Tecnicatura Universitaria

en Desarrollo de Software

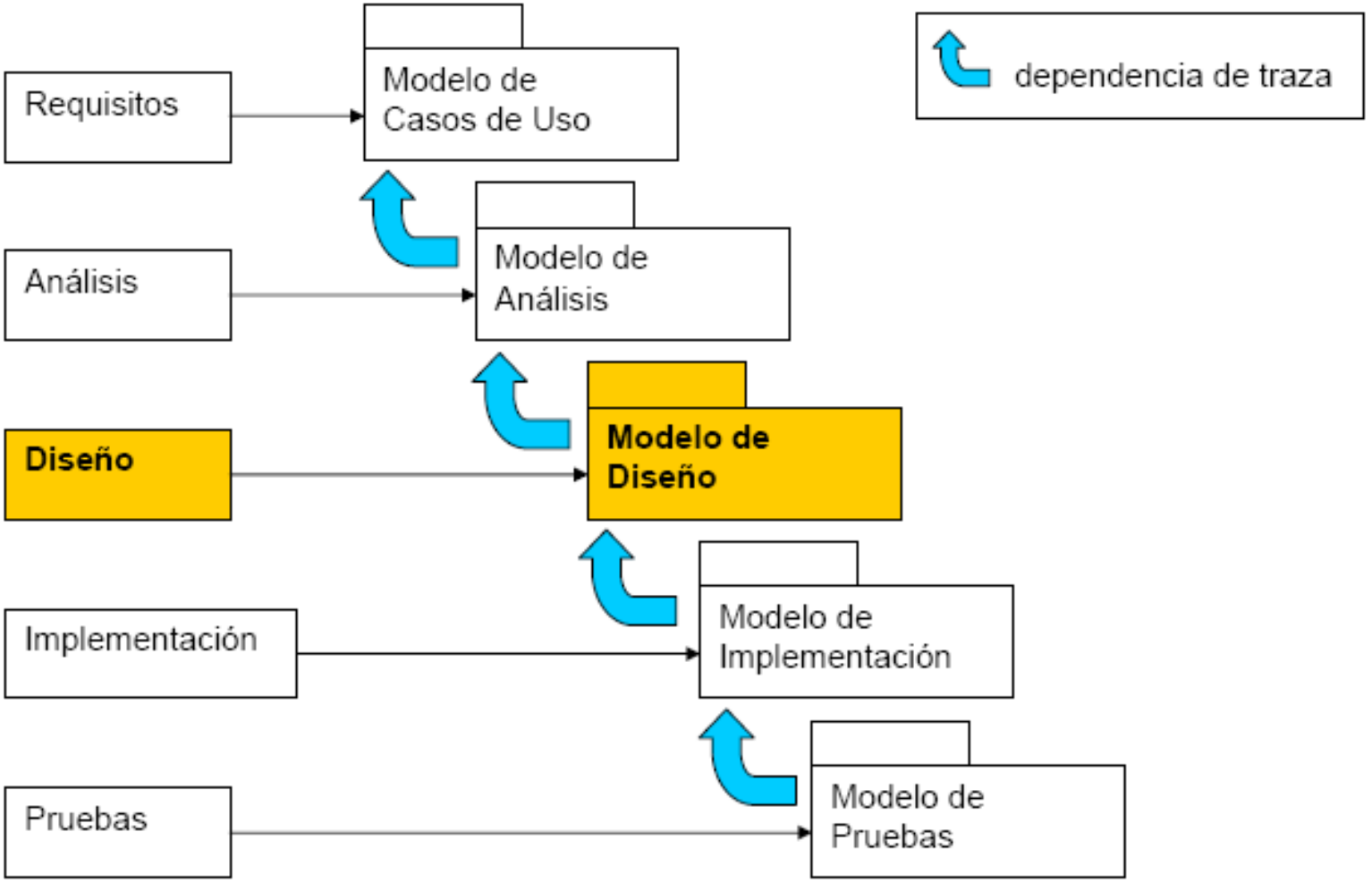
Ingeniería del Software

2019

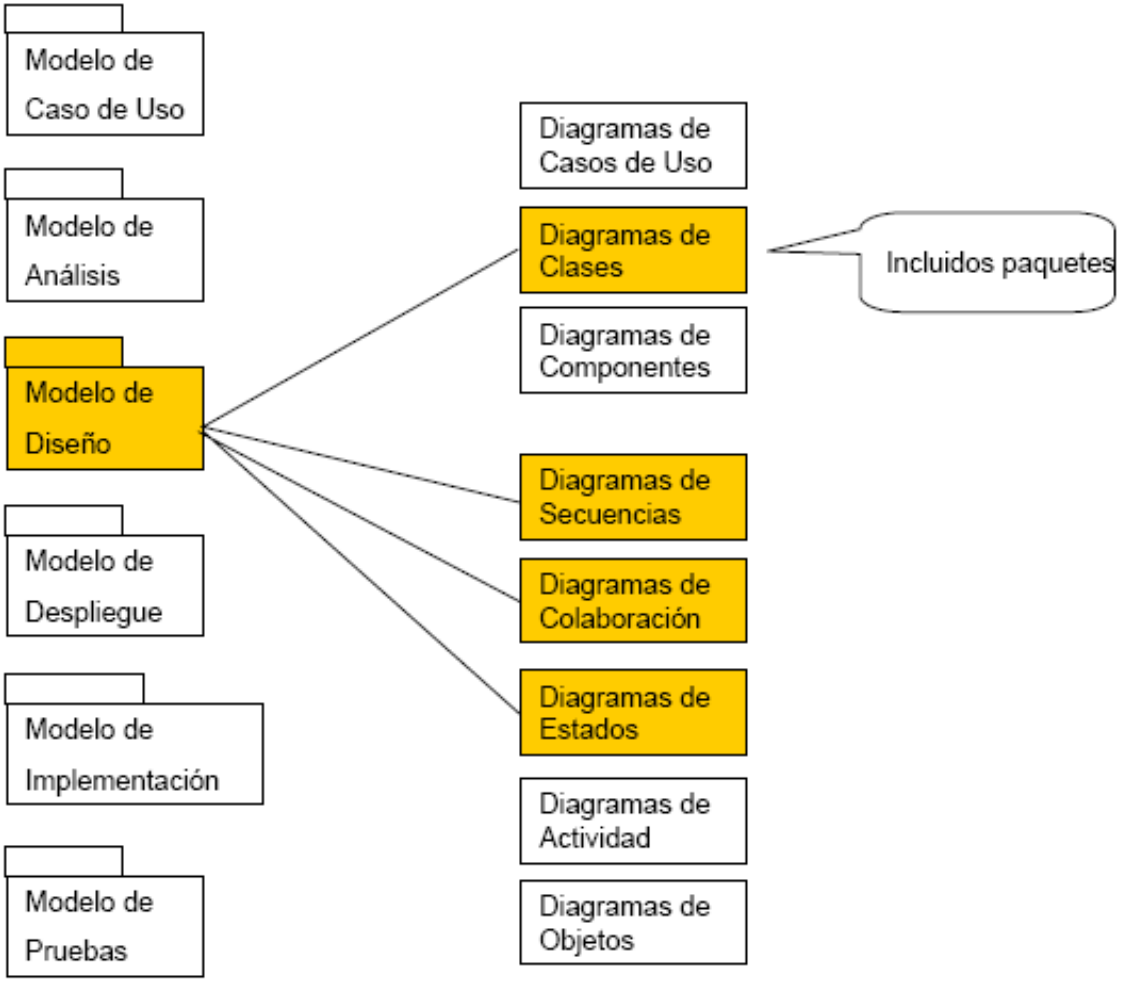
Diseño: Diagrama de Clases

Ing. Juan José Saez

Visión General



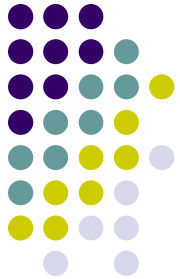
Diagramas Uml



M. de análisis Vs. M. de Diseño

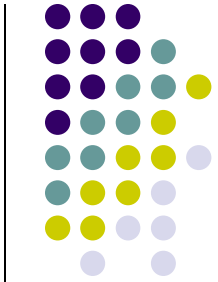
Modelo de Análisis	Modelo de Diseño
Modelo conceptual, porque es una abstracción del sistema y permite aspectos de implementación	**Modelo Físico**, porque es un plano de implementación
Genérico respecto al diseño (aplicable a varios diseños)	**No genérico**, específico para una implementación
Menos formal	**Mas formal**
Menos caro de desarrollar (ratio al análisis 1:5)	**Mas caro de desarrollar (ratio al análisis 5:1)**
Menos capas	**Mas capas**
Tres estereotipos conceptuales sobre las clases: Control, Entidad e Interfaz	**Cualquier número de estereotipos** sobre las clases, dependiendo de lenguaje de implementación
Dinámico (no muy centrado en la secuencia)	**Dinámico** (muy centrado en la secuencia)
Bosquejo del diseño del sistema, incluyendo su arquitectura	**Manifiesto del diseño** del sistema, incluyendo su arquitectura
Creado principalmente como "trabajo a pie" en talleres o similares	**Creado principalmente como "programación visual"** usando ingeniería directa e inversa con el modelo de implementación
Puede no estar mantenido durante todo el ciclo de vida del software	**Debe ser mantenido** durante todo el ciclo de vida del software
Define una estructura escencial para modelar el sistema	**Da forma al sistema** e intenta preservar la estructura definida en el análisis lo mas posible

Modelo de Diseño



- El modelo de diseño se crea tomando el modelo de análisis como entrada principal.
- Mientras que el modelo de análisis sirve como una primera aproximación del modelo de diseño, éste último funciona como un esquema de implementación.
- El modelo de diseño se adapta al entorno de implementación en cambio el de análisis no (es mas conceptual).
- El modelo de diseño también define clasificadores (clases, subsistemas e interfaces), relaciones entre esos clasificadores, y colaboraciones.

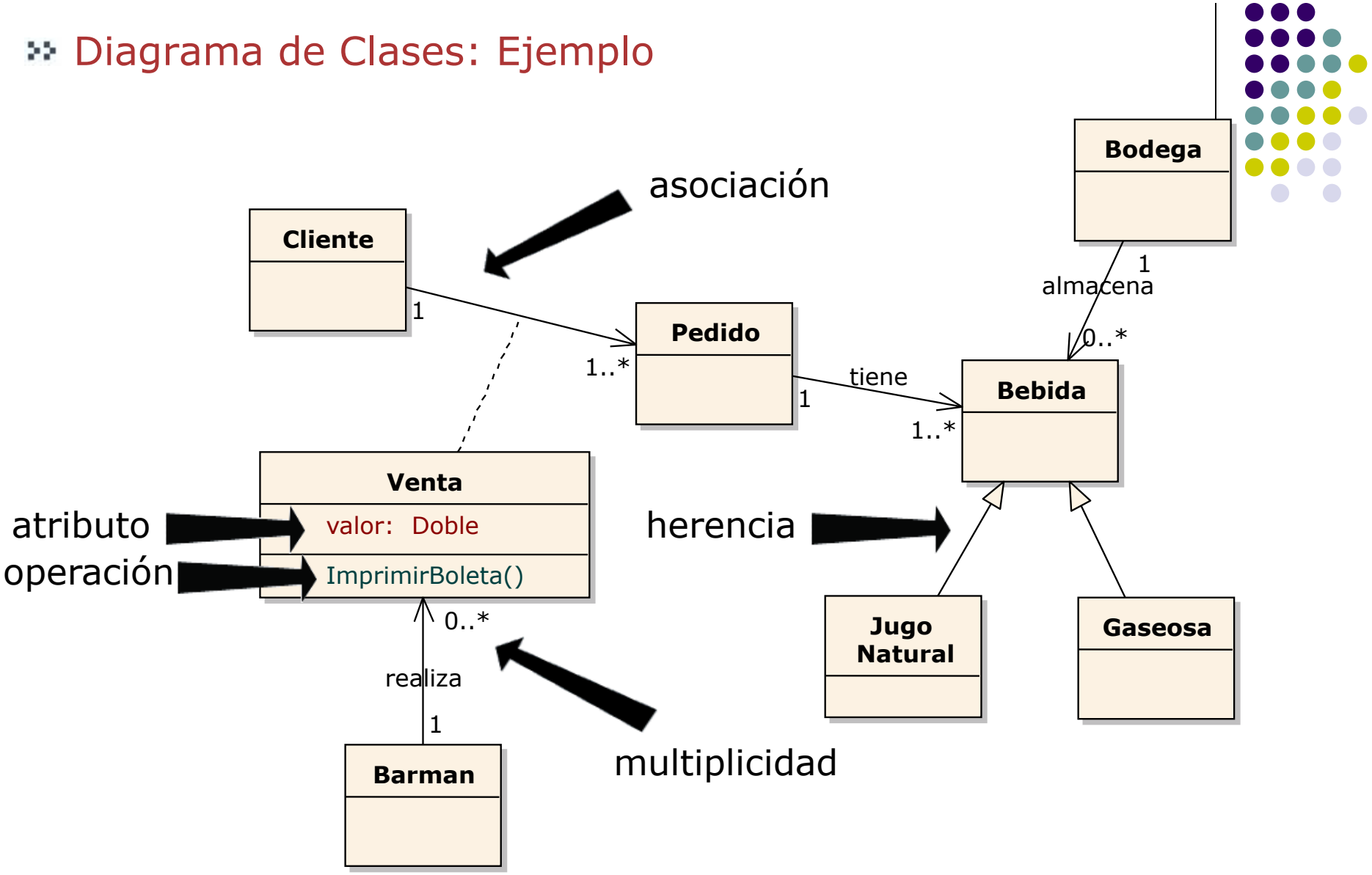
Diagrama de clases



- Sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contenimiento.

- Esta compuesto por los siguientes elementos
  - Clase: atributos, métodos y visibilidad.
  - Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

Diagrama de Clases: Ejemplo



Clase de Diseño

- Es la unidad básica que encapsula toda la información de un Objeto .

Un objeto es una instancia de una clase.

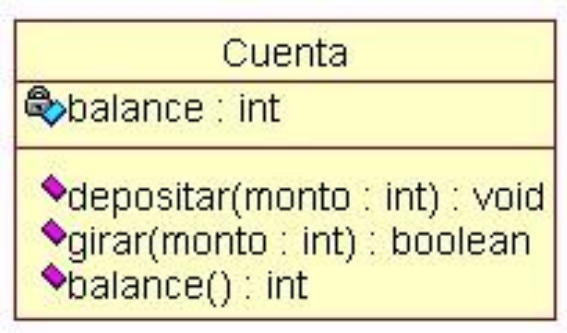
- A través de ella podemos modelar el entorno en estudio (una Casa, un Auto, una Cuenta Corriente, etc.).

Ejemplo: Una Cuenta Corriente que posee como característica:

- Balance

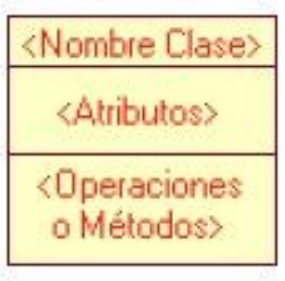
Puede realizar las operaciones de:

- Depositar
- Girar
- Balance



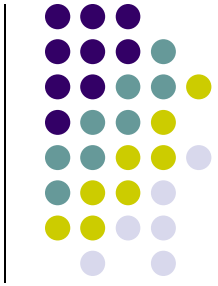
Clase de Diseño

- En UML, una clase es representada por un rectángulo que posee tres divisiones:



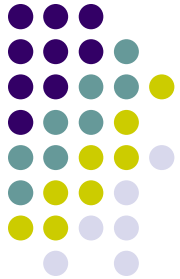
  - **Superior**: Contiene el nombre de la Clase
  - **Intermedio**: Contiene los atributos (o variables de instancia) que caracterizan a la Clase (pueden ser private, protected o public).
  - **Inferior**: Contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno (dependiendo de la visibilidad: private, protected o public).

Clase de Diseño



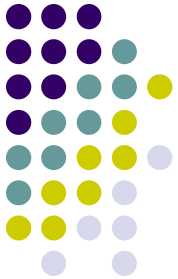
- Una clase de diseño es una abstracción de una clase de implementación
- Las operaciones, atributos, tipos, visibilidad (public, protected, private ...), etc se pueden especifican con la sintaxis del lenguaje elegido
- Las relaciones entre clases de diseño se traducen de manera directa al lenguaje:
  - generalización: herencia
  - asociaciones, agregaciones: atributos
- Se pueden postergar algunos requisitos a implementación (por ejemplo: manera de nombrar los atributos, operaciones, ...)

Atributos



- Los atributos o características de una Clase pueden ser de tres tipos, los que definen el grado de comunicación y visibilidad de ellos con el entorno, estos son:
  - **public** (+,): Indica que el atributo será visible tanto dentro como fuera de la clase, es decir, es accsesible desde todos lados.
  - **private** (-,): Indica que el atributo sólo será accesible desde dentro de la clase (sólo sus métodos lo pueden accesar).
  - **protected** (#,): Indica que el atributo no será accesible desde fuera de la clase, pero si podrá ser accesado por métodos de la clase además de las subclases que se deriven (ver herencia).

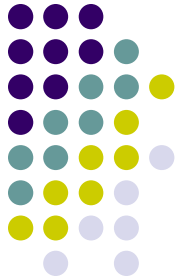
Atributos: Notación



[visibilidad] nombreAtributo [: tipo] [[multiplicidad]] [= valorIncicial]

- Ejemplos:
- +provincia: String =“San Luis”
- -origen: Punto
- -EstadoReserva: Integer [0..1]
- -idUnico: Long
- #prioridad: Entero =1

Métodos



- Los métodos u operaciones de una clase son la forma en como ésta interactúa con su entorno, éstos pueden tener las características:

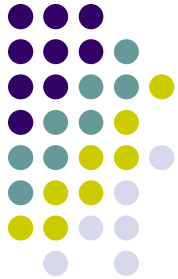
  - **public** (+,): Indica que el método será visible tanto dentro como fuera de la clase, es decir, es accsesible desde todos lados.
  - **private** (-,): Indica que el método sólo será accesible desde dentro de la clase (sólo otros métodos de la clase lo pueden accesar).
  - **protected** (#,): Indica que el método no será accesible desde fuera de la clase, pero si podrá ser accesado por métodos de la clase además de métodos de las subclases que se deriven (ver herencia).

Métodos: Notación

[visibilidad] nombreOperación [(listaParámetros)] [:tipoRetorno]

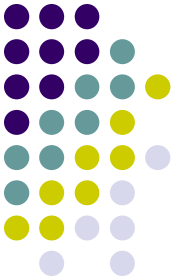
- mover(Pos1: Int, Pos2: Int)
- +añadirCurso(c:Curso):Booleano
- +ponerAlarma(t:temperatura)
- -compactar( )
- +MostrarCopiasEnEstantería( ):Int
- #comprobarErrores( )

Relaciones y Cardinalidad



- Se pueden interrelacionar dos o más clases (cada uno con características y objetivos diferentes)
- Antes es necesario explicar el concepto de cardinalidad de relaciones
- En UML, la cardinalidad de las relaciones indica el grado y nivel de dependencia, se anotan en cada extremo de la relación y éstas pueden ser:
  - **uno o muchos**: 1..* (1..n)
  - **0 o muchos**: 0..* (0..n)
  - **número fijo**: m (m denota el número).

El número de instancias de una clase que se relacionan con UNA instancia de la otra clase.



- Cada asociación tiene dos multiplicidades (una para cada extremo de la relación).
- Para especificar la multiplicidad de una asociación hay que indicar la multiplicidad mínima y la multiplicidad máxima (mínima..máxima)

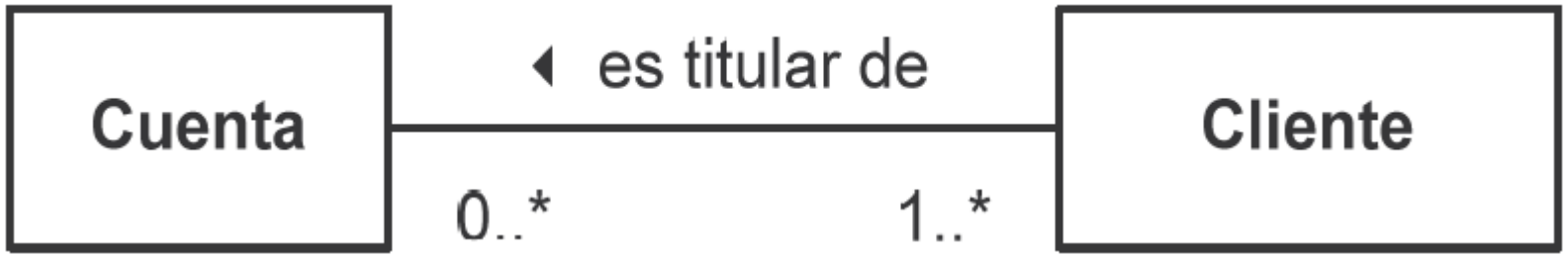
Multiplicidad	Significado
`1`	Uno y sólo uno
`0..1`	Cero o uno
`N..M`	Desde N hasta M
`*`	Cero o varios
`0..*`	Cero o varios
`1..*`	Uno o varios (al menos uno)

Relaciones y Cardinalidad

Relaciones y Cardinalidad

- Cuando la multiplicidad mínima es 0, la relación es opcional.

- Una multiplicidad mínima mayor o igual que 1 establece una relación obligatoria.



Relación opcional
Un cliente puede o no ser titular de una cuenta

Relación obligatoria
Una cuenta ha de tener un titular como mínimo

Relaciones:

Generalización (Herencia)

Relación Dependencia

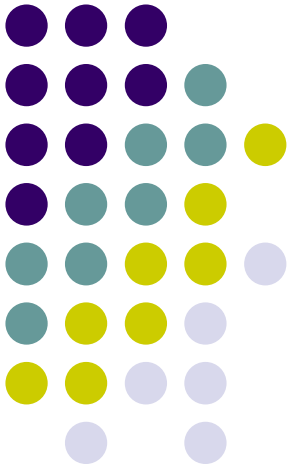
Relación Asociación

Relación Involutiva

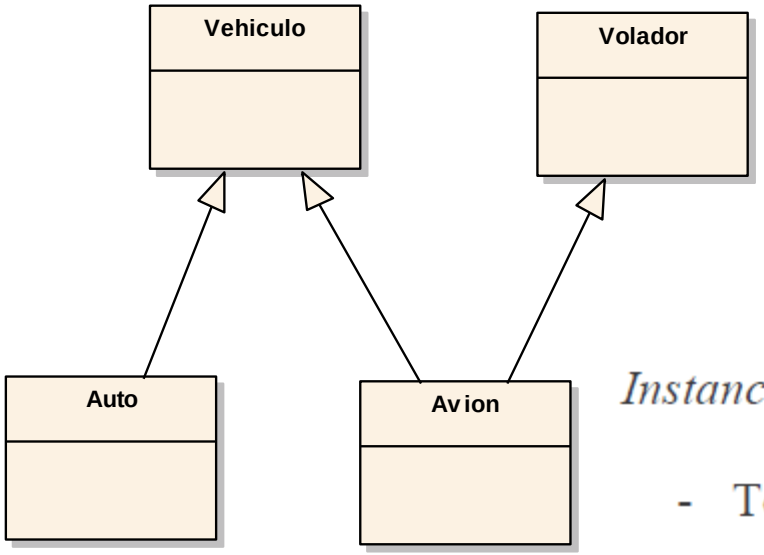
Relación Composición

Relación Agregación

Clase Asociativa



Relaciones en Diagramas de Clases



Instancias: **coches ⊂ vehículos**

- Todo coche es un vehículo.
- Algunos vehículos son coches.

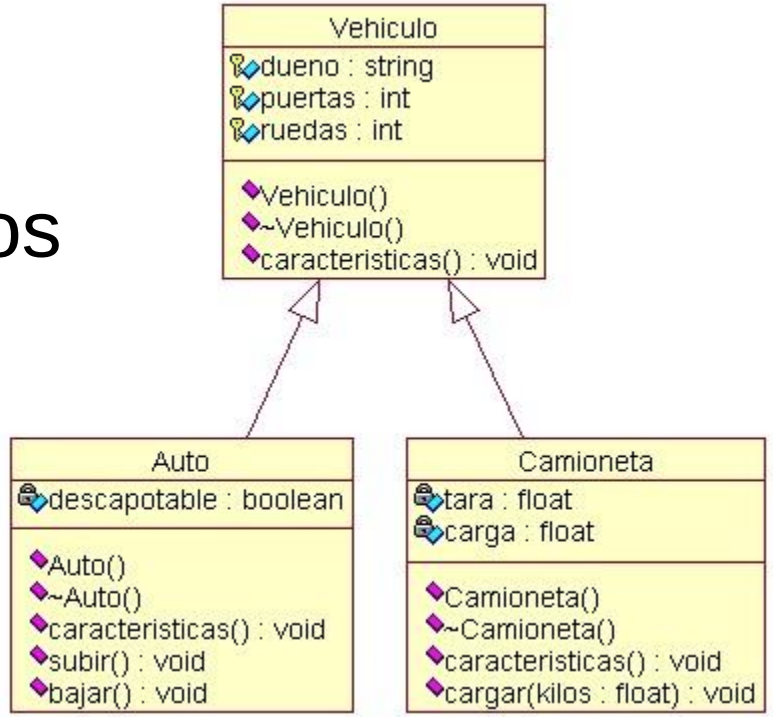
Herencia

- La clase **heredera** es una especialización de su **súper clase**
- **Herencia Múltiple**
  Una clase puede ser heredada de varias súper clases

Relaciones: Herencia

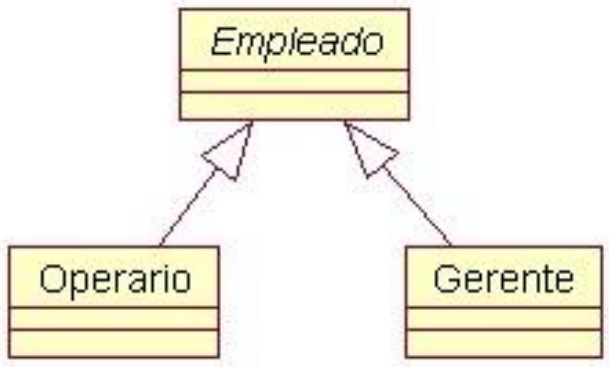
- **(Especialización/Generalización)**: Indica que una subclase hereda los métodos y atributos especificados por una Super Clase

- La Subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la Super Clase (public y protected)



Clase Abstracta

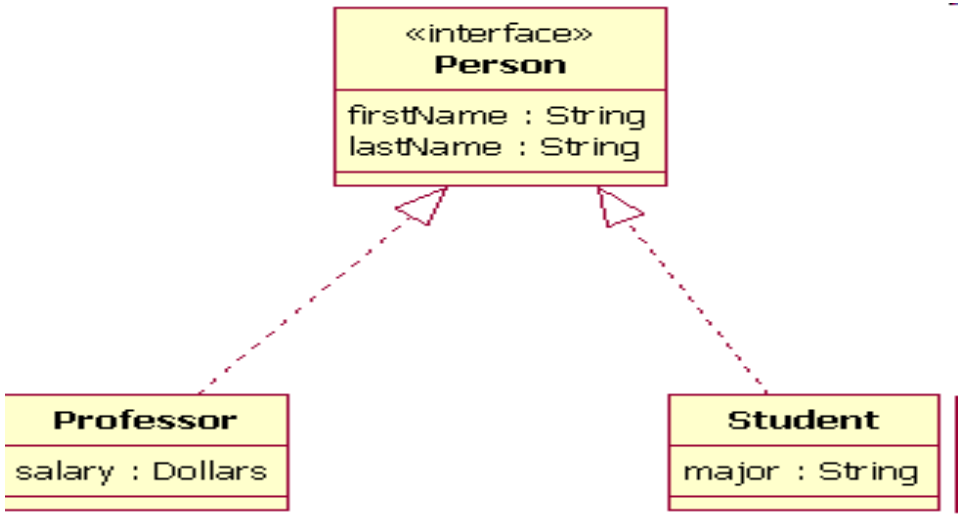
- Una clase abstracta se denota con el nombre de la clase y de los métodos con letra "itálica".
- Esto indica que la clase definida no puede ser instanciada pues posee métodos abstractos (aún no han sido definidos, es decir, sin implementación).
- La única forma de utilizarla es definiendo subclases, que implementan los métodos abstractos definidos.



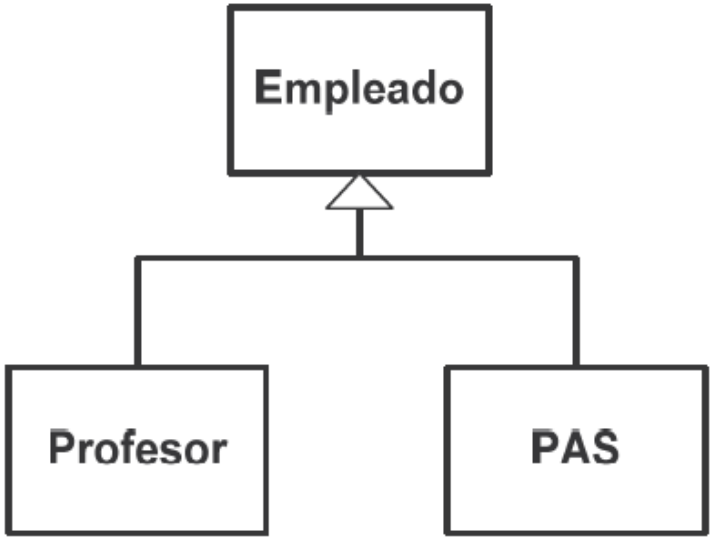
Interfaces

- Una clase tiene una instancia de su tipo, mientras que una interface debe tener al menos una clase para implantarla.
- En UML, una interface es considerada como una especialización de una clase.
- Se dibuja como una clase, pero arriba aparece un texto que indica que se trata de una interface y no de una clase.
- Una interface

  no es una clase.



Relaciones: Herencia

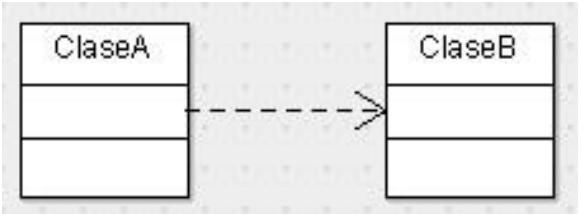



```
public class Empleado
{
    ...
}

public class Profesor extends Empleado
{
    ...
}

public class PAS extends Empleado
{
    ...
}
```

En el diagrama de clases,
los atributos, métodos y relaciones de una clase se muestran en el nivel más alto de la jerarquía en el que son aplicables.

Dependencia

- **Definición:** Es una relación de uso entre dos clases (una usa a la otra). Esta relación es la más básica entre clases y comparada con los demás tipos de relación, la mas débil.
- Se representa con una flecha discontinua que parte desde una clase y apunta a otra. El sentido de la flecha nos indica quien usa a quien.
- Podemos observar que:
  - La *ClaseA* usa a la *ClaseB*.
  - La *ClaseA* depende de la *ClaseB*.
  - Dada la dependencia, todo cambio en la *ClaseB* podrá afectar a la *ClaseA*.
  - La *ClaseA* conoce la existencia de la *ClaseB* pero la *ClaseB* desconoce que existe la *ClaseA*.

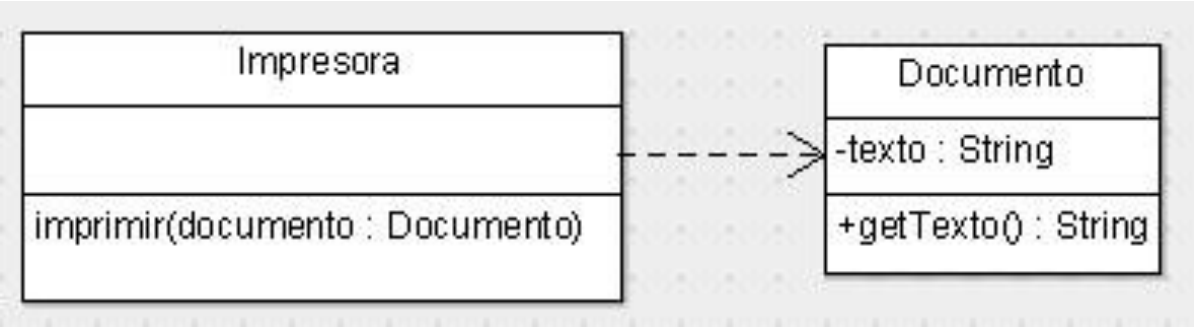


Dependencia

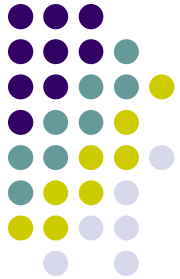
- **Dependencia**: Es una relación mas débil que una asociación simple. Se usa para indicar la relación de una clase cliente hacia la bibliotecas de métodos o librerías que proveen servicios.
  - Cliente es el objeto que solicita el servicio
  - Servidor es el objeto que provee esl servicio solicitado
- Ej. Bibliotecas o DLL io.sys, Math.sys

Dependencia

- En la practica este tipo de relación se interpreta como que la *ClaseA* hace uso de la *ClaseB* ya sea instanciandola directamente, o bien, recibiéndola como parámetro de entrada en uno de sus métodos.
- **Ejemplo Práctico:** Supongamos lo siguiente:
  - Tenemos una clase *Impresora.*.
  - Tenemos una clase *Documento* con un atributo *texto*.
  - La clase *Impresora* se encarga de *imprimir* los *Documentos.*
  - Para esto generamos una relación de dependencia:



Traduciendo a código: (Java)



- Documento.java

```java
/* Clase Documento */
class Documento {
    private String texto;

    public Documento(String texto) {
        this.texto = texto;
    }

    public String getTexto() {
        return this.texto;
    }
}
```

Traduciendo a código: (Java)

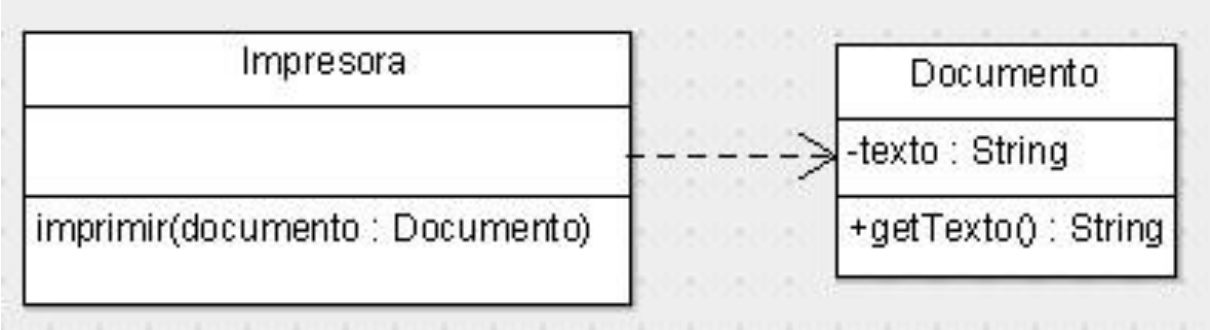
- Impresora.java

```java
/* Clase Impresora */
class Impresora {

    public Impresora() {

    }

    public void imprimir(Documento documento) {
        String texto = documento.getTexto();
        System.out.println(texto);
    }

}
```

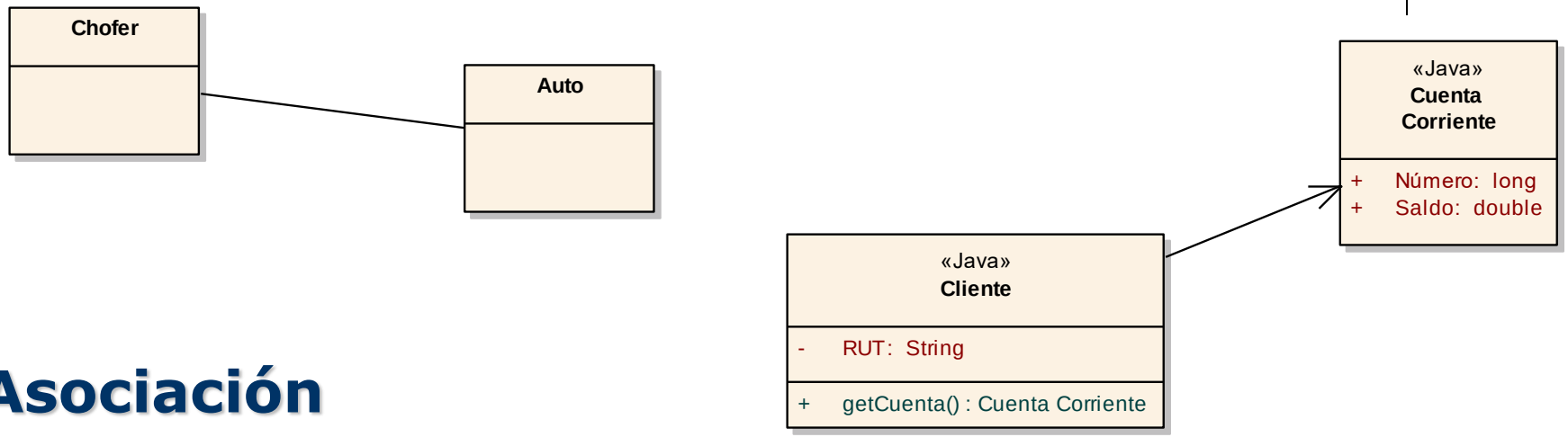
Traduciendo a código: (Java)

- Solo bastaría instanciar las clases y hacer uso del método que usa a la otra clase como parámetro:

```
1  Documento miDocumento = new Documento("Hello World!");
2  Impresora miImpresora = new Impresora();
3  miImpresora.imprimir(miDocumento);
```



Relaciones en Diagramas de Clases

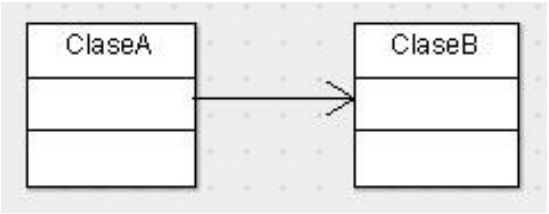


Asociación

- Representa la **relación genérica** entre dos clases
- Permite asociar objetos que colaboran entre si.
- Cabe destacar que no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro.
- Ejemplo: Un cliente puede tener asociadas muchas Ordenes de Compra, en cambio una orden de compra solo puede tener asociado un cliente.

Asociación

- **Definición:** Es una relación de estructura entre clases, es decir, una entidad se construye a partir de otra u otras.
- Aunque este tipo de relación es mas fuerte que la Dependencia es más débil que la Agregación, ya que el tiempo de vida de un objeto no depende de otro.
- **Representación UML:**Se representa con una flecha continua que parte desde una clase y apunta a otra. El sentido de la flecha nos indica la clase que se compone (base de la flecha) y sus componentes (punta de la flecha).

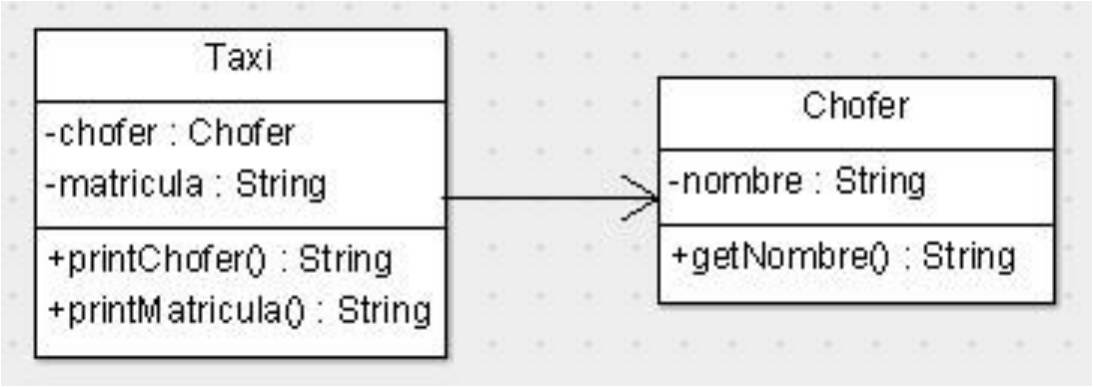


- Podemos observar que:
  - La *ClaseA* depende de la *ClaseB*.
  - La *ClaseA* está asociada a la *ClaseB*.
  - La *ClaseA* conoce la existencia de la *ClaseB* pero la *ClaseB* desconoce que existe la *ClaseA*.
  - Todo cambio en la *ClaseB* podrá afectar a la *ClaseA*.

Asociación

- **Esto significa que la *ClaseA* tendrá como atributo un objeto o instancia de la *ClaseB* (su componente).**
- **La*ClaseA* podrá acceder a las funcionalidades o atributos de su componente usando sus métodos.**

- **Ejemplo Práctico**
  - Tenemos una clase *Taxi* con un atributo *matricula*.
  - Tenemos una clase *Chofer* con un atributo *nombre*.
  - Cada *Taxi* necesita ser conducido por un *Chofer.*
  - *Taxi* necesita acceder a algunos de los atributos de su *Chofer* (por ejemplo, su nombre).



Traduciendo a código: (Java)

- Chofer.java

```java
/* Clase Chofer */
class Chofer {
    private String nombre;

    public Chofer(String nombre) {
        this.nombre = nombre;
    }

    public String getNombre() {
        return this.nombre;
    }
}
```

Traduciendo a código: (Java)

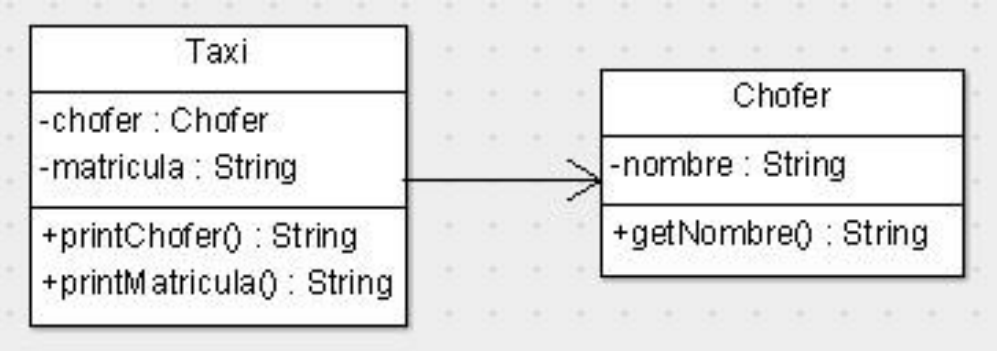
- Taxi.java

```java
/* Clase Taxi */
class Taxi {
    private Chofer chofer;
    private String matricula;

    public Taxi(Chofer chofer, String matricula) {
        this.chofer = chofer;
        this.matricula = matricula;
    }

    public void printMatricula() {
        System.out.println(this.matricula);
    }

    public void printChofer() {
        String nombreChofer = this.chofer.getNombre();
        System.out.println(nombreChofer);
    }
}
```

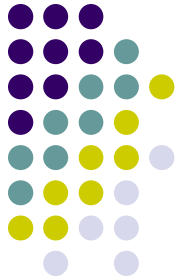
Traduciendo a código: (Java)

- Solo bastaría instanciar las clases y hacer uso de sus métodos:

```
1  Chofer miChofer = new Chofer("Pedro");
2  Taxi miTaxi = new Taxi(miChofer, "AHJ-1050");
3  miTaxi.printChofer();
4  miTaxi.printMatricula();
```



Agregación y Composición



- Para modelar objetos complejos, no bastan los tipos de datos básicos que proveen los lenguajes: enteros, reales y secuencias de caracteres.
  - Se requiere componer objetos que son instancias de clases definidas por el desarrollador de la aplicación.
  - Tenemos dos posibilidades

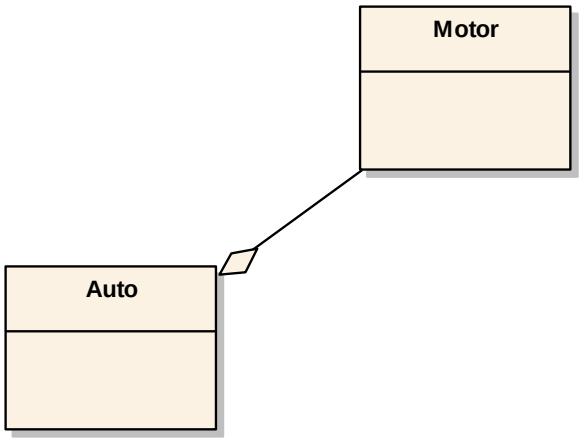
Asociación

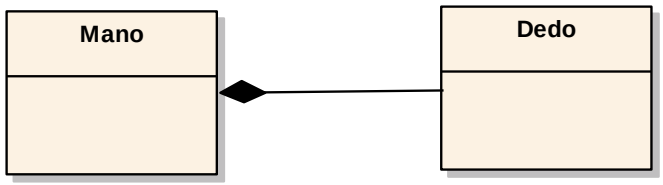
- Objeto de una clase **SE RELACIONA** con un objeto de otra clase

Agregación

- Objeto de una clase **CONTIENE** un objeto de otra clase

Relaciones en Diagramas de Clases

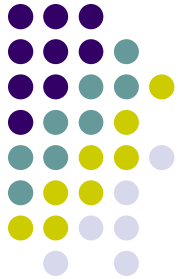




Agregación, Composición

- Representan una **asociación "fuerte"** entre dos clase
- No existe una separación estricta entre asociación y agregación o composición
  - Asociación representa una relación en el sentido más general
  - Agregación puede significar que una clase está compartida
  - Composición indica un vínculo exclusivo y permanente

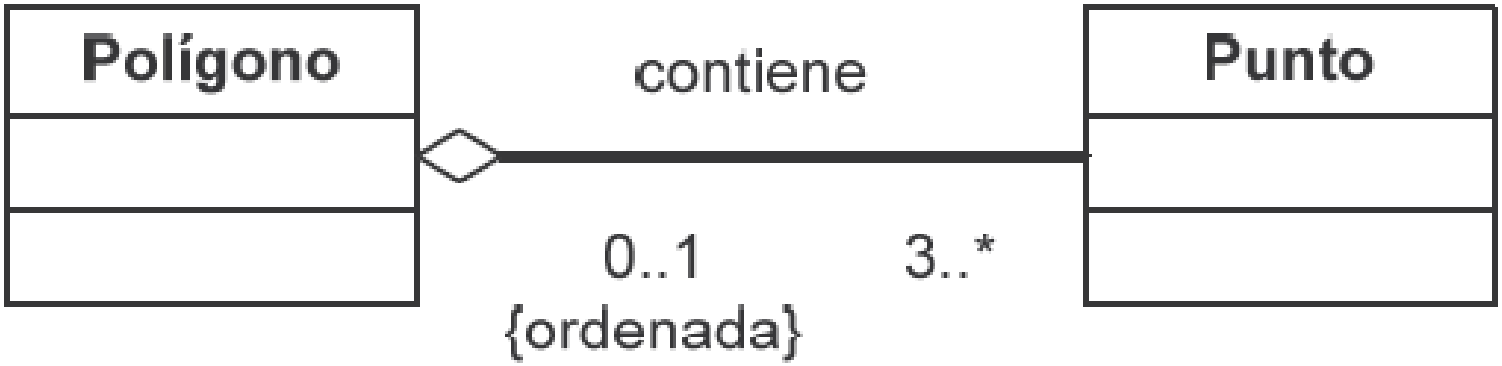
Relaciones: Agregación 



- **Por Referencia**: Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye.
- Este tipo de relación es comúnmente llamada **Agregación** (el objeto base utiliza al incluido para su funcionamiento).
- Pero al destruir el objeto principal, no afecta a los que lo componen, pues solo se rompe el enlace (referencia) hacia el otro objeto.

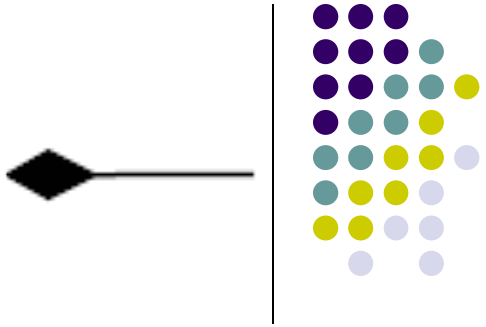
Agregación

Las partes pueden formar parte de distintos agregados.



Gráficamente,
se muestran como asociaciones con un rombo en uno de los extremos.

Relaciones: Composición



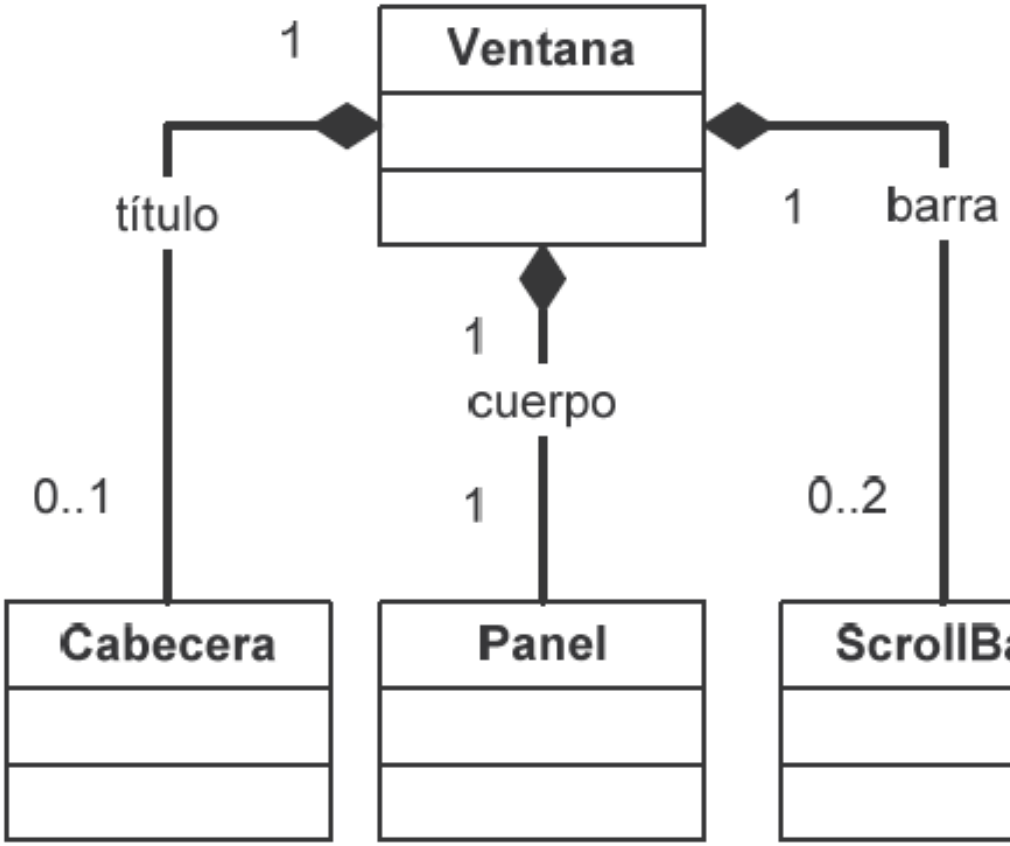
- **Por Valor**: Es un tipo de relación estática, en donde el tiempo de vida del objeto incluido esta condicionado por el tiempo de vida del que lo incluye.
- Este tipo de relación es comúnmente llamada **Composición** (el Objeto base se construye a partir del objeto incluido, es "parte/todo").
- Un objeto formado por otro que existe exclusivamente para formar al otro objeto, al destruir el objeto principal, se rompen todos los que lo componen en cascada.

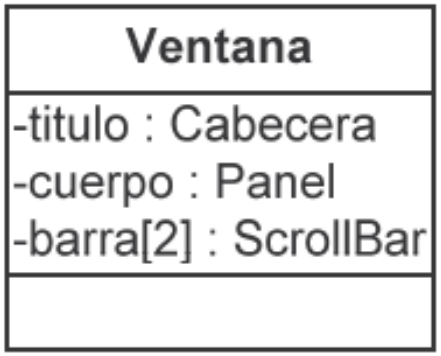
Composición

Agregación disjunta y estricta:

Las partes sólo existen asociadas al compuesto

(sólo se accede a ellas a través del compuesto)





Ejemplo

- Un Almacén posee Clientes y Cuentas (los rombos van en el objeto que posee las referencias).
- Cuando se destruye el Objeto Almacén también son destruidos los objetos Cuenta asociados, en cambio no son afectados los objetos Cliente asociados.
- La composición (por Valor) se destaca por un rombo relleno.
- La agregación (por Referencia) se destaca por un rombo transparente.

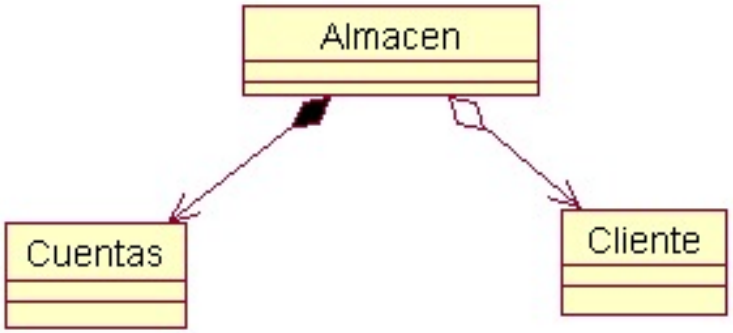
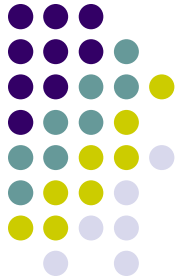


Diagrama de Clase: Agregación y Composición

- Cada agregación es un tipo de asociación.
- Cada composición es una forma de agregación.

AGREGACIÓN BASICA



- Es un tipo especial de asociación utilizado para modelar una relación “whole to its parts”.
- Por ejemplo, Coche es una entidad “whole” y Llanta es una parte del Coche.
- Una asociación con una agregación indica que una clase es parte de otra clase.
- En este tipo de asociación, la clase hijo puede sobrevivir sin su clase padre.

AGREGACIÓN BASICA

Para representar una relación de agregación, se dibuja una línea sólida de la clase padre (total) a la clase hijo (parte), y con un diamante en el lado de la clase padre.

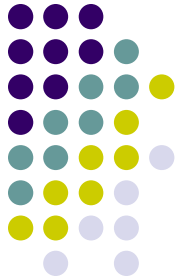
Una llanta puede existir sin automóvil

AGREGACIÓN/COMPOSICIÓN

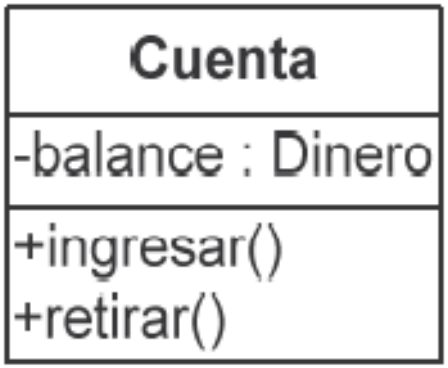
- La vida de una instancia de la clase hijo depende de la vida de una instancia padre.
- A diferencia de la agregación básica, para representarla el diamante no es hueco.
- Una instancia de la clase Company debe tener al menos una en la clase Departamento.

• En este tipo de relaciones, si una la instancia Company se elimina, automáticamente la instancia Departamento también se elimina.

•La clase hijo solo puede relacionarse con una instancia de la clase padre.

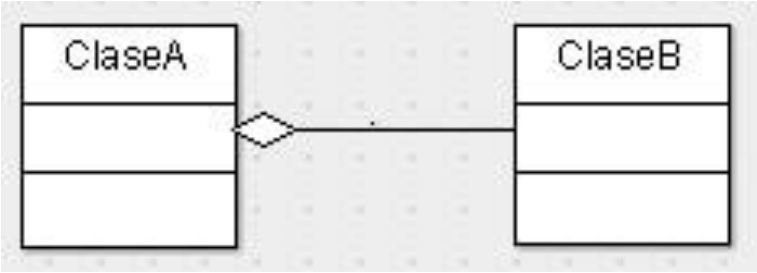


```
class Cuenta
{
  private Dinero balance;

  public void ingresar (Dinero cantidad)
  {
    balance += cantidad;
  }

  public void retirar (Dinero cantidad)
  {
    balance -= cantidad;
  }

  public Dinero getSaldo ()
  {
    return balance;
  }
}
```



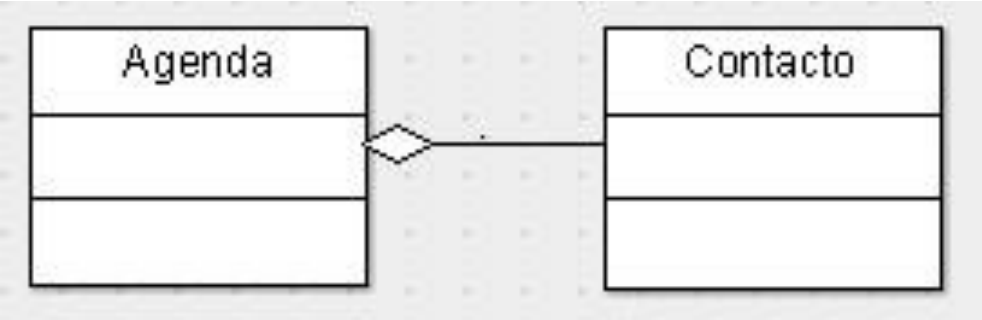
Relación de Agregación

- **Agregación:** Es muy similar a la relación de Asociación.
- Varía en la multiplicidad, ya que en lugar de ser una relación "uno a uno" es de "uno a muchos".
- **Representación UML :**Se representa con una flecha que parte de una clase a otra en cuya base hay un rombo de color blanco.



Agregación: Ejemplo

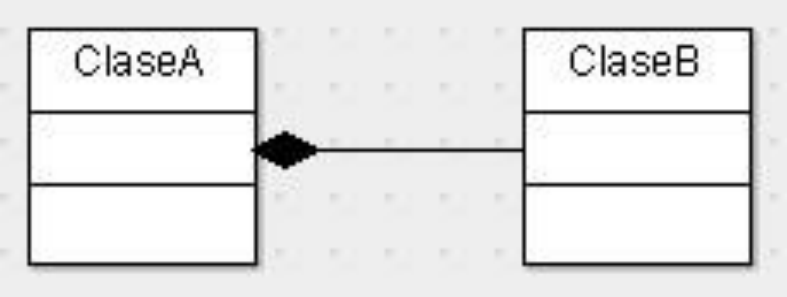
- La *ClaseA* agrupa varios elementos del tipo *ClaseB.*
- **Ejemplo**
  - Tenemos una clase *Agenda.*
  - Tenemos una clase *Contacto*.
  - Una *Agenda* agrupa varios *Contactos*.



Relación de Composición:

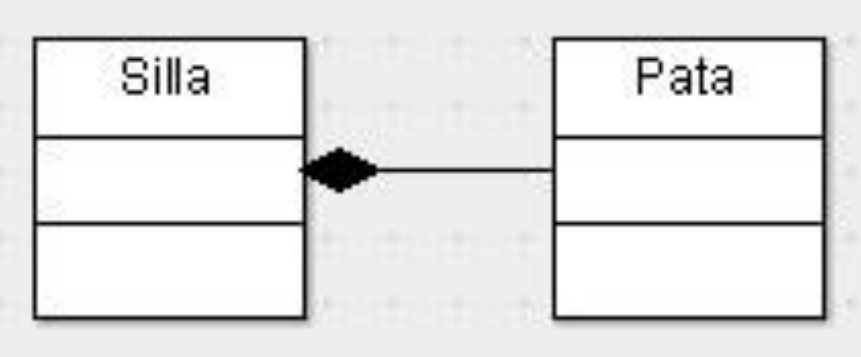
- **Composición:** Similar a la relación de Agregación solo que la Composición es una relación mas fuerte. Aporta documentación conceptual ya que es una "relación de vida", es decir, el tiempo de vida de un objeto está condicionado por el tiempo de vida del objeto que lo incluye.
- **Representación UML** Se representa con una flecha que parte de una clase a otra en cuya base hay un rombo de color negro.

- La *ClaseA* agrupa varios elementos del tipo *ClaseB.*
- El tiempo de vida de los objetos de tipo *ClaseB* está condicionado por el tiempo de vida del objeto de tipo *ClaseA*.



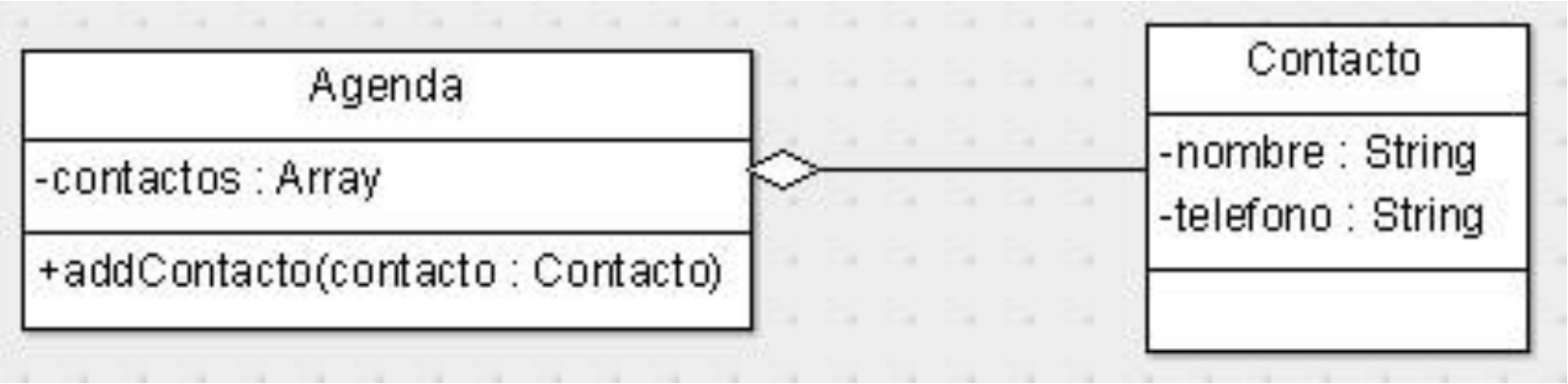
Composición: Ejemplo

- Tenemos una clase *Silla*.
- Un objeto *Silla* está a su vez compuesto por cuatro objetos del tipo *Pata*.
- El tiempo de vida de los objetos *Pata* depende del tiempo de vida de *Silla*, ya que si no existe una *Silla* no pueden existir sus *Patas*.

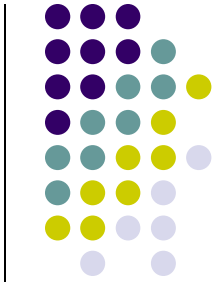


Traduciendo a Código

- ***La forma de traducir ambos tipos de relación a código es tener un atributo en la clase compuesta donde almacenaremos una colección de los objetos que la componen***.
- Dependiendo del lenguaje podremos utilizar diferentes estructuras de datos que nos permitan almacenar la colección de objetos, aunque generalmente se utilizan arreglos para este fin.
- Además debemos de proporcionar un método para agregar elementos a la colección.
- En el ejemplo de la *Agenda*:

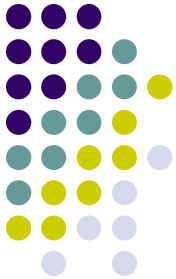


Código traducido a PHP: Contacto.php



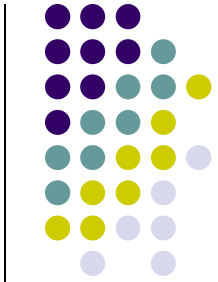
```php
<?php
/* Clase Contacto */
Class Contacto
{
    private $_nombre;
    private $_telefono;

    public function __construct($nombre, $telefono) {
        $this->_nombre = $nombre;
        $this->_telefono = $telefono;
    }
}
?>
```

Código traducido a PHP: Agenda.php



```php
<?php
/* Incluimos la clase Contacto */
require_once 'Contacto.php';

/* Clase Agenda */
class Agenda
{
    private $_contactos = array();

    public function __construct() {

    }

    public function addContacto(Contacto $contacto)
    {
        $this->_contactos[] = $contacto;
    }
}
?>
```

Código traducido a PHP: En la práctica

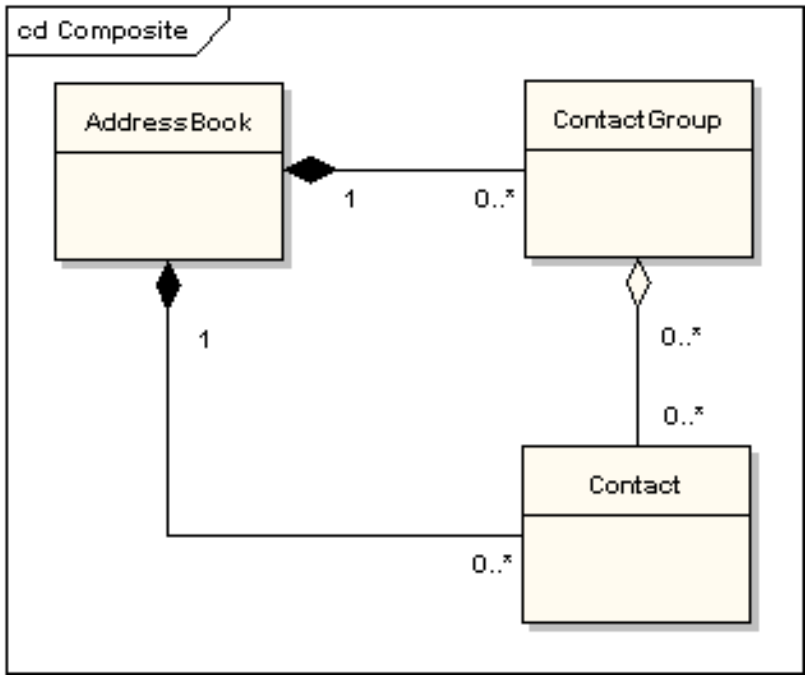


- Podemos entonces crear una instancia de Agenda, y luego ir agregando a esa agenda los contactos.

```php
<?php
/* Creamos una instancia de Agenda */
$miAgenda = new Agenda();

/* Agregamos algunos Contactos*/
$miAgenda->addContacto(new Contacto("Pepe", "3-21-54-87"));
$miAgenda->addContacto(new Contacto("Juan", "3-32-65-98"));
$miAgenda->addContacto(new Contacto("Luis", "3-78-13-46"));
?>
```

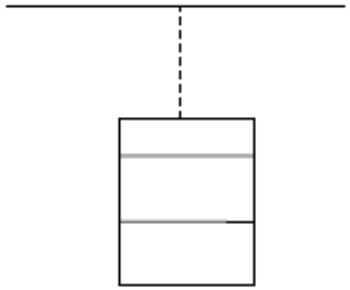
Diferencia entre Agregación débil y fuerte (Composición)

- Un libro de direcciones esta conformado de múltiples contactos y grupos de contacto
- Un contacto se puede incluir en más de un grupo de contacto.



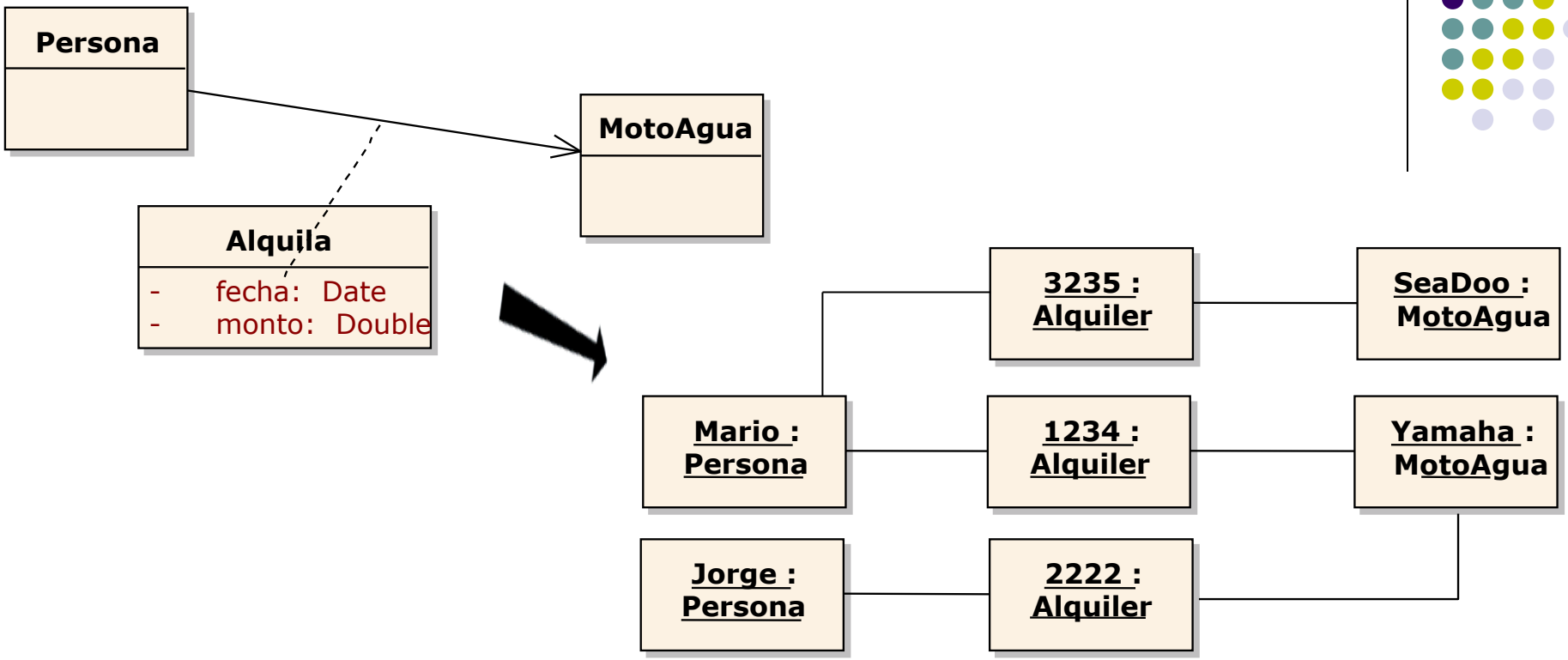
- Si elimina un libro de direcciones, todos los contactos y grupos de contactos se eliminarán también;
- Si elimina un grupo de contacto, ningún contacto se eliminará.

Case Asociativa



- Describe una asociación que refiere a una familia de relaciones entre objetos sobre las que se perciben propiedades que son propias de las relaciones.
- Existen asociaciones que contienen además de información de las clases asociadas, información propia de la relación.
- Ej: una reserva, préstamo, factura de compra, etc.

Relaciones en Diagramas de Clases

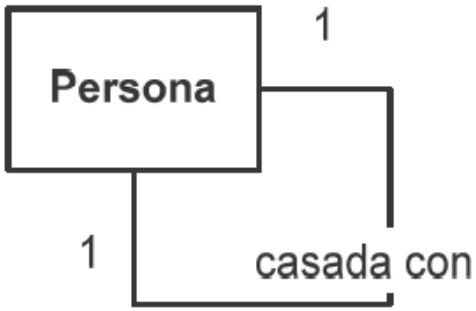


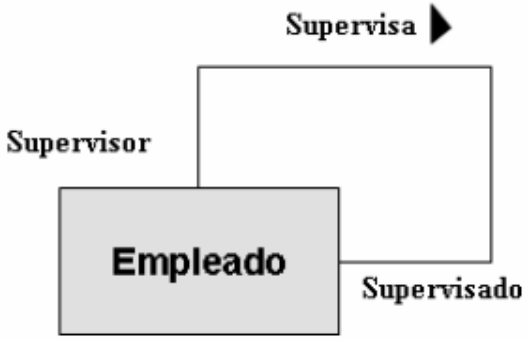
Clase asociativa

- Es una clase creada de una **asociación**
- Enlace entre dos objetos corresponden al objeto asociado
- Se usa cuando la asociación no es suficiente para describir la relación
- Habitualmente se le agregarán atributos o aún métodos

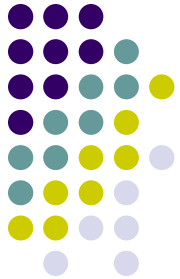
Relaciones Involutivas

- Pueden existir relaciones recursivas entre objetos de una misma clase, como la de los empleados que supervisan a otros empleados. O personas casadas contras personas.
- Aquí una misma clase aparece en ambos extremos de la relación.





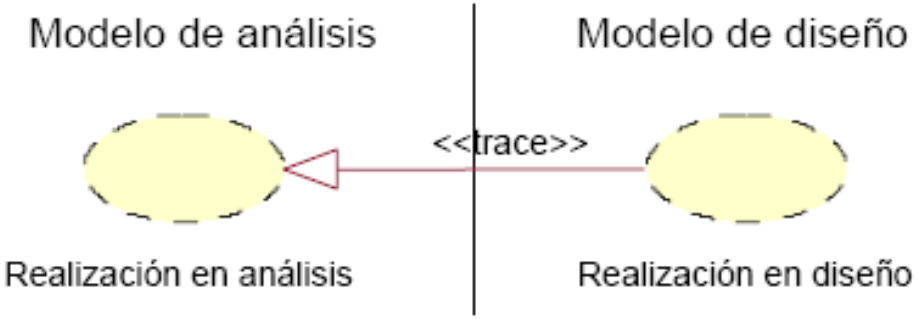
Construyendo el Diagrama de Clases



1. Identificar las **clases**
2. Mostrar los atributos y operaciones
3. Identificar, nombrar y definir las **asociaciones** entre pares de clases. Etiquetar asociaciones y en caso necesario los roles
4. Tener cuidado con clases reflexivas, asignar multiplicidad.
5. Evaluar cada asociación para determinar si debe ser una agregación y cada agregación para ver si debe ser una composición
6. Evaluar las clases para posible generalización (herencia).

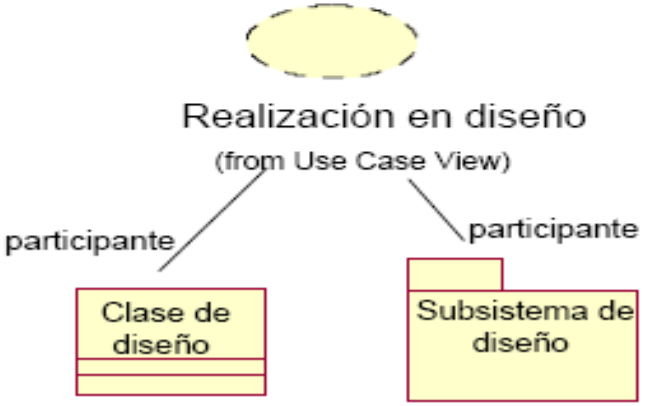
Realización CU Diseño

- Es una colaboración que describe cómo se realiza en diseño un caso de uso en términos de clases de diseño y sus interacciones



Realización CU Diseño

- La realización en diseño de un caso de uso, incluye:
  - diagramas de clases: clases participantes
  - diagramas de interacción: escenarios del caso de uso
  - Requisitos de implementación
  - Opcionalmente, subsistemas e interfaces



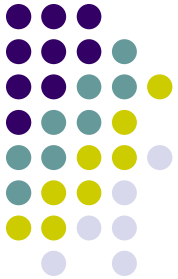
Diagramas de Interacción

Diagramas de interacción

- La secuencia de acciones en un caso de uso comienza cuando un actor envía un mensaje a un objeto de diseño.

- Utilizar mejor diagramas de secuencia que de colaboración. Nos interesa la secuencia cronológica de las interacciones.

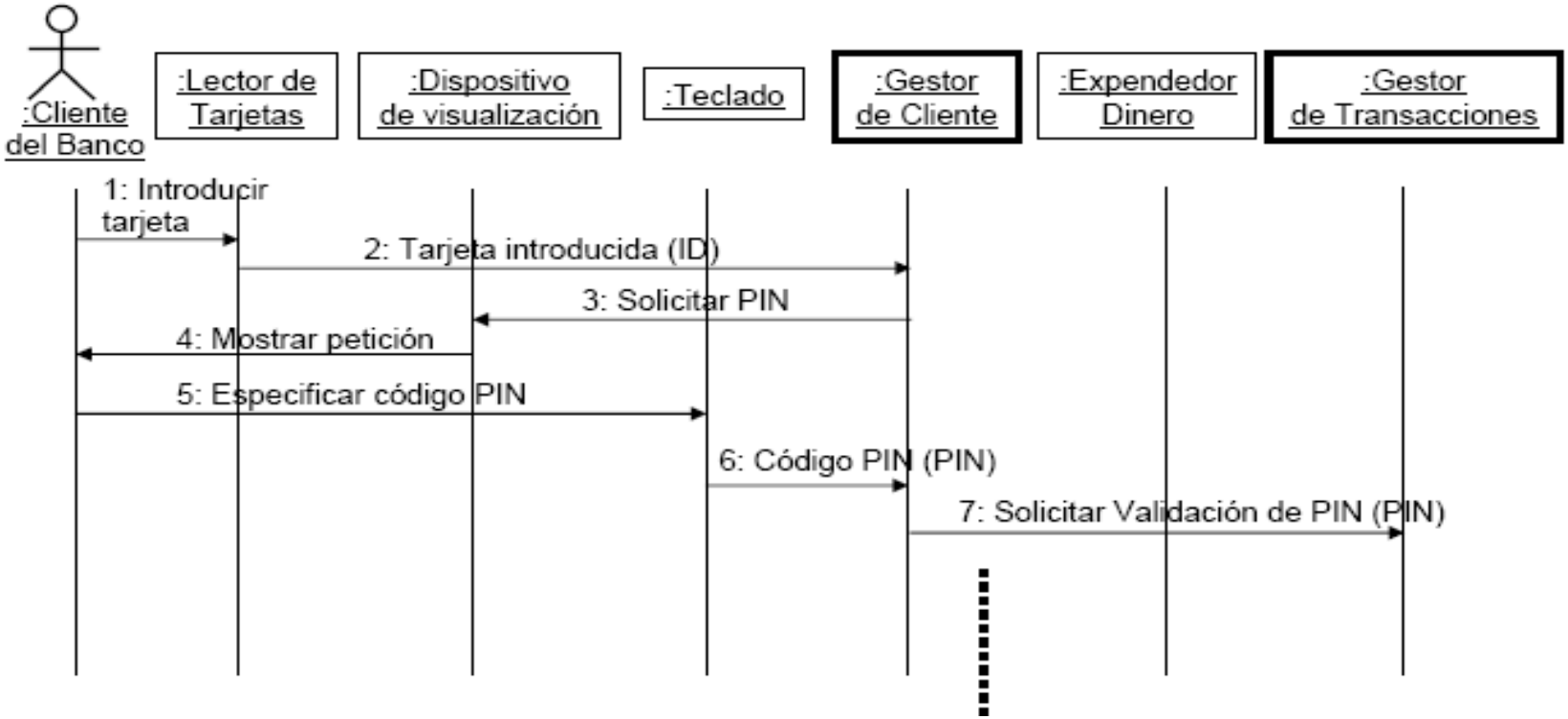
Diagrama de Secuencia (Diseño)



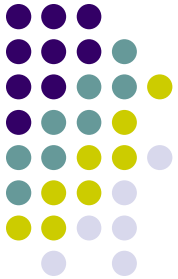
- Utilizamos principalmente **diagramas de secuencia** para modelar las interacciones entre los objetos de diseño.
- El diagrama de secuencia muestra como el control pasa de un objeto a otro a medida que se ejecuta el CU y a medida que se envían mensajes entre objetos.
- Un mensaje enviado por un objeto dispara la toma del control en el objeto receptor.

Diagramas de Interacción

Diagramas de interacción



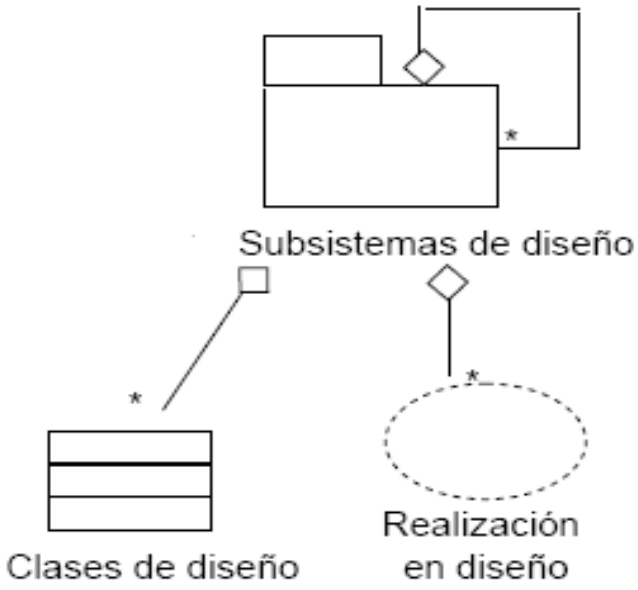
Subsistemas de Diseño



- Un subsistema es un agrupamiento de clases u otros subsistemas. Clases se agrupan en subsistemas.
- Posee un conjunto de interfaces que definen el contexto del subsistema (actores y otros subsistemas y clases).
- Los subsistemas pueden diseñarse descendente o ascendentemente.
  - Cuando se hace de manera ascendente, los desarrolladores proponen subsistemas basados en clases que ya se han identificado que empaquetan.
  - Cuando se hace en forma descendente se identifican los subsistemas de más alto nivel e interfaces antes que se hayan identificado las clases.

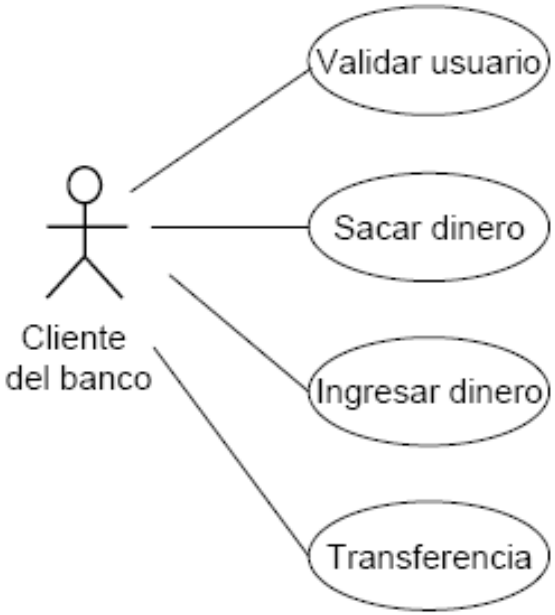
Subsistemas de diseño

- Para organizar los artefactos de diseño: clases de diseño, realización de casos de uso, interfaces y otros subsistemas.
- Fuertemente cohesionados y débilmente acoplados.

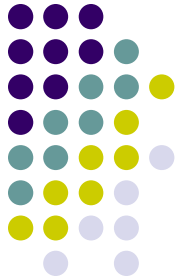


Ejemplo: Realización CU

- Para ilustrar las actividades, utilizaremos el ejemplo del cajero automático



Ejemplo: Realización CU



2.1 Artefactos.

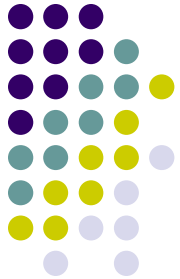
- 2.1.1 Modelo de análisis.
- 2.1.2 Clases de análisis.
- 2.1.3 Realización en análisis de los C.U.
- 2.1.4 Paquetes de análisis

2.2 Actividades.

- **2.2.1. Diseño de los casos de uso.**
- 2.2.2. Diseño de las clases.
- 2.2.3. Diseño de los paquetes.

- Identificar las clases de diseño y/o subsistemas necesarios para la realización del caso de uso.
- Distribuir el comportamiento del caso de uso entre las clases y/o subsistemas de diseño.

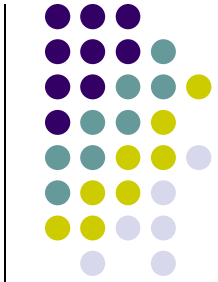
Realización CU Diseño



Identificar las clases de diseño

- Derivar las clases de diseño de las correspondientes clases de análisis que participan en el caso de uso.
- Estudiar los requisitos especiales del caso de uso: realizarlos con los mecanismos genéricos de diseño o con clases de diseño.
- Asignar responsabilidades a las clases identificadas.
- Realizar un diagrama de clases que muestre las clases de diseño que intervienen en la realización del caso de uso y las relaciones entre ellas.

Realización CU Diseño

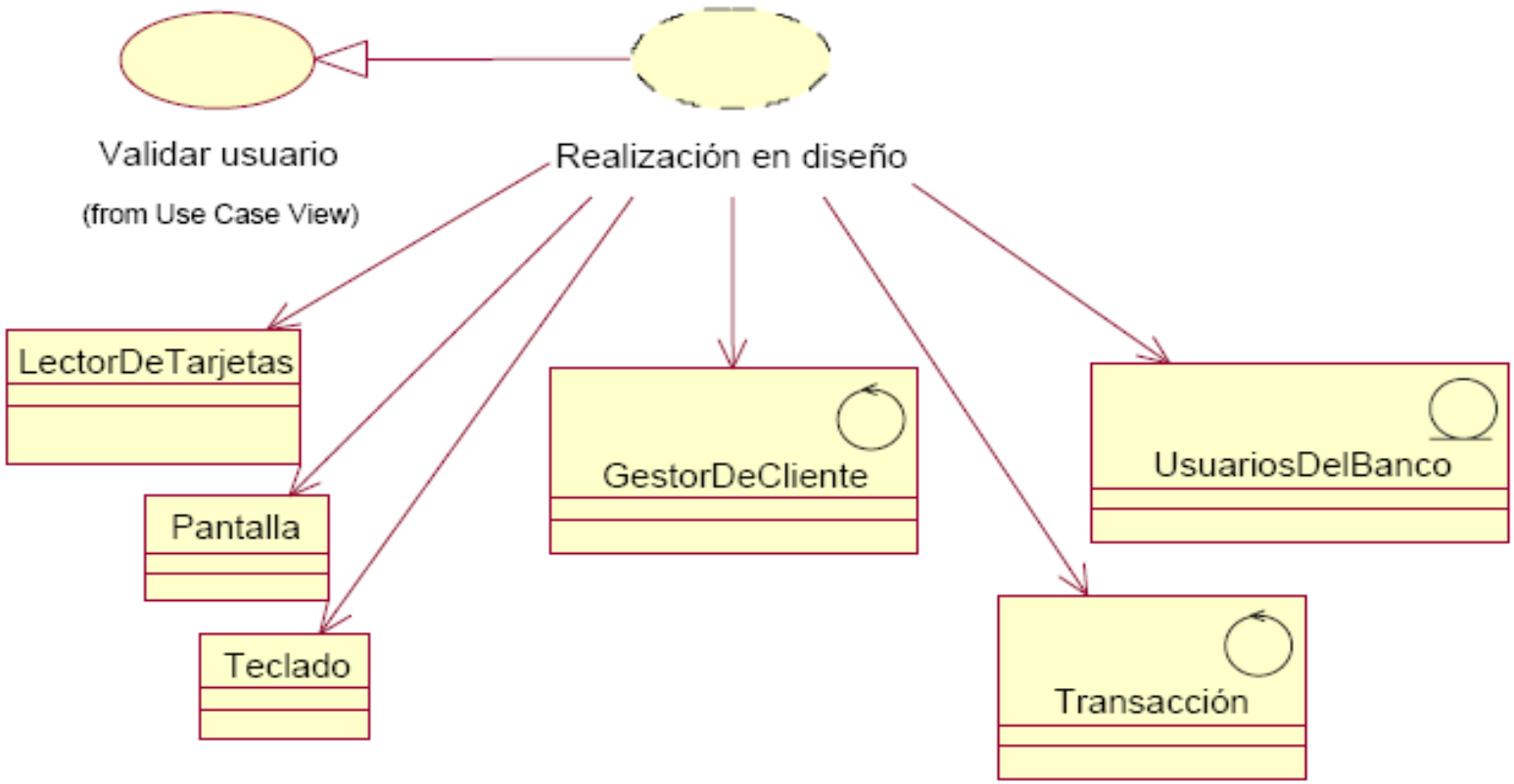


Describir interacciones entre objetos de diseño

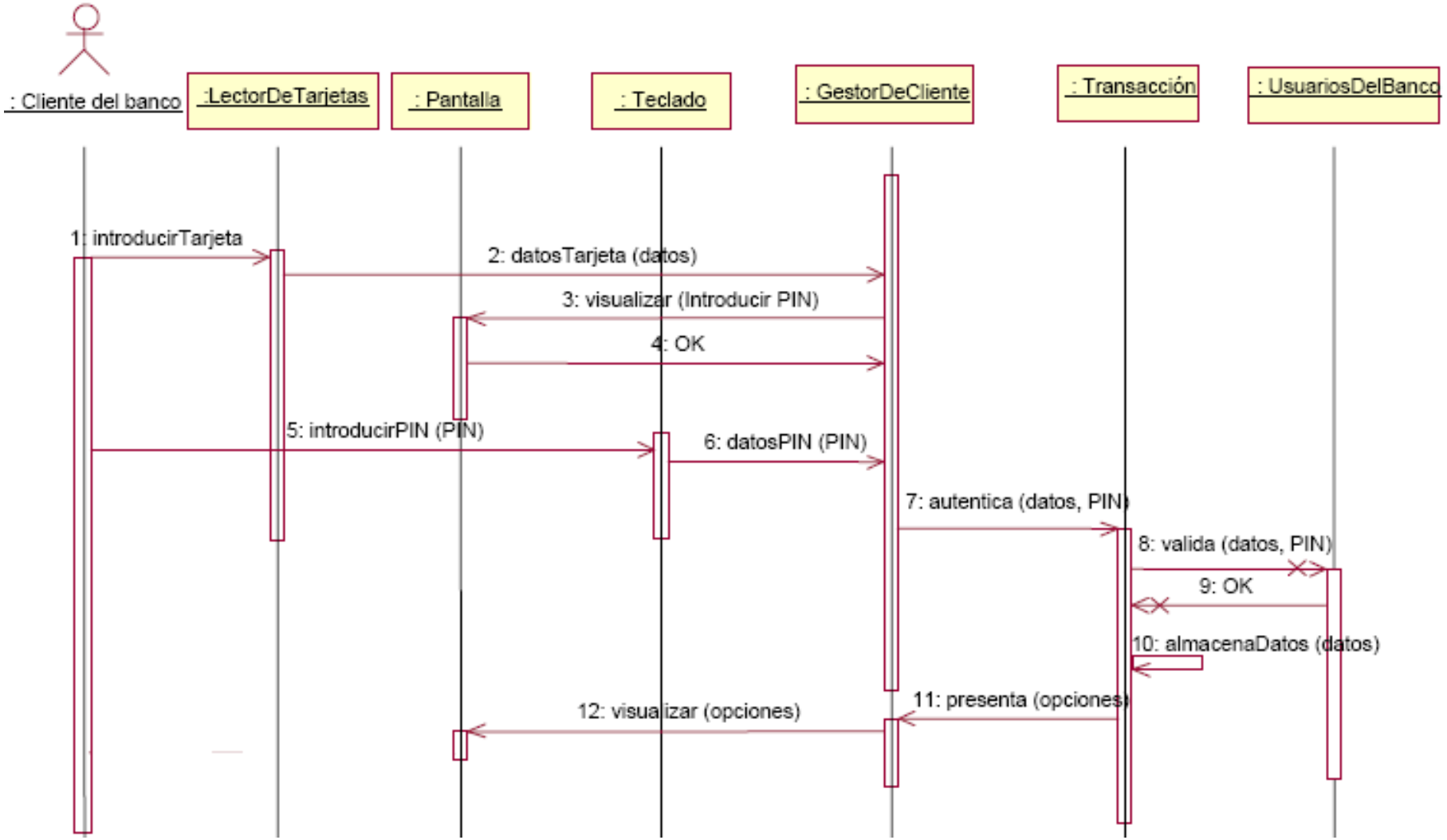
- Utilizar diagramas de secuencia
  - objetos, instancias de actores, enlaces
- Crear un diagramas de secuencia
- Comenzar estudiando la realización en análisis del c.u.
- Sobre los diagramas de secuencia:
  - el caso de uso comienza cuando una instancia de un actor envía un mensaje a un objeto interfaz.
  - cada clase de diseño identificada debería tener al menos un objeto participando en el diagrama de secuencia.
- En esta fase gestionar excepciones y errores (entradas incorrectas, situaciones anormales, etc.)

Realización CU Diseño

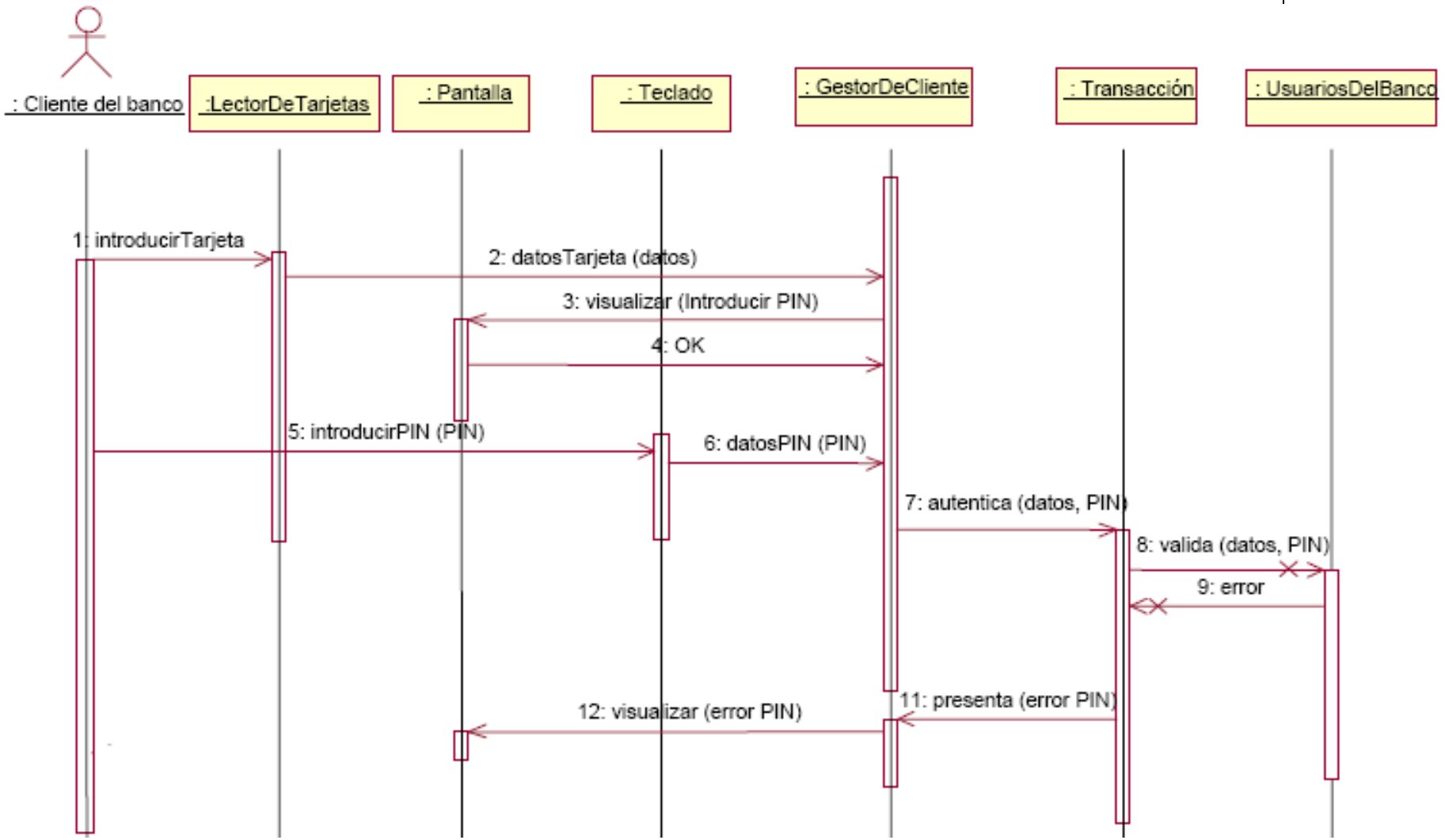
Diseño del caso de uso: “Validar usuario”



Diseño del CU: “Validar usuario”
CB- *Secuencia correcta*

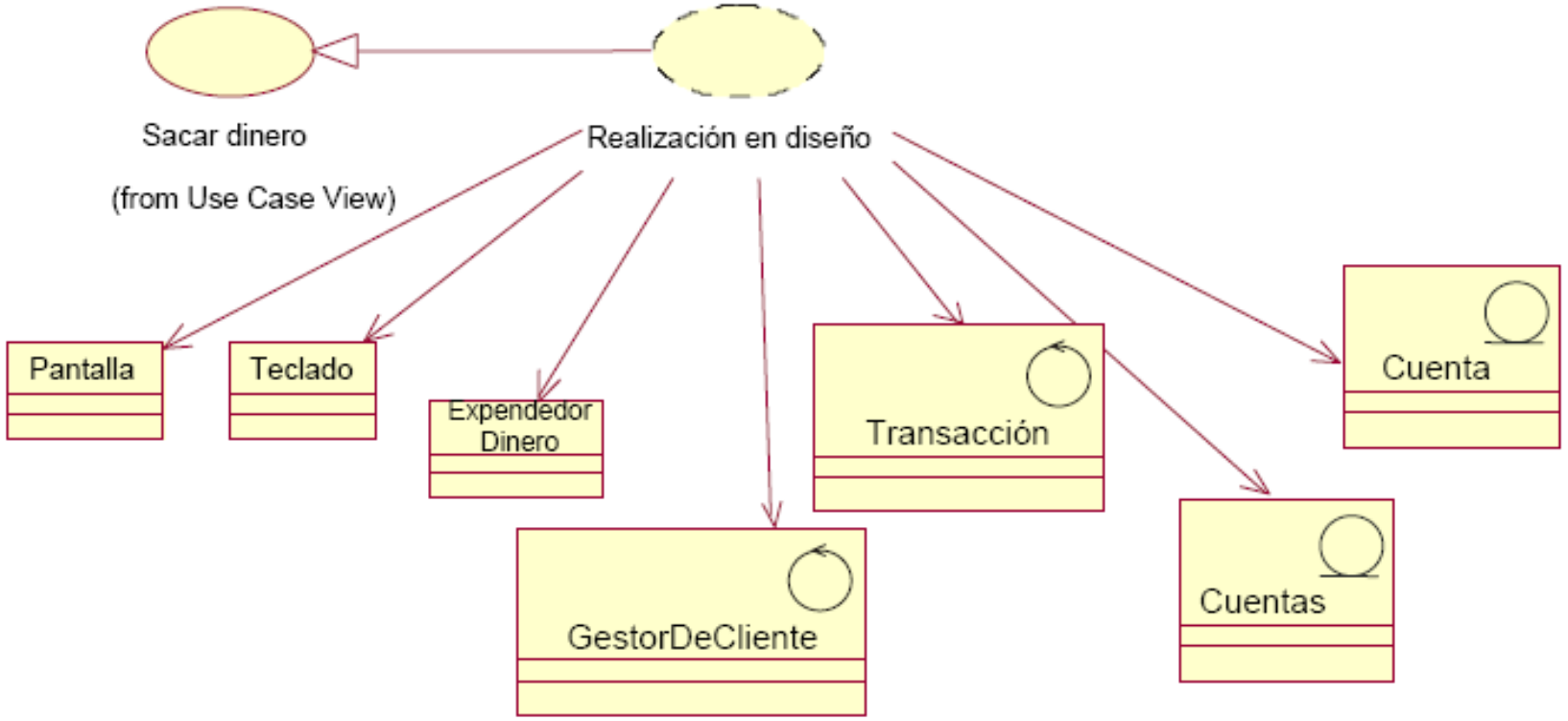


Diseño del CU: "Validar usuario"
CA- *Código Incorrecto*

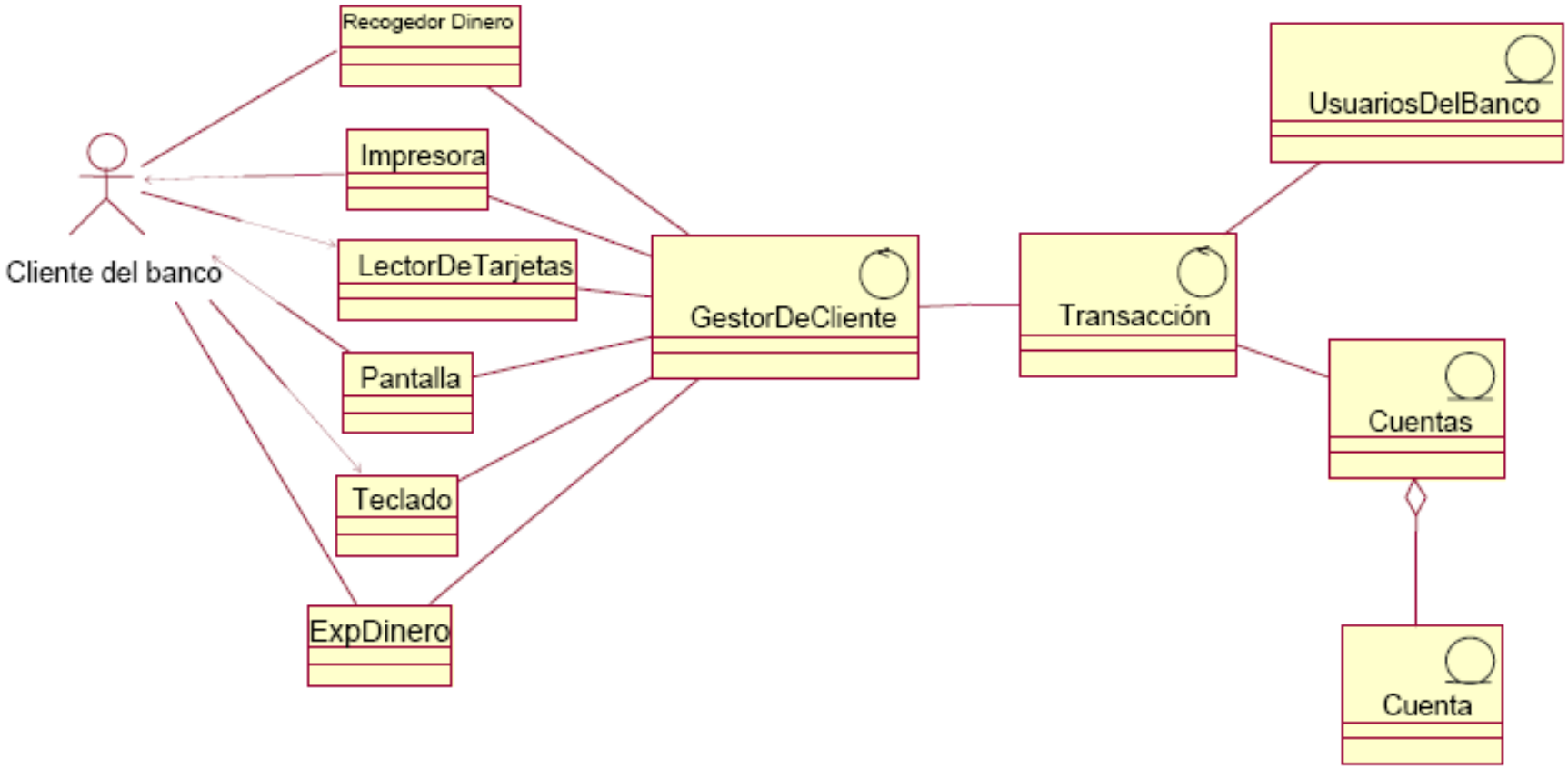


Realización CU Diseño

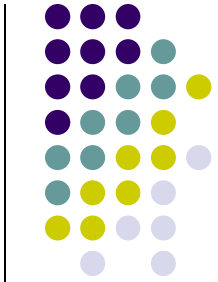
- Suponemos que el usuario ya ha sido identificado (se ha ejecutado el caso de uso anterior).
- Ahora selecciona la opción “sacar dinero”.



Modelo de clases de diseño



Diseño de las clases



2.1 Artefactos.

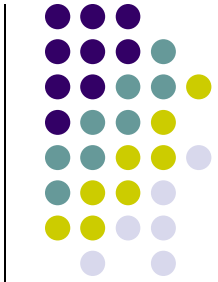
- 2.1.1 Modelo de análisis.
- 2.1.2 Clases de análisis.
- 2.1.3 Realización en análisis de los C.U.
- 2.1.4 Paquetes de análisis

2.2 Actividades.

- **2.2.1. Diseño de los casos de uso.**
- 2.2.2. Diseño de las clases.
- 2.2.3. Diseño de los paquetes.

- Identificar las responsabilidades de las clases de diseño (papeles en los casos de uso)
- Identificar:
  - operaciones
  - atributos
  - relaciones en las que participa
  - estados (diagramas de estados)
  - métodos que soportan sus operaciones
  - Requisitos nuevos…

Diseño de las clases



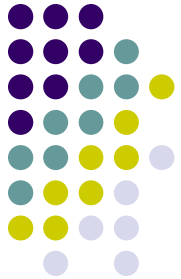
1 Identificar operaciones

- En el lenguaje de implementación
- Mirar responsabilidades que tiene en los casos de uso

2 Identificar atributos

- Describirlos en el lenguaje de programación
- Considerar los atributos de las clases de análisis de las que se derivan

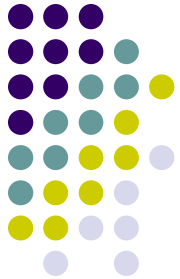
Diseño de las clases



3 Identificar asociaciones y agregaciones

- Las interacciones en los diagramas de secuencia precisan de asociaciones entre las clases que interactuan.
- Minimizar el número de relaciones entre clases (disminuir el acoplamiento).
- Refinar multiplicidad, papeles, etc.
- Refinar la navegabilidad (dirección) de las asociaciones en base a los diagramas de secuencia.
- Identificar generalizaciones-especializaciones.

Diseño de las clases



4 Describir métodos

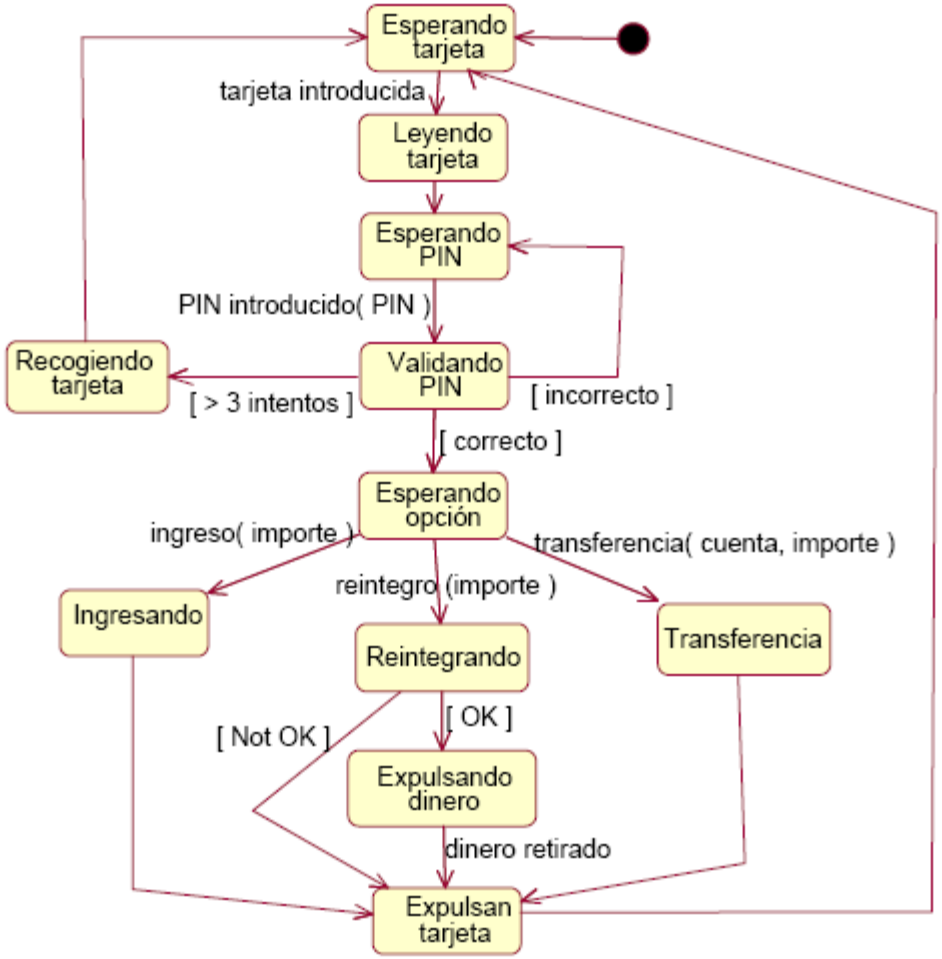
- Algoritmos para implementar alguna operación (lenguaje natural).
- Esqueletos de métodos generado por la herramienta.
- En general, esto **se suele hacer en implementación.**

5 Describir estados

- Algunos objetos reaccionan en función de su estado actual. Utilizar diagramas de transición de estados.

DTE

Diseño de la clase “GestorDeCliente”



La única clase que tiene un comportamiento parecido a una máquina de estados es **GestorDeCliente**.

Realizar el diagrama de transición de estados del sistema Cajero Automático.

Realización CU Diseño

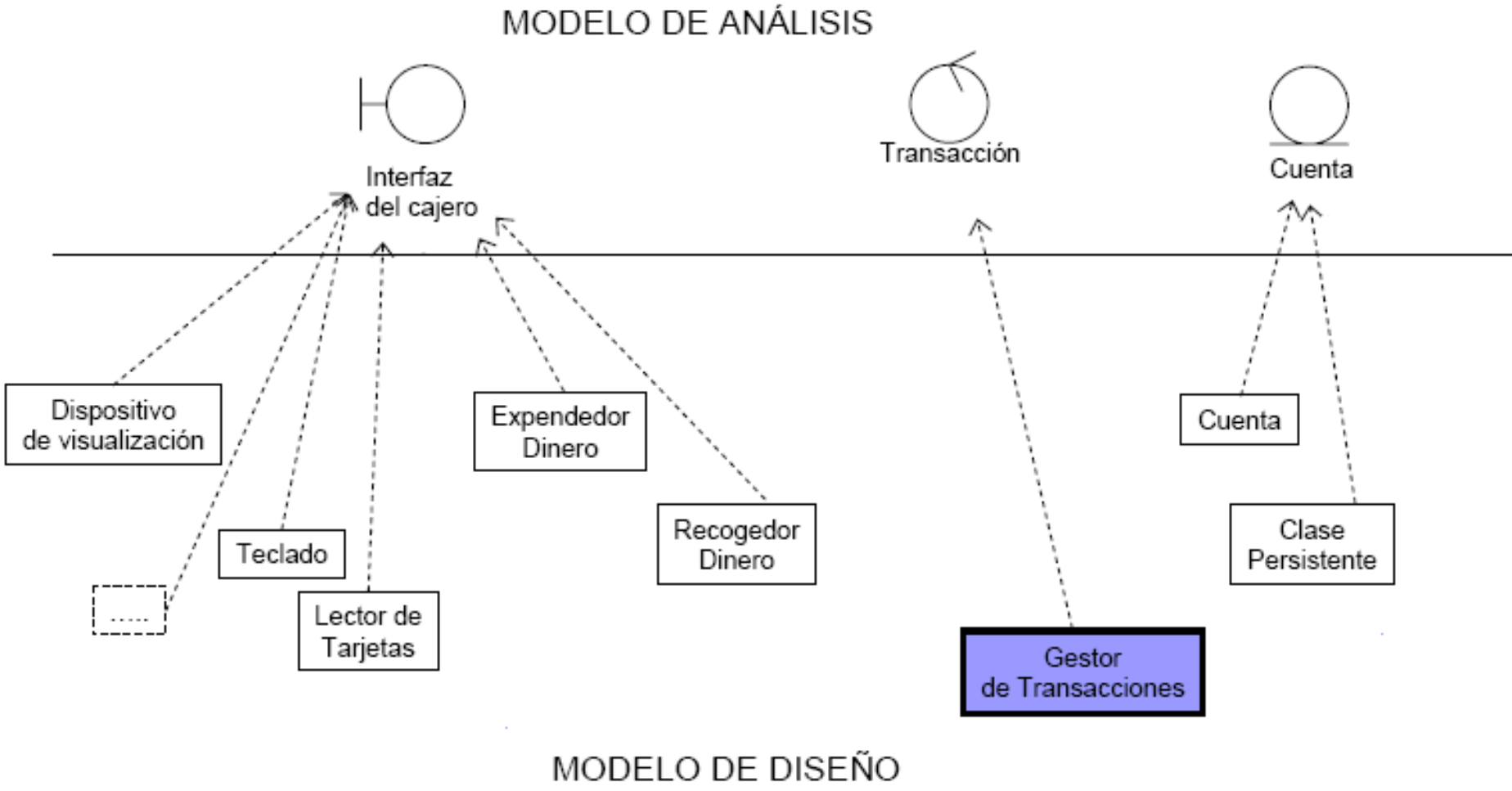
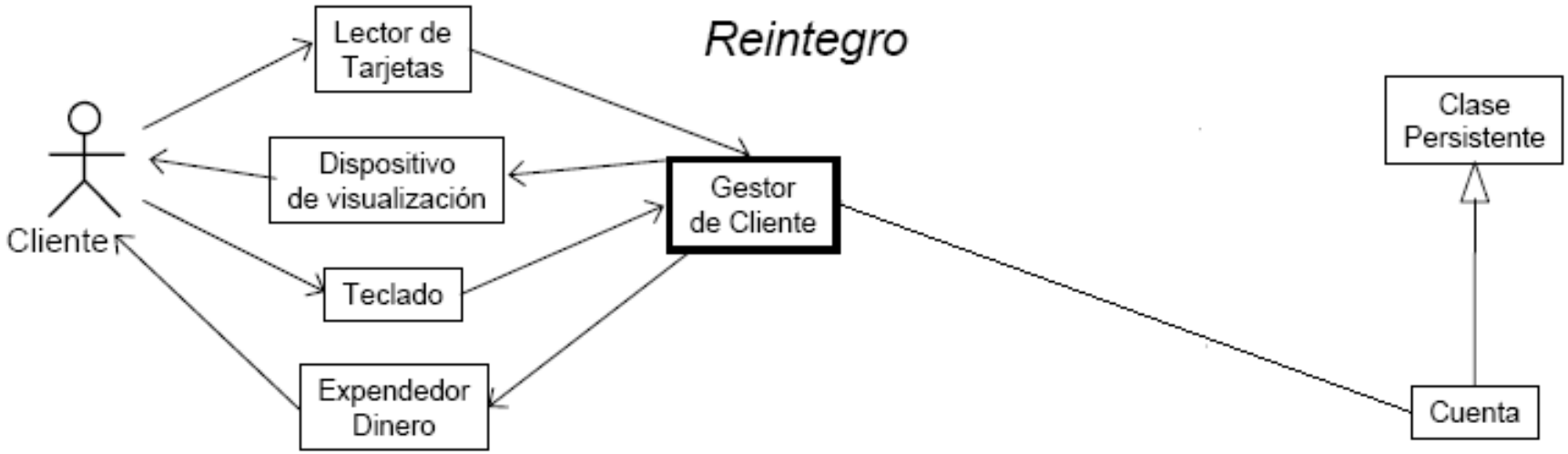


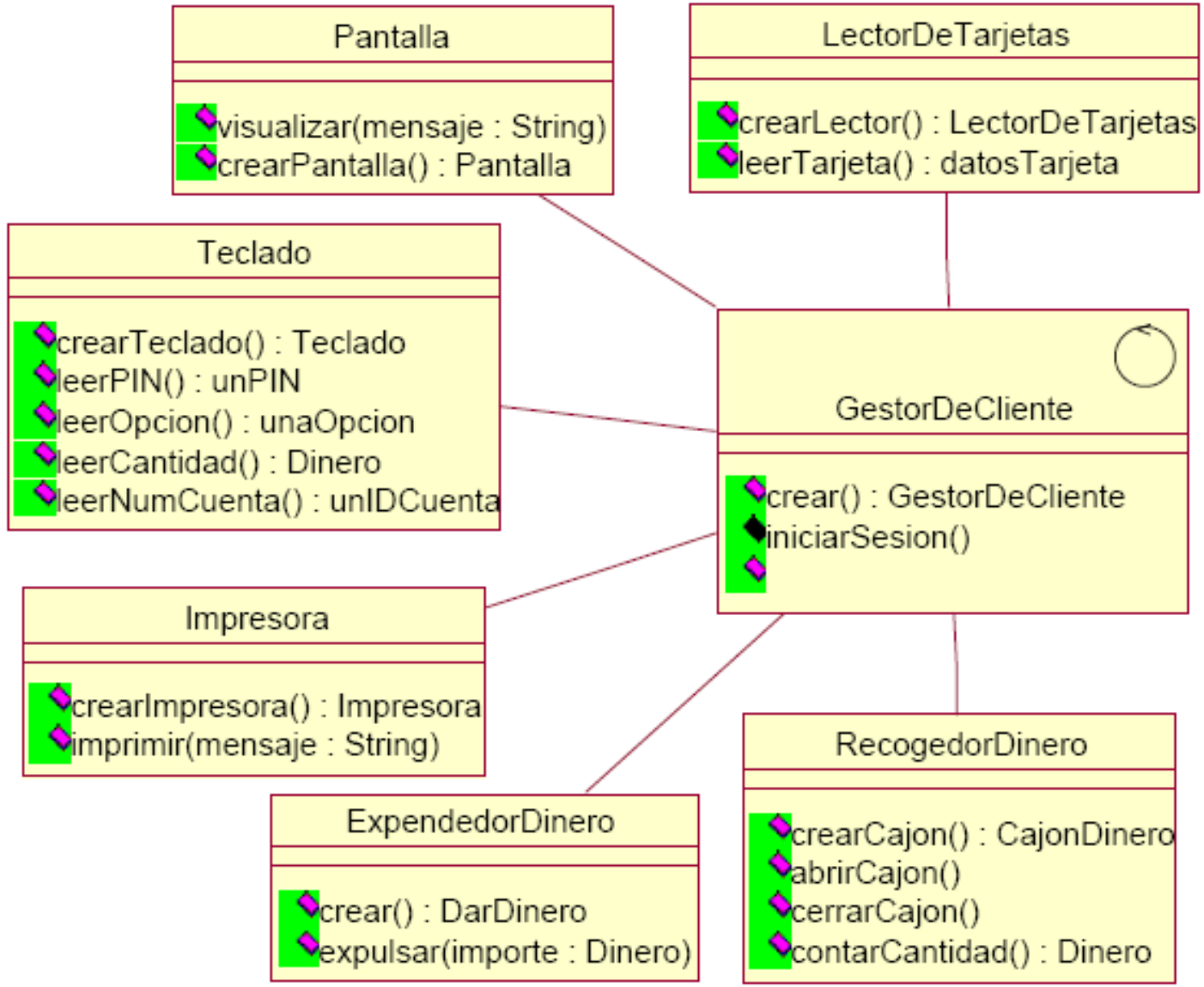
Diagrama de Clase Diseño

Diagramas de clase

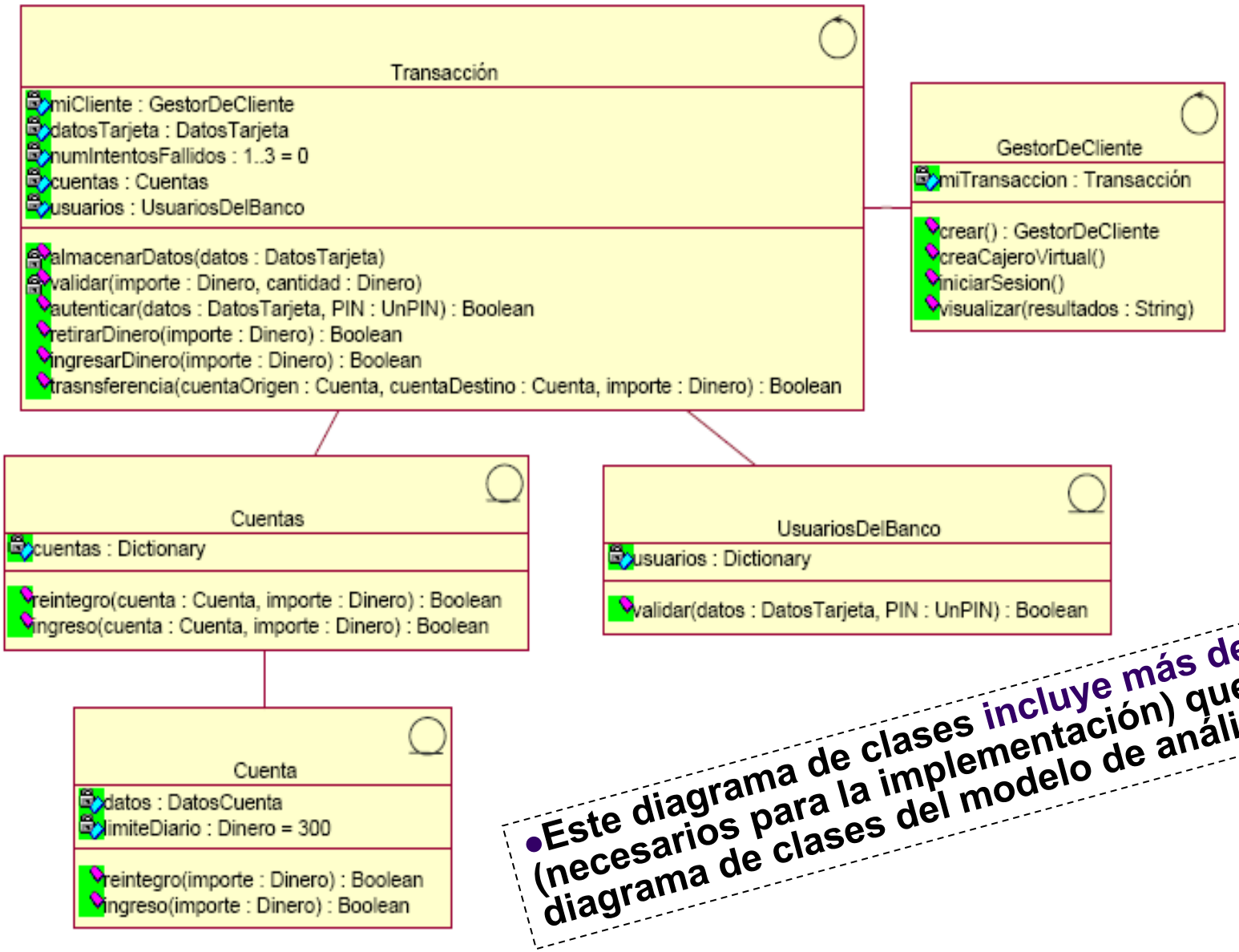
- Una clase de diseño puede participar en varios casos de uso
- Algunas responsabilidades, atributos y asociaciones suelen ser específicos de un sólo caso de uso.



Modelo de clases de diseño



Modelo de clases de diseño



- **Este diagrama de clases incluye más detalles (necesarios para la implementación) que el diagrama de clases del modelo de análisis.**