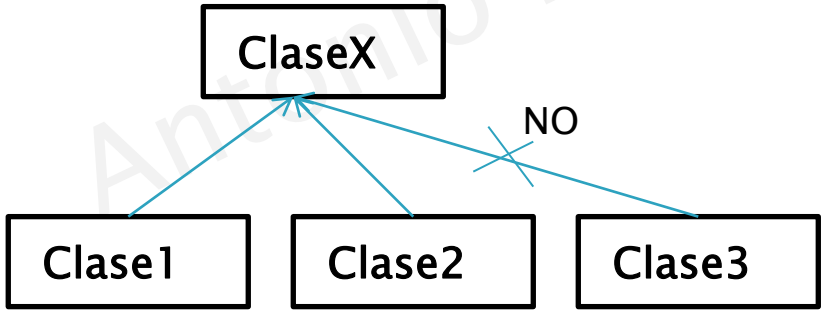
Sealed clases

sealed class

- Clases que pueden ser heredadas solo por ciertas clases.
- Más permisivo que final, menos que el comportamiento por defecto.

```
public sealed class ClaseX permits Clase1,Clase2{
   …
}
```



Las subclases deben ser declaradas como non-sealed, sealed o final

```
public non-sealed Clase1 extends ClaseX{
   …
}
public final Clase2 extends ClaseX{
   …
}
//error de compilación
public Clase3 extends ClaseX{
   …
}
//error de compilación
public class Clase2 extends ClaseX{

}
```

sealed class. Consideraciones

- Las clases implicadas deben formar parte del mismo módulo o, en caso de ser módulos anónimos, deben estar en el mismo paquete.
- Cada clase permitida, debe extender directamente la sealed class.
- Las subclases deben ser definidas obligatoriamente con alguno de estos modificadores: *non-sealed*, *sealed* o *final*
- Si una subclase permitida se define como *non-sealed*, todas las subclases de esta tendrían acceso a la clase base:

```
sealed class Padre permits Hija1{
   public void m() {
      System.out.println("m de base");
   }
}

non-sealed class Hija1 extends Padre{}

class Nieta1 extends Hija1{
   public void test() {
      super.m();
   }
}
```

sealed interface

- Las interfaces también pueden ser *sealed*, permitiendo la herencia e implementación solo a ciertas interfaces y clases:

```
public sealed interface IPadre permits IHija1,Prueba{

}
non-sealed interface IHija1 extends IPadre{

}
non-sealed class Prueba implements IPadre{

}
```

- En el caso de una interfaz permitida, esta sólo podrá ser declarada como *non-sealed* o *sealed*, nunca *final*.

clausula permits opcional

- Si la clase/interfaz sealed están en el mismo fichero que las permitidas, puede omitirse la clausula *permits*:

MyClass.java

```
public sealed MyClass {}
final class Other extends MyClass{}
```

- También si se trata de una clase/interfaz anidada:

```
public sealed interface MyInter{
        non-sealed class C1 implements MyInter{}
}
```

Revisión conceptos

Dada la siguiente definición de interfaz:

```
public sealed interface MyInter permits C1, C2, I1{}
```

Indica cual de las siguientes definiciones de clases/interfaces es correcta, suponiendo que se encuentran en el mismo paquete que MyInter

a. class C1 implements MyInter{}
b. final class C2 implements MyInter{}
c. sealed class C3 implements MyInter{}
d. final interface I1 extends MyInter{}

Respuesta

La respuesta es la **b**. La a no es correcta porque la clase C1 debe llevar alguno de los modificadores sealed, non-sealed o final. La b es correcta porque si cumple con la regla anterior. La c es incorrecta porque si C3 es sealed debería indicar la lista de subclases. La d es incorrecta porque una interfaz no puede ser nunca final.

Revisión conceptos

Indica que ocurrirá al ejecutar el siguiente código:

```
sealed class ClaseA extends ArrayList  permits ClaseB{ //linea 1
      void test() {System.out.println("ClaseA");}
}
non-sealed class ClaseB extends ClaseA{}

class ClaseC extends ClaseB{//linea 2
        void test() {System.out.println("ClaseC");}
}

public class Test{
       public static void main(String[] args) {
            ClaseA ca=new ClaseC(); //linea 3
            ca.test();
       }
}
```

a. error de compilación en línea 1
b. error de compilación en línea 3
c. Se mostrará "ClaseA"
d. Se mostrará "ClaseC"

Respuesta

La respuesta es la **d**. Es posible extender una clase y permitir a otras. Dado que puede heredar ClaseB por ser non-sealed, ClaseC es también un ClaseA