car();

new Car();

Employee e1 = new Employee();





Employee Date Car Association

```
class Employee{
//Association
Date dobj; // reference
Car cobj; // reference

}


class System{
// Association
InputStream in; // reference
PrintStream err; // reference
PrintStream out; //reference
}
```

Composition ->
- Create the object of date class inside the employee class
- object will be created by using new Date()

Aggegration ->
- Create the object of the Car class outside the employee class
- object will be created by using new Car();

is-a Relationship
inheritance

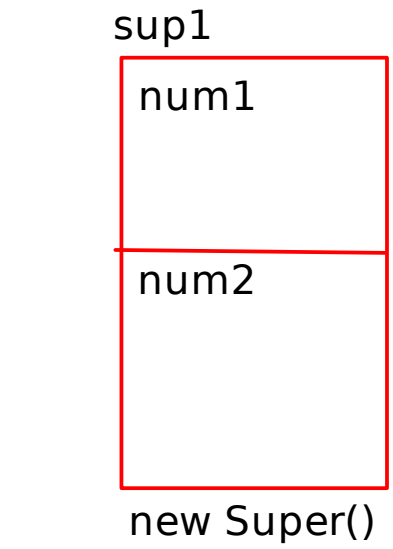
eg ->
Apple is a fruit
Laptop is a device
Employee is a Person
Manager is a Employee

```
class Base// Parent
{

}

class Derived:Base // Child{

}
```

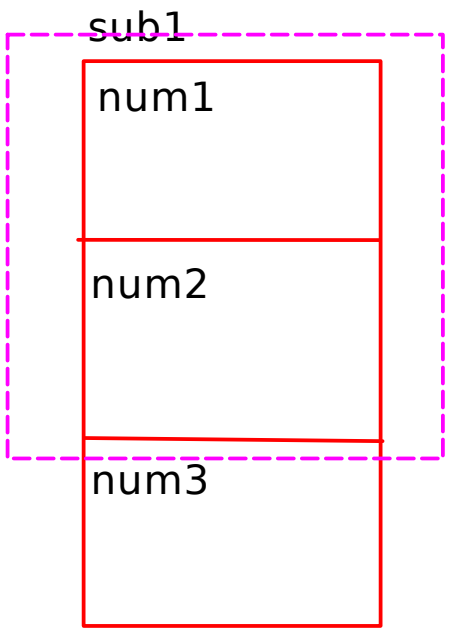
```
class Super // Parent
{

}

class SubClass extends Super // Child
{


}
```

Types of Inheritance
~~Mode of Inheritance~~
Hybrid Inheritance
~~Diamond Problem~~
~~Virtual Base class~~
~~virtual function~~
~~pure virtual function~~
Early binding
Late Binding
Upcasting
downcasting
RTTI
casting_operators
virtual destructor

abstact method
abstract class
interface
super



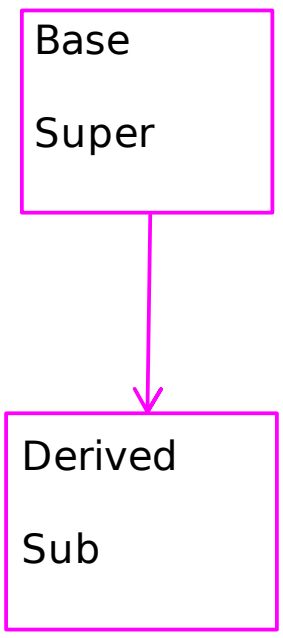
Super class will initialize the members num1 and num2



Super class will initialize the members num1 and num2
Sub class will initialize the member num3

CPP

Java



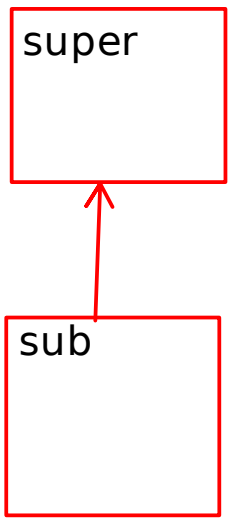
All the members of base class inherites into the derived class except
All the members of Super class inherits into the sub class except
1. Constructor

Ctor calling sequence
Base -> Derived
Super-> Sub

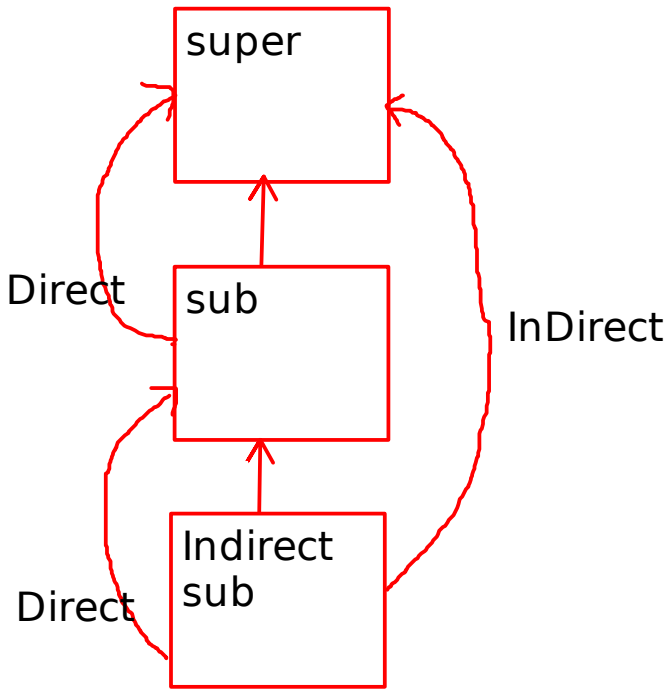
<

- When we create object of subclass then super class ctor gets called first and then the sub class ctor.
- Always the parameterless ctor of super class gets called.
- to call the parameterized ctor of super class from the sub class ctor use the super() statement.
- to perform ctor chaning for the same class we use this() statement
- to perform ctor chaning in between super and sub class we use super() statement
- this() or super() should be the first statement in the ctor body.

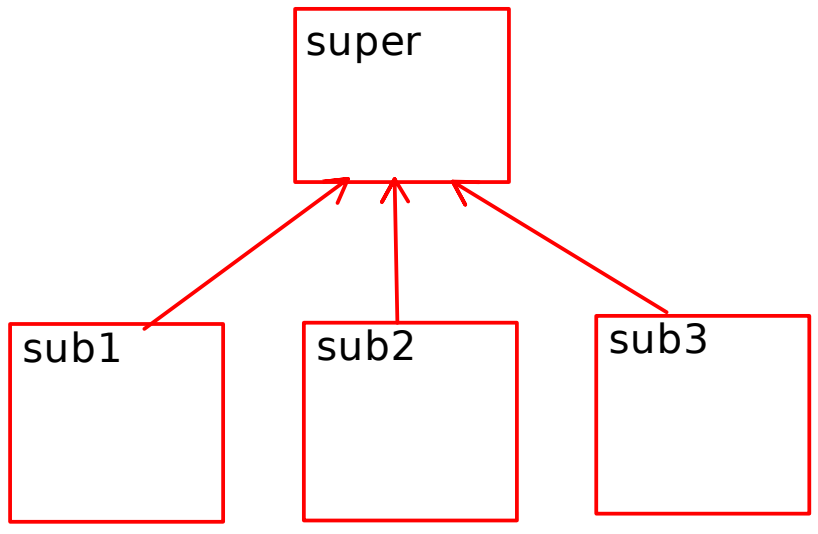
Types of inheritance



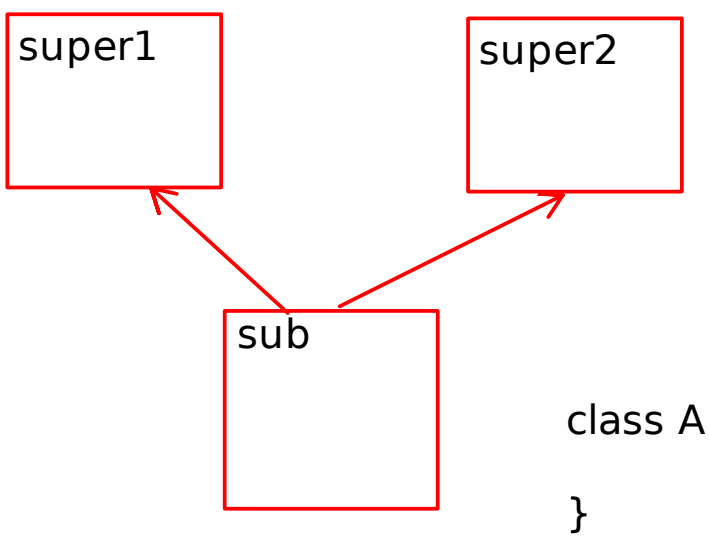
1. Single Inheritance



2. Multilevel Inheritance



3. Hirerachical Inheritance



4. Multiple Inheritance

5. Hybrid Inheritance
- Combine any 2 type of inheritance
to create hybrid inheritance

Java does not support multiple implementation inheritance
Java does support multiple interface inheritance

```
class A{

}

class B{

}

class C extends A,B // Not Allowed
{

}
```

```
interface E{

}

interface F{

}

interface G extends E,F // Allowed
{

}
```

Rules of Method Overrding

1. Name and the signature of the method in sub class should be as that in super class
2. visibility of the overriden method in sub class should be wider that than that of in super class.
   eg -> If method of super class is protected or default then
   we can make the overriden method in subclass as public.
   we cannot make the overriden method in subclass as private.

3. Return type -> We will see after upcasting
4. Exception list

Final

1. variable -> once initialized cannot change the value inside it
2. field -> once initialized cannot change the value inside it
3. method -> cannot override in sub class
4. class -> cannot inherit this class in to the further sub classes