



# Clases Abstractas e Interfaces

## Clases Abstractas

Es una clase muy similar a una clase común, posee métodos, atributos y constructor pero tiene al menos un **método abstracto**, donde un **método abstracto** es un método vacío, cuya utilidad es definir *qué* puede hacer la clase pero *no el cómo*.

- Una clase se convierte en abstracta si posee al menos **un método abstracto**.
- Una clase se puede declarar abstracta y no poseer métodos abstractos.
- Puede ser heredada, y las clases hijas deben implementar los métodos abstractos.
- Si bien poseen constructor, las clases abstractas **NO PUEDEN SER INSTANCIADAS**. Su constructor está pensado para que sea heredado por sus clases hijas.
- Sirven para agrupar comportamiento similar de otras clases.
- Sirven como “plantilla” para otras clases que tengan atributos o métodos en común.

## Ejemplo de una clase Abstracta

```
public abstract class Figure {  
    private Integer numberOfSides;  
  
    public Figure(Integer numberOfSides) {  
        this.numberOfSides = numberOfSides;  
    }  
    public Integer getNumberOfSides() {  
        return numberOfSides;  
    }  
    public void setNumberOfSides(Integer numberOfSides) {  
        this.numberOfSides = numberOfSides;  
    }  
    public abstract Double calculateArea();  
}
```

Se declara con la palabra reservada abstract

Puede poseer un constructor

Posee métodos comunes

Posee un método abstracto

## Interfaces

Se puede definir como una agrupación de métodos abstractos y atributos constantes (o inexistentes), que especifica una serie de métodos, dejando que el comportamiento sea definido por las clases hijas que **implementen** a la interfaz.

- Solo pueden extender/implementar otras interfaces.
- Permiten implementación múltiples, es decir una clase puede implementar varias interfaces al mismo tiempo.
- Al igual que las clases abstractas **NO PUEDEN SER INSTANCIADAS**.
- Solo pueden tener métodos públicos (no pueden ser ni private ni protected)
- Solo puede tener atributos constantes (es decir, definidos como static y final)

### Ejemplo 3: Implementación

```
public class RepositoryClient implements RepositoryClient {
    @Override
    public void findById(Integer id) {
        return null;
    }
    @Override
    public Client save(Client obj) {
        return null;
    }
    @Override
    public List<Client> findAll() {
        return null;
    }
}
```

Se usa la palabra reservada implements para hacer que la clase implemente la interfaz.

Al implementar se indica la clase a implementar.

Los métodos que están implementados, definidos según el generador especificado.

### Ejemplo 3: Interfaz genérica

```
public interface RepositoryClient<T> {
    public T findById(Integer id);
    public T save(T obj);
    public List<T> findAll();
}
```

Se utiliza con el operador <T> que indica un generico.

Los métodos pueden declarar el tipo generico.

También pueden recibir el tipo generico en un método.

Si se usa interfaces especificando el generico.

### Ejemplo 2: Método default

```
public interface Figure {
    final static Double PI = 3.14;
    public Double area();
    public Integer hashCode();
    default Boolean isRectangle(Figure f) {
        if (f.hashCode() == 4)
            return true;
        else return false;
    }
}
```

Se puede definir un método default en una interfaz.

## Diferencias entre Interfaces y clases Abstractas

- En las interfaces no puede haber elementos privados, mientras que en una clase abstracta si
- En las interfaces no se establecen jerarquías de clases, por lo tanto clases sin relación alguna pueden implementar la misma interfaz.
- Las interfaces no poseen constructores y las clases abstractas si (por más que ninguna de las dos sean instanciadas).
- Las interfaces no implementan métodos, las clases abstractas pueden hacerlo.
- Una clase puede implementar varias interfaces, pero no puede extender de varias clases abstractas o clases