Semestre 2 - Programación orientada a objetos

Introducción - Lenguajes y programación (mar. 26 ago. 2025)

Diferencias entre lenguajes: Bajo y alto nivel

- **C:** es de bajo nivel, orientado al bajo nivel y el rendimiento en entorno crítico, se usa en controladores, SOs, sistemas embebidos. Es compilado, por lo que pasa a binario antes de su uso.
- **C++:** introduce programación orientada a objetos a C, se utiliza en desarrollo de videojuegos, sistemas operativos, aplicaciones, simulaciones, etc. También es compilado como C.
- **Java:** menos abstracto y más portable y automático que C y C++, introduce el paradigma híbrido para la ejecución usado la *Java Virtual Machine* y el enfoque completo hacia la POO. Usado en aplicaciones empresariales, software, backend y desarrollo antiguo en Android.
- **Python:** es un lenguaje de muy alto nivel, multiparadigma (lo que significa que admite diferentes métodos de programación, como POO o programación imperativa). Es muy legible y rápido de desarrollar, aunque es más lento. Se usa en ciencia de datos, IA, scripting, automatizaciones y desarrollo de prototipos.

Dentro de estos lenguajes, C y C++ son lenguajes de **bajo nivel**, lo cual quiere decir que son más cercanos al hardware y por ende más rápidos y manuales. Mientras que Java y Python son de **alto nivel**, es decir, más cerca del programador.

Comparativa

Lenguaje	C	C++	Java	Python
Enfoque	Basado en hardware, control manual	Introducir POO a la programación de alto rendimiento de C	Seguridad, portabilidady control menos manual	Facilidad, legibilidad y velocidad de desarrollo
Paradigma principal	Procedural	POO	POO	Multiparadigma
Ofrece POO	No	Si	Si	Si
Fortaleza	Rendimiento, control y eficiencia	Rendimiento, flexibilidad, mayor nivel	Seguridad, eficiencia, portabilidad	Automatizacion, simplicidad y ecosistema
Debilidad	Gran curva de aprendizaje, poca portabilidad, lentitud de desarrollo	Curva de aprendizaje, menor rendimiento que C, baja portabilidad	Verbosidad, peso y curva de aprendizaje	Lentitud, menos eficiencia

Lenguaje	C	C++	Java	Python
Dificultad	Alta	Alta	Media	Baja

Semejanzas entre todos: Son multiplataforma en mayor o menor medida al ser menos dependientes de manejo exacto de componentes en hardware. Se usan en aplicaciones de gran peso en diferentes áreas tecnológicas. Están orientados a la programación por humanos.

Diferencia entre todas: Su rendimiento, enfoque, paradigmas y aplicaciones varían entre lenguajes, desde la velocidad y eficiencia de C++ hacia la practicidad algo lenta de Python. Se usan en casos muy diferentes.

Algoritmos

Un algoritmo es un conjunto de pasos **finito, determinista y preciso** para completar un objetivo o realizar una acción. Se puede definir de formas diferentes, como por ejemplo:

Pseudocódigo

Es una descripción al estilo del código, que usa lenguaje natural y lógica básica para explicar un algoritmo, usando las estructuras y sintaxis de lenguajes reales sin especificación.

Ejemplo de pseudocódigo para sumar una lista de números

```
ALGORITMO calcular_suma

ENTRADA: una lista de números `lista_numeros`
SALIDA: la suma de los números en la lista

1. INICIALIZAR una variable `suma_total` a 0.
2. PARA CADA `numero` EN `lista_numeros`:
3.     AÑADIR `numero` a `suma_total`.
4. DEVOLVER `suma_total`.

FIN ALGORITMO
```

Puede y debe ser el primer paso a la hora de hacer la programación de un proceso a fin de hacerlo entendible y ordenado.

Diagramas de flujo

Son diagramas que esquematizan el flujo de un programa usando un diagrama ordenado donde cada paso es un componente. Los diagramas de flujo son estandarizados y se usan de forma alternativa o complementaria al pseudocódigo.

UML - Lenguaje unificado de modelado

Es un lenguaje unificado que fue creado en los 90 para estandarizar la representación de sistemas y componentes, en la documentación y desarrollo de sistemas informáticos. Se usan diferentes tipos, incluyendo diagramas de clases, máquinas de estados y procesos, por ejemplo.

Componentes de programación y su uso en Java

Estructuras de control

Los programas siguen generalmente un proceso **secuencial**, donde cada paso puede también ser una estructura de control, como los condicionales `if` o los bucles `while`.

> ⚠ **Algunas especificaciones de Java**
>
> 1. Es sensible a la sintaxis, mas no a la identación
> 2. Todas las instrucciones terminan en punto y coma
> 3. Java es sensible a las mayúsculas
> 4. Los bloques van dentro de llaves `{}`
> 5. Puedes comentar con doble barra `//`. Los comentarios no se ejecutan

> ☰ **Ejemplo**
>
> Condicional simple
>
> ```
> if (grade >= 60) {
>     System.out.println("Passed");
> } else {
>     System.out.prinln("Not passed");
> }
> ```
>
> Equivalente en **operador ternario**
>
> ```
> System.out.println(grade >= 60? "Passed": "Not passed");
> ```

Operadores

Operadores Aritméticos

- Suma: `+`
- Resta: `-`
- Multiplicación: `*`

- División: `/`
- Módulo (residuo): `%`

Operadores Relacionales

- Igual a: `==`
- Diferente de: `!=`
- Mayor que: `>`
- Menor que: `<`
- Mayor o igual que: `>=`
- Menor o igual que: `<=`

Operadores Lógicos y Condicionales

- Condicional AND: `&&`
- Condicional OR: `||`
- Negación lógica: `!`
- OR a nivel de bits (booleano inclusivo): `|`
- XOR a nivel de bits (booleano exclusivo): `^`

Asignaciones

- Asignación simple: `c = 3`
- Asignación con operación: `c += 3` (también aplica para `-=`, `*=`, `/=`, `%=`)
- Incrementador posfijo: `i++`
- Decremento prefijo: `--b`

> **Sobre los incrementadores de prefijo y posfijo**
>
> La diferencia principal entre los operadores de pre-incremento/decremento (`++i`, `--i`) y post-incremento/decremento (`i++`, `i--`) es el momento en que se realiza la actualización de la variable y el valor que devuelve la expresión.
>
> - **Prefijo (`++i`, `--i`):** La variable se incrementa o decrementa *antes* de que se evalúe la expresión. El valor que se utiliza en la expresión es el valor *después* de la modificación.
> - **Posfijo (`i++`, `i--`):** La variable se incrementa o decrementa *después* de que se evalúa la expresión. El valor que se utiliza es el *original* de la variable, antes de la modificación.

Tipos primitivos:

- Enteros: `int`
- Flotantes: `float` y `double`
- Enteros de tamaños: `short`, `long` y `byte`

- Texto: `char`
- Valores booleanos: `boolean`

Introducción a la programación orientada a objetos (jue. 28 ago. 2025)

Sobre la programación orientada a objetos

Mientras la **programación estructurada** usa estructuras de control y funciones para el desarrollo de algoritmos y programas, la **programación orientada a objetos** aprovecha conceptos como clases, objetos, y demás en los mismos procesos, al modelar componentes como elementos virtuales, u **objetos**, que pueden abstraerse en componentes base que son las **clases**.

Objetos

Los objetos con instancias de una *clase*, que tienen **atributos**, que son sus características, y **métodos** que son sus acciones.

> **Ejemplo**
>
> Un carro tiene atributos como su marca y color, y métodos como arrancar o frenar. En código:

```
// El carro tiene el color rojo, sin velocidad y aceleración de 8 unidades
// También tiene un método que aumenta la velocidad

// Atributos
float Carro.vel = 0;
float Carro.acel = 8;

// Método para acelerar
float Acelerar(float vel, float acel) {
    vel += acel;
    return vel;
}

// Llama al método
float newVel = Acelerar(Carro.vel, Carro.acel);
```

Clases

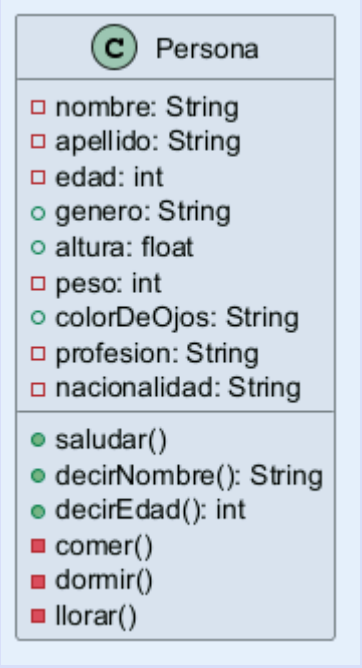
Una **clase** es un *molde* de la cual se pueden derivar más objetos al *instanciarlos*. Por ejemplo, una base de marcador de la cual se fabrican los demás puede considerarse una clase, mientras que los marcadores fabricados son sus instancias.

Diagramas de clases en UML

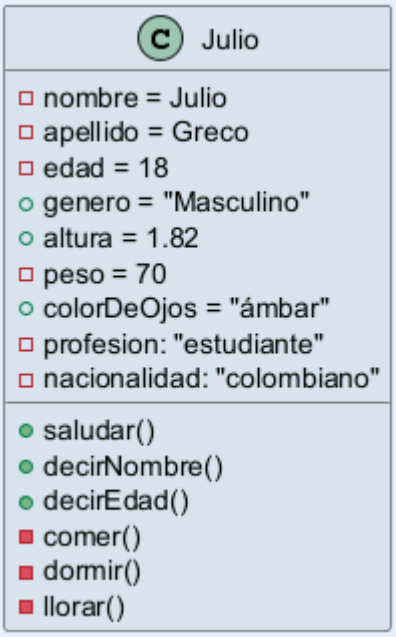
Un diagrama de clases en UML usa texto llano para representar clases, sus características y componentes, en texto llano o de forma gráfica.

Ejemplo

Para una persona, tenemos el siguiente diagrama:



En caso de instanciar un objeto, usamos un diagrama de objetos, que es parecido:



En Java, las clases se escriben con la siguiente notación:

```
// Las clases se indican con class, seguido del nombre de la clase en
PascalCase
class Car {
    // Atributos, en camelCase
    float speed = 0;

    // Método para aceleración. float es el tipo de dato que retorna
    float Acelerate(float speed, float acc) {
        speed += acc;
        return speed;
    }
}

// ... en el código principal
// Para acceder a un atributo, se usa la notación Clase.atributo
float currentSpeed = Car.speed;
float currentAcc = 10.0f;
float newSpeed = Acelerate(currentSpeed, currentAcc);
```

Sintaxis básica de Java (mar. 02 sep. 2025)

Tipos en Java

- Entero: `int`
- Decimales: `float` o `double` (más precisión)
- Valores lógicos: `boolean`
- Caracteres: `char`

Clases

Las clases se definen con la palabra clave `class` seguido por el nombre de la clase, unas llaves `{}` y el contenido de la clase en ellas:

```java
class HolaMundo {
    public static void main(String[] args) {
        System.out.println("Hola, POO en Java");
    }
}
```

> **`public static void main` y `System.out.println`**
>
> El `public static void main` es la parte principal de la clase y es la que se ejecuta primero, puede compararse con el `int main()` de C++ o el `if __name__ == __main__` de Python. Por otro lado, `System.out.println` escribe en pantalla, como el `print` de Python o el `cout` de C++.

Objetos

Los objetos usan la sintaxis `new [clase]` para instanciarse. Se usa la misma clase como tipo de dato:

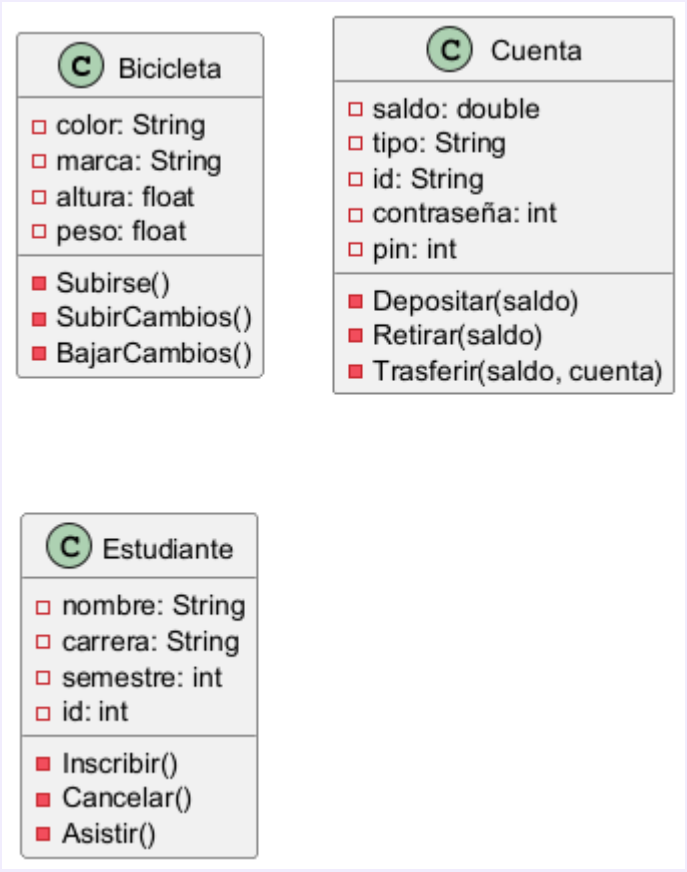
```java
// Todo va en una clase más grande
class Bus {
    String marca = "Volvo";
    int maxCapacidad = 200;
    float speed = 10f;

    int acelerar(speed) {
        return speed + 1;
    }
}

// Se instancia un bus de la clase Bus
Bus transmi = new Bus();
```

> **Ejemplos de objetos**

UML:



Código básico (con `Estudiante`):

```
class Estudiante {
    String nombre = "Juan";
    String carrera = "Ingeniería";
    int semestre = 2;
    int id = 1072072838;

    void Inscribir() {
        semestre += 1;
    }

    void Cancelar() {
        semestre = 0;
    }

    void Asistir() {
        System.out.println("¡Llegué a clases!");
    }
}
```

```java
// Ejemplo de instancia
Estudiante estudiante1 = new Estudiante();
```

Métodos en Java (jue. 04 sep. 2025)

Método `main`

El método `main` en Java se escribe de la siguiente manera:

```java
public static void main(String[] args) {
    // Código de la clase
}
```

- `public` significa que es accesible en otros lugares
- `static` significa que no depende de un objeto, sino que pertenece a la clase
- `void` quiere decir que no retorna nada
- `main(String[] args)` quiere decir que es la clase principal `main` y que recibe texto como argumento (por ejemplo, en la consola)

Métodos propios

Una clase contiene **métodos** que son funciones que están dentro. Se escriben como funciones de Java de la siguiente manera:

```java
// ... Dentro de una clase
int Sumar(int a, int b) {
    return a + b;
}
```

Ejemplo

Crea un programa que maneje una alcancía.

Este programa tiene los métodos `Depositar`, `Romper` y `CalcularEspacio` para manejar el espacio y saldo de la alcancía.

```java
// Clase principal
public class Alcancia {
   // main exige que los métodos sean static también
```

```java
    // Depositar suma dinero solo si no está rota la alcancía
    static int Depositar(int dinero, int saldo, boolean estaRoto) {
        if (estaRoto){
            System.out.println("Error: No puedes depositar en una alcancía
rota");
            return 0;
        }

        return saldo + dinero;
    }

    // Romper suelta el saldo solo si no está rota la alcancía
    static int Romper(int saldo, boolean estaRoto) {
        if (estaRoto) {
            System.out.println("Error: No puedes romper una alcancía ya
rota");
            return 0;
        }

        return saldo;
    }

    // Calcula cuanto espacio queda en la alcancía
    static int CalcularEspacio(int saldo, int capacidad, boolean estaRoto)
{
        if (estaRoto) {
            System.out.println("Error: No hay espacio");
            return 0;
        }

        return capacidad - saldo;
    }

    public static void main(String[] args) {
        Alcancia a = new Alcancia();

        // Atributos
        int saldo = 0;
        int capacidad = 1000;
        boolean estaRoto = false;

        // Datos iniciales
        System.out.println("Saldo inicial: " + saldo);
        System.out.println("Espacio restante: " + CalcularEspacio(saldo,
capacidad, estaRoto));
        System.out.println("----------");
```

```
        // Depositar 10 dólares
        System.out.println("Depositar 10 dólares");
        saldo = Depositar(10, saldo, estaRoto);
        System.out.println("Nuevo saldo: " + saldo);
        System.out.println("Espacio restante: " + CalcularEspacio(saldo,
capacidad, estaRoto));
        System.out.println("----------");

        // Romper la alcancía
        System.out.println("Rompimos la alcancía");
        saldo = Romper(saldo, estaRoto);
        System.out.println("Dinero extraido: " + saldo);
        System.out.println("Espacio restante: " + CalcularEspacio(saldo,
capacidad, estaRoto));
        System.out.println("----------");
        estaRoto = true;

        // Depositar 10 dólares otra vez
        // No funcionará porque la alcancía está rota
        System.out.println("Depositar otros 10 dólares");
        saldo = Depositar(10, saldo, estaRoto);
        System.out.println("Nuevo saldo: " + saldo);
        System.out.println("Espacio restante: " + CalcularEspacio(saldo,
capacidad, estaRoto));
        System.out.println("----------");

        // Romper la alcancía otra vez
        // No funcionará porque la alcancía está rota
        System.out.println("Rompimos la alcancía");
        saldo = Romper(saldo, estaRoto);
        System.out.println("Dinero extraido: " + saldo);
        System.out.println("Espacio restante: " + CalcularEspacio(saldo,
capacidad, estaRoto));
        System.out.println("----------");
    }
}
```

Estructuras de control y arreglos (mar. 09 sep. 2025)

Estructuras de control

Permiten modificar el flujo de ejecución de un programa.

Condicionales

`if else`

Se utiliza para ejecutar un bloque de código si una condición es verdadera y, opcionalmente, otro bloque si es falsa.

```
int edad = 18;

if (edad >= 18) {
    System.out.println("Es mayor de edad.");
} else {
    System.out.println("Es menor de edad.");
}
```

Los `if else` también se pueden anidar, aunque no es lo más lógico:

```
if (edad >= 18) {
    System.out.println("Es mayor de edad.");
} else {
    if (edad >= 13) {
        System.out.println("Es adolescente.");
    } else {
        System.out.println("Es un niño.");
    }
}
```

`if else if else`

Permite evaluar múltiples condiciones en secuencia.

```
int nota = 85;

if (nota >= 90) {
    System.out.println("Calificación: A");
} else if (nota >= 80) {
    System.out.println("Calificación: B"); // Esta se ejecutará
} else {
    ...
}
```

`switch`

Es útil para seleccionar uno de varios bloques de código a ejecutar según el valor de una variable. Funciona con tipos de datos como `byte`, `short`, `char`, `int`, `enum`, `String` y algunas clases envolventes.

```
int dia = 4;
String nombreDelDia;

switch (dia) {
    case 1:
        nombreDelDia = "Lunes";
        break;
    case 2:
        nombreDelDia = "Martes";
        break;
    ...
    default:
        nombreDelDia = "Día inválido";
        break;
}
System.out.println("Hoy es " + nombreDelDia); // Imprimirá "Hoy es Jueves"
```

Bucles (Loops)

`for`

Ejecuta un bloque de código un número específico de veces. Es ideal cuando sabes de antemano cuántas iteraciones necesitas.

```
for (int i = 0; i < 5; i++) {
    System.out.println("El valor de i es: " + i);
}
```

`while`

Repite un bloque de código mientras una condición sea verdadera. La condición se evalúa *antes* de cada iteración.

```
int contador = 0;

while (contador < 5) {
    System.out.println("Contador: " + contador);
    contador++; // Importante para no crear un bucle infinito
}
```

`do-while`

Similar al `while`, pero el bloque de código se ejecuta al menos una vez, ya que la condición se evalúa *después* de la iteración.

```
int i = 0;
do {
    System.out.println("El valor de i es: " + i);
    i++;
} while (i < 5);
```

Arreglos

Los arreglos son listas de valores asociados a una variable. Para definirlos usamos la sintaxis:

```
// Arreglo de 12 enteros
int c[] = new int[12];

// Arreglo de 10 flotantes, declarado en dos líneas
float d[];
d = new float[10];

// Cantidad de elementos de la lista, calculado con el atributo ".length"
dl = d.length;
```

Los elementos de los arreglos de indexan desde 0, por lo que `c[0]` es el primer elemento de `c`.

También es posible inicializar los arreglos con valores directamente en la declaración:

```
// Arreglo de enteros inicializado con valores
int numeros[] = {10, 20, 30, 40, 50};

// Arreglo de Strings
String nombres[] = {"Juan", "Ana", "Pedro"};
```

Arreglos multidimensionales

Los arreglos multidimensionales son arreglos anidados, los cuales permiten **múltiples dimensiones** por ende. Su notación es la siguiente:

```
// Arreglo de 10 * 10 elementos enteros
int m[][] = new int[10][10];
```

```java
// Primer elemento del arreglo
int p = m[0][0];
```

Estructura y utilidades de Java (jue. 11 sep. 2025)

Estructura y utilidades del código en Java

> **Características de Java**
>
> Java es un lenguaje orientado a objetos, multiplataforma y de estándar abierto para todo el mundo. Además contamos con diferentes acrónimos para los componentes de Java:
>
> 1. **JVM (Java Virtual Machine):** Es un motor que proporciona un entorno de ejecución para ejecutar el código Java o las aplicaciones. Convierte el bytecode de Java en lenguaje de máquina. La JVM es parte del JRE.
> 2. **JRE (Java Runtime Environment):** Es un conjunto de herramientas de software para el desarrollo de aplicaciones Java. Contiene la JVM, las clases principales y las bibliotecas de soporte.
> 3. **JDK (Java Development Kit):** Es un kit de desarrollo de software que se utiliza para desarrollar aplicaciones Java. Contiene el JRE y herramientas de desarrollo como el compilador (`javac`) y el depurador.

La estructura de un programa en Java incluye las siguientes normas y utilidades:

4. Solo debe haber una clase por archivo. Se define como:

> **Ejemplo**
>
> ```java
> public class MiClase {
>     // Código
> }
> ```

- `public` quiere decir que es público y es accesible. También podemos hacerlo privado.
- `class` hace la clase con el nombre `MiClase` . Los nombres deben ser cortos, con primera letra mayúscula y un nombre descriptivo.

4. Las clases tienen métodos, incluyendo el `main` si es el programa principal. Los métodos tienen las mismas convenciones de nombres, solo que la primer letra debe ser en minúscula.

Ejemplo

```
public static void main(String[] args) {
    // Código
}
```

4. Los comentarios ayudan a mejorar la legibilidad y la lógica del código. Comentar ayuda a mejorar el trabajo para todos. Empiezan con `//`

Ejemplo

```
// Comentario útil
```

4. Los operadores tienen la siguiente precedencia:

Signos	Orden de asociación	Tipo
`*` `/` `%`	Izquierda a derecha	Multiplicativo
`+` `-`	Izquierda a derecha	Aditivo
`<` `>=` `>` `>=`	Izquierda a derecha	Relacional
`==` `!=`	Izquierda a derecha	Igualdad
`=`	Derecha a izquierda	Asignación

Los operadores con la misma precedencia se aplican en el orden de asociación.

4. Puedes hacer salidas en consola con `System.out`, que es el estándar para ello:
   - `println` imprime una línea de forma literal
   - `printf` imprime con formato, pasándole valores
5. Existen secuencias de escape para escribir caracteres especiales:
   - `\n`: salto de línea
   - `\t`: tabulación
   - `\r`: retorno de carro. Devuelve el cursor al inicio de la línea y la sobrescribe con lo que siga
   - `\\`: backslash
   - `\"` y `\'`: comillas
   - Entre otros

Ejemplo

```java
// \r hace que la próxima vez se sobreescriba la misma línea. %d adquiere el valor de val
System.out.printf("Progreso: %d\r", val);
```

4. Puedes usar `System.in` para hacer entradas, por ejemplo con `scanner` para pedir en consola. Importa `java.util.Scanner` para que funcione

Ejemplo

```java
import java.util.Scanner; // Importar la clase Scanner

// ... En el código principal
// Crear un Scanner para leer la entrada de la consola
Scanner scanner = new Scanner(System.in);

// Pedir un nombre al usuario
System.out.print("Introduce tu nombre: ");
String nombre = scanner.nextLine(); // Leer una línea de texto

// Pedir la edad al usuario
System.out.print("Introduce tu edad: ");
int edad = scanner.nextInt(); // Leer un número entero

// Mostrar la información recibida
System.out.printf("Hola, %s. Tienes %d años.\n", nombre, edad);

// Cerrar el scanner para liberar recursos
scanner.close();
```

Métodos de salida

Comando `printf`

`System.out.printf` es un método de Java que permite escribir con formato en lugar de escribir texto directamente.

Ejemplo

```java
System.out.printf("%s%8s\n", "index", "value");
```

Este comando en específico:

1. Sustituye `%s` con el primer valor `"index"`. `%d` es para enteros, `%s` es para textos (strings)
2. Sustituye `%8s` con el segundo valor `"value"`, con 8 espacios extra
3. `\n` es un salto de línea, ya que `printf` no hace uno automáticamente como lo hace `println`

Secuencias de formato de valores

Las secuencias de formato se usan en `printf` para colocar valores y modificar su tipo a la hora de imprimir. Algunas secuencias disponibles incluyen:

- `%d` para enteros
- `%x` o `%X` para hexadecimal, con letras en minúscula o mayúscula
- `%o` para octal
- `%nd` establece un margen de `n` espacios. Funciona con cualquier valor
- `%s` y `%S` para textos (strings) en minúscula o mayúscula
- `%.nf` pone `n` cifras decimales en números flotantes
- `%.ne` pone `n` cifras decimales en notación científica

Para fechas y horas, se utiliza el prefijo `%t` o `%T`. Este se combina con otros caracteres para formatear diferentes partes de una fecha.

- `%tY` o `%TY`: Año en 4 dígitos (ej: 2025)
- `%tm`: Mes en 2 dígitos (01-12)
- `%td` o `%Te`: Día del mes en 2 dígitos (01-31)
- `%tH`: Hora en 24h (00-23)
- `%tM`: Minuto (00-59)
- `%tS`: Segundo (00-60)
- `%tL`: Milisegundo (000-999)
- `%tF`: Fecha completa en formato `AAAA-MM-DD`
- `%tT`: Hora completa en formato `HH:MM:SS`

Ejemplo

```
import java.util.Date;

public class FechaFormato {
    public static void main(String[] args) {
        Date hoy = new Date();
        System.out.printf("La fecha es: %tF\n", hoy); // Salida: La
fecha es: 2025-09-11
        System.out.printf("La hora es: %tT\n", hoy);   // Salida: La
hora es: HH:MM:SS
```

```
    }
}
```

UML - Lenguaje unificado de modelado (mar. 16 sep. 2025)

¿Qué es UML?

Es un lenguaje de modelado diseñado para ser una estructura sencilla y uniforme para el desarrollo de sistemas en desarrollo de software y programación. Fue creado por los "tres amigos" Grady Booch, James Rumbaugh e Ivar Jacobson entre 1996 y 1999.

Está diseñado para ser abstracto, semántico y legible por humanos. Para el caso de la programación orientada a objetos, se usan principalmente los **diagramas de clases**, junto con los de objetos, componentes, paquetes e implementación.

Glosario básico

1. **UML**: Lenguaje Unificado de Modelado.
2. **Diagrama de clases**: El diagrama más común en la POO, que describe la estructura de un sistema mostrando las clases, sus atributos y operaciones, y las relaciones entre ellas.
3. **Clase**: Una plantilla para crear objetos que comparten una estructura y comportamiento comunes.
4. **Objeto**: Una instancia concreta de una clase.
5. **Paquete**: Un mecanismo para agrupar clases y otros elementos relacionados en el diagrama, ayudando a organizar la información.
6. **Niveles de visibilidad**: Símbolos que indican el nivel de acceso a los atributos y métodos de una clase. Los más comunes son:
   - `+` : **Público**: Accesible desde cualquier clase.
   - `-` : **Privado**: Solo accesible dentro de la propia clase.
   - `#` : **Protegido**: Accesible desde la propia clase y sus subclases (herederos).

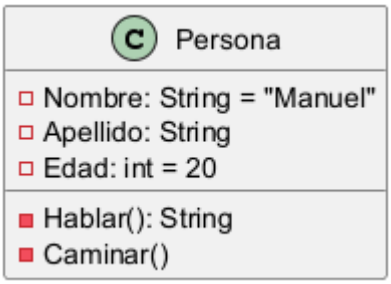
Para el proceso de modelado es evidente también el tener en cuenta los conceptos básicos de la programación orientada a objetos, como los objetos, clases, abstracción, encapsulamiento, herencia y polimorfismo.

Diagramas de clases

Las clases se representan con una forma rectangular con tres divisiones horizontales: el nombre de la clase, sus atributos y sus métodos. También se pueden especificar sus valores y tipos de datos de forma análoga a la forma del código.

Ejemplo

Una persona es un objeto que tiene nombre, apellido, edad, entre otras cosas, como sigue.

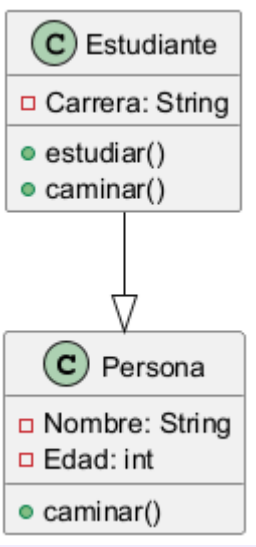


Herencia

Cuando una clase hereda de otra, se hace una flecha desde la clase hija hacia la clase padre.

Ejemplo

La flecha `Estudiante --|> Persona` muestra que la clase `Estudiante` hereda de la clase `Persona`. Esto significa que `Estudiante` obtendrá los atributos y métodos de `Persona`, como `Nombre`, `Edad` y `caminar()`, además de los suyos propios.



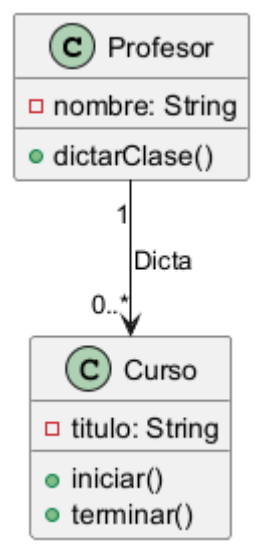
Asociaciones

Muestran una relación estructural entre clases.

- **Asociación simple:** línea continua entre clases.
- **Multiplicidad:** especifica cuántas instancias participan (`1`, `0..*`, `1..*`).
- **Navegabilidad:** una flecha indica que solo una de las clases conoce a la otra.

Ejemplo

Dado un `Profesor` este puede dictar cero o muchos `Curso`. Cada `Curso` es dictado por un solo `Profesor`.



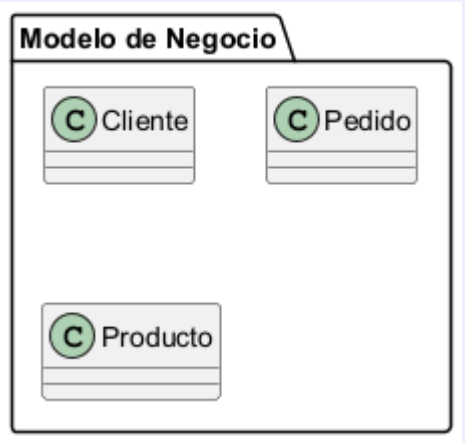
Paquetes

Sirven para **agrupar clases** relacionadas y mantener el modelo ordenado. Se representan como carpetas con una pestaña superior.

Dentro de un diagrama de clases, un paquete puede contener otros paquetes.

Ejemplo

El cliente, el pedido y el producto con parte del paquete `Modelo de negocio`.



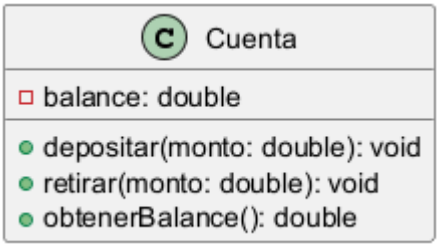
Niveles de visibilidad

Además de `+` público, `-` privado y `#` protegido, en algunos casos se añaden:

- `~` **Paquete (default)**: accesible solo dentro del mismo paquete.
- `/` **Derivado**: atributo calculado a partir de otros.

Ejemplo

La clase `Cuenta` tiene varios atributos y métodos. Solo uno es privado.



Instancias (Diagramas de objetos)

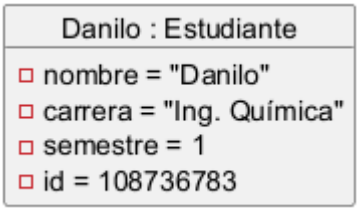
Un **diagrama de objetos** muestra ejemplos concretos de clases en un momento dado.

- Cada objeto se escribe como `nombreObjeto : NombreClase`.
- Permite visualizar valores de atributos en un instante de ejecución.

El aspecto de estos diagramas es idéntico al de los diagramas de clases

Ejemplo

`Danilo` es una instancia de la clase `Estudiante` que contiene valores establecidos y cambios respecto a la clase original.



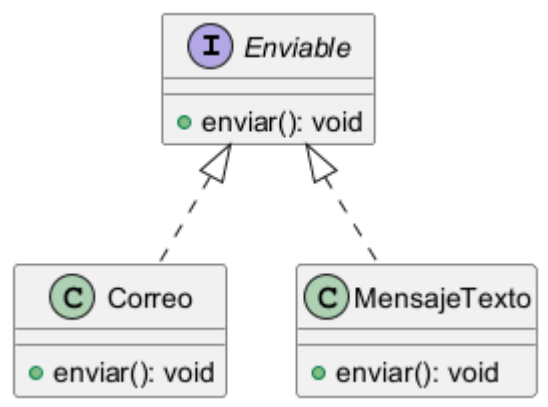
Interfaces

Una **interfaz** define operaciones que las clases que la implementan deben cumplir.

- Se dibuja como un círculo con estereotipo `<<interface>>` o un rectángulo con esa etiqueta.
- La relación de implementación se representa con una línea discontinua y triángulo vacío.

Ejemplo

`Enviable` es una interfaz que define el método `enviar()`. Las clases `Correo` y `MensajeTexto` implementan la interfaz (línea discontinua con triángulo).



Abstracción, encapsulación y utilidades (jue. 18 sep. 2025)

Casting de tipos

El casting consiste en el cambio entre tipos de datos en Java. Algunas veces este proceso es automático, aunque también puede ser manual dependiendo de la conversión.

Las conversiones entre tipos siguen el esquema de esta tabla:

Valor a asignar ↓ / Variable destino →	`int`	`long`	`float`	`double`	`char`	`byte`	`short`	`boolean`
`int`	-	A	A	A	C	C	C	N
`long`	C	-	A	A	C	C	C	N
`float`	C	C	-	A	C	C	C	N
`double`	C	C	C	-	C	C	C	N
`char`	A	A	A	A	-	C	C	N
`byte`	A	C	A	A	C	-	A	N
`short`	A	A	A	A	C	C	-	N
`boolean`	N	N	N	N	N	N	N	-

- **A:** Automático. Java se encarga de hacer la conversión al valor más pertinente.

```
int i = 10;
double d = i; // int → double automático
```

- **C:** Conversión. El usuario debe hacer la conversión de forma manual.

```
double d = 9.8;
int i = (int) d; // double → int, se trunca la parte decimal
```

- **N:** No compatible. No permite hacer la conversión porque no hay una equivalencia relevante entre valores.

```
boolean b = true;
int n = (int) b; // No es compatible
```

Números aleatorios

Para usar números aleatorios en Java usamos el paquete Random. Se importa como:

```
import java.util.Random;
```

Luego, se invoca un generador con:

```
Random r = new Random();
```

A partir de aquí, llamamos a los métodos que necesitemos. Algunos de ellos son:

1. `nextInt(int max)` : entero entre 0 y `max` (excluyendo este último)

```
// Va de 0 a 9
int n = r.nextInt(10);
```

2. `nextDouble()` : número flotante doble entre 0.0 y 1.0

```
double d = r.nextDouble();
```

Esto también lo puedes hacer con `Math.random()` sin importar nada:

```
double d = Math.random();
```

3. `nextBoolean()` : `true` o `false` aleatorio

```
boolean b = r.nextBoolean();
```

4. `nextInt()` : entero dentro de todo el rango de `int`

```
int i = r.nextInt();
```

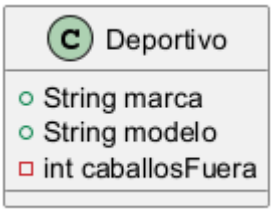
Abstracción

La abstracción es el proceso que involucra el reconocimiento y el enfoque en las características relevantes de un objeto, ignorando sus detalles irrelevantes.

Ejemplo

Si pensamos en un vehículo, pensamos en un carro, luego en un deportivo y luego en un modelo. En cada paso solo nos fijamos en las características esenciales.

Los deportivos tienen características como modelo, caballos de fuerza y tamaño. Esto nos hace diferenciarlo de, por ejemplo, un triciclo.



En la programación orientada a objetos se toman estas abstracciones y las convierten en sistemas de objetos y clases para su aprovechamiento y manejo.

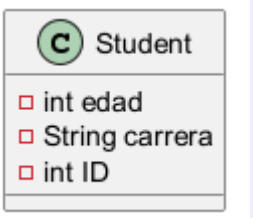
Objetos

Loa objetos reales y los conceptuales difieren un poco. Mientras los objetos reales están llenos de detalles, los conceptuales son simplificaciones manejables de forma virtual.

Los objetos tienen un estado, unos atributos y y datos almacenados en ellos.

Ejemplo

Un estudiante tiene atributos como su edad, carrera e identificación, entre otros. Cada uno de ellos estará asociado a un dato.



Clases

Las clases poseen una lista de atributos y comportamientos comunes a un conjunto de objetos que son sus instancias.

Instanciación y variables por referencia

Al instanciar un objeto con la notación `Class object = new Class()` obtenemos una referencia a una instancia de objeto, una por cada vez que se hace un `new Class()`, ya que esto consiste en reservar memoria para ese proceso.

En cambio, cuando asignamos el valor de un objeto a otro creado:

```
// Primer objeto
Class object1 = new Class();

// Segundo objeto desde el primero
Class object2 = object1;
```

El segundo objeto es una referencia a la instancia del primero.

Recolección de basura en Java

La recolección de basura en Java funciona cuando la JVM detecta que una clase no tiene instancias activas y libera memoria en consecuencia.

Encapsulación

La encapsulación es la forma en la que las clases mantienen privadas o accesibles sus atributos o métodos a fin de mantener la privacidad y hacer más robustos los programas. En UML se logra con íconos por tipo de protección:

Y en Java con las palabras clava `public`, `private` y `protected` respectivamente.

Es común asociar la encapsulación con el uso de **getters** y **setters** que son métodos personalizados para acceder y establecer el valor de una variable, respectivamente

Ejemplo

Una clase almacena un número y usa un getter y un setter para obtener su valor, ya que está protegido.

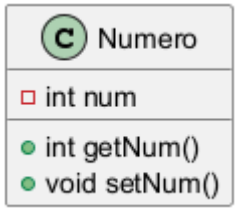
Código Java

```java
public class Numero {
    // Como el número es privado, no es accesible directamente
    private int num;

    // El getter obtiene su valor de forma controlada y pública
    public int getNum() {
        return this.num;
    }

    // El setter establece su valor desde afuera
    public void setNum(int newNum) {
        this.num = newNum;
    }
}
```

Diagrama UML



Redundancia de datos, listas y proyectos (mar. 23 sep. 2025)

Redundancia e integridad de datos

Si los datos de un programa se almacenan en diferentes instancias, se considera que es información **redundante**, la cual puede afectar el consumo y usabilidad de la misma en un programa, especialmente en términos de la **integridad de la información**.

#Pendiente ejemplo y posible conexión con los demás temas de la materia

Listas

#Pendiente consultar y desarrollar cada una

- Instanciar
- Añadir
- Obtener objetos
- Iterar listas con un bucle for

Como organizar proyectos en Java

#Pendiente consultar sobre la organización de proyectos en Java

---

Métodos y constructores en Java (jue. 25 sep. 2025)

Organización de paquetes

Un proyecto esta organizado por capas:

1. Capa de datos: donde se almacena la información que usa el proyecto
2. Capa de lógica: contiene la parte principal y funcional del proyecto
3. Interfaz de usuario: es la parte usable para el usuario final, protegiendo el interior del sistema y su integridad

Definición de métodos

Un método corresponde a una función, aunque también a un servicio. En el proceso de abstracción definimos cuales usar y como funcionan. Por supuesto, los métodos pueden recibir datos para actuar de la forma deseada. Así mismo, los métodos pueden retornar datos que pueden usarse después.

Ejemplo

Un método para crear cursos puede recibir datos como el nombre y número, y retornar otros como el curso ya creado.

```
public static Course createCourse(String name, int number) {
    Course course = new Course();

    course.setName(name);
    course.setNumber(number);

    return course;
}
```

Luego, podemos usar ese objeto creado y retornado por el método.

Los métodos reciben datos llamados **parámetros**, que son variables que van en los paréntesis `()` de la función y se usan solo dentro de ella. Para usarlos, el llamado a la función los pasa en el mismo orden con su valor deseado.

Ejemplo

Dado el método definido antes, podemos invocarlo con los parámetros que requiere. Los parámetros deben coincidir en orden y tipo, y evidentemente, tener un valor adecuado.

```
// Definición de variables que usaremos para el nuevo curso
String nombreDelCurso = "Programación Orientada a Objetos";
int numeroDelCurso = 2025;

// Llamada a la función, pasando las variables como argumentos.
// Ahora la variable 'miCurso' contiene el objeto Course que fue creado
Course miCurso = createCourse(nombreDelCurso, numeroDelCurso);

// Imprimimos los datos del objeto
System.out.println("Curso creado: " + miCurso.getName() + ", Número: " +
miCurso.getNumber());
```

Por el lado del retorno, el valor de retorno es necesario para cualquier función que no sea de tipo `void`. Este valor tiene el tipo que se indica en la declaración de la función y actúa como valor del mismo cuando se le llama. Solo se ejecuta el primer `return` que el código encuentre en su camino, el cual detiene la ejecución del método.

Ejemplo

Un método puede tener varios lugares de retorno, funcionar de manera condicional y juntarse con más instrucciones. Pero solo se ejecutará el primer `return` que el código alcance, impidiendo que funcione cualquier cosa después de él en el método.

```
public static String verificarEdad(int edad) {
    // Si se cumple la condición, retorna el String y la ejecución del
método termina
    if (edad >= 18) return "Mayor de edad";

    // Si la condición de arriba NO se cumple, la ejecución continúa y
retorna este otro String.
    return "Menor de edad";int
```

```
    // Cualquier código que se ponga después del primer 'return' que se
ejecuta es inalcanzable.
    System.out.println("Esto nunca se imprime");
}

// Ejemplo de uso
String resultado1 = verificarEdad(20); // -> "Mayor de edad"
String resultado2 = verificarEdad(15); // -> "Menor de edad"
```

Firma de un método

La **firma de un método** es la combinación del nombre del método y la lista de tipos de datos de sus parámetros, en el orden en que aparecen. Es como el "identificador único" del método dentro de una clase. El tipo de retorno no es parte de la firma.

Ejemplo

Si un método tiene por definición:

```
public static Course createCourse(String name, int number) {
    // Contenido
}
```

Su firma solo contiene el nombre del método y los tipos de sus parámetros, en orden:

```
createCourse(String, int)
```

Overloading o sobrecarga de métodos

El **overloading** (o sobrecarga) de métodos y constructores es esencial. Es la capacidad de tener varios métodos (o constructores) con el **mismo nombre** en la misma clase, pero que realizan su función de forma diferente dependiendo de los datos que reciben.

Para diferenciarlos entra la firma de los métodos. Dos métodos pueden sobrecargarse cuando tienen el mismo nombre pero diferentes firmas.

Ejemplo

Podemos sobrecargar el método `sumar` dentro de una misma clase al hacer tres firmas diferentes, aunque de hecho hacen esencialmente lo mismo.

```java
public class Calculadora {
    // Sobrecarga para sumar dos enteros
    public int sumar(int a, int b) {
        System.out.println("Suma de 2 enteros");
        int s = a + b;

        // Imprime el resultado y lo retorna
        System.out.println("El resultado es: " + s);
        return s;
    }

    // Sobrecarga para sumar tres enteros (difiere en la cantidad de parámetros)
    public int sumar(int a, int b, int c) {
        System.out.println("Suma de 3 enteros");
        int s = a + b + c;

        // Imprime el resultado y lo retorna
        System.out.println("El resultado es: " + s);
        return s;
    }

    // Sobrecarga para sumar dos números decimales (difiere en el tipo de parámetros)
    public double sumar(double a, double b) {
        System.out.println("Suma de 2 dobles (double)");
        double s = a + b;

        // Imprime el resultado y lo retorna
        System.out.println("El resultado es: " + s);
        return s;
    }
}
```

El método exacto al que se llama depende automáticamente del tipo de datos al que alude su llamado en el código.

```java
// Dos enteros: llama al primer método
int s1 = sumar(561, 762)
// -> Suma de 2 enteros
// El resultado es: 1323

// Tres enteros: llama al segundo método
int s2 = sumar(4561, 2565, 43)
// -> Suma de 3 enteros
```

```
    // El resultado es: 7169

    // Dos doubles: llama al tercer método
    double s3 = sumar(1.566, 2.87)
    // -> Suma de 2 dobles (doubles)
    // El resultado es: 4.436
```

Constructores

El constructor de una clase es la forma en la que se construyen sus instancias. Para crearlo tenemos dos métodos:

1. Por defecto: si no lo ponemos, el constructor existe de forma implícita como un *constructor por defecto*.

```
public class Student {
    // El constructor no esta declarado
}

// Llamada al constructor por defecto
Student s = new Student()
```

El constructor por defecto instancia el objeto completo, pero lo deja todo vacío.

2. Constructor explícito: invocamos el constructor con la sintaxis:

```
public class Persona {
    // Atributos
    private String name;
    private int age;

    // Constructor. No posee tipo y tiene el mismo nombre que la clase
    // El constructor toma sus parámetros e inicializa valores con ellos
    public Persona(String name, int age) {
        // El uso de this es una forma de distinguir los atributos de otras
variables con el mismo nombre
        this.name = name;
        this.age = age;
    }
}

// Instancia con el constructor y sus parámetros
Persona e = new Persona("Andrés", 16);
```

Sobrecarga de constructores

Los constructores pueden recibir overloading como cualquier método normal. Esto nos ayudará a que instanciar objetos sea más dinámico al adaptarse a los tipos de datos disponibles. El constructor que se termina llamando depende de los parámetros que se le pasen, como en la sobrecarga normal.

Ejemplo

```
public class Persona {
    // Atributos
    private String name;
    private int age;

    // 1. Constructor con dos parámetros
    public Persona(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // 2. Overloading: Constructor con un parámetro
    public Persona(String name) {
        this.name = name;
        // La edad se inicializa con un valor por defecto
        this.age = 0;
    }

    // 3. Overloading: El constructor por defecto, que solo inicializa los
atributos con sus valores predeterminados
    public Persona() {
        // No hace falta poner código, o puedes poner valores por defecto
si lo deseas.
    }
}

// Instancias usando los diferentes constructores.
// Usa el constructor 1
Persona e1 = new Persona("Andrés", 16);
// Usa el constructor 2
Persona e2 = new Persona("Carla");
// Usa el constructor 3
Persona e3 = new Persona();
```

Forzar el constructor por defecto

El constructor por defecto (que instancia a un objeto vacío) desaparece cuando se escribe de forma explicita. En el ejemplo anterior se muestra como recuperarlo para su uso:

```
public Persona() {
    // No hace falta poner código, o puedes poner valores por defecto si lo deseas.
}
```

También puedes poner valores por defecto por tu propia cuenta en el constructor vacío, pero es opcional.

Atributos estáticos y relaciones entre objetos (mar. 30 sep. 2025)

Atributos estáticos

#Pendiente explicación

#Pendiente ejemplo

Relaciones entre objetos

#Pendiente explicaciones más adecuadas y notación UML y Java asociadas a cada una

Asociaciones

Son simples relaciones entre dos clases, como acceso a métodos, acciones o similares

#Pendiente ordenes de asociaciones, asociaciones de orden superior

Agregación

Composición