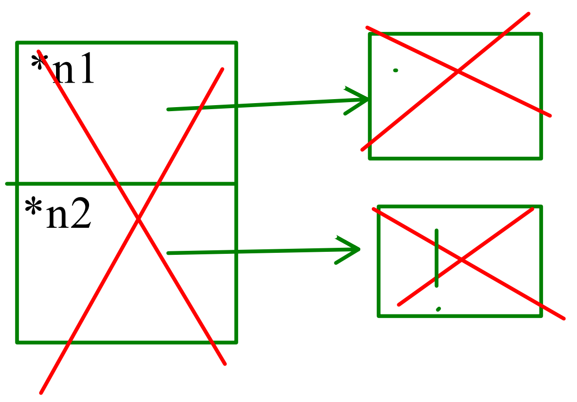
Revison
1. Variable arity Method
2. Method Overloading
3. Pass By Value/Reference
4. Final
5. Static

Singleton Design Pattern

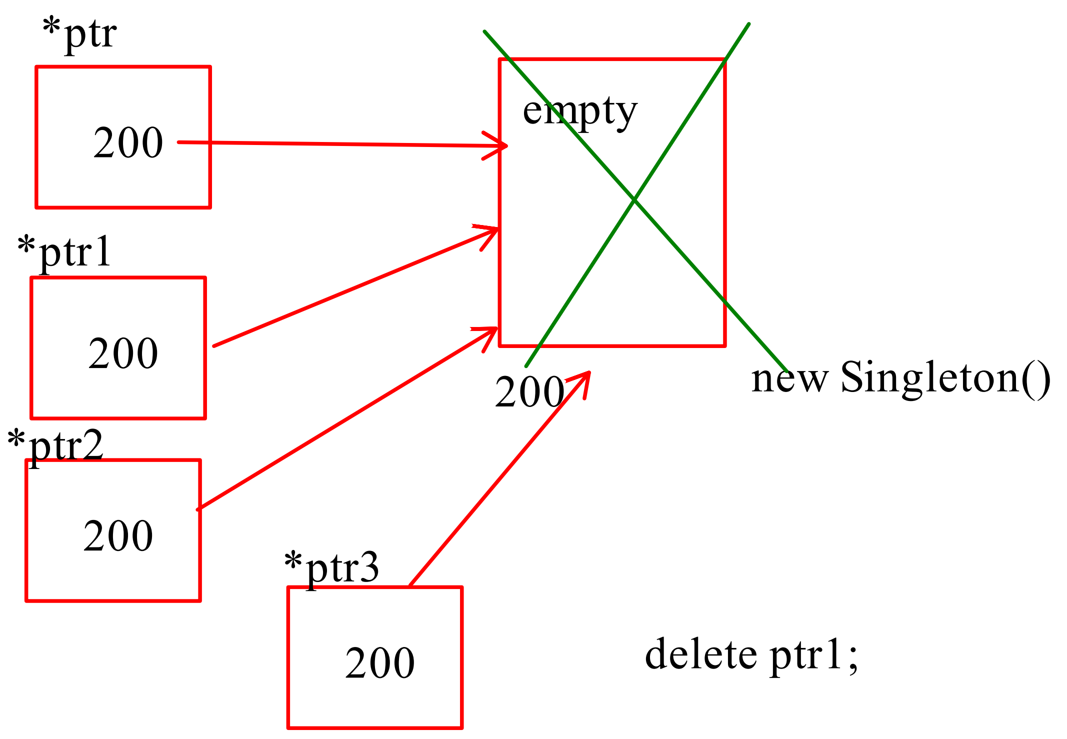
```
class Singleton{
// step-3
static Singleton ref = null;

// step-1
private Singleton(){
}

//step-2
static Singleton getInstance(){
        if(ref == null)
                ref = new Singleton();
        return ref;
}

}
```

```
main(){
Singleton s1 = Singleton.getInstance();
Singleton s2 = Singleton.getInstance();
Singleton s3 = Singleton.getInstance();
Singleton s4 = Singleton.getInstance();
}
```

```
class test {
int *n1;
int *n2;
}

test *t1 = new test();
delete t1;
```



```
Base *bptr = new Derived();
Derived *dptr =  (Derived*) bptr;
delete dptr;
```



delete ptr1;

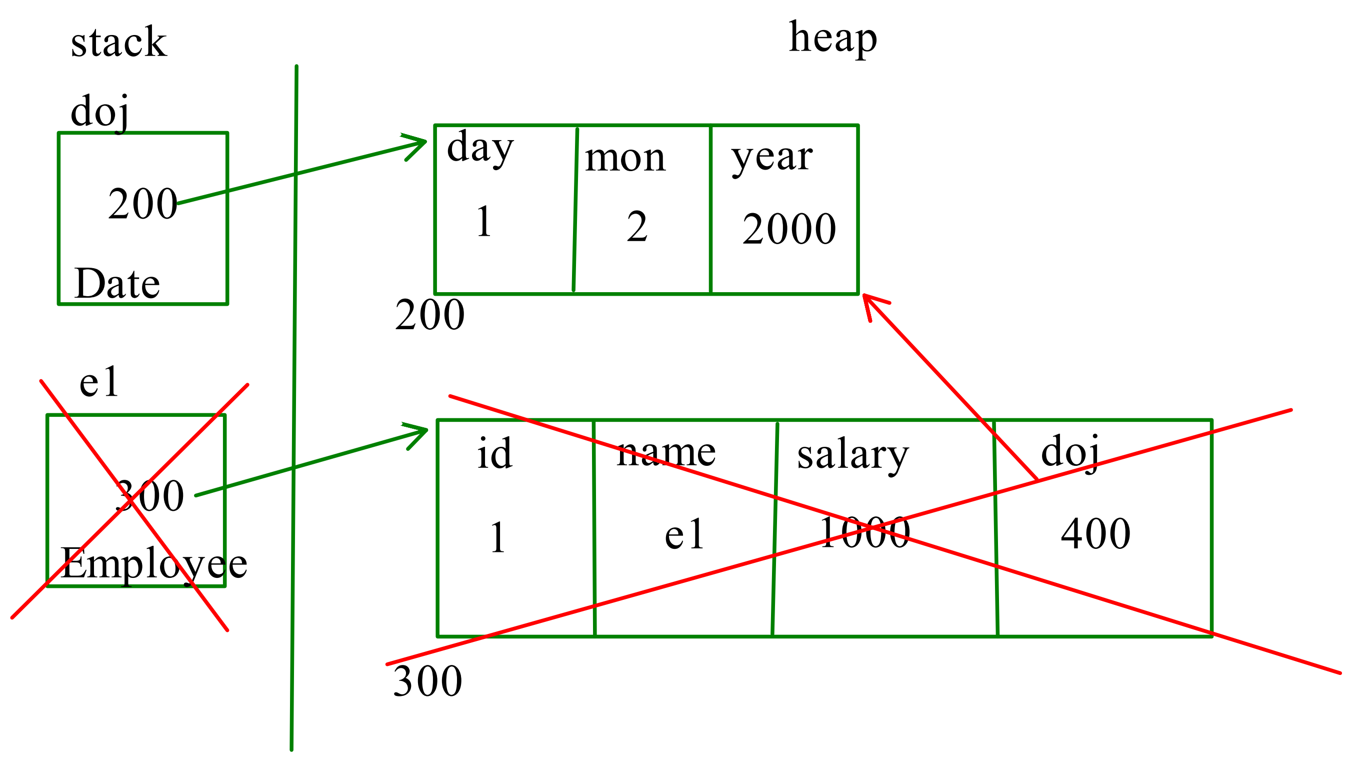
Hirerachy

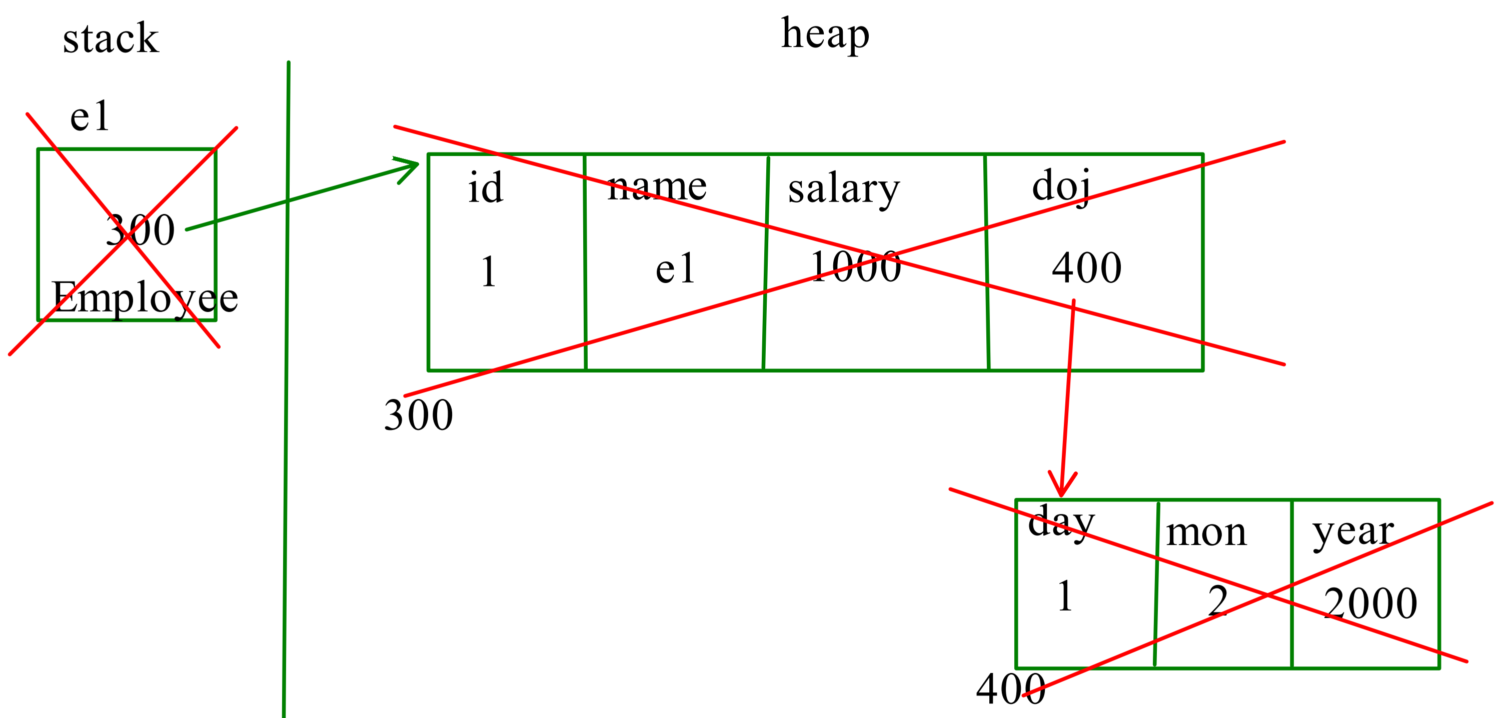
1. has-a relationship -> Association
    - Composition->Tight coupling
    - Aggegration -> loose Coupling
2. is-a relationship -> Inheritance

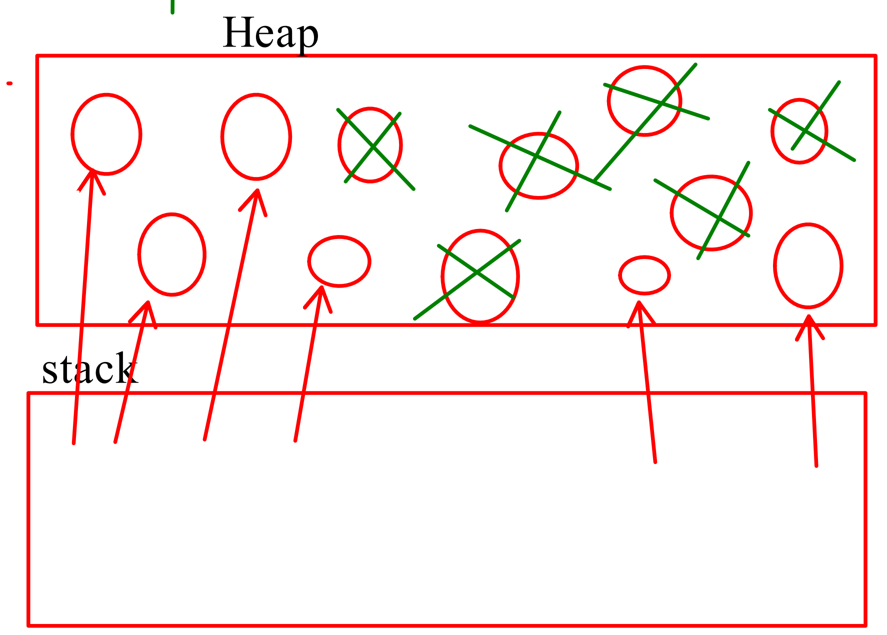
Human has-a heart
Car has-a engine
Room has-a wall

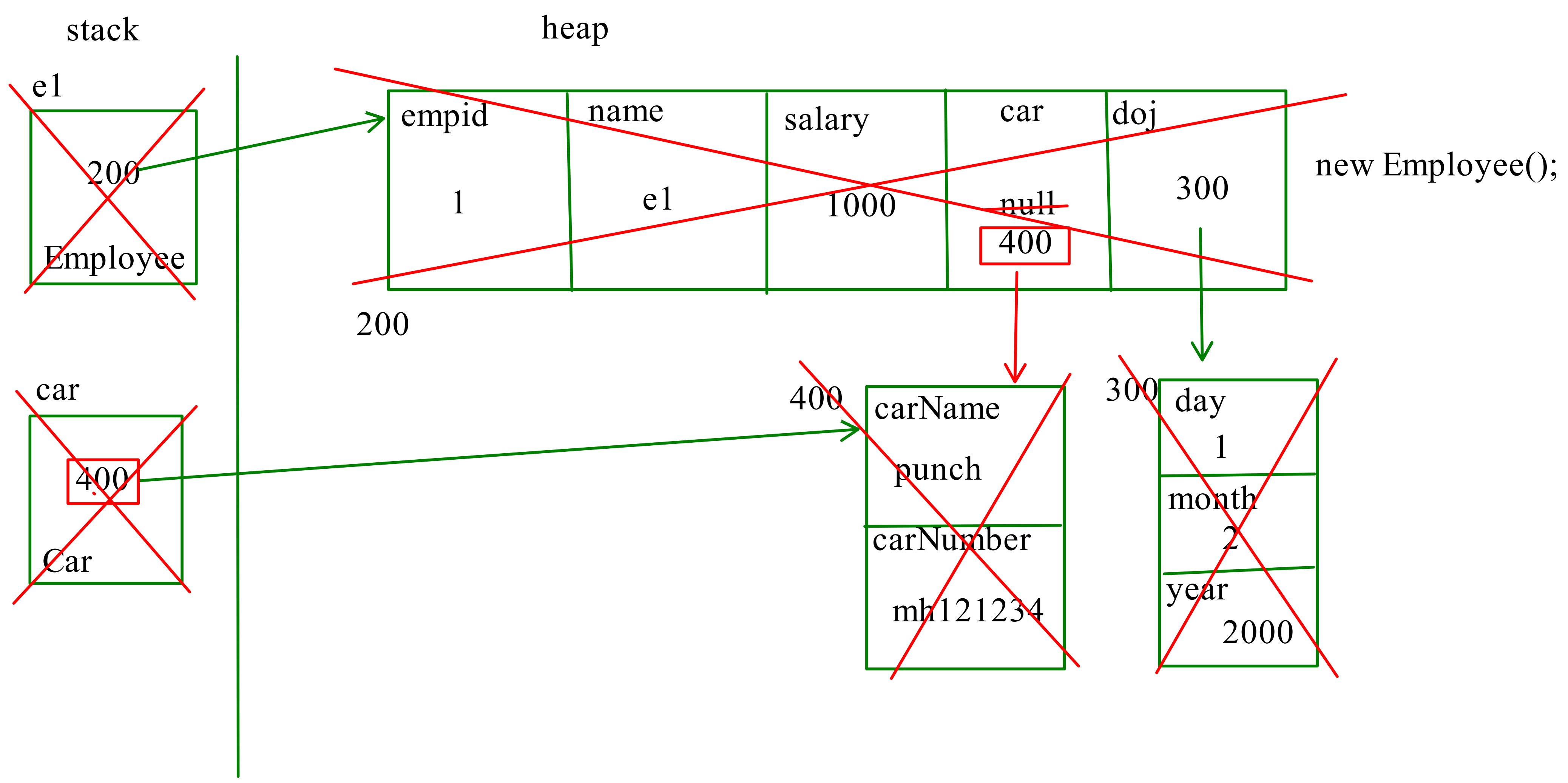
Employee has-a doj
Employee has-a car

```
Employee{
Date doj; // Composition
Car *cptr; // Agggegration

}
```









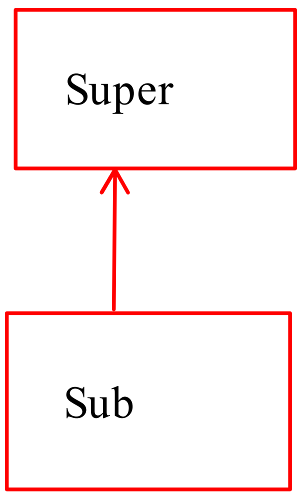
Association
- If has-a relationship exists between two entities then use Association
- In java, we can create references of other classes as the field inside our class.
- Composition
    - In these references if we create the object inside the constructor then we are acheiving composition
- Aggegration
    - If the reference are kept null and the objects in these refrences are passed from out side the class using setters or any other methods then we are acheiving aggegration

is-a relationship -> Inheritance

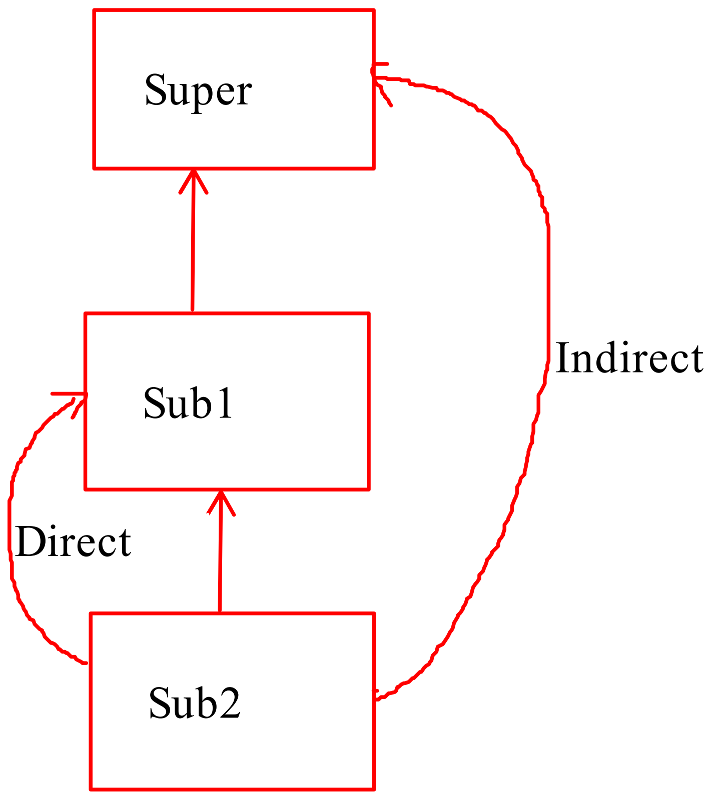
Employee is-a person
Manager is-a Employee
Apple is-a Fruit
Car is-a Vehicle
....

```
class Parent // Base -> Super
{

}

class Child extends Parent // Derived-> SubClass
{

}
```

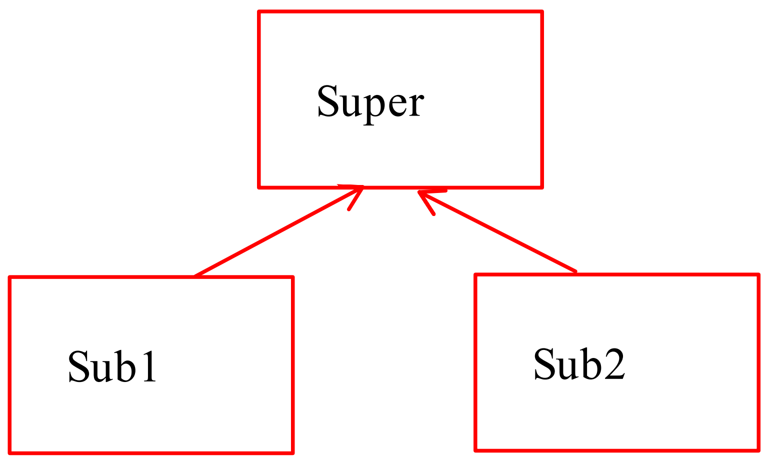
Types of Inheritance



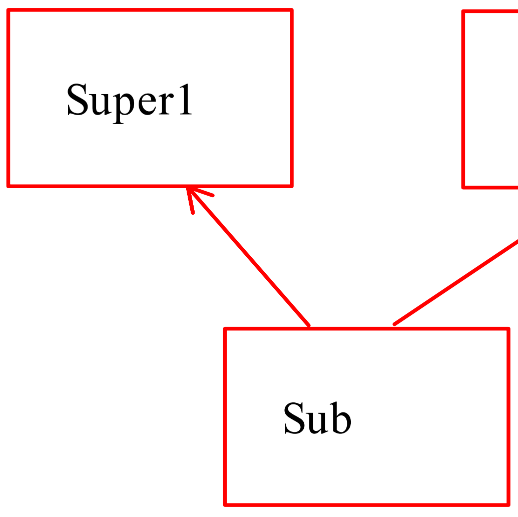
1. Single Inheritance



2. Multilevel Inheritance



3. Hirerachical Inheritance



4. Multiple Inheritance

5. Hybrid Inheritance

```
class A{
}

class B extends A{
}
```

single inheritance

```
class A{
}

class B extends A{
}

class C extends B{
}
```

Multilevel inheritance

```
class A{
}

class B extends A{
}

class C extends A{
}
```

hirerachical inheritance

```
class A{
}

class B{
}

class C extends A,B{

}
```

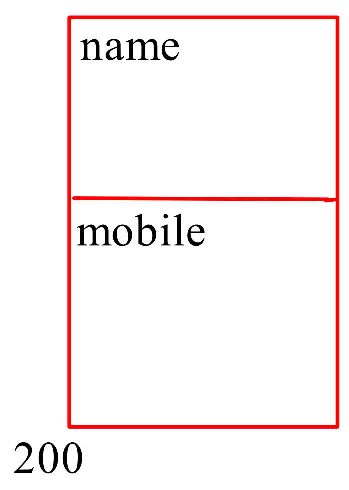
// multiple class inheritance
non supported in java

```
interface A{
}

interface B{
}

class C implements A,B{

}
```

multiple interface inheritance
supported in java

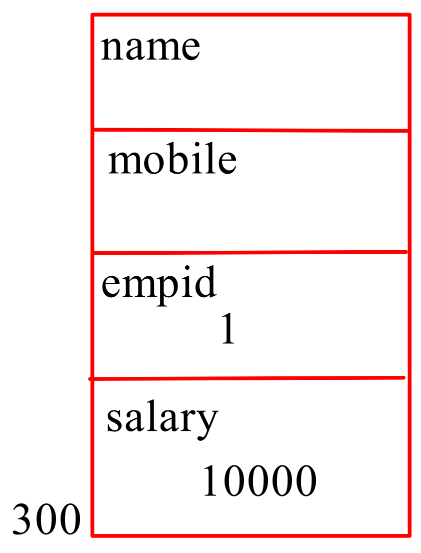
```
interface A{
}

interface B{
}

interface C extends A,B{
}
```

Person p = new Person();



```
Base *bptr = new Base();
Derived *dptr = new Derived();

bptr->m2();
dptr->m2(); // Early Binding
```

Employee e = new Employee();



Method Overriding
- Redefining the method of super class once again in sub class with same name and signature is called as method overriding.
- Why to do method overriding
  1. Implemenetation of super class method is 100% incomplete
  2. Implementation of super class method is partial complete
  3. If the required implementation in sub class is completely different from the super class method
- Rules for Method Overiding
1. The name and signature of the overriden method must be same as that of super class method
2. The visibility modifier of the overriden method must be same or of wider type as that of super class
3. The return type of the overriden method should be same or it should be the sub class of the return type of the method in super class
4. The exception list of overriden method should be same or subset of the exception list from the super class method

this and super
- this is used to point at the memebrs of the same class
- super is used to point at the memebers of the super class

## super
- To call the methods of super class inside the overriden methods of sub class use `super`. method_name
- To invoke ctor of super class from the sub class ctor use super() statement;

Runtime Polymorphism

```
Base *bptr = new Derived();
bptr->f1();
```

```
virtual void f1(){

}
```

Downcasting
RTTI -> typeid()

```
int arr[] = {10,20}
```