Interfaces In Java

Interface mainly used in order to achieve polymorphism
→ interface does not need to borrow from parent
→ Borrowing mean inheriting
→ Abstract class used for inheriting & achiving Polymorphism
→ The interface is abstract class which only has abstract methods & don't have any kind of concrete methods

```
abstract class A{
abstract void hell1();
abstract void hell2();
}
```

⇒

```
interface A{
void hell1();
void hell2();
}
```

so above abstract class don't have anything thar abstract methods. so we can use interfaces over using abstract class

interfaces don't need keyword abstract as it by default abstract methods.

→ abstract classes is class And interface is not a class so you can declare

```
interface name{
returntype function();
}
```

To take the function in a class we Inherit for inherting methods from interfaces we use keyword "implements"

Such like that one class is can be created from Interface ⇒

```
class T2 implements name{
}
```

⇒ classes are extended.
⇒ interfaces are implemented.

→ for interface also you can't create object (that is obvious as interfaces are not ~~objects~~ class) but you can use it as reference and as dynamic method dispatch

code to code difference betn interfaces & abstract class

Interfaces don't have concrete class

```
abstract class One{
    abstract public void meth1();
    abstract public void meth2();
    public void meth3(){
        S.O.P("Hello world");
    }
}
```

```
interface One{
    void meth1();
    void meth2();
}
//Interfaces don't have
concrete functions
as it does not used for
~~behavior~~ borrow
```

In subclass we implement function from interfaces

```
class hell extends One{
    public void meth1(){
        ==|code|==;
    }
    public void meth2(){
        ==|code|==;
    }
}
```

```
class hell implements One{
    Public void meth1(){
        =|code|=;
    }
    public void meth2(){
        =|code|=;
    }
}
```

object:

you can't create for both abstract classes & interface object but can create refence using DMD

```
One n = new hell();
```

```
One n = new hell();
```

with this we can execute overrided function in subclass & only function which present in reference.

→ functions in child/sub class can't be accessed with such references (abstract class or interface references)

→ Java does not support multiple inheritance that is a child can't have multiple parent

⇒ But a class can implements from multiple Parent interface

Ex:

```
class A {};
class B
class C extends A,B {
}
```

this is not possible in Java but can be possible for Interface

```
interface A{}
interface B{}
class C implements A,B {
}
```

this is possible in Java

→ So with help of interfaces java can attempt to create multiple inheritance

→ there is no need to add access specifier in interface for function you can do it is it's child class

→ interface can act like non-class Parent.

---

a single can have a abstract class and interface both

```
class A{}
interface B{}
class C extends A implements B{
}
```

such case also can be possible

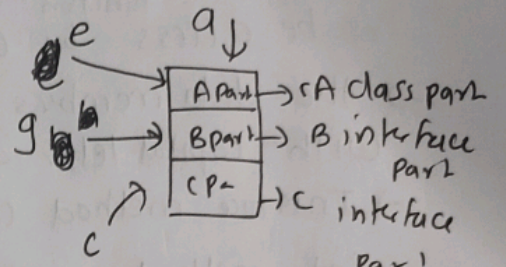
so it is not compulsion to create object for each ref-

so

```
class A{}
interface B{}
interface C{}
```

```
class z extends A implements B,C
{
}
```

```
so    z a = new z();
      A e = a;
so    B g = a;
      C h = a;
```

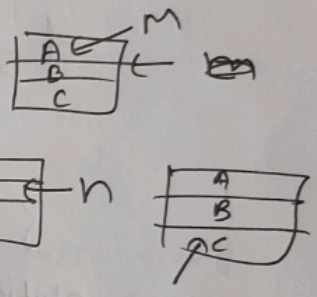


---

so for a same object each reference will ~~with~~ be point to it's source zone in that object

when you do

```
A m = new z();
B n = new z();
C p = new z();
```



So each time object is create to fetch single object always create object one & assign reference to that object only instead of creating new object each time

---

for a single object you can use multiple type of references to that that is beauty of <u>oop</u>

#Callback functions

d d d.

Does & Don't in interface

→ In interface functions are by default public & abstract

→ Interface ~~class~~ can't be ~~p~~ private as they need to be access function by child class

→ the data members inside interface ~~f~~ should start with capital letter & they are static & final by default

→ Interface method cant have body

~~int~~ methods with static can have body in Int

```
interface a {
    static void meth() {
        s.o.p ("blah blah");
    }
}
```

an can be accessed by

```
a.meth();
```

interface name

→ static function can have body in interface

→ Interface can be extended

→ Ex:

```
interface A {
    void f1();
}
interface B extends A {
    void f2();
}
```

```
class C ~~imp~~ implements A,B {
    ~~s~~ // ——
}
```

↑ so class C should override all functions of A + all functions of B

default methods can also have body in interface

```
default void fun(){
    $op("---");
}
```

and by using object you can access m

→ so if you want access method inside interface with interface name declare it with as static & if want to access throught object declare it with default

→ you can define private method in interface but they should not be abstract

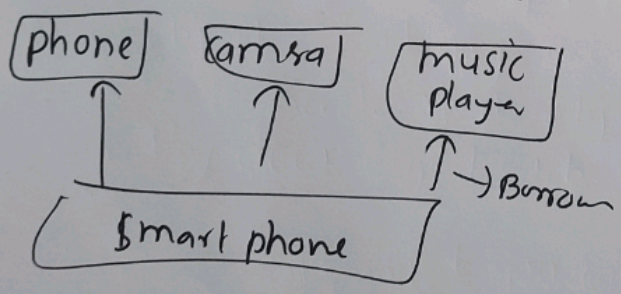
→ they can act like internal mechanism in default classes

---

multiple inheritance vs interface

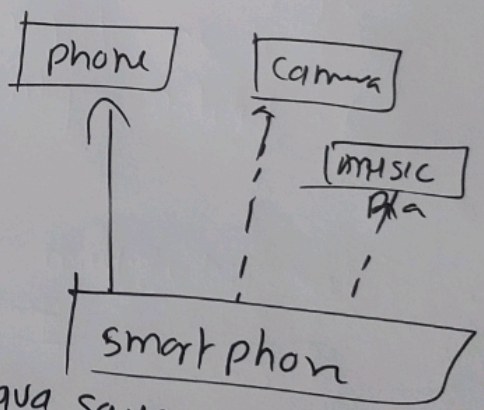
→ Java don't have multiple inheritance

One class created from multiple parent class & this is multiple inheritance



cpp says

all phone camera music class from a smartphone



Java says the smartphone is mainly consist a class phone & also has some features from interface camera & music

Java has most accurate approach toward oop