CARACTERÍSTICAS DEL POO

- **Abstracción**
- **Encapsulamiento**
- **Herencia**
- **Polimorfismo**
- **Clases y Objetos**
- **Métodos y Atributos**
- **Modularidad**
- **Reusabilidad**

¿QUÉ ES ABSTRACCIÓN?

Una abstracción es una manera de reducir la complejidad y permitir un diseño e implementación más eficientes en sistemas de software complejos. Oculta la dificultad técnica de los sistemas detrás de APIs más simples.

VENTAJAS DE LA ABSTRACCIÓN

- ➢ Ayuda al usuario a evitar escribir código de bajo nivel.
- ➢ Evita duplicar código y aumenta la reusabilidad.
- ➢ Se puede cambiar la implementación interna de la clase de forma independiente sin afectar al usuario.
- ➢ Ayuda a aumentar la seguridad de la aplicación o programa ya que solo los detalles importantes son proporcionados al usuario.

EJEMPLO:

Imaginemos que estamos creando un programa para gestionar una biblioteca. Podríamos abstraer el concepto de «**libro**» como un objeto. En lugar de preocuparnos por todos los detalles internos de un libro, como su estructura física o el algoritmo utilizado para almacenarlo, podemos centrarnos en las características esenciales que lo definen, como el título, el autor y la fecha de publicación..

```
class Libro:

    def __init__(self, titulo, autor, fecha_publicacion):

        self.titulo = titulo

        self.autor = autor

        self.fecha_publicacion = fecha_publicacion



    def obtener_titulo(self):

        return self.titulo



    def obtener_autor(self):

        return self.autor



    def obtener_fecha_publicacion(self):

        return self.fecha_publicacion
```

¿QUÉ ES HERENCIA?

Cuando hablamos de herencia en programación no nos referimos precisamente a que algún familiar lejano nos ha podido dejar una fortuna, ya nos gustaría. En realidad se trata de uno de los pilares fundamentales de la programación orientada a objetos.

Terminología importante:

- Superclase: la clase cuyas características se heredan se conoce como superclase (o una clase base o una clase principal).
- Subclase: la clase que hereda la otra clase se conoce como subclase (o una clase derivada, clase extendida o clase hija). La subclase puede agregar sus propios campos y métodos, además de los campos y métodos de la superclase.
- Reutilización: la herencia respalda el concepto de “reutilización”, es decir, cuando queremos crear una clase nueva y ya hay una clase que incluye parte del código que queremos, podemos derivar nuestra nueva clase de la clase existente. Al hacer esto, estamos reutilizando los campos/atributos y métodos de la clase existente.

EJEMPLOS:
Para entender el concepto mejor, crearemos una superclase llamada Dos Dimensiones, que almacena el ancho y la altura de un objeto bidimensional, y una subclase llamada Triángulo que usaremos la palabra clave extends para crear esa subclase.

```
//Clase para objetos
de dos dimensiones
class DosDimensiones{
    double base;
    double altura;

    void mostrarDimension(){
        System.out.println("La
        base y altura es: "+base+"
        y "+altura);
    }
}
```

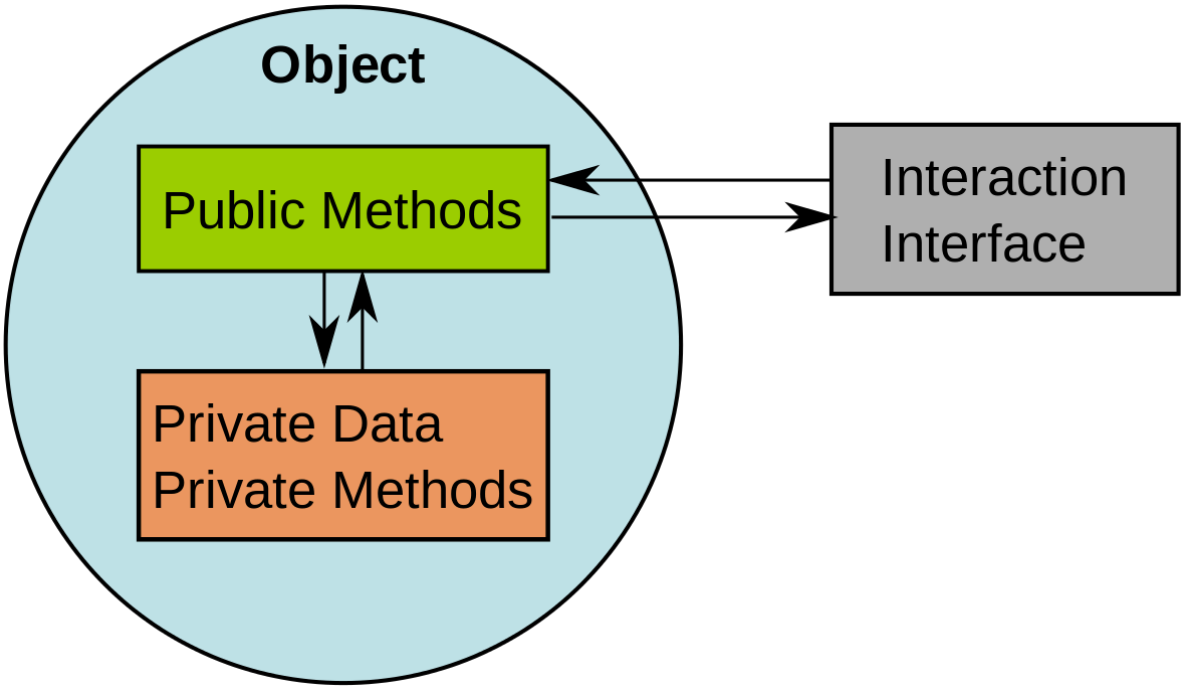
```
//Una subclase de DosDimensiones
para Triangulo
class Triangulo
extends DosDimensiones{
 String estilo;

    double area(){
     return base*altura/2;
    }

    void mostrarEstilo(){
     System.out.println
   ("Triangulo es: "+estilo);
  }
}

class Lados3{
  public static void
  main(String[] args) {
   Triangulo t1=new Triangulo();
   Triangulo t2=new Triangulo();

   t1.base=4.0;
   t1.altura=4.0;
   t1.estilo="Estilo 1";

   t2.base=8.0;
   t2.altura=12.0;
   t2.estilo="Estilo 2";

    System.out.println("Información
   para T1: ");
    t1.mostrarEstilo();
    t1.mostrarDimension();
    System.out.println("Su área es:
     "+t1.area());

    System.out.println();

    System.out.println("Información
   para T2: ");
    t2.mostrarEstilo();
    t2.mostrarDimension();
    System.out.println("Su área es:
     "+t2.area());

    }
}
```

¿QUÉ ES ENCAPSULAMIENTO?

Es un principio que consiste en ocultar los detalles internos de un objeto y exponer sólo las operaciones que pueden ser realizadas sobre él. Esto se logra mediante el uso de clases, que actúan como contenedores para los datos (atributos) y los métodos (funciones) que operan sobre esos datos.

El encapsulamiento tiene varios beneficios:

- **Abstracción de datos:** Permite definir una interfaz clara y consistente para interactuar con un objeto, ocultando los detalles internos de su implementación.
- **Modularidad**: Facilita la división del código en componentes más pequeños y manejables, lo que hace que el desarrollo y el mantenimiento del software sean más sencillos.
- **Protección de datos:** Al ocultar los detalles internos de un objeto, se evita que los datos sean modificados de manera inapropiada desde fuera de la clase. Esto ayuda a mantener la integridad de los datos y a prevenir errores.
- **Reusabilidad:** Permite reutilizar objetos sin preocuparse por cómo están implementados internamente, lo que promueve un diseño de software más flexible y adaptable.



¿QUÉ ES EL POLIFORMISMO?

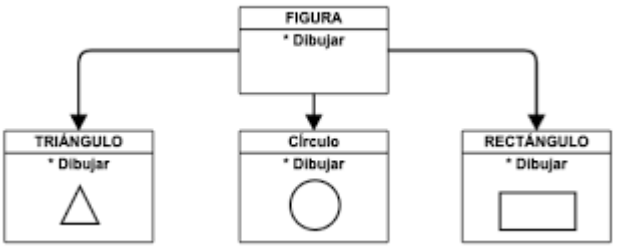
El polimorfismo es un principio de la programación orientada a objetos (POO) que permite que un objeto pueda comportarse de diferentes maneras según el contexto en el que se encuentre. En términos más simples, el polimorfismo permite que los objetos de diferentes clases puedan ser tratados de manera uniforme a través de una interfaz común.

Hay dos tipos principales de polimorfismo en POO:

1. **Polimorfismo de sobrecarga (overloading):** Se refiere a la capacidad de definir múltiples métodos con el mismo nombre en una clase, pero con diferentes tipos de parámetros o número de parámetros. El compilador o intérprete determina qué método utilizar en función de la firma de los parámetros que se le pasan. Esto se conoce como sobrecarga de métodos.

2. **Polimorfismo de sobrescritura (overriding):** Se refiere a la capacidad de que una clase hija provea una implementación específica de un método que ya está definido en su clase padre. Esto permite que los objetos de la clase hija puedan ser tratados como objetos de la

clase padre cuando se utilizan a través de una referencia de la clase padre.

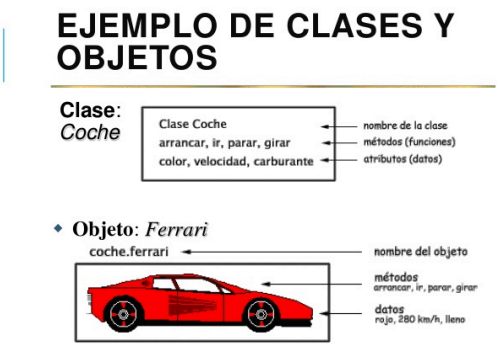


¿QUÉ SON LOS MÉTODOS Y ATRIBUTOS?

En la programación orientada a objetos (POO), los métodos y los atributos son componentes fundamentales de las clases, que son las plantillas para crear objetos. Aquí te explico ambos conceptos:

1. **Atributos:** Los atributos son variables que representan el estado de un objeto. Pueden ser cualquier tipo de dato, como números, cadenas de texto, booleanos, u otros objetos. Los atributos describen las características de un objeto y definen su estado en un momento dado. Por ejemplo, si estamos modelando una clase "Persona", los atributos podrían ser "nombre", "edad", "género", etc.

2. **Métodos:** Los métodos son funciones que definen el comportamiento de un objeto. Representan las acciones que un objeto puede realizar o las operaciones que se pueden realizar sobre él. Los métodos pueden acceder y manipular los atributos de un objeto, así como realizar otras operaciones relevantes para el objeto. Siguiendo el ejemplo de la clase "Persona", podríamos tener métodos como "caminar", "hablar", "comer", etc.



¿QUÉ ES REUSABILIDAD?

La reusabilidad en programación orientada a objetos (POO) se refiere a la capacidad de utilizar y reutilizar componentes de software (como clases, objetos, métodos, etc.) en diferentes partes de un programa o en programas diferentes. Este concepto es fundamental en la POO y se basa en la idea de que los objetos y las abstracciones de software deben diseñarse de manera modular y general para que puedan ser aplicados en múltiples contextos.

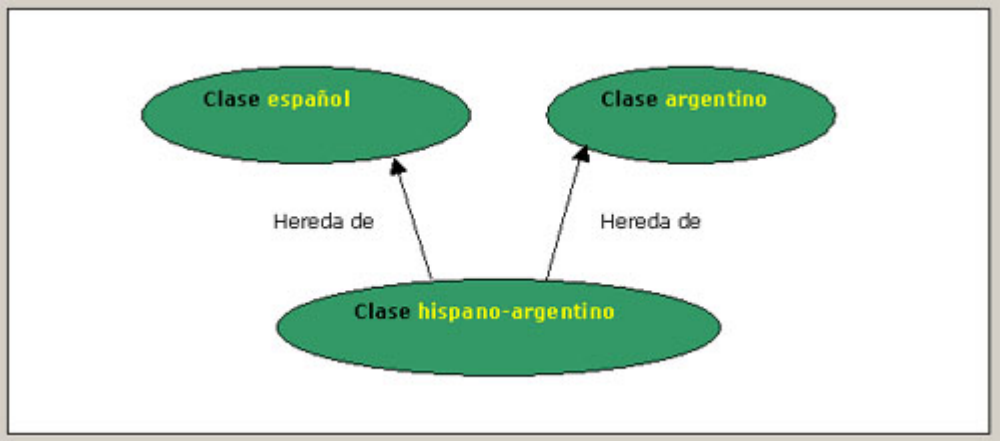
Hay varias formas en las que se puede lograr la reusabilidad en POO:

1. **Herencia:** La herencia permite que una clase hereda atributos y métodos de otra clase, lo que facilita la reutilización de código. Una clase hija puede extender o especializar el comportamiento de su clase padre, lo que promueve la reusabilidad y la extensibilidad del software.

2. **Composición:** En lugar de heredar comportamiento de una clase base, un objeto puede contener otros objetos como parte de su estructura interna. Esto se conoce como composición y permite construir objetos complejos mediante la combinación de objetos más simples.

3. **Interfaces:** Las interfaces definen un conjunto de métodos que una clase debe implementar. Esto permite que diferentes clases implementan la misma interfaz de manera diferente, lo que promueve la reusabilidad al definir un contrato común para objetos que pueden tener diferentes implementaciones.

4. **Bibliotecas y Frameworks:** El uso de bibliotecas y frameworks proporciona acceso a un conjunto de componentes de software predefinidos que pueden ser reutilizados en diferentes proyectos. Estas bibliotecas y frameworks a menudo están diseñados para ser genéricos y modulares, lo que facilita su integración y reutilización.



¿QUÉ ES MODULARIDAD?

La modularidad en programación orientada a objetos (POO) se refiere a la capacidad de dividir un sistema de software en módulos independientes y autónomos, cada uno de los cuales cumple una función específica y se puede desarrollar, probar, mantener y compilar de forma independiente. Estos módulos encapsulan diferentes aspectos del sistema y están diseñados para interactuar entre sí a través de interfaces bien definidas.

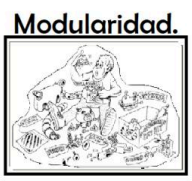
La modularidad es un principio fundamental en la POO y ofrece varios beneficios:

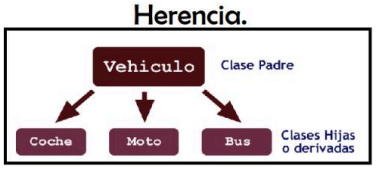
1. **Facilita el desarrollo:** Dividir un sistema en módulos más pequeños permite que múltiples desarrolladores trabajen en diferentes partes del sistema de manera simultánea y sin interferir entre sí. Esto acelera el proceso de desarrollo y mejora la productividad del equipo.

2. **Promueve la reutilización:** Los módulos bien diseñados y encapsulados pueden ser reutilizados en diferentes partes del sistema o incluso en sistemas diferentes. Esto reduce la duplicación de código y facilita el mantenimiento del software.

3. **Mejora la legibilidad y mantenibilidad:** Al dividir un sistema en módulos más pequeños y cohesivos, el código se vuelve más legible y comprensible. Además, los cambios y las actualizaciones pueden realizarse de manera más segura y eficiente en módulos independientes, lo que facilita el mantenimiento del software a largo plazo.

4. **Facilita la prueba y depuración:** Los módulos independientes son más fáciles de probar y depurar, ya que los errores tienden a estar más localizados y son más fáciles de aislar. Esto simplifica el proceso de identificación y corrección de errores en el software.





¿QUÉ ES CLASES Y OBJETOS?

En programación orientada a objetos (POO), las clases y los objetos son conceptos fundamentales:

1. **Clases:** Una clase es una plantilla que define las propiedades y comportamientos comunes a un conjunto de objetos. En otras palabras, una clase es un modelo o un plano a partir del cual se pueden crear objetos. Las clases se utilizan para representar entidades del mundo real y definir cómo se comportan y qué pueden hacer. Por ejemplo, si estamos construyendo un sistema de gestión de biblioteca, podríamos tener una clase llamada "Libro" que define los atributos (título, autor, año de publicación, etc.) y los métodos (prestar, devolver, etc.) asociados con los libros.

2. **Objetos:** Un objeto es una instancia específica de una clase. Se crea a partir de la plantilla proporcionada por la clase y representa una entidad concreta del mundo real. Cada objeto tiene un estado (determinado por sus atributos) y un comportamiento (determinado por sus métodos). Siguiendo el ejemplo anterior, si creamos un objeto llamado "miLibro" a partir de la clase "Libro", este objeto tendría valores específicos para sus atributos (como título, autor, etc.) y podría realizar acciones específicas (como prestar, devolver, etc.).