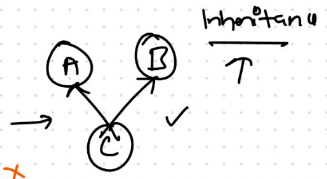
Interface

def: It is a collection of abstract methods

```
method (); {
         ammes
}
```

```
~~class~~ {
   main() {
   "Hello world"
   }
}
```

RPG



multiple
class ✗
interface ✓

eg:

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

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

intuface   MySport  extends  X, MyGame {

      void c();

}
```

R{ }7 , S DS
50 feature
3 hr → part I

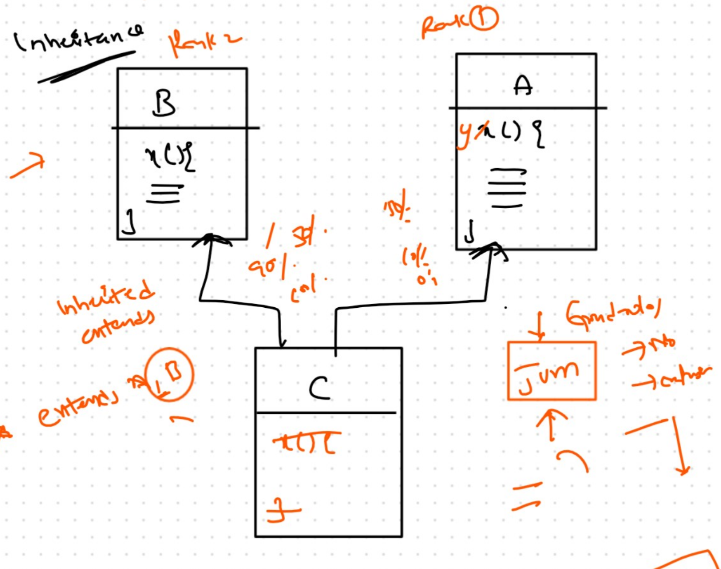
```
abstract  class  Part1  implements  MyGame, X {   ✓
             void a() {
                S.O.P ("Hello player");
             }
}
```

30%
new Part()

100%
a(){ } | b(){ } | d(){ }

MRO

difference b/w multiple Inheritance & Interface



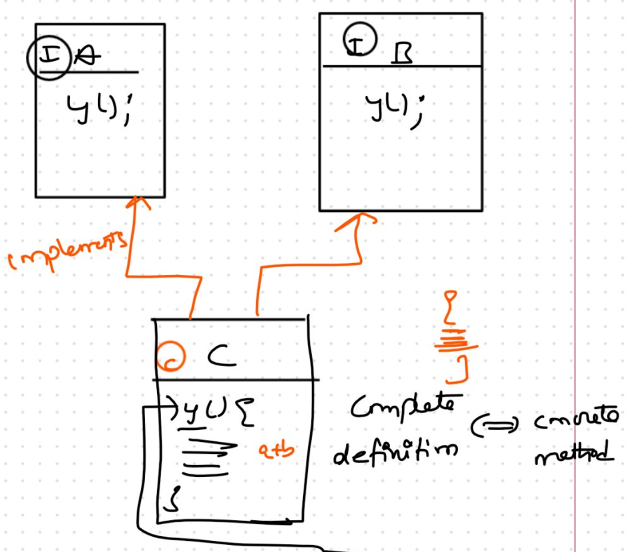
C obj = new C();

obj.x() // confused

Avoid ⇒ ambiguous situation

Compiler Design
↓
Java Language

Interface



C obj = new C()

→ obj.y()

Rules :

→ Inorder to declare interface, we can declare it by keyword "interface" followed by interface name

```
interface myGame { }
```

→ A class doesnot extends interface, rather it "implements" the interface

ex:

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

class Driver implements X {

}
```

→ An interface always extends another interface

```
interface A {              interface B extends A {
                               ~~void a();~~
    void a();                  void b();
                           }
}
```

→ All the methods in interface is abstract & we cannot provide implementation (body) for it

```
interface A {
    void a() {
    ====          ×  //Not allowed
    }
}
```

→ 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 {

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

②

```
abstract class Launch
        implements A {

}
```

→ We cannot create an object of interface, it throws error called

[Cannot Instantiate the type X]

→ We can create a reference of interface   interface X { }

```
interface Game {          |   class Modules implements Game {
                          |       void a() {
    void a();             |           S.O.P("Hello");
                          |       }
}                         |   }
```

P — C

p ← low capacity

C = new C()

```
class Launch {
psvm main() {
    Game myGame = new Modules();
     P                 Child
}  }
   Modules
     X
```

→ { A class can implement multiple interfaces (but not multiple classes)
→ { An interface can extend multiple interfaces

eg:

```
interface X {                 interface Y {

    void play();                  void sleep();

}                             }
```

```
                                          Z → [X, Y]
interface (Z) extends X, Y {   |  class Launch implements ~~X, Y~~ {
                               |      void play() {
    void dance();              |          ==
                               |      }
}   void play()                |      void sleep() {
    void sleep()               |          ==
                               |      }  void dance() {
         ↑                     |         }
         I                     |  }
```

→ A class can extend another class & also implements multiple interfaces at the same time

eg:

```
class Parent {        |  Syntax  class child extends Parent implements X, Y {
                      |          void play() {
                      |              ==
}                     |          }
                      |          void sleep() {
                      |
                      |          }
```

→ the methods present inside the interface is public & abstract by default (i.e we declare it | not)

```
interface X {
        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();
```

$\boxed{\begin{matrix}\text{protected} & \text{final}\\ \text{private} & \text{static}\end{matrix}}$ ✗

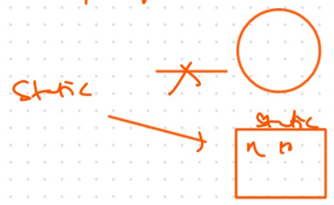
```
protected abstract void play()           ✗
public abstract final void play()        ✗
private abstract void play()             ✗
public abstract static void play()       ✗
```

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

compulsory

eg

```
interface X {
    int n;   ✗
    int n =10;
    n();
}
```



→ The variables inside the interface are public, final & static by default

eg:

```
int n = 10;
static int n =10;
final int n =10 ;
public int n =10 ;
static final int n =10 ;
final public int n = 10 ;
public static final int n = 10;
```