

# Fundamentos de la Programación Orientada a Objetos



# Conceptos básicos - Abstracción

- ¿De qué hablamos cuando hablamos de abstracción?
  - Es el proceso de supresión de detalles respecto de un fenómeno, entidad o concepto.
  - El objetivo es concentrarse en los aspectos más significativos.
- Algunas definiciones
  - *Un concepto general formado a partir de la extracción de características comunes tomadas de ejemplos específicos.*
  - *El proceso mental donde las ideas son separadas de los objetos concretos.*

## Conceptos básicos – Abstracción y O.O.

- La Programación Orientada a Objetos nos permite independizarnos del modelo de cómputo subyacente.
- Luego, podemos concentrarnos en resolver nuestro problema.
- En la POO la abstracción se plantea en términos de similitudes entre fenómenos, conceptos, entidades, etc.
- De esta manera, logramos identificar conceptos generales (persona, auto, pelota, etc.) que puedan ser traducidos a construcciones básicas (objetos) en nuestro paradigma.

## Conceptos básicos - Modelo

- ¿Qué es un modelo?
  - Es una versión simplificada de algún fenómeno o entidad del mundo real.
- ¿Qué significa modelar?
  - Es un proceso de abstracción.
  - Tomamos una versión reducida de lo que queremos representar. Sólo especificamos aquellas cosas que son relevantes.

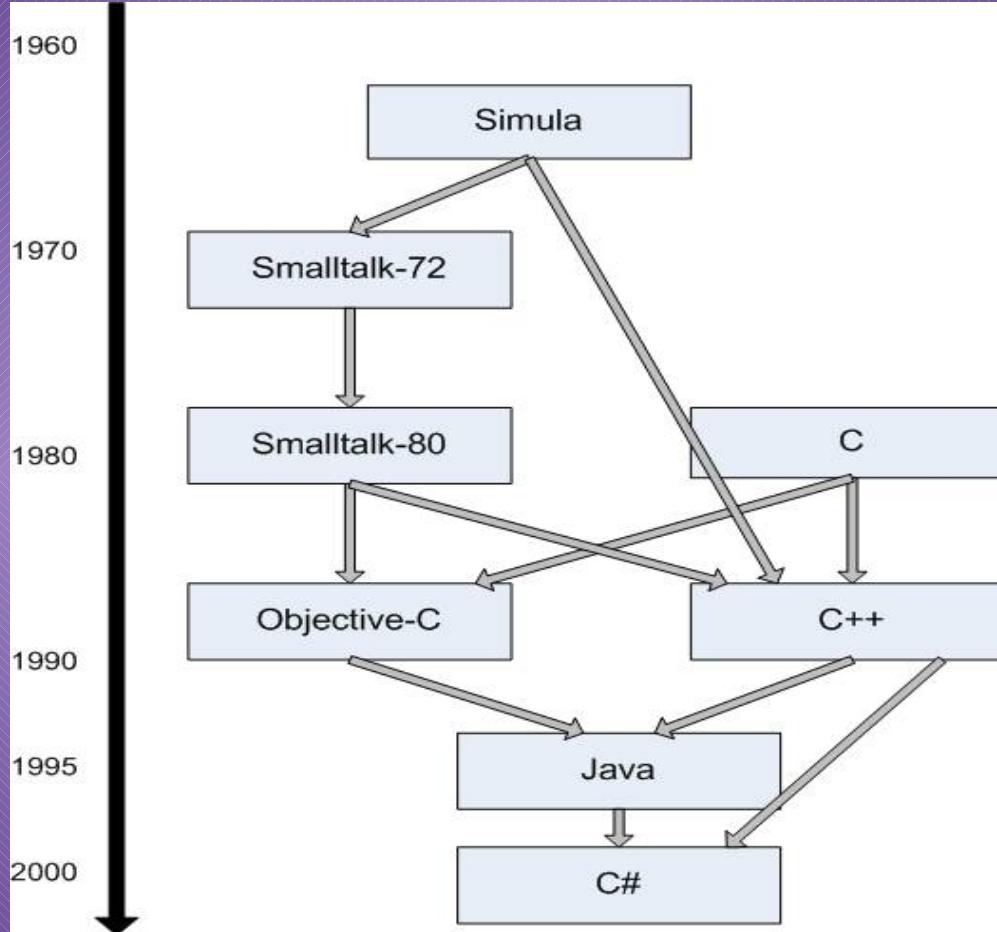
## **Conceptos básicos – Modelo O.O.**

- Los conceptos del dominio se representan como objetos.
- Los objetos se componen y colaboran con otros objetos para formar un modelo.
- La ejecución de un programa OO puede verse como un modelo simulando el comportamiento de una parte del Mundo.

# Reseña Histórica

- Años 60: Simula. (Norwegian Computing System)
  - Inspirado por los problemas involucrados en la simulación de sistemas de la vida real.
- Años 70: Smalltalk (Xerox PARC)
  - Alan Kay y su equipo de investigación desarrollan Smalltalk basándose en los conceptos que introducía Simula.
- Años 80: Smalltalk 80 y C++ (Laboratorios Bell)
  - Casi contemporáneo al trabajo de Kay desde los laboratorios Bell desarrollan C++ como extensión del lenguaje C. Comienzan proyectos similares, como Objective-C, Eiffel, Actor, entre otros.
- Años 90: Java (Sun Microsystems)
  - Es el resultado de la búsqueda de una plataforma independiente del hardware adoptando los conceptos de OO.
- 2002: C# y .NET (Microsoft)
  - Microsoft lanza su plataforma .NET para el desarrollo de aplicaciones multiplataforma. Junto con la plataforma aparece el lenguaje C#.

# Reseña histórica - Línea temporal

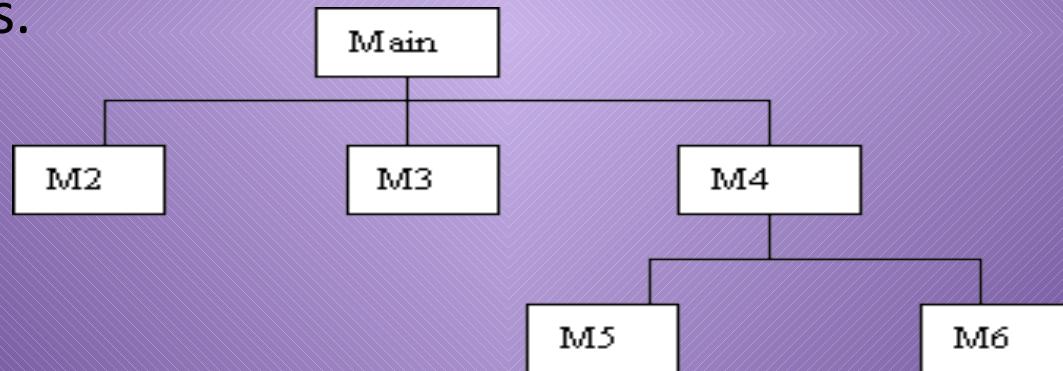


# Paradigma de Programación

- Conjunto de
  - Elementos.
  - Reglas.
- Brinda un marco para el diseño y la construcción de Programas.
- Ejemplos:
  - Estructurado.
  - Funcional.
  - Lógico.
  - Orientado a Objetos.
- ¿Qué es un *programa* en un determinado paradigma?

# Programación Estructurada

- Los sistemas contienen datos y algoritmos.
- Los programas manipulan los datos.
- Los programas están organizados por:
  - Descomposición funcional.
  - Flujo de Datos.
  - Módulos.



- Asignación, secuencia, iteración, condicionales

# Programación Orientada a Objetos

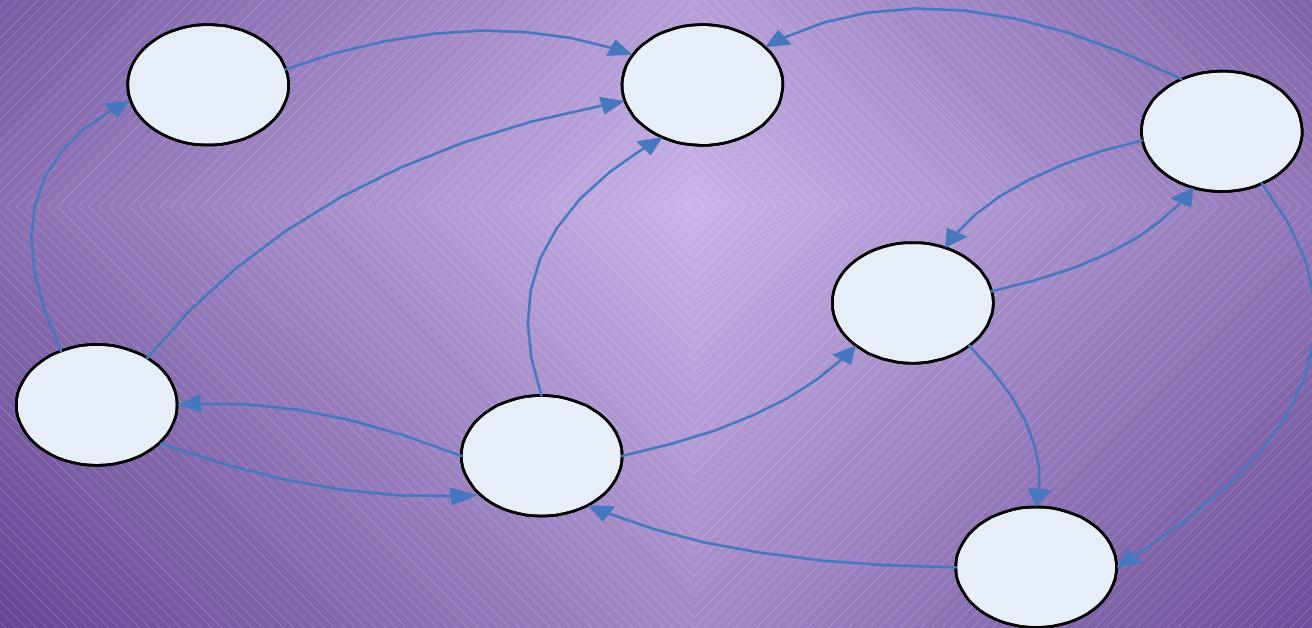
- Los sistemas están compuestos por un conjunto de **objetos**.
- Los objetos son **responsables** de llevar a cabo ciertas acciones.
- Los objetos **colaboran** para llevar a cabo sus responsabilidades.
- Principios de la programación orientada a objetos según Alan Kay, el creador de Smalltalk:
  1. Todo es un objeto
  2. Los objetos se comunican enviando y recibiendo **mensajes**
  3. Los objetos tienen su propia memoria (en términos de objetos)

## Programación Orientada a Objetos (cont.)

- La principal construcción es la noción de objetos.
- Los objetos pueden componerse o conocer otros objetos.
- Los programas están organizados en base a clases y jerarquías de herencia.
- La forma de pedirle a un **objeto** que lleve a cabo una determinada tarea es por medio del envío de un **mensaje**.

# Programa Orientado a Objetos

Un conjunto de *objetos* que *colaboran* enviándose *mensajes*



# **Elementos Básicos de la Programación Orientada a Objetos**

# Objetos

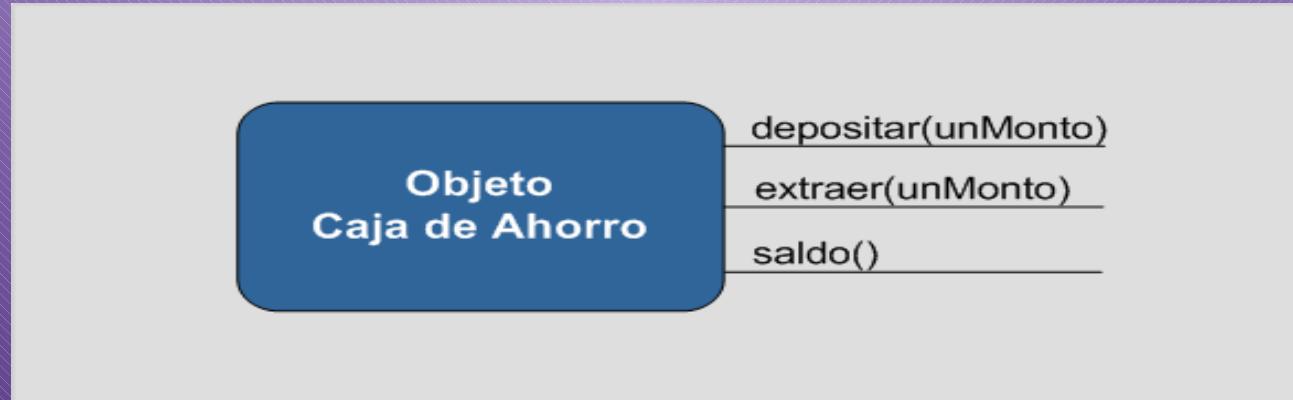
- ¿Qué es un *objeto*?
- Los objetos son los elementos primarios que utilizamos para construir programas.
- Todo es un objeto en la programación orientada a objetos.
- Un objeto es una *abstracción* de una *entidad* del *dominio del problema*.

# Características de los Objetos

- Un objeto tiene:
  - Un *comportamiento* bien determinado.
    - ¿Qué hace el objeto y cómo lo hace?
  - Un *estado* interno o *estructura interna*.
    - El conjunto de variables de instancia.
  - Una *identidad*.
    - ¿Cómo podemos distinguir un objeto de otro?
- Además:
  - Todo objeto es instancia de una *clase*.

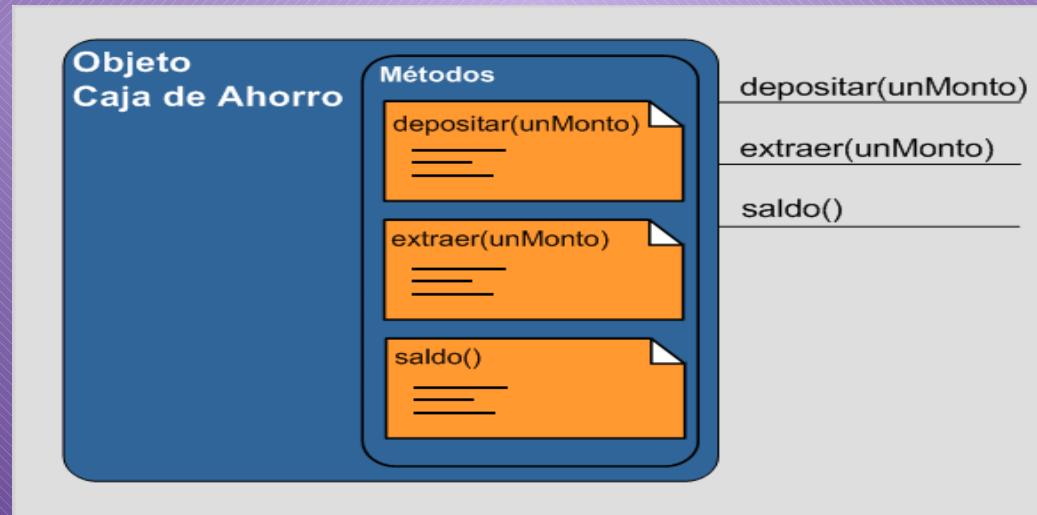
# El Comportamiento – ¿Qué hace el objeto?

- Un objeto se define en términos de su comportamiento.
- El comportamiento indica qué sabe hacer el objeto. Cuáles son sus *responsabilidades*.
- Se especifica a través del conjunto de *mensajes* que el objeto sabe responder: *protocolo*.
- Ejemplo:



# El Comportamiento – ¿Cómo lo hace?

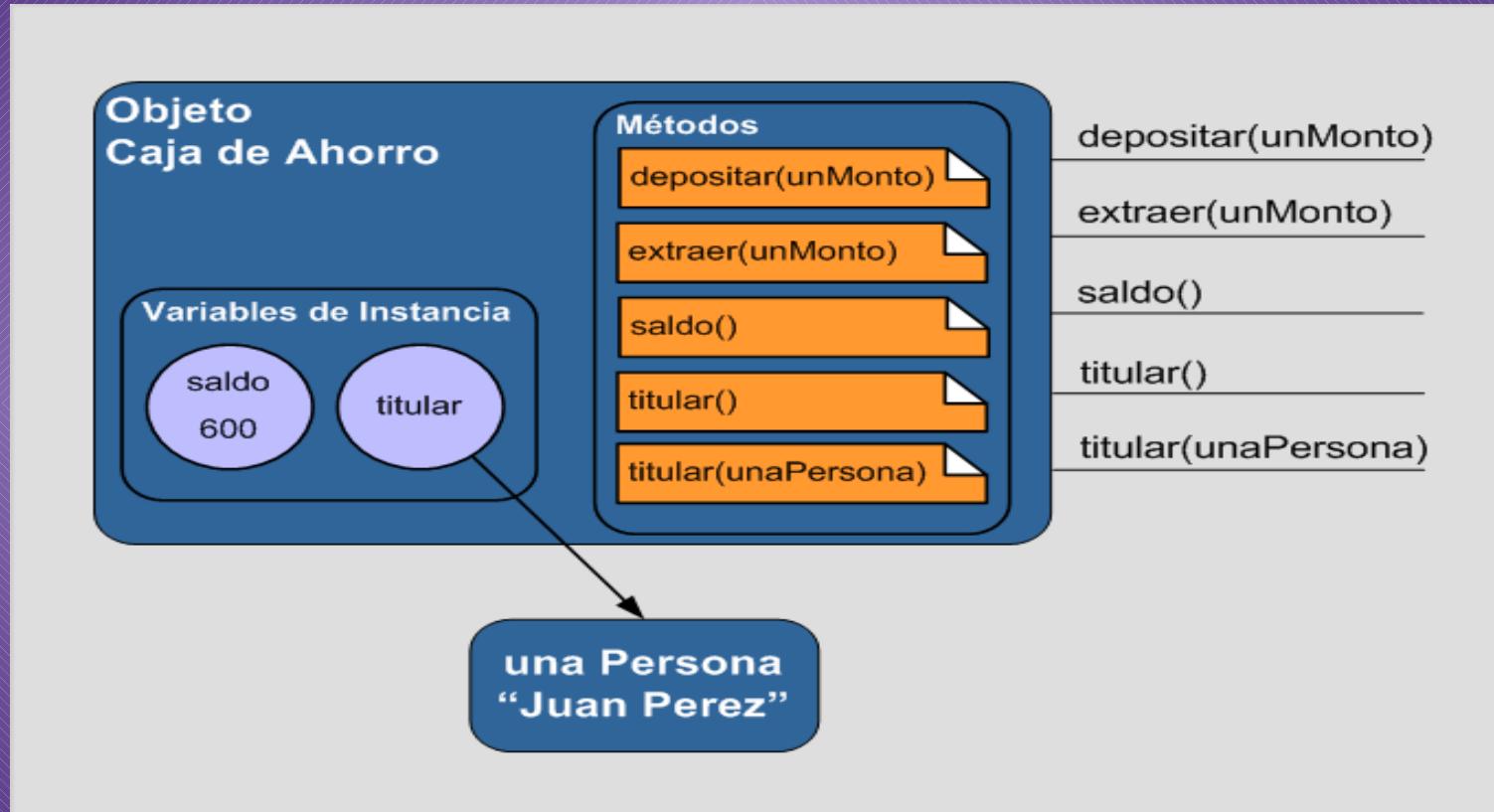
- La *implementación* indica *cómo* hace el objeto para responder a sus mensajes.
- Se especifica a través de un conjunto de *métodos*.
- Es *privada* del objeto. Ningún otro objeto puede accederla.



## El estado interno

- Está compuesto por las *variables de instancia* del objeto.
- Las variables de instancia pueden hacer referencia a:
  - Propiedades intrínsecas del objeto (un número, un valor booleano, etc).
  - Otros objetos con los cuales pueda colaborar para llevar a cabo sus responsabilidades.
- Es **privado** del objeto. Ningún otro objeto puede accederlo.

# Ejemplo Caja de Ahorro



## Practica: Identificar objetos y propiedades

- Agrupe los elementos de la siguiente lista en objetos y propiedades.
- Banco,
- Extraer,
- Apellido,
- Depositar,
- DNI,
- CuentaBancaria,
- Cliente,
- CrearCuenta,
- Saldo,
- CerrarCuenta,
- Nombre.

## Practica: Identificar objetos y propiedades

- Identifique, en el siguiente enunciado, objetos y propiedades:
- Una computadora está formada por tres partes principales, un monitor, un gabinete y un teclado. Cada una de estas partes posee un modelo y un número de serie. Además el gabinete puede determinar su temperatura y la cantidad de puertos USB que posee (tanto disponibles como en uso).

## Envío de un mensaje

- El comportamiento de un objeto está definido en términos de los mensajes que éste entiende.
- Para poder enviarle un mensaje a un objeto, hay que conocerlo.
- Al enviarle un mensaje a un objeto, éste responde activando el método asociado a ese mensaje (siempre y cuando exista).
- Como resultado del envío de un mensaje puede retornarse un objeto.

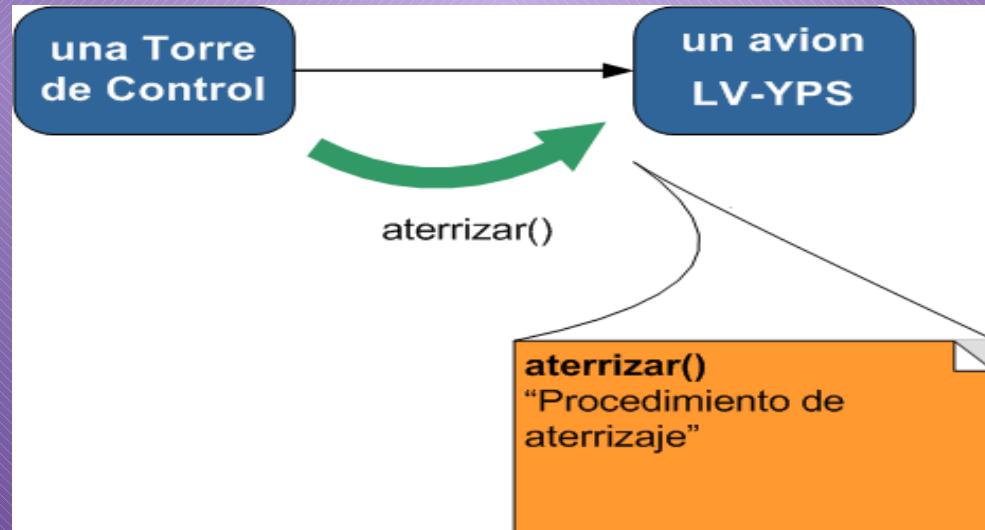
## Envío de un mensaje - ejemplo

- Contamos con un objeto que representa la torre de control de un aeropuerto.
- La torre desea avisar al avión *LV-YPS* que debe aterrizar.
- Primero debe conocerlo.



## Envío de un mensaje - ejemplo

- La torre envía el mensaje 'aterrizar' al avión.
- El avión activará un método con los pasos necesarios para aterrizar, si el avión no tiene tal método se producirá un error.



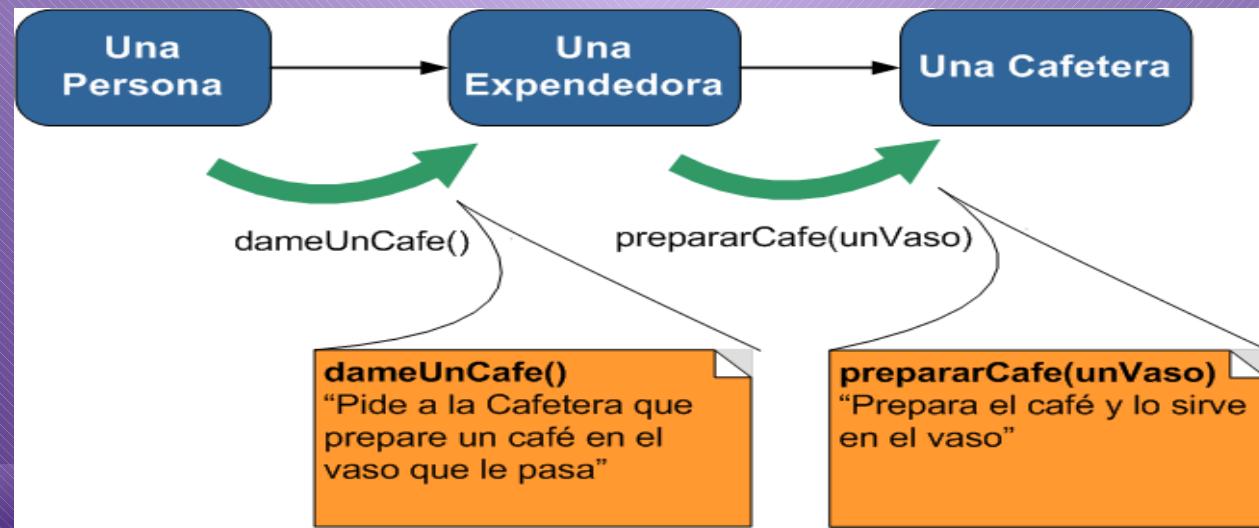
## **Envío de un mensaje - ejemplo**

- Se tiene modelados una persona, una expendedora y una cafetera.
- La persona le pide un café a la expendedora. Ésta delega la preparación a la cafetera. Luego, la cafetera retorna el café a la expendedora que se lo entrega a la persona.

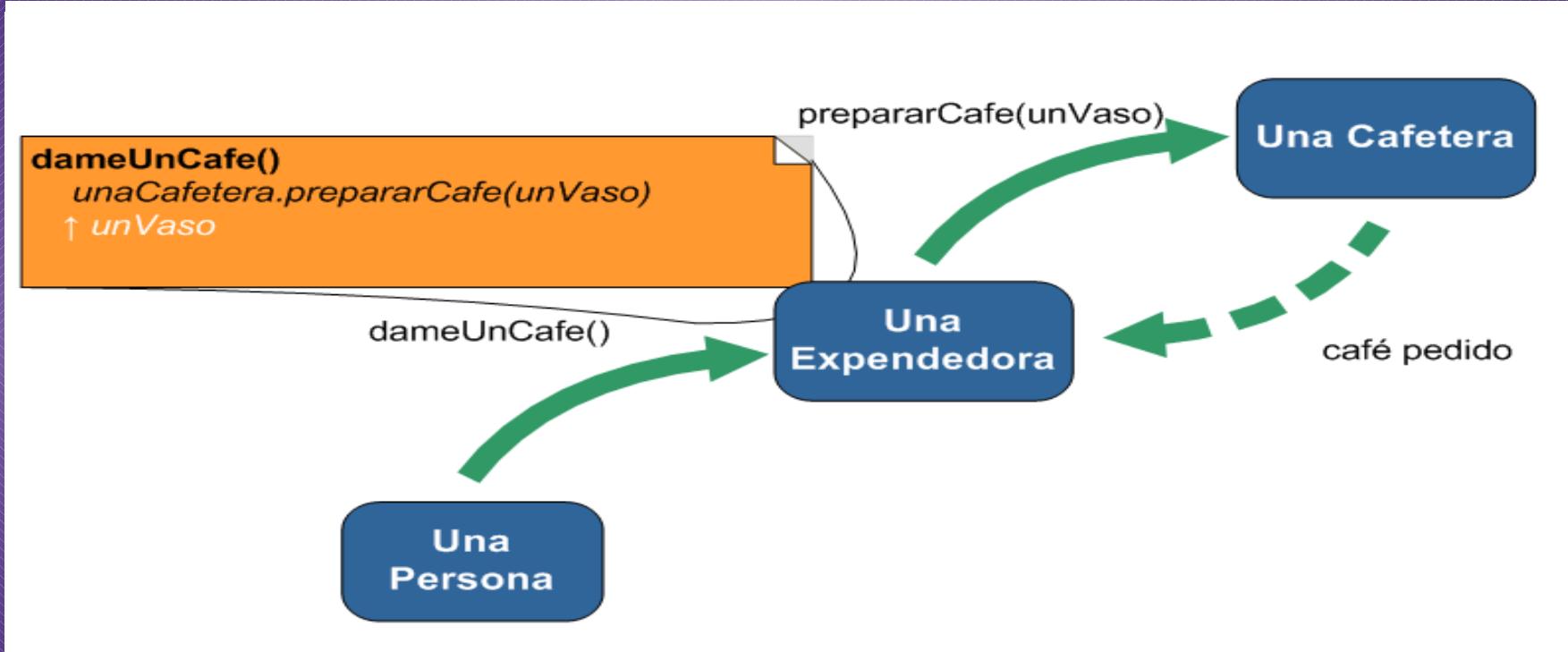


## Envío de un mensaje - ejemplo

- La persona envía el mensaje `#dameUnCafe()` a la expendedora.
- La expendedora le pide a la cafetera que le sirva el café en el vaso que le da.
- La cafetera retorna el vaso con el café preparado.
- La expendedora entrega el vaso con café a la persona



# Envío de un mensaje



# Especificación de un Mensaje

- ¿Cómo se especifica un mensaje?
  - Se especifica con el *nombre* correspondiente al protocolo del objeto receptor.
  - Se indica cuáles son los *parámetros*, es decir, la información necesaria para resolver el mensaje.
- Cada lenguaje de programación propone una sintaxis particular para indicar el envío de un mensaje.
- A lo largo del curso utilizaremos la siguiente sintaxis:

```
<objeto receptor>.<nombre de mensaje> (<parámetros>) ;
```

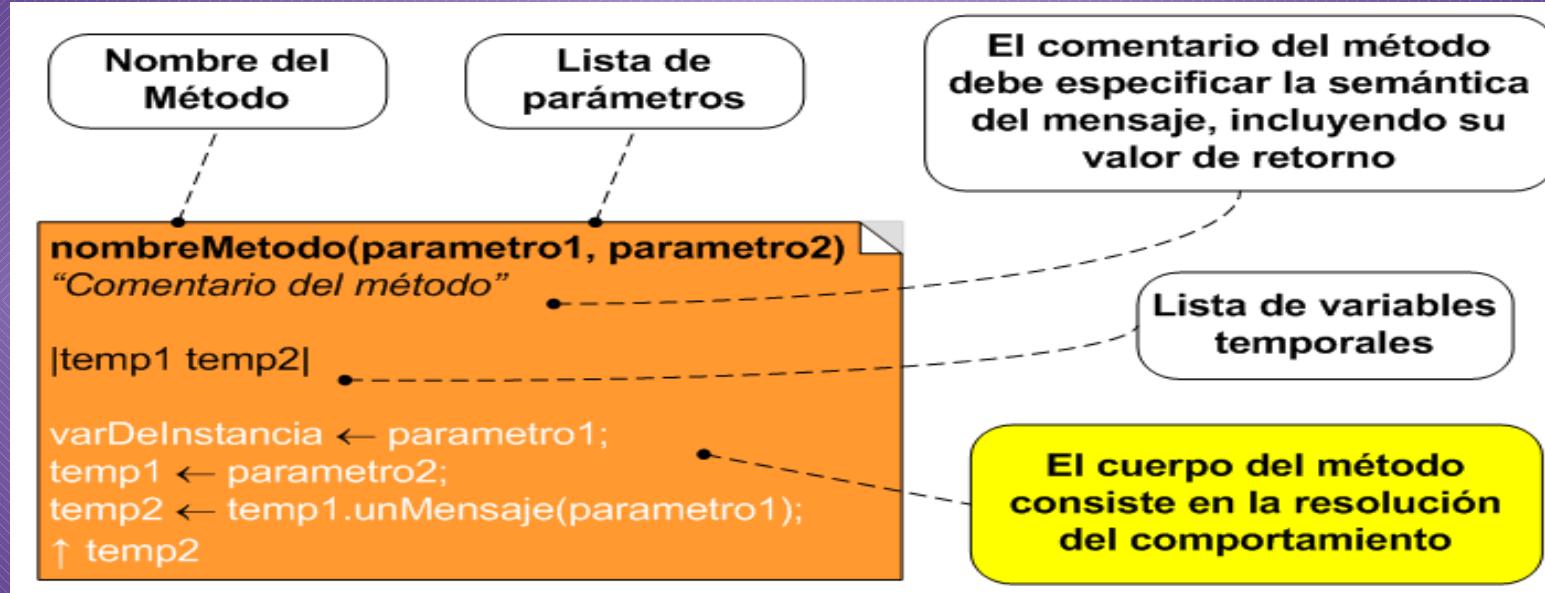
- Ejemplo empleando la sintaxis propuesta
  - Decirle a una cuenta bancaria que deposite \$100 se escribe como:

```
unaCuenta.depositar(100) ;
```

# Métodos

- ¿Qué es un método?
  - Es la contraparte funcional del mensaje.
  - Expresa la forma de llevar a cabo la semántica propia de un mensaje particular (el *cómo*).
- Un método puede realizar básicamente 3 cosas:
  - Modificar el estado interno del objeto.
  - Colaborar con otros objetos (enviándoles mensajes).
  - Retornar y terminar.

# Especificación de un Método



# Ejemplo – Depositar en Cuenta Bancaria

**depositar(unMonto)**

*“Agrega unMonto al saldo actual de la cuenta”*

saldo ← saldo + unMonto

# Ejemplo – Cajero

**realizarDeposito(unMonto, nroCuenta)**

*“Deposita unMonto en la cuenta numero nroCuenta”*

|cuenta|

```
cuenta ← banco.buscarCuenta(nroCuenta);  
cuenta.despositar(unMonto)
```

## Practica : Saldo

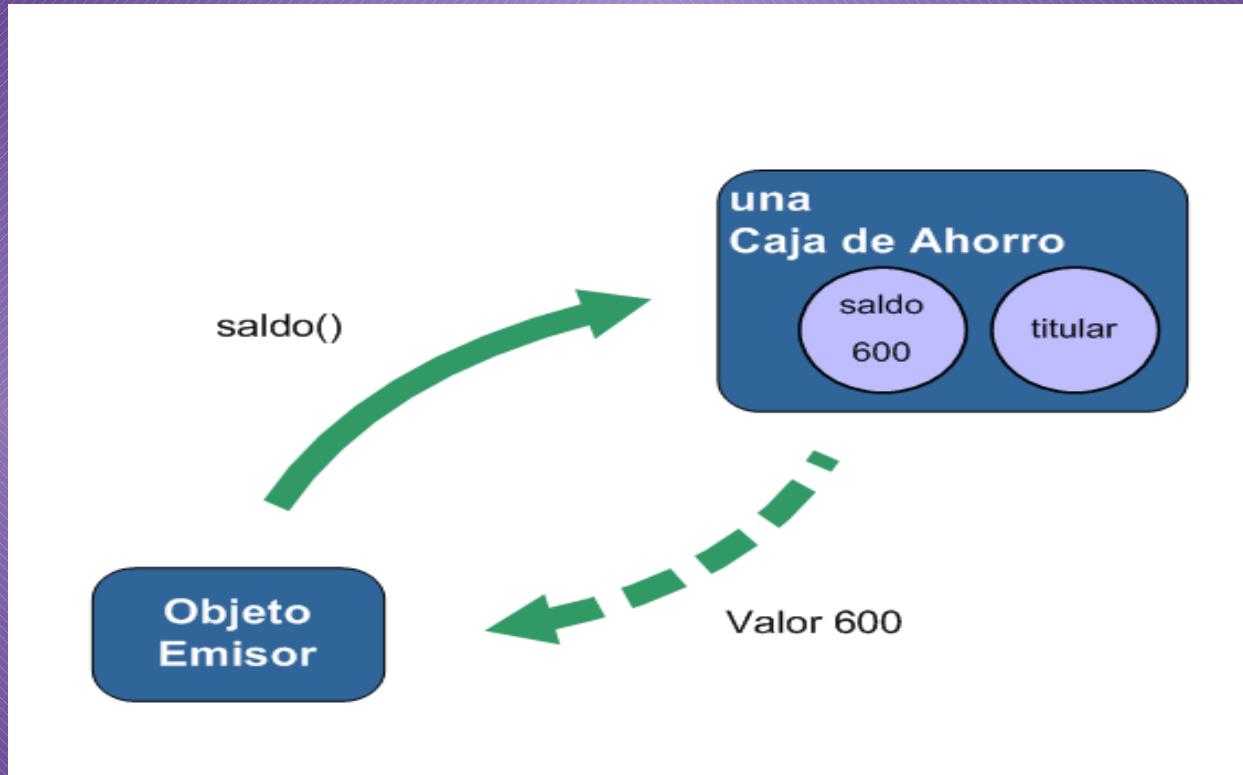
- ¿Cómo sería el método del mensaje `saldo`?
- Recordar el ↑ (return)

`saldo()`

*“Retorna el saldo de la cuenta”*

↑ saldo

# Envío del mensaje saldo



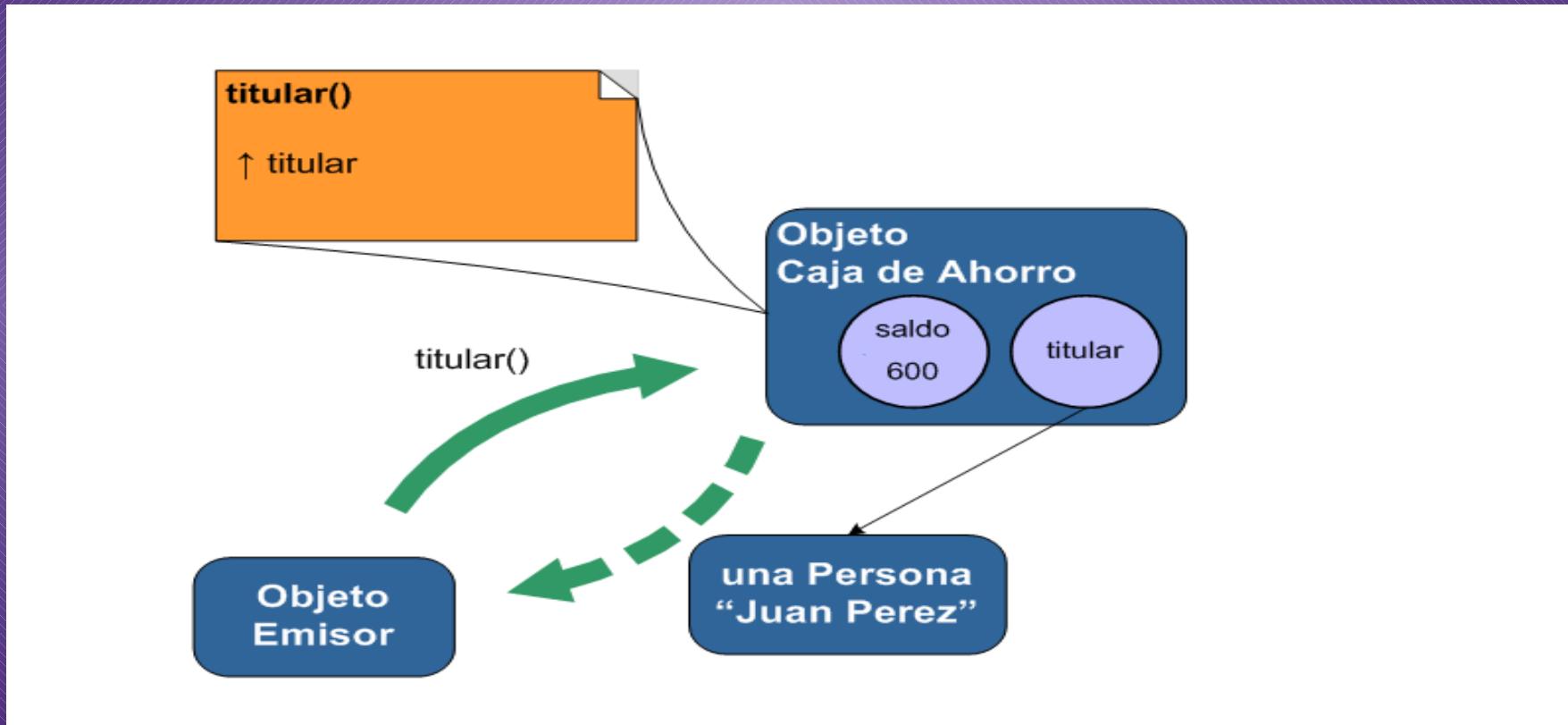
# Cambio del estado interno



una Persona  
“Luis García”

# Ejemplo – Retornar el titular

- ¿Cómo sería el método del mensaje `titular()`?



# Ejercicio – Depósito y Extracción

- ¿Cómo escribimos el método **extraer(unMonto)** ?



**depositar(unMonto)**

*“Agrega unMonto al saldo actual de la cuenta”*

saldo  $\leftarrow$  saldo + unMonto

# Ejercicio – Identificar objetos y mensajes

cuenta.extraer(unMonto)

cuenta.titular()

cajero.obtenerCuenta(numeroDeCuenta)

(cajero.obtenerCuenta(210321)).saldo()

banco.suspender(cuenta.titular())

banco.suspender(banco.obtenerCuenta(211101).titular())

## Practica: Cuenta Bancaria

- Modelar el proceso de extracción de dinero de un cajero automático.
- El cliente inserta la tarjeta e ingresa su PIN.
- Luego selecciona la opción extracción e ingresa el monto que desea extraer.
- Con esta información, el cajero le pide al banco que se ocupe de realizar la operación.

## Practica: Cuenta Bancaria

- El banco busca la cuenta con el número ingresado. Para esto el banco busca la cuenta del cliente, y le pide a la cuenta que realice la operación (actualiza el saldo de la cuenta).
- El banco entiende el mensaje `#buscarCuenta`: que dado un número de cuenta retorna la cuenta que corresponde a ese número.

## Comportamiento y estado interno

- Dijimos que el comportamiento tiene una implementación privada.
- También que el estado interno es privado.
- ¿De qué hablamos cuando hablamos de privacidad?

# Encapsulamiento

*“Es la cualidad de los objetos de ocultar los detalles de implementación y su estado interno del mundo exterior”*

- Características:
  - Esconde detalles de implementación.
  - Protege el estado interno de los objetos.
  - Un objeto sólo muestra su “cara visible” por medio de su protocolo.
  - Los métodos y su estado quedan escondidos para cualquier otro objeto. Es el objeto quien decide *qué* se publica.
  - Facilita modularidad y reutilización.

## Encapsulamiento – importancia en OO

- Característica fundamental del paradigma OO.
  - Motiva a que el acoplamiento entre objetos sea bajo.
  - Permite que el software escale mejor frente a cambios.
- Un objeto debe saber lo *mínimo indispensable* sobre los objetos que conoce.
  - De esta manera, los cambios internos no impactan en los otros objetos del sistema.
- Programar en términos del protocolo:
  - La representación interna de los objetos tiende a cambiar.

# Encapsulamiento



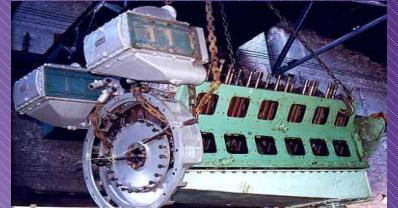
## Ejercicio - Punto

- Modelar un objeto *punto* que entienda el siguiente protocolo:
  - **getX()** “*Devuelve la coordenada x del punto*”
  - **setX(unNúmero)** “*Cambia la coordenada x del punto por unNúmero*”
  - **getY()** “*Devuelve la coordenada y del punto*”
  - **setY(unNúmero)** “*Cambia la coordenada y del punto por unNúmero*”
  - **igualA(otroPunto)** “*Devuelve si el punto es igual a otroPunto*”

## Ejercicio - Locomotora

- Modelar un juego de trenes eléctricos donde hay una locomotora con un motor y que se ubica sobre una vía
- La vía recibe indicaciones de un pulsador, con el cual se indica a la locomotora que avance o que se detenga
- Realizar el diagrama de secuencia del mensaje #avanzar y #detener del pulsador

# Ejercicio – Locomotora



avanzar

avanzar

avanzar

## Ejercicio – Locomotora

- Extender la solución anterior pero ahora el motor de la locomotora tiene 3 estados:
  - **Hacia atrás**
  - **Hacia adelante.**
  - **Apagado .**

## Comportamiento común entre Objetos

- Volvamos al ejemplo del banco: ¿Cuántos objetos Caja de Ahorro habrá?
- ¿Es necesario especificar el comportamiento de *cada* Caja de Ahorro? ¿O el comportamiento debería ser común a todas?
- ¿Qué cosas son comunes a todas las Cajas de Ahorro y qué cosas son particulares de cada una?
- Entonces ... ¿Cómo representamos este comportamiento común, de manera que cada Caja de Ahorro pueda reutilizarlo?

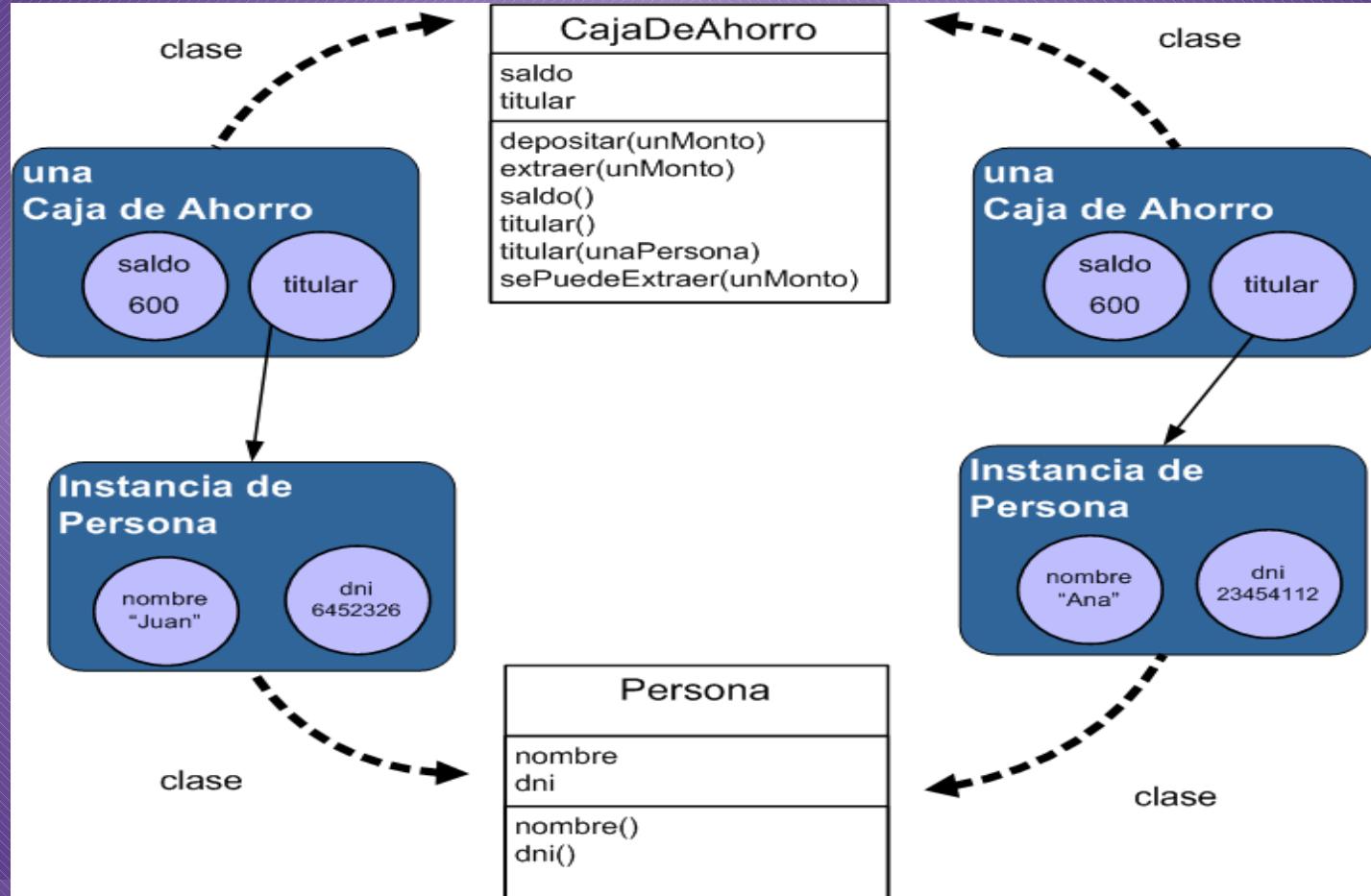
# Clases

- Una clase es una descripción abstracta de un conjunto de objetos.
- Las clases
  - Describen el formato de los objetos.
  - Agrupan comportamiento en común.
  - Pueden pensarse como *moldes* de un tipo específico de objeto

## Clases e instancias

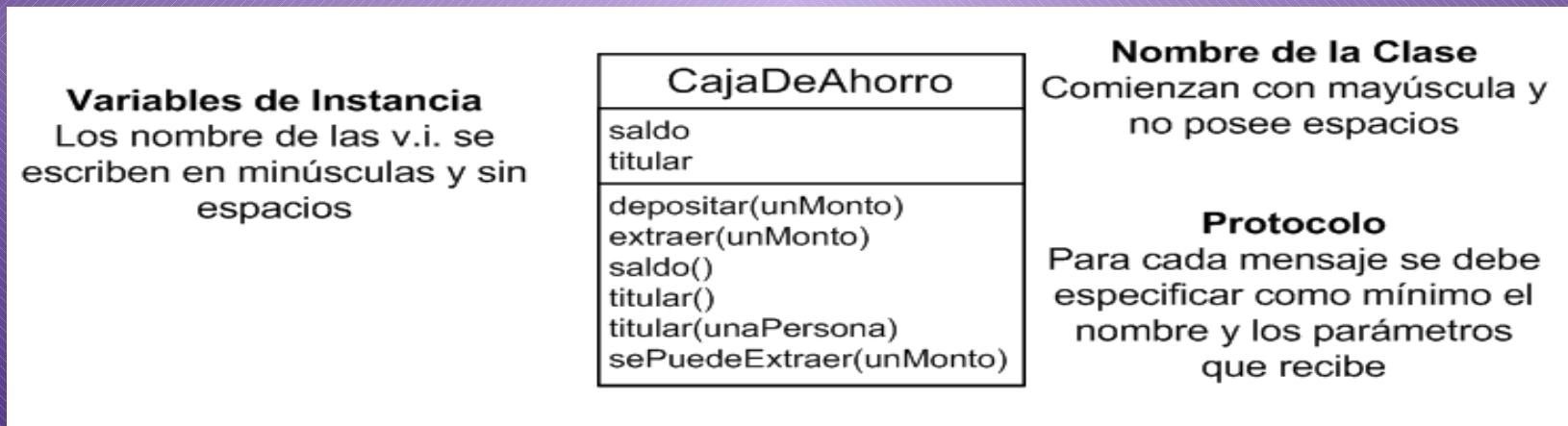
- Una clase es responsable de crear sus instancias, es decir, las clases se comportan como fábricas.
- Entonces las clases cumplen tres roles:
  - Agrupan el comportamiento común a sus instancias.
  - Definen la *forma* de sus instancias.
  - *Crean objetos que son instancia de ellas*
- En consecuencia todas las instancias de una clase se comportan de la misma manera
- Cada instancia mantendrá su propio estado interno

# Ejemplo de clases e instancias

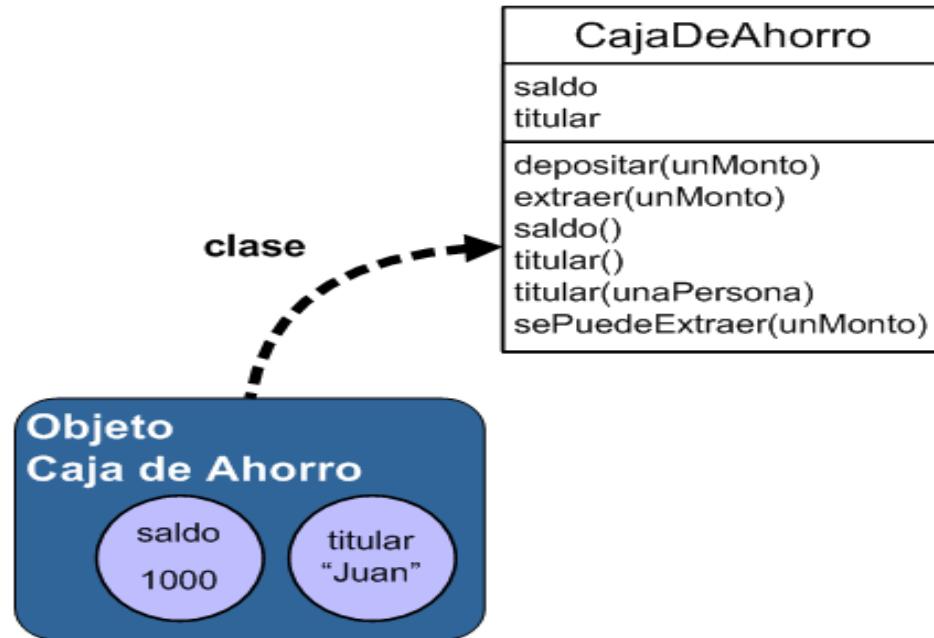


# Especificación de Clases

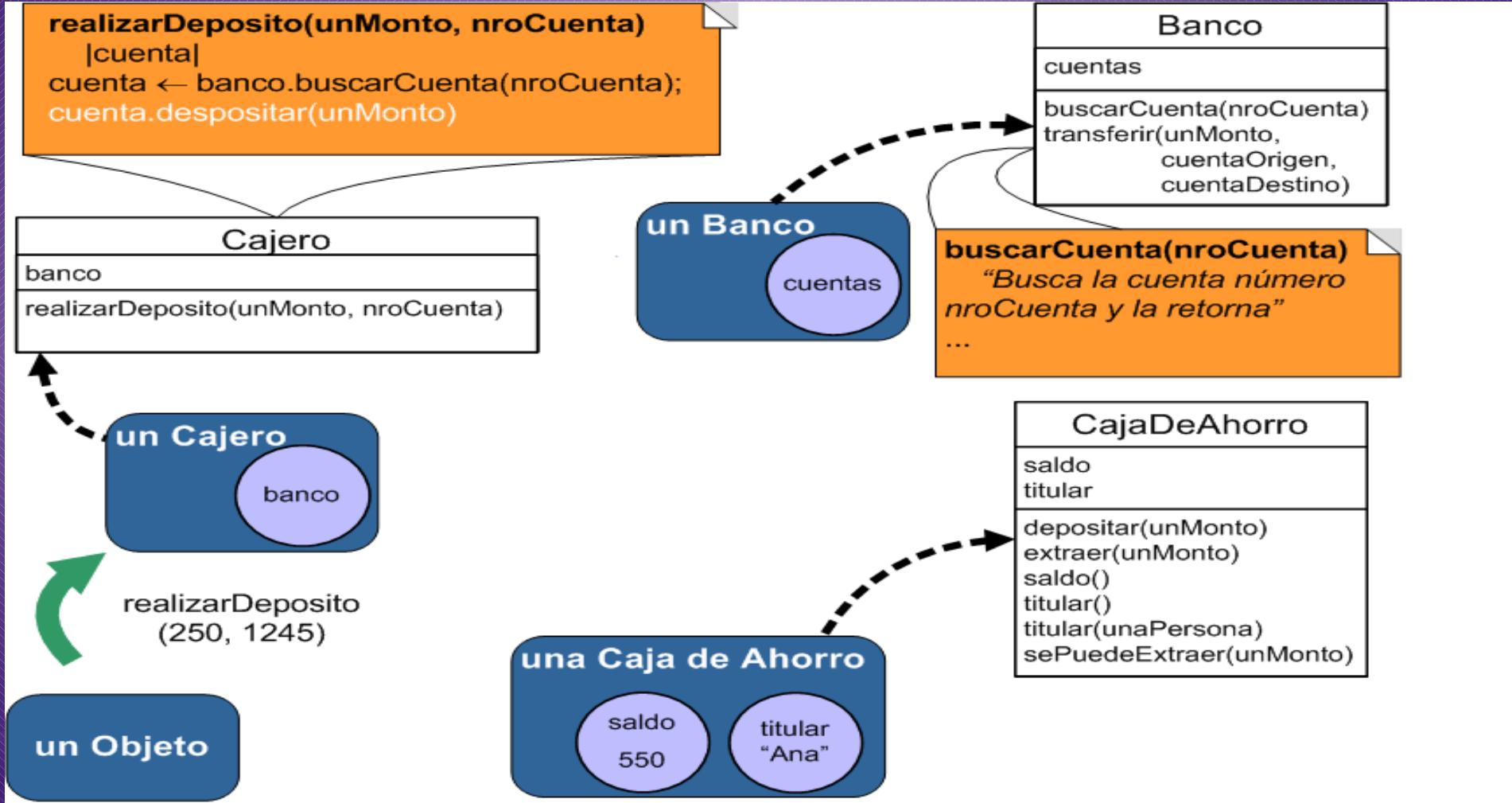
- Las clases se especifican por medio de un nombre, el estado o estructura interna que tendrán sus instancias y los métodos asociados que definen el comportamiento
- Gráficamente:



# Envío de mensajes con clases I



# Envío de mensajes con clases II



## Ejercicio – *Punto*

Agregar a la clase Punto los siguientes métodos

- mas (otroPunto)
- menos (otroPunto)

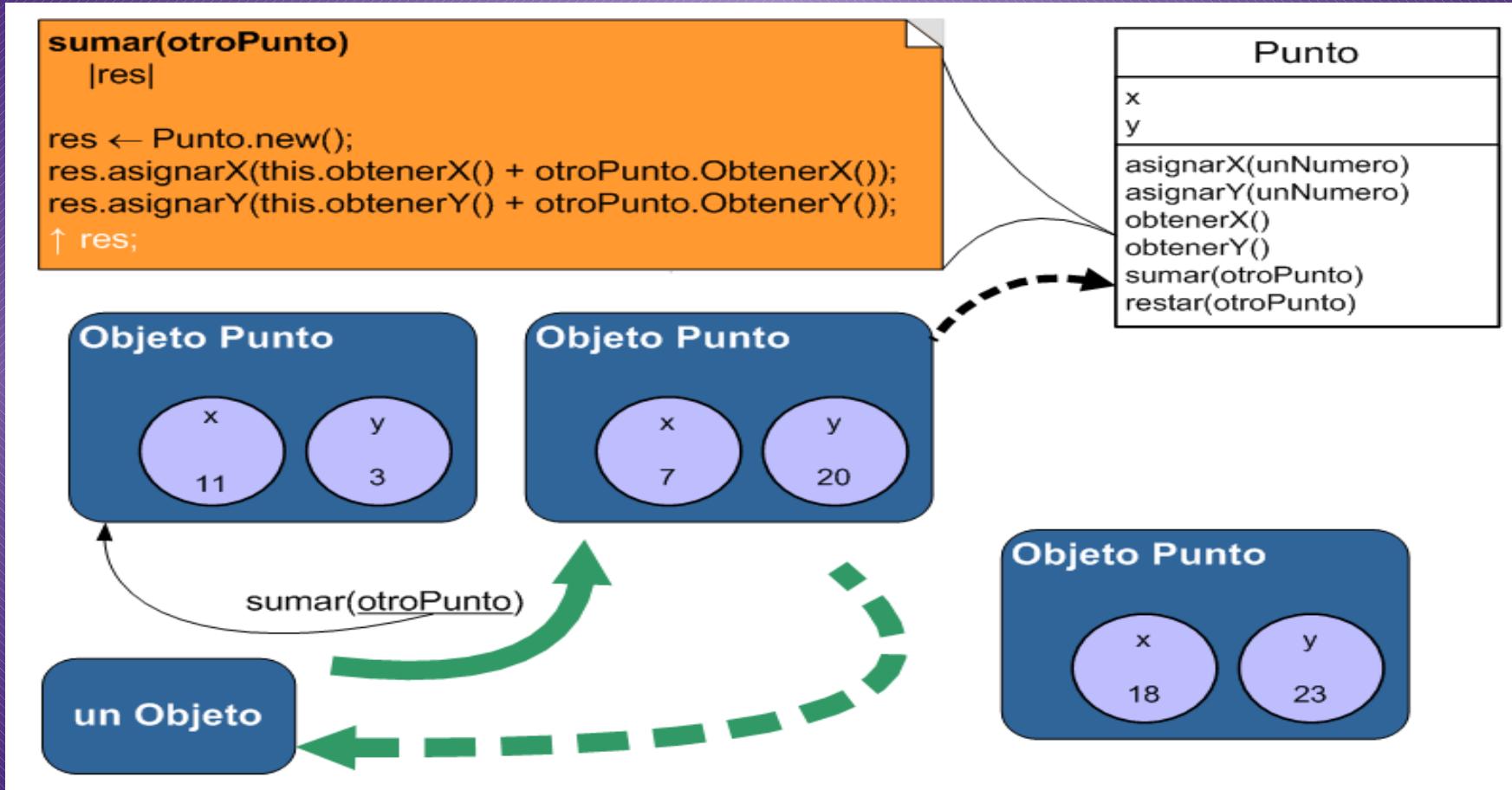
# Instanciación

- Es el mecanismo de creación de objetos.
- Los objetos se *instancian* a partir de un molde.
- La **clase** funciona como molde.
- Un nuevo objeto es una ***instancia*** de una clase.
- Todas las instancias de una misma clase
  - Tendrán la misma estructura interna.
  - Responderán al mismo protocolo (los mismos mensajes).

## Instanciación – uso de ***new***

- Comúnmente se utiliza la palabra reservada ***new*** para instanciar nuevos objetos.
- Según el lenguaje
  - ***new*** es un mensaje que se envía a la clase.
  - ***new*** es una operación especial.
- En C# el ***new*** es una operación especial.

# Instanciación

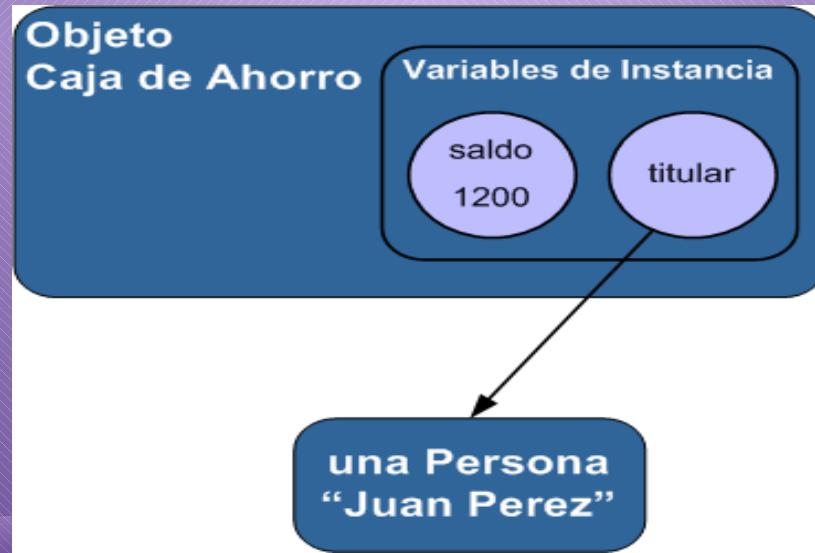


# Formas de Conocimiento

- Para que un objeto conozca a otro lo debe poder nombrar. Decimos que se establece una ligadura (binding) entre un nombre y un objeto.
- Podemos identificar tres formas de conocimiento o tipos de relaciones entre objetos.
  - Conocimiento Interno: Variables de instancia.
  - Conocimiento Externo: Parámetros.
  - Conocimiento Temporal: Variables temporales.
- Además existe una cuarta forma de conocimiento especial: las seudo-variables.

# Variables de instancia

- Definen una relación entre un objeto y sus atributos.
- Se definen explícitamente como parte de la estructura de la clase.
- La relación dura tanto tiempo como viva el objeto.



## Parámetros

- Se refiere a los parámetros de un mensaje.
- El nombre de la relación se define explícitamente en el nombre del método.
- La relación de conocimiento dura el tiempo que el método se encuentra activo.
- La ligadura entre el nombre y el objeto no puede alterarse durante la ejecución del método.

# Parámetros

```
transferir(unMonto, unaCuenta, otraCuenta)
```

```
    unaCuenta.extraer(unMonto);  
    otraCuenta.depositar(unMonto)
```

Objeto  
Banco

## Variables temporales

- Definen relaciones temporales dentro de un método.
- La relación con el objeto se crea durante la ejecución del método.
- El nombre de la relación se define explícitamente en el método.
- La relación se mantiene dentro del contexto donde fue definida la variable.
- Durante la ejecución del método, la ligadura entre el nombre y el objeto puede alterarse.

# Variables temporales

**realizarDeposito(unMonto, nroCuenta)**

*"Deposito unMonto en la cuenta nroCuenta"*

| cuenta |

```
cuenta ← banco.buscarCuenta(nroCuenta);
cuenta.depositar(unMonto);
```

Objeto  
Cajero  
Automatico

## La Pseudo variable: this

- Vimos que para mandarle un mensaje a un objeto hay que poder nombrarlo.
- ¿Cómo hace un objeto para mandarse un mensaje a sí mismo?
- *Pseudo variable*: como una variable ordinaria pero:
  - No se declara
  - No puede modificarse
- Con esta pseudo variable el objeto puede hacer referencia a sí mismo.

# this

**extraer(unMonto)**

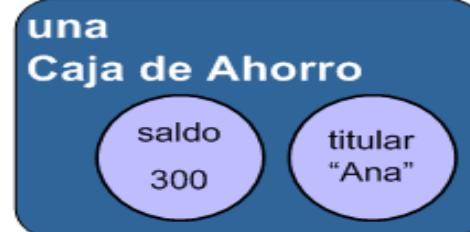
```
If(this.sePuedeExtraer(unMonto)) then  
    saldo ← saldo – unMonto  
else “error: no se puede girar en  
descubierto”
```

CajaDeAhorro
saldo
titular
depositar(unMonto)
extraer(unMonto)
saldo()
titular()
titular(unaPersona)
sePuedeExtraer(unMonto)



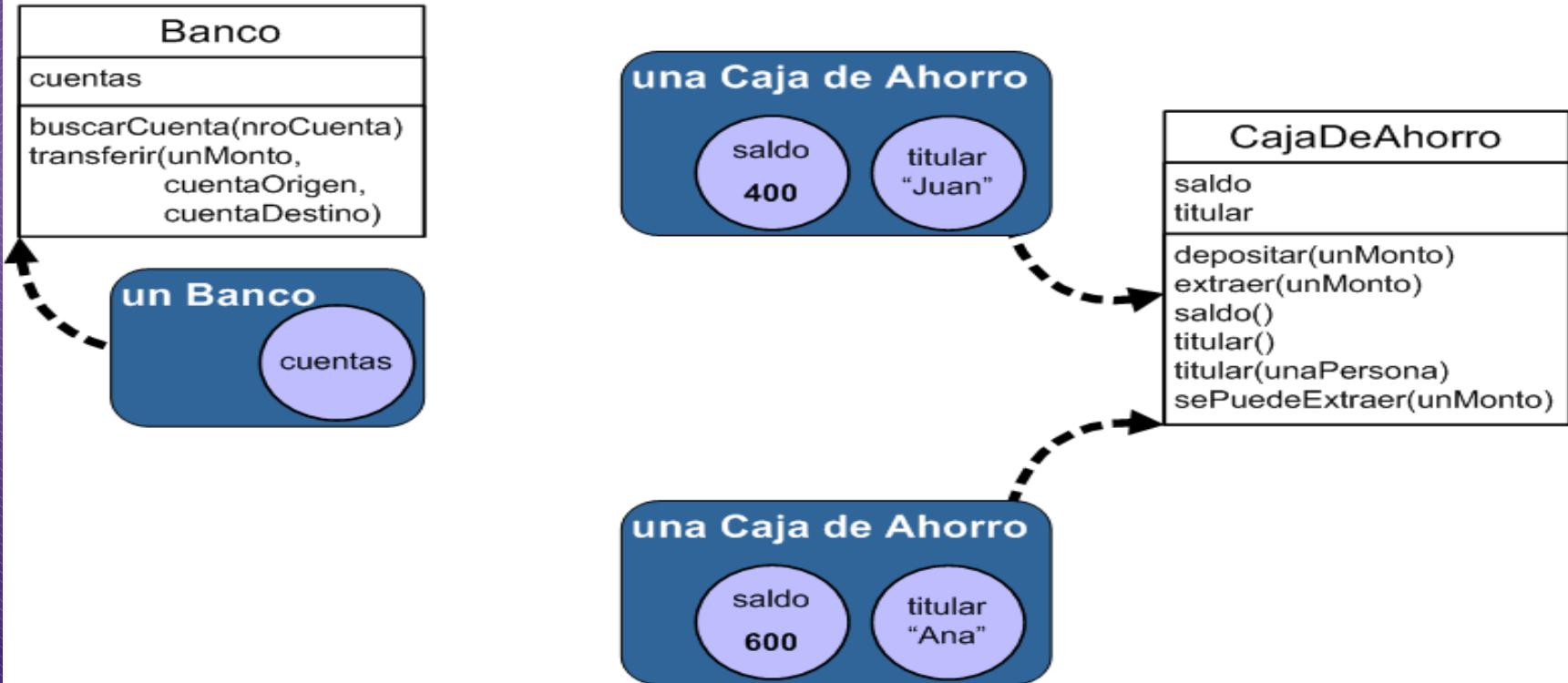
extraer(400)

un Objeto



ar

# Veamos otro ejemplo this



## Ejercicio - Fechas

- Modelar un objeto que represente una fecha
- Protocolo
  - `getDia()` / `setDia(unNúmero)`
  - `getMes()` / `setMes(unNúmero)`
  - `getAño()` / `setAño(unNúmero)`
  - `igualA(otraFecha)`
  - `mayorQue(otraFecha)` , `menorOIgualQue(otraFecha)` ,  
`mayorOIgualQue(otraFecha)`
  - `seEncuentraEntre(unaFecha, otraFecha)`
- Suponga que existe implementado el método  
`menorQue(otraFecha)` .

## Ejercicio – Lapso de tiempo

- Diseñar e implementar un objeto lapso de tiempo
- Protocolo
  - `desde()`
  - `hasta()`
  - `desdeHasta(unaFecha, otraFecha)`
  - `cantidadDeDias()`
  - `incluye(unaFecha)`
- ¿Podemos utilizar una implementación alternativa?

## Doble encapsulamiento

*“Es la cualidad de los objetos de ocultar los detalles de implementación y su estado interno del mundo exterior”*

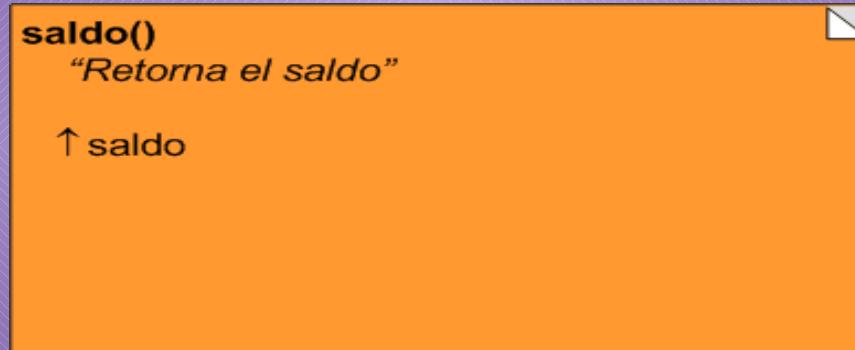
- Llevemos la idea de encapsulamiento un poco más lejos.
- Ahora pensemos que el objeto trate de usar solamente su propio protocolo lo más posible, sin poder modificar su estructura interna salvo a través de mensajes.

## Doble encapsulamiento - Ejemplo

- Recordemos las clase CajaDeAhorro, supongamos que se le cambia el nombre a la variable “saldo” por “montoInterno”.
- ¿En cuántos lugares del código hay que cambiar la referencia: “saldo” por “montoInterno”?
- ¿Cómo se puede solucionar este problema?

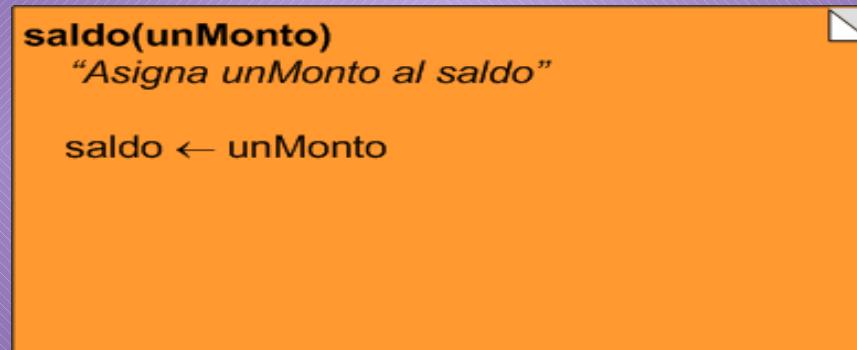
## Doble encapsulamiento - Ejemplo

- Cambiar las referencias de saldo por:



## Doble encapsulamiento - Ejemplo

- Cambiar las referencias de la asignación de saldo por:



# Doble encapsulamiento - Ejemplo

- Ahora el método depositar quedaría:

```
depositar(unMonto)
“Deposita unMonto al saldo”

this.saldo( this.saldo () +unMonto)
```

## Doble encapsulamiento - Ejemplo

- Volviendo a lo pedido, ahora ¿en cuántos lugares hay que cambiar las referencias?

**saldo()**  
“Retorna el saldo”  
↑ montoInterno

**saldo(unMonto)**  
“Asigna unMonto al saldo”  
montoInterno ← unMonto

## Doble encapsulamiento

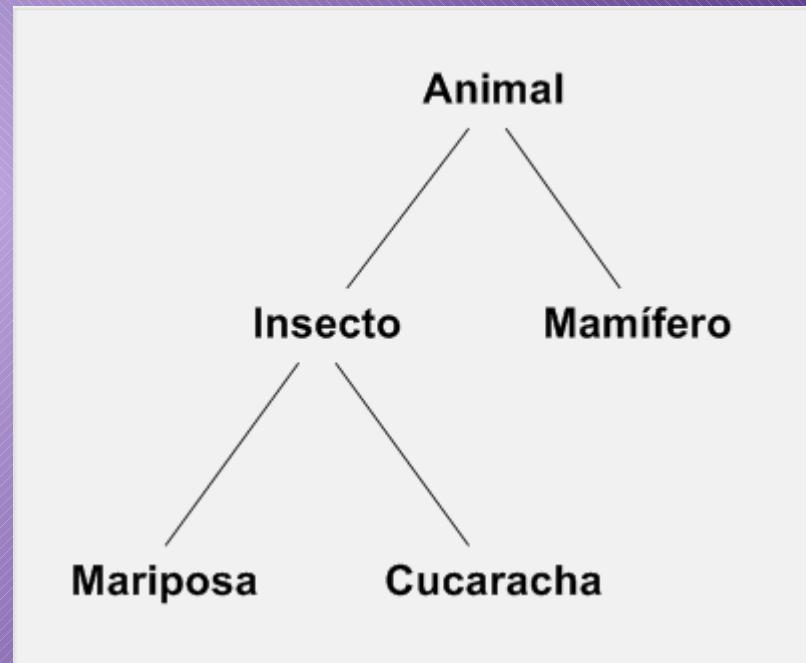
- Ventajas:
  - Permite la evolución del objeto agregando nuevos mensajes sin preocuparnos por su implementación.
  - Genera desacoplamiento interno.
  - Reuso y extensibilidad garantizada.

## Representación de conocimiento

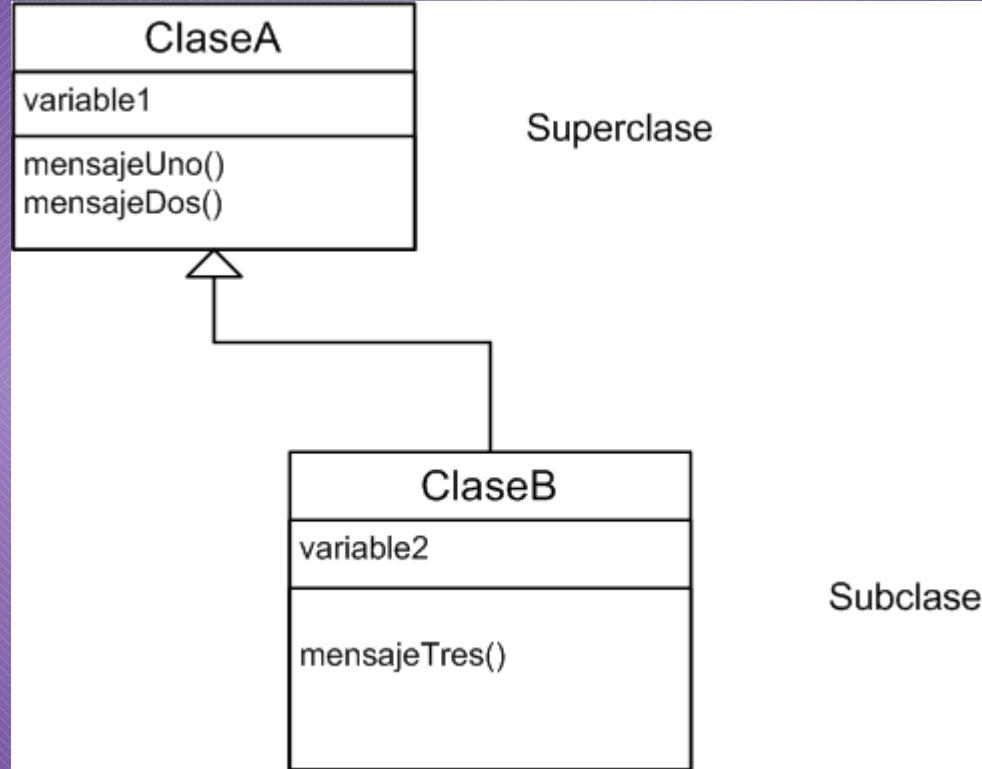
- Una clase representa un concepto en el dominio del problema.
- Usamos las clases para representar parte del conocimiento que adquirimos del dominio del problema.
- ¿Qué sucede cuando las clases comparten parte del conocimiento al cual representan?
  - Subclasificación

## **Subclasificación**

- Se reúne el comportamiento y la estructura común en una clase, la cual cumplirá el rol de superclase.
- Se conforma una jerarquía de clases.
- Luego otras clases pueden cumplir el rol de subclases, heredando ese comportamiento y estructura en común.
- Cumple la relación es-un.



# Subclasiﬁcación



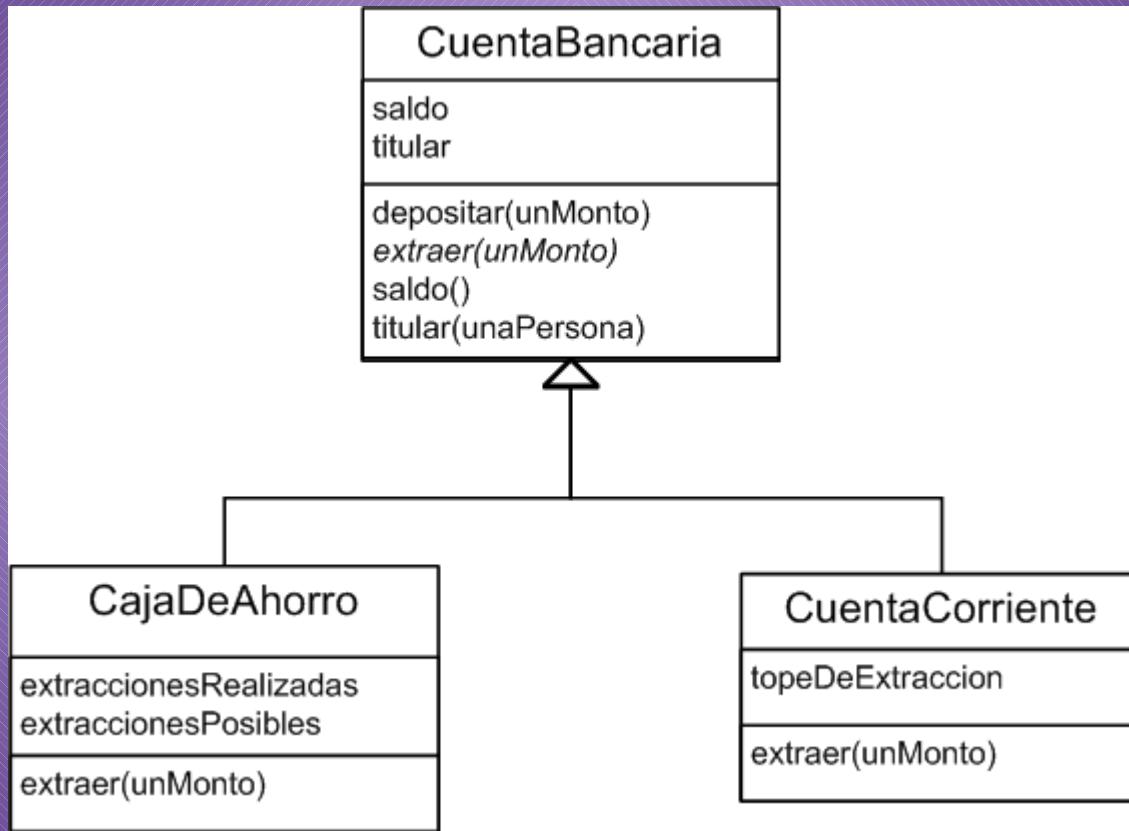
## Ejemplo de una Jerarquía de Clases

- Existen dos tipos de cuentas bancarias:
  - Cuentas corrientes.
  - Cajas de ahorro.
- Si revisamos el comportamiento nos encontraremos con las siguientes características en común:
  - Ambas llevan cuenta de su saldo.
  - Ambas permiten realizar depósitos.
  - Ambas permiten realizar extracciones.

## Ejemplo de una Jerarquía de Clases

- Pero cada una tiene un tipo de restricción distinto en cuanto a las extracciones:
  - Cuentas corrientes: permiten que el cliente gire en descubierto (con un tope pactado con cada cliente).
  - Cajas de ahorro: poseen una cantidad máxima de extracciones mensuales (para todos los clientes). No se permite girar en descubierto.
- ¿Cómo podemos reutilizar las características en común?

# Ejemplo de una Jerarquía de Clases



## Relación es-un

- En toda jerarquía de clases, se debe respetar la relación es-un entre una clase y su superclase.
- Por ejemplo
  - Una **caja de ahorro** es-una **cuenta bancaria**.
  - Un **círculo** es-una **figura**.
  - Una **figura** es-un **objeto** ...

## Ejercicio – Subclasificación

- Dadas las siguientes clases agrúpelas en las jerarquías que considere necesarias.

Terrestre

Vientos

Moto

Deporte

Tuba

Futbol

Violin

Tenis

Barco

Acuático

Trompeta

Omnibus

Guitarra

Buque

InstrumentoMusical

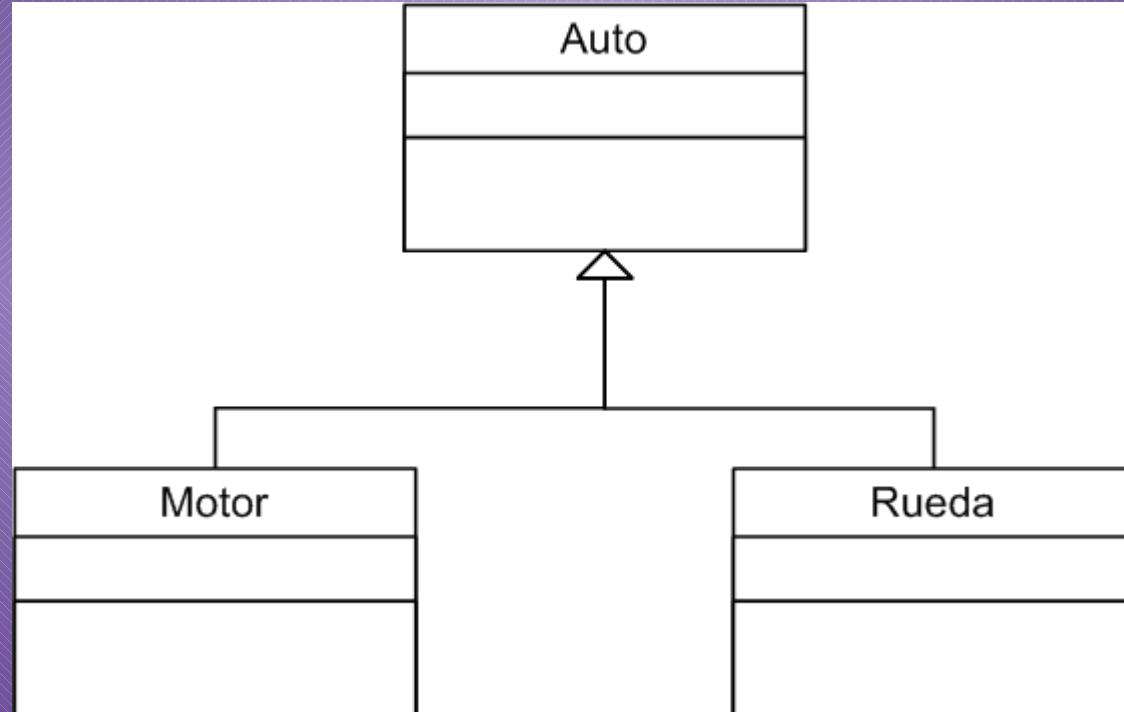
Rugby

Vehiculo

Cuerdas

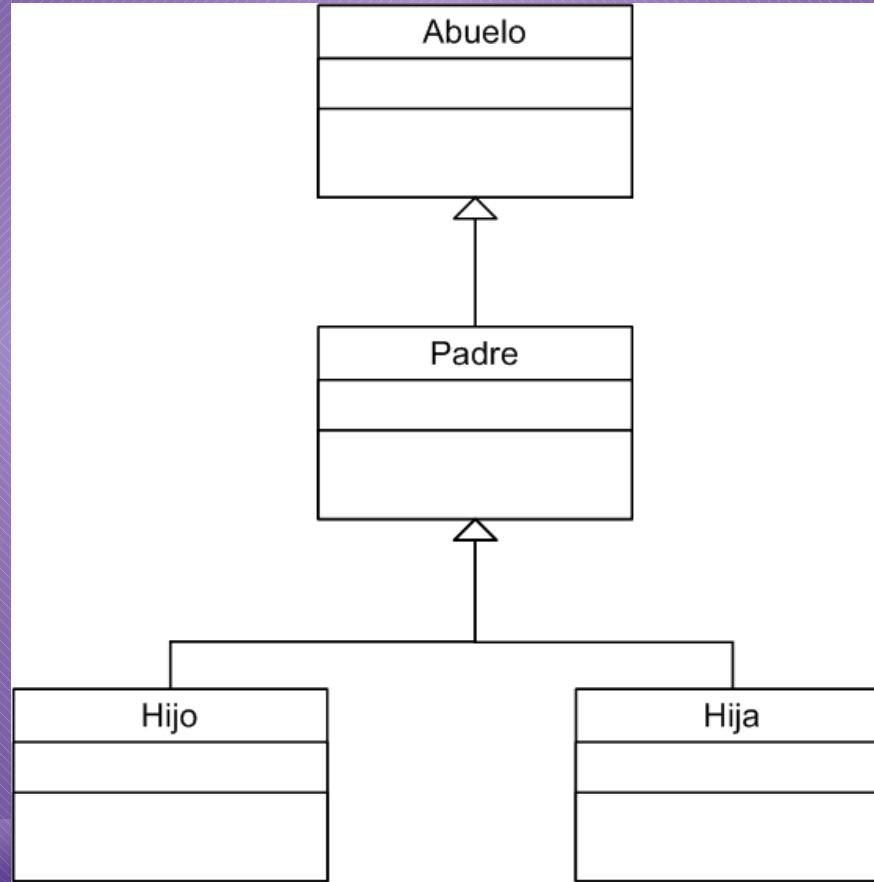
# Ejercicio – Subclasiﬁcación

- ¿Es correcta la siguiente jerarquía?



# Ejercicio – Subclasiﬁcación

- ¿Es correcta la siguiente jerarquía?



# Herencia

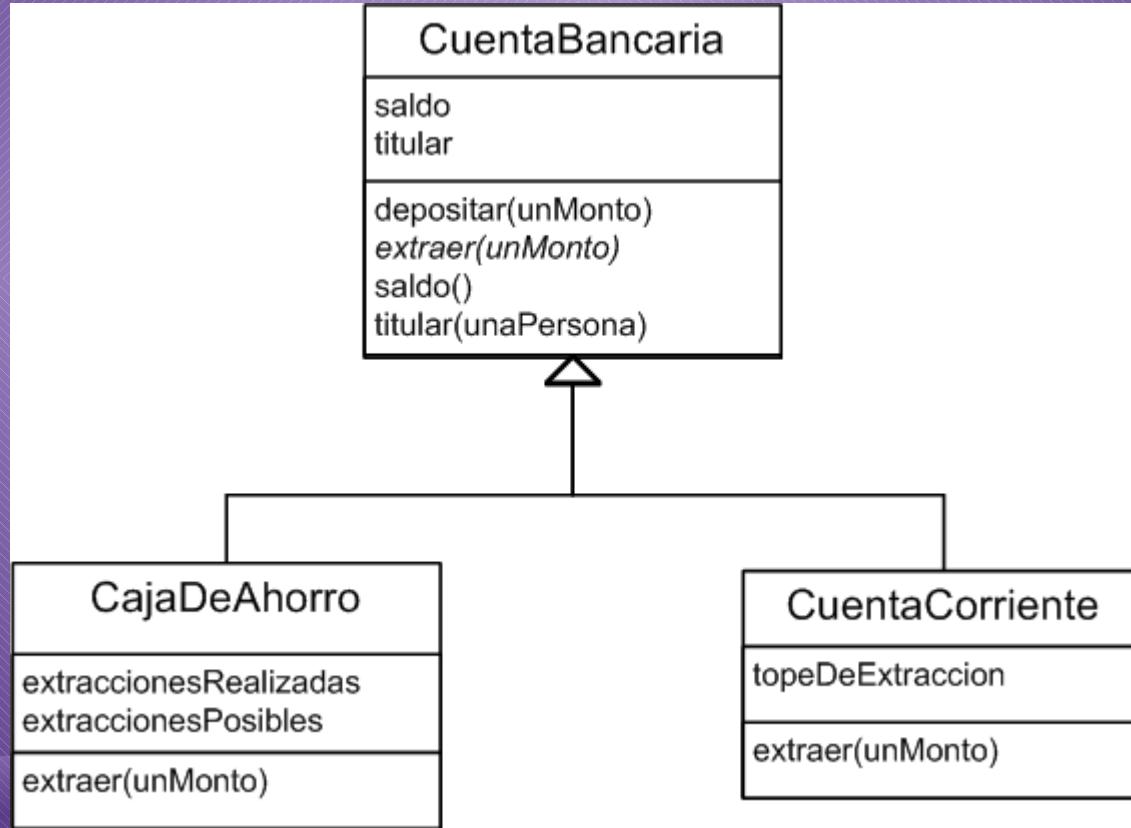
- Es el mecanismo por el cual las subclases reutilizan el comportamiento y estructura reunido en sus superclases.
- La herencia permite:
  - Crear una nueva clase como refinamiento de otra.
  - Diseñar e implementar sólo la diferencia que presenta la nueva clase.
  - Abstraer las similitudes en común.

# Herencia

- Toda relación de herencia implica:
  - Herencia de comportamiento
    - Una subclase hereda todos los métodos definidos en su superclase.
    - Las subclases pueden redefinir el comportamiento de su superclase.
  - Herencia de estructura
    - No hay forma de restringirla.
    - No es posible redefinir el nombre de un atributo que se hereda.

# Ejercicio – Cuenta Bancaria

- Implementar el mensaje extraer(unMonto) en cada una de las subclases de CuentaBancaria.



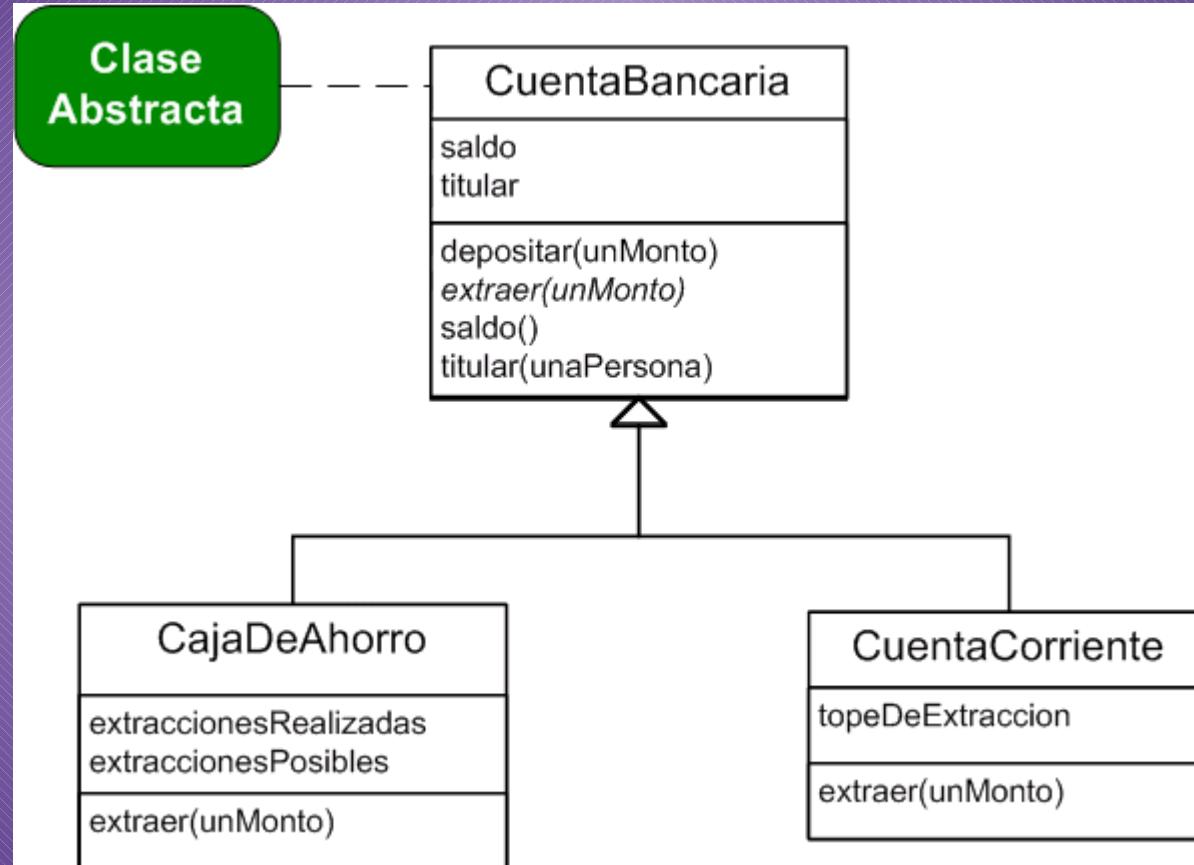
## Ejercicio – Cuenta Bancaria

- Recordemos las restricciones:
  - Cuentas corrientes: permiten que el cliente gire en descubierto (con un tope pactado con cada cliente).
  - Cajas de ahorro: poseen una cantidad máxima de extracciones mensuales (para todos los clientes). No se permite girar en descubierto.

# Clases Abstractas

- Son clases a partir de las cuales no pueden crearse instancias.
- ¿Entonces, para qué sirven?
  - La herencia es un mecanismo poderoso para factorizar comportamiento común
  - Se puede mejorar el algoritmo de la superclase y automáticamente lo heredarán todas sus subclases.
  - No necesitan estar completamente implementadas
  - Pueden especificar métodos que será definidos por sus subclases

# Clases Abstractas



# Bibliografía y Licencia

- Textos tomados, corregidos y modificados de diferentes páginas de Internet, tutoriales y documentos.
- Este documento se encuentra bajo Licencia Creative Commons Attribution – NonCommercial - ShareAlike 4.0 International (CC BY-NC-SA 4.0), por la cual se permite su exhibición, distribución, copia y posibilita hacer obras derivadas a partir de la misma, siempre y cuando se cite la autoría del Prof. Matías E. García y sólo podrá distribuir la obra derivada resultante bajo una licencia idéntica a ésta.
- Autor:

*Matías E. García*

---

Prof. & Tec. en Informática Aplicada  
[www.profmatiasgarcia.com.ar](http://www.profmatiasgarcia.com.ar)  
[info@profmatiasgarcia.com.ar](mailto:info@profmatiasgarcia.com.ar)



[www.profmatiasgarcia.com.ar](http://www.profmatiasgarcia.com.ar)