Polimorfismo

Contenido

Polimorfismo
- ¿Qué es el polimorfismo?
- Tipos de polimorfismo.
- Sobrescritura de equivalencias.

Clases abstractas
- ¿Qué es una clase abstracta?
- Miembros abstractos.
- Implementación de miembros abstractos.

¿Qué es el polimorfismo?

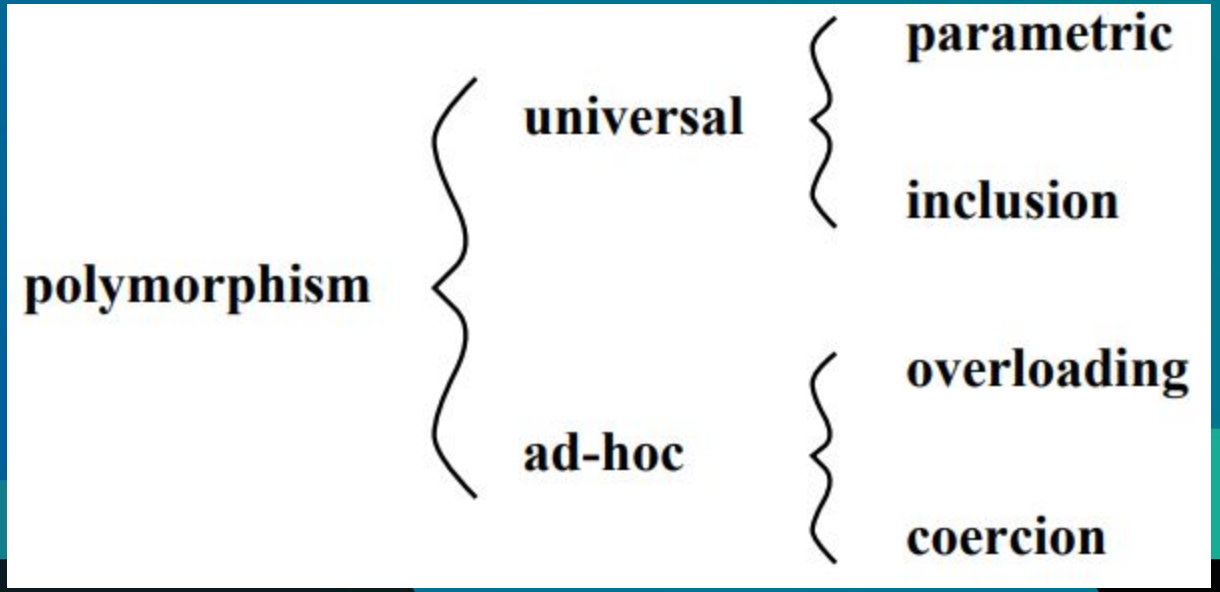
El término *polimorfismo* tiene origen en las palabras *poly* (muchos) y *morfo* (formas).

Aplicado a la POO hace referencia a que objetos de diferentes clases puedan ser accedidos utilizando la misma interfaz (método), mostrando un comportamiento distinto según cómo sean accedidos.

En POO, *polimorfismo* es el pilar fundamental que permite programar de forma genérica, generando aplicaciones escalables, reutilizables y mantenibles.

Tipos de polimorfismo

La clasificación de Cardelli y Wagner abarca dos grandes categorías: **polimorfismo universal** y **polimorfismo ad-hoc**.



Tipos de polimorfismo

Polimorfismo ad-hoc:
Ocurre cuando un método puede operar sobre diferentes tipos de datos, pero de una manera específica para cada tipo.

Se divide en **sobrecargas** (de métodos y de operadores (*)) y por **coerción**.

- **Sobrecarga** de métodos:
  Varios métodos en la misma clase tienen el mismo nombre pero diferentes parámetros.

- (*) *Sobrecarga de operadores*:
  No aplica para Java.

```java
class Calculadora {

    public int sumar(int a, int b) {
        return a + b;
    }

    public double sumar(double a, double b) {
        return a + b;
    }

    public int sumar(int a, int b, int c) {
        return a + b + c;
    }
}
```

Tipos de polimorfismo

Polimorfismo ad-hoc por coerción:
Ocurre cuando un lenguaje de programación permite que un tipo de dato se convierta automáticamente en otro tipo en el contexto de una operación, permitiendo que una operación o método se aplique a diferentes tipos de datos de manera uniforme.

```java
public static void main(...) {

    int a = 5;
    double b = 4.5;

    // 'a' SE CONVIERTE AUTOMÁTICAMENTE A 'double' ANTES DE LA SUMA
    double resultado = a + b;

    System.out.println(resultado); // 9.5

}
```

Tipos de polimorfismo

Polimorfismo universal:
Se refiere a la capacidad de una única interfaz para funcionar con múltiples tipos de datos.

Se subdivide en **polimorfismo paramétrico** y **polimorfismo por inclusión**.

Tipos de polimorfismo

Polimorfismo paramétrico:
Ocurre cuando un método o clase es **genérica** y puede trabajar con cualquier tipo de datos.

En Java, esto se implementa mediante **generics**.
(se verá más adelante)

```java
class Caja<T> {

    private T contenido;

    public void guardar(T contenido) {
        this.contenido = contenido;
    }

    public T obtener() {
        return this.contenido;
    }
}
```

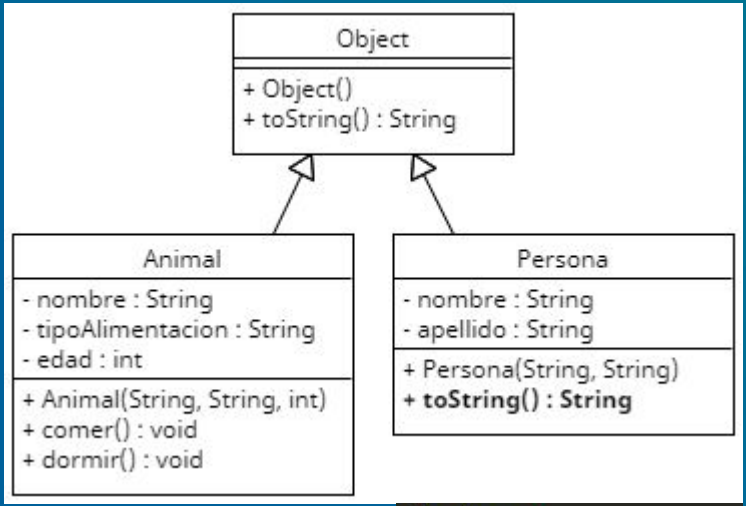
Tipos de polimorfismo

Polimorfismo por inclusión:
Se basa en la **herencia** y la **sobrescritura** de métodos.

Una referencia a una clase base puede apuntar a un objeto de una subclase, y se llama al método de la subclase correspondiente en *tiempo de ejecución*.

Cuando hablamos de polimorfismo a secas, por lo general nos referimos particularmente al polimorfismo por inclusión.

Ejemplos




```java
class Object {

    ...

    public String toString() {
        return getClass().getName() +
        '@' + Integer.toHexString(hashCode());
    }

}
```

```java
class Persona{

    ...

    @Override
    public String toString() {
        return this.nombre + " " +
        this.apellido;
    }

}
```

```java
public static void main(...) {

    Animal a = new Animal("sheik", "carne", 4);

    System.out.println(a.toString());
    // Animal@435c0d8e

    Persona p = new Persona("John", "Doe");

    System.out.println(p.toString());
    // John Doe
}
```

Sobrescritura de equivalencias

De ***object*** se heredan los métodos ***equals*** y ***hashCode***.

Por defecto: Dos objetos son iguales si tienen la misma dirección de memoria.

Ambos métodos pueden redefinirse en las clases derivadas con una nueva implementación.

Un *hashcode* es un valor numérico que se utiliza para identificar y comparar objetos, por ejemplo en las colecciones de tipo HashSet.

Dos objetos iguales deberían retornar el mismo hashcode.

```java
public class Persona {

    private String nombre;
    private int dni;

    @Override
    public boolean equals(Object o){

        boolean rta = false;

        if(o instanceof Persona)
        {
            rta = this.nombre.equals(((Persona)o).nombre) &&
                (this.dni == ((Persona)o).dni);
        }

        return rta;
    }

    @Override
    public int hashCode() {

        int hash = 7;
        hash = 23 * hash + Objects.hashCode(this.nombre);
        hash = 23 * hash + this.dni;

        return hash;
    }
}
```

Clases abstractas

¿Qué es una clase abstracta?

Su propósito es proporcionar una definición común que modele una jerarquía de herencia (representar algo abstracto, por lo tanto no se pueden instanciar).

Se puede decir que son clases incompletas cuyas piezas faltantes son aportadas por sus clases derivadas.

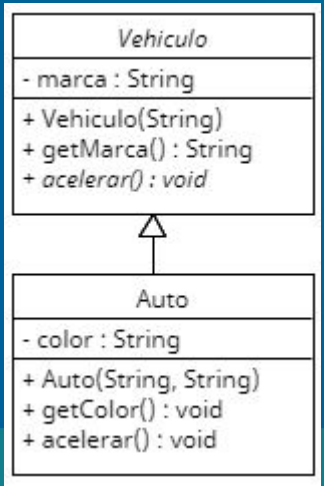
Los **miembros abstractos** definen una operación pero no su implementación.

Las clases derivadas concretas **deben** aportar dicha implementación.

Las clases abstractas son las únicas que pueden contener miembros abstractos.

Miembros abstractos

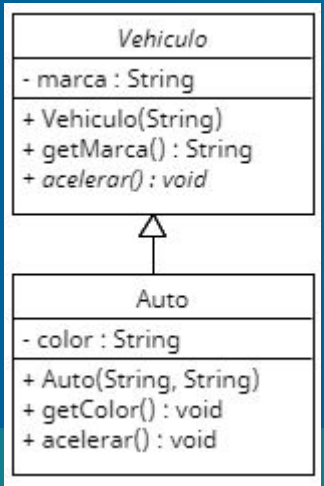
Los miembros que se pueden declarar abstractos en Java son los ***métodos***.




```java
public abstract class Vehiculo {

    private String marca;

    public Vehiculo(String marca){
        this.marca = marca;
    }

    public String getMarca(){
        return this.marca;
    }

    public abstract void acelerar();
}
```

Implementación de miembros abstractos

Los miembros **deben** ser implementados por la primera clase derivada concreta.




```java
public class Auto extends Vehiculo {

    private String color;

    public Auto(String marca, String color) {
        super(marca);
        this.color = color;
    }

    public String getColor(){
        return this.color;
    }

    @Override
    public void acelerar() {
        System.out.println("El auto está acelerando...");
    }
}
```

Ejercitación