Orientacion a Objetos 1

- **Profesores:**

Dra. Alicia Diaz
Dra. Roxana Giandini
Dr. Gustavo Rossi

E-mails: [alicia, giandini, gustavo]@lifia.info.unlp.edu.ar

www.lifia.info.unlp.edu.ar/es/alicia.htm
http://www.lifia.info.unlp.edu.ar/es/roxana.htm
http://www.lifia.info.unlp.edu.ar/es/rossi.htm

Contenidos del Curso

- Conceptos basicos de Objetos
- Introduccion a la Modelizacion
- Introduccion al desarrollo de sistemas de gran porte
- “Cultura” general respecto a lo que pasara en los proximos 5 años (al menos)

Conocimientos que suponemos: programación básica, concepto de variable, estructuras elementales, concepto de puntero

Contexto

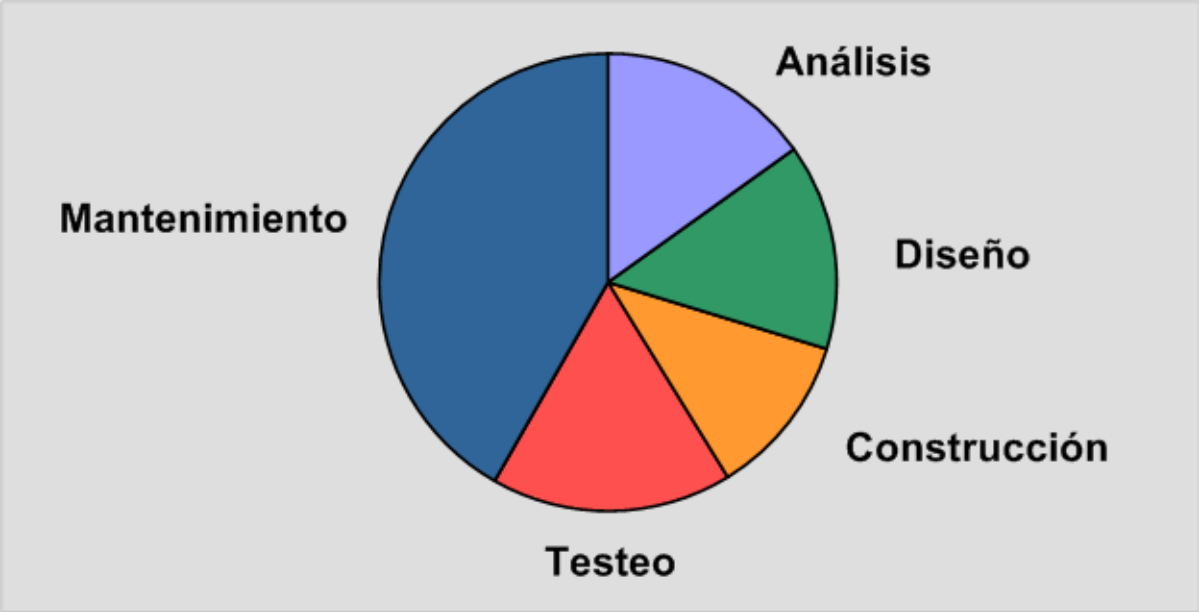
- En los 90: Computacion interactiva y Web Estatica
- En los 2000: Evolucion de la Web. Aparicion de la telefonia movil

En los 2010: Internet Movil, Internet de las cosas, Computacion en la Nube

En Software: Metodos Agiles vs Unificados, Desarrollo conducido por modelos, Software mas “volatil”, Requerimientos cambiantes permanentemente. Clientes y usuarios mejor formados y piden funcionalidad mas sofisticada

Motivación

- Manejar complejidad
- Flexibilidad al cambio
- Mejorar la reusabilidad
- **Minimizar costos de mantenimiento**



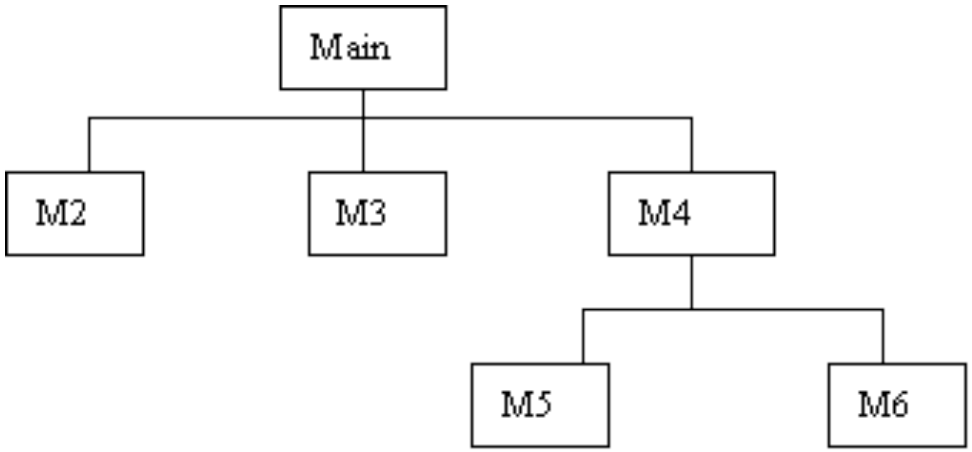
Costo asociado a cada etapa del desarrollo

Cómo minimizar los costos del desarrollo?

- Software mantenible
- Software extensible
- Software fácil de leer y entender

Programación Estructurada

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



- Asignación, secuencia, iteración, condicionales

Práctica OO1 - 2015

Medios de comunicación
(Toda notificación importante se hará por estos medios)

- Google Site: https://sites.google.com/site/objetos12015/
- Google Group: https://groups.google.com/forum/#!forum/objetos-1-2015
  - Solicitar suscripción enviando **Nro de Alumno**

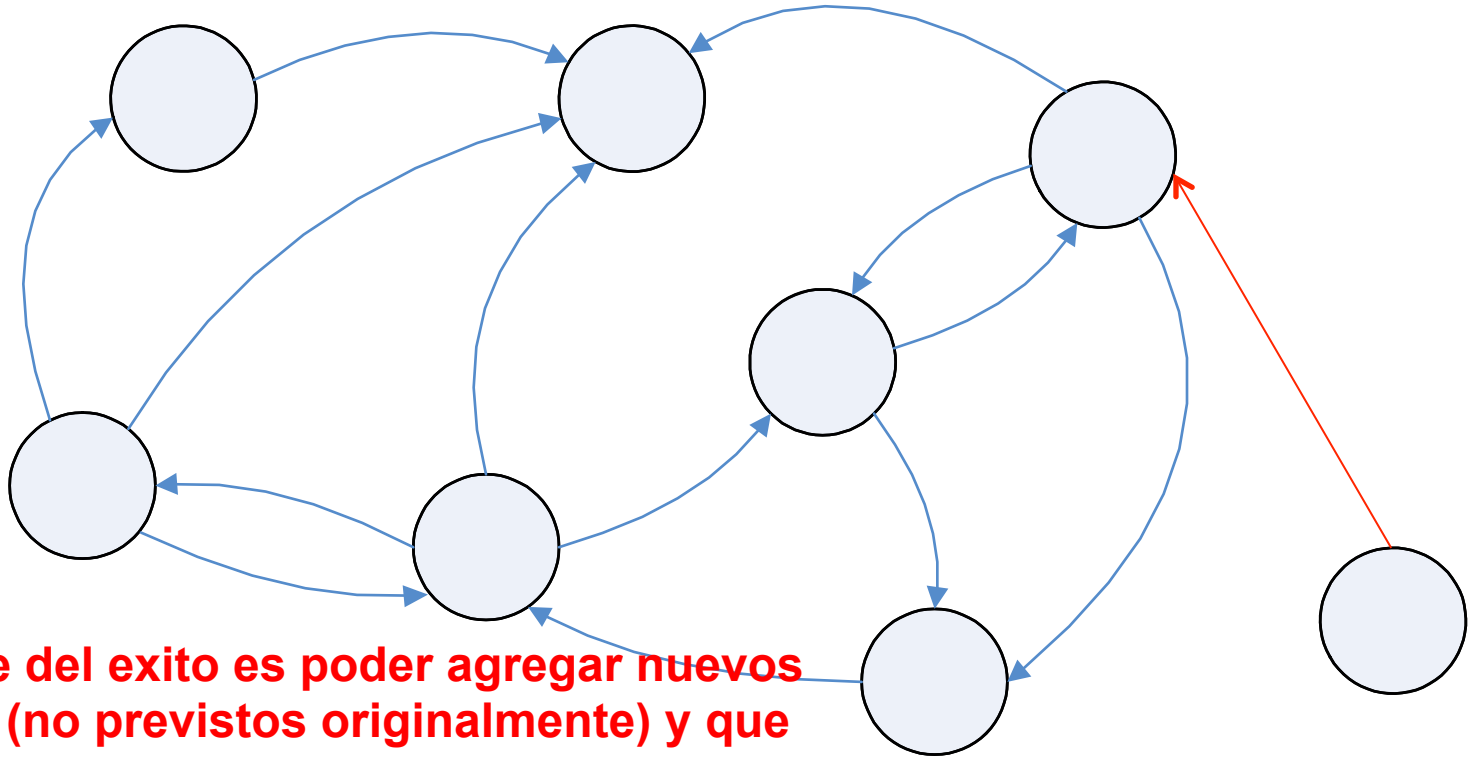
Actividades Prácticas

- Turnos de práctica: inscripción semana del 24/Ago
- Comienzo de actividades prácticas (consultas y explicaciones): semana del 31/Ago

Programa Orientado a Objetos

- ¿Qué es un programa OO?

Un conjunto de ***objetos*** que ***colaboran*** enviándose ***mensajes***. Todo computo ocurre “dentro” de los objetos



La clave del exito es poder agregar nuevos objetos (no previstos originalmente) y que el sistema “no se entere”

Programación Orientada a Objetos

- Los sistemas están compuestos por un conjunto de **objetos**.
- Los objetos son **responsables** de:
  - conocer sus propiedades,
  - conocer otros objetos (con los que colaboran) y
  - 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)

¿Qué es un objeto?

- Elemento primario que utilizamos para construir programas, desde lo más básico a lo más complejo.

- Es una ***abstracción*** de una ***entidad*** del ***dominio del problema***. Ejemplos?

- *Puede representar conceptos del espacio de la solucion (estructuras de datos, tipos "basicos", archivos, ventanas, iconos..)*

- Un objeto tiene un ***comportamiento*** asociado.

Características de los Objetos

- Un objeto tiene:
  - ***Identidad***.
    - para distinguir un objeto de otro
  - ***Conocimiento***.
    - En base a sus relaciones con otros objetos y su estado interno
  - ***Comportamiento***.
    - Conjunto de mensajes que un objeto sabe responder

Identidad

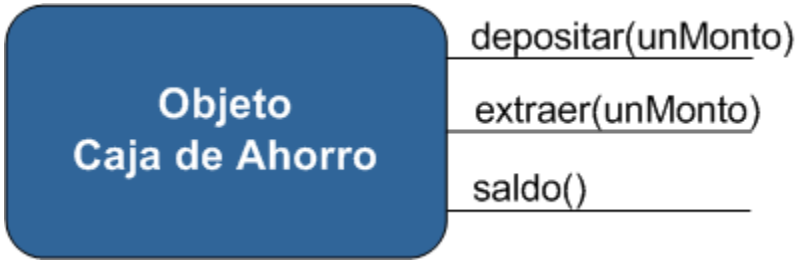
- Es una propiedad intrínseca de los objetos.
- Un objeto sólo es idéntico a sí mismo.
- No es lo mismo que *igualdad*.
  - Dos objetos pueden tener las mismas propiedades pero no son *el mismo* objeto

El estado interno

- El estado interno de un objeto determina su *conocimiento*.
- El estado interno está dado por:
  - Propiedades intrínsecas del objeto.
  - Relaciones con otros objetos con los cuales colabora para llevar a cabo sus responsabilidades.
- El estado interno se mantiene en las *variables de instancia (v.i.)* del objeto.
- Es **privado** del objeto. Ningún otro objeto puede accederlo.

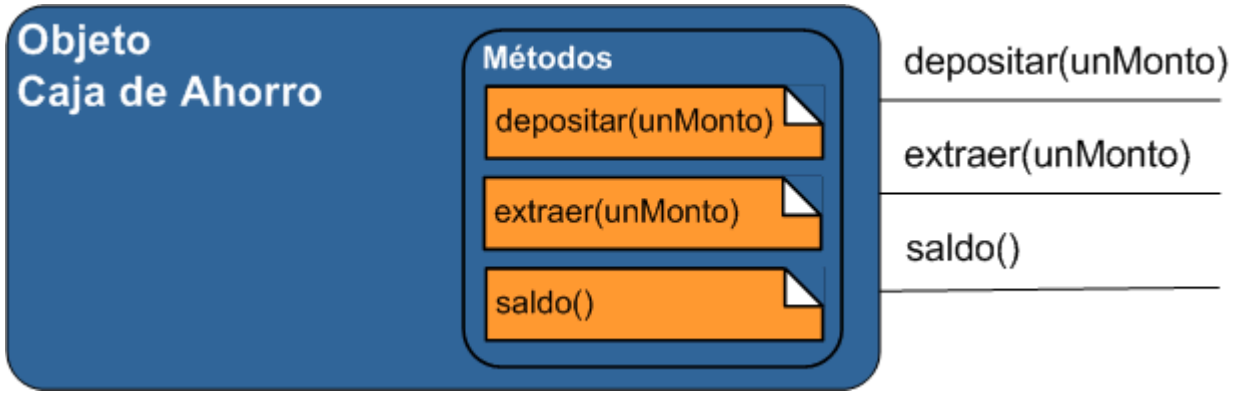
Comportamiento

- 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:

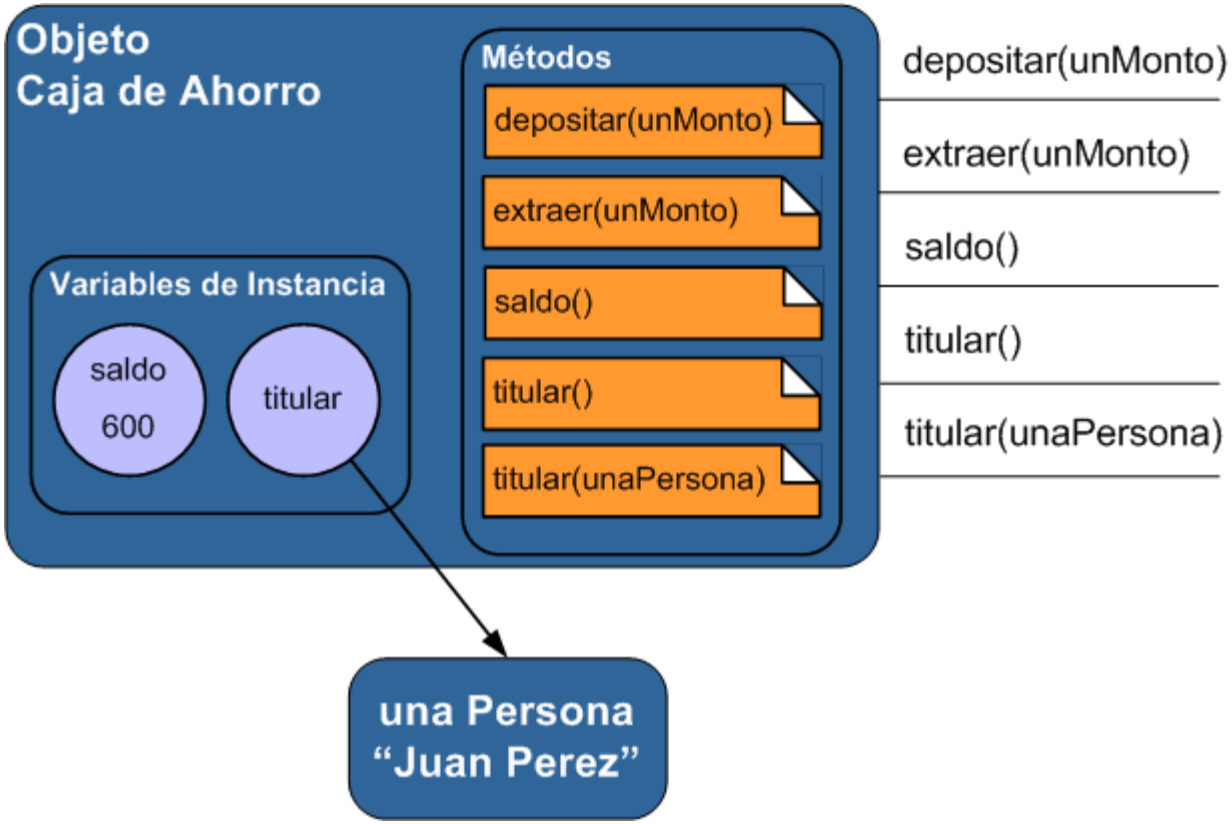


Comportamiento – implementación

- La **implementación** de cada mensaje (es decir, la manera en que un objeto responde a un mensaje) se especifica a través de un ***método***.
- Cuando un **objeto** recibe un **mensaje** responde activando el ***método*** asociado.
- El que envía el mensaje ***delega*** en el receptor la manera de resolverlo, que es ***privada*** del objeto.



Ejemplo Caja de Ahorro



Envío de un mensaje

- 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.

Especificación de un Mensaje

- ¿Cómo se especifica un mensaje?
  - ***Nombre***: correspondiente al protocolo del objeto receptor.
  - ***Parámetros***: 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
```

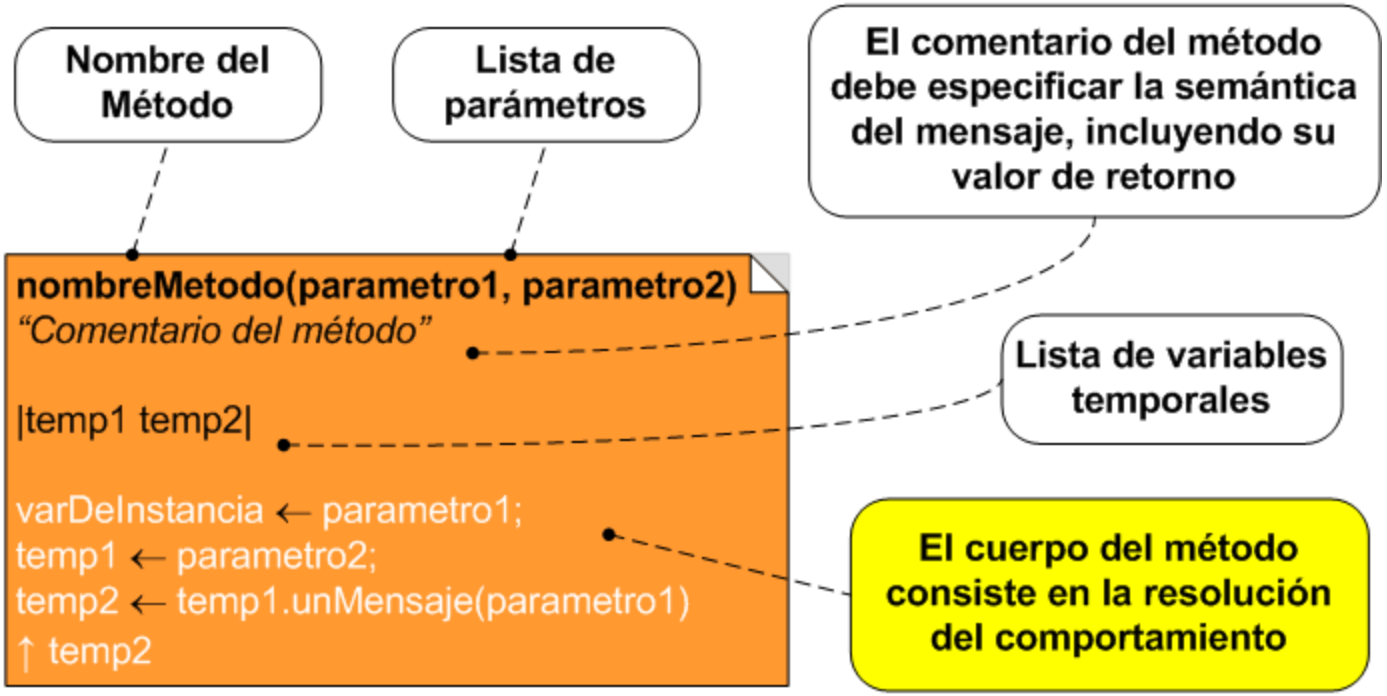
- Ejemplo empleando la sintaxis propuesta
  - Decirle a una cuenta bancaria que deposite $100 se escribe como:

```
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(pseudo lenguaje a la Java)



Ejemplo – Depositar en Cuenta Bancaria

depositar(unMonto)

"Agrega unMonto al saldo actual de la cuenta"

saldo ← saldo + unMonto

Ejemplo – Cajero

realizarDeposito(unMonto, nroCuenta)

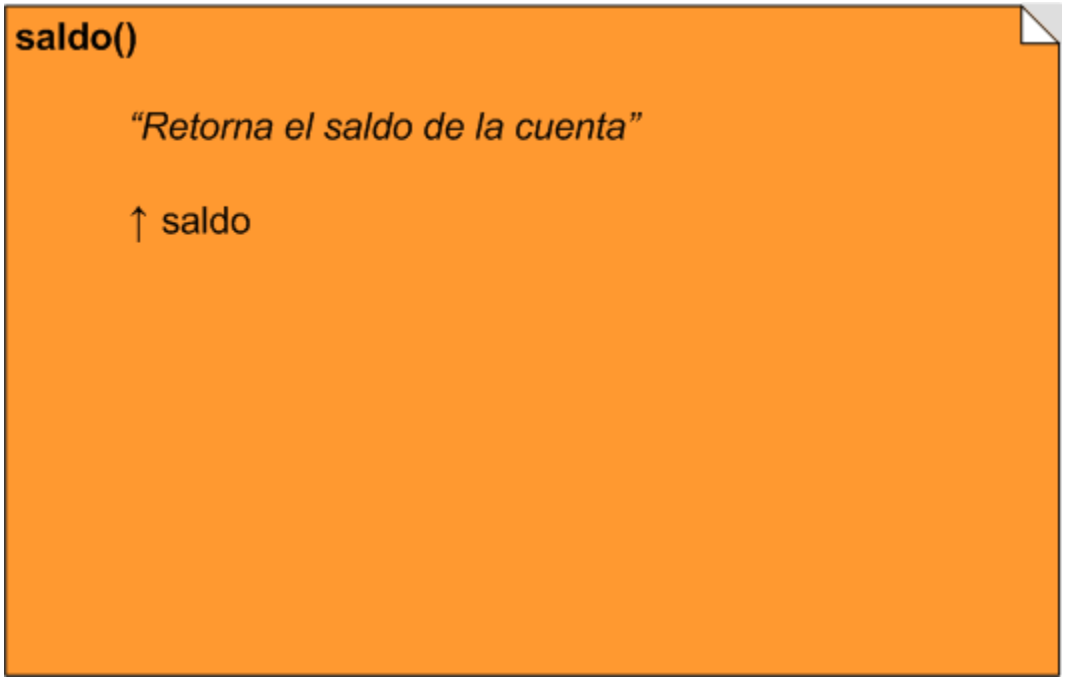
"Deposita unMonto en la cuenta numero nroCuenta"

|cuenta|

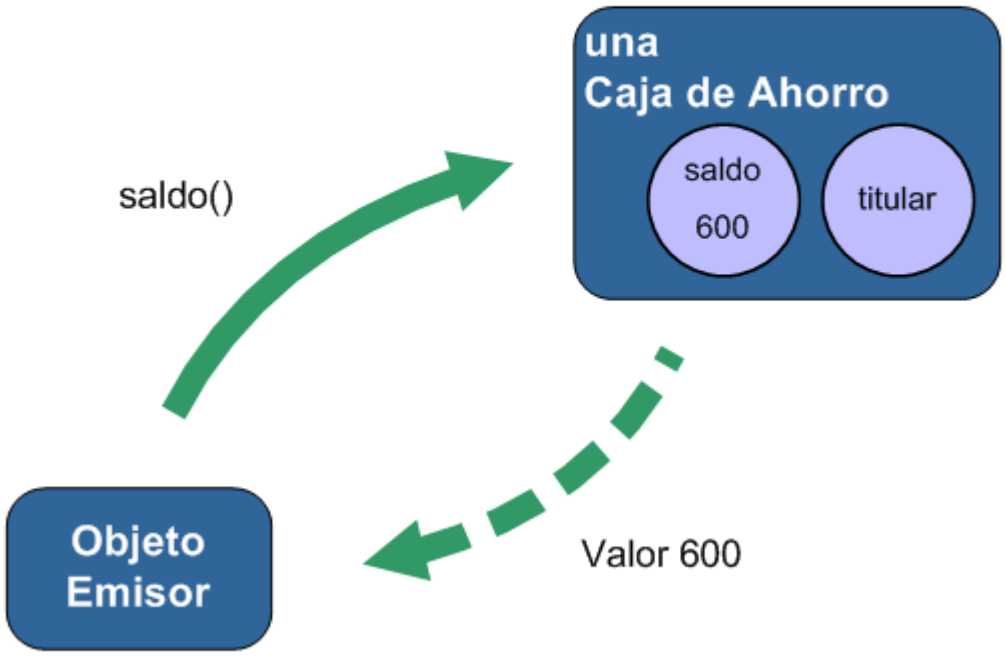
cuenta ← banco.buscarCuenta(nroCuenta);
cuenta.despositar(unMonto)

Saldo

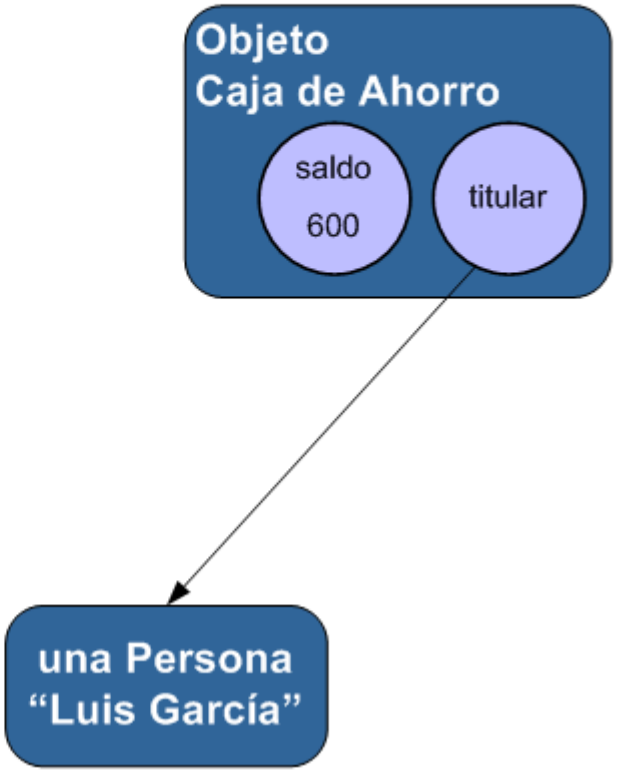
- ¿Cómo sería el método del mensaje **`saldo`**?



Envío del mensaje saldo

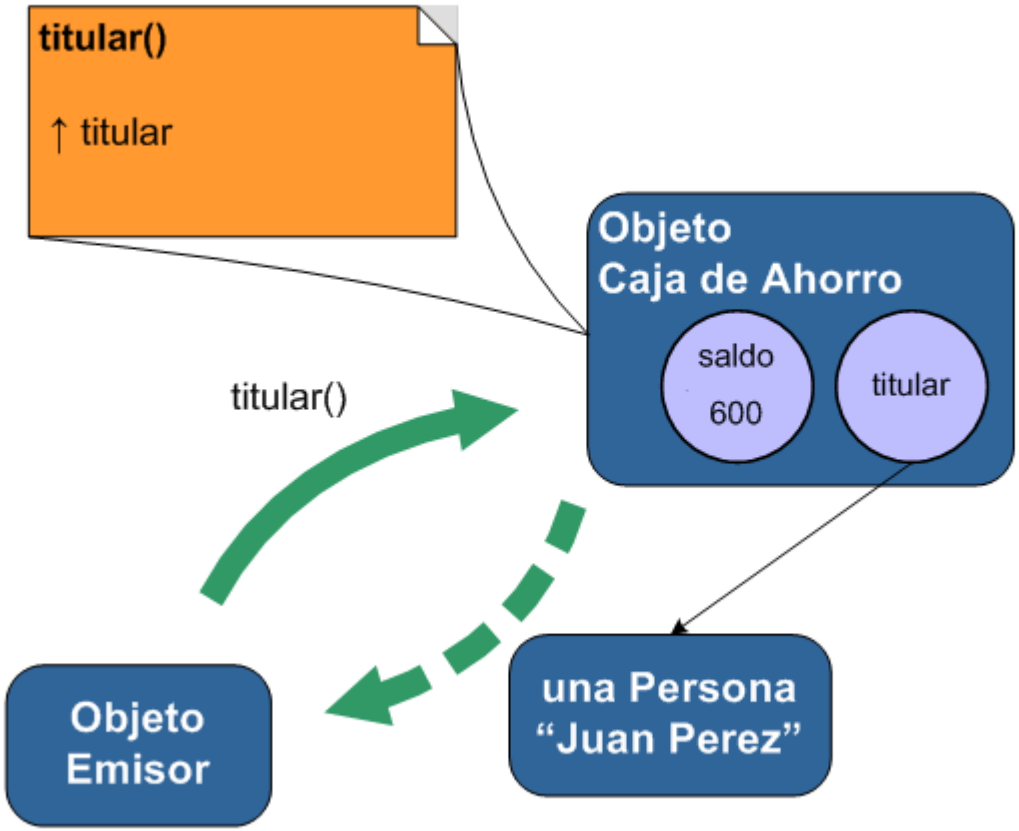


Cambio del estado interno



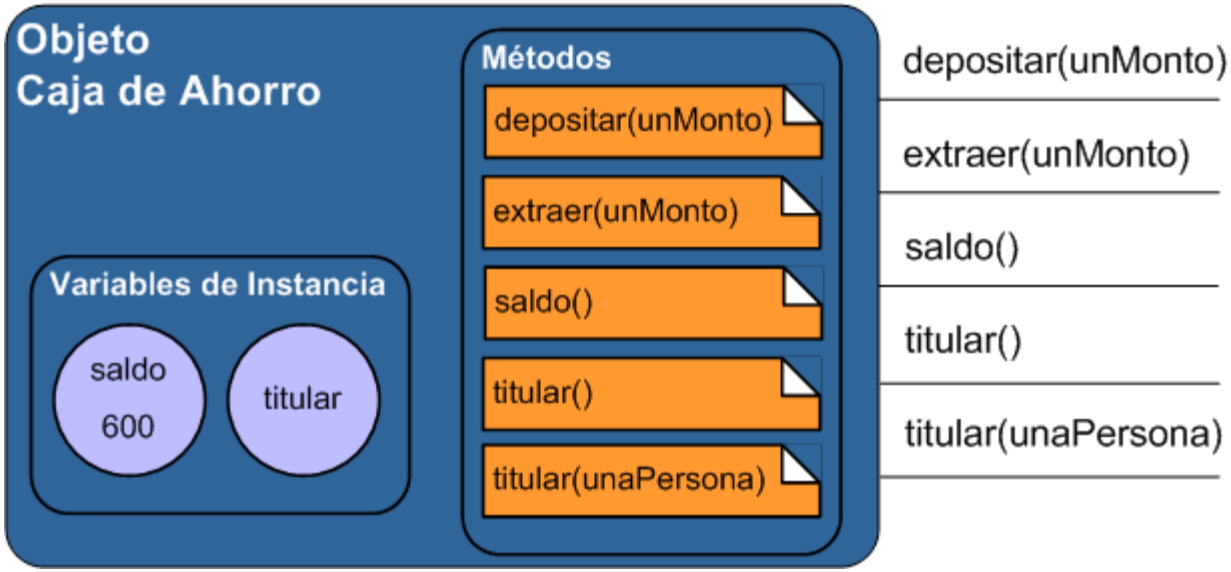
Ejemplo - Retornar el titular

- ¿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 ← saldo + unMonto

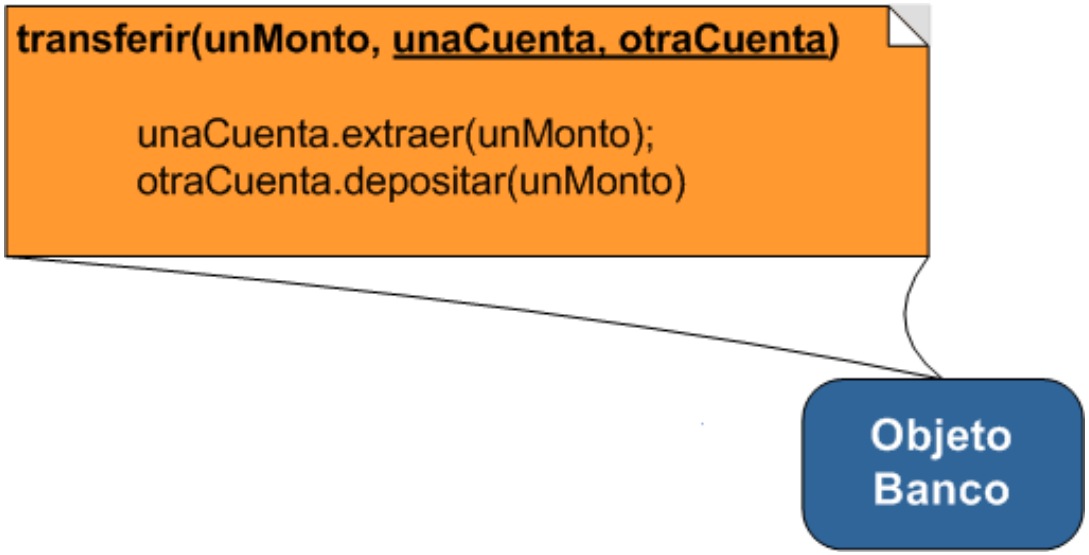
Formas de Conocimiento

- 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 pseudo-variables.

Parámetros

- Se refiere a los parámetros de un mensaje.
- 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.

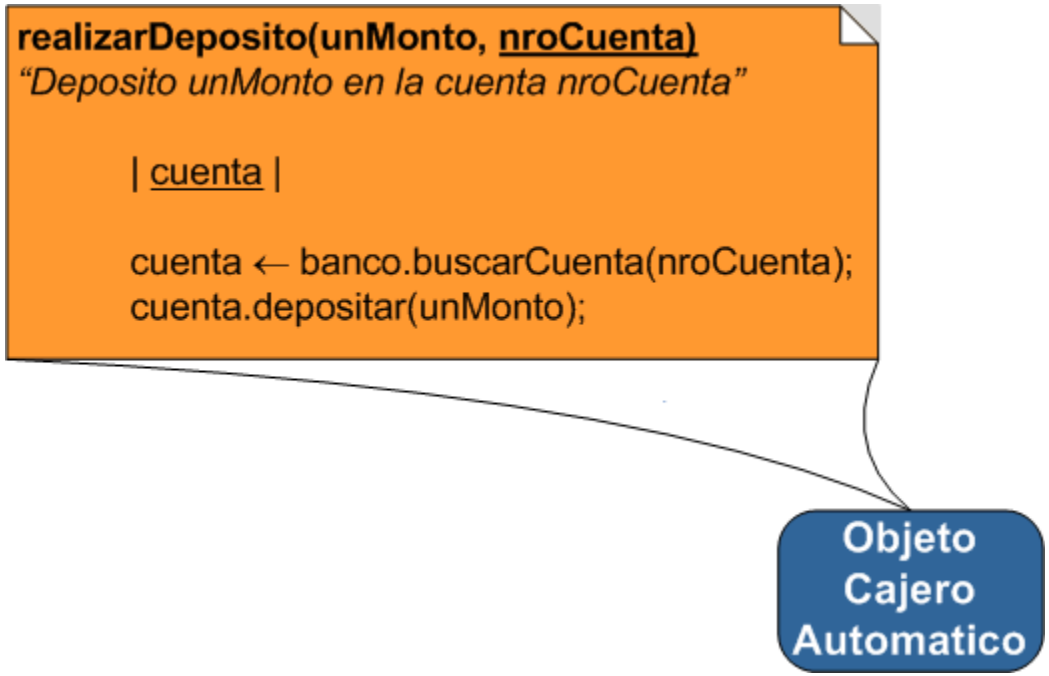
Parámetros


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.
- 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



Comportamiento y estado interno

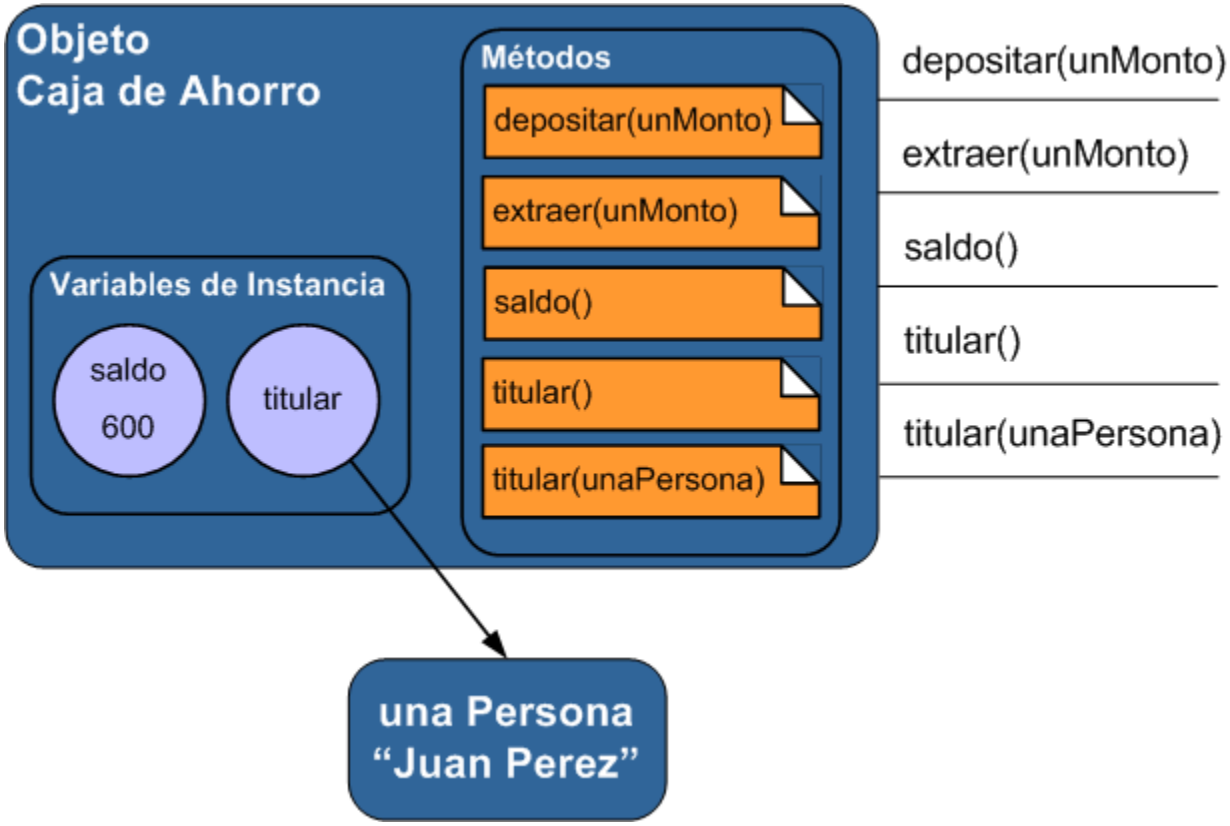
- Dijimos que el comportamiento tiene una implementación privada.
- También que el estado interno es privado.
- ¿De qué hablamos cuando hablamos de privacidad?

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



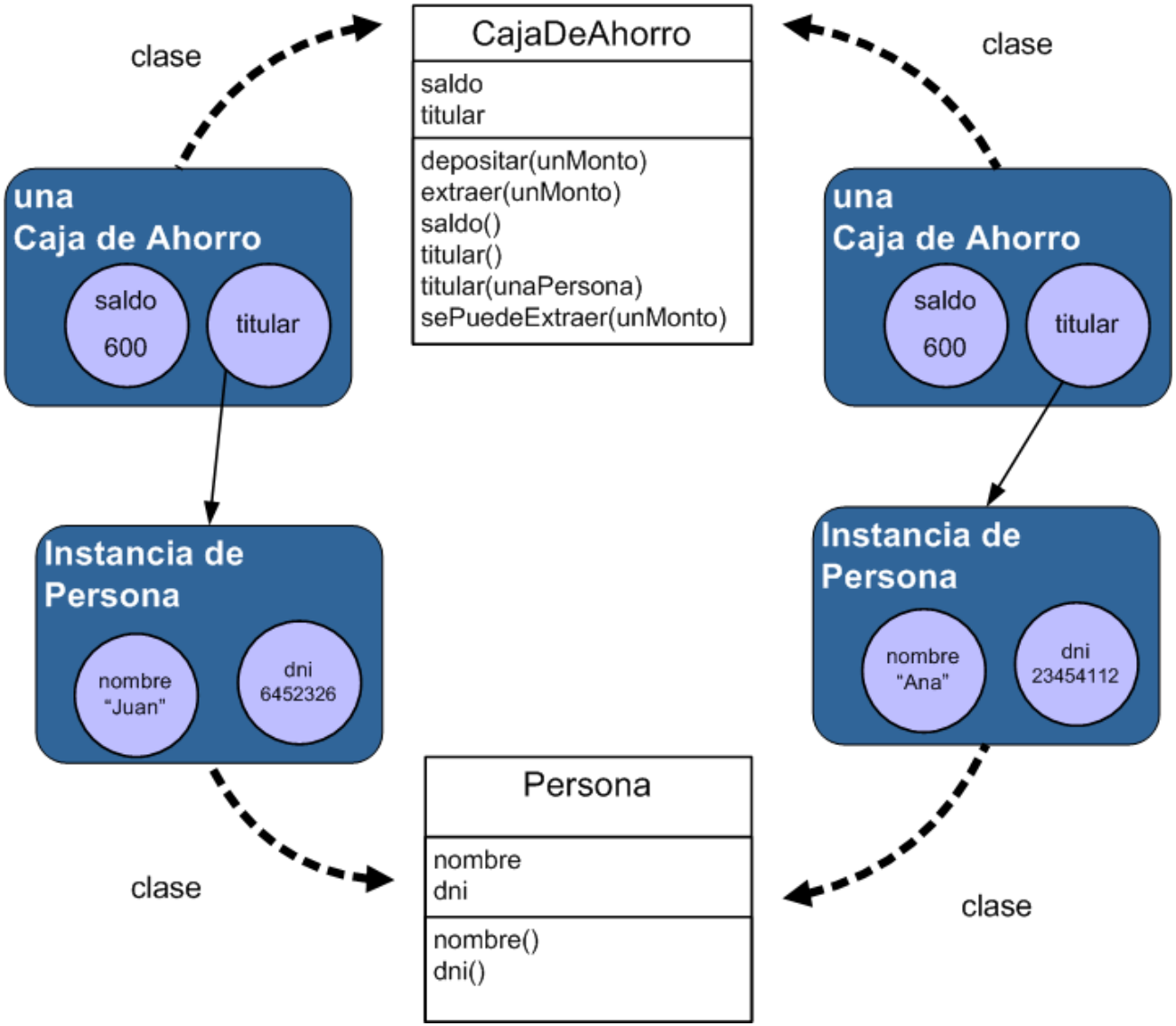
Comportamiento común entre Objetos

- 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 e instancias

- Una clase es una descripción abstracta de un conjunto de objetos.
- 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.

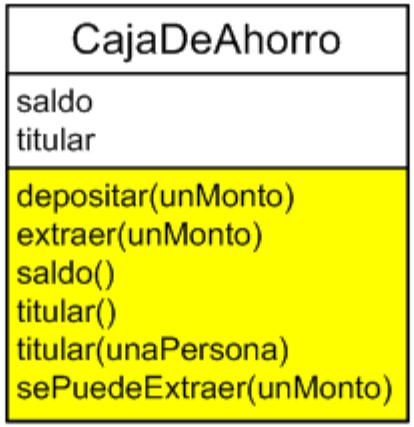
Ejemplo de clases e instancias



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:

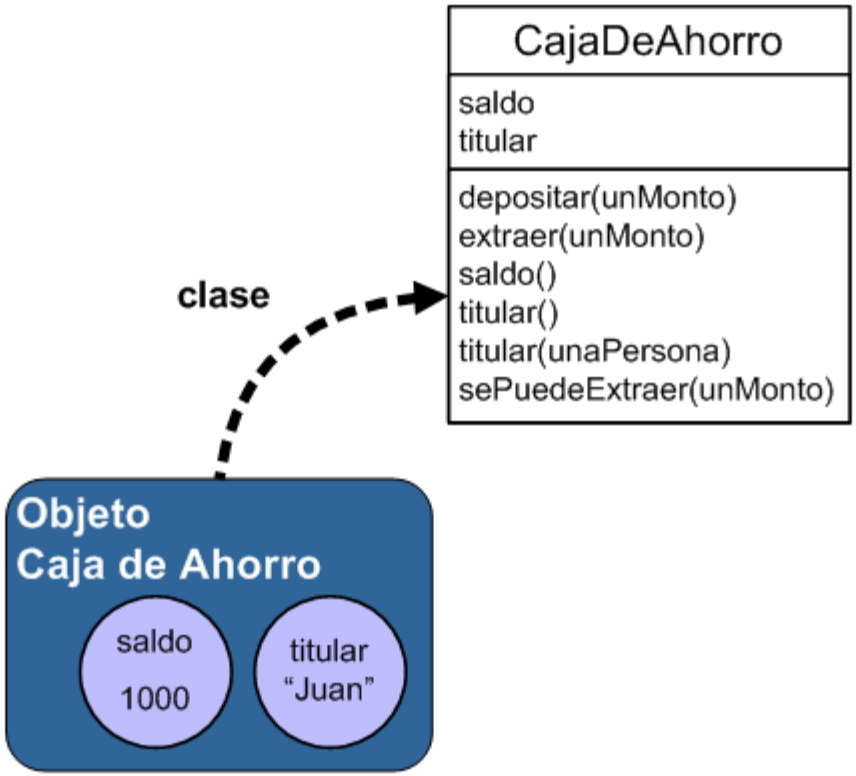
Variables de Instancia
Los nombre de las v.i. se escriben en minúsculas y sin espacios



Nombre de la Clase
Comienzan con mayúscula y no posee espacios

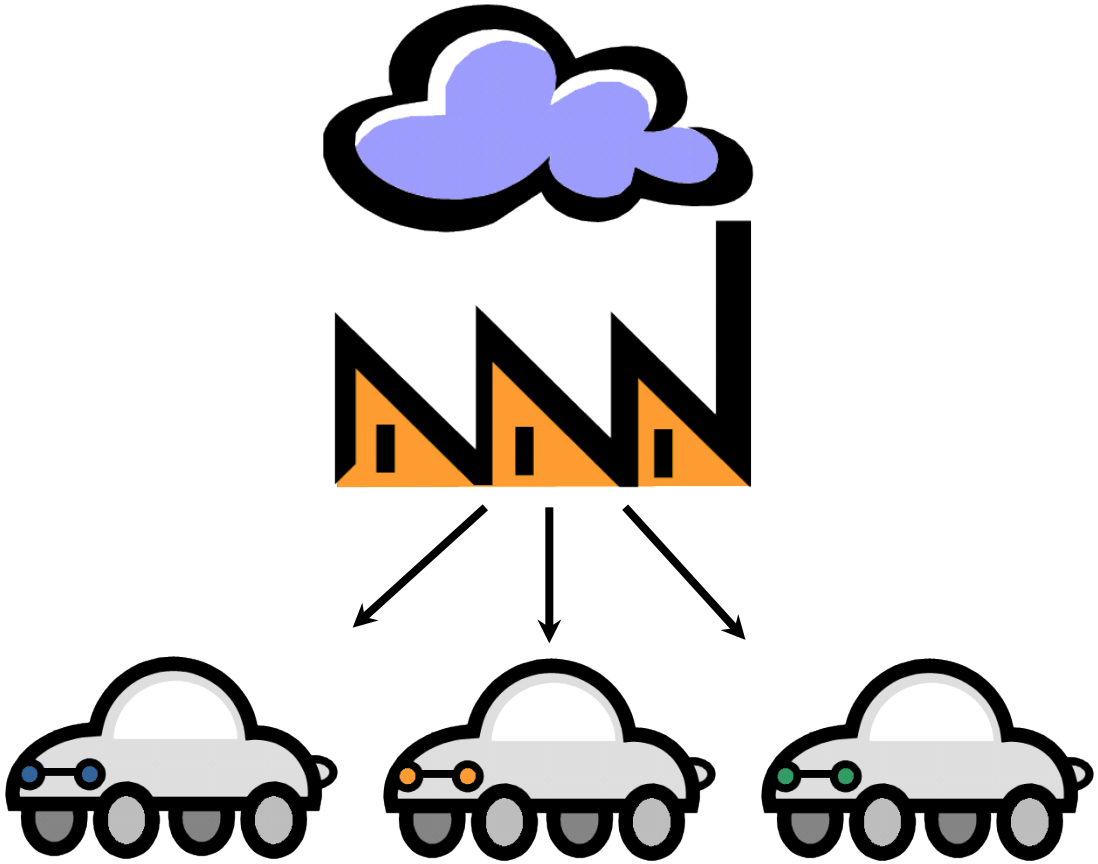
Protocolo
Para cada mensaje se debe especificar como mínimo el nombre y los parámetros que recibe

Envío de mensajes con clases I



Creación de Objetos

- ¿Cómo creamos nuevos objetos?
- Instanciación



Creación de Objetos

- 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.

- Nosotros vamos a usar al new como mensaje de clase. En este caso, la ejecución del método retorna una nueva instancia de la clase a la que se le envía el mensaje.

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) de la misma manera (los mismos métodos).

Una instancia recién creada …

- ¿está lista para poder colaborar con otros objetos?
  - Pensemos en la creación de un objeto *fecha*.
  - ¿Podemos sumar un punto recién creado?
  - Una cuenta bancaria recién creada ¿sabe quién es su titular?

Inicialización

- Para que un objeto esté listo para llevar a cabo sus responsabilidades hace falta *inicializarlo*.
- Inicializar un objeto significa darle valor a sus variables.
- ¿De dónde sacamos esos valores iniciales?

Clases y subclases

- Una clase representa un concepto en el dominio de problema.

- ¿Qué sucede cuando las clases tienen comportamiento común?

→ **Subclasificación**

Ejemplo de cuentas bancarias

- 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 cuentas bancarias

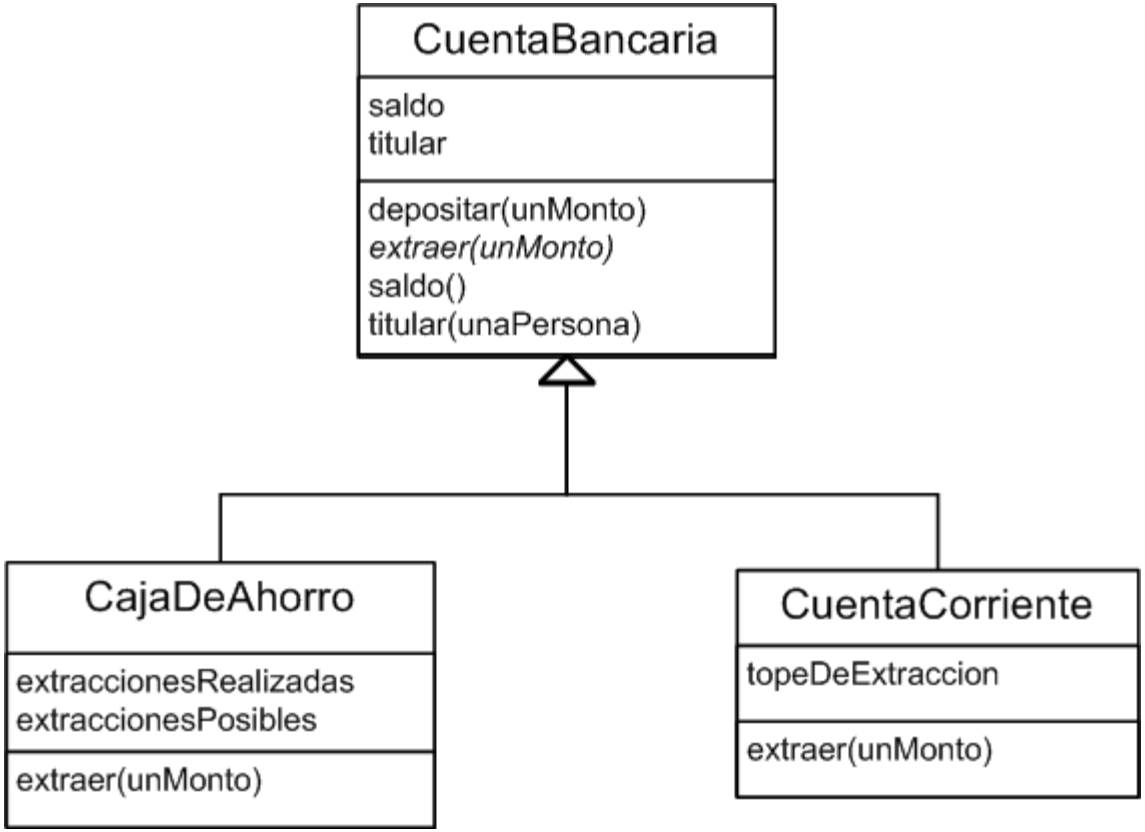
- Pero cada una tiene distintas restricciones en cuanto a las extracciones:
  - Cuentas corrientes: permiten que el cliente extraiga en descubierto (con un tope pactado con cada cliente).
  - Cajas de ahorro: poseen una cantidad máxima de extracciones mensuales (para todos los clientes). No se permite extraer en descubierto.

- ¿Cómo podemos reutilizar las características en común?

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.
- Debe cumplir la relación ***es-un***.

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.

- Verdadero o falso?
  - Una **CajaDeAhorro** *es-una* **CuentaBancaria**
  - Una **CuentaBancaria** *es-un* **Banco**
  - Un **TitularDeCuenta** *es-un* **EmpleadoDelBanco**

Herencia

- Es el mecanismo por el cual las subclases reutilizan el comportamiento y estructura de su superclase.

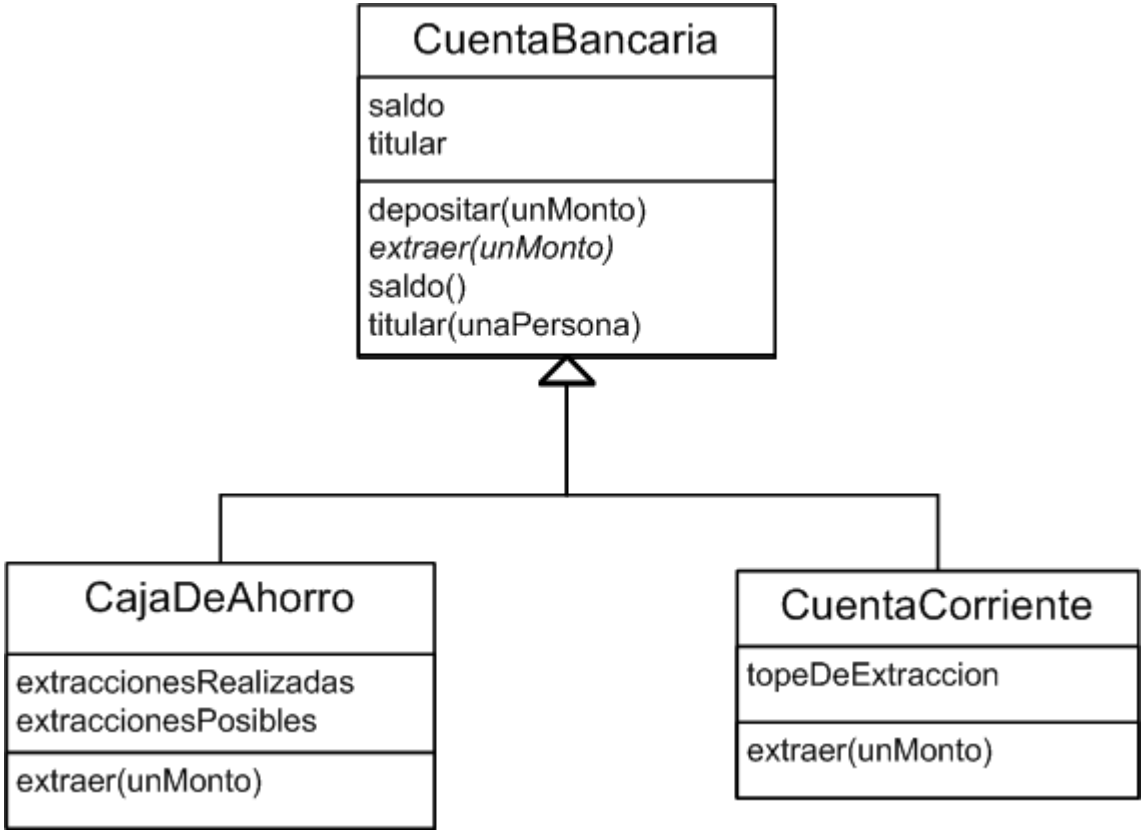
- La herencia permite:
  - Crear una nueva clase como refinamiento de otra.
  - Diseñar e implementar sólo la diferencia que presenta la nueva clase.
  - Factorizar similitudes entre clases.

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

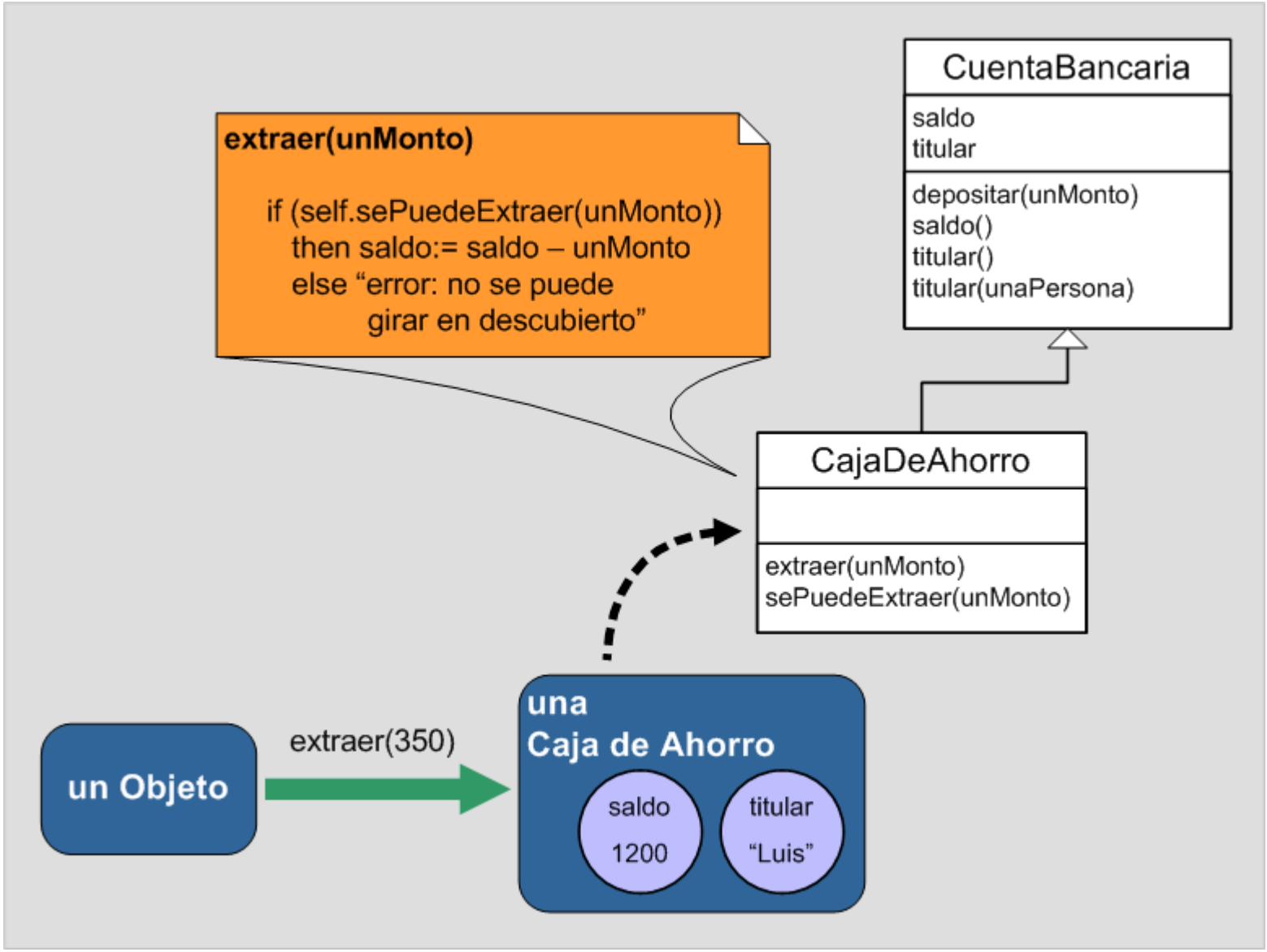
- Recordemos las restricciones:
  - Cuentas corrientes: permiten que el cliente extraiga en descubierto (con un tope pactado con cada cliente).
  - Cajas de ahorro: poseen una cantidad máxima de extracciones mensuales (para todos los clientes). No se permite extraer en descubierto.

Method Lookup

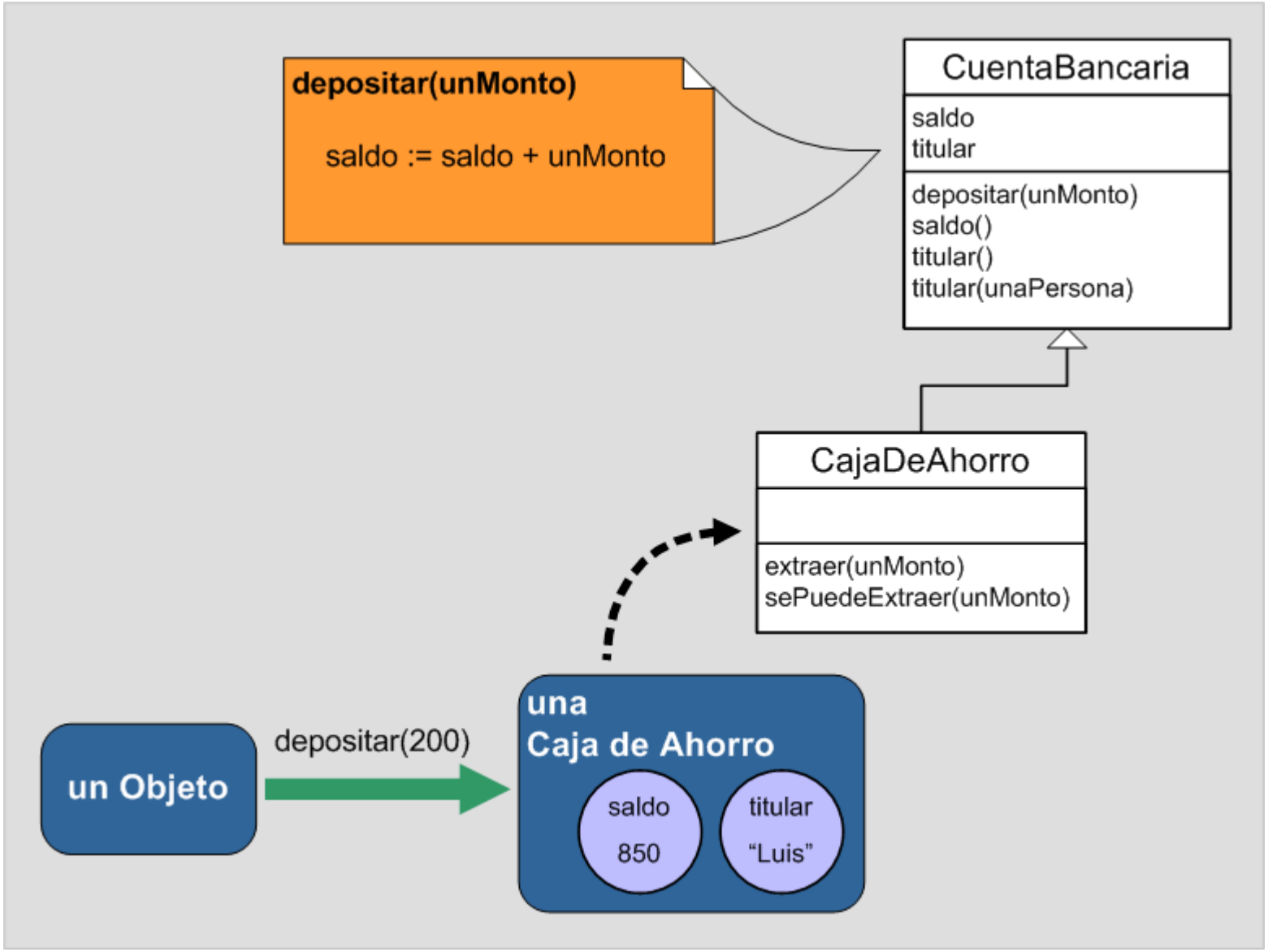
- Al enviarse un mensaje a un objeto:
  - Se determina cuál es la clase del objeto.
  - Se busca el método para responder al envío del mensaje en la jerarquía, comenzando por la clase del objeto, y subiendo por las superclases hasta llegar a la clase raíz (Object)

- Este proceso se denomina ***method lookup***

Method Lookup



Method Lookup



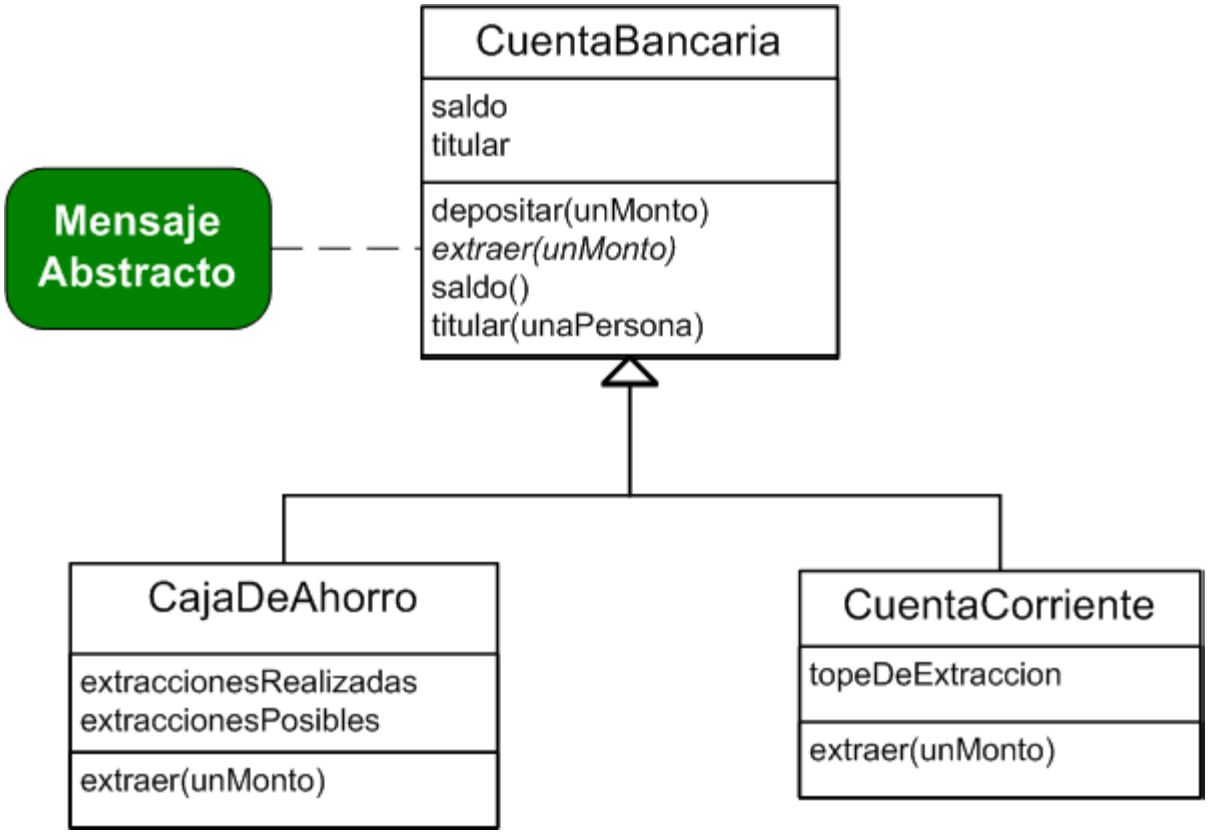
Métodos abstractos

- Son importantes para establecer el protocolo de una jerarquía de clases
- No se especifica el comportamiento, ya que a nivel de la superclase no se puede prever.
- En los diagramas de clase UML los métodos abstractos se escriben en *letra cursiva*.

Métodos abstractos

- Las subclases concretas deben tener implementaciones de los métodos abstractos, ya sea dentro de la clase o en alguna de sus superclases.

Mensajes abstractos



Clases Abstractas

- Una **clase abstracta** es una clase que no puede ser instanciada.

- ¿Entonces, para qué sirven?
  - Se diseña sólo como clase padre de la cual derivan subclases.
  - Representan conceptos o entidades abstractas.
  - Sirven para factorizar comportamiento común.
  - Usualmente, tiene partes incompletas.
    - Las subclases completan las piezas faltantes, o agregan variaciones a las partes existentes.

Relaciones entre herencia y encapsulamiento

- Toda subclase hereda estructura y comportamiento.

- Entonces puede acceder a:
  - Las variables de instancia.
  - Los métodos.

  definidos en la superclase.

Polimorfismo

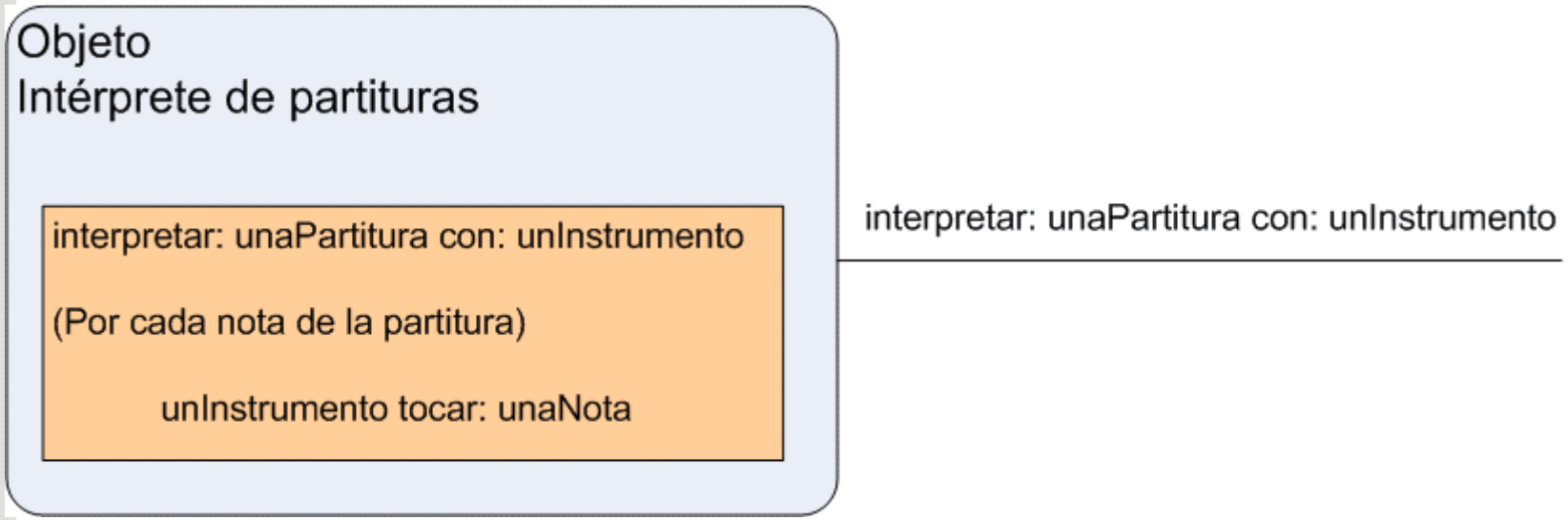
- Dos o más objetos son ***polimórficos*** con respecto a un mensaje, si todos pueden entender ese mensaje, aún cuando cada uno lo haga de un modo diferente

  - *Mismo mensaje puede ser enviado a diferentes objetos*
  - *Distintos receptores reaccionan diferente (diferentes métodos)*

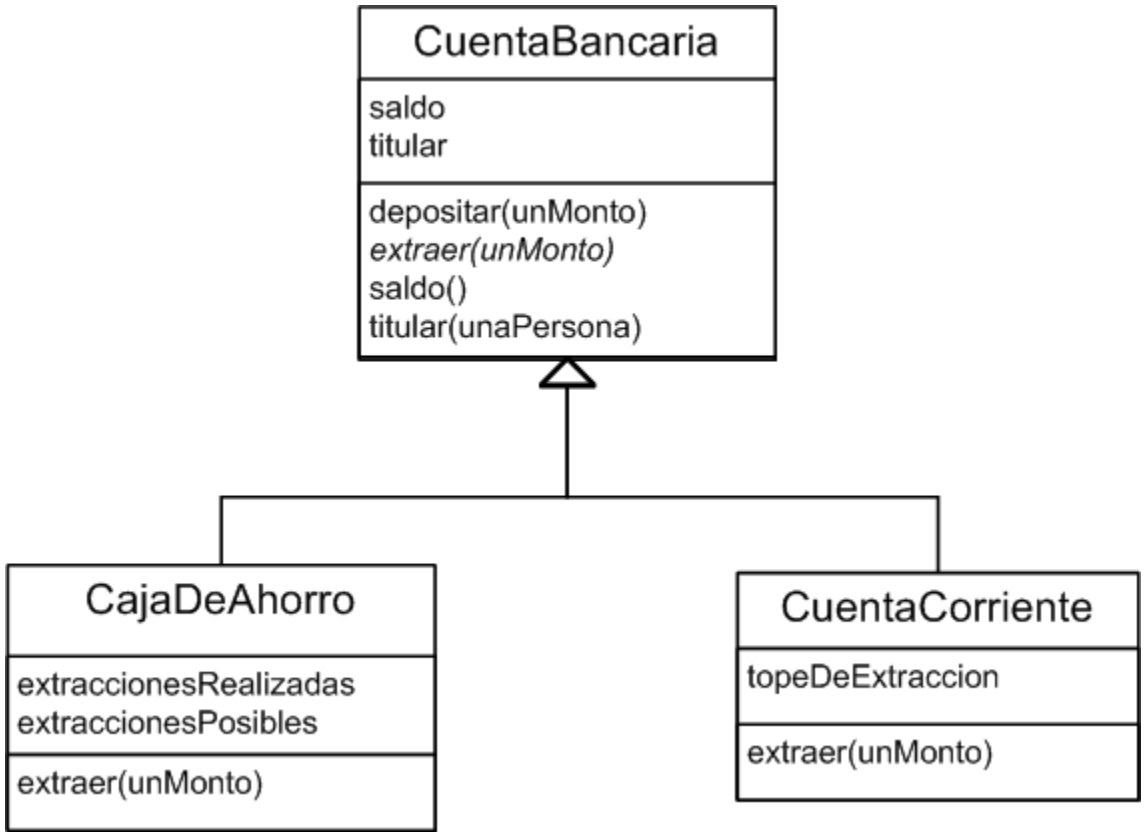
Ejemplo de objetos polimórficos

Polimorfismo

- Ejemplo:



Ejemplo: Cuentas Bancarias



Ventajas del uso del polimorfismo

- *Código* genérico
- Objetos desacoplados
- Objetos intercambiables
- Objetos reutilizables
- Programar por protocolo, no por implementación

Polimorfismo + Binding Dinamico

- Que es binding? Asociacion de operadores y operandos
- Ej: X= A + B….. Que significa?
- Ej (Procedural): Dibujar (X) Que significa?
- Ej: (OO): x dibujar…..Que significa?

Cuando se hace el binding?
  Procedural?
  OO?

El problema del “if”

- El **if** es a la Programación Orientada a Objetos lo que el **goto** era para la Programación Estructurada
  - Los lenguajes OO proveen **if** pero...
  - **El mal uso de If** significa la falta de objetos polimórficos
    - Las decisiones son tomadas por los programadores
    - Mala asignación de responsabilidades
- ¿Cuándo se justifica su uso?

¿Qué pasaría sin polimorfismo?

- Veamos el siguiente método.

```
if (unaCuenta.esCajaDeAhorro())
        then unaCuenta.extraerDeCajaDeAhorro(100)
        else unaCuenta.extraerDeCuentaCorriente(100)
```

- ¿Qué problema presenta?
- ¿Qué sucede si aparece un nuevo tipo de cuenta, por ejemplo CuentaUniversitaria?

Polimorfismo

- Hay que modificar el código:

```
if (unaCuenta.esCajaDeAhorro())
   then unaCuenta.extraerDeCajaDeAhorro(100);
   else if (unaCuenta.esCuentaCorriente())
      then unaCuenta.extraerDeCuentaCorriente(100);
      else unaCuenta.extraerDeCuentaUniversitaria(100);
```

- Si delegamos la lógica en cada cuenta, el código del método sería:

```
unaCuenta.extraer(100);
```

Ejercicio - Figuras

- Supongamos que tenemos que diseñar un editor gráfico para figuras en dos dimensiones
- Las figuras pueden ser
  - Rectángulos
  - Triángulos
  - Círculos
- El editor debe poder dibujar las figuras, pero naturalmente dibujará cada figura de distinta manera

El problema del “if”

- Analicemos el siguiente método del editor: Dibujar todas las figuras definidas

```
For i= 1 to N
    Si (figuras[i] es rectanculo) entonces
            dibujarCuadrado.
    Si (figuras [i] es circulo) entonces
            dibujarCirculo.
    Si (figuras [i] es triangulo) entonces
            dibujarTriangulo.
```

For i= 1 to N

Figuras [i] dibujar

Ventajas?

- Por que es diferente?
- Por que es mejor?