CLASES ABSTRACTAS
E INTERFACES

CLASES ABSTRACT

- Clase definida como **abstract**.
- No se pueden crear instancias de una clase abstracta.
- Puede tener **métodos abstractos**, **no abstractos** y **atributos**.
- Contendrá el comportamiento general que deben tener todas las subclases, las cuales implementarán todos los métodos abstractos heredados.

```
[modificador] abstract class NombreClase {
      ...
}
```


MÉTODOS ABSTRACT

- Deben estar en una clase definida como **abstract**.
- Definen la firma del método, pero sin implementación.
- Sus subclases se comprometen a implementarlo.
- Si no lo hacen, también **deben ser abstractas**.
- Pueden convivir con métodos normales.

```
public abstract class Abstracta {

        public abstract void metodoAbstracto(String
        s);

        public void saludar() {
                System.out.println("Hola mundo!!!");
        }
}
```


INTERFACES

- Creadas con la intención de ser clases 100% abstractas.
- Colección de constantes y métodos **abstractos** (hasta Java 8)
- Desde Java 8, pueden incluir también métodos con cuerpo estáticos y por defecto (**static** y **default**).
- Desde Java 9, pueden incluir también métodos privados
- Contrato de compromiso: Conjunto de operaciones que una clase se compromete a implementar.
- La interfaz marca **qué** hay que hacer, no **cómo**.

```
public interface Nombre {
        // constantes son public, static y final
        // métodos public y abstract (excepto estáticos y por defecto)
}
```


DEFINICIÓN DE INTERFACES

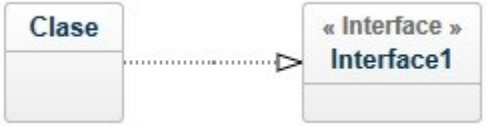
- **interface**
- Mismas normas de acceso y reglas de nombre que una clase.
- Existe herencia de interfaces (**extends**). Puede ser múltiple.

```
public interface NombreInterface [extends Interface1,
Interface2, …] {
    //declaraciones constantes
    double PI = 3.1416;

    // métodos
    void hacerAlgo(int i, double x);

    int hacerAlgoMas(String s);
}
```


IMPLEMENTACIÓN DE INTERFACES



- **implements**
- Una clase puede implementar más de una interfaz

```
public class Clase1 implements InterfaceA [,InterfaceB, ...,
InterfaceN] {
        ...
}
```

Si además la clase hereda de una superclase, se pone primero **extends**

```
public class Clase1 extends SuperClase implements InterfaceA {
        …
}
```


IMPLEMENTACION DE JERARQUÍA DE INTERFACES

```
interface SuperBase1 { . . . ]
interface Base1 extends SuperBase1 { . . . }
interface Base2 extends SuperBase1 { . . . }
interface ComunDerivado extends Base1, Base2 { . . . }

class nomeClase implements ComunDerivado {
       // todos los métodos de la jerarquía de
interfaces
}
```


MÉTODOS POR DEFECTO

- A partir de Java 8
- **default**.
- Un método puede tener una implementación por defecto descrita en la interfaz.
- Se pueden redefinir en las clases que implementan la interfaz

```
public interface NombreInterface {

    default public void metodoPorDefecto() {
        System.out.println("Este es un método por
                                        defecto");
    }
}
```


MÉTODOS ESTÁTICOS

- A partir de Java 8
- **static**.
- Misma sintaxis que los métodos estáticos en clases
- No se pueden redefinir en las clases que implementan la interfaz

```
public interface NombreInterface {

    public static void metodoEstatico()
       {  System.out.println("Método estático en un
       interfaz");
 }
}
```

- Para usarlo: `NombreInterface.metodoEstatico`

INTERFACES COMO TIPO

- Una interfaz se puede usar como nombre del tipo de dato para crear una instancia de un objeto.
- La clase del objeto a crear debe implementar dicha interfaz.

```
ClaseQueImplementaInterfaz objeto1 = new
ClaseQueImplementaInterfaz();

InterfazAImplementar objeto2 = new ClaseQueImplementaInterfaz();
```

- Podemos crear un array de tipo de la interfaz y almacenar objetos de clases que implementen dicha interfaz

```
InterfazAImplementar miArray = new InterfazAImplementar[5];
miArray[0] = objeto1;
miArray[1] = new OtraClaseQueImplementaInterfaz();
…
```


POLIMORFISMO CON INTERFACES

- Java también hace uso de polimorfismo con la herencia de interfaces y las clases que lo implementan.

```
ClaseQueImplementaInterfaz obj1 = new ClaseQueImplementaInterfaz();

obj1.saludar("Hola Mundo");

InterfaceHija obj2 = new
ClaseQueImplementaInterfaz();  obj2.saludar("Hola
Mundo, otra vez");

InterfaceBase obj3 = new
ClaseQueImplementaInterfaz();  obj3.saludar("Hola
Mundo, por tercera vez");
```

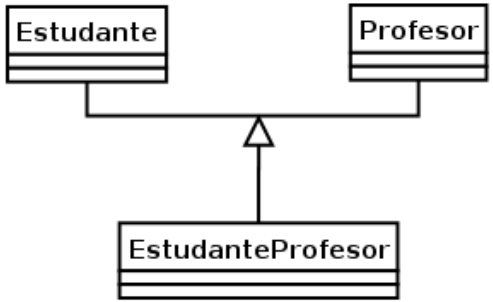
```
public interface InterfaceBase
 {
    void saludar(String
mensaje);
```


CLASES ABSTRACT QUE IMPLEMENTAN UNA INTERFAZ

- Una clase que implementa una interfaz tiene obligación de implementar todos sus métodos.

- Sin embargo, una clase **abstract** puede dejar métodos sin implementación, obligando a quienes la extiendan a hacerlo.

SIMULAR HERENCIA MÚLTIPLE




```
class Estudiante {
        //Atributos y métodos
}

class Profesor {
        //Atributos y métodos
}

interface IEstudiante {
        //mismos métodos que la clase estudiante
}

class EstudianteProfesor extends Profesor implements IEstudiante {
        …
}
```

CLASES ABSTRACTAS o INTERFACES: ¿Qué usar?

- **Clases Abstractas:**

  - ✓ Compartir código con clases muy relacionadas.
  - ✓ Necesitamos definir atributos que no sean estáticos o constantes.
  - ✓ Las subclases usarán métodos *protected*

- **Interfaces:**

  - ✓ Definir un tipo de comportamiento, sin importar quien lo implemente
  - ✓ Si necesitamos **herencia múltiple**.

CLASES ABSTRACTAS vs INTERFACES

INTERFACES	CLASES ABSTRACTAS
No se pueden instanciar	No se pueden instanciar
Métodos sin implementación	Métodos sin implementación
Métodos con implementación por defecto y estáticos	Métodos con implementación
Atributos estáticos o constantes	Cualquier tipo de atributos
Métodos públicos, privados o por defecto	Métodos públicos, privados, protegidos o por defecto
Una clase puede implementar varios interfaces	Una clase solo puede heredar de otra

¿QUÉ USAMOS?

INTERFACES	CLASES ABSTRACTAS
Clases no relacionadas podrán implementar los métodos.	Compartir código con clases muy relacionadas.
Si se quiere indicar que existe un tipo de comportamiento, pero no sabemos quien lo implementa.	Las clases derivadas usarán métodos *protected*
Si necesitamos tener **herencia múltiple.**	Queremos definir atributos que no sean estáticos o constantes.