```
int size = sc.nextInt();
Point2D [] arr = new Point2D[size];
for(int i =0;.....)
```

1, 5

```
if(arr[1].isEquals(arr[5]))
        sysout(Points are equal);
else
arr[1].calculateDistance(arr[5]);
```

Array

```
int arr[] = new int[5];

int arr[][] = new int [2][3];

// Ragged Array
int arr[][] = new int[2][];
arr[0] = new int[3];
arr[1] = new int[5];
```

```
    void add(int ... arr)
```

#Final
Varaibles
Fields
Methods
Class

```
class Circle{
int radius;
final static double PI = 3.14;
}
```

Static
- Sharing
- Class level Members
- Designed to be accessed on the classname using ` . ` operator

OOP
1. Abstraction
    - System.in, System.out, Scanner, System.out.println()
2. Encapsulation
    - Class (eg -> Employee, Student, Date, Time)
3. Modularity
    - .java files, packages
4. Hirerachy
    - has-a (Association)
    - is-a (Inheritance)

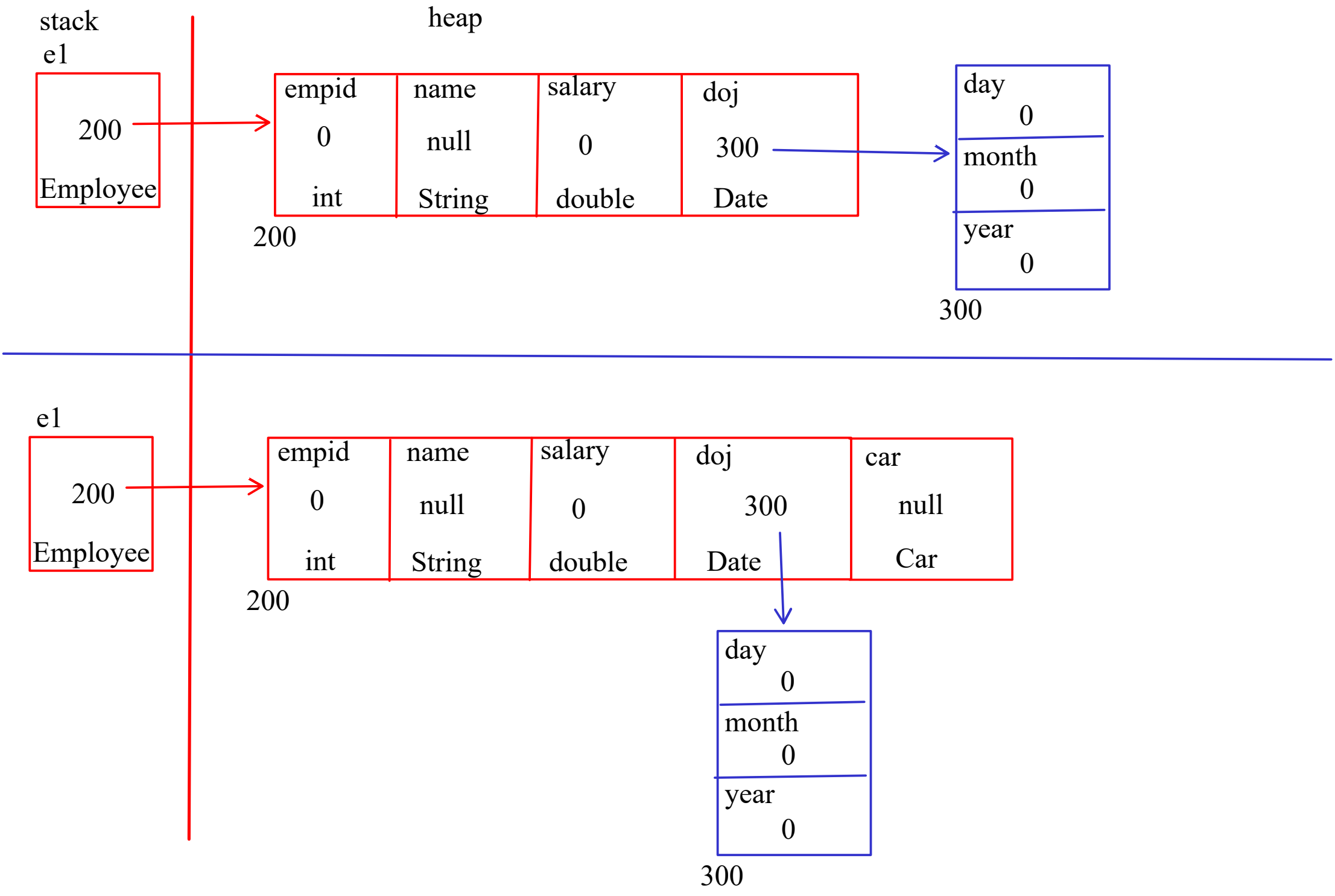
1. Typing/Polymorphism
    - Compile Time -> Method Overloading
    - RunTime ->
2. Concurrency
3. Persistance

Association
has-a relationship
Human has-a Heart
Car has-a Engine
Employee has-a Doj
Room has-a Window

```
// Dependency
class Date{

}

// Dependent
class Employee{
//field
Date doj; //reference
Date dob; // reference

}
```

```
class Person{
String name;
String mobile;
}
```

Composition
    - If the entities are tightly coupled
Aggegration
    - If the entities are loosely coupled

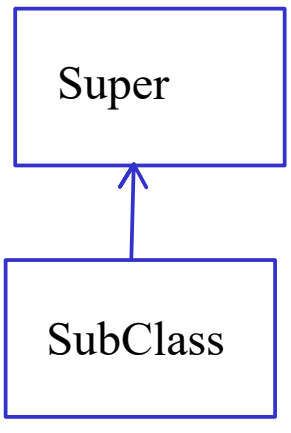


Inheritance
- is-a Relationship exists between 2 entities use inheritance
- eg -> Employee is-a Person
  Circle is-a Shape
  Mobile is-a Device

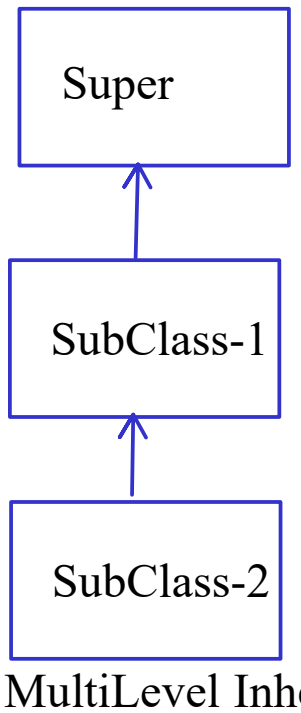
Person -> Parent (CPP-> Base) (Java -> Super)
Employee -> Child (CPP-> Derived) (Java -> SubClass)

```
class A{

}

class B :  A{

}

class C : A{

}
```

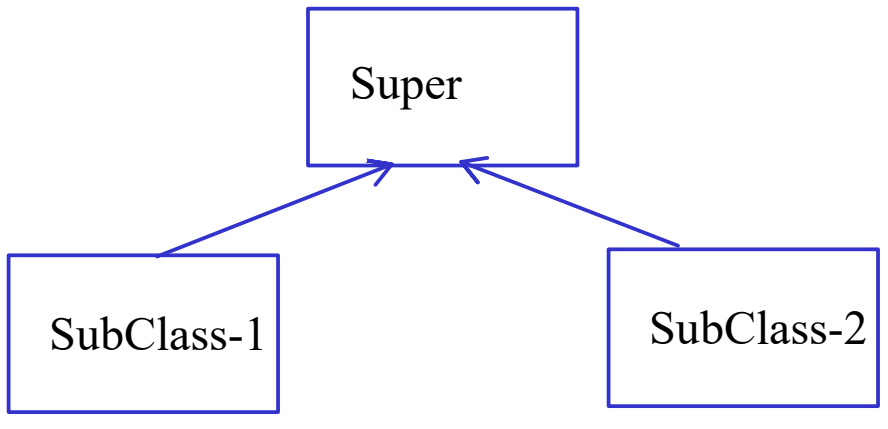
#Types of Inheritance



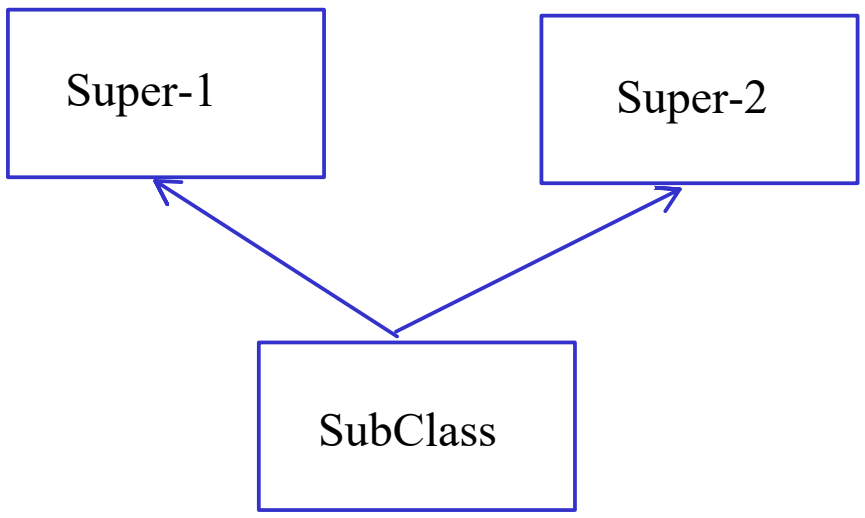
1. Single Inheritance



2. MultiLevel Inheritance

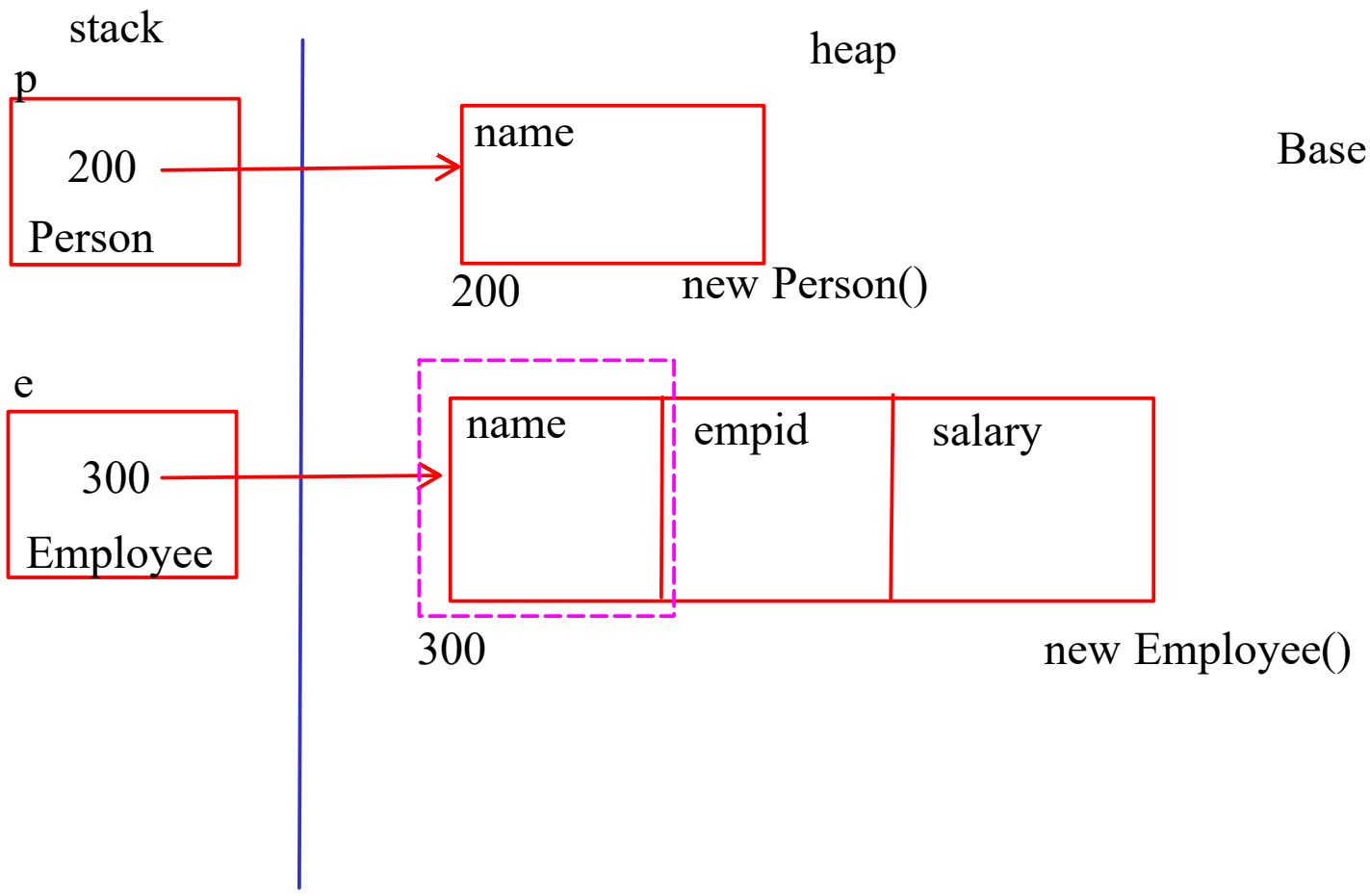


3. Hirerachical Inheritance



- Java Does not support Multiple Class Inheritance
- Java Does Support Multiple Interface Inheritance

4. Multiple Inheritance



Base *bptr = new Base();

super()
- If we want to do the ctor chaining within the same class then we use this() statement
- If we want to do the ctor chaining between the subclasss and the super class use super() statement
- super() statement should be the first statement inside the ctor body.

Upcasting
- storing the object of subclass into the superclass reference is called as upcasting

Object Slicing
- When upcasting is done then the super class reference can point only at the super class members. It cannot point at the sub class members.
- this is called as object slicing

Downcasting
- Converting the Super class reference into the sub class reference is called as Downcasting
- At the time of downcasting explict typecasting is mandatory
- If downcasting fails it throws ClassCastException

```
vector<Person*> personList;
personList.push_back(new Employee());
personList.push_back(new Student());
```

```
vector<Employee *> empList;
empList.pushBack(new Manager());
empList.pushBack(new SalesMan());
empList.pushBack(new SalesManager());
```

Method Overriding

- Defining the method of super class once again into the subclass with same name and signature is called as Method overriding
- Why to perfom method overriding
    1. If implemenation of super class method is partial complete
    2. If implementation of super class method is 100% in complete
    3. If implementation of sub class method needs to be different from the super class method.
- In Overriding the visibility of the sub class methods should be same as that of super class methods or it should be of wider type
- In Overriding the return type of sub class should be same as that of super class method or it should be the subclass of the return type of super class method
- In Overriding the sub class method exception list should be same as that of super class method exception list or it should be the subset of it.

Object Class