Aplicación del paradigma orientado a objetos

Breve descripción:

Durante el transcurso de este componente formativo, se abordan contenidos relacionados con la implementación de clases, objetos, atributos, constructores, métodos, herencia y relaciones en el lenguaje de programación Java.

Junio 2024

Tabla de contenido

Introducción

La Programación Orientada a Objetos (POO) se caracteriza por aplicar principios de abstracción, encapsulación, herencia y polimorfismo. En POO, los datos (atributos) y los métodos (comportamientos) se encapsulan dentro de objetos relacionados entre sí, permitiendo la ocultación de detalles de implementación. Esta ocultación es fundamental, ya que, similar a cómo se puede manejar una moto sin conocer el funcionamiento interno de sus sistemas, en la ingeniería de software permite la comunicación entre objetos a través de interfaces definidas sin revelar cómo están implementados otros objetos.

Este componente formativo ofrece una inmersión en los principios y aplicaciones de la Programación Orientada a Objetos, guiando a los estudiantes a través de los siguientes aspectos clave:

Introducción a las características fundamentales de la Programación Orientada a Objetos (POO).

Profundización en la implementación de clases y objetos, abarcando:

- Atributos y métodos de las clases.
- Constructores y destructores para la gestión de objetos.
- Métodos accesores y modificadores, esenciales para la manipulación de datos.
- Sobrecarga de métodos (overload), facilitando la flexibilidad y reutilización de código.

Exploración de la comunicación entre clases mediante:

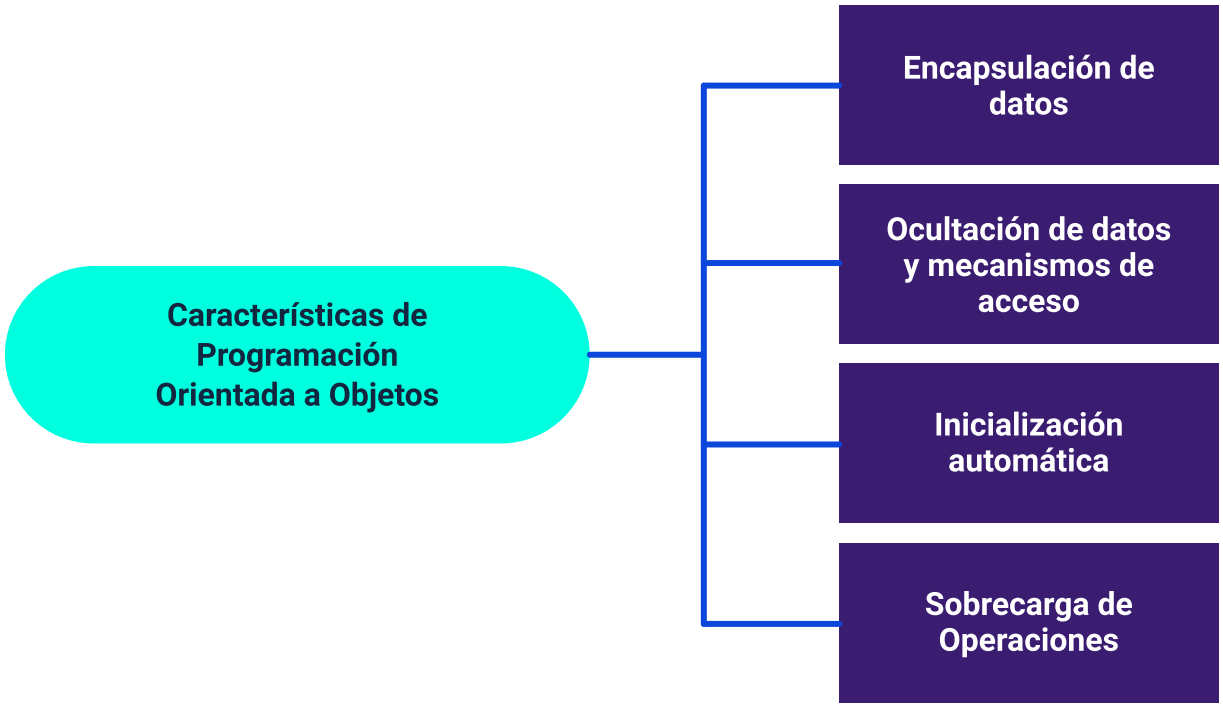
- Implementación de asociaciones, para relacionar objetos de manera lógica.

- Implementación de composición/agregación, definiendo relaciones de pertenencia.
- Implementación de herencia, permitiendo la extensión y reutilización de funcionalidades.

1. Características del lenguaje de programación orientada a objetos

Aunque es posible implementar Programación Orientada a Objetos (POO) en lenguajes como C y PASCAL, esta se torna compleja y confusa en programas de gran tamaño. Resulta más apropiado utilizar un lenguaje específicamente diseñado para soportar los principios de la POO para su implementación. Esto se debe a que, dependiendo de las características que ofrecen, los lenguajes de programación se pueden clasificar en dos categorías: lenguajes basados en objetos y lenguajes orientados a objetos. La programación basada en objetos es un estilo que soporta, principalmente, la encapsulación e identidad de los objetos, centrando sus características principales en:

Figura 1. Características de POO



Se dice que los lenguajes de programación basados en objetos son aquellos que soportan el uso de objetos, pero no incorporan todas las características de la

programación orientada a objetos, como la herencia o el polimorfismo. Un ejemplo de lenguaje de programación que no es completamente orientado a objetos es Ada, que aunque utiliza objetos, no incluye la herencia ni la ligadura dinámica, y es además un lenguaje multipropósito.

La programación orientada a objetos (POO), por otro lado, incluye todas las características de los lenguajes basados en objetos y añade funcionalidades clave como la herencia y la ligadura dinámica (también conocida como polimorfismo), que permiten una mayor flexibilidad y reutilización del código.

Las características de programación orientada a objetos son:

- Herencia
- Ligadura dinámica
- Características basadas en objetos

Entre los lenguajes que incluyen soporte para la programación orientada a objetos se cuentan C++, Smalltalk, Object Pascal y Java. Existe una amplia variedad de lenguajes de programación que ofrecen funcionalidades tanto de programación basada en objetos como de orientación a objetos. En particular, Java es un lenguaje de programación orientada a objetos que fue desarrollado por Sun Microsystems y lanzado en 1995. Java compila el código fuente en un `bytecode` que es ejecutado por la Máquina Virtual de Java (JVM), lo que lo diferencia de lenguajes como C y C++, que son principalmente compilados a código máquina y no interpretados. Java combina características de lenguajes interpretados a través de su JVM, garantizando portabilidad y eficiencia en la ejecución de sus programas.

Sus características principales son:

- **Sencillo**

  Es fácil de entender, elimina la complejidad de otros lenguajes.

- **Orientado a objetos**

  Al ser un paradigma orientado a objetos la filosofía de programación orientada a objetos facilita la creación y mantenimiento de programas.

- **Independiente de la arquitectura y portable**

  Al compilar un programa en Java, se obtiene un tipo de código binario conocido como Java Bytecode. Este código es interpretable por diferentes computadoras de igual manera. Como el código compilado de Java es interpretado, un programa compilado de Java puede ser utilizado por cualquier computadora que tenga implementado el intérprete de Java.

- **Robusto**

  Java simplifica la gestión de la memoria.

- **Multitarea**

  Java puede ejecutar diferentes hilos de código al mismo tiempo.

- **Dinámico**

  En Java no es necesario cargar completamente el programa en memoria, sino que las clases compiladas pueden ser cargadas en bajo demanda en tiempo de ejecución.

Además de las características mencionadas, hay diversos entornos de desarrollo integrados (IDE, por sus siglas en inglés "Integrated Development Environment") disponibles para la programación en Java. Estos entornos proporcionan a los

programadores un espacio de trabajo unificado que facilita todo el proceso de desarrollo de aplicaciones. Entre los IDE más reconocidos se encuentran:

Tabla 1. Entornos de desarrollo Java

IDE	Enlace para descarga
NetBeans	https://netbeans.apache.org/download/index.html
Eclipse	http://www.eclipse.org/downloads/
JBlue	https://bluej.org/versions.html

2. Implementación de clases y objetos

Los elementos centrales de un lenguaje de programación son las clases, ya que se utilizan para realizar todas las operaciones lógicas durante el desarrollo de software. Las clases actúan como estructuras o plantillas que modelan objetos del mundo real y pueden definirse como las abstracciones de estos objetos en el código. Los objetos derivados de estas clases pueden representar entidades físicas o conceptuales, como cosas, lugares o personas. Por lo general, las clases definen propiedades, comportamientos y relaciones con otras clases.

En Java, las convenciones de codificación estándar incluyen reglas específicas para nombrar clases, métodos y atributos. Las reglas para nombrar clases son las siguientes:

a) La primera letra de cada nombre de clase debe ser mayúscula.
b) Se debe utilizar la nomenclatura camelCase para nombres compuestos, donde la primera letra de cada palabra interna es mayúscula; por ejemplo, "CuentaAhorros".
c) Los nombres de las clases deben ser sustantivos, indicativos de lo que representan.

Estructura de una clase en Java

Una clase en Java se compone por las siguientes declaraciones:

- Paquete.
- Comentarios.
- Definición de clases.
- Atributos.

- Constructores.
- Métodos.

Ejemplo de estructura:

public class Nombre de la Clase {}

- Donde la palabra public indica que el alcance de la clase será público.
- La palabra class indica que es una clase.
- Y las llaves {} se usan para agrupar los bloques de código.

Basado en lo anterior, a continuación, se relaciona un ejemplo sobre el diseño de la clase de producto con sus atributos y métodos:

Producto

- -código: int
- -nombre: String
- -precio: double
- +obtenerPrecio()
- +calcularTotal()

En Java este diseño se representaría de la siguiente manera:

Tabla 2. Implementación de la clase Producto en Java

Línea de código	Programación	Explicación
12 13	public clases Producto { // Clase	Donde la línea 12 del código representa la estructura de la clase que es de acceso público y su nombre es Producto.

Línea de código	Programación	Explicación

14
15
16
17
	private int codigo; // atributos
	private String nombre; // atributos
	private double precio; // atributos
		De la línea 14 a la 16 representa los atributos de la clase.

18
19
20
21
22
23
24
25
26
	public void obtenerPrecio() { // método
	System.out.println("El precio es" + precio);
	}
	
	public void calcularTotal() { // método
	double total = precio*5000;
	}
	
	}
		A partir de la línea 18 se encuentran los métodos.

La creación de un objeto en Java se realiza mediante la instanciación de una clase. Al instanciar una clase, se crea un objeto que hereda los atributos y métodos de dicha clase. Es posible crear múltiples instancias de una misma clase.

Por ejemplo, al instanciar un objeto de la clase Producto, asignamos valores a sus atributos, como código, nombre y precio, y podemos utilizar sus métodos, tales como obtenerPrecio() y calcularTotal(). En Java, la instanciación se lleva a cabo con la palabra reservada new, seguida del constructor de la clase. La sintaxis para crear un objeto de la clase Producto sería:

Producto p = new Producto();

Una vez instanciado, el objeto p tiene acceso a los métodos y atributos de la clase Producto. La forma correcta de invocar los métodos sería a través del objeto creado:

p.obtenerPrecio();

p.calcularTotal();

2.1. Atributos y métodos de la clase

Los atributos de una clase son definidos según esta sintaxis: [modifVisibilidad] tipo nombre Variable [= valor Inicial]:

- **Public**

  Indica que los atributos son accesibles desde cualquier lugar del programa. No hay restricciones en el alcance de los miembros de datos públicos.

- **Private**

  Indica que los atributos solo son accesibles dentro de la clase en la que se declaran.

- **Protected**

  Indica que los atributos son accesibles dentro del mismo paquete o sub-clases en paquetes diferentes. Al heredar si se puede usar desde la clase derivada.

- **Tipo**

  Es el tipo de la variable, pudiendo ser un tipo básico o un objeto.

A continuación, se relacionan los tipos de datos que existen en Java:

Tabla 3. Tipos de datos en Java

Tipo	Formato	Descripción
Byte	8 bit	Entero de un byte

Tipo	Formato	Descripción
Short	16 bit	Entero corto
Int	32 bit	Entero
Long	64 bit	Entero largo
Float	32 bit	Flotante simple
Double	64 bit	Flotante doble
Char	16 bit	Un solo carácter

El nombre que se asigna a una variable, conocido como nombreVariable, debe adherirse a ciertas convenciones en Java:

a) Los nombres de las variables deben comenzar con una letra minúscula. En el caso de nombres compuestos, se emplea la nomenclatura camelCase, donde la primera letra de cada palabra subsiguiente es mayúscula.
b) Los nombres de variables no pueden contener espacios, iniciar con números ni incluir caracteres especiales.
c) No se permite que una variable tenga el mismo nombre que una palabra clave reservada por el lenguaje.
d) Dentro de un mismo ámbito, cada variable debe tener un nombre único y no puede repetirse el nombre de otras variables previamente declaradas.

A continuación, se proporcionan ejemplos que siguen estas convenciones:

Ejemplo variable datos Java

primerNombre

apellidoCliente

edad

La explicación indica que se puede inicializar una variable con un valor utilizando el formato = valorInicial;

Ejemplo variable datos Java:

Se permite definir más de una variable, separándolas por coma, por ejemplo:

public int a = 5, b, c = 4;

Ejemplos de declaración de atributos o variables:

public int edad=5;

private char genero='M';

private double estatura;

Nota: todos los atributos terminan en punto y coma.

Los métodos se definen la siguiente manera:

[modifVisibilidad] tipo nombreMetodo (listaParámetros) {}

En los atributos y métodos para las clases también existen:

- **Modificador de visibilidad**

  Aplica las mismas normas que para los atributos. Indica el nivel de acceso: público, privado o protegido.

- **Tipo**

  Corresponde al tipo de valor que el método retorna, como int, double, char, String, etc.

- **Nombre del método**

  Debe ser un verbo en minúscula o un compuesto iniciando con mayúscula en sustantivos, ej. revisarCuenta().

- **Lista de parámetros**

  Incluye los parámetros entre paréntesis; son los datos que recibe el método, y puede ser vacía.

- **Cuerpo del método {}**

  Contiene el código a ejecutar encerrado en llaves; no termina en punto y coma como los atributos.

  A continuación, se presentan algunos ejemplos de métodos:

Ejemplo método atributos y métodos:

```
public int obtenerEdad ()
{
   return edad;
}
```

Este es un método público que retorna un valor entero, cuyo nombre es obtenerEdad, no recibe parámetros y devuelve la edad.

Ejemplo listado de parámetros:

Ejemplo método:

```
public void cambiarEdad (int laEdad)
{
   edad = laEdad;
```

}

Este es un método público que no retorna ningún valor porque tiene la palabra reservada `void`, su nombre es cambiarEdad y recibe un parámetro de tipo entero, en el cuerpo del método asigna a la variable edad el valor del parámetro que viene almacenado en la variable laEdad.

2.2. Constructores y destructores

El constructor de una clase es un método diseñado específicamente para inicializar los objetos de dicha clase. Por otro lado, un destructor es un método que se invoca automáticamente cuando el objeto es destruido. Se utiliza un constructor al instanciar un objeto para inicializar sus variables con valores adecuados. Un constructor, que es un método especial de la clase, tiene las siguientes características:

El nombre del constructor coincide con el nombre de la clase.

- No retorna ningún valor.
- Es posible definir múltiples constructores, siempre que tengan diferentes listas de parámetros.
- El código dentro de un constructor generalmente incluye inicializaciones de variables y objetos para que el objeto sea creado con los valores iniciales deseados.

La sintaxis para definir un constructor es la siguiente:

```
[modificadorDeVisibilidad]

NombreDelConstructor(listaDeParámetros) {

  // Código de inicialización
```

}

Donde:

- modificadorDeVisibilidad indica si el constructor es public, private o protected, siguiendo las mismas normas que para los atributos y métodos.
- NombreDelConstructor debe coincidir con el nombre de la clase.
- listaDeParámetros es la enumeración de parámetros que el constructor acepta, separados por comas y definidos por su tipo y nombre, por ejemplo, (int a, int b).

En el ejemplo de la clase Producto, se crea un constructor público llamado Producto con una lista de parámetros. Este constructor asigna los valores de los parámetros a las variables de instancia de la clase. La palabra reservada this se utiliza para referirse a los miembros de la propia clase en el objeto actual.

- **Un constructor**

  Sin parámetros se denomina constructor vacío. Si no se define explicitamente un constructor, el compilador proporciona uno por defecto, permitiendo la creación de objetos sin argumentos mediante new Clase{}.

- **Los destructores**

  Son métodos que liberan la memoria ocupada por un objeto. En Java, no existen los destructores como tal debido al recolectar la basura de la máquina virtual de Java, que administra la memoria de forma automática, liberándola cuando detecta objetos que ya no se utilizan.

Aunque el manejo de memoria en Java es automático gracias al recolector de basura, los usuarios pueden sugerir la ejecución del recolector de basura mediante una

instrucción específica, aunque el momento exacto de la recolección de basura no es garantizado.

2.3. Métodos accesores y modificadores

En la programación orientada a objetos, existen dos tipos fundamentales de modificadores de acceso para los miembros de una clase: público (`public`) y privado (`private`).

- **Público (`public`)**

  Un miembro público es accesible desde cualquier parte del código, incluso fuera de su clase de definición. Esto permite que cualquier otra clase o componente del programa interactúe con el miembro público sin restricciones. El acceso es universal, permitiendo que cualquier otro código, independientemente de su paquete o clase, utilice el miembro público.

- **Privado (`private`)**

  Un miembro privado solo puede ser accedido por otros métodos dentro de la misma clase. Esto se utiliza para controlar la exposición y manipulación de los datos internos del objeto, evitando el acceso o modificación directa por parte de código externo a la clase. El acceso está estrictamente limitado a la clase donde el miembro privado es declarado. Otros componentes del programa no pueden acceder directamente a estos miembros privados.

Hay cuatro tipos de modificadores de acceso disponibles en Java:

- **Predeterminado (`default`)**

Cuando no se especifica un modificador de acceso para una clase, método o miembro de datos, se aplica el modificador de acceso predeterminado por defecto. Los miembros de datos, clases o métodos con acceso predeterminado son accesibles solo dentro del mismo paquete.

Por ejemplo, la clase Producto y el método mostrar, al tener modificador de acceso predeterminado, no pueden ser accedidos desde otro paquete.

- **Privado (`private`)**

  Se indica con la palabra clave "private". Los métodos o miembros de datos declarados como privados son accesibles únicamente dentro de la clase donde se declaran.

  Ninguna otra clase del mismo paquete puede acceder a los miembros privados.

  Las clases e interfaces no pueden declararse como privadas.

- **Protegido (`protected`)**

  Se especifica con la palabra clave "`protected`". Los métodos o miembros de datos declarados como "`protected`" son accesibles dentro del mismo paquete o en subclases situadas en paquetes diferentes.

  Accesible dentro del mismo paquete y por subclases en paquetes diferentes, promoviendo así una forma de herencia protegida.

- **Público (`public`)**

  Se indica con la palabra clave "`public`". Este modificador de acceso tiene el alcance más amplio de todos.

  Las clases, métodos o miembros de datos declarados como públicos son accesibles desde cualquier lugar del programa, sin restricciones en el alcance. Por ejemplo, declarar el método mostrar de la clase Producto

como público permite que se acceda a este método desde cualquier parte del programa.

Métodos accesores (get, set)

Los métodos accesores permiten obtener o modificar los atributos de un objeto y se clasifican en dos tipos: "get" y "set".

- Los métodos "get" permiten acceder al valor de un atributo.
- Los métodos "set" permiten modificar el valor de un atributo.

El nombre de estos métodos comienza con "get" o "set", seguido del nombre del atributo con la primera letra en mayúscula. Su modificador de acceso es siempre público para facilitar el acceso o modificación de los atributos desde fuera de la clase. Por ejemplo, "getNombre" o "setNombre".

Ejemplo de sintaxis del método "get":

```
public tipoDatoAtributo getAtributo() {
   return atributo;
}
```

Ejemplo de sintaxis del método "set":

```
public void setAtributo(tipoDatoAtributo variable) {
   this.atributo = variable;
}
```

Ejemplo implementación de métodos get y set en Java:

```
public class Producto { // Clase
```

```
    private int codigo; // atributos
    private String nombre; // atributos

    public int getCodigo() { // método get del atributo codigo
        return codigo; // obtiene el valor del atributo codigo
    }

    public void setCodigo(int codigo) { // método set del atributo codigo
        this.codigo = codigo; // cambia el valor del atributo codigo
    }

    public String getNombre() { // método get del atributo nombre
        return nombre; // obtiene el valor del atributo nombre
    }

    public void setNombre(String nombre) { // método set del atributo nombre
        this.nombre = nombre; // cambia el valor del atributo nombre
    }
}
```

En el ejemplo se presenta la clase Producto con sus atributos código y nombre, por cada uno de estos atributos encontramos sus métodos `get` y `set` para obtener y cambiar el valor de cada uno de ellos.

2.4. Sobrecarga de métodos (overload)

La sobrecarga de métodos permite definir múltiples constructores o métodos con el mismo nombre dentro de una clase, bajo la condición de que estos no compartan la misma cantidad y tipo de parámetros. En esencia, la sobrecarga facilita la declaración de métodos homónimos que difieren en los parámetros que aceptan. Por lo tanto, no es posible tener dos métodos con idéntico nombre y parámetros coincidentes. En este contexto, los argumentos proporcionados al llamar al método determinan cuál de ellos se ejecuta.

Ejemplo implementación de sobrecarga de constructores y métodos en Java:

```
public class Producto { // Clase

    private int codigo; // atributos

    private String nombre; // atributos

    private double precio; // atributos


    public Producto() {

    }


    public Producto(int codigo, String nombre, double precio) {
```

```
        this.codigo = codigo;
        this.nombre = nombre;
        this.precio = precio;
    }

    public void descontarProducto(double precio, String nombre) {
        double descuento = precio * 0.10;
        System.out.println("El producto " + nombre + " tiene descuento");
    }

    public void descontarProducto(int precio) {
        double descuento = precio * 0.10;
    }
}
```

El ejemplo presenta la sobrecarga tanto de constructores como de métodos. Se evidencia sobrecarga en los constructores de la clase Producto, donde se han creado dos constructores con el mismo nombre pero con distintos parámetros. Una situación similar ocurre con el método descontarProducto, que se define dos veces, cada uno con diferentes parámetros.

3. Comunicación entre clases

Las clases, así como los objetos, no operan de manera aislada. La Programación Orientada a Objetos (POO) busca simular las aplicaciones del mundo real de la manera más precisa posible, lo cual implica reflejar las relaciones entre clases y objetos de forma adecuada. De este modo, la comunicación entre clases se facilita exclusivamente a través de sus interfaces públicas.

Existen tres tipos fundamentales de relaciones entre objetos:

- **Asociación**

  Describe cómo los objetos se relacionan y se utilizan entre sí.

- **Agregación/Composición**

  Define una relación en la que un objeto está compuesto por uno o más objetos, mostrando una relación de tipo 'parte-todo'.

- **Generalización/Especialización (Herencia)**

  Expresa una jerarquía entre clases donde una clase derivada (subclase) hereda comportamientos y atributos de una clase base (superclase), permitiendo la reutilización y extensión de código.

Implementación de asociaciones

En Java, la implementación de la asociación se lleva a cabo mediante campos de instancia. Esta relación puede ser bidireccional, donde cada clase posee una referencia hacia la otra. Tanto la agregación como la composición son variantes específicas de relaciones de asociación. A continuación, se detallan estos tipos de relaciones:

- **Asociación unidireccional**

  Una asociación unidireccional ocurre cuando una clase está vinculada a otra, pero solo una de ellas conoce la existencia de la otra. Esto significa que un objeto de la primera clase puede acceder a los miembros de la segunda clase, pero no a la inversa.

  Para implementar una asociación unidireccional en Java, se debe incluir una referencia en la clase que conoce a la otra. Esto se logra declarando un atributo del tipo de la clase conocida dentro de la clase que posee el conocimiento.

  Por ejemplo, en una relación entre una clase Pedido y una clase Cliente, la clase Pedido puede tener una referencia a la clase Cliente para saber a quién pertenece, pero la clase Cliente no necesita tener una referencia a la clase Pedido si la lógica del negocio no requiere que un cliente conozca sus pedidos.

  Aquí está cómo se podría ver esto en el código:

```
public class Cliente {
    private String nombre;
    // Otros atributos y métodos del Cliente
}


public class Pedido {
    private Cliente cliente; // Referencia unidireccional
    private double total;
```

```
    public Pedido(Cliente cliente) {
        this.cliente = cliente;
    }

    // Otros atributos y métodos del Pedido

}
```

En este caso, Pedido conoce a #Cliente, pero Cliente no tiene conocimiento de Pedido, lo que hace que la relación sea unidireccional desde Pedido hacia Cliente.

- **Asociación bidireccional**

  En una asociación bidireccional, dos clases están vinculadas entre sí de tal manera que ambas pueden conocer la existencia y utilizar los miembros de la otra. Esto significa que un objeto de una clase mantiene una referencia a uno o varios objetos de la otra clase y viceversa; ambos tienen la capacidad de interactuar entre sí.

  Para implementar una asociación bidireccional en Java, se deben crear instancias mutuas en ambas clases. Cada clase tendrá un atributo que referencia a la otra, y típicamente se proporcionarán métodos para establecer y modificar estas referencias de manera controlada.

Tomando como ejemplo una relación entre Cliente y Cuenta, no solo el Cliente tendría una Cuenta, sino que la Cuenta también tendría una referencia a su Cliente para formar una relación bidireccional completa.

Aquí se muestra cómo se podría representar en el código:

```
public class Cliente {
    private String nombre;
    private Cuenta cuenta; // Referencia bidireccional a Cuenta

    // Constructor, getters y setters

    public void vincularCuenta(Cuenta cuenta) {
        this.cuenta = cuenta;
        cuenta.vincularCliente(this); // Establece la referencia bidireccional
    }
}

public class Cuenta {
    private double saldo;
    private Cliente cliente; // Referencia bidireccional a Cliente
```

```
    // Constructor, getters y setters


    public void vincularCliente(Cliente cliente) {
        this.cliente = cliente;
        // Si es necesario, también establecer la referencia desde Cliente a Cuenta
        if (cliente.getCuenta() != this) {
            cliente.vincularCuenta(this);
        }
    }
}
```

En este código, tanto la clase Cliente como la clase Cuenta tienen métodos que permiten vincular una a la otra. Es importante manejar estas referencias cuidadosamente para evitar bucles infinitos o inconsistencias en las relaciones. Cuando se establece la vinculación desde una instancia de Cliente a Cuenta, también se llama a un método en Cuenta para establecer la vinculación inversa, asegurando así una asociación bidireccional coherente.

Implementación de composición/agregación

La agregación es una relación entre clases que modela una jerarquía de tipo "todo-parte", donde las partes pueden existir independientemente del todo. En cambio, la composición es una forma específica de agregación en la que las partes están exclusivamente asociadas a un único todo y su existencia depende de ese todo.

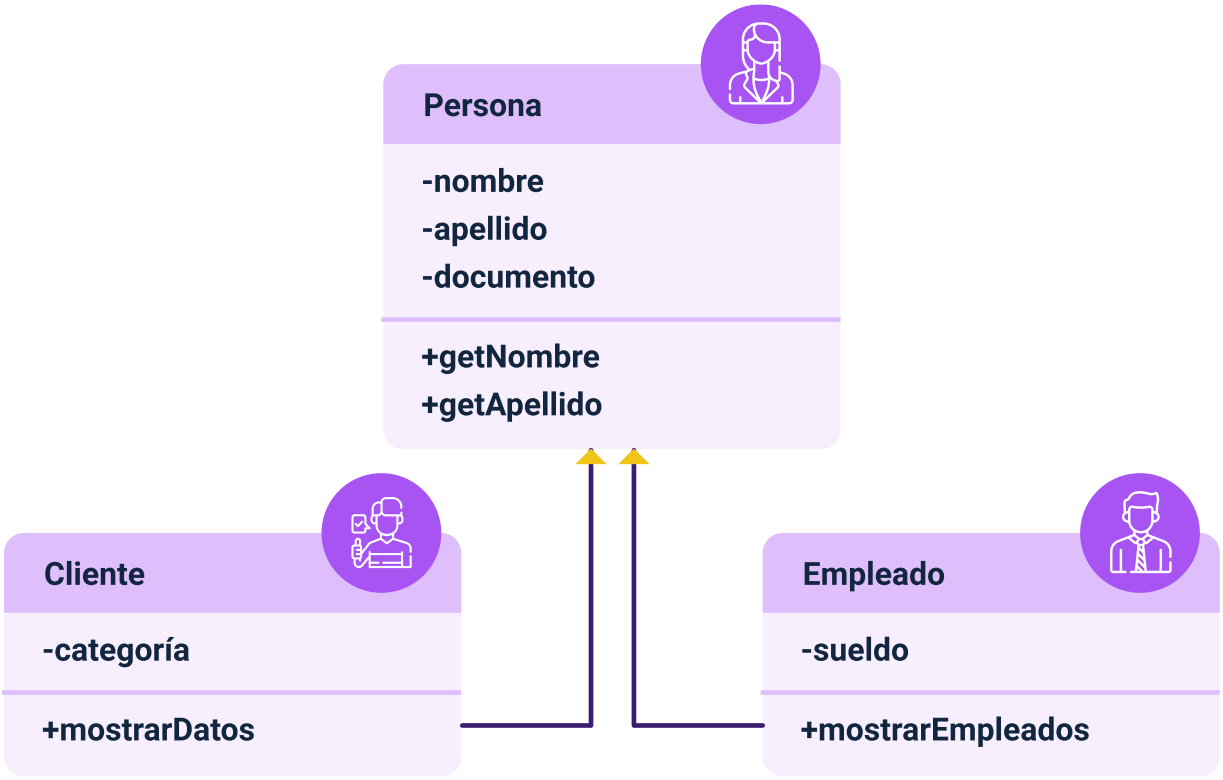
En Java, la composición se implementa creando una clase que contiene objetos de otras clases como atributos privados, inicializándolos generalmente dentro del constructor de la clase contenedora para asegurar el acoplamiento de sus ciclos de vida.

Implementación de herencia

La herencia es el mecanismo en la programación orientada a objetos que permite que una clase, conocida como subclase, herede propiedades y comportamientos de otra clase, llamada superclase.

Por ejemplo, a partir de la clase Persona, es posible definir una subclase Empleado, que hereda las características de Persona. La herencia permite la reutilización de código y la extensión de funcionalidades. En Java, la herencia se implementa mediante la palabra clave `extends`, lo que indica que la subclase Empleado adquiere los métodos y atributos accesibles de la clase Persona.

Figura 2. Ejemplo de herencia en UML



La superclase Persona posee los atributos nombre, apellido y documento, que son igualmente necesarios para las clases Cliente y Empleado. La herencia permite la reutilización de código, de modo que las clases derivadas no necesitan declarar nuevamente estos atributos, ya que los heredan directamente de la clase Persona.

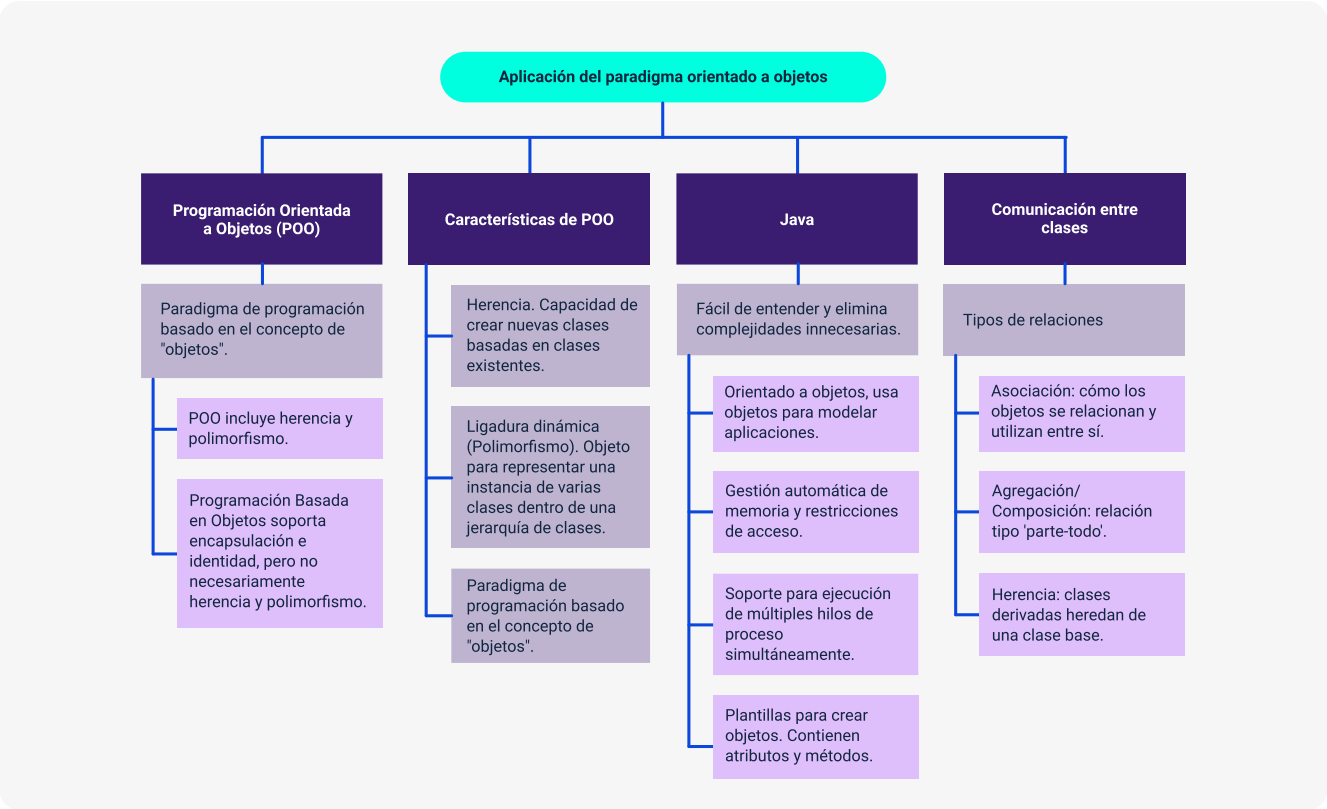
Para implementar la relación de herencia del ejemplo que incluye a las clases Persona, Cliente y Empleado en Java, se procedería de la siguiente manera:

Se crea la clase Persona, la superclase, con sus atributos, constructores y métodos “get” y “set”.

Se crea la clase Cliente con su propio atributo categoría y su método mostrarDatos. Para heredar los atributos de la superclase Persona, se utiliza la palabra reservada “extends”.

Síntesis

A continuación, se presenta una síntesis de la temática estudiada en el componente formativo.



Material complementario

Tema	Referencia	Tipo de material	Enlace del recurso
Implementación de clases y objetos java	Códigos de Programación - MR. (2017). Curso Java - 7: Clases, Métodos y Objetos.	Video	https://youtu.be/Z6ispQvMy8s?feature=shared
Sobrecarga de métodos (overload)	Códigos de Programación - MR. (2017). Curso Java - 8: Constructores y Sobrecarga de Métodos.	Video	https://www.youtube.com/watch?v=_DUSaU58GDk&ab_channel=C%C3%B3digosdeProgramaci%C3%B3n-MR
Comunicación entre clases	TodoCode. (2022). RELACIONES entre CLASES en JAVA POO.	Video	https://www.youtube.com/watch?v=XKk5k9OrAUg&ab_channel=TodoCode

Glosario

Agregación: tipo especial de asociación que representa una relación "parte de" donde las partes pueden existir independientemente del todo.

Asociación: relación entre dos objetos en la que uno utiliza las capacidades o servicios del otro.

Bytecode: código intermedio más abstracto que el código máquina, que es ejecutado por la JVM.

Clases: estructuras fundamentales de la POO que definen las propiedades y comportamientos de los objetos.

Composición: forma de agregación con una relación más fuerte donde las partes no pueden existir independientemente del todo.

Herencia: característica de la POO que permite que una clase derive de otra, heredando sus métodos y atributos.

Java: lenguaje de programación orientado a objetos, independiente de la plataforma, que compila a `bytecode`, el cual se ejecuta en la Máquina Virtual de Java (JVM).

JVM (Máquina Virtual de Java): entorno de ejecución para programas Java que permite la portabilidad del código entre diferentes plataformas.

Objetos: instancias de clases que contienen datos y comportamientos definidos por su clase.

Polimorfismo (Ligadura Dinámica): capacidad de una variable de tipo base para referirse a objetos de tipos derivados, permitiendo que se ejecuten diferentes métodos a través de una interfaz común.

POO (Programación Orientada a Objetos): paradigma de programación que utiliza objetos y sus interacciones para diseñar aplicaciones y programas informáticos.

Referencias bibliográficas

Archivo General de la Nación (2022). Guía para la formulación de un esquema de metadatos para la gestión de documentos. https://www.archivogeneral.gov.co/sites/default/files/Estructura_Web/5_Consulte/Recursos/Publicacionees/GuiaDeMetadatos.pdf

Instituto Colombiano de Normas Técnicas y Certificación. (2022). Normas técnicas de la información y seguridad de la familia ISO 27000. Icontec. https://www.icontec.org/servicio-educacion/modulares-de-educacion/

Ministerio de Educación. (2022). Guía para la clasificación de la información. https://www.mineducacion.gov.co/1759/articles-407695_galeria_14.pdf

Ministerio de Tecnologías de la Información y Comunicaciones - MinTIC. (2022). Entidades del sector. MinTIC. https://www.mintic.gov.co/portal/inicio/Ministerio/Entidades-del-sector/

Créditos

Nombre	Cargo	Centro de Formación y Regional
Milady Tatiana Villamil Castellanos	Responsable del Ecosistema	Dirección General
Olga Constanza Bermúdez Jaimes	Responsable de Línea de Producción	Centro de Servicios de Salud - Regional Antioquia
Zulema León Escobar	Experta Temática	Grupo De Apoyo Administrativo - Regional Distrito Capital
Paola Alexandra Moya Peralta	Evaluadora Instruccional	Centro de Servicios de Salud - Regional Antioquia
Juan Daniel Polanco Muñoz	Diseñador de Contenidos Digitales	Centro de Servicios de Salud - Regional Antioquia
Jhon Jairo Urueta Álvarez	Desarrollador Fullstack	Centro de Servicios de Salud - Regional Antioquia
Edgar Mauricio Cortés García	Actividad Didáctica	Centro de Servicios de Salud - Regional Antioquia
Luis Gabriel Urueta Álvarez	Validador de Recursos Educativos Digitales	Centro de Servicios de Salud - Regional Antioquia
Margarita Marcela Medrano Gómez	Evaluador para Contenidos Inclusivos y Accesibles	Centro de Servicios de Salud - Regional Antioquia
Daniel Ricardo Mutis Gómez	Evaluador para Contenidos Inclusivos y Accesibles	Centro de Servicios de Salud - Regional Antioquia