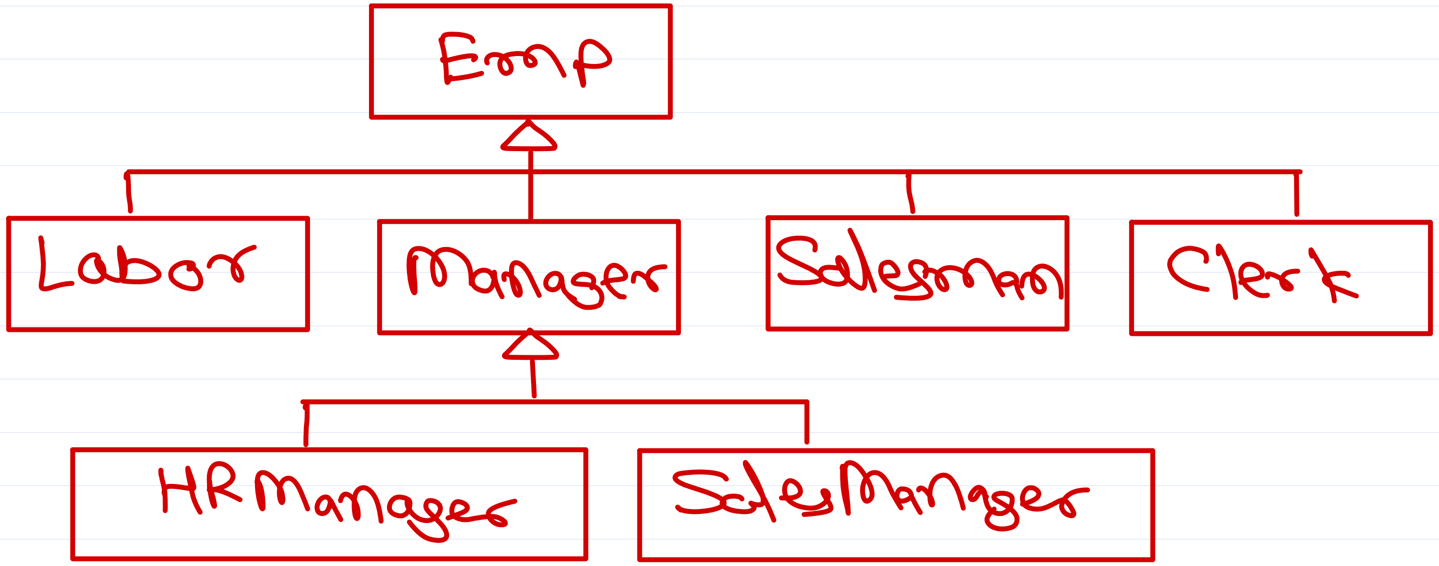
C#

C#.NET @ Sunbeam Infotech

Trainer: Nilesh Ghule

Inheritance



classes
- fields + methods (reuse).
- virtual methods (polymorphism)
- objects

Object Oriented Analysis & Design (OOAD)
- Grady Booch.

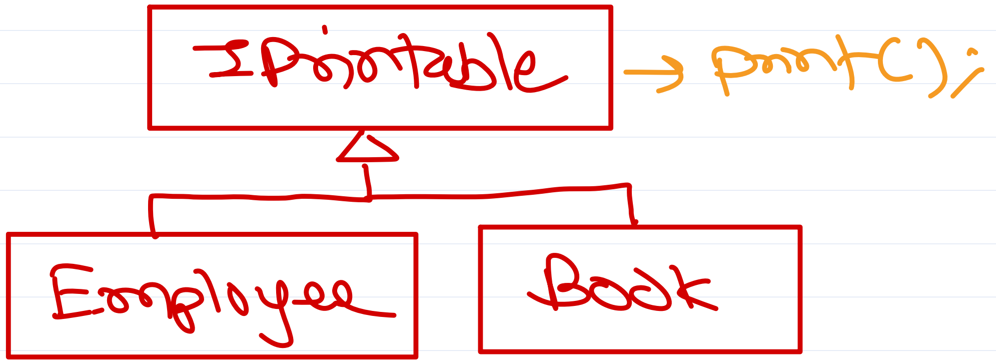
abstract classes
- fields + methods (reuse).
- virtual methods (polymorphism)
- abstract methods (contract)
- conceptual entity (no objects)

* SOLID
* Design Patterns

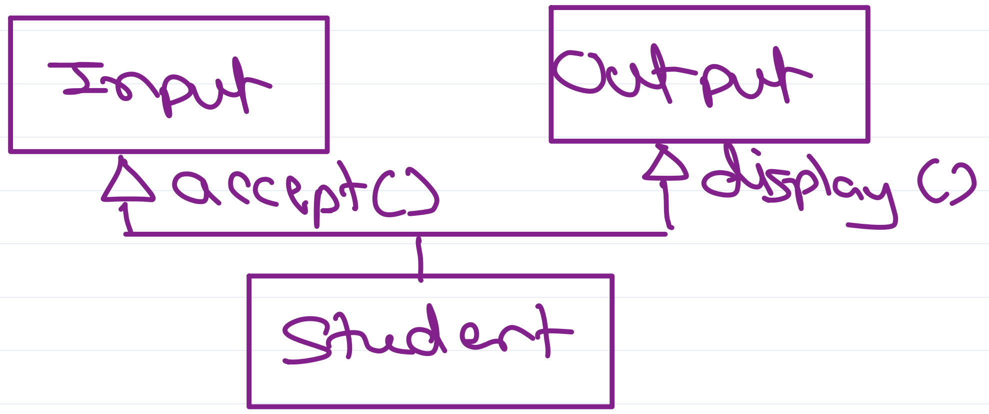
Interfaces

interfaces

- → no reusability
  (no fields, methods,
  no ctors)
- → all abstract methods
  (contract → standard)
  i.e. specification → guarantee
  that derived class has
  the functionality
- → can group unrelated classes
  (Polymorphism)
- → avoids fragile base class
  problem → Immutable
- → no inheritance → implementation



one class may impl
multiple interfaces.



Delegates

delegate → like fn alias (in JS) but typesafe.

object oriented typesafe function pointer.

C → fn pointer →

```
int sum(int a, int b){
    return a+b;
}
```

1000

```
typedef int (*mathop)(int, int);
mathop ptr = sum;
res = ptr(22, 7);
```

$\boxed{1000}$ ptr

→ declare fn ptr type ①

→ declare fn ptr & initialize ②

→ call fn.

Delegates

```
void sum(int a, int b) {
    cw(a+b);
}
```

Step1 → declare delegate (fn ptr) type.

Syntax: delegate ret-type deltypename (params & types);

```
delegate void MathOp(int a, int b);
```

Step2 → create delegate obj & init it (with fn addr).

```
MathOp ptr = new MathOp(Sum);
```

if static ↲ ClassName.Sum

objName.Sum ↲ if non static

↳ if local fn

Step 3 → call the fn

```
ptr(22, 7);
```

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>