Interfaces *

Unit-2

- I/f is an abstract type wh. acts as a blueprint for a class.
- In simple words, i/f is collection of undefined (or abstract) functions.
- I/f r meant to be implemented by class using implements kw
- When a class implements an i/f, it must define all the fun in the i/f.
- It is similar to an abs. class.

---

eg interface myinterface { // Create an i/f.

```
    void set(int x, int y, int z);
    void get();
}
```

- Any class wh. implements this i/f must override // (define) these 2 fun w/ same name & param.
- These fun must be public in the class.

```
class box implements myinterface {
    private int l, w, h;
    // implemented from i/f
public  void set (int x, int y, int z) {
        l = x; w = y; h = z;
    }
public  void get () {
        sopln(l); (w); (h);
    }
// set & get must be o/ridden (defined) by
   box class.
public void vol () {
   // This is not implemented from interface
        sopln( l * w * h);
    }
}
class PSVM (...) {
    // create obj
    box obj = new box();
    obj.set (1, 2, 3);
    obj.get ();
    }
}
```

public → must be public

public

Rules for i/f:

1) fun in i/f must be public or no access specifier
2) when a class defines (overrides) i/f fun then the fun must be public
3) Any class sub. implements an i/f must o/Ride all the fun in the i/f.
4) class can implement i/f using implements kw.
5) obj of an i/f can't be created, ~~cat~~ bcoz it contains undefined fun. However, ref of an i/f can be created. Ref can be used to pt. to a class obj.

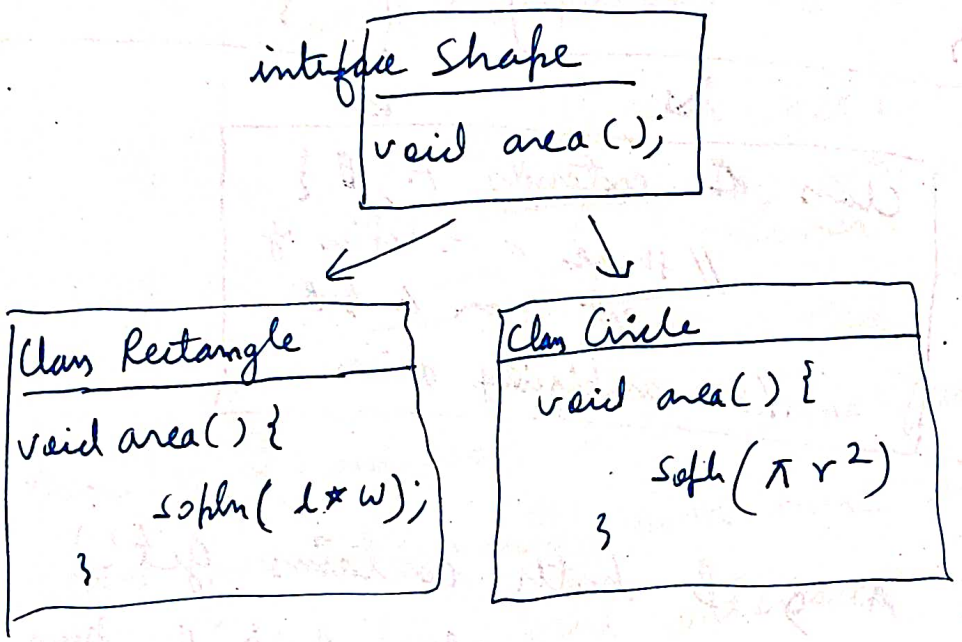
---

what r the uses of an i/f

1) Abstraction
2) Multiple Inheritance
3) Dy. method Resolution (Runtime poly).

1) Abstraction :-

- it provides abstraction, wch means it simply specifies the fun name and param.
- These fun r implemented by the clases in their own way.



- eg // There is an interface Shape.
  It contains a blank/abstract/undefined fun area();
- when rectangle class implements the interface then area is calculated as l*w
- when circle class " " " πr²
- So each class implements/defines/overides the fun differently.

2) Multiple Inheritance :-

- Java doesn't support multiple inh. three classes., bcoz. Mul inh causes ambiguity

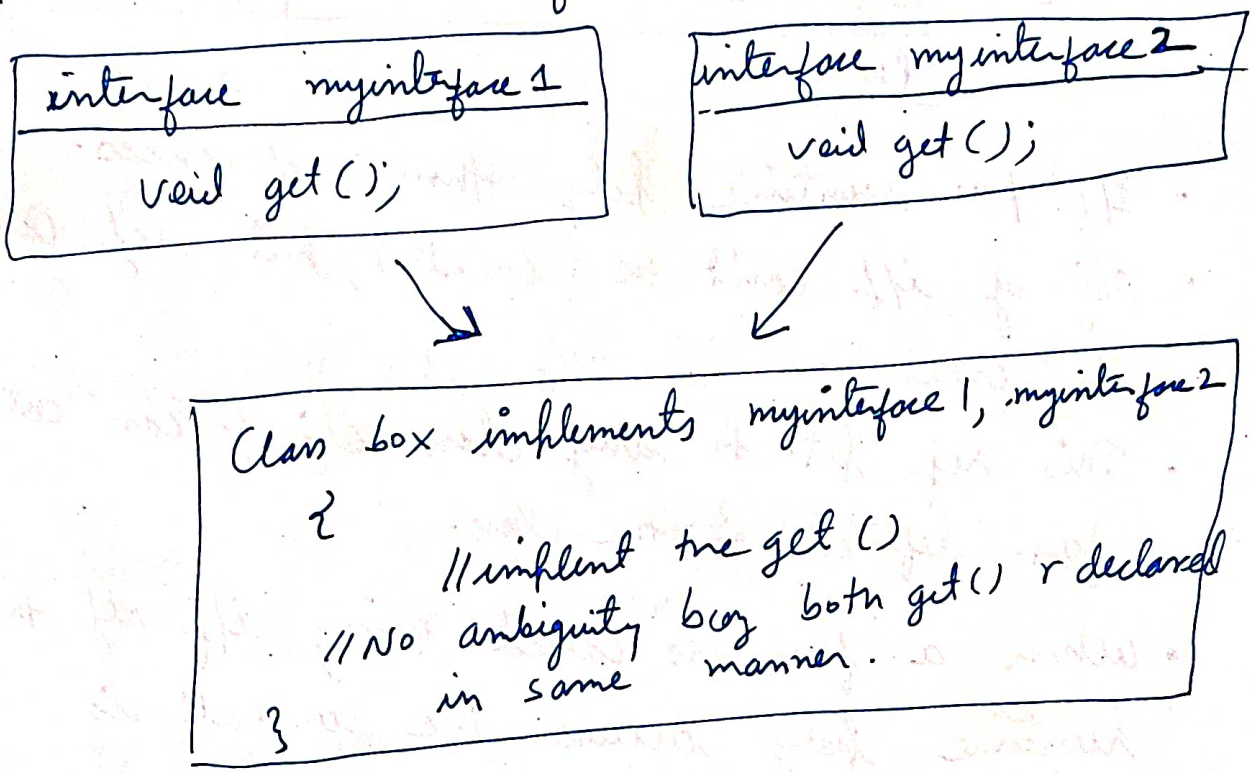
```
Class A {
    public void get();
}
```

```
Class B {
    pub. void get();
}
```

```
Class C extends A, B {
    // there r 2 copies of
       get from A & B
    // so ambiguity arises
}
```

eg
- Class A & B both contains get()
- Suppose Class C inherits A & B, then there will be 2 different copies of get().
- Both get are defined in diffent manner so ambiguity can't be resolved.

Mul inh. thru interface :-



- There r 2 interfaces with same function get().
- Class box implements both these interfaces, so implementing mul interfaces is allowed.
- * Why is there no ambiguity with mul interface:
  - Bcoz interface only provide declaration of fun so there will be 2 copies in box class with same declaration.
  - With mul classes, there will be 2 diffent implementations, so ambiguity arises.
  - With mul. interfaces, implementation will be provided by the class wch. implements these interface.

3- Dy. Method Resolution / Runtime Poly

- I/f b/v runtime poly these references.
- Obj of i/f can't be created, but ref can be created.
- This ref pt. to any class obj & can call fun. defined in the class.
- when a fun is called using i/f ref, then runtime poly occurs. i.e fun call is resolved at runtime dynamically.

```
interface myinterface {
    public void get();
}
```

```
class box implements myinterface {
    //get is defined here.
}
```

```
class student implements myinterface {
    //get is defind.
}
```

box & student class implements i/f

eg "myinterface" is implemented by 2 classes box & student.

- To achieve RT poly, & create a ref of interface & point to class obj.
- ~~Call~~ Then call get() fun using ref.

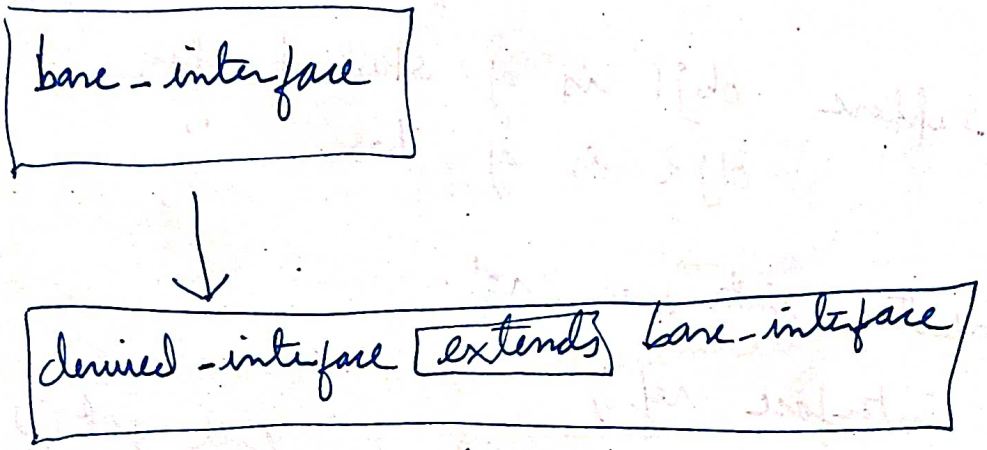
- Suppose obj1 is of student class
  obj2 is of box ''

- Create interface ref.

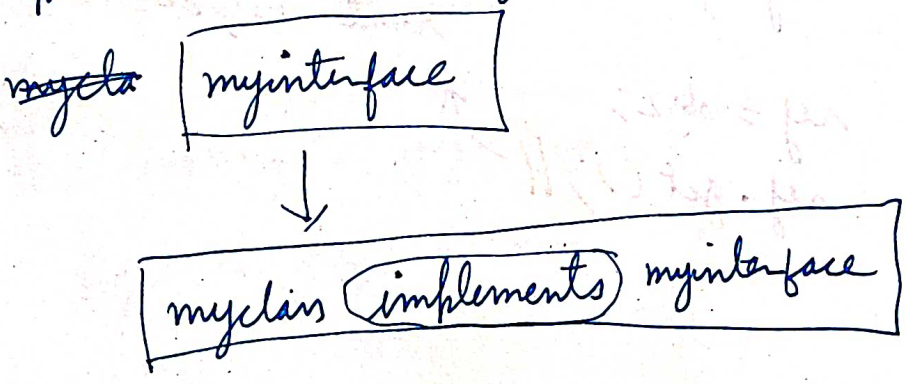
```
interface ref;
// Point ref to objects & call get()
ref = obj1;
ref.get(); // Fun called is resolved at runtime, so RT poly occurs

ref = obj2;
ref.get(); //
```

What r the different ways to use an i/f:

- inheritance can be used w/ i/f
  i.e one i/f can inherit another i/f



- one i/f can't implement another i/f.
- interface can be implemented — one class can implement other i/face.



***

Abstract Class vs Interface

Abstract Class	Interface
1) Ab. class can have abstract (undefined) as well as concrete methods.	1) I/f can have only abstract methods
2) abstract KW is used	2) interface KW is used.
3) Ab. class can't support mul inh.	3) interface can support mul inh ***.
4) ab class can have final, ~~none for~~ non-final, Static & non static var	4) i/f can have only Static & final var.
5) ab class can implement an i/f	5) i/f can't extend/ implement abs class.
6) ab. class can extend another class	6) i/f can be implemented w/ implements KW.
7) members can be private, protected etc	7) members r public by default.