

# Recordar

- Más allá de que en el código de nuestra aplicación (e.g. en Java) haya un “cuerpo” de código semejante al “main” procedural incluso con ese nombre, no existe el equivalente “estructural” al main que “conduce” el flujo en cada ejecución
- Los objetos se parecen a “registros” PERO:
  - Están encapsulados
  - Contienen comportamiento (que cosas se les puede pedir)
  - Conocen a otros objetos mediante referencias y no mediante datos
- Ejemplo: en un Objeto Alumno, la carrera que cursa NO ES el nombre de la carrera, es un “puntero” al objeto carrera.

- Una clase representa un concepto en el dominio de problema (o de la solución).
- ¿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?

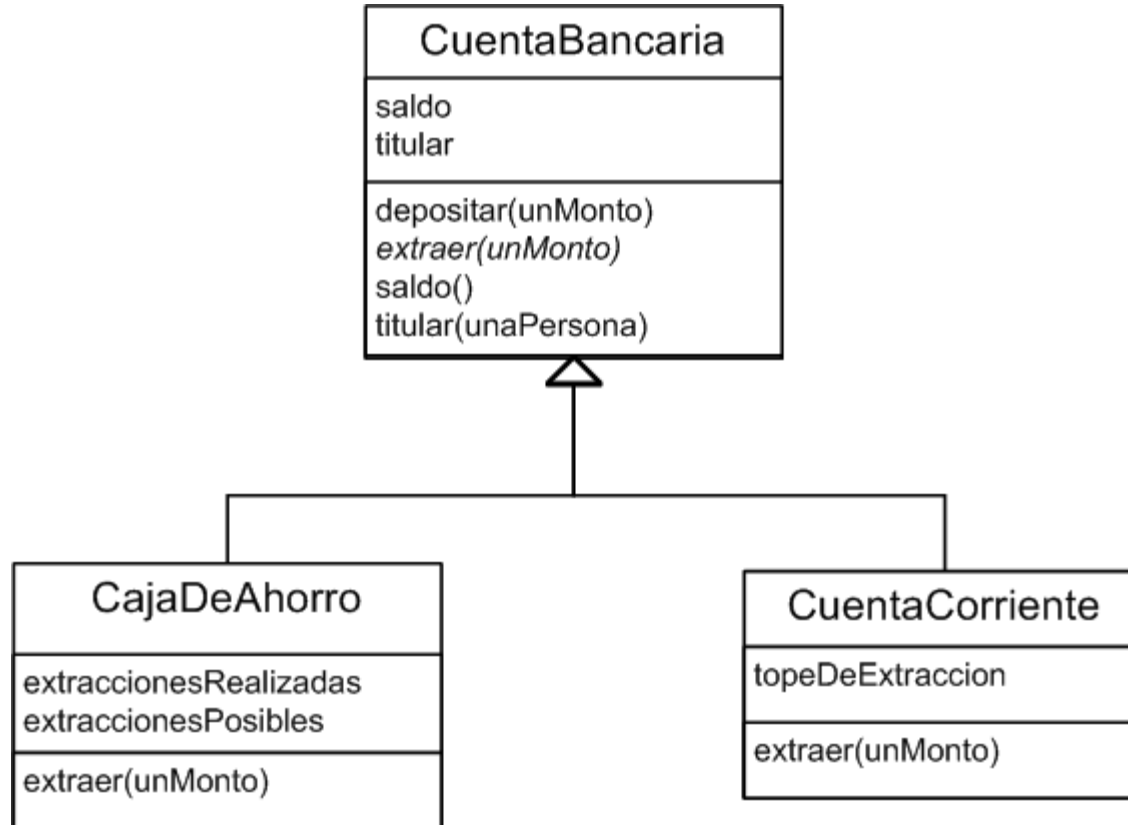
# Subclasificación

- 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 (y a veces algo mas)*.
- Es facil encontrar jerarquias? Cuando las encontramos?

# Para que buscar jerarquias?

- Entender mejor el dominio
- Reducir algo de código y descripciones redundantes
- Aprovechar mejor el polimorfismo

# Ejemplo de una Jerarquía de Clases



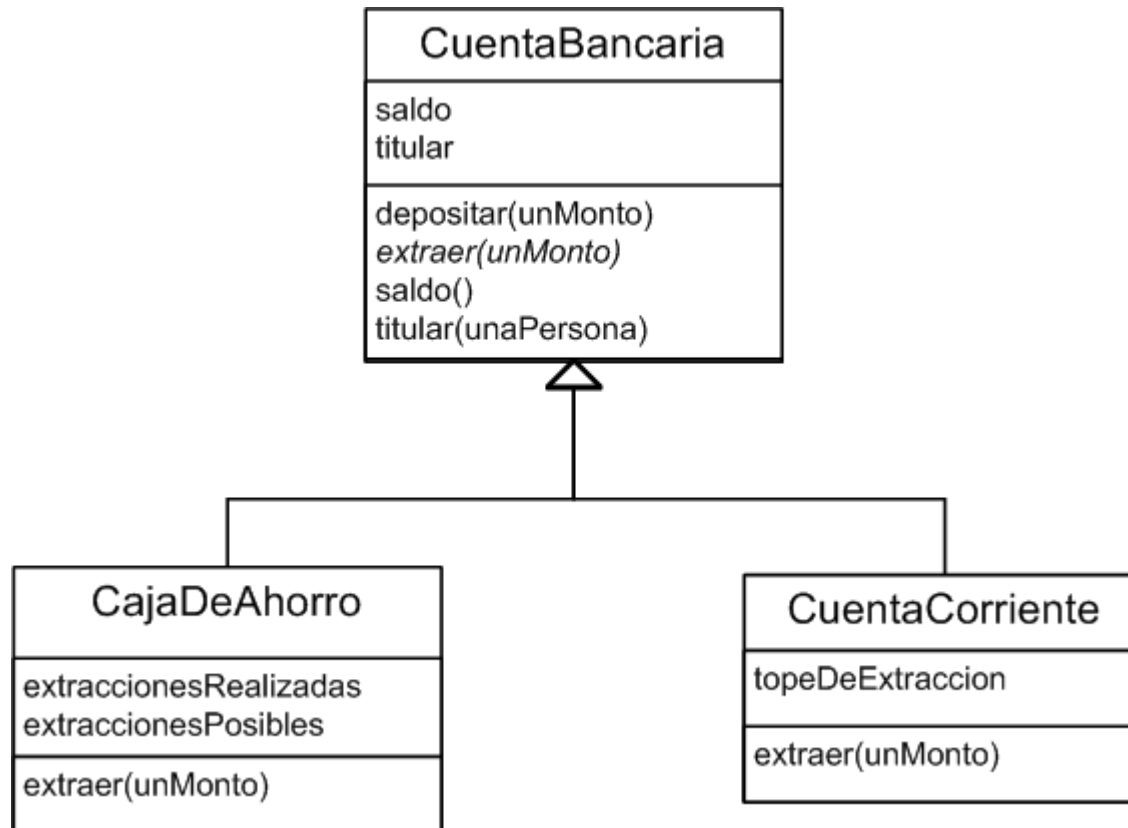
- Es el mecanismo por el cual las subclases reutilizan el comportamiento y estructura de su superclase.
- La herencia permite:
  - Crear una nueva clase como refinamiento de otra.
  - Diseñar e implementar sólo la diferencia que presenta la nueva clase.
  - Factorizar similitudes entre clases.



- 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.

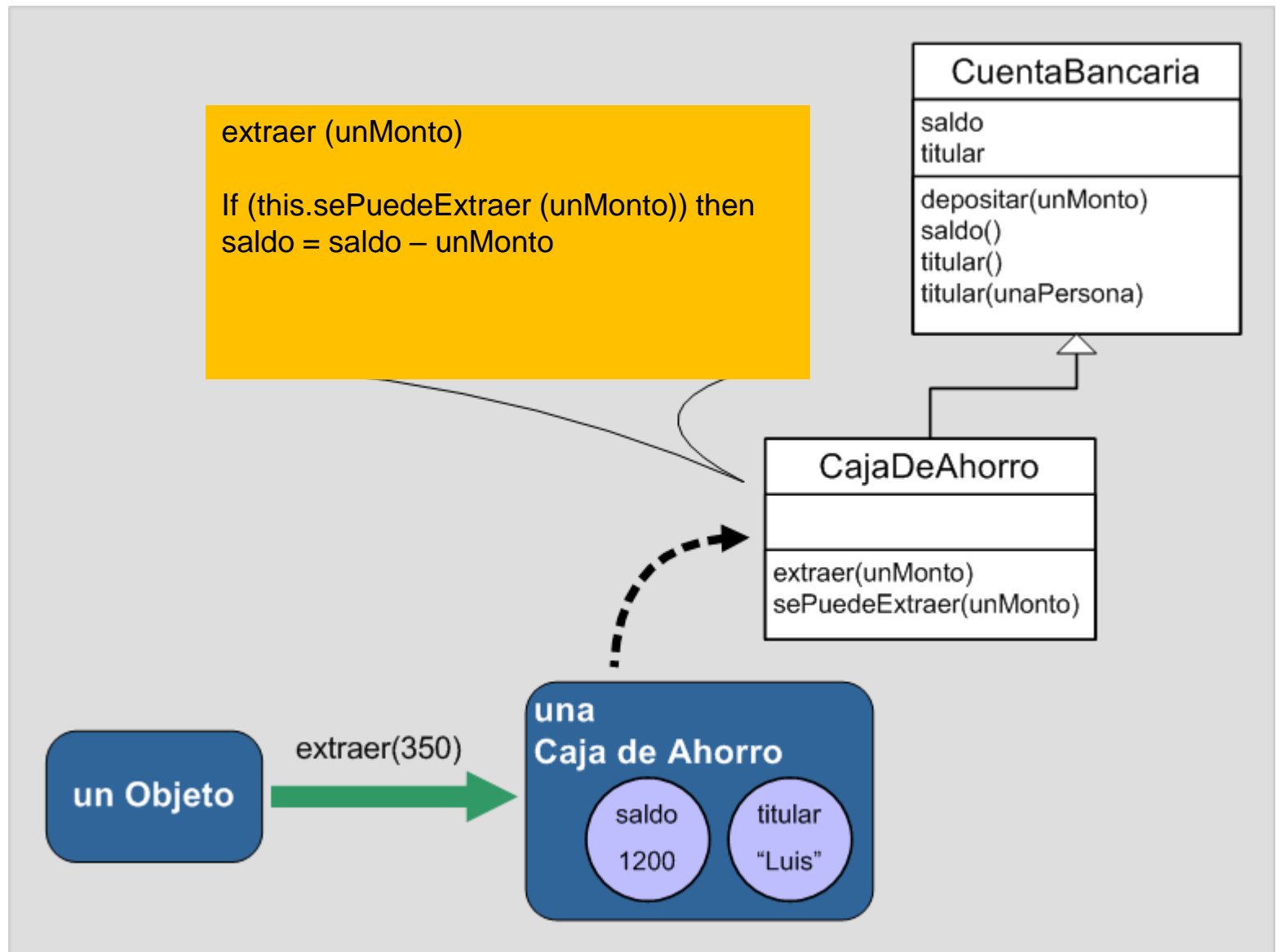
# Ejercicio - Cuenta Bancaria

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

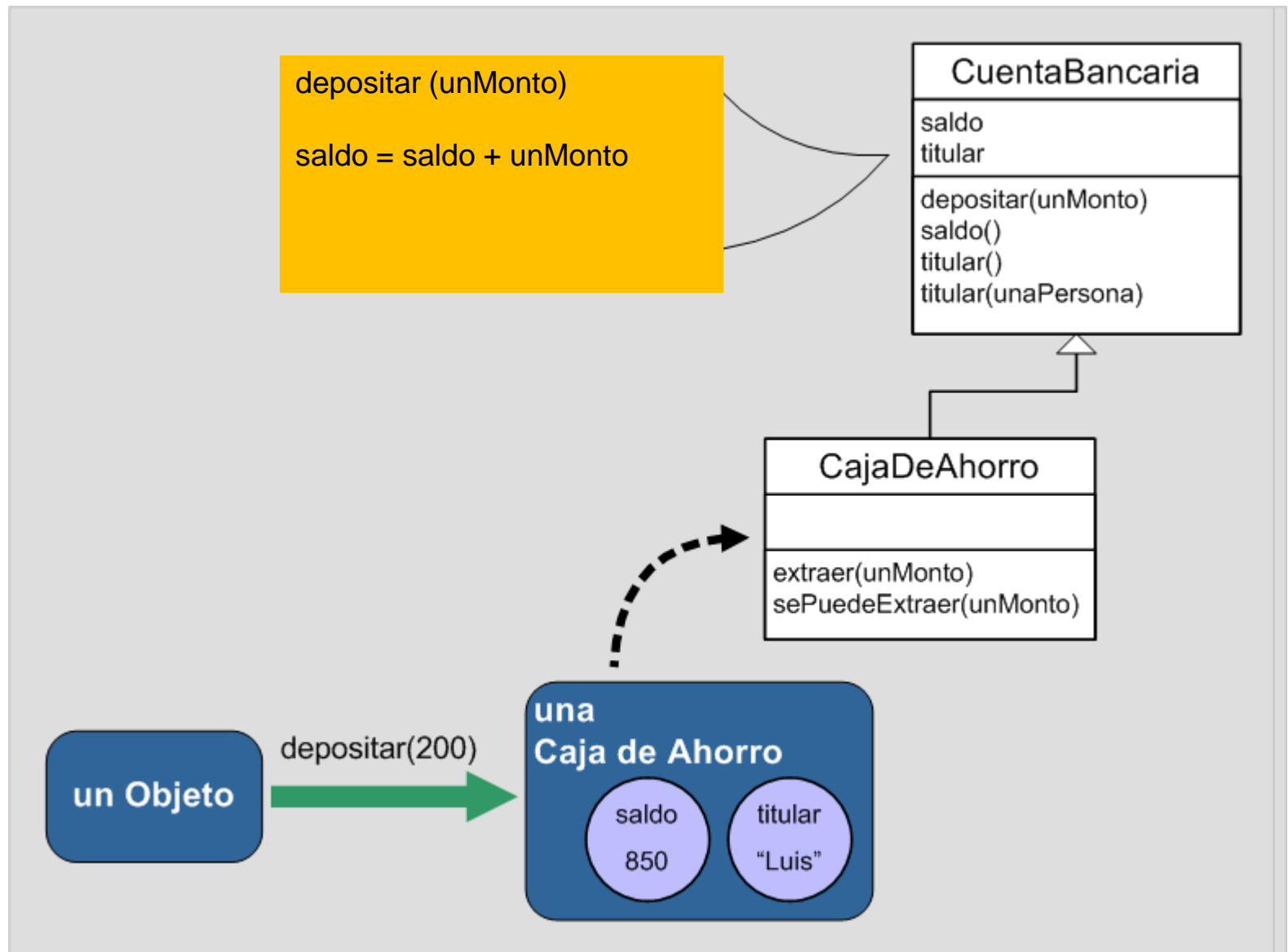


- Al enviarse un mensaje a un objeto:
  - Se determina cuál es la clase del objeto.
  - Se busca el método para responder al envío del mensaje en la jerarquía, comenzando por la clase del objeto, y subiendo por las superclases hasta llegar a la clase raíz (Object)
- Este proceso se denomina *method lookup*

# Method Lookup



# Method Lookup



- 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.

- 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)*
- El overloading es una forma particular de polimorfismo

# Ejemplo de objetos polimórficos

Objeto  
Piano

tocar (unaNota) \_



Objeto  
Guitarra

tocar (unaNota) \_



Objeto  
Flauta

tocar (unaNota) \_



- Ejemplo:

Objeto

Intérprete de partituras

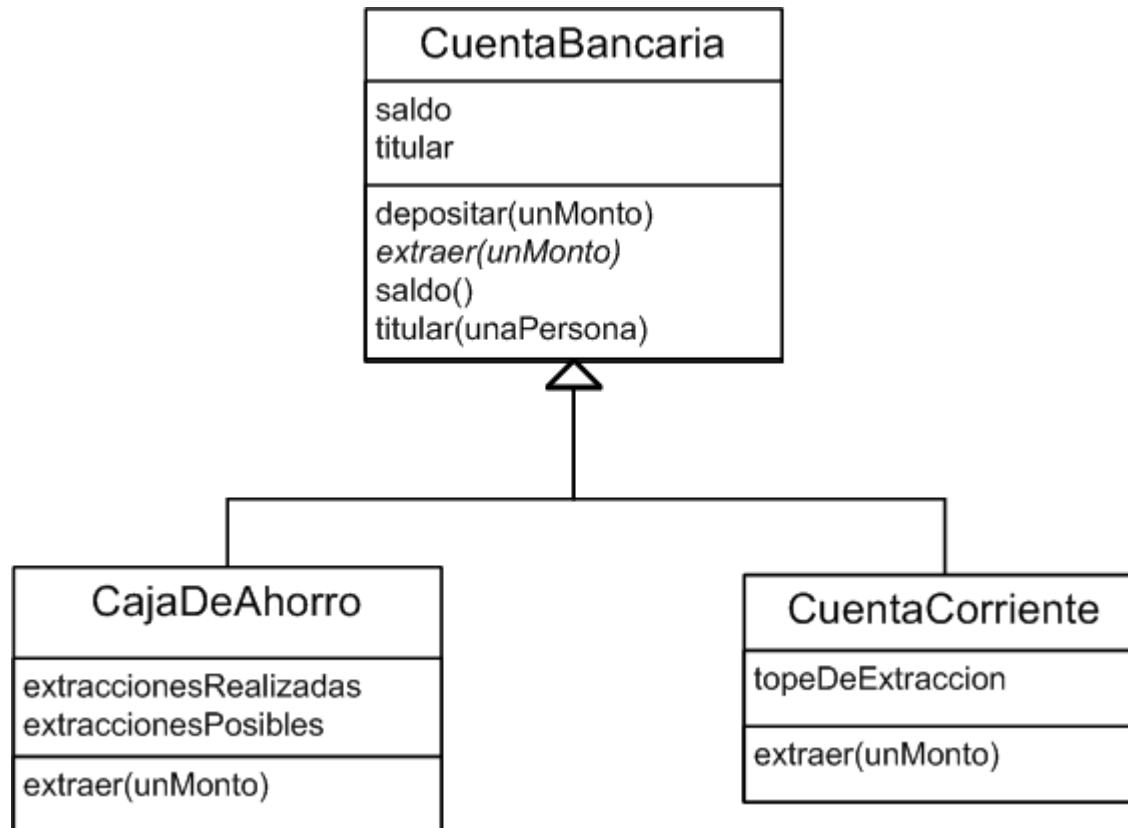
interpretar (unaPartitura, instrumento)

Para cada nota de unaPartitura

instrumento.tocar (nota)

interpretar (unaPartitura, instrumento)

# Ejemplo: Cuentas Bancarias



# Ventajas del uso del polimorfismo

- Código genérico
  - Objetos desacoplados
  - Objetos intercambiables
  - Objetos reutilizables
  - Programar por protocolo, no por implementación
- 
- Bien usado, implica menos código

# 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?
- Por que es diferente?

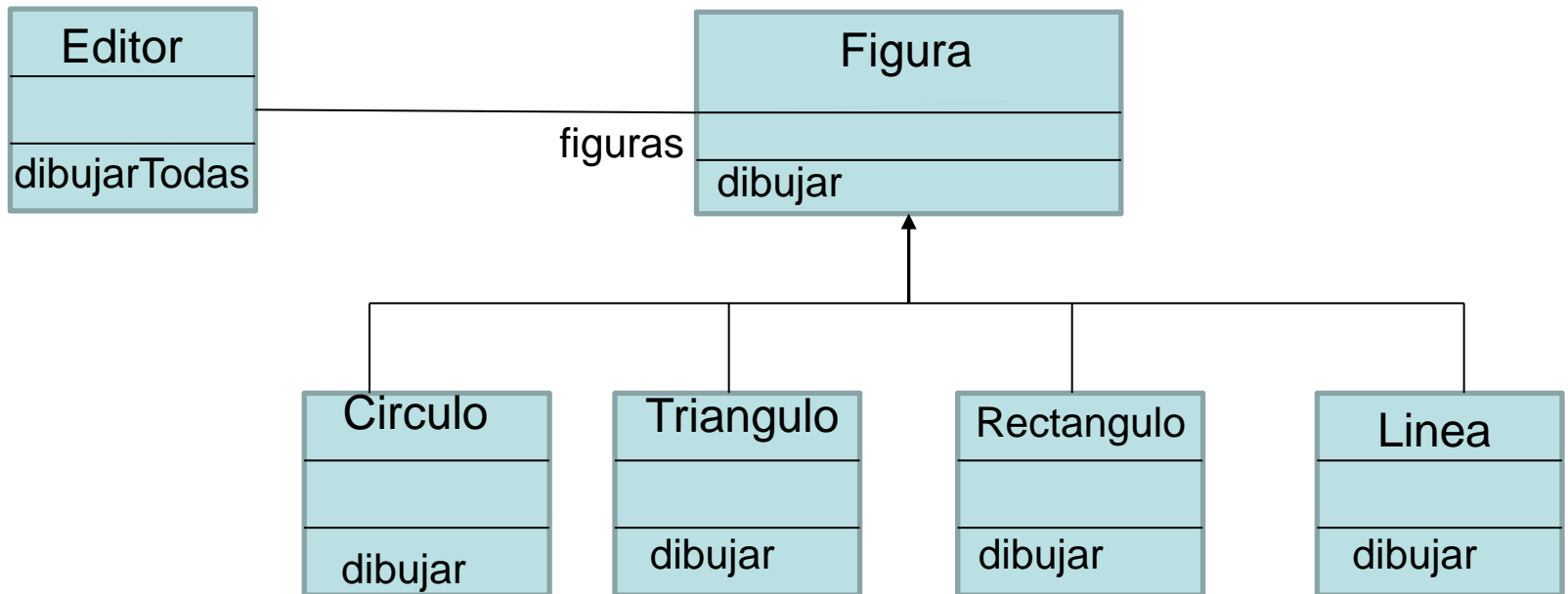
Cuando se hace el binding? Por que?

Procedural?

OO?

## 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



Observese que en la jerarquia no estamos “ahorrando” codigo....

Es interesante además pensar si necesitamos una jerarquía o una Interface (Java)

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

**For** *i* = 1 to *N*

*Si* (figuras[*i*] es rectangulo) entonces  
    dibujarrectangulo.

*Si* (figuras [*i*] es circulo) entonces  
    dibujarcirculo.

*Si* (figuras [*i*] es triangulo) entonces  
    dibujartriangulo.

For i= 1 to N  
figuras [i] dibujar

De que tipo es el contenido del Array?  
Observen el binding



# Ejemplo: Gloovo

Tu dirección de entrega (La Plata)  
La Plata  
Detalles: [Sin detalles](#)



Todas

Restaurantes <sup>ZZ</sup>

Farmacia <sup>ZZ</sup>

Mercados <sup>ZZ</sup>

Lo Que Sea <sup>ZZ</sup>

Recoger O Enviar <sup>ZZ</sup>

Regalos Y Más <sup>ZZ</sup>

Desayunos & Snacks <sup>ZZ</sup>

Kiosco <sup>ZZ</sup>

Buscar

## Pide lo que quieras de tu ciudad

¡Te lo traemos en minutos!



Restaurantes



Farmacia



Mercados



Lo que sea



Recoger o  
Enviar



Regalos y  
más



Desayunos &  
Snacks



Kiosco

Pst, ¿te gustan nuestras categorías? [¡Juega al minijuego!](#)

# Objetivos del Ejemplo

- Repasar los conceptos basicos sobre un diseño realista
- Vamos a analizar el diseño sin preocuparnos por COMO llegamos a el (por ahora).

Se trata de describir con objetos la funcionalidad mas importante de un sistema del tipo Rapi, Glovo o PedidosYa, que permite que **cualquier usuario registrado (cliente)** pida un **producto** en un **comercio registrado** y alguien (otros usuarios especiales que se postulan como **"Gloovers"**) le lleve el producto a la casa.

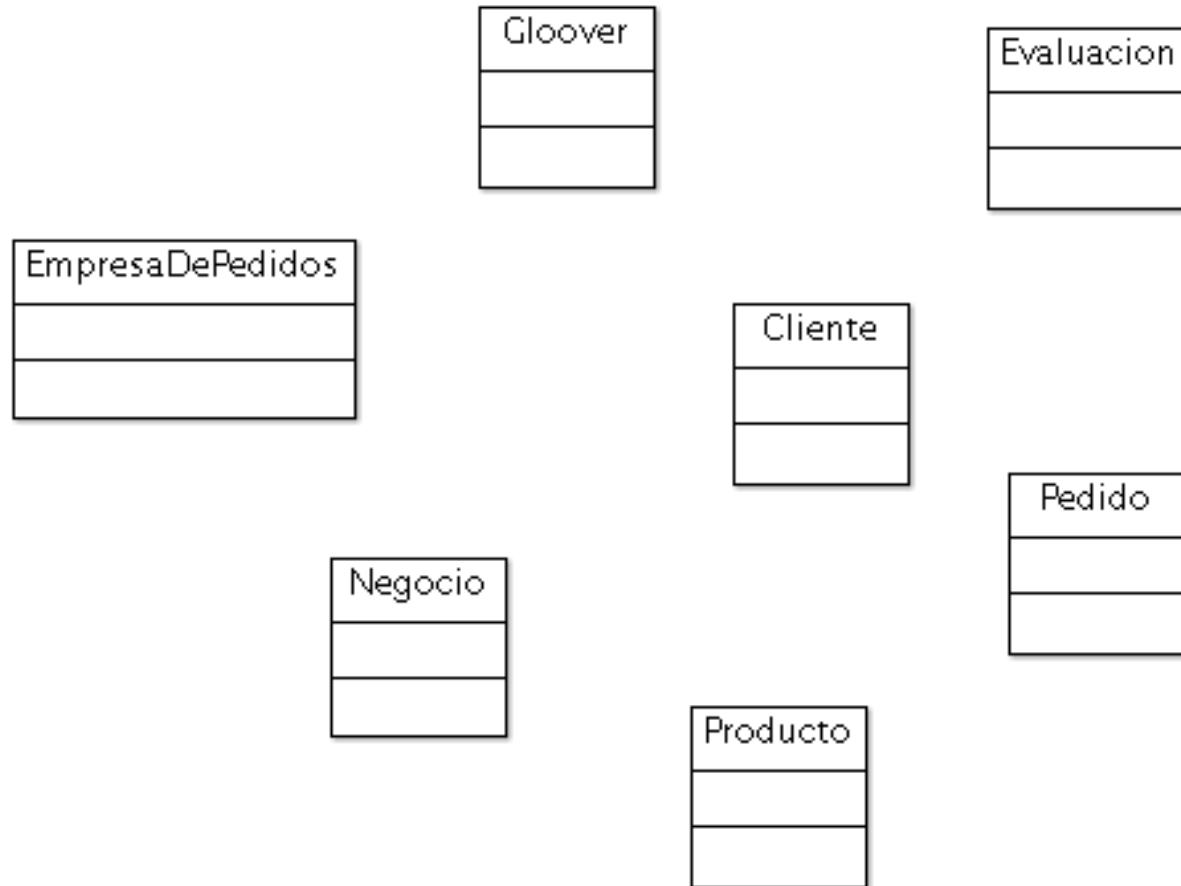
Con un conjunto de opciones desplegables el "cliente" elige producto (o producto y negocio en el que lo quiere comprar) y registra un pedido. En ese momento sabe el precio que pagará tanto por el producto como por el envío. El sistema todavía no puede estimar un tiempo de entrega. El "sistema" recibe el pedido y lo postea entre los usuarios "Gloovers". Cuando alguno de ellos lo "toma", el pedido se asigna al "acarreador" (un gloover) y se comunica al negocio que provee el producto para que lo prepare. El Gloover va al negocio y retira el producto. Utiliza la aplicación para confirmar que retiro el producto. Se dirige a la dirección de entrega y entrega el producto. Utiliza la aplicación para confirmar que lo entregó.

Cuando el cliente recibe el pedido, utiliza la aplicación para confirmar la recepción. En ese momento, se carga el costo a su medio de pago, se para al negocio, y se paga al "acarreador". Adicionalmente el cliente puede calificar al gloover con un puntaje y un comentario

# Casos de Uso del sistema Gloovo



# “Diseño” inicial



# Suposiciones iniciales

- Inicialmente no tenemos gloovers, ni clientes, ni negocios, ni productos.



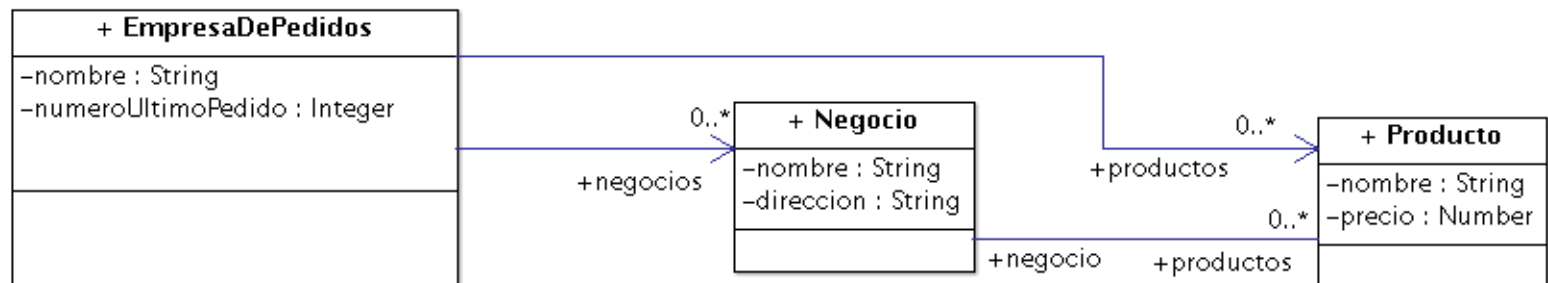
# Como entender/diseñar Gloovo/ Pseudo codigo Java

-El sistema “arranca” con la creacion de los objetos de interfaz apropiados y un objeto de la clase EmpresaDePedidos (llamemosle glovo).

**-Para simplificar asumimos que todos los eventos generados por la interfaz son atendidos por este objeto, aunque un diseño mas realista podria incluir otros objetos (para hacer de “fachada” del sistema) para que la EmpresaDePedidos no se transforme en monolítica**

-Asumimos tambien que los negocios y productos se agregaron al sistema (quizas con interfaces adaptadas a los dueños de los negocios

**(Observar la notacion)**



# Creacion del objeto gloovo

gloovo= new EmpresaDePedidos (“Gloovo S.A”)

-Esto nos crea un objeto gloovo cuya variable de instancia nombre sera “Gloovo S.A”

El metodo constructor contendrá

productos= new SomeCollection <Producto>

negocios:= new SomeCollection <>

clientes:= new SomeCollection <>

gloovers:= new SomeCollection <>

# Agregar un Negocio

- En Clase Empresa de Pedidos

*agregarNegocio (nombre, direccion)*

negocio=new Negocio (nombre,direccion)

negocios.add (negocio)



- El constructor de la Clase Negocio

this.nombre = nombre

this.direccion = direccion

productos= new SomeCollection <Producto>

# Agregar un Producto a Gloovo

- Este proceso depende de nuestro esquema de negocios.
- En Empresa de Pedidos

*agregarProducto (nombre,precio,negocio)*

```
produ = new Producto (nombre, precio, negocio)  
productos.add (produ)  
negocio.agregarProducto (produ)
```

# Agregar un producto a un Negocio

- En Clase Negocio

*agregarProducto (p)*  
productos.add (p)

# Es realista?

- Da lo mismo agregar los productos de la pizzeria del barrio que los productos de Garbarino?
- Problemas?: Cantidad de productos, Quien lo hace?
- Opciones?: Hacerlo “por programa”

- **Registrarse como Cliente**

- El usuario interactua con un formulario, llena ciertos campos y envia el formulario que es recibido por el objeto gloovo
- gloovo (instancia de EmpresaDePedidos) ejecuta el metodo de dicha clase que debe:
  - Crear el objeto Cliente
  - Agregarlo a la “coleccion” de clientes del Negocio

***registrarCliente (nombre, direccion, email, password)***

c= new Cliente (nombre, direccion, email, password)

clients.add (c)

**VER CONSTRUCTOR EN Clase Cliente**

**DONDE CHEQUEAMOS CONSISTENCIA DE ESTA INFORMACION?**

# Temas de discusion en este metodo

- Que es “direccion”
  - String? “Calle 9 Nro 1354 La Plata Buenos Aires”
  - Un objeto de otra clase?
- Una suposicion razonable es que la interfaz crea objetos de clase “Direccion” con atributos estandar (calle, numero, ciudad, codigo postal, provincia, pais) y que el Cliente tiene una variable de instancia “direccion” que es una referencia a dicho objeto

- **Registrarse como Gloover**

- Este proceso puede ser mas “largo” y requerir chequeos no atómicos por seguridad. El candidato llena un formulario y lo submite. El objeto gloovo (en realidad el objeto referenciado por la variable gloovo) recibe el mensaje y el metodo correspondiente deberia:
  - Pedir validar los datos basicos del Gloover
  - Disparar un proceso de validacion mas elaborado (offline) y luego el responsable (con otro form) disparara el metodo agregarGloover

***registrarGloover (nombre, direccion, email,.....)***

*g new Gloover ( , , , , )*

*this.chequearDatos (g)*

*.....En otro momento, y disparado desde otra interfaz...*

***aceptarGloover (g)***

*gloovers.add (g)*

# Temas en este metodo y otros

- Notificacion de los gloovers? Donde? Cuando?
- Validacion “profunda”: otro sistema? El mismo que se comunica con otros?



# Diseño hasta ahora

