

# UNIDAD 4

# DIAGRAMAS DE CLASES

---

ENTORNOS DE DESARROLLO

# DIAGRAMAS DE CLASE

---

- Forman parte de la visión estática del sistema.
- Se definen:
  - Características de las clases que forman parte del sistema.
  - Interfaces.
  - Colaboraciones.
  - Relaciones de dependencia.
  - Generalizaciones.
- Es decir, se definen las clases y se implementan las relaciones.

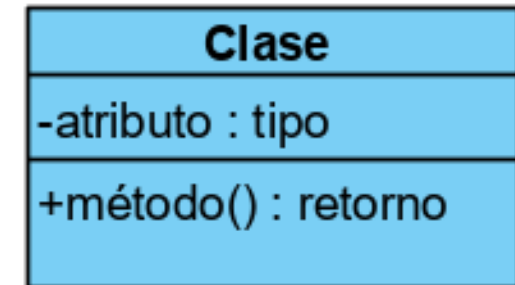
# DIAGRAMAS DE CLASE: ELEMENTO “CLASE”

---

- Se representa en UML de la siguiente manera:

- **Caja 1: NombreDeLaClase.**

- El nombre tiene que ser único.
- No se recomienda utilizar acentos.



- **Caja 2: Atributos.**

- [+|-] nombre: tipo.
- Visibilidad: “+” si es visible, “-” si está oculto, normalmente los atributos están todos ocultos.

# DIAGRAMAS DE CLASE: ELEMENTO “CLASE”

---

- Se representa en UML de la siguiente manera:

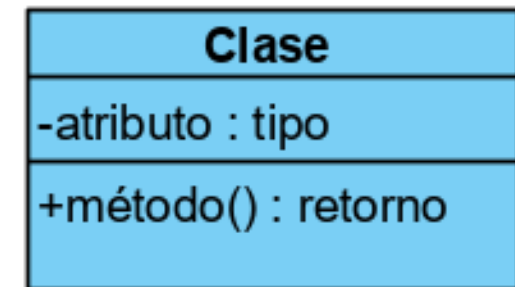
- **Caja 2: Atributos.**

- *Nombre*: nombreDelAtributo.

- Evitar el uso de acentos.

- *Tipo*: tipo de dato del atributo.

- Puede ser un tipo de dato nativo de cualquier lenguaje de programación u otra clase.



# DIAGRAMAS DE CLASE: ELEMENTO “CLASE”

---

- **Caja 3: Métodos** (Interfaz de la clase).
  - `[+ | -]nombre([parametro:tipo],[parametro:tipo],...):tipoRetorno`
  - *Visibilidad*: “+” si es visible, “-” si está oculto, normalmente los métodos son visibles.
  - *Nombre*: nombreDelMetodo.
    - Evitar el uso de acentos.

Clase
-atributo : tipo
+método() : retorno

# DIAGRAMAS DE CLASE: ELEMENTO “CLASE”

## ■ Caja 3: **Métodos** (*Interfaz de la clase*).

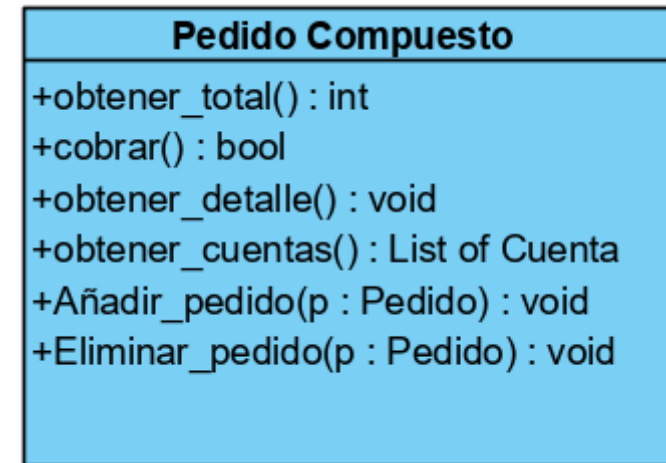
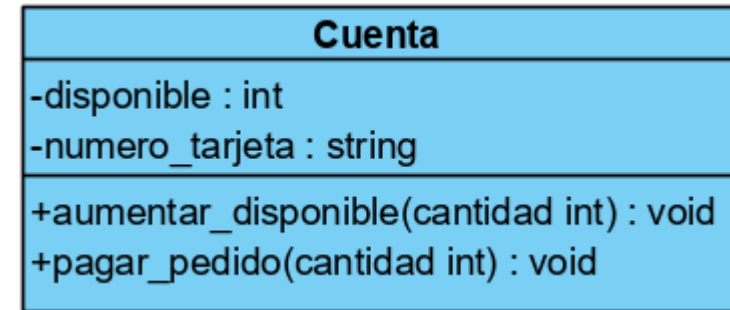
### ➤ *Lista de parámetros* (lo que hay entre ( )):

- Lista de tamaño limitado y que puede estar vacía.
- Cada parámetro es una pareja “nombre: tipo”.
- Se separa un parámetro de otro con una “,”.
- Los paréntesis tienen que aparecer siempre aunque el método no tenga parámetros.

### ➤ *Tipo retorno*:

- Después de los “:” se indica el tipo de dato que devolverá el método.

### ➤ Los tipos de datos pueden ser un tipo de dato nativo de cualquier lenguaje de programación u otra clase.

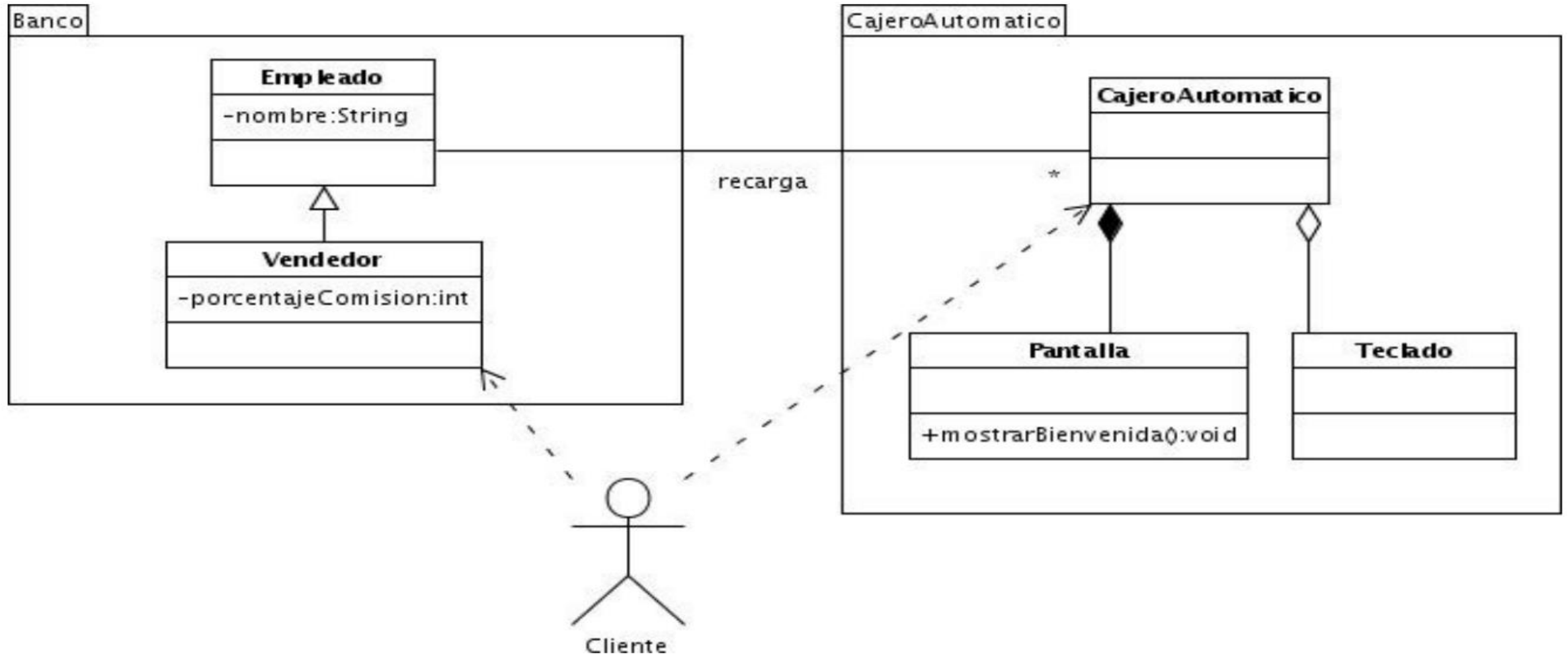


# DIAGRAMAS DE CLASE: RELACIONES

---

- La programación OO permite la colaboración entre las distintas clases de un sistema.
- El diagrama de clases permite mostrar cuales son los medios por los cuales los distintos objetos se podrían comunicar.
- En un diagrama podemos encontrar:
  1. Asociación.
  2. Agregación.
  3. Composición.
  4. Generalización (Herencia).
  5. Paquete.
  6. Actor.

# DIAGRAMAS DE CLASE: RELACIONES

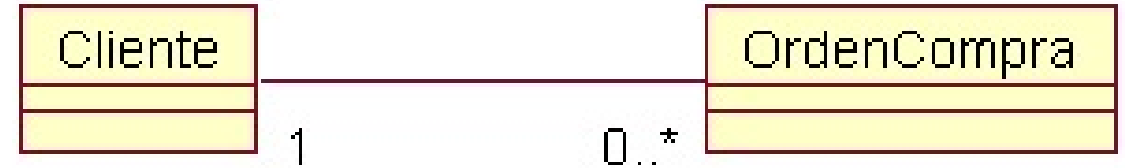




# DIAGRAMAS DE CLASE: RELACIONES

---

## 1. Asociación:



- Es una relación simple entre dos clases u objetos.
- Representa un camino de comunicación entre las dos clases.
- Se dibuja con una línea recta con flechas opcionales que definen la “navegabilidad”.
  - **Si no hay flechas** los objetos de ambas clases pueden enviarse mensajes mutuamente.
  - **Si hay alguna flecha**, los mensajes sólo pueden ir en la dirección de la flecha.

# DIAGRAMAS DE CLASE

## RELACIONES

### 1. Asociación:

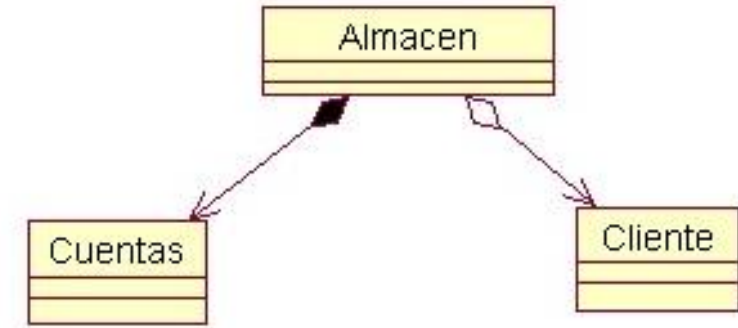
- Tiene “*multiplicidad*”:
  - Indica el número de objetos de una clase (“ocurrencias”) que puede haber en cada lado.
  - Si no hay número  $\rightarrow 1$ .
  - $*$   $\rightarrow$  0 o más.
  - $1..*$   $\rightarrow$  1 o más.
- Suelen disponer de **nombre en sus extremos** para clarificar su significado, aunque puede haber un único nombre en el medio si está suficientemente claro.

# DIAGRAMAS DE CLASE: RELACIONES

## 2. Agregación:



- Es un tipo de asociación “*especial*”.
- Se representa como una línea recta con un rombo vacío en el extremo “*agregado*”.
- Tiene los mismos calificadores que una asociación normal (1, \*, 1..\*) pero en el extremo agregado no es habitual tener más de 1.
- La agregación significa que “*el agregado está compuesto por un conjunto de...*”, siendo los elementos que lo componen las clases que están unidas al agregado.
  - ***Un ordenador está compuesto por un conjunto de componentes.***



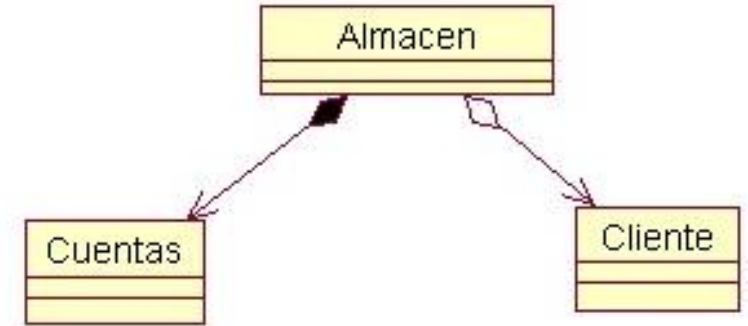
# DIAGRAMAS DE CLASE: RELACIONES

## 3. Composición:



- Es un tipo de agregación “*especial*”.
- Los componentes en una composición pertenecen sólo a un “*todo*”. Si eliminamos ese “*todo*”, los componentes desaparecen.
- Se representa igual que la agregación pero con **el rombo relleno**.
- Suele significar “se compone de...”.

➤ *Una casa se compone de una sala, un baño, un dormitorio...*

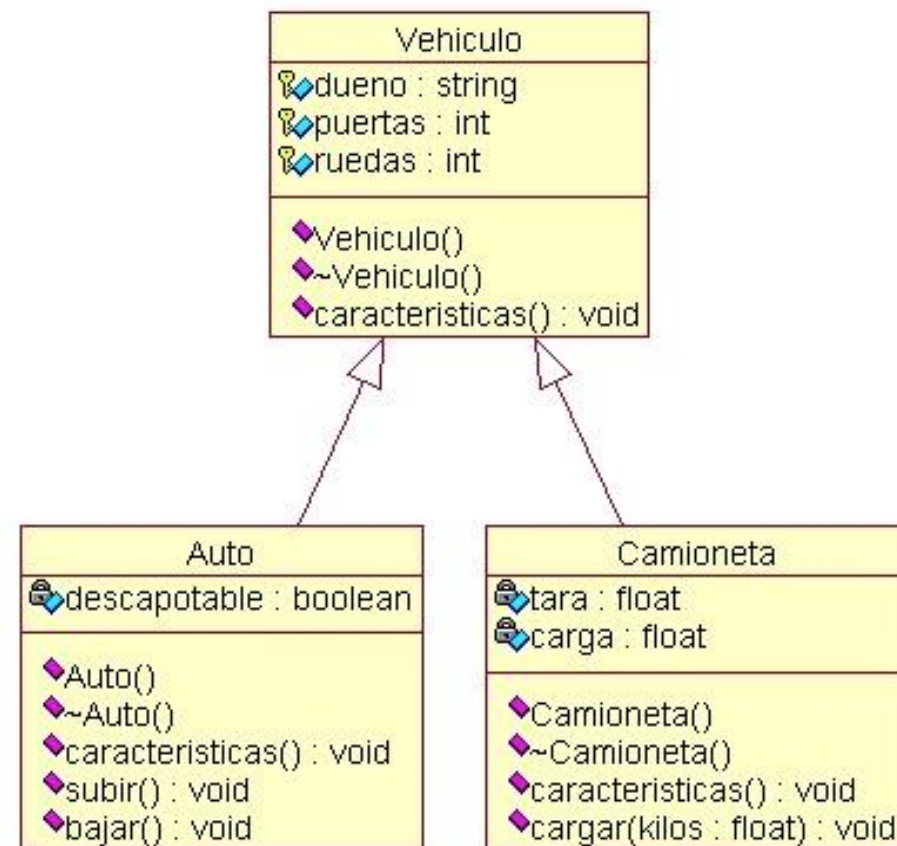


# DIAGRAMAS DE CLASE

## RELACIONES

### 4. Generalización (Herencia):

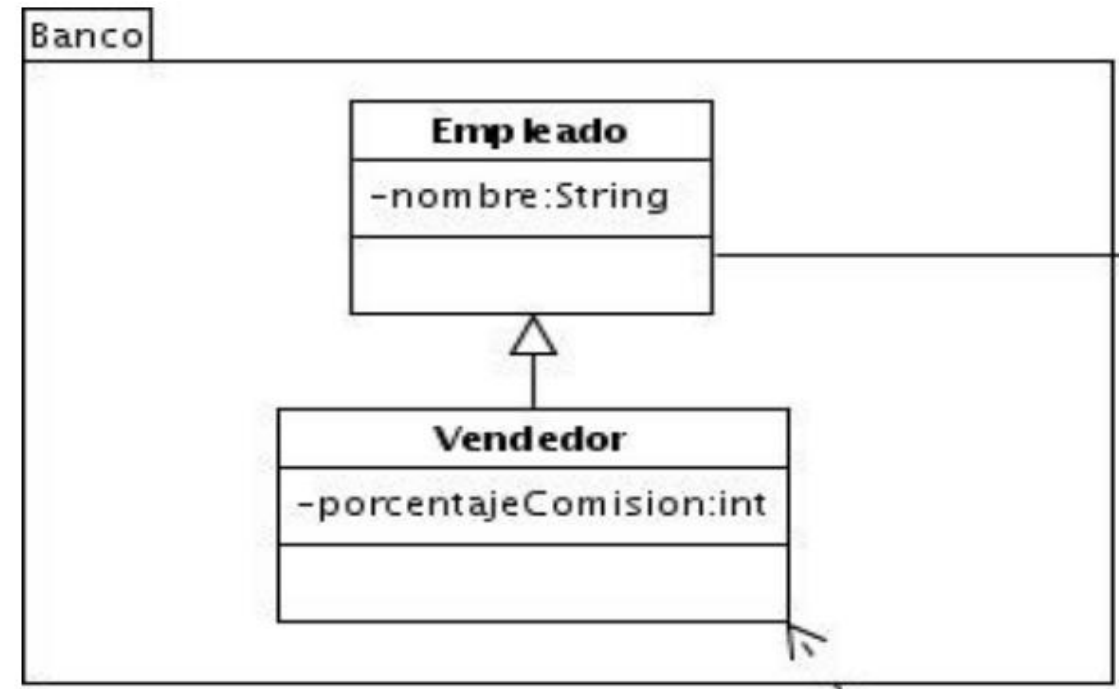
- Se representa con una flecha de cabeza hueca desde la clase hija a la clase padre.
- Significa que la clase hija extiende o amplía la funcionalidad de la clase padre.



# DIAGRAMAS DE CLASE: RELACIONES

## 5. Paquetes:

- Tienen forma de ficha con un título en la parte superior izquierda.
- Agrupa partes de nuestro sistema y clarifica el diseño.
- Tiene una finalidad meramente organizativa.

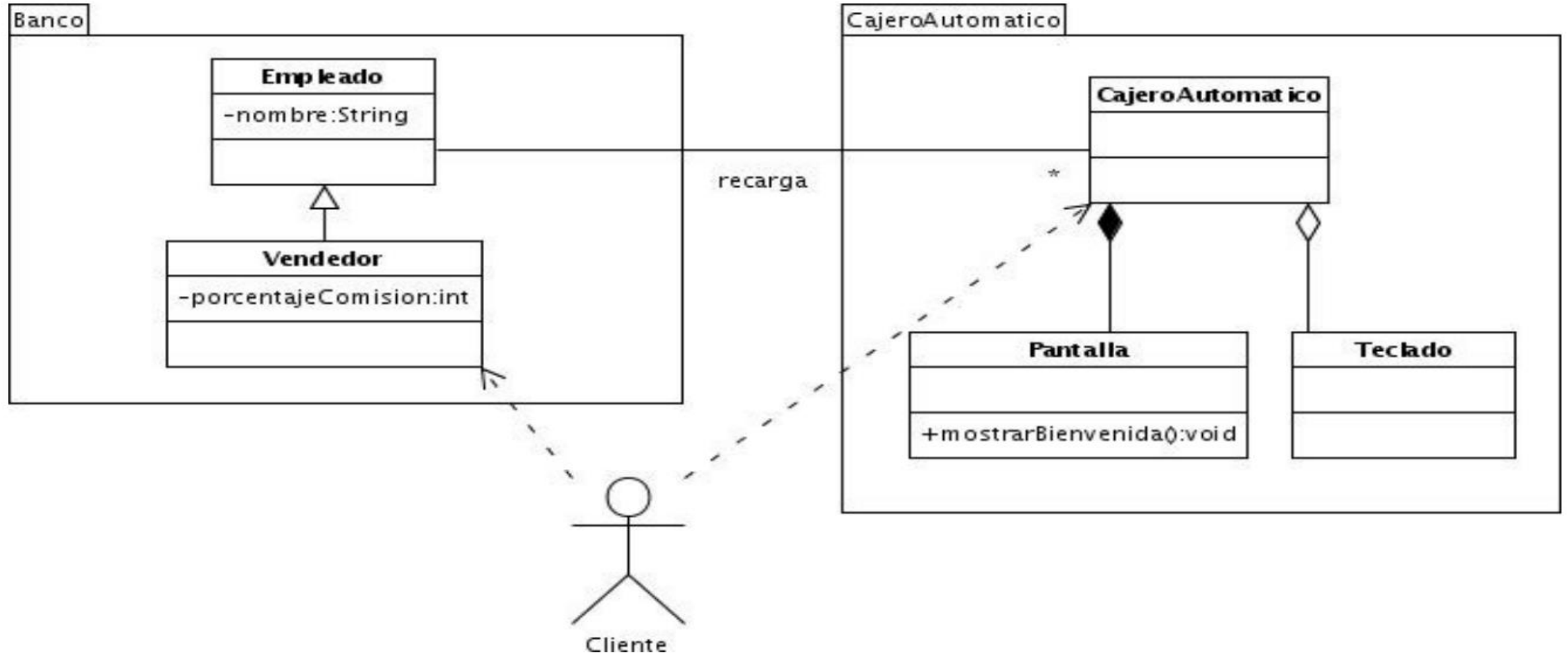


# DIAGRAMAS DE CLASE: RELACIONES

## 6. Actor:

- Es la figura de un “*monigote*”.
- **No pertenece estrictamente al diagrama** de clase pero se puede incluir para clarificar cómo se produce la interacción entre el usuario y las clases del sistema.
- Se pueden utilizar las relaciones de dependencia (flechas punteadas)

# DIAGRAMAS DE CLASE: RELACIONES





# DIAGRAMAS DE CLASE

## ELEMENTO “CLASE” - EJERCICIOS

**Ejercicio1.** Crea el diagrama de clases de un programa que solicite al usuario dos números enteros y la operación matemática simple que se desea desarrollar sobre ellos (suma o resta). Tras la recepción de los datos, el programa mostrará el resultado de la operación.

**Ejercicio 2.** Crea el diagrama de clases de un programa para calcular una línea de un ticket de supermercado. El usuario introduce el nombre del producto, el precio por unidad y el número de unidades. El programa sacará en pantalla el nombre del producto, las unidades vendidas y el precio total.

**Ejercicio 3.** Crea un diagrama de clases de un programa que genera un listín de teléfonos con los datos de los alumnos de clase. De cada alumno guarda su código, nombre, domicilio y teléfono.

# DIAGRAMAS DE CLASE

## EJERCICIOS

**Ejercicio 1:** Modela dos clases con una relación de 1 a 1 entre ellas. Observa el código que se genera.

**Ejercicio 2:** Modela dos clases con una relación de 1 a \* entre ellas. Observa el código que se genera.

**Ejercicio 3:** Modela dos clases con una relación de \* a \* entre ellas. Observa el código que se genera.

**Ejercicio 4:** Modela dos clases con una relación de 1 a 0..\* entre ellas. Observa el código que se genera.

**Ejercicio 5:** Modela cuatro clases con una relación de **herencia** entre ellas (persona, trabajador, cliente, proveedor). Observa el código que se genera.

# DIAGRAMAS DE CLASE

## EJERCICIOS

**Ejercicio 6:** Modela cuatro clases con una relación de **agregación** entre ellas (ordenador, teclado, pantalla, ratón). Observa el código que se genera.

**Ejercicio 7:** Modela dos clases con una relación de tipo **composición** entre ellas (empresa, departamentos). Observa el código que se genera.

**Ejercicio 8:** Tras observar el código que nos genera y donde lo deja. Une dos proyectos (uno Java y otro UML) y aprende a utilizar el concepto de la reingeniería inversa en los dos sentidos.

**Ejercicio 9:** Crea y utiliza desde el método main, alguna clase generada automáticamente a través del diagrama de clases.