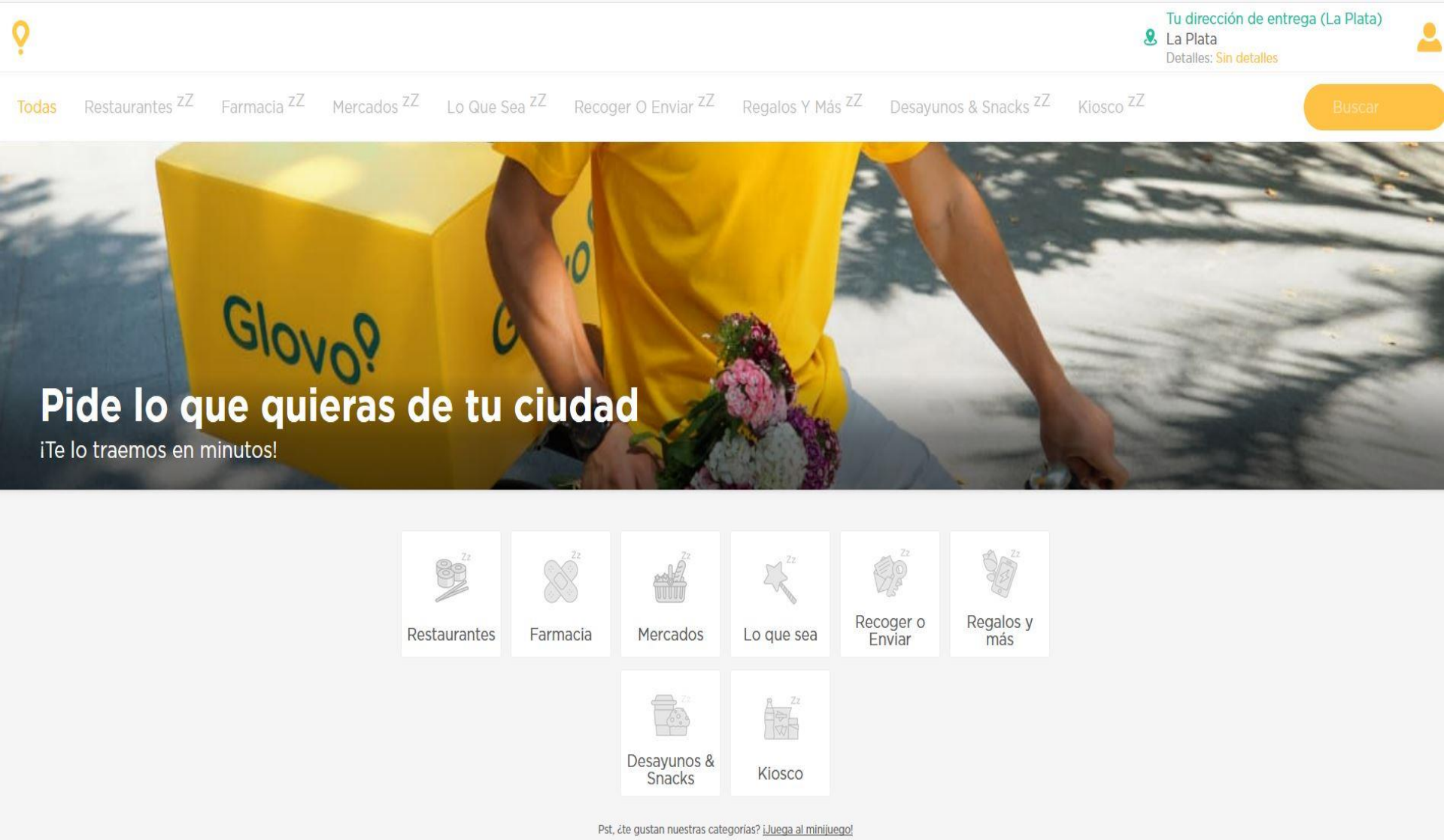
# Ejemplo: Gloovo



# 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.

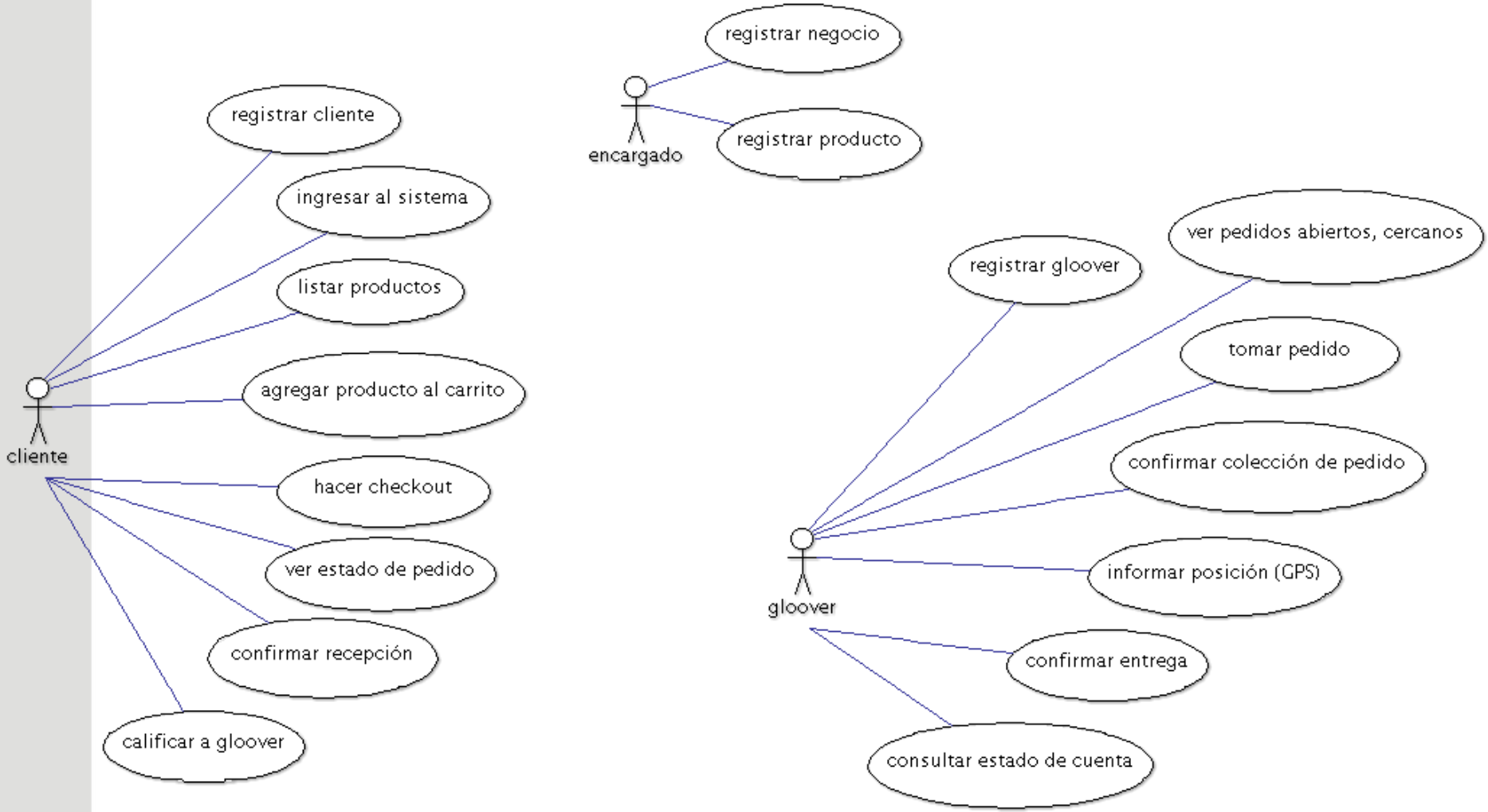
Cuando el cliente recibe el pedido, utiliza la aplicación para confirmar la recepción. En ese momento, se carga el costo a su medio de pago, se para al negocio, y se paga al "acarreador". Adicionalmente el cliente puede calificar al gloover con un puntaje y un comentario

# Contexto del Ejemplo y restricciones

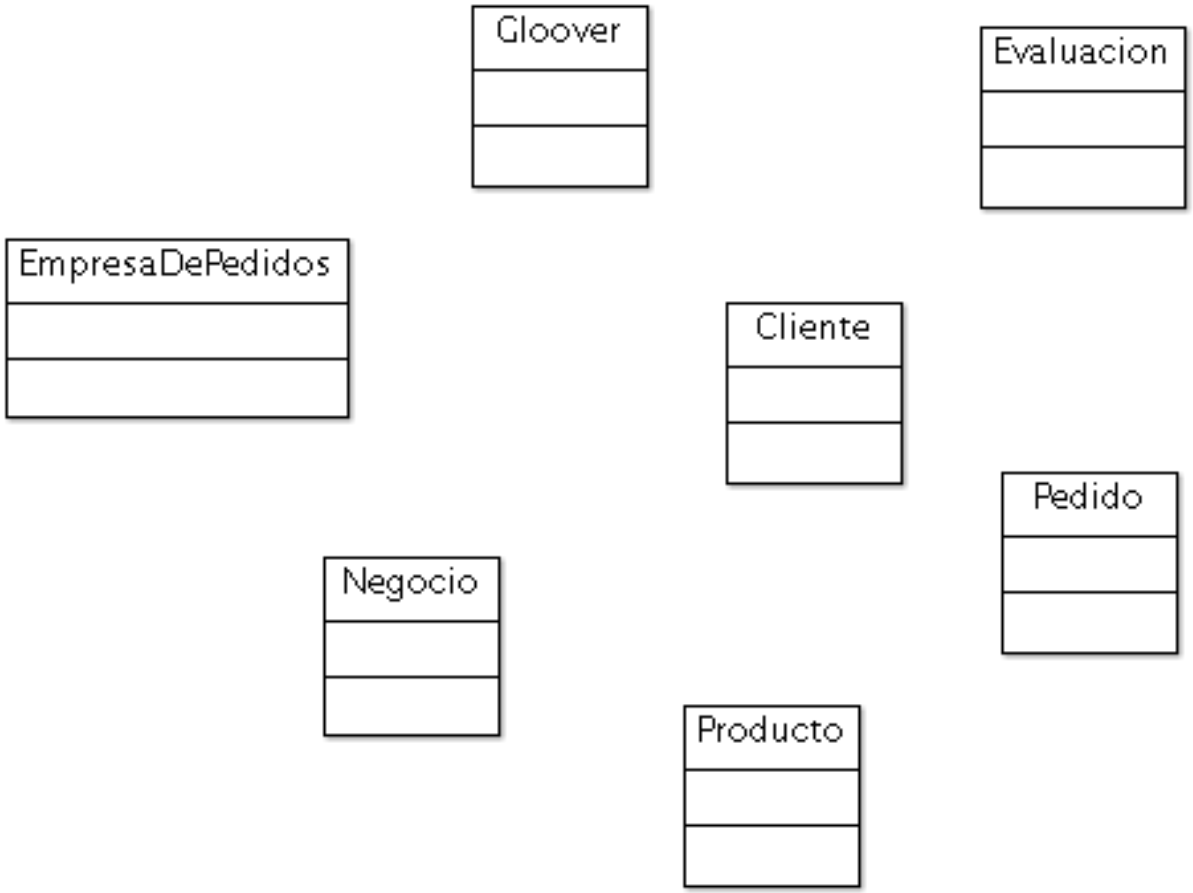
- No vamos a preocuparnos (ahora) por el diseño de las interfases o los formularios.
- No vamos a preocuparnos por la comunicacion entre los usuarios (desde telefonos, tabletas, computadoras, etc) y la aplicacion
- Vamos a ignorar el “almacenamiento” de los datos (en archivos, bases de datos, centralizado,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

# Casos de Uso del sistema Gloovo



# “Diseño” inicial



# Suposiciones iniciales

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

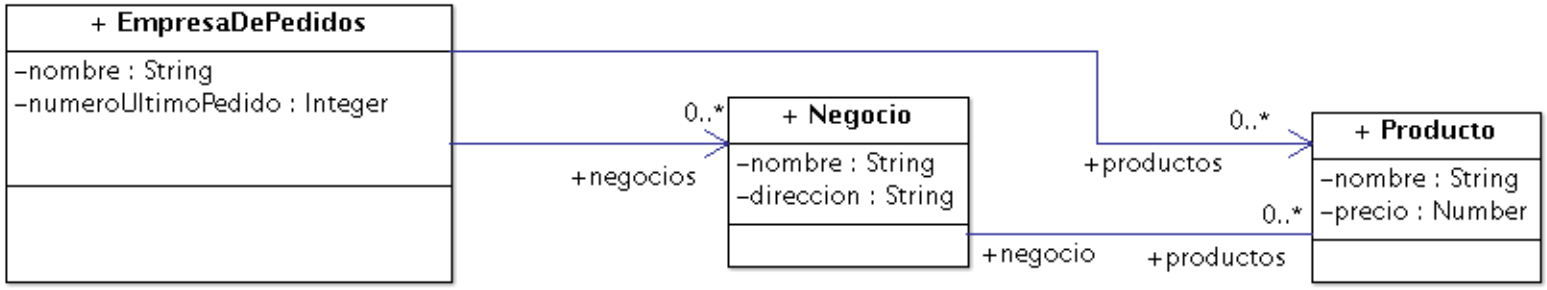
# Como entender/diseñar Gloovo/ Pseudo codigo Java

-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

```
SomeCollection <Producto> 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)
```

# Inicializar 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”

# 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= 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

# 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 new Gloover ( , , , ,)

this.chequearDatos (g)

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

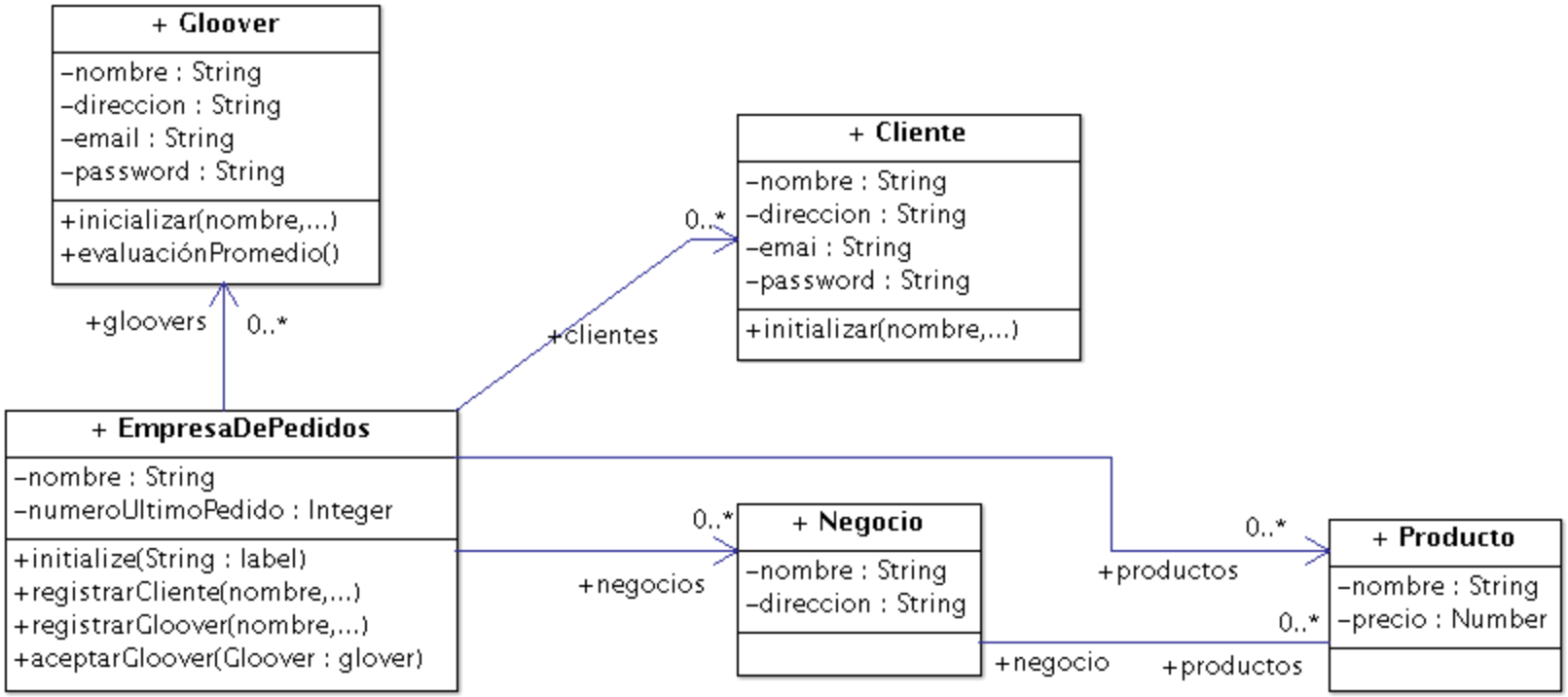
***aceptarGloover (g)***

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



# 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 = clients.findAndValidate (email, password)

return (cliente)

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

# 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)

# Constructor Cliente corregido

***Cliente (nomb, dir, mail, pass)***

nombre= nomb

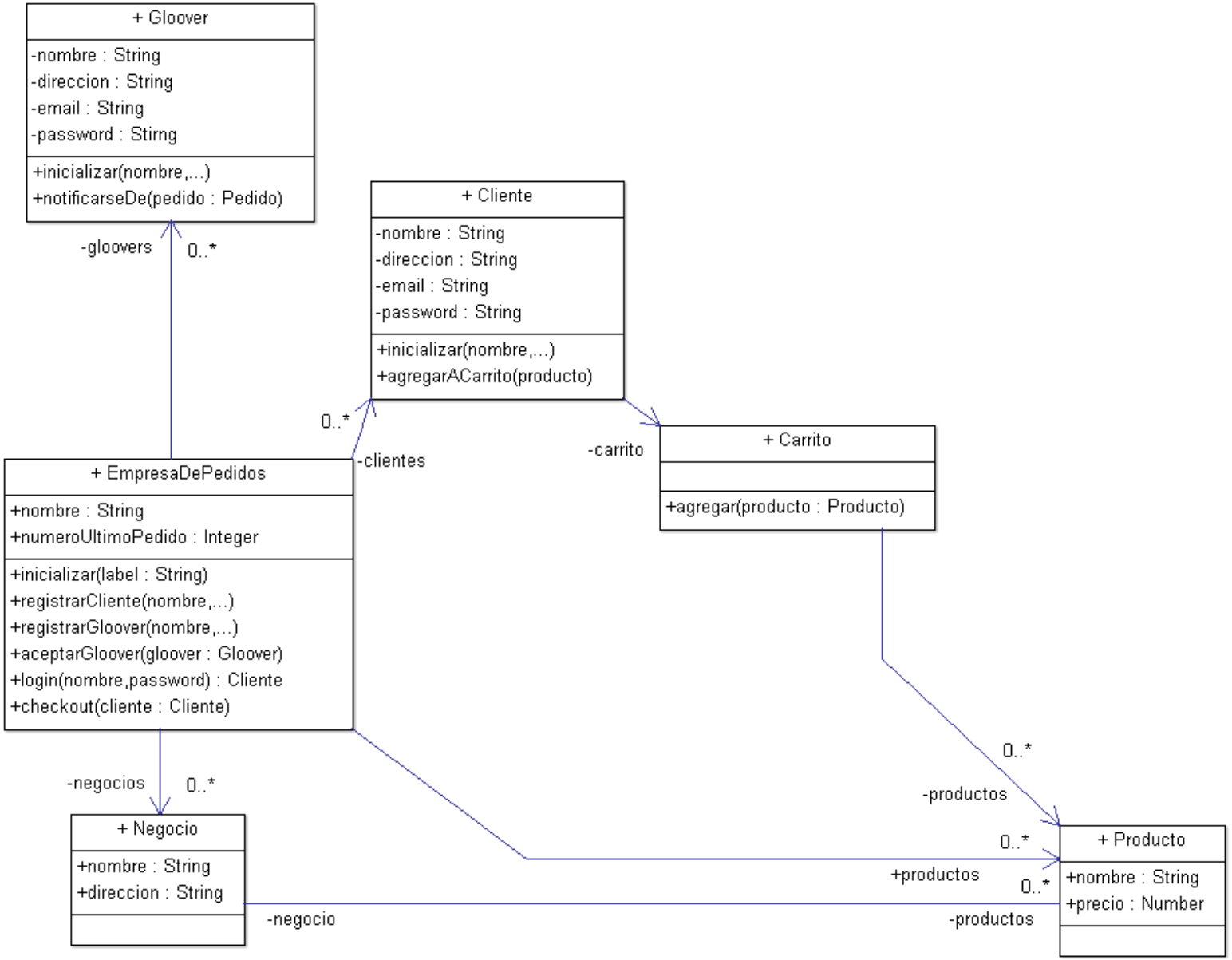
direccion= dir

email = mail

password= pass

carrito= new Carrito

# ClienteConCarrito



# 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)

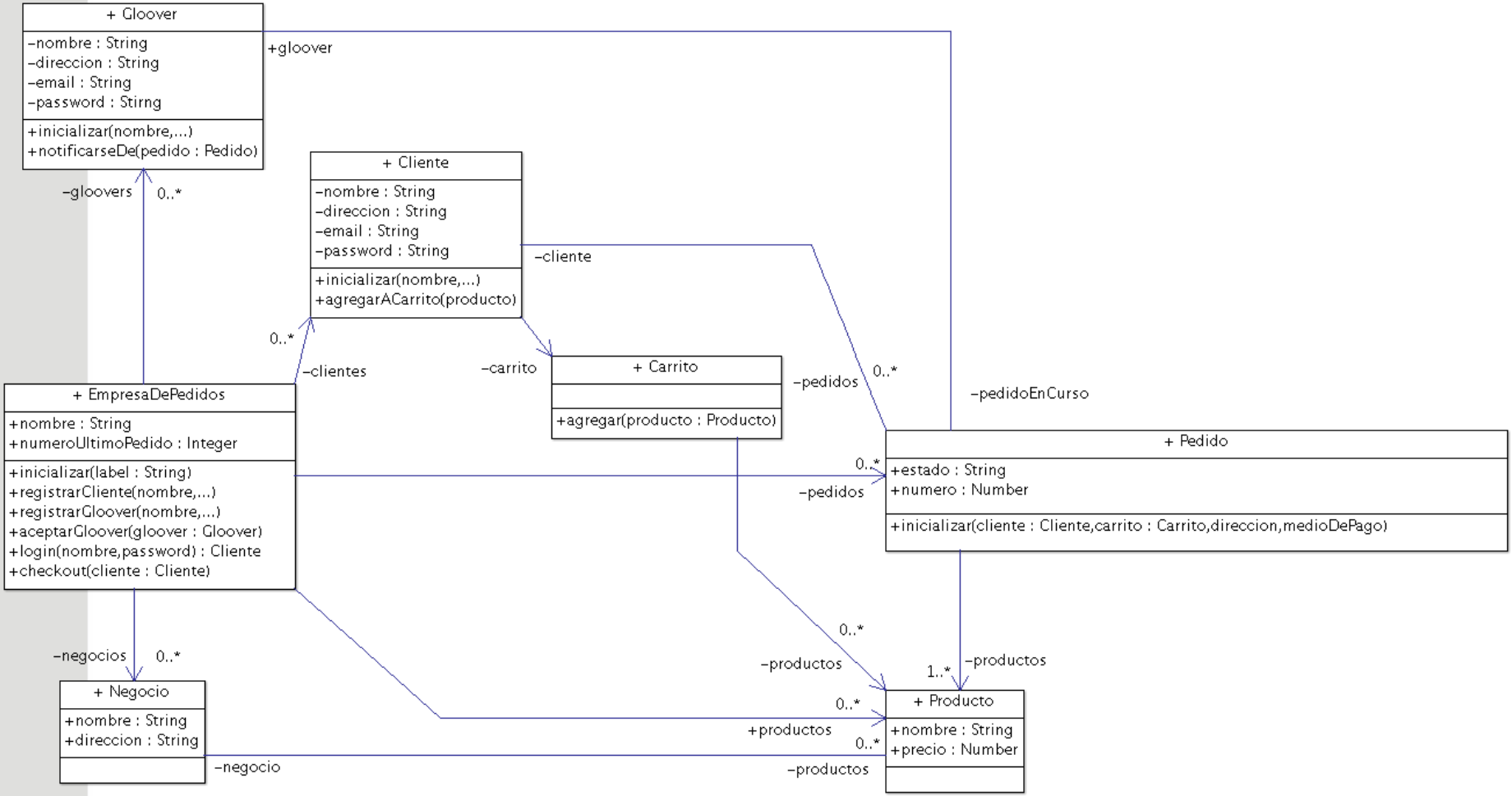
p= new Pedido (cliente, cliente.carrito, direccionEnvio, medioDePago)

gloovers.notificarseDe (p)  ***(Como escribimos esto?)***

pedidos.add (p)

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

# Diseño actual



# Notificacion de un pedido nuevo

- En la clase Gloover tenemos un metodo

  notificarseDe (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)

# Notificacion…

- Entonces en Clase Gloover

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

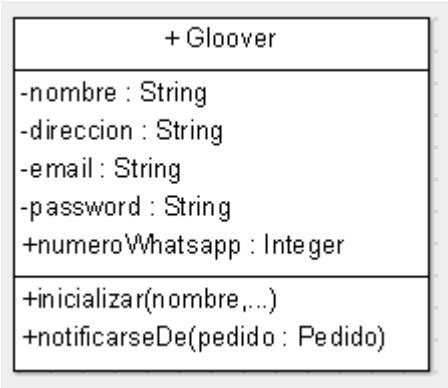
Entonces:

un gloover debe conocer su numeroWhatsapp

Que pasa con p (el pedido actual)?

Que mandamos por whatsapp? UN TEXTO

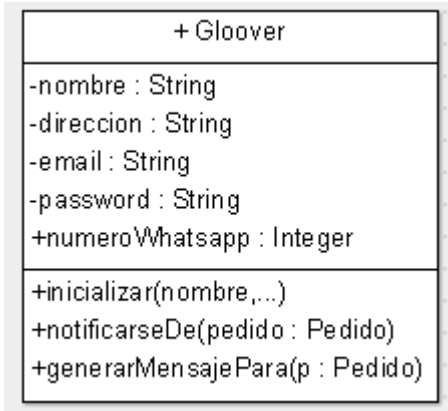
Puede recibir el objeto c un pedido?



# Notificacion....

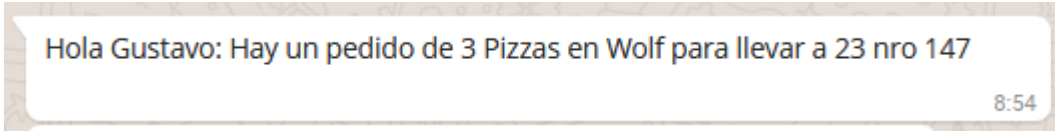
***notificarseDePedido (p)***

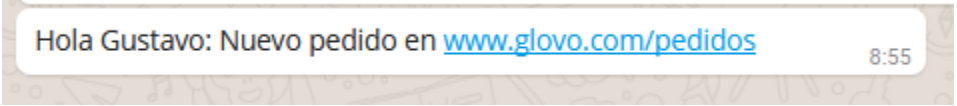
```
c = new ConexionConWhatsapp
texto= self.generarMensajePara (p)
c.enviarMensaje (numeroWhatsapp, texto)
```




***generarMensajePara (p)***

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



# 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)  Y Tendria que avisarle a gloovo (lo conoce?)

  - En la clase EmpresaDePedidos

    postularGlover (gloover, pedido)

# Un gloover se postula….y se asigna

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

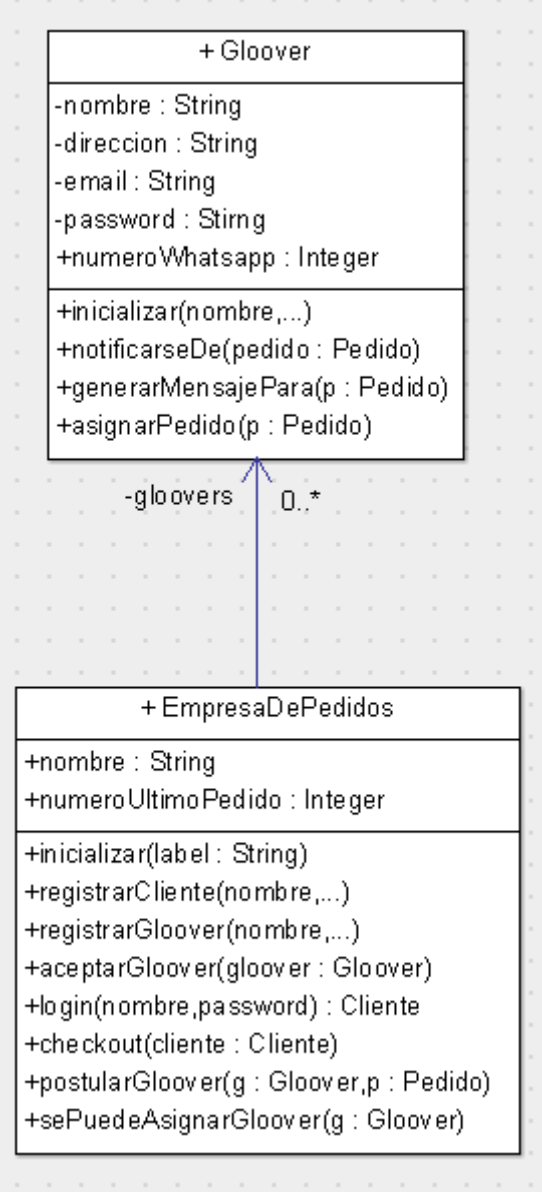
***postularGloover (gloover, pedido)***

this.sePuedeAsignarGloover (gloover)  Mirar si el tipo tiene denuncias, etc

gloover.asignarPedido (p)

LE TENEMOS QUE AVISAR AL CLIENTE? Como hacemos?

# EmpresaDePedidos y Gloover modificados



# 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?

# Temas en Gloovo que dan origen a jerarquias

- Variacion en las formas de notificar a los Gloovers
- Variacion en como decidir a que Gloovers notificar
- Medios de pago del cliente
- Distintos tipos de pedidos….

# Notificacion a los Gloovers

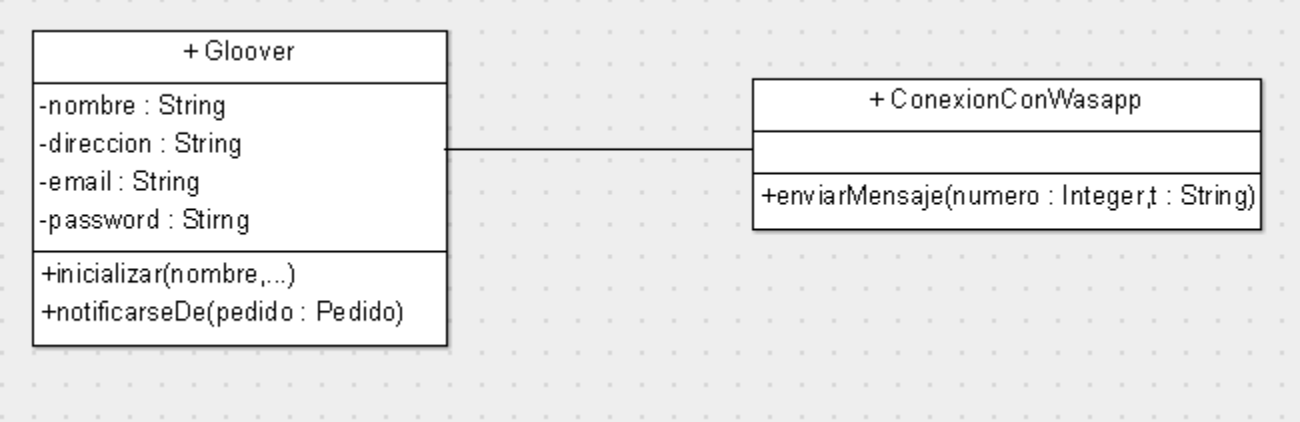
- En la Clase Gloover teniamos

***notificarseDePedido (p)***

c=new ConexionConWhatsapp

texto = this.generarMensajePara (p)

c.enviarMensaje (numeroWhatsapp, texto)




Que hacemos si queremos darle al Gloover la posibilidad de elegir como notificarse: Telegram, Facebook, Whatsapp, etc….

# Opciones….

- Ponerle un “tipoDeNotificacion” como variable y luego el if correspondiente.

- Problemas con esto? Que info tenemos en las variables de instancia? El numero de cada posible cuenta?

# Solucion: Sacar el problema de Gloover

- Tenemos un objeto notificador

- Entonces el gloover DELEGA en ese objeto y en vez de:

***notificarseDePedido (p)***

```
    c:=new ConexionConWhatsapp
    texto= this.generarMensajePara (p)
    c enviarMensaje (numeroWhatsapp, texto)
```

.....

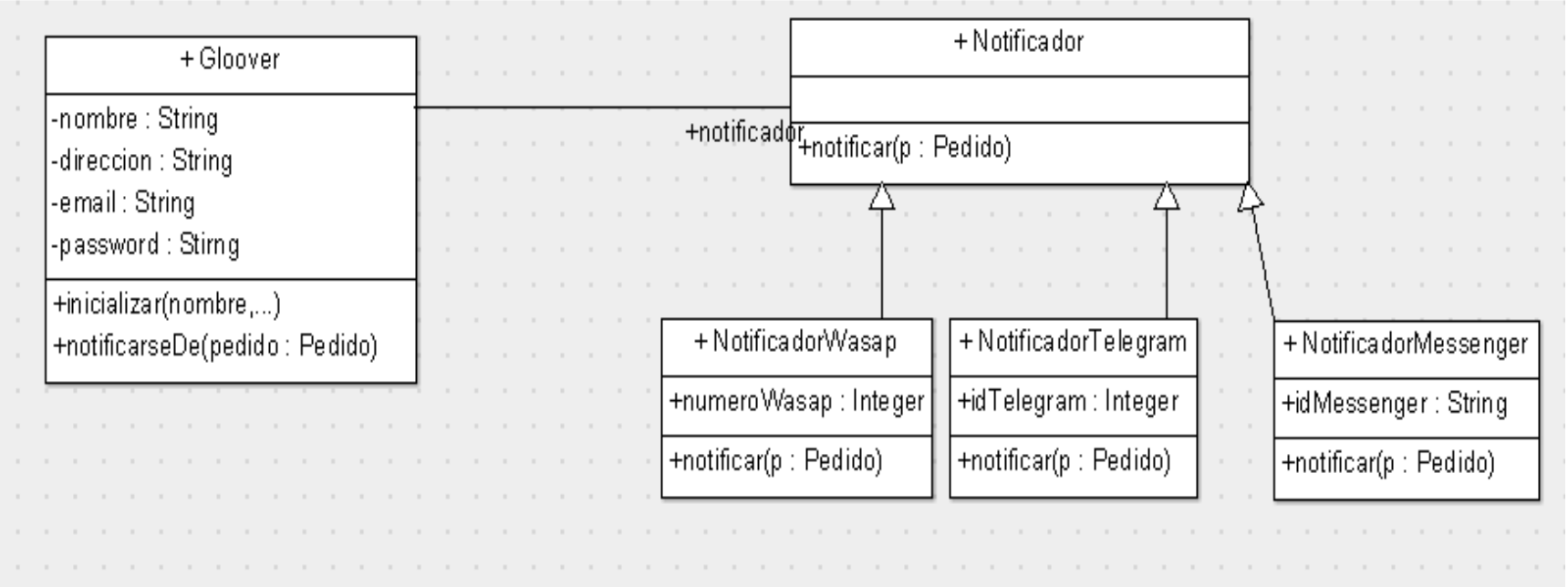
***notificarseDePedido (p)***

```
 notificador.notificar (p)
```

- Que tiene que saber hacer el notificador?
- Como lo configuramos? Cuando?

# Jerarquia de Notificadores

- Como manejamos la variabilidad de formas de notificacion



# Configuracion de los Notificadores

- Cada gloover conoce un objeto notificador al que le delega la tarea de notificarlo (con los datos que tiene el notificador)
- Cuando inicializamos el gloover, tenemos que crear un Notificador (de la clase correspondiente) e inicializarlo
- Como seria este proceso? Donde lo hacemos? Cuando? De donde sacamos los datos?

# Inicializacion Notificadores

- En Clase Gloover

***inicializarNotificadorWasap (numeroWasap)***

notificador= new NotificadorWasap (numeroWasap)

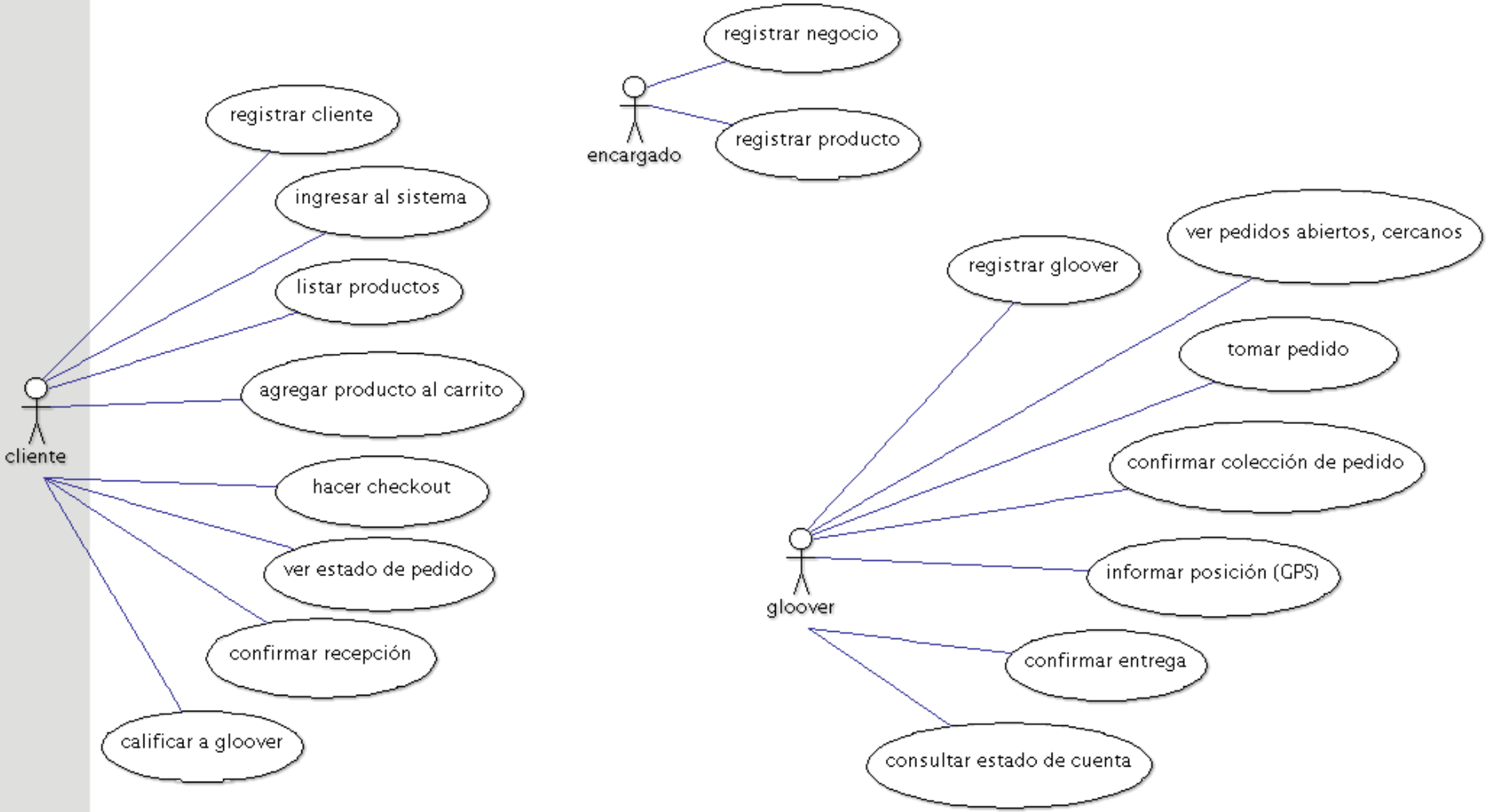
***inicializarNotificadorTelegram (idTelegram)***

notificador= new NotificadorTelegram(idTelegram)

notificador initialize

Como y cuando se disparan estos metodos?

# Cobrarle al cliente



# Gloover entrega un pedido

- Opciones:

  1-Cuando Gloover entrega se cobra al cliente

  2-Recien lo hacemos cuando el cliente confirma que lo recibio

- La opcion 1 parece mas cercana a los estandars de comercio electronico. De hecho se cobra cuando se envia el producto y no hay notificacion de recepcion

- La opcion 2 parece mas “end user-friendly” pero es poco realista

# Gloover entrega pedido

- En Clase EmpresaDePedidos

***pedidoEntregado (pedido)***

this.actualizarPedidosPendientes (pedido)

(pedido.cliente).pagar (pedido)

- Observese que el cliente puede tener muchos pedidos abiertos

# Cliente paga

- En Clase Cliente

***pagar (pedido)***

precio = pedido calcularPrecio

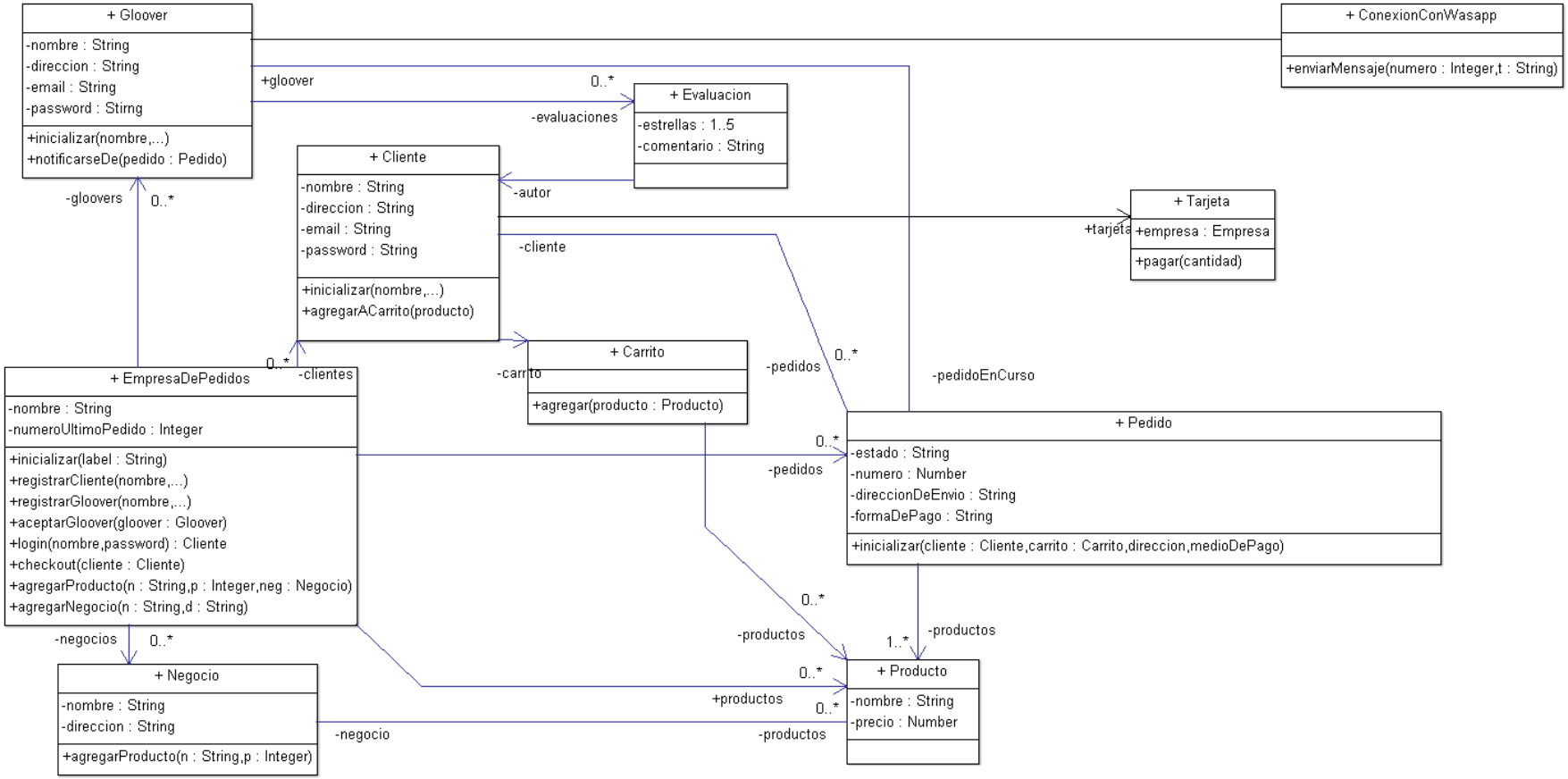
tarjeta.pagar (precio)

- En Clase Pedido

***calcularPrecio***

return (suma de precios de productos + precio envio)

# Incluimos la tarjeta en el modelo



# Mirando el pago con detalle

- La expresion:

  tarjeta.pagar (precio)

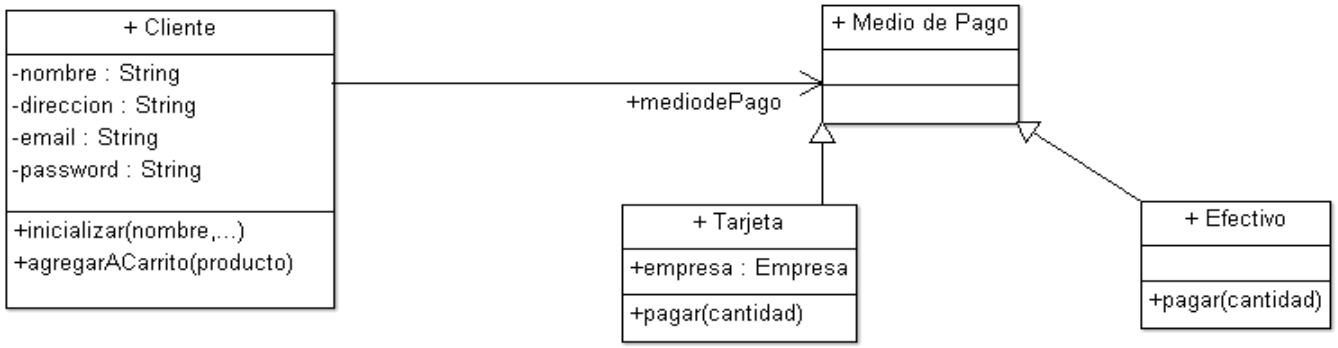
- Asume:

  El cliente conoce a un objeto “tarjeta” (de credito) que sabe “pagar”

- Como manejariamos el caso de otras formas de pago (efectivo, debito, etc)?

- La solucion es “generalizar” el concepto de tarjeta en “medio de pago”

# Medio de pago




- Tenemos que redefinir la variable de instancia “tarjeta”
- Tenemos que implementar el metodo pagar en la clase Efectivo