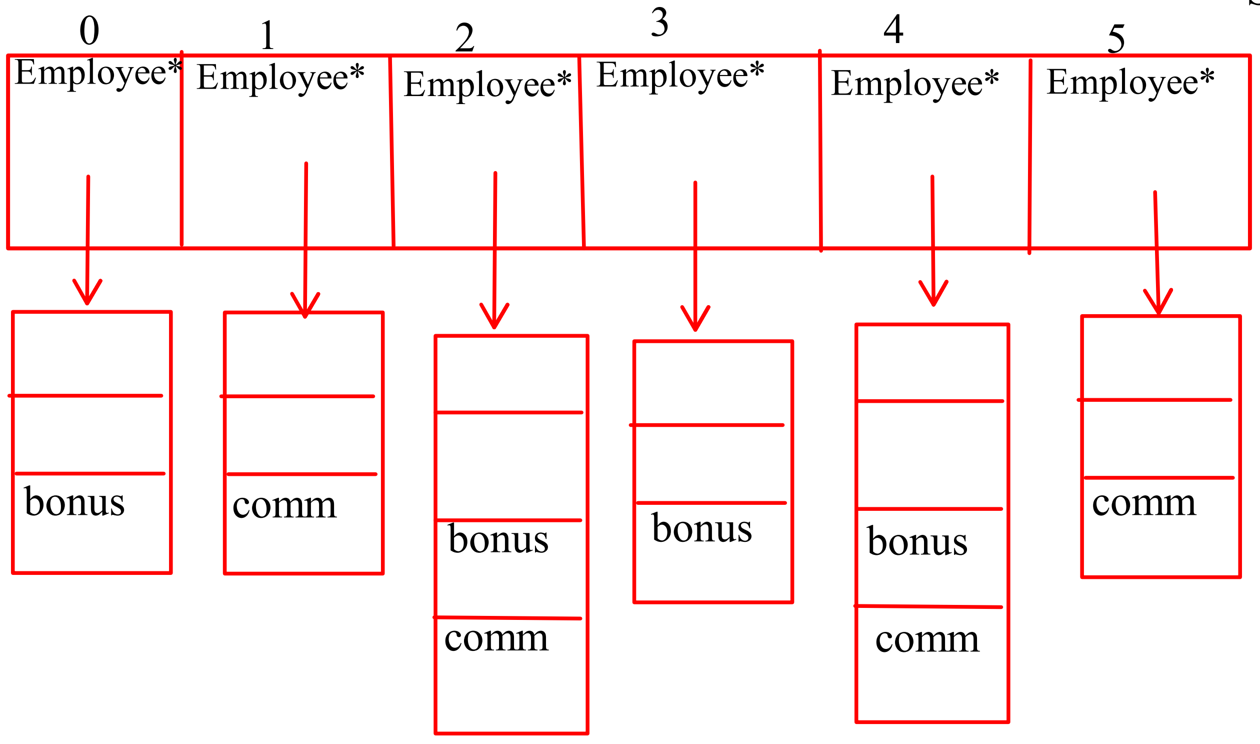
Product
virtual void calculateDiscout()=0;
- Book
- Tape
Product *arr[3];
arr[i] = new Tape()
arr[i]->

Manager
void accept()
void display()

Salesman
void accept()
void display()

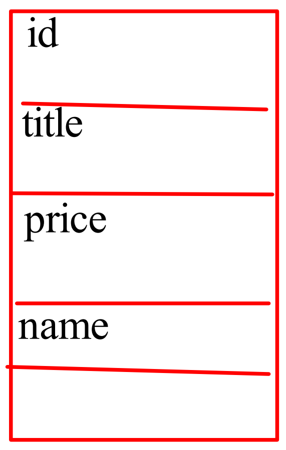
SalesManager
Manager::accept();
Salesman::accept();



Employee* arr[6]

0.Exit
1. manager
2. salesman
3. salesmanager
4. Display managers
5. Display Salesman
6. Display Salesmanager

In case of upcasting to make sure the derived class destructor gets called make the base class destructor as virtual



Book{
author = name
}

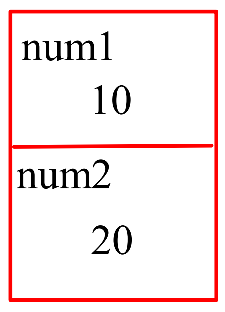
Tape{
artist = name
}

Employee *eptr = new Salesman();

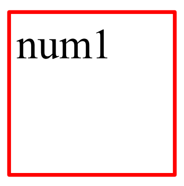
Manger *mptr = (Manager*) eptr;

TypeCasting

int *ptr = (int *)malloc();



Advanced casting operators
1. dynamic_cast
   - When classes are polymorphic
   - When downcasting fails it return NULL
2. static_cast
   - When classes are not polymorphic.
   - But inheritance is required.
3. reinterpret_cast
   - To convert one type into another
   - Most riskest type of conversion
4. const_cast



try
- To check for the exceptions occurring
catch
- To handle the exceptions
throw
- Generate the exceptions

InvalidException{
string message
}

InvalidDateException

InvaliTimeException;

class Date{

}

class Employee {
Date doj;
Date dob;
}

```
int main(){
Employee *arr[10];
int index=0;

a. add Manager
        if(index<10){
        arr[index] = new Manager();
        arr[index]->accept();
        index++;
        }
b. display all salesman
        for(i=0;i<index;i++)
        if(typeid.......salesman)
                arr[i]->display();
}
```

Virtual Destructor

- When upcasting is done, when the object goes out of scope the the destructor of the base class gets called instead of derived class.
- The dtor gets called by looking at the pointer that is created and not on object
- To call the derived class dtor i.e to call the dtor by looking at the object that is created declare the base class dtor as virtual

Advanced Casting Opeartor

1. dynamic_cast
2. static_cast
3. reinterpret_cast
4. const_cast

Exception Handling

-To sperate business Logic form Error Handling Logic
- Keywords
    1. try -> To check the exceptions
    2. catch -> To handle the exceptions
    3. throw -> To generate the exceptions
- Every try should have atleast 1 catch block
- try can have multiple catch block
- Generic catch block can handle all the type of exception,hence it should be provided at the last in the catch block series
- Nested try-catch block

# Template