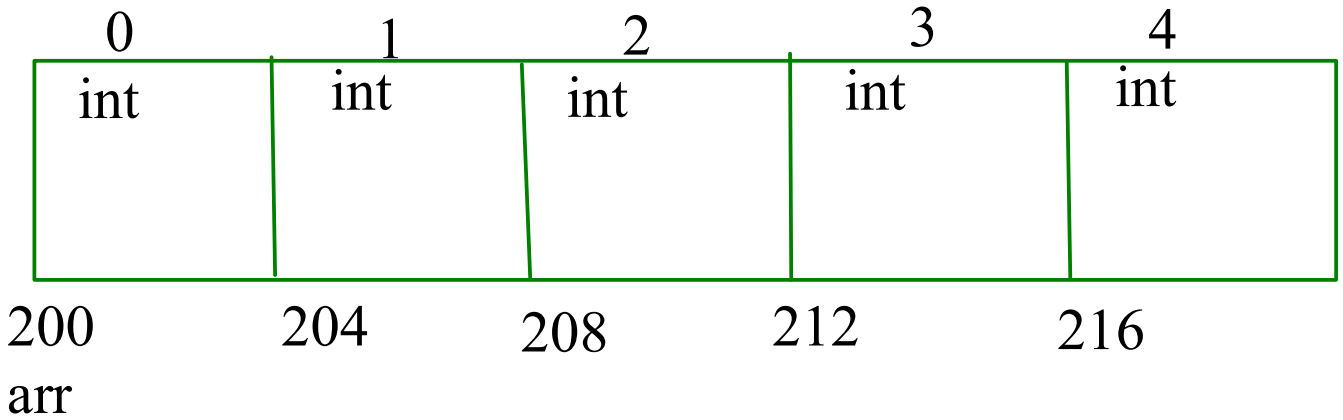
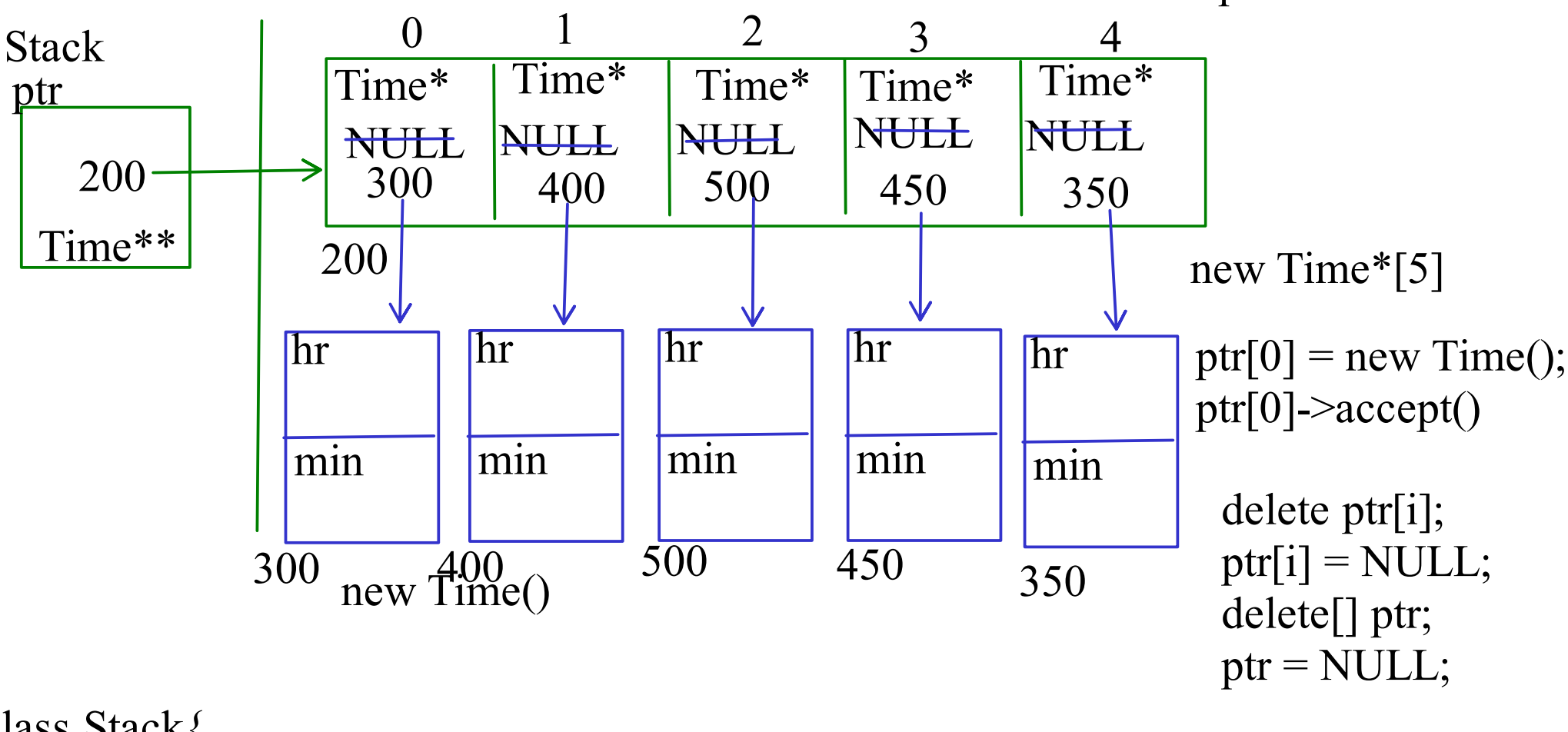
Static
- Datamember
    - Memory is allocated on data section only once at the time of program loading
    - It needs to initialized outside the class on global scope using class name and ::
- Member Functions
    - These are designed to be accessed on classname using ::
    - static member functions do not get this pointer.
    - In these functions we can access only static data members, we cannot access non static data members



int* arr[5];

int* e=arr[0]
arr[0] = &

new Employee*[5];
Time**ptr = new Time*[5];



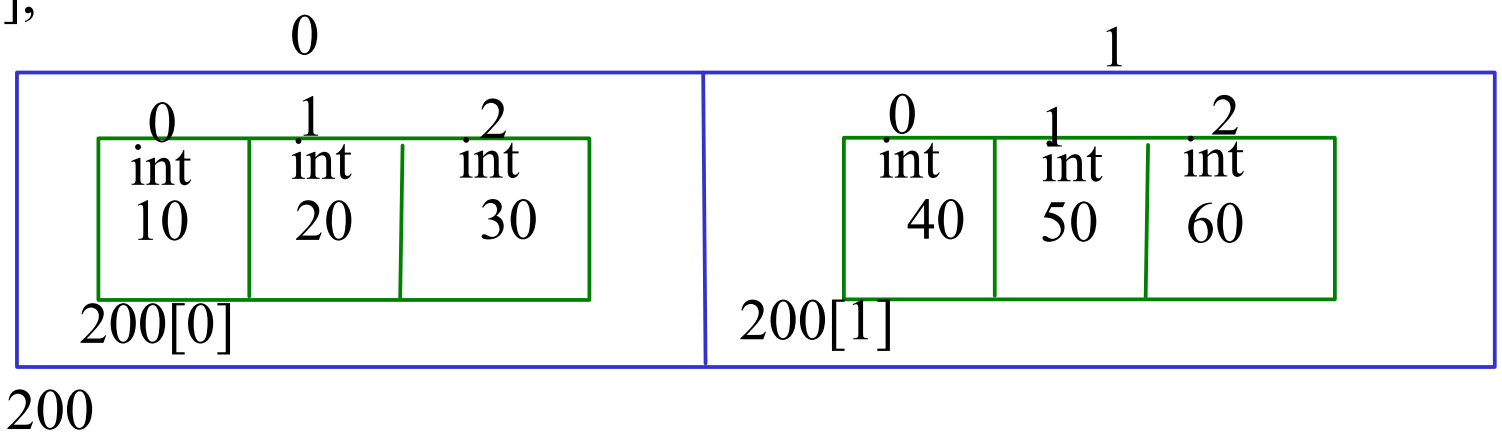
ptr[0] = new Time();
ptr[0]->accept()

delete ptr[i];
ptr[i] = NULL;
delete[] ptr;
ptr = NULL;

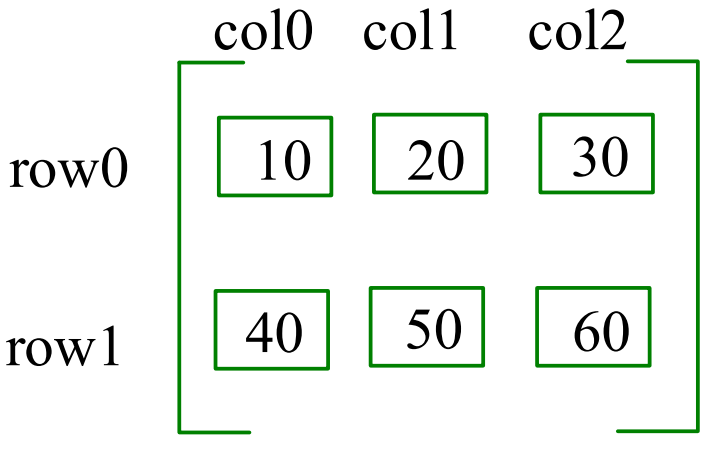
```
class Stack{
int size;
int *ptr;
int top;

public:
        Stack(int size=5){
        this->size = size;
        top = -1;
        ptr = new int[size];
}

}
```

```
pop(){

}

peek(){

}

push(int element){
if(!isFull()){
ptr[++top] = element;
}
```

int arr[2][3];

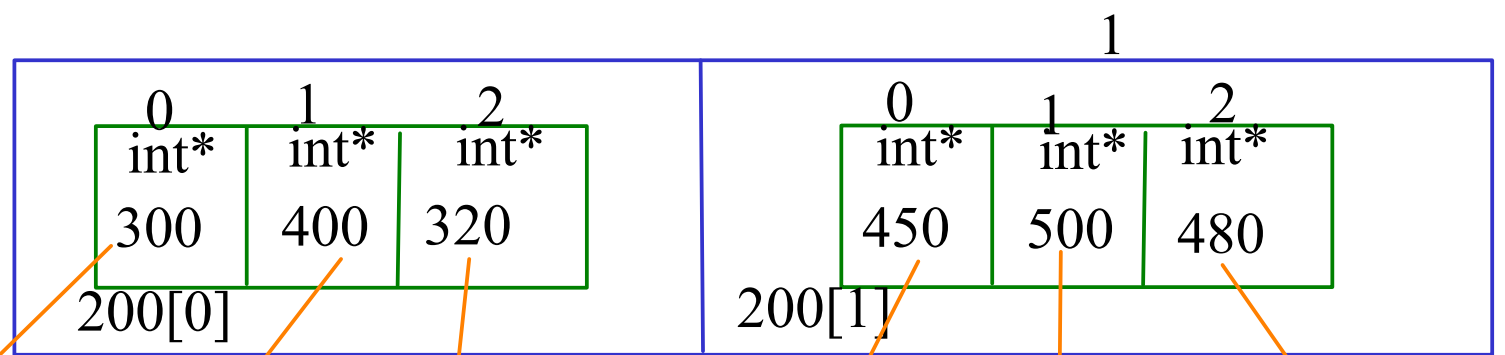


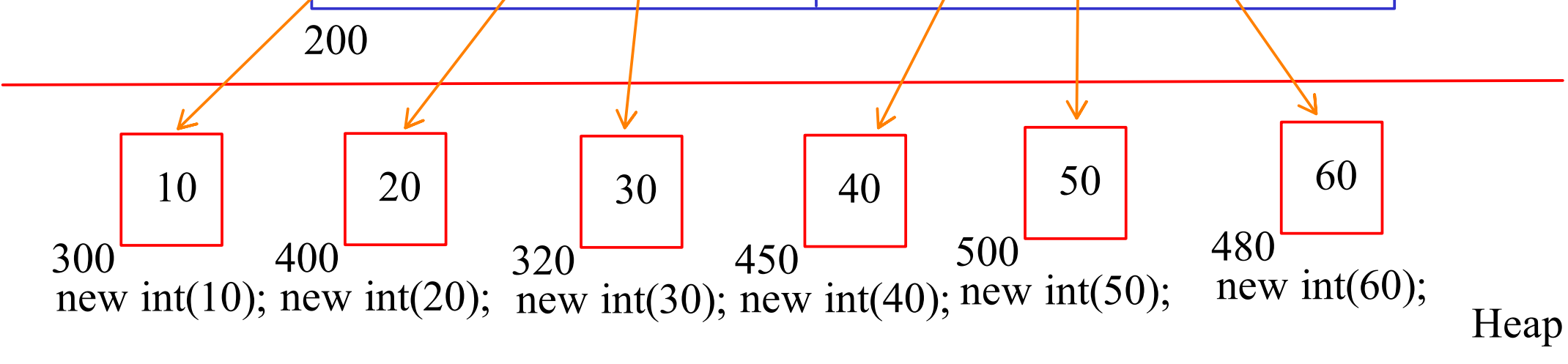
200[0][2]



int *arr[2][3]

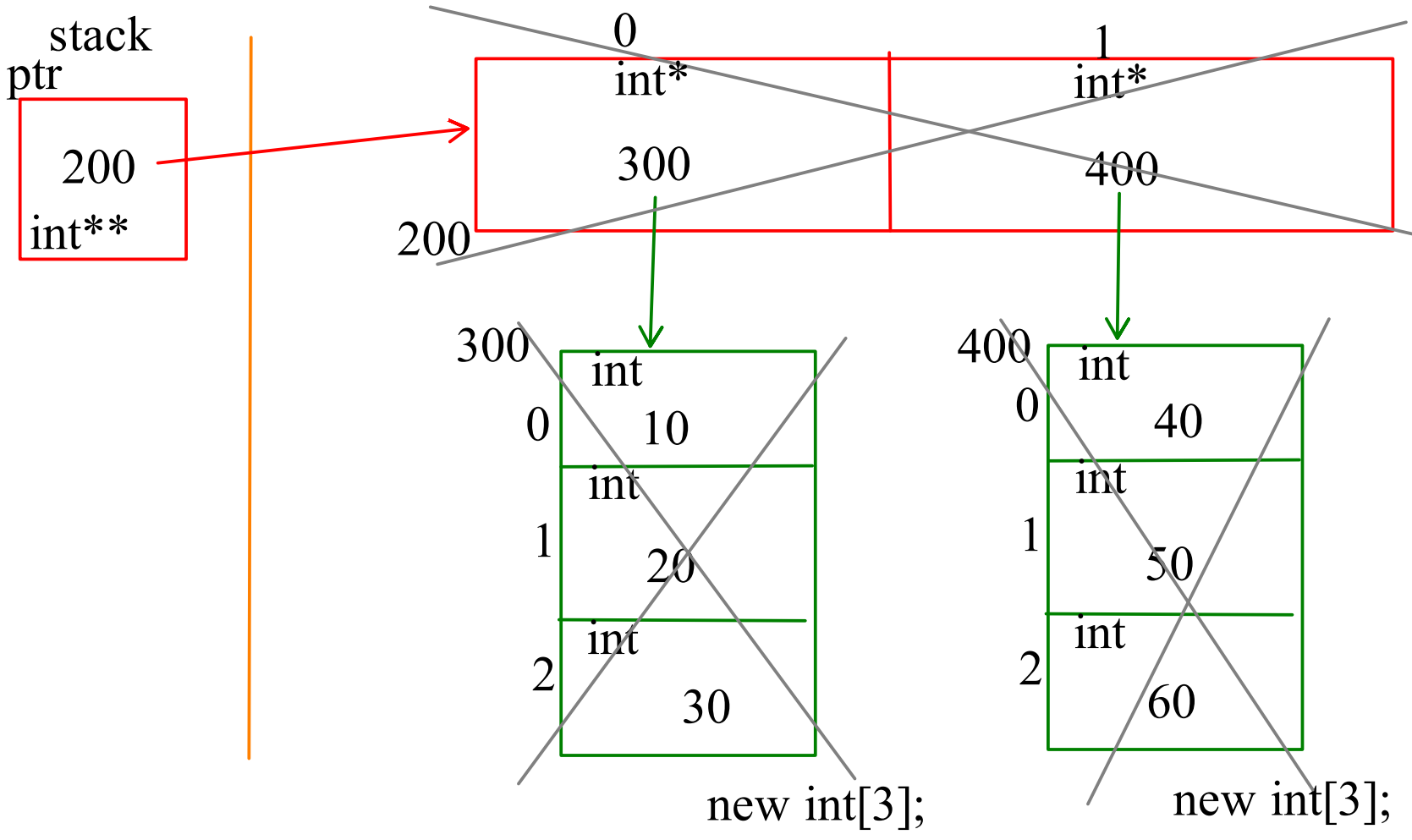
Stack





new int(10); new int(20); new int(30); new int(40); new int(50); new int(60);

Heap



Heap

new int[2];
ptr[0] = new int[3];
ptr[1]=new int[3];

//300[0] =10;
//200[0][0] = 10;
ptr[0][0] = 10;

//delete []300;
//delete []200[0];
delete []ptr[0];
delete []ptr[1];
delete [] ptr;

enum
- It is a enumarated user defined type
- It is used to provide string identifiers for the integer constants

```
displayDate(){
if(weekday == WEEKDAY.SUNDAY)
cout<<"Sunday"<<endl;
else if(weekday == 2)
cout<<"Monday"<<endl;
}

int main(){

displayDate();
}
```

```
enum WEEKDAY{
Sunday=1,
Monday,
Tuesday,
Wednesday,
Thursday,
Friday,
Saturday
}
```

Hirerachy
- type of Relationship
1. ASSOCIATION (has-a)
2. INHERITANCE(is-a)

```
Employee
Person
Car
```

```
Date{}
Doj
Dob
Manfacturing_date
```

Association
- When ever has-a relationship exists between two entities then we use association

eg- Human(Dependent) has-a heart (dependency)
Car has-a engine
Room has-a wall

- It is further classified into two types
1. Composition - It represents tight coupling
2. Aggegration - It represents loose coupling

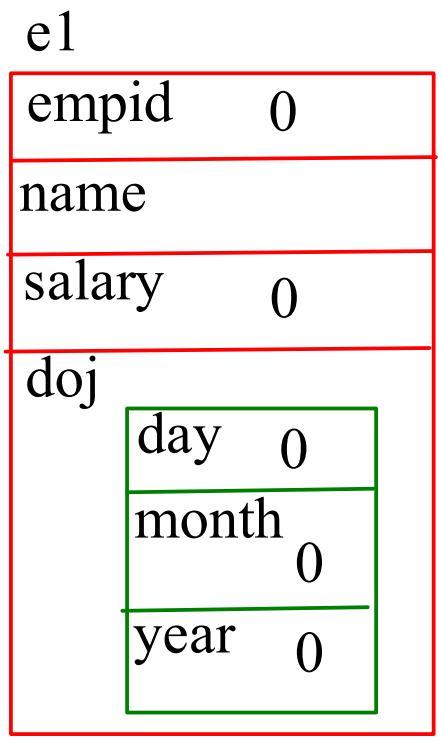
eg - Room has-a Window
Bike has-a Storage

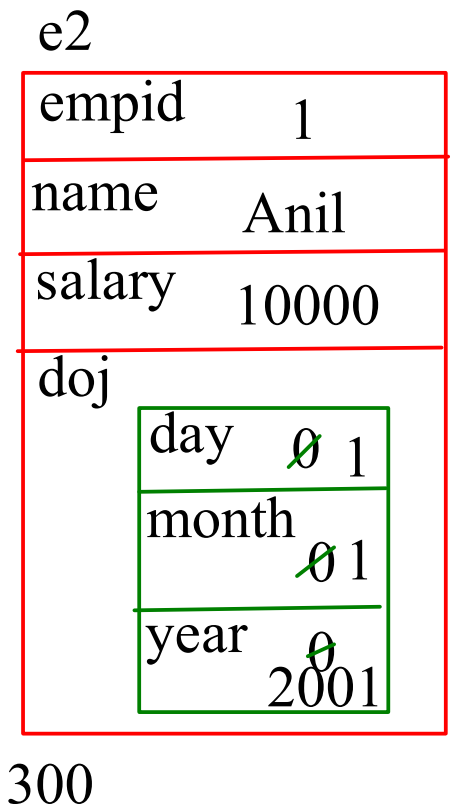
Employee has-a Doj

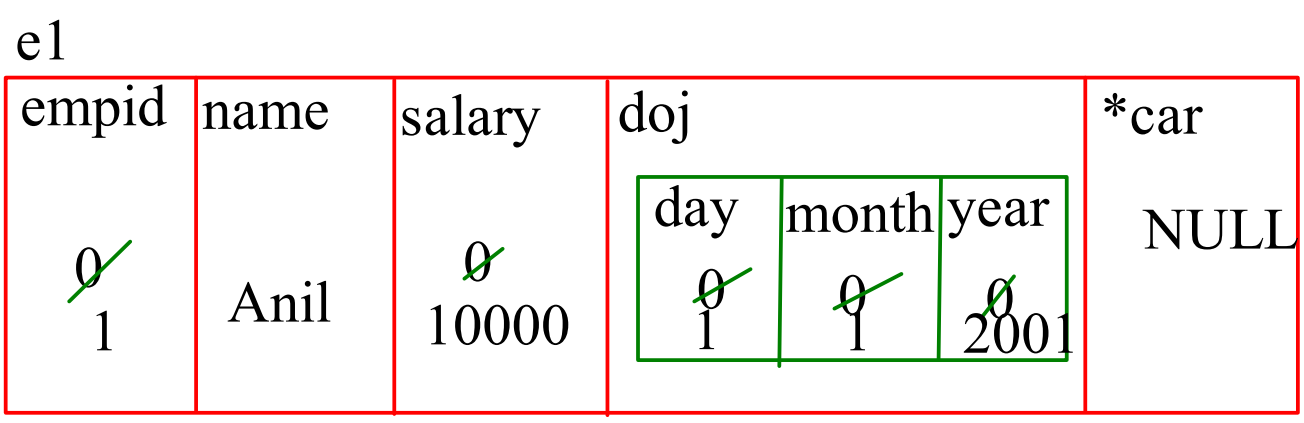
```
// dependent class
Employee{
int id;
double salary;
// dependency
Date doj; // Association - Composition
Car *c; // Association - Aggegration
}
```

```
Doj
Car
```

```
Employee e1;
```







Class,
Object

Time t1;
cout<<&t1;
cout<<sizeof(t1)

Employee **arr = new Employee*[5]

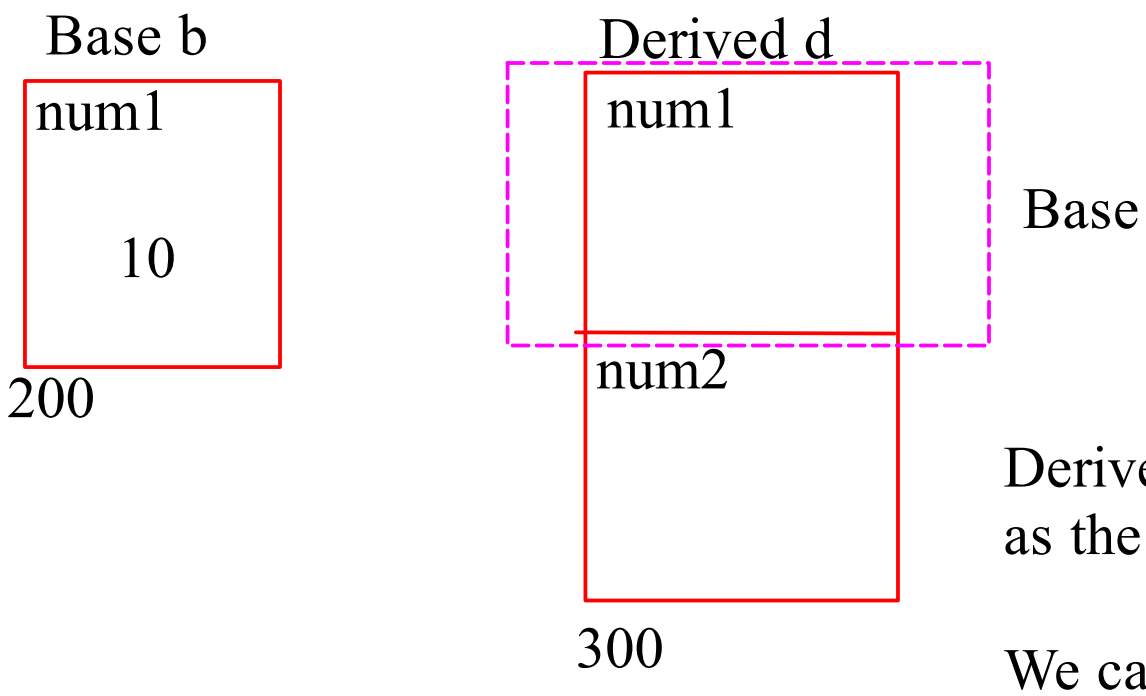
Hirerachy
- Reusability
    - has-a - Association (Composition, Aggegration)

    - is-a - inheritance

```
// Parent/ Base
class Person{

}
// Child/ Derived
Employee : Person {

}
```

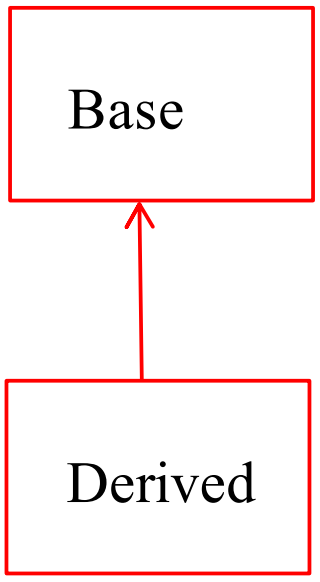
Employee(Child/Derived) is-a Person (Parent/ Base)
Student is-a Person

In Inheritance all the members of the Parent class inherits into the derived class except the below 5.
1. Constructor
2. Destructor
3. Copy Constructor
4. Assignment Operator function
5. Friend Fucntion

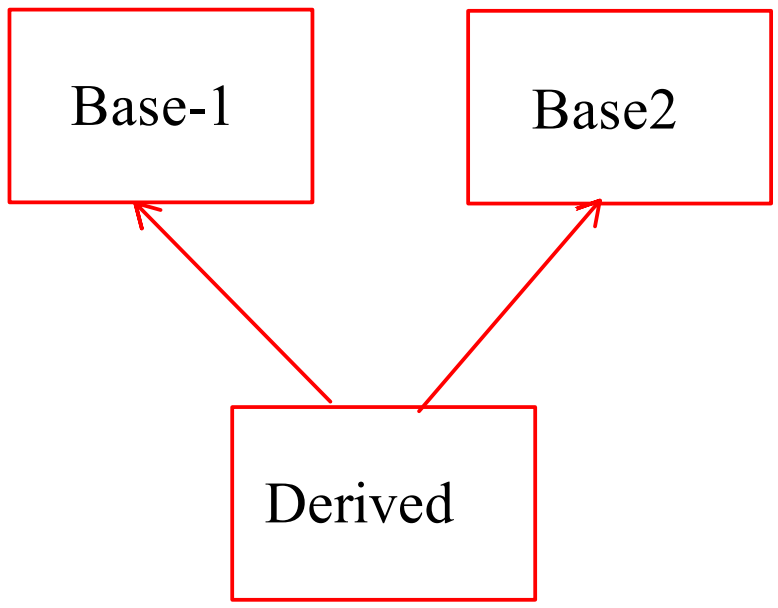


Derived class object is also considerd as the base class object.

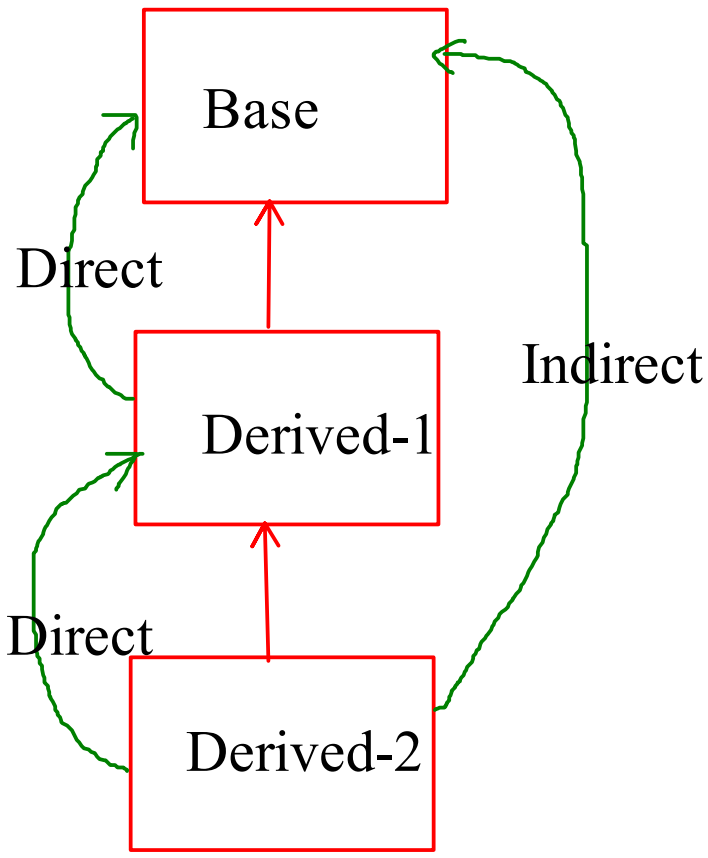
We cannot say that base class object is a derived class object



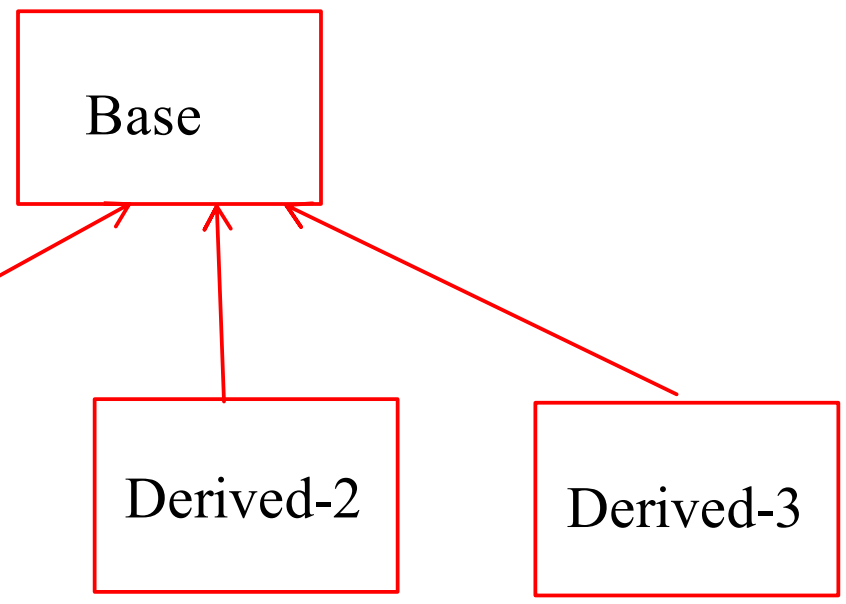
1. Single Inheritance



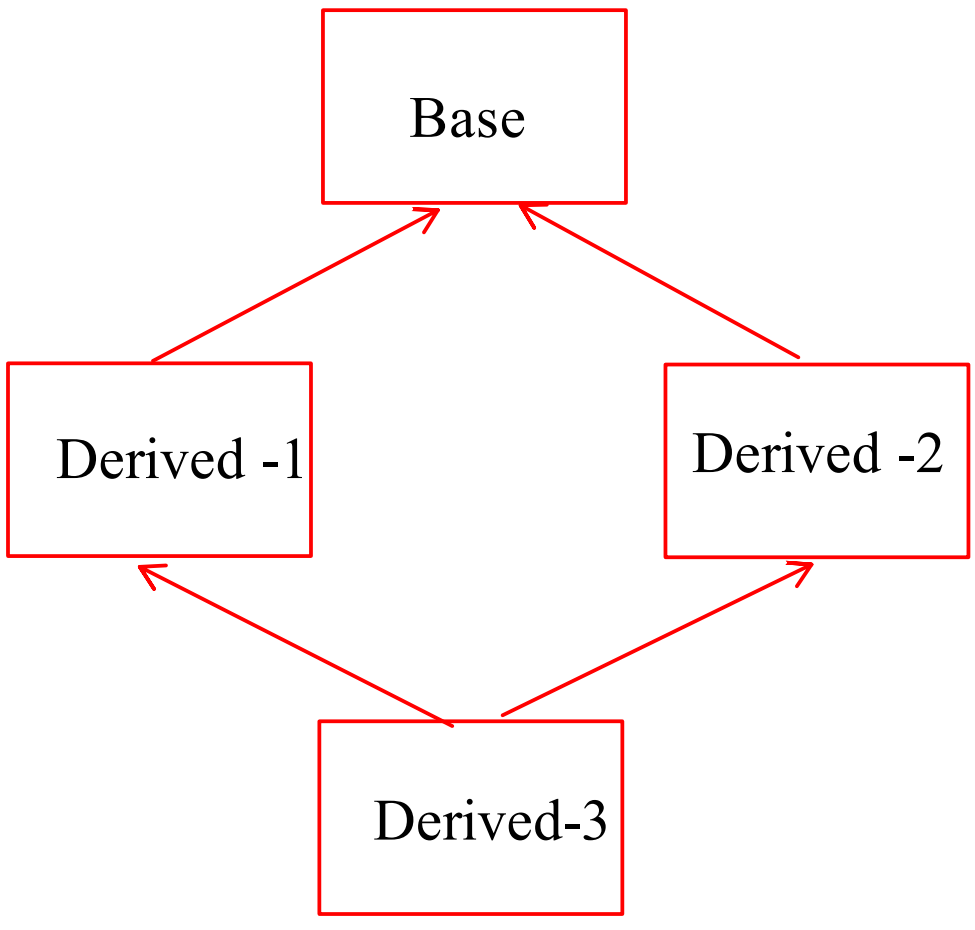
2. Multiple Inheritance



3. Multilevel Inheritance



4. Hirerachical Inheritance



5. Hybrid Inheritance

When hybrid inheritance is performed it causes a diamond problem