CICLO 2

[FORMACIÓN POR CICLOS]

Programación Básica JAVA

Semana 2

Ingeni@ Soluciones TIC

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Programa

Semana 2	Introducción a la programación orientada a objetos	2	5	7
	Clases, objetos, métodos y atributos	2	5	7
	Diagramas de clases en UML 2.5	3	7	10

Contenido

- Introducción a programación Orientada a Objetos
- Diagramas UML
- Quizziz

- Clases
- Objetos
- Atributos
- Métodos
- Quizizz

- Ejemplo de Implementación: UML + POO Java

Ejemplo Tienda

Diagrama de clases en UML 2.5

Para crear programas eficientes se debe seguir un proceso de **análisis detallado** para determinar los requerimientos de un proyecto (*definir qué se supone que debe hacer el sistema*) y desarrollar un diseño que los satisfaga (*decidir cómo debe hacerlo el sistema*). Lo ideal sería pasar por este proceso y revisar el diseño con cuidado (*además de pedir a otros profesionales de software que revisen su diseño)* **antes de escribir cualquier código.**

Si este proceso implica analizar y diseñar su sistema desde un punto de vista orientado a objetos, se denomina proceso de **análisis y diseño orientado a objetos (A/DOO).** Los lenguajes como Java son orientados a objetos. La programación en un lenguaje de este tipo, conocida como programación orientada a objetos (POO), le permite implementar un diseño orientado a objetos como un sistema funcional.

Diagrama de clases en UML 2.5

Aunque existen muchos procesos de A/DOO distintos, hay **un solo lenguaje gráfico** para comunicar los resultados de cualquier proceso de A/DOO que se utiliza en la mayoría de los casos. Este lenguaje, conocido **como Lenguaje unificado de modelado (UML),** es en la actualidad el esquema gráfico más utilizado para modelar sistemas orientados a objetos.

En la industria, los diagramas de UML ayudan a los diseñadores de sistemas a especificar un sistema de una forma concisa, gráfica e independiente del lenguaje de programación, antes de que los programadores implementen el sistema en un lenguaje de programación específico.

Diagrama de clases en UML 2.5

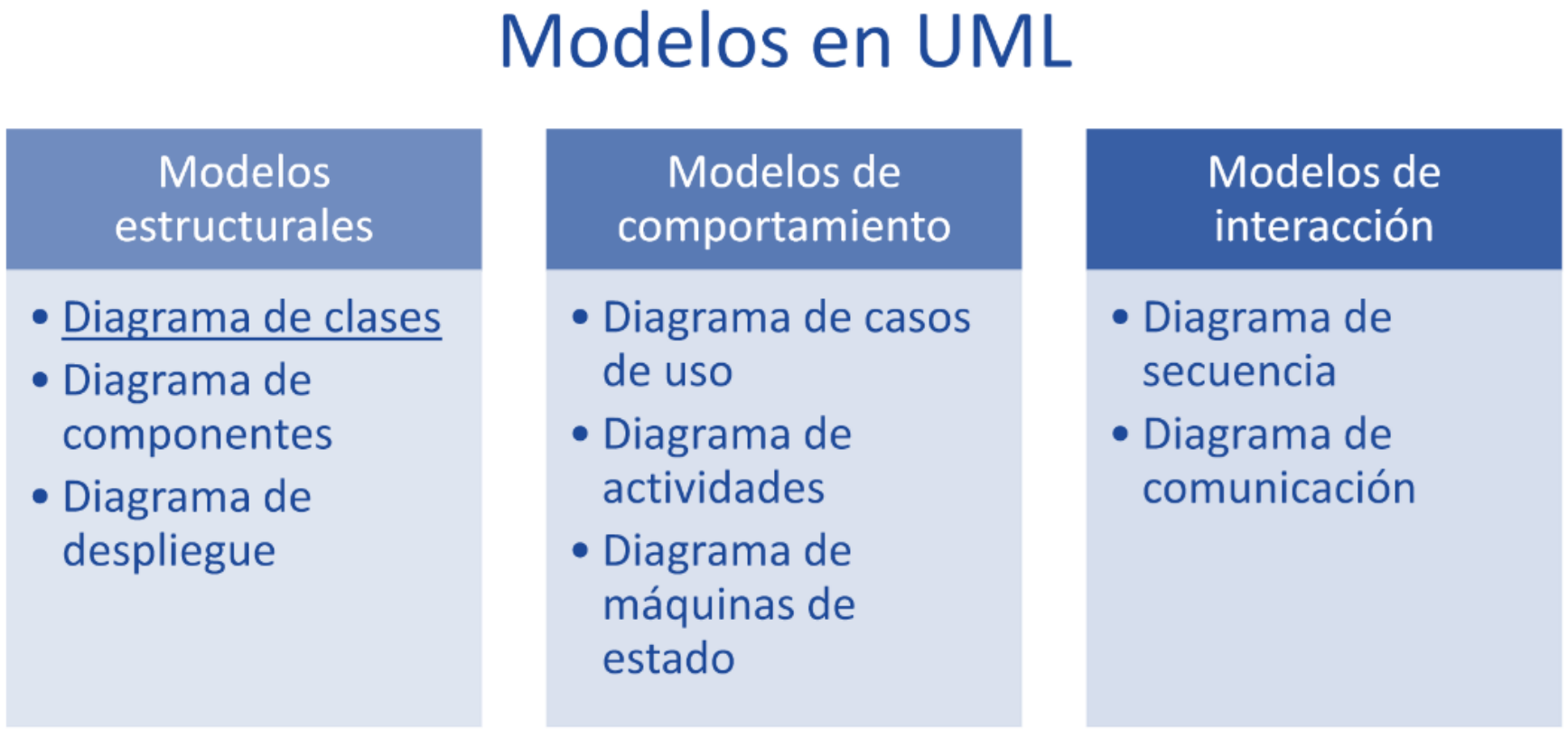
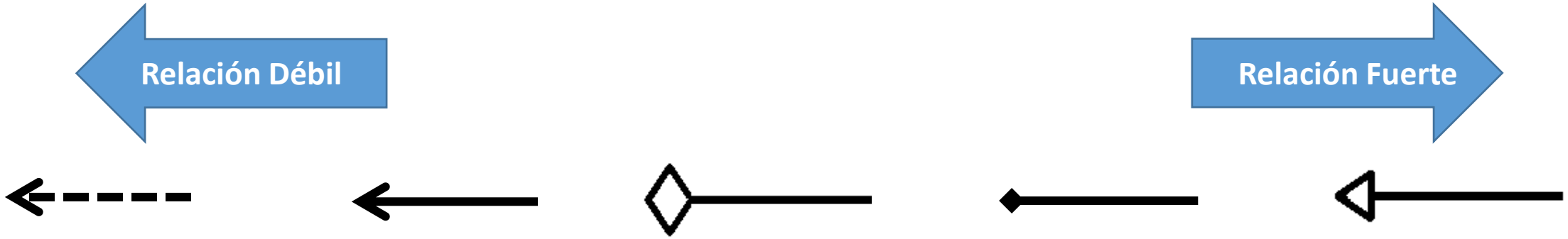


Diagrama de clases en UML 2.5



Dependencia

Cuando objetos de una clase trabajan poco con objetos de otra clase

Empleado Herramienta

Asociación

Representar una relación débil entre dos clases, sin un significado especial

Tienda y producto

Agregación

Relación más fuerte, pero en la cual el ciclo de vida de un objeto de la clase donde termina la relación no depende del ciclo de vida de un objeto de la clase donde inicia

Producto y Bimbo

Composición

La relación es aún más fuerte, y el ciclo de vida de un objeto de la clase donde termina la relación siempre dependerá del ciclo de vida de un objeto de la clase donde inicia esta

Tienda y RUT

Herencia

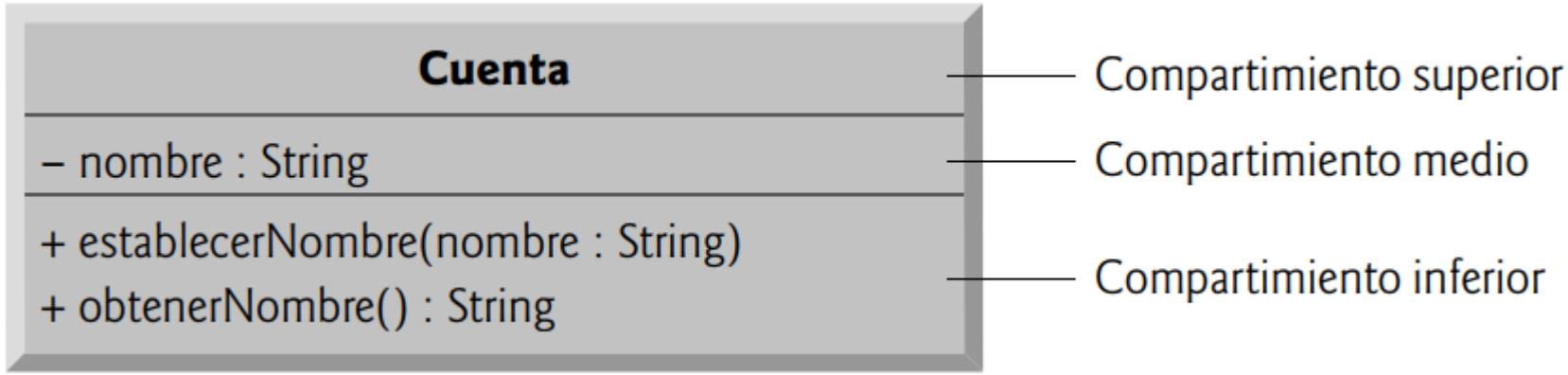
Cuando una clase es del tipo de otra

Tienda y panadería

Diagrama de clases en UML 2.5

```
5  public class Cuenta
6  {
7      private String nombre; // variable de instancia
8
9      // método para establecer el nombre en el objeto
10     public void establecerNombre(String nombre)
11     {
12        this.nombre = nombre; // almacenar el nombre
13     }
14
15     // método para obtener el nombre del objeto
16     public String obtenerNombre()
17     {
18        return nombre; // devuelve el valor de nombre a quien lo invocó
19     }
20 } // fin de la clase Cuenta
```

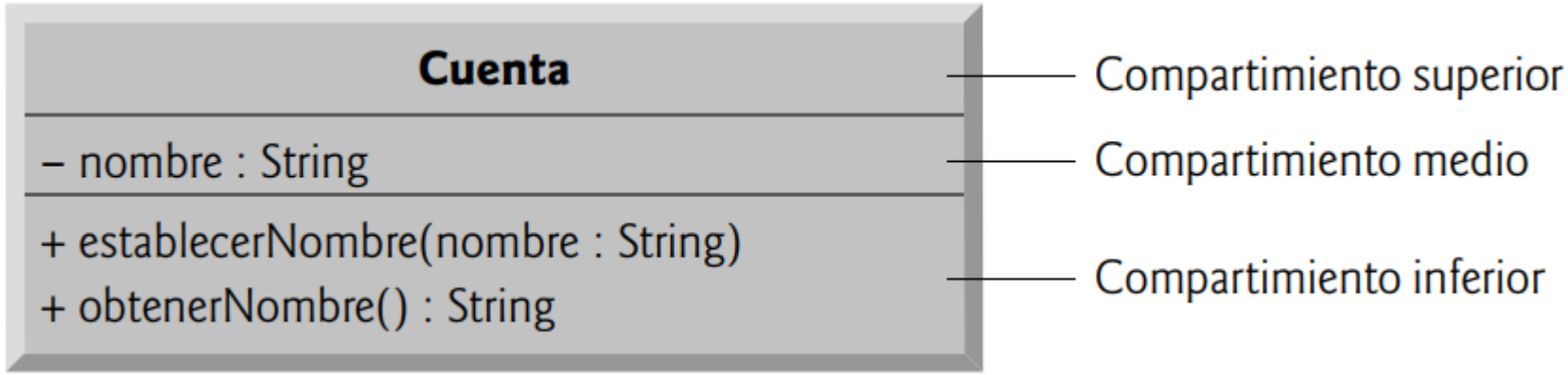
Diagrama de clases en UML 2.5



Compartimiento superior

En UML, cada clase se modela en un diagrama de clases en forma de un rectángulo con tres compartimientos. En este diagrama el compartimiento superior contiene el nombre de la clase Cuenta, centrado horizontalmente y en negrita

Diagrama de clases en UML 2.5



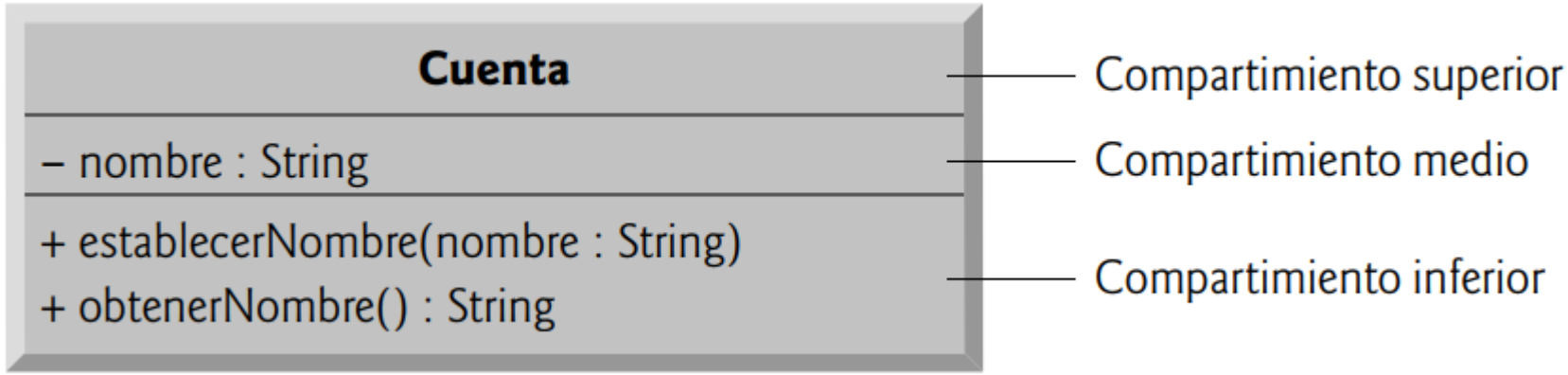
Compartimiento medio

El compartimiento medio contiene el atributo de la clase nombre, que corresponde a la variable de instancia del mismo nombre en Java.

La variable de instancia nombre es private en Java, por lo que el diagrama de clases en UML muestra un modificador de acceso de ***signo menos (-)*** antes del nombre del atributo.

Después del nombre del atributo hay un signo de dos puntos y el tipo del atributo; en este caso String.

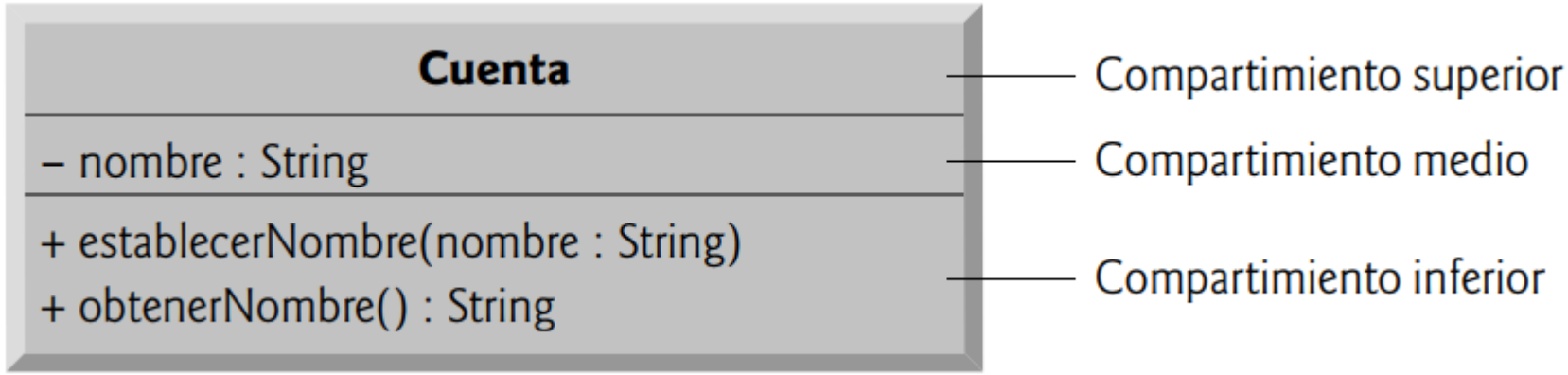
Diagrama de clases en UML 2.5



Compartimiento inferior

El compartimiento inferior contiene las operaciones de la clase, ***establecerNombre*** y ***obtenerNombre***, que en Java corresponden a los **métodos**. Para modelar las operaciones, UML lista el nombre de la operación precedido por un modificador de acceso; en este caso, + ***obtenerNombre***. Este signo más **(+)** indica que ***obtenerNombre*** es una operación **pública (public)** en UML (porque es un método public en Java). La operación ***obtenerNombre*** no tiene parámetros, por lo que los paréntesis después del nombre de la operación en el diagrama de clases están vacíos. La operación ***establecerNombre***, que también es pública, tiene un parámetro String llamado nombre

Diagrama de clases en UML 2.5

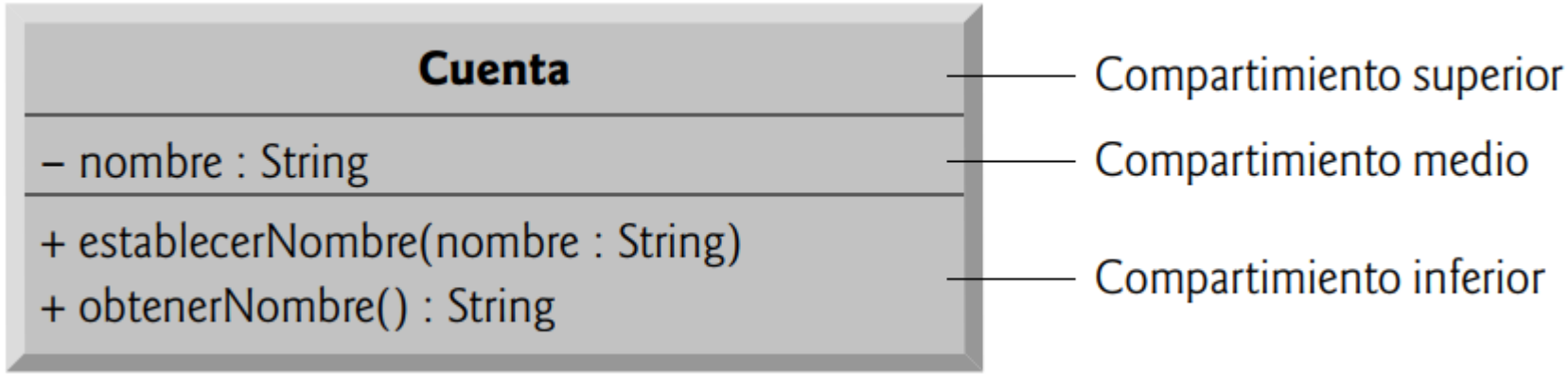


Tipos de valores de retorno

UML indica el tipo de valor de retorno de una operación colocando dos puntos y el tipo de valor de retorno después de los paréntesis que le siguen al nombre de la operación. El método ***obtenerNombre*** de la clase Cuenta tiene un tipo de valor de retorno String.

El método ***establecerNombre*** no devuelve un valor (void), por lo que el diagrama de clases de UML no especifica un tipo de valor de retorno después de los paréntesis de esta operación.

Diagrama de clases en UML 2.5



Parámetros

La forma en que UML modela un parámetro es un poco distinta a la de Java, ya que lista el nombre del parámetro, seguido de dos puntos y del tipo del parámetro entre paréntesis después del nombre de la operación. UML tiene sus propios tipos de datos, que son similares a los de Java, pero por cuestión de simpleza usaremos los tipos de datos de Java. El método ***establecerNombre*** de Cuenta tiene un parámetro String llamado nombre, por lo que en la figura se lista a ***nombre : String*** entre los paréntesis que van después del nombre del método.

Diagrama de clases en UML 2.5

Existen distintos tipos de relaciones:

-Asociación (conexión entre clases)

-Dependencia (relación de uso)

-Generalización/especialización (relaciones de herencia).

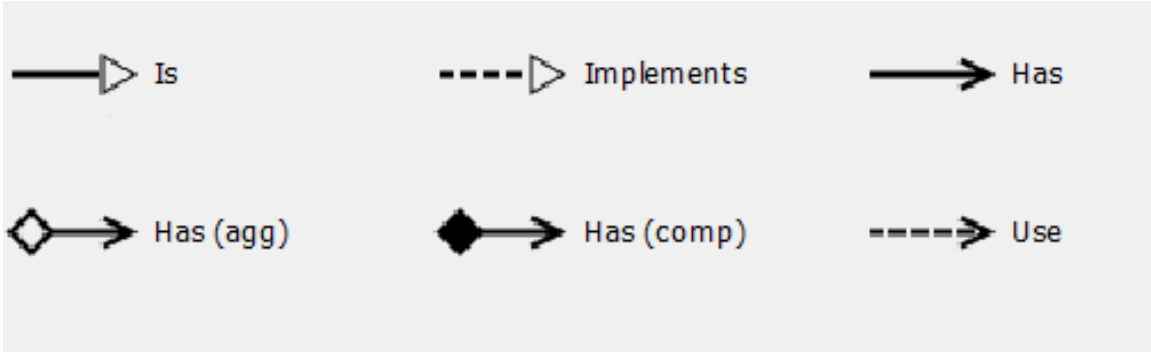
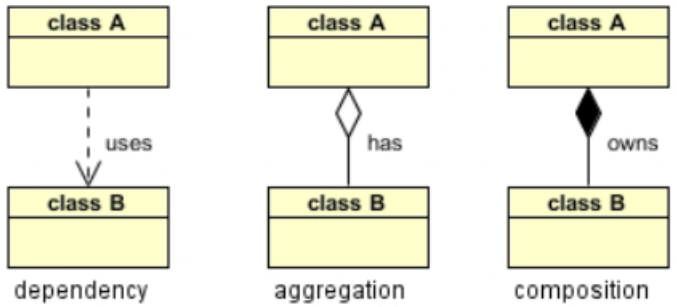
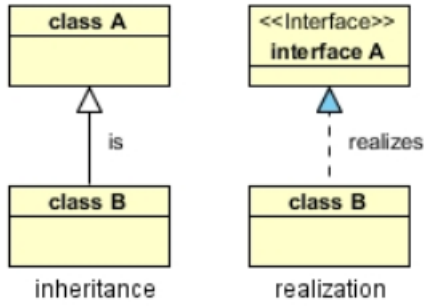


Diagrama de clases en UML 2.5





Relaciones UML Class

Relaciones:

Asociación (Dependencia): Clase A utiliza la clase B. Relación débil.

Agregación: Clase A tiene una clase B. Relación fuerte.

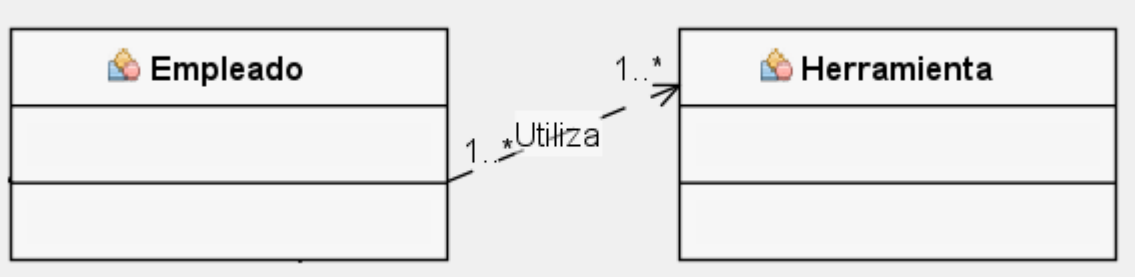
Composición: La clase A es propietaria de una clase B.
Relación fuerte y dependiente

Herencia: La clase B es una clase A (o la clase A se extiende por clase B)

Realización: Clase B se da cuenta de la Clase A (o la clase A se realiza por clase B)

Diagrama de clases en UML 2.5

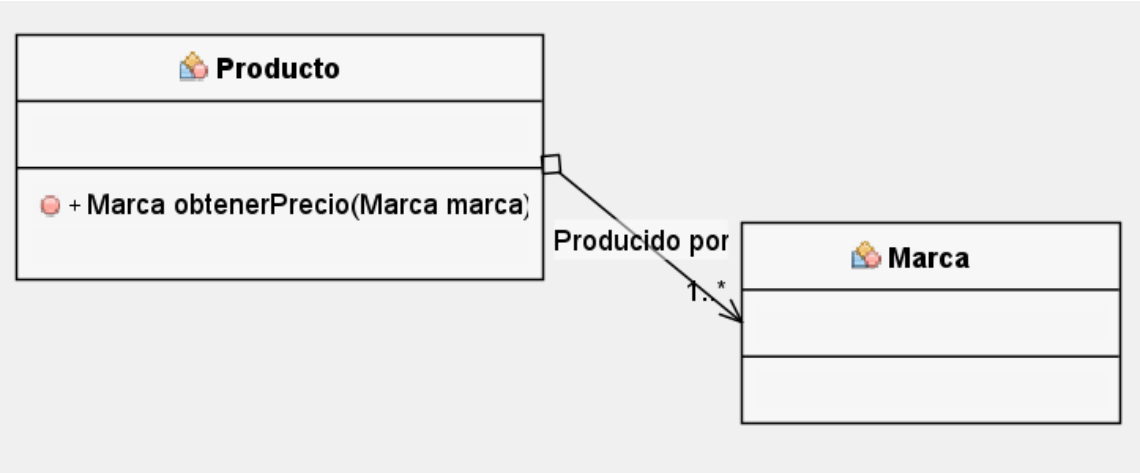
La dependencia se representa cuando se pasa una referencia a una clase como un parámetro a otro método de clase. Por ejemplo, se pasa una instancia de la clase B a un método de la clase A:




```
1  público  de clase  A {
2
3      público  vacío  doSomething (B b) {
```

Diagrama de clases en UML 2.5

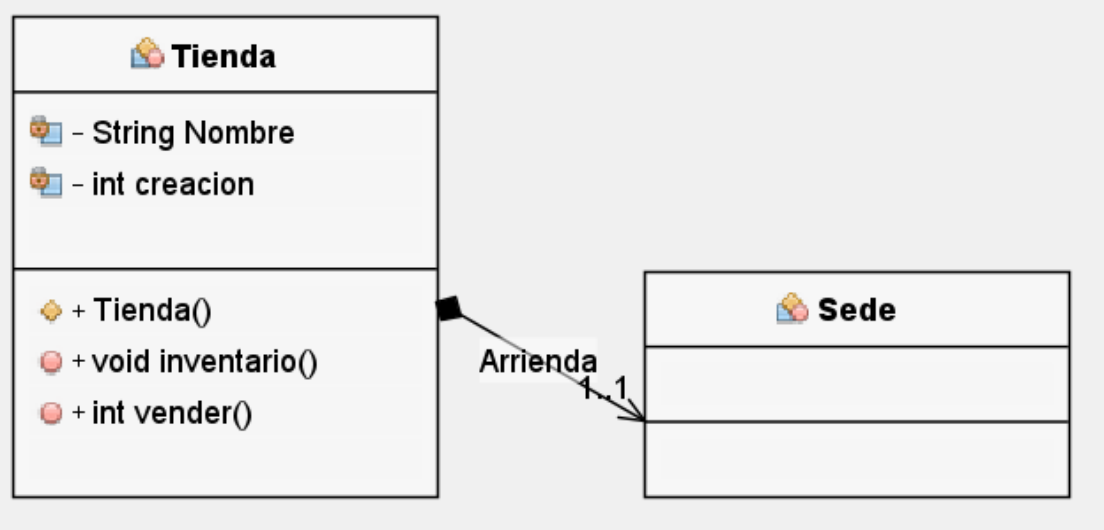
Ahora bien, si la clase A almacena la referencia a la clase B para su uso posterior, tendríamos una relación diferente llamado **agregación** . Un ejemplo más común y más obvio de agregación sería a través de la inyección de setter:




```
1  público  de clase  A {
2
3      privado  _b B;
4
4      público  vacío  SETB (B b) {_b = b; }
5
```

Diagrama de clases en UML 2.5

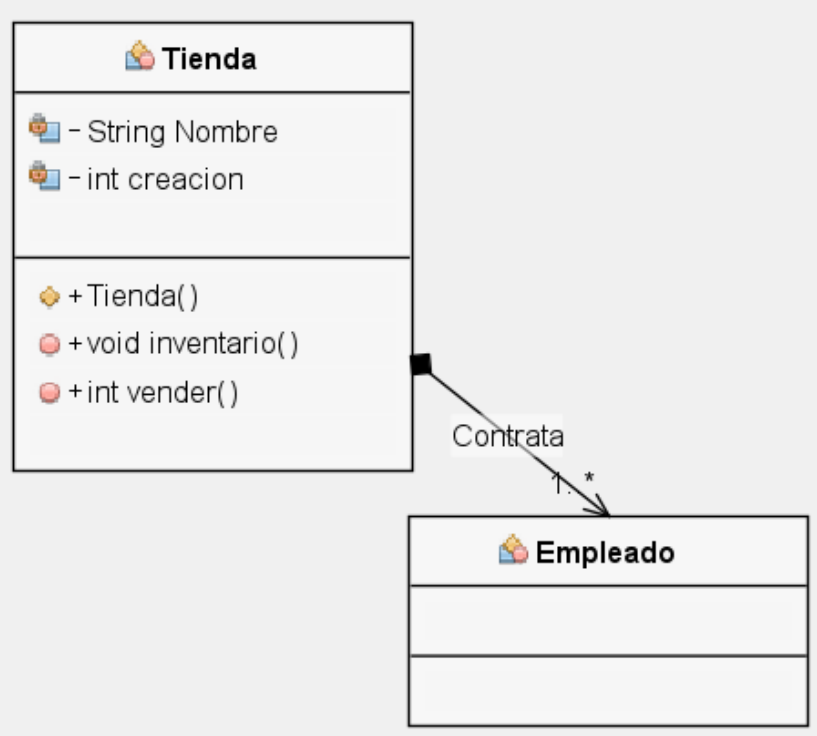
La agregación es la forma más débil de contención objeto (un objeto contiene otros objetos). La forma más fuerte se llama **Composición** . En Composición del objeto que contiene es responsable de la creación y ciclo de vida del objeto contenido (ya sea directa o indirectamente). Los siguientes son algunos ejemplos de Composición. En primer lugar, a través de la inicialización miembro:




```
1  público  de clase  A {
2
       privado  _b B = nuevo  B ();
3
```

Diagrama de clases en UML 2.5

Composición, a través de inicialización de constructor:




```
1  público  de clase  A {
2
       privado  _b B;
3
       público  A () {
4
           _b = nuevo  B ();
5
       } // constructor por defecto
6
7
```

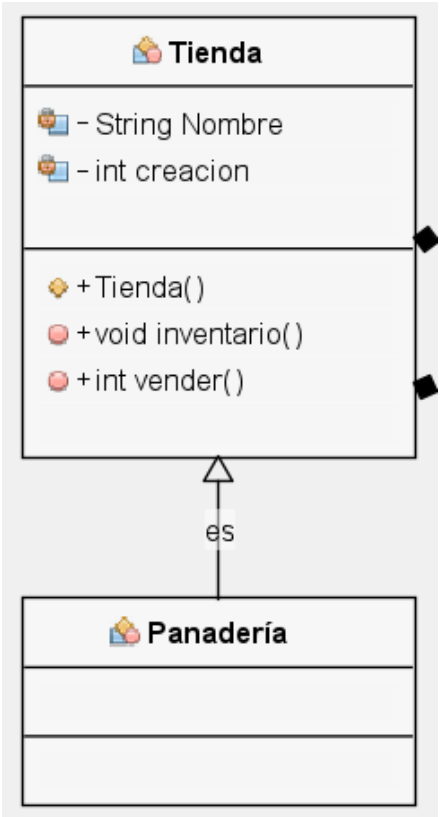
Diagrama de clases en UML 2.5

Composición, a través de init perezoso (Para ocultar por completo referencia a B):

```
1  público  de clase  A {
2
3      privado  _b B;
4      público  vacío  doSomethingUniqueToB () {
5          si  ( nulo  == _b) {
6              _b = nuevo  B ();
7          }
8          volver  _b.doSomething ();
9      } // doSomethingUniqueToB ()
10
```

Diagrama de clases en UML 2.5

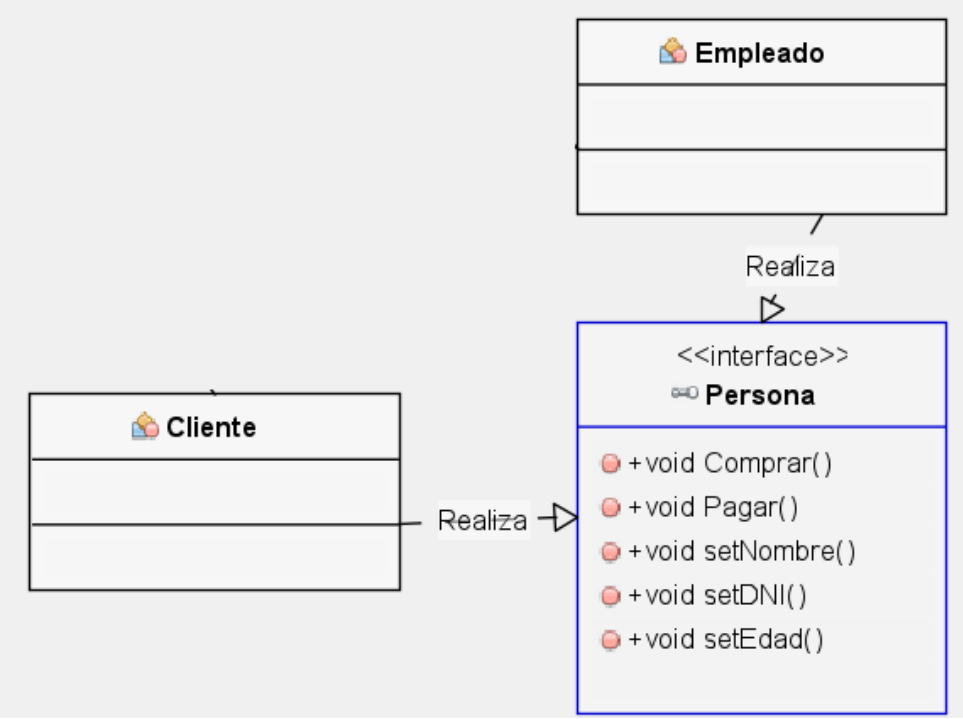
La herencia es una relación bastante directa para representar en Java:




```
1  público  de clase  A {
2
       ...
3
4  } // clase A
5
   público  de clase  B se extiende  A {
6
7      ....
8
   } // clase B
9
```

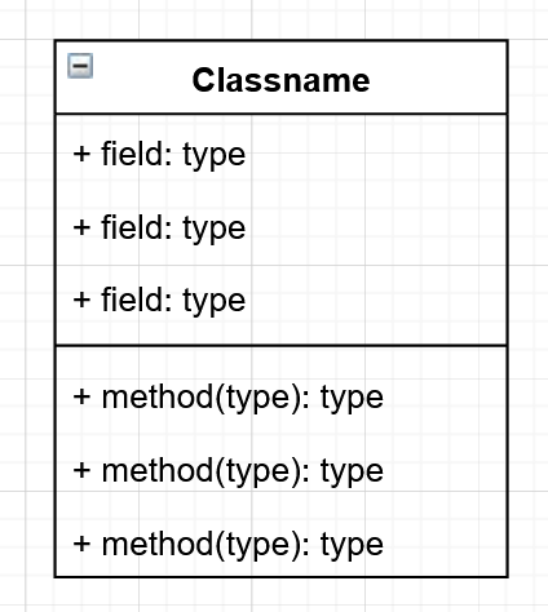
Diagrama de clases en UML 2.5

La realización también es directa en Java y se ocupa de la implementación de una interfaz:




```
1  pública  interfaz  A {
2
3      ...
4  } // interfaz A
5
6  público  de clase  B implementa  A {
7
8      ...
9  } // clase B
```

Software para hacer diagramas



https://draw.io

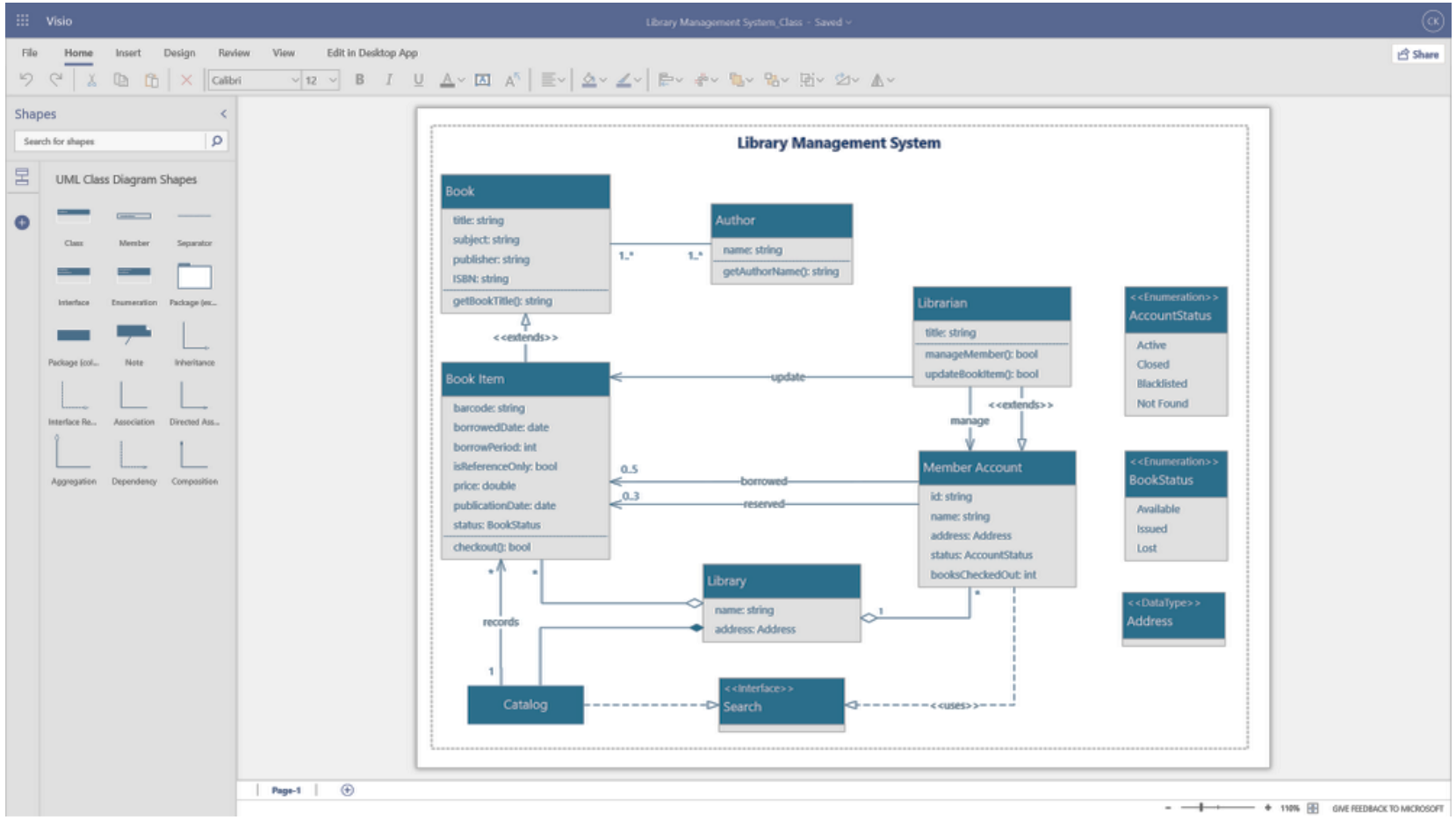
CTRL+D: Añadir otro atributo o método

SHIFT + (Flecha): Mover el campo

Click derecho: Back - Front

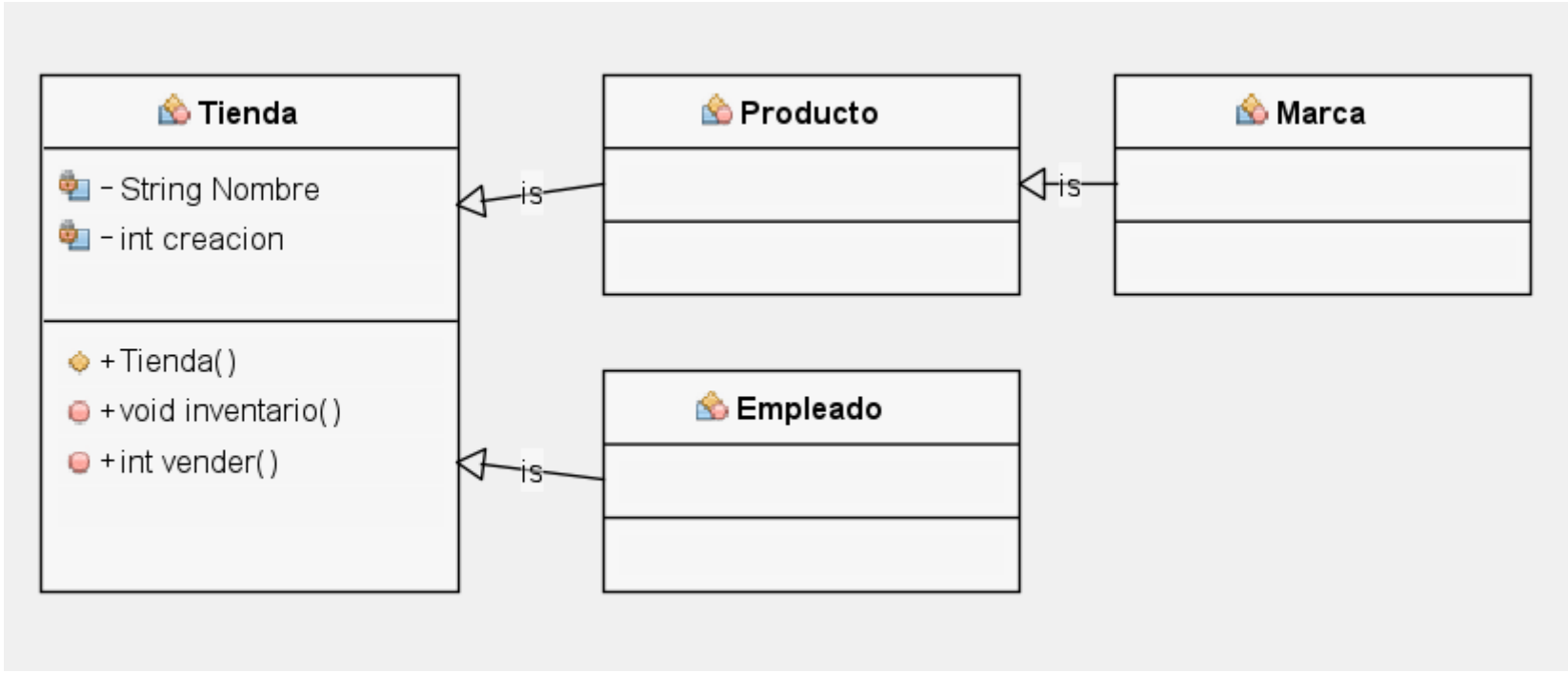
Software para hacer diagramas

Microsoft Visio



Software para hacer diagramas

Easy UML - Netbeans



Tutorial de como instalarlo: https://www.youtube.com/watch?v=mHDPLSs2d_w

https://www.mediafire.com/file/k6pzviekt0izhmw/easyUML.zip/file

Quizizz

https://quizizz.com/join?gc=20020262

Clases en Java

- Clases
- Atributos
- Métodos

Atributos

Los atributos, **también llamados datos o variables** miembro son porciones de información que un objeto posee o conoce de sí mismo. Una clase puede tener cualquier número de atributos o no tener ninguno. Se declaran con un identificador y el tipo de dato correspondiente. Además los atributos y tienen asociado un modificador que define su visibilidad según se muestra en la siguiente tabla.

Modificador	Visibilidad
`public`	Pública (+)
`protectec`	Protegida / en la herencia(#)
`private`	Privada(-)
`package`	De paquete (~)

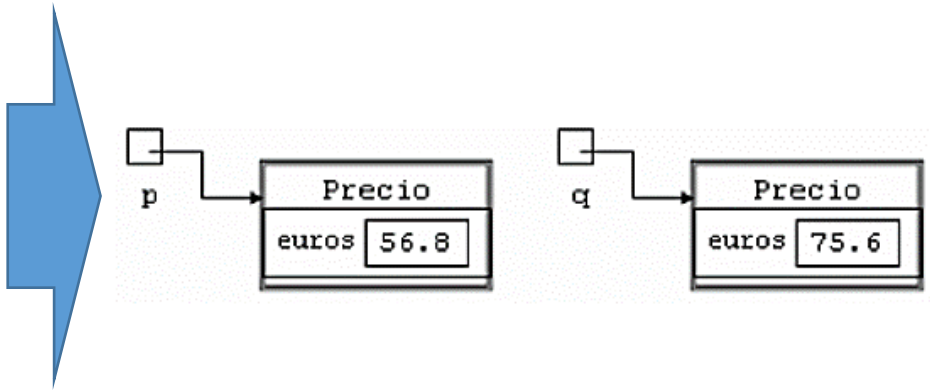
Tipos de variables en Java

Variables de instancia (campos no estáticos), son las variables que están definidas dentro de un objeto pero que no tienen un modificador de estáticas (static). Suelen llevar un modificador de visibilidad (public, private, protected) definiéndose.

```
public class Precio {
    // Declaracion de atributos o variables miembro
    public double euros;

    // Declaracion de metodos . . .
}
```

```
// Creacion de dos instancias de la clase precio
Precio p = new Precio();
p.pone(56.8);
Precio q = new Precio();
q.pone(75.6);
```



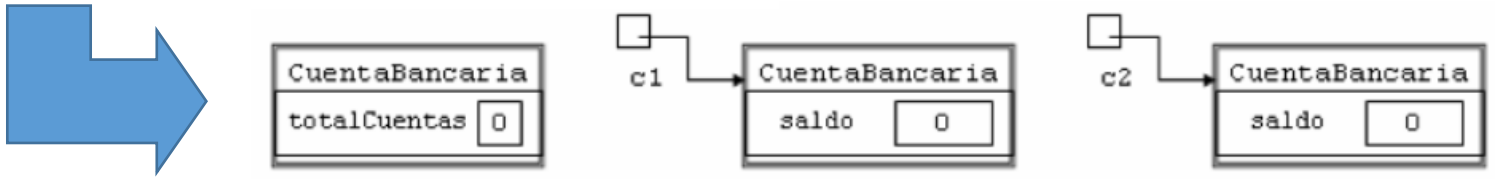
Tipos de variables en Java

Variables de clase (campos estáticos), son aquellas variables que están precedidas del modificador static. Esto indica que solo hay una instancia de dicha variable. Es decir, aunque tengamos N objetos de la clase, la variable estática solo se instancia una vez.

```
public class CuentaBancaria {
    // Atributos o variables miembro
    public double saldo;                // Variable de instancia
    public static int totalCuentas=0;   // Variable de clase

    // Declaraciones de metodos...
}
```

```
// Creacion de dos instancias de la clase CuentaBancaria
CuentaBancaria c1 = new CuentaBancaria();
CuentaBancaria c2 = new CuentaBancaria();
```



Tipos de variables en Java

Variables Finales o Constantes, una clase puede contener atributos de valor constante o variables finales. Este tipo de atributo se indica con la palabra reservada final. Las variables finales se suelen declarar además como variables de clase (static final) por razones de ahorro de memoria ya que, al no modificar su valor sólo suele ser necesaria una copia en memoria por clase (y no una por instancia

```
public class Circulo {
    // Atributos o variables miembro
private static final double PI = 3.141592;     // Constante de clase
    private double radio;

    // Declaracion de metodos ...
}
```

Tipos de variables en Java

Variables locales, son variables temporales cuyo ámbito de visibilidad es el método sobre el que están definidas. No pueden ser accedidas desde otra parte del código. Se las distingue de las variables de instancia ya que estas no llevan modificadores de visibilidad delante.

```
int variable = 2;
```

Tipos de variables en Java

Parámetros, son las variables recibidas como parámetros de los métodos. Su visibilidad será el código que contenga dicho método.

```
public Triangulo(long base, long altura){...}
```

Métodos de clase y métodos de instancia (Objeto)

- Un método es una abstracción de una operación que puede hacer o realizarse con un objeto.

- Una clase puede declarar cualquier número de métodos que lleven a cabo operaciones de lo más variado con los objetos.

- Los métodos se clasifican en dos grupos:
  - Los métodos de instancia y los métodos de clase.

Métodos de Instancia (Objeto)

En principio, los métodos de clase **no operan** sobre las variables de los objetos.

Los métodos de clase pueden trabajar con las variables de clase pero no pueden acceder a las variables de instancia declaradas dentro de la clase, a no ser que se crea una nueva instancia y se acceda a las variables de instancia a través del nuevo objeto.

Los métodos de clase también pueden ser llamados precediéndolos con el identificador de la clase, sin necesidad de utilizar el de una instancia.

Métodos de Instancia (Objeto)

```
public double saldo() {
    return this.saldo;
}

public void transferencia( CuentaBancaria origen ) {
   this.saldo += origen.saldo;
   origen.saldo = 0;
}
```

```
CuentaBancaria c1 = new CuentaBancaria();
CuentaBancaria c2 = new CuentaBancaria(20.0);

c1.transferencia(c2);
System.out.println("Cuenta con: " + c1.saldo() + " euros");
```

Accedo a través de una instancia (Objeto)

Métodos de clase

En principio, los métodos de clase no operan sobre las variables de instancia de los objetos.

Los métodos de clase pueden trabajar con las variables de clase pero no pueden acceder a las variables de instancia declaradas dentro de la clase, a no ser que se crea una nueva instancia y se acceda a las variables de instancia a través del nuevo objeto.

Los métodos de clase también pueden ser llamados precediéndolos con el identificador de la clase, sin necesidad de utilizar el de una instancia.

Métodos de clase

```
public static void incCuentas () {
    totalCuentas++;
}
```

```
CuentaBancaria.incCuentas();
```

Accedo
a través de la clase

Métodos de clase y métodos de instancia (Objeto)

Métodos...	Modificador en la declaración	Sintaxis de llamada	Operan normalmente...
... de instancia	-- (por defecto)	`instancia.metodo(parametros)`	Sobre variables de instancia
... de clase	`static`	`Clase.metodo(parametros)`	Sobre variables de clase o sobre otros datos

Los métodos de clase o estáticos se pueden considerar equivalentes a las rutinas (globales) de otros lenguajes de programación como Pascal o C.

Como ejemplos típicos de métodos estáticos pueden indicarse los métodos de Java correspondientes a las funciones matemáticas sin, cos, exp, pow... de la clase java.lang.Math.

Las llamadas a estos métodos se realizan anteponiendo el identificador de la clase Math al identificador del método: Math.sin(angulo)

Quizizz

https://quizizz.com/join?gc=55147558