

Novedades en interfaces



Métodos default

- Proporciona una implementación por defecto, que puede ser utilizada por las clases que implementan la interfaz.
- Se definen con la palabra reservada *default*.

```
public interface Operaciones{  
    default void girar(int grados){  
        System.out.println("gira "+grados+" grados);  
    }  
    int invertir();  
}  
:  
public class Test implements Operaciones{  
    //solo tiene que implementar el abstracto  
    //aunque, si se quiere, se puede sobrescribir  
    //también el default  
    public int invertir(){  
        :  
    }  
}
```



```
public class Prueba{  
    public static void main(String[] args){  
        Test ts=new Test();  
        //utiliza la implementación por defecto  
        ts.girar(30); //muestra gira 30 grados  
    }  
}
```

Problema de herencia múltiple

- Si una clase implementa dos interfaces con el mismo método default, está obligada a sobrescribirlo.

```
interface InterA{
    default void m(){
        System.out.println("default InterA");
    }
}
interface InterB{
    default void m(){
        System.out.println("default InterB");
    }
}

class Test implements InterA,InterB{
    //si no se sobrescribiese, error de compilación
    public void m(){
        System.out.println("Implementación Test");
    }
}
```



```
public class Prueba{
    public static void main(String[] args){
        Test ts=new Test();
        //utiliza la implementación de la clase Test
        ts.m(); //muestra Implementación Test
    }
}
```

Métodos estáticos

- Desde Java 8, las interfaces pueden incluir métodos estáticos al igual que las clases.
- El método está asociado a la interfaz, no es heredado por las clases que la implementan.

```
interface InterA{  
    static void m(){  
        System.out.println("estático InterA");  
    }  
}  
public class Test implements InterA{  
  
}
```



```
public class Prueba{  
    public static void main(String[] args){  
        Test ts=new Test();  
        ts.m(); //error de compilación  
        Test.m(); //error de compilación  
        InterA.m(); //correcto, muestra estático InterA  
    }  
}
```

Métodos privados

➤ A partir de la versión Java 9 se pueden incluir métodos privados en las interfaces. Son utilizados desde métodos *default*

```
interface Inter1{  
    //uso interno en la interfaz  
    private int mayor(int a, int b){  
        return (a>b)?a:b;  
    }  
    private int menor(int a, int b){  
        return (a<b)?a:b;  
    }  
    default int suma(int a, int b){  
        int s=0;  
        //llamada a métodos privados  
        for(int i=menor(a,b);i<mayor(a,b);i++){  
            s+=i;  
        }  
        return s;  
    }  
}
```



```
public class Prueba{  
    public static void main(String[] args){  
        ClasePrueba cp=new ClasePrueba();  
        System.out.println("suma "+cp.suma(10, 5));  
    }  
}
```

Los métodos privados también pueden ser estáticos, pudiendo ser llamados desde otros métodos estáticos o default de la interfaz

Revisión conceptos


Indica porqué motivos la siguiente interfaz no compilaría

```
public interface MyInterfaz{  
    protected void m();  
    default int m1(){return 10;}  
    static void m2();  
    abstract void m3(){}  
}
```

Respuesta

No compila porque los métodos m, m2 y m3 están incorrectamente definidos

Interfaces funcionales

- Concepto introducido en Java 8 para denominar a las interfaces que *disponen de un único método abstracto*.
 - Se pueden crear implementaciones de estas interfaces a través de expresiones lambda (se estudian más adelante).
 - Pueden, opcionalmente, estar definidas con la anotación `@FunctionalInterface`
- 

Ejemplos

Funcionales

```
interface InterA{
    default void m(){
        System.out.println("default InterA");
    }
    int metodo();
}
interface InterB extends InterA{
    static void print(){
        System.out.println("static InterA");
    }
}
@FunctionalInterface
interface InterC{
    void m();
    String toString(); //los métodos abstractos que
                        //coincidan con algún método de
                        //Object no se tiene en cuenta de
                        //cara a la característica de ser funcional
}
```

No funcionales

```
interface InterD extends InterA{
    int metodo(int p); //no puede haber dos abstractos, aunque sea
                        //sobrecarga
}
interface InterE extends InterA{
    //implementa metodo, luego ya no tiene métodos abstractos
    default int metodo(){
        return 10;
    }
}
```


Revisión conceptos



Indica si la interfaz I2 es o no funcional

```
public interface I1{  
    default int m1(){return 10;}  
    static void m2(){}  
    boolean equals(Object ob);  
}  
public interface I2 extends I1{  
    int print();  
}
```

Respuesta

- Si es funcional, ya que, aunque tiene dos métodos abstractos (print y equals), el método equals coincide con uno de los métodos de Object, por lo que técnicamente solo cuenta con un método abstracto.