Interface

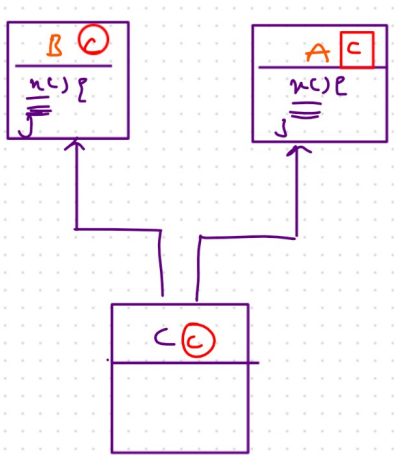
Def: Interface is a collection of abstract methods

```
interface X{
    void d();
}

interface MyGame{
    void a();
    void b();
}

interface MySport extends MyGame,X{
    void c();
}

abstract class Part1 implements MyGame,X{
    public void d() {
        System.out.println("Hello");
    }
}
```

Difference b/w multiple Inheritance & Interface

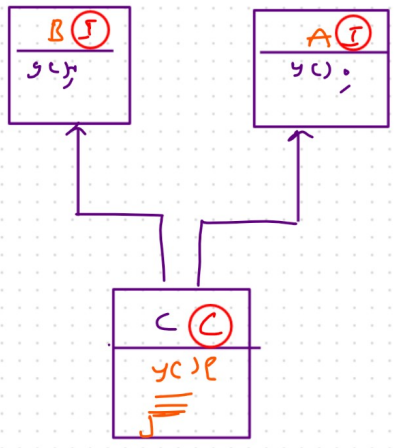
(failed)

classes



C obj = new C();

obj.x() // confusion



C obj = new C();

obj.y(); // It calls overriden method y() i.e in class C

Rules:

→ In order to declare an interface, we can declare it by interface (keyword) followed by interface name

eg: interface myGame { }

→ A class doesn't extends interface, rather it "implements" the interface

eg: interface X {
    void a();
}

class Driver implements X {

}

→ An interface always extends another interface

```
Interface A {
    void a1();
}

Interface B extends A {
    void b();
}
```

→ All the methods in interface is abstract { we cannot provide implementation for it.

```
Interface A {
    void play();
    void sleep();
}
```

→ When a class implements an interface it is compulsory to give the body for the unimplemented methods

(or)

we have to make the class as abstract

```
Interface A {
    void play();
    void sleep();
}
```

(1) class Launch implements A {

option

```
    void play() {
        S.o.p("playing");
    }
    void sleep() {
        S.o.p("sleeping");
    }
```

option (2)

```
Abstract class Launch
    implements A {

}
```

→ we cannot create an object of interface, it throws error

[Cannot instantiate the type X]

→ We can create a reference of interface

```
interface Game {
    void a();
}

class Module1 implements Game {
    void a() {
        sop("Hello");
    }
}

class Launch {
    Game myGame = new Module1();
}
```

→ A class can implement multiple interfaces } encompo refer above code

→ A interface can extend multiple interfaces

→ A class can extend another class & also implements multiple interfaces at the same time // refer above code

```
class Part2 extends Part1 implements MyGame {
    public void a() {
        System.out.println("");
    }
    public void b() {
        System.out.println("");
    }
}
```

→ the methods present inside the interface is public & abstract by default

```
void play();
public void play();
abstract void play();
public abstract void play();
```

} ✓

→ we can't have these modifiers in the methods of interface

public abstract void play(); ✓

protected	final
private	static

✗

→ We can declare a variable inside the interface by assigning value to it (compulsory)

ex:

```
interface X {
    int x = 10;
    abstract public void x();
}
```

→ The variables inside the interface are by default

(1) public
(2) final
(3) static

int n = 10 ⟶ public static final int x = 10;

(by default)

→ If a class is implementing from two different interfaces & if those interfaces has methods with the same name & signature, then the class gives a common implementation for both the methods

ex:

```
interface X {
    void play();
}

interface Y {
    void play();
}

class Parent implements X, Y {
    void play() {
        // No problem it is allowed
    }
}
```

→ If a class implements multiple interfaces & if those interfaces have methods with same name & different signature, Then we have to provide multiple implementations i.e [both method overloading & method overriding]

eg:

```
interface X {
    void play();
}

interface Y {
    void play(int a);
}

class Parent implements X, Y {
    void play {
        // overriding
    }
    void play(int a) {   // overloading
    }
}
```

→ If we have multiple interfaces, having the methods with same name, signature & different return types, then we cannot implement there interfaces to a single at the same time.

```
interface X {
    int play();
}

interface Y {
    byte play();
}

class Alpha implements X, Y {
    int play() {   // shows error incompatible return type for Y.play()
    }
    byte play() {   // show error incompatible return type for X.play()
    }
}
```

we cannot given common implementation

// so the only solution is in this case implement only one interface (either X or Y) to the class

```
class Alpha implements X { }
```

→ we cannot have constructors inside the interface & any other blocks also not allowed

→ If a class is implementing two interfaces & if those interfaces having variables with the same name, Then the class cannot access directly variable, because there is a ambiguous situation,

→ to overcome this the variables can be accessed with the help of interface name.

eg:

```
interface X {
    int a = 10;
}

interface Y {
    int a = 20;
}

class Alpha implements X, Y {
    p.s.v.m(String args[]){
        S.o.p(a); // field ambiguous
        S.o.p(X.a);
        S.o.p(Y.a);
    }
}
```

Note [possibilities]

→ (C) A implements (I) B

→ (C) A implements (I) B, (I) C, (I) D

→ A extends B — A, B are classes either A, B are interface

→ (C) A extends (C) B implements (I) C

→ (I) A extends (I) B, (I) C, (I) D

When we should we go for class, ~~Abstract~~ Class & Interface.

Concrete class : When we know the complete implementation

Abstract Class : when we know partial implementation

Interface : When we have no idea about implementation / Zero knowledge