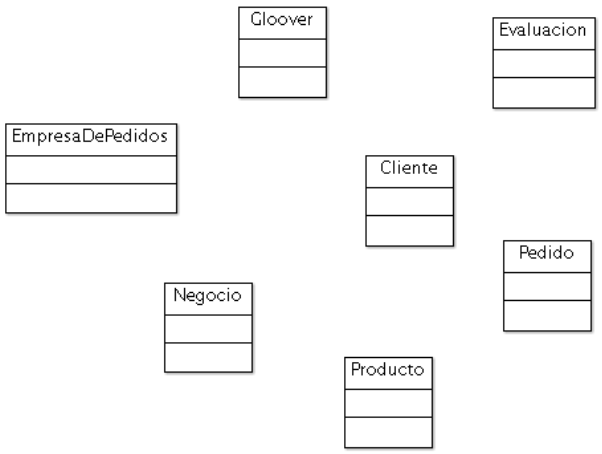
## Contexto del Ejemplo y restricciones

- No vamos a preocuparnos (ahora) por el diseño de las interfases o los formularios.
- No vamos a preocuparnos por la comunicacion entre los usuarios (desde telefonos, tabletas, computadoras, etc) y la aplicacion
- Vamos a ignorar el "almacenamiento" de los datos (en archivos, bases de datos, centralizado, distribuido, etc)

- Sin embargo podemos decir:
  - Es correcto asumir que esos aspectos no implicaran cambio alguno en el sistema (nuevas interfases por ejemplo, nuevos dispositivos, etc)
  - La "comunicacion" es transparente al software (casi siempre)

Lifia

## "Diseño" inicial




Lifia

## Casos de Uso del sistema Gloovo


Lifia

## Suposiciones iniciales

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


Lifia

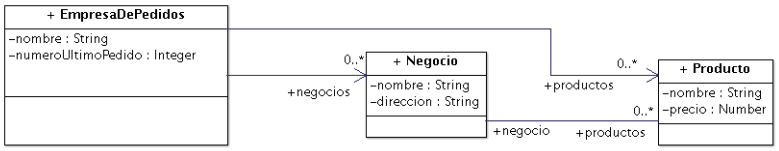
## Como entender/diseñar Gloovo

-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)**

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

(**Observar la notacion**)




Lifia

## Creacion del objeto gloovo

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

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

El metodo

*initialize (label)*
nombre:=label
productos:=Collection new
negocios:= Collection new
clientes:= Collection new
gloovers:= Collection new

Lifia

## Algo mas modular y preparado para cambios

```
initialize (label)
        nombre:=label
        self initializeCollections

initializeCollections
        productos:=SomeCollection new
        negocios:= SomeCollection new
        clientes:= SomeCollection new
        gloovers:= SomeCollection new
```

(QUE PEDIREMOS DE ESTAS COLECCIONES?)

Lifia

## Agregar un Negocio

- En Clase Empresa de Pedidos

```
agregarNegocio (n, d)
 negocio:=Negocio new
 negocio inicializar (n,d)
 negocios add (n)
```

Lifia

## Inicializar Negocio

- En Clase Negocio

```
inicializar (n, d)
 nombre:=n
 direccion:= d
 productos:= Collection new
```



## Agregar un Producto a Gloovo

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

```
agregarProducto (n,p, negocio)
   produ:= Producto new
   produ inicializar (n,p,negocio)
   productos add (produ)
   negocio agregarProducto (produ)
```

## Agregar un producto a un Negocio

- En Clase Negocio

```
agregarProducto (p)
 productos add (p)
```

Lifia

## 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”

Lifia

## Agregando Productos de Garbarino

- Creamos un objeto BotGarbarino, le pedimos que busque productos y se los pase a Garbarino

b:=BotGarbarino new

b buscarProductosPara(n).....quien es n? de donde lo sacamos?

En la Clase BotGarbarino

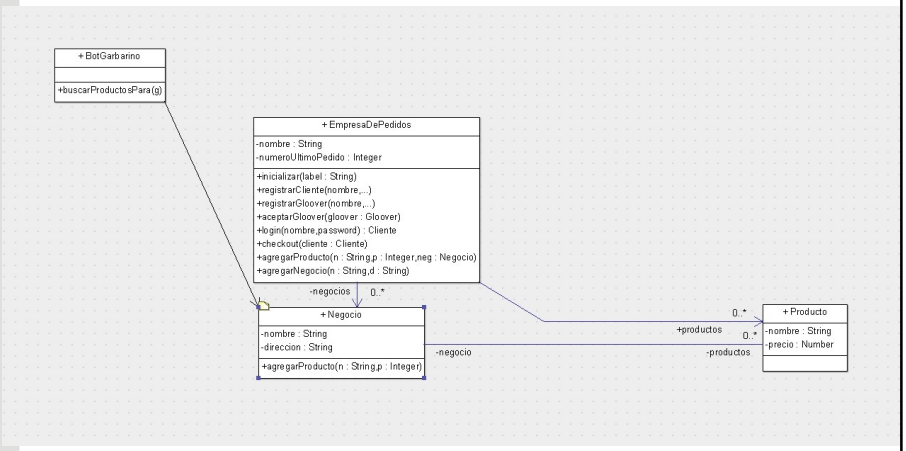
buscarProductosPara(n)

*codigo especifico para obtener productos de Garbarino*

n agregarProducto() de a 1 o todos juntos? Quien crea productos?

Lifia

## Bot




Lifia

## Implementando casos de uso del sistema

- **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:= Cliente new
c inicializar (nombre, direccion, email, password)
clientes add (c)

**DONDE CHEQUEAMOS CONSISTENCIA DE ESTA INFORMACION?**

Lifia

## Clase Cliente-Inicializacion

Inicializar (nomb, dir, mail, pass)

nombre:= nomb
direccion:= dir
email:= mail
password:= pass

Lifia

## 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

Lifia

## Otros casos de Uso

- **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:= Gloover new

g inicializar (…..)

self chequearDatos (g)

……En otro momento, y disparado desde otra interfaz…

*aceptarGloover (g)*

gloovers add: g

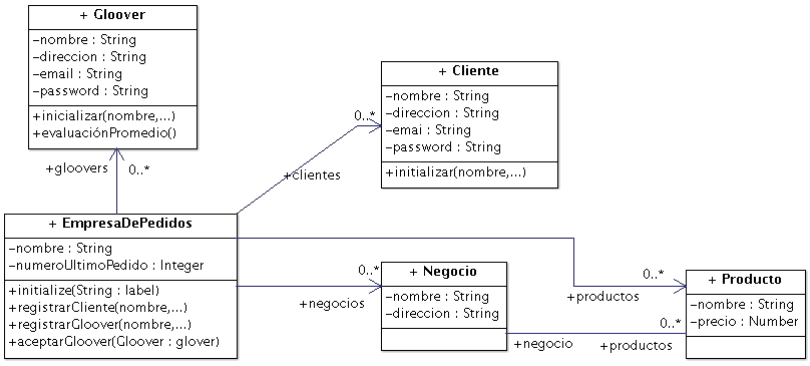
Lifia

## Temas en este metodo y otros

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

Lifia

## Diseño hasta ahora




Lifia

## El usuario compra....

- Para comprar el cliente debe:
  - Loguearse
  - Poner cosas en el carrito
  - Hacer el pedido: con todo lo que hay en el carrito

- Mensaje login (email, password)  Enviado a gloovo

```
login (email, password)
  cliente:= clientes findAndValidate (email, password)
  return (cliente)
```

Tenemos que ver como tratar el caso de error!!!

Lifia

## Manejar el carrito de compras

- Necesitamos una clase Carrito. Un objeto carrito (por ahora) tiene una coleccion de productos (en esta version un producto tiene un negocio asociado).
- Quien conoce al carrito? (**ver inicializaciones**)

- Mensaje  agregarProducto (p)  en Clase Cliente

```
agregarProducto (p)
  carrito agregar (p)
```

Lifia

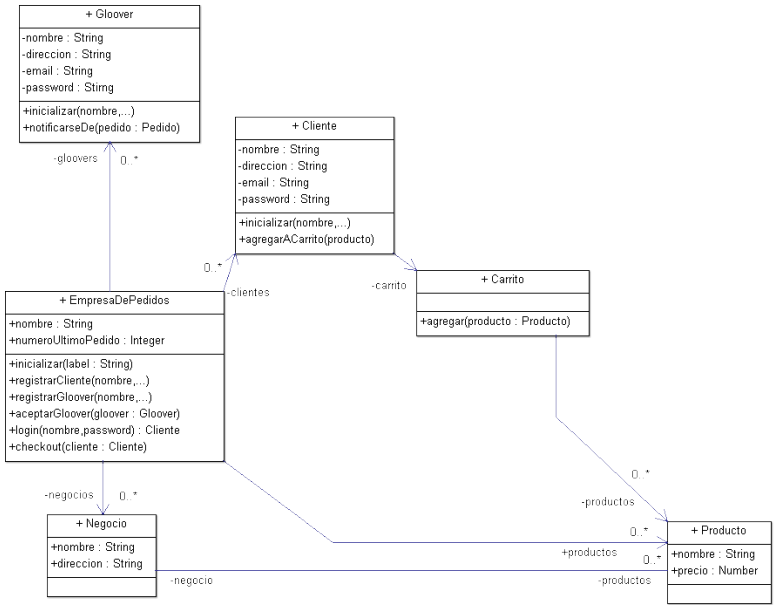
## Inicializacion Cliente corregida

Inicializar (nomb, dir, mail, pass)

nombre:= nomb
direccion:= dir
email:= mail
password:= pass
carrito:= Carrito new

Lifia

## ClienteConCarrito




Lifia

## Generar un pedido

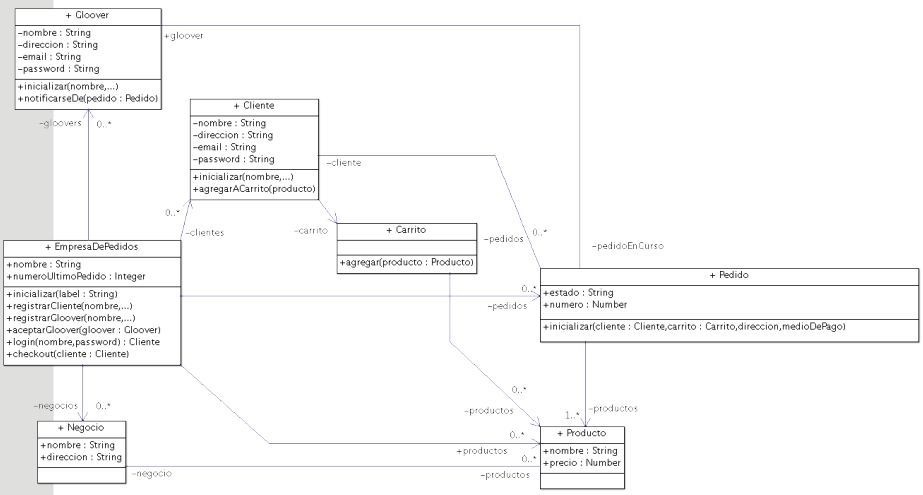
- El cliente decide “cerrar” su pedido y comprar
- Mensaje checkout (cliente) en objeto gloovo
- Por que en gloovo y no en cliente? (ver proceso checkout)

```
checkout (cliente, direccionEnvio,medioDePago)
  p:= Pedido new
  p inicializarCon (cliente, cliente carrito, direccionEnvio,
                    medioDePago)
  gloovers notificarseDe (p)
  pedidos add (p)
```

**Observar relacion Cliente-Pedido, Metodo carrito en Cliente?**

Lifia

## Diseño actual




Lifia

## Notificacion de un pedido nuevo

- En la clase Gloover tenemos un metodo
  notificarseDePedido (p)
- Supongamos que la notificacion es via Whatsapp.
- En la clase Gloover tendriamos codigo especifico necesario para enviar un Whatsapp desde nuestro sistema.
- No parece ser una incumbencia de esa clase.
- Una solucion mejor es delegar esta tarea en un objeto de una clase especifica (ComunicacionWhatsapp)

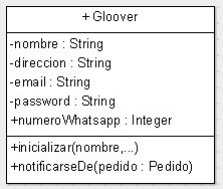
Lifia

## Notificacion…

- Entonces en Clase Gloover

```
notificarseDePedido (p)
  c:=ConexionConWhatsapp new
  c enviarMensaje (numeroWhatsapp, p)
```

Entonces:
un gloover debe conocer su numeroWhatsapp




Que pasa con p (el pedido actual)?
Que mandamos por whatsapp? UN TEXTO
Puede recibir el objeto c un pedido?

Lifia

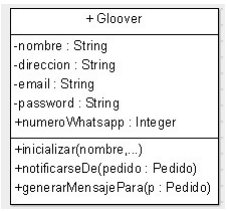
## Notificacion….

notificarseDePedido (p)

c:=ConexionConWhatsapp new

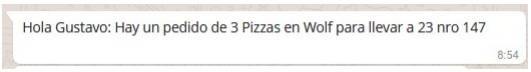
texto:= self generarMensajePara (p)

c enviarMensaje (numeroWhatsapp, texto)




generarMensajePara (p)

-que necesitamos aca para retornar un mensaje util para el gloover?





Lifia

## Un gloover se postula…

- El gloover recibe una notificacion de pedido y se postula
- Donde esta el metodo correspondiente?
- Quien lo “dispara”?

- Opciones:
  - En la clase Gloover

    postularsePara (p:Pedido)  Y Tendria que avisarle a gloovo (lo conoce?)

  - En la clase EmpresaDePedidos

    postularGlover (g:Gloover, p: Pedido)

Lifia

## Un gloover se postula….y se asigna

- Este donde este el “primer” metodo (dependera de la configuracion de la interfaz entre otras cosas), en EmpresaDePedidos se procesa el pedido

postularGloover (g:Gloover, p:Pedido)

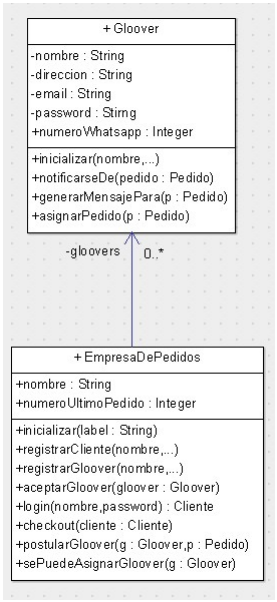
self sePuedeAsignarGloover (g)  Mirar si el tipo tiene denuncias, etc

g asignarPedido (p)

LE TENEMOS QUE AVISAR AL CLIENTE? Como hacemos?

Lifia

## EmpresaDePedidos y Gloover modificados




Lifia

## Asignar un pedido a un gloover, avisarle al usuario

- En la Clase Gloover tenemos el metodo

```
asignarPedido (pedido)
  pedidoEnCurso:=pedido
  pedido asignarGloover (self)
```

- En la clase Pedido

```
asignarGloover (g)
 gloover:=g
 estado:= “glooverAsignado”
```

CUAL ERA EL ESTADO INICIAL?

Lifia

## Clases y subclases

- Una clase representa un concepto en el dominio de problema (o de la solucion).
- ¿Qué sucede cuando las clases tienen comportamiento común?

→ **Subclasificación**

Lifia

## 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.

Lifia

## 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?

Lifia

## 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?

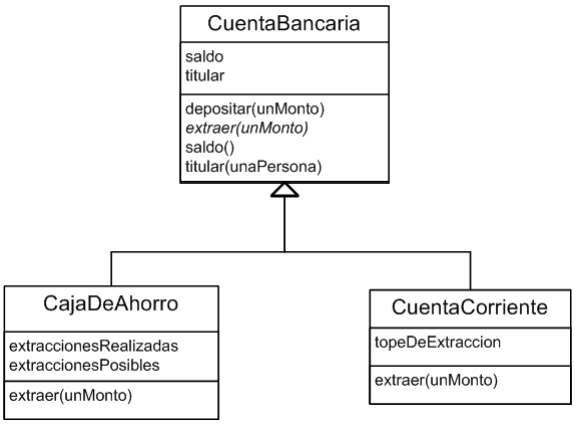
Lifia

## Para que buscar jerarquias?

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

Lifia

## Ejemplo de una Jerarquía de Clases




Lifia

## 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.

Lifia

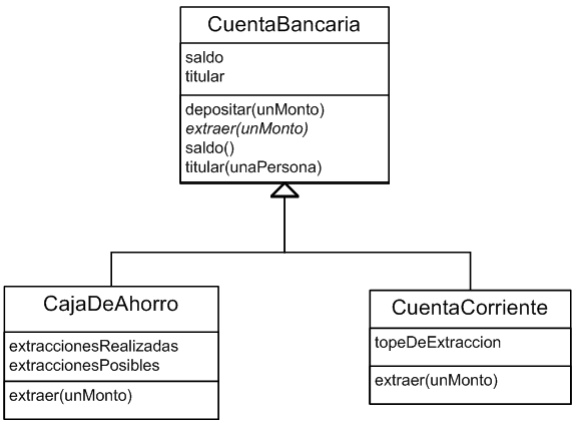
## Herencia

- 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.

Lifia

## *Ejercicio - Cuenta Bancaria*

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



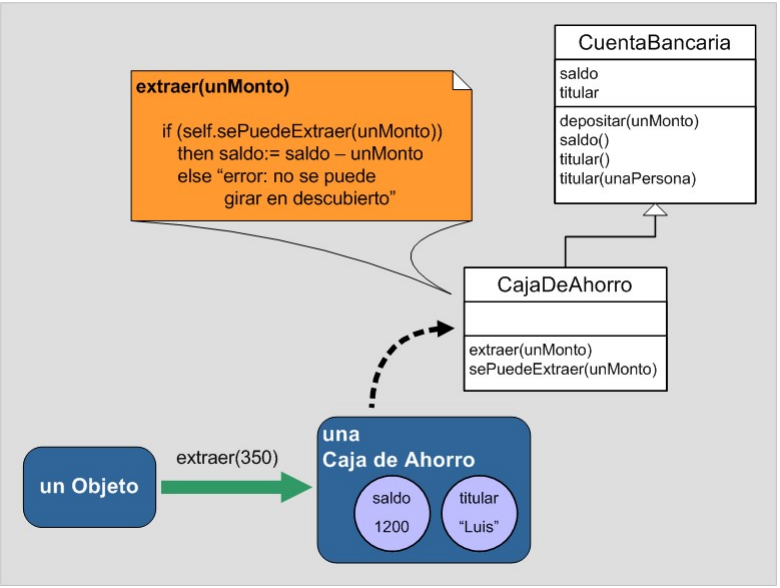

Lifia

## 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***

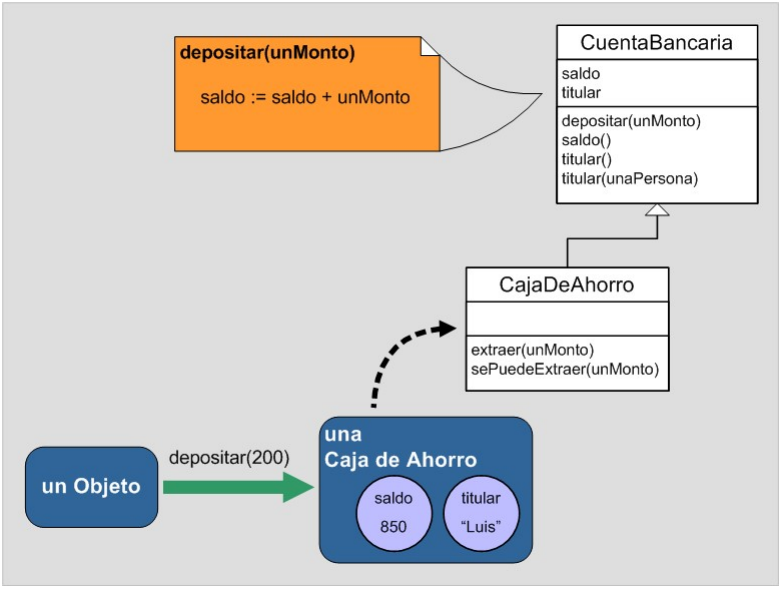
Lifia

## Method Lookup




Lifia

## Method Lookup




Lifia

## 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.

Lifia