1.1. SISTEMA BANCARIO

En un sistema bancario existen diferentes familias de productos con características distintas. Están asociados a varios tipos de cuenta que dependen del tipo de cliente que las abra:

• Cuenta Joven, para clientes < 25 años
• Cuenta 10, para clientes entre 26 y 65 años y con nómina domiciliada
• Cuenta Oro, para mayores de 65 años con pensión
• Cuenta Estándar, para clientes que no encajan en las anteriores

Las características de los productos se resumen en la siguiente tabla:

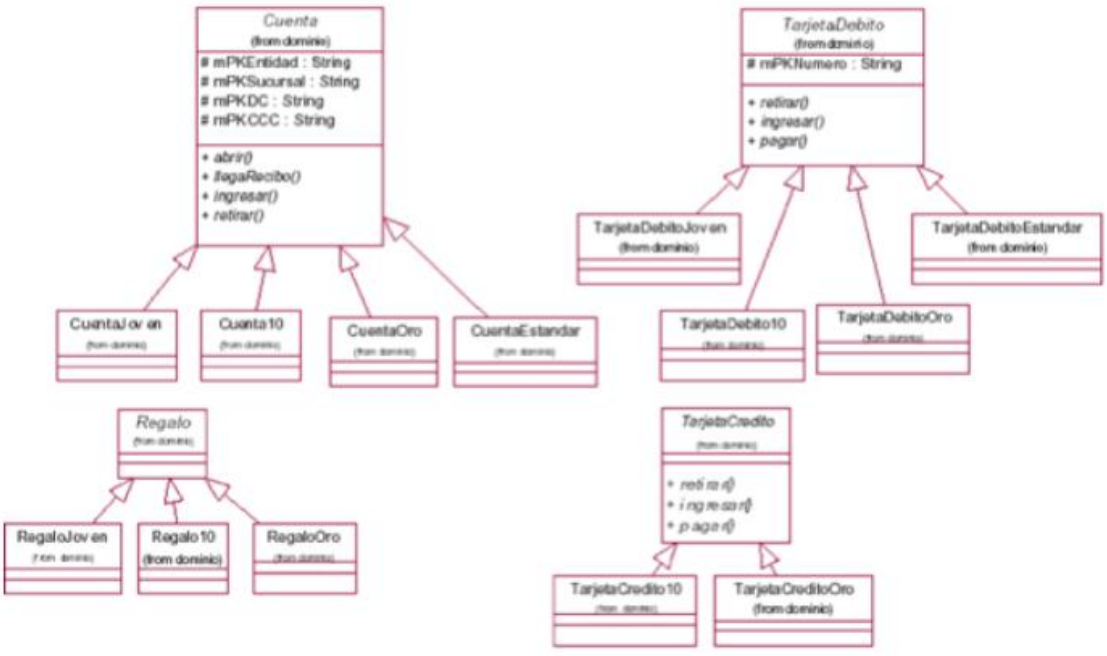
Tipo de Cuenta	Cuenta	Tarjeta de Debito	Tarjeta de crédito	Regalo
Jove	2 % de interés	Gratuita	NO	CD de Música
10	1% de interés 50% descubierto	Gratuita	40 pesos + IVA	Reproductor de Música
Oro	1,5 %	Gratuita	Gratuita 60% pensión	Seguro
Estándar	0,5 %	20 Pesos + IVA	NO	NO

Tal y como se ve, aunque puede parecer que se ofrecen siempre los mismos productos, existen ciertas particularidades entre ellos que hacen que sean distintos, ciertas características que hacen que se construyan de manera distinta.

Se Pide:

i. Representar el conjunto de productos mediante un diagrama de clases

• Diagrama de clases para el conjunto de productos:



ii. **Diseñar una opción para que al crear una cierta cuenta se crearan automáticamente los productos asociados al tipo de cuenta**

Podríamos crear uno a uno los elementos correspondientes a cada uno de los tipos de cuentas, pero no es performante.

Una buena opción es que al crear una cierta cuenta automáticamente se crearan los productos asociados al tipo de cuenta.

El patrón **Abstract Factory**:

Especifica como crear familias de objetos que guardan cierta relación entre si.

Solo hay que utilizar las interfaces que se proponen para interactuar con el sistema, sin importar el tipo de elemento a instanciar, ya que lo hará la fábrica.

Se generaliza el comportamiento de una familia de clases

El sistema es independiente de como se crean, componen o representan las entidades representadas.

Los sistemas de vuelven fácilmente extensibles.

Fábrica Abstracta

ABSTRACT FACTORY

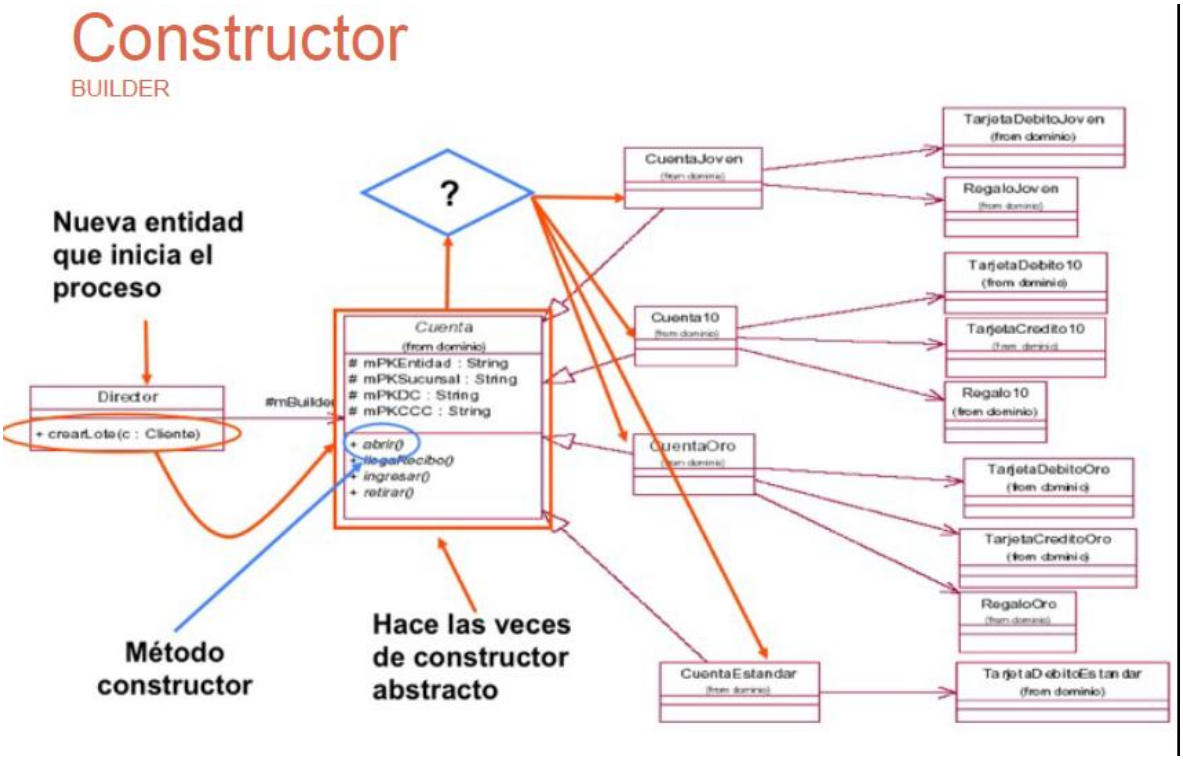
- La fábrica abstracta tiene operaciones abstractas (por cada producto)
- Tenemos tantos tipos de fábricas concretas como de productos.
- Cada fábrica concreta conoce a los productos que puede fabricar.

[para simplificar sólo están indicadas las relaciones de FabricaJoven]

iii. Diseñar la opción para construir objetos complejos que pueden ser de distinto tipo (los tipos de cuenta).

Para asegurar que se utiliza siempre el mismo proceso (La misma secuencia de pasos) para crear una cuenta de un determinado tipo o de un lote formado por varias cuentas,

La idea de buscar un desacople de las funciones del constructor (la clase cuenta de la fabrica abstracta)



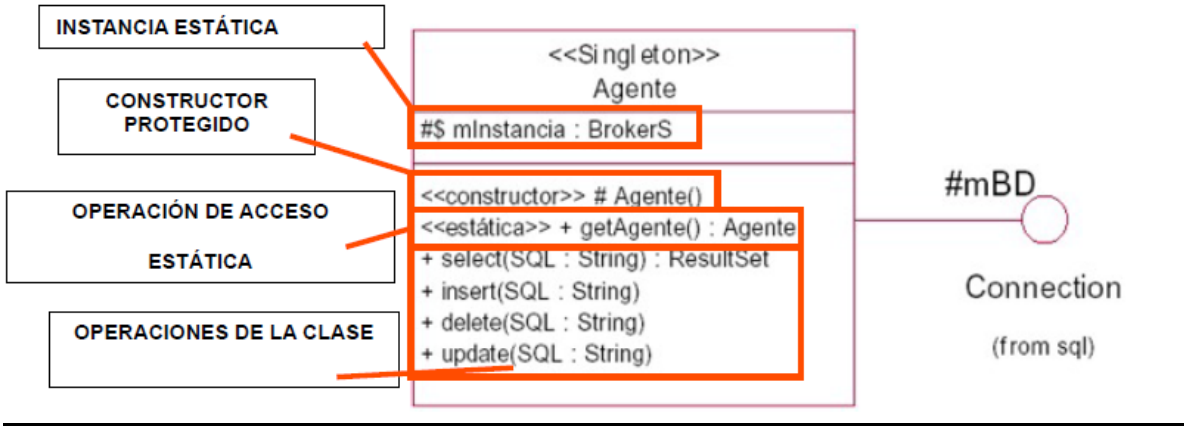
iv. Diseñar un agente de base de datos (o Broker) en el que se centralice el acceso a la BBDD. Asegurando que existe una única instancia de ese agente, para que todos los objetos que la usen estén tratando con la misma instancia.

Se desea asegurar que exista una única instancia de ese agente, para que todos los objetos que la usen estén tratando con la misma instancia, accedan a ella de la misma forma.

Usar una variable global no garantiza que solo se instancie una vez.

Singleton

- Los clientes acceden a la única instancia mediante la operación getAgente()
- Esta operación es también responsable de su creación (si existe nos da la instancia y si no existe, la crea)

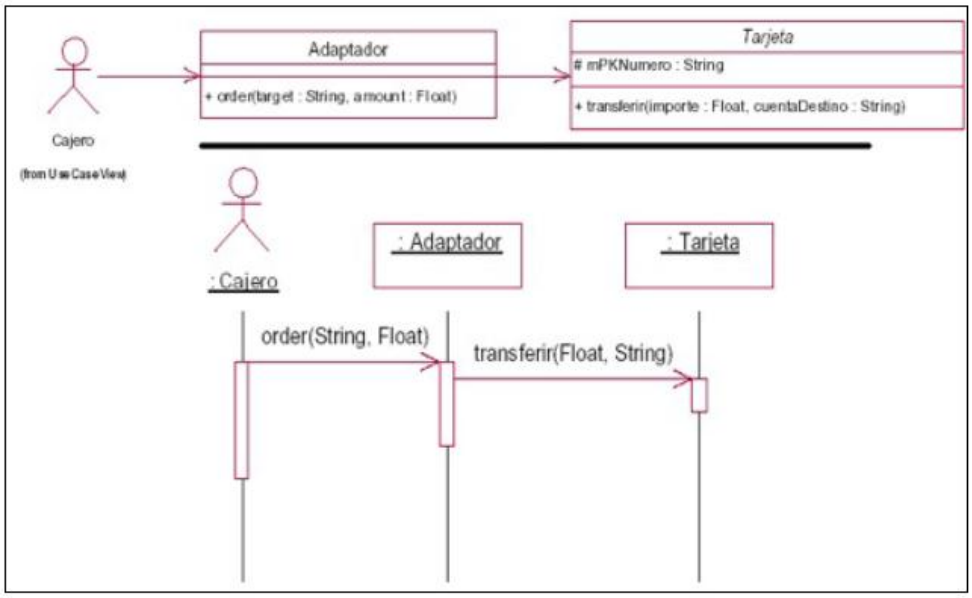


v. **Supongamos que en el sistema bancario**
La clase "Tarjeta" tiene un método transferir (float importe, String cuentaDestino),
La clase "Cuenta" tiene un método transferir (float importe, String cuentaDestino)
El banco ha adquirido un modelo novedoso de cajeros automáticos que permiten hacer transferencias, pero la interfaz esperada para esta operación es: order (String target, float amount)
¿Cómo es posible hacer que las instancias de Tarjeta y Cuenta respondan adecuadamente a los nuevos cajeros?

Una solución seria modificar el código de tarjeta (y de cuenta), añadiendo un nuevo método.

ADAPTER

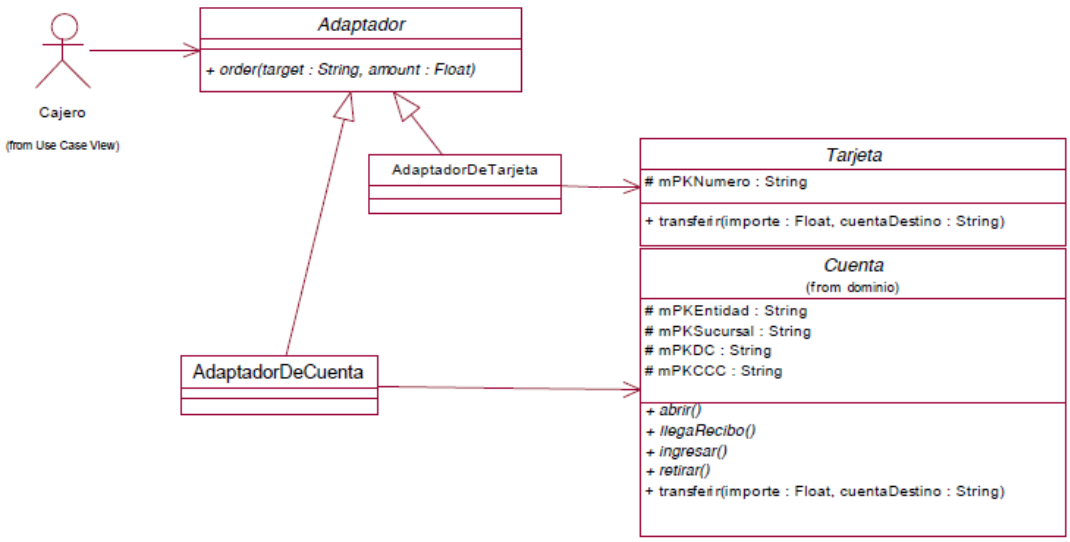
- Solución: Utilizar el patrón Adaptador
- (mucho más elegante y no hay que tocar el código de la clase)



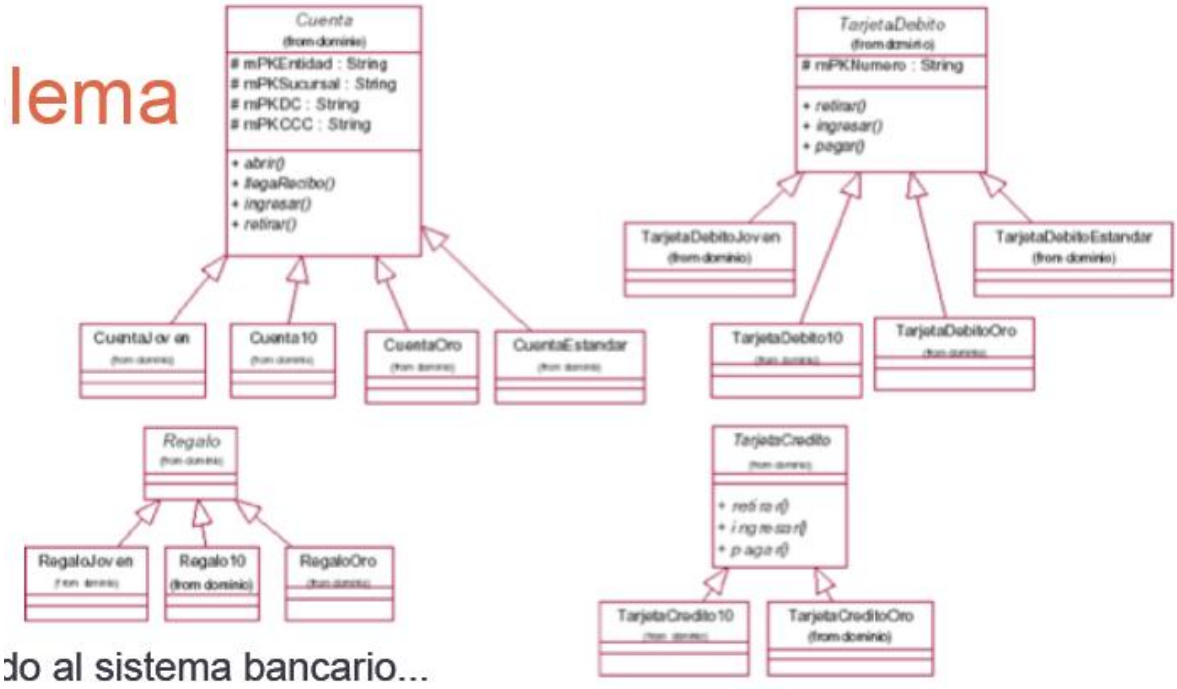
Adaptador

ADAPTER

- El Adaptador podría ser una clase abstracta que sería instaciada según el caso. (hacemos del adaptador una fábrica abstracta).

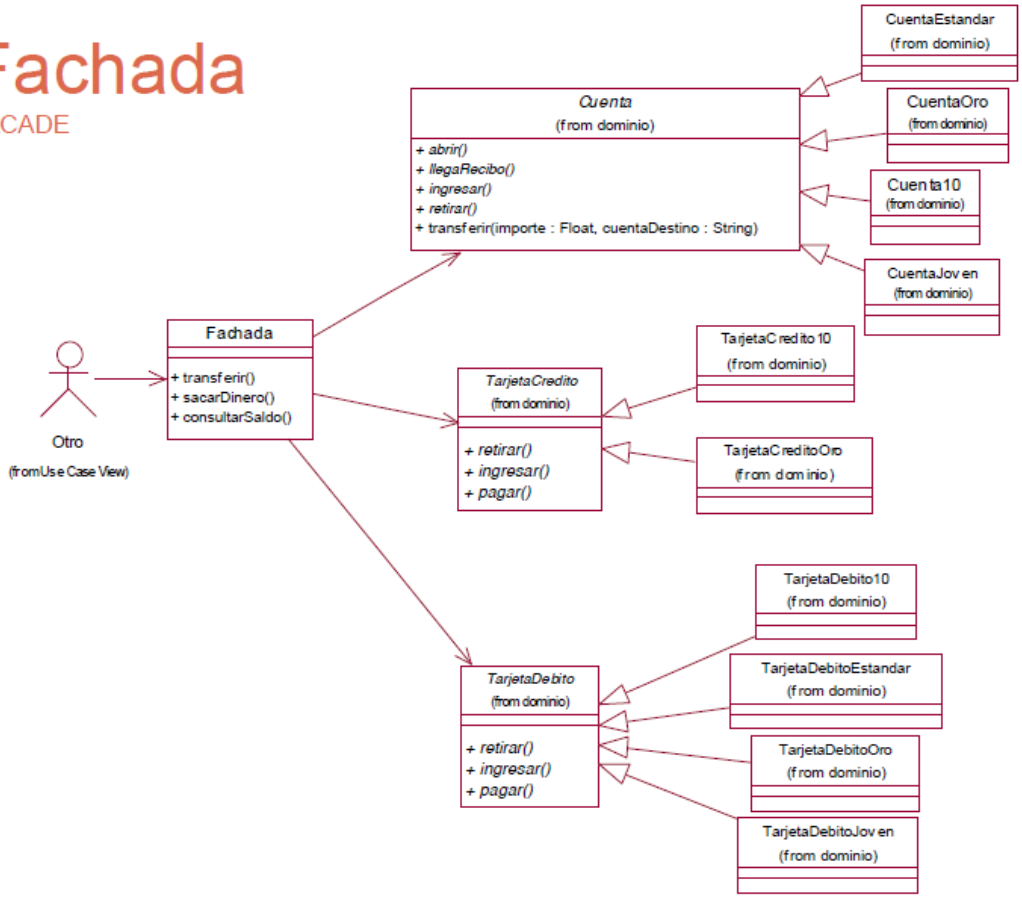


vi. Se desea dotar a los desarrolladores de la capa de presentación de un mecanismo unificado de acceso a la capa de dominio ofreciéndoles las siguientes operaciones públicas:
transferir(), sacarDinero(), y consultarSaldo()
¿Cómo es posible modelar esto?



Fachada

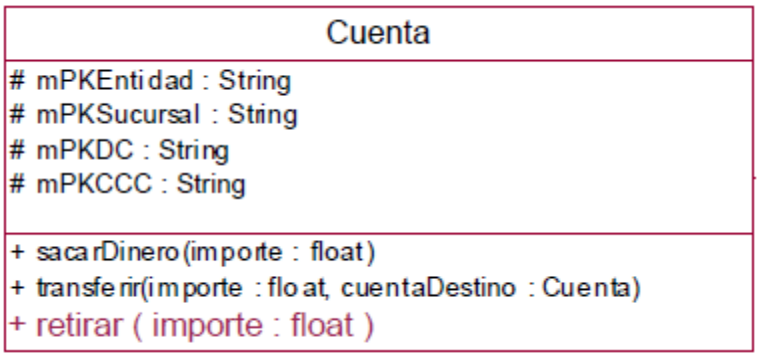
FACADE



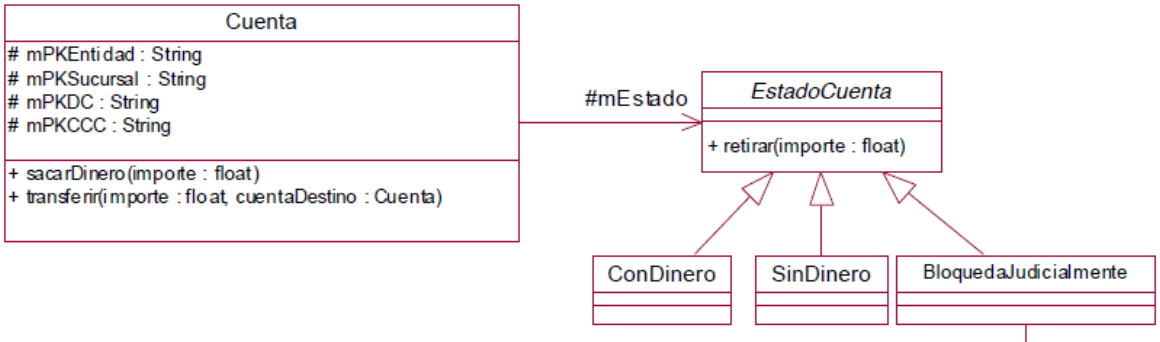
vii. Para la clase “Cuenta” , es conocido que una parte importante del comportamiento de este objeto para la operación “retirar(float importe)” depende del estado de la cuenta, que puede ser:

• Cuenta ConDinero
• Cuenta SinDinero
• Cuenta BloqueadaJudicialmente

Se desea delegar el comportamiento de la operación “retirar”, ¿cómo es posible modelarlo?



STATE



- De esta manera, agregar nuevos estados y modificar los existentes es muy sencillo.
- En ningún caso habría que tocar el código de la clase *Cuenta*.