Object Oriented Programming – 2

Inheritance

```
class Bird {
  private:
        double wt, ht;

        string color;

  public:
        void fly(){
            cout<<"I am flying";
        }
}
```

Encapsulation

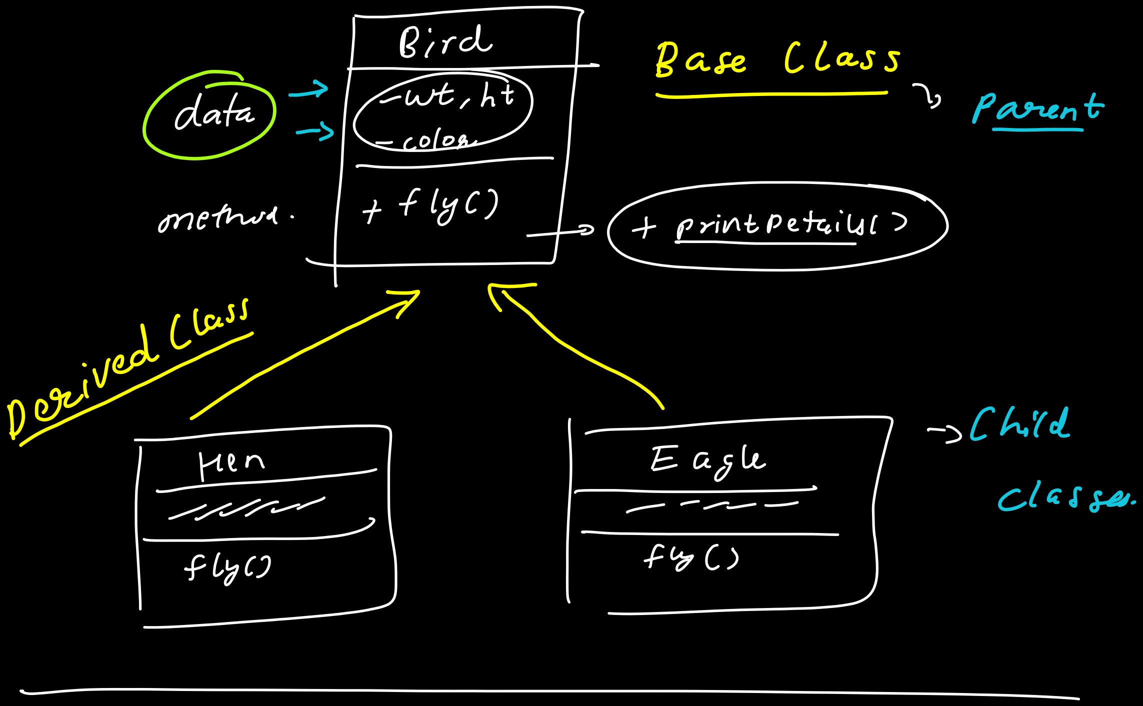
↳ binding methods with data

```
Bird hen;

Bird hen = new
          Bird();

Bird eagle = new
          Bird();
```

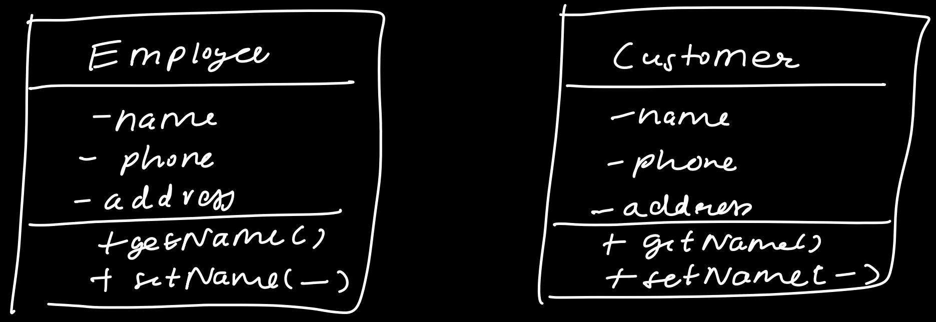
```
class Bird {
  private:
        double wt, ht;

        string color;

  public:
        void fly(){
            cout<<"I am flying";
        }
}
```

✓ Method II :

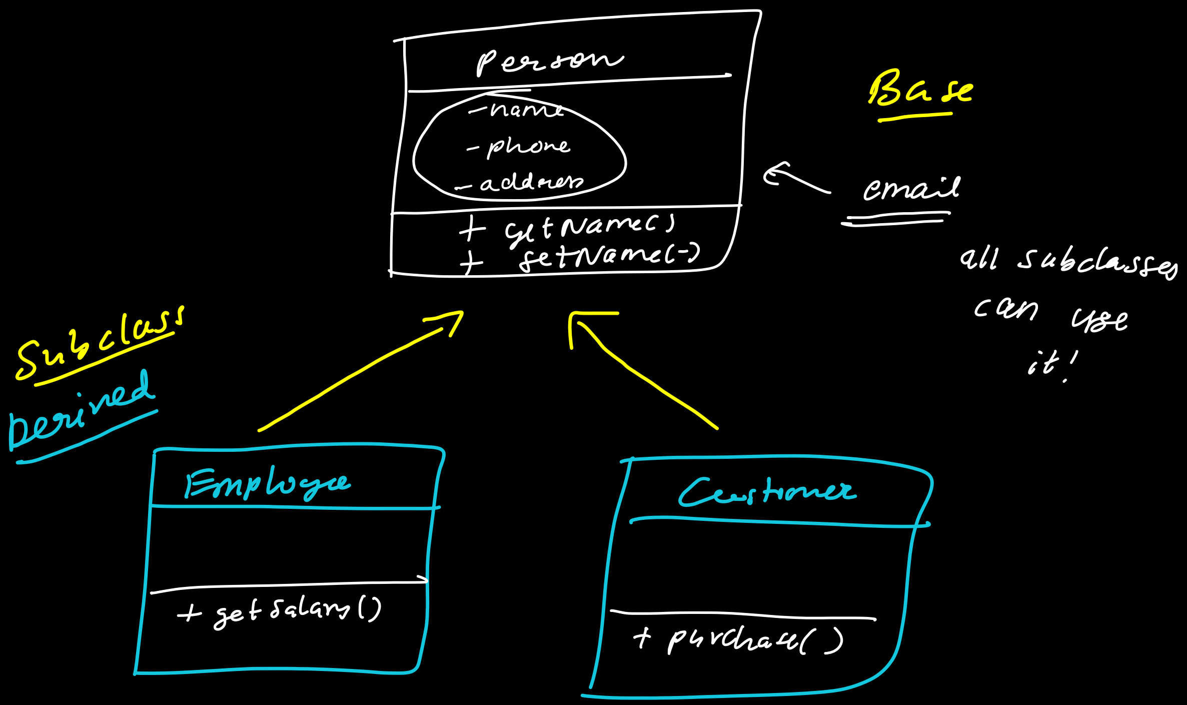
```
void fly (string birdType){
    if ( birdType == "hen")
         //way 1
    else if( — == "eagle")
         //way 2
    //- - -
}
```

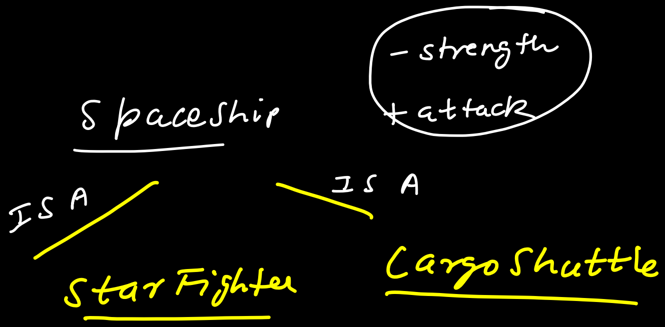

Bird
-wt, ht
-color
+ fly()
data
method.
Base Class
Parent
+ printDetails()
Derived Class
Hen
fly()
Eagle
fly()
Child
classes.

Identify whether we need Inheritance.



shared attributes & methods





$\boxed{\text{IS A}}$ indicator

inheritance.

- → Abstract Classes
- → Interfaces

```
⟹ class Bird {
     private:
         double wt, ht;
         string color;

     public:
         void fly(){
             cout<<"I am flying";
         }
   }
```

method I X

↓ move it to an interface.

Angry Bird Game

Bird
- Hen
- Eagle
- Kiwi

```
Kiwi kiwi1;
kiwi1.fly();
```

Silent Killers

should not fly.

Method II :

override method.

```
void fly(){
    return;
}
```

Interfaces

capability to fly

```
interface Flyable{
    + void fly(){ — };
}
```

Java

```
public class Hen extends Bird
                 implements Flyable {

}
```

Inheritance.

Interface.

Abstraction

data hiding

→ just an abstract idee of stack.

interface Stack {
    public
    void push (int x);
    int getTop ();
    int pop ();
}

- collection of methods that class needs to implement
- Signing a contract / all methods must be implemented.

Space Ship	Asteroid
+ move	+ move
+ draw ()	+ draw ()

```
interface Moveable {
        void move() { - } ;

}
interface Drawable {
        void draw() ;
}

class SpaceShip implements Moveable, Drawable {


}
```

+ Program to an interface, not to an implementation

preferred over inheritance depending on use case!

```
class StackUsingArrays implements Stack {
    private ArrayList<int> a;
    ____________________________________________
    public void push(int x){
        -> add(x);
    }
    ____________________________________________
    private void add(int x){
            a.append(x);
    }
}
```

Abstraction

- data hiding
- impl. hiding.

```
class Bird {
  private:
      double wt, ht;
      string color;

  public:
      void fly(){
          cout << "I am flying";
      }
}
```

abstract function

```
public class Vulture
      extends Bird {

}
```

Vulture v = new Vulture();

v.fly(); Silent Killer

Force all the bird subclasses to implement the fly method!

pure virtual C++

abstract function Java

General term →

abstract void fly();

Any inheriting class must implement their own fly method;

Q) Will we ever need to create an object of Bird class?

→ NO

Abstract Class.

→ must have atleast one abstract method..

→ only for the purpose of inheritance.

→ no object should be created for it.

⇒ Abstract class can have non-abstract methods.