



Concepto Abstracción

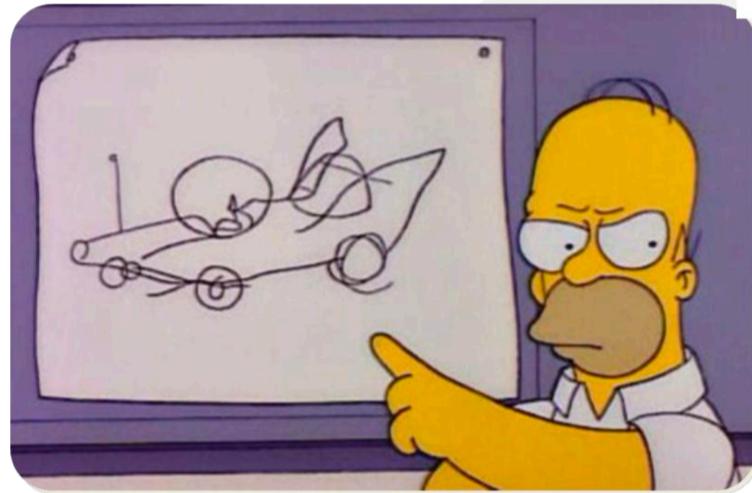
Clases Abstractas en Java

La **abstracción** permite resaltar las partes más representativas de algo, ignorando los detalles para centrarse en lo esencial.



Por ejemplo, cuando vemos a una persona, no analizamos cada detalle, sino que identificamos elementos básicos como cabeza, tronco, brazos y piernas. Eso basta para reconocer que es un ser humano.

Con la abstracción Hay un Enfasis en el **¿Qué hace?** mas que en el **¿Como lo hace?**



Homero Simpson construyendo el auto de sus sueños



¿Que vemos aquí?



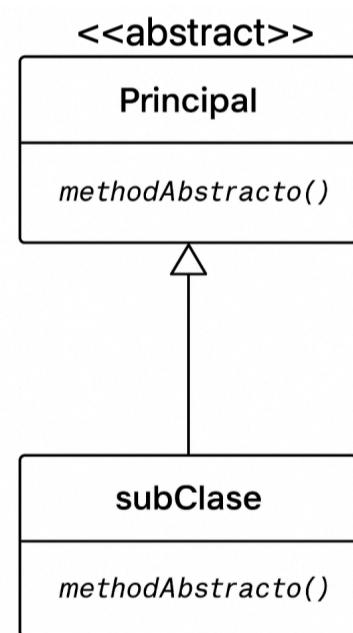
En Java, la abstracción se aplica mediante el uso de **clases abstractas**, las cuales funcionan principalmente dentro del concepto de **herencia**.

Una **clase abstracta** puede contener uno o más **métodos abstractos**, es decir, métodos **sin implementación**, cuyo comportamiento deberá ser definido por las clases **concretas** que la hereden.

Estas clases permiten definir comportamientos comunes para un conjunto de clases relacionadas, sin importar las diferencias en cómo cada una implementa dichos métodos.

De esta forma, logramos una aplicación más **organizada, flexible y reutilizable**.

Como se representa.



```

public abstract class Principal{
    /**Método Abstracto sin implementación*/
    public abstract void metodoAbstracto();
}
    
```

Método sin implementación

```

class subClase extends Principal{
    @Override
    public void metodoAbstracto() {
        /**Implementación definida por la clase concreta*/
    }
}
    
```

Gracias a la herencia se puede usar el método para darle el comportamiento deseado

Cuándo usar clases abstractas

Se utilizan cuando existen varias clases con **características o acciones comunes**, pero que presentan **diferentes comportamientos** en su implementación.

Por ejemplo, si todas las clases comparten una estructura general pero cada una ejecuta sus métodos de forma distinta.

Reglas de las clases abstractas

- Una **clase abstracta no puede ser instanciada** (no se pueden crear objetos directamente con `new`).
- Si al menos un método es `abstract`, la clase **debe declararse como abstracta**.
- Una clase abstracta **puede tener métodos concretos** (con cuerpo) además de métodos abstractos.
- Los **métodos abstractos no tienen cuerpo**, solo se declaran (sin `{}`).
- La **primera subclase concreta** que herede de una clase abstracta **debe implementar todos los métodos abstractos** de su clase padre, incluyendo el árbol de herencia.

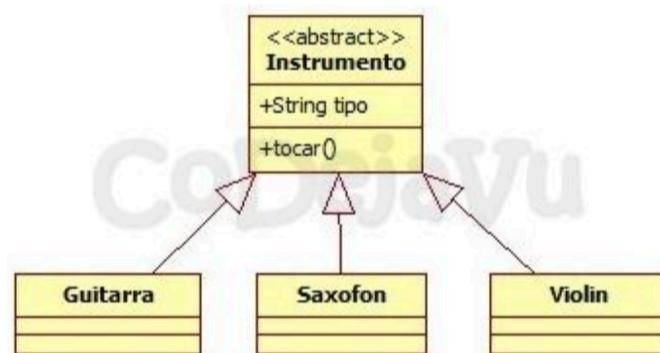
Ventajas

- Mantienen la aplicación **más organizada y fácil de entender**.
- Garantizan que ciertas propiedades o métodos solo estén disponibles para las **clases hijas**.
- Permiten definir **procesos generales** que luego serán implementados de forma personalizada por las clases concretas.

Ejemplo conceptual

Si se tiene una clase que hereda de otra Abstracta, java obliga a poner en el código todos los métodos abstractos de la clase padre para que sean implementados, **así se convierten en concretos y su funcionalidad o cuerpo será definido dependiendo de para que la necesite**, de esa manera si se tiene otra clase que también hereda del mismo parente, se implementará el mismo método pero con un comportamiento distinto

Por ejemplo, Todas las clases de **instrumentos musicales** pueden heredar de una clase abstracta `Instrumento`, la cual define el método `tocar()`.



Cada instrumento (por ejemplo, `Guitarra`, `Saxofon`, `Violin`) implementará este método de manera distinta, pero todos comparten la misma estructura general.

Veamos la clase abstracta `Instrumento`.

```
// Clase abstracta: define la estructura general
public abstract class Instrumento {
    // Atributo común para todos los instrumentos
    protected String tipo;

    // Constructor para inicializar el tipo
    public Instrumento(String tipo) {
        this.tipo = tipo;
    }

    // Método abstracto (sin cuerpo)
    public abstract void tocar();

    // Método concreto (con cuerpo)
    public void imprimir() {
        System.out.println("Este es un " + tipo);
    }
}
```

```

public void afinar() {
    System.out.println("Afinando el instrumento de tipo: " + tipo);
}

// Método getter para consultar el tipo
public String getTipo() {
    return tipo;
}
}

```

ten en cuenta que una. **clase abstracta puede tener constructor aunque no se pueda instanciar**, porque ese constructor **se ejecuta cuando una subclase concreta crea un objeto**.

Esto sirve para **inicializar los atributos comunes** que todas las subclases heredarán.

Veamos este proceso con sus clases Hijas **Guitarra** y **Saxofon**

```

// Clase concreta que hereda de Instrumento
public class Guitarra extends Instrumento {

    // Constructor: llama al constructor de la clase padre
    public Guitarra() {
        super("Cuerda"); // Establece el tipo de instrumento
    }

    // Implementa el método abstracto
    @Override
    public void tocar() {
        System.out.println("Tocando la guitarra 🎸 con las cuerdas");
    }
}

```

Como vemos en el constructor de la clase se hace el llamado a `super("Cuerda")`; enviando el tipo

```

// Clase concreta que hereda de Instrumento
public class Saxofon extends Instrumento {

    // Constructor: llama al constructor de la clase padre
    public Saxofon() {
        super("Viento"); // Establece el tipo de instrumento
    }

    // Implementa el método abstracto
    @Override
    public void tocar() {
        System.out.println("Tocando el saxofón 🎵, soplando fuerte");
    }
}

```

Lo mismo sucede con el constructor de **Saxofon**.

Finalmente podemos tener la lógica de creación de objetos hijos, donde vemos como se accede a el método concreto **afinar()** (común para todos) y usamos la implementación individual de **tocar()**.

```

// Clase principal para probar
public class Main {
    public static void main(String[] args) {
        // No se puede hacer: Instrumento i = new Instrumento(); ✗
    }
}

```

```

// Sí se pueden crear objetos de las subclases concretas
Guitarra guitarra = new Guitarra();
Saxofon saxofon = new Saxofon();

// Mostrar tipo de instrumento antes de usarlos
System.out.println("Instrumento: " + guitarra.getTipo());
guitarra.afinar();
guitarra.tocar();

System.out.println("\nInstrumento: " + saxofon.getTipo());
saxofon.afinar();
saxofon.tocar();
}

}

```

Qué ocurre aquí:

- `Instrumento` es una **clase abstracta** que define lo común entre todos los instrumentos.
- `tocar()` es un **método abstracto**: no tiene implementación.
- `afinar()` es un **método concreto**: tiene un comportamiento compartido.
- `Guitarra` y `Saxofon` **heredan de** `Instrumento` y cada una **implementa su versión personalizada** de `tocar()`.

Así se demuestra cómo la **abstracción** permite definir un modelo general, dejando los detalles a las clases que heredan.

Instructor: Cristian David Henao Hoyos