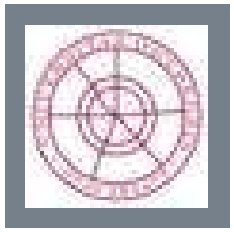


Arquitectura Software Dirigida por Modelos.

Antonio J. Suárez Fábrega

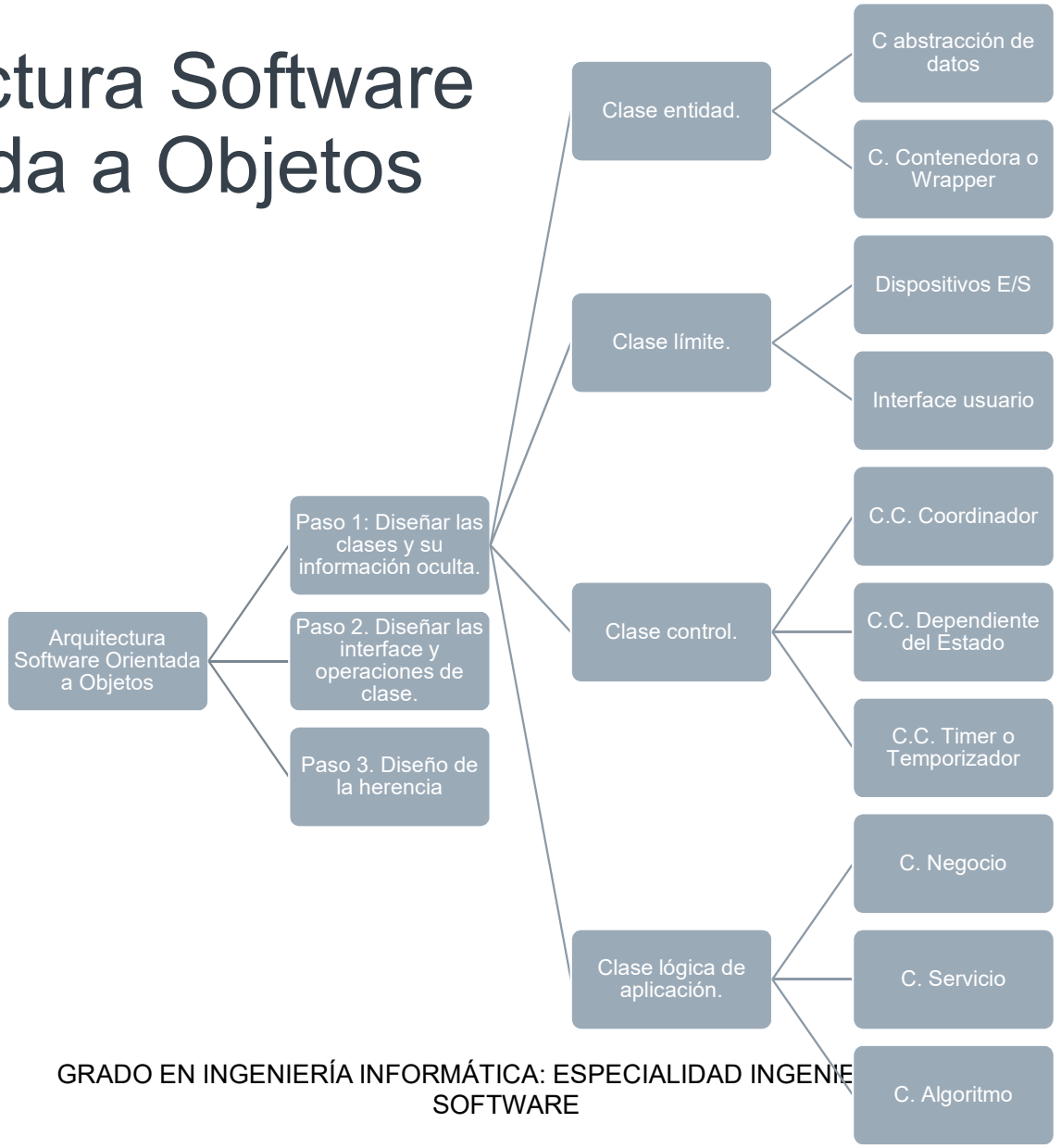
TEMA 1: ARQUITECTURA SOFTWARE

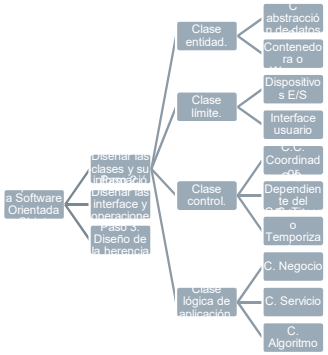
- 1.1. Introducción
- 1.2. Patrones Arquitectónicos, estructura
- 1.3.- Patrones Arquitectónico, comunicaciones.

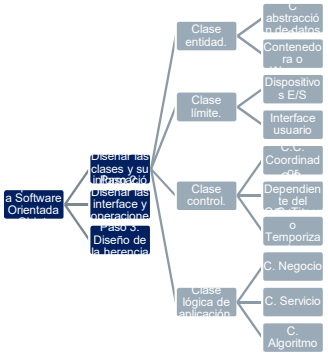
3.- Patrones Arquitectónicos.

3.1.- Arquitectura Software en Subsistemas
3.2.- **Arquitectura Software Orientada a Objetos**
3.3.- Arquitectura Software Cliente/Servidor
3.4.- Arquitectura Software Orientada a Servicios
3.5.- Arquitectura Software Orientada a Componentes

Arquitectura Software Orientada a Objetos







Arquitectura Software Orientada a Objetos

Paso 1: Diseñar las clases y su información oculta.

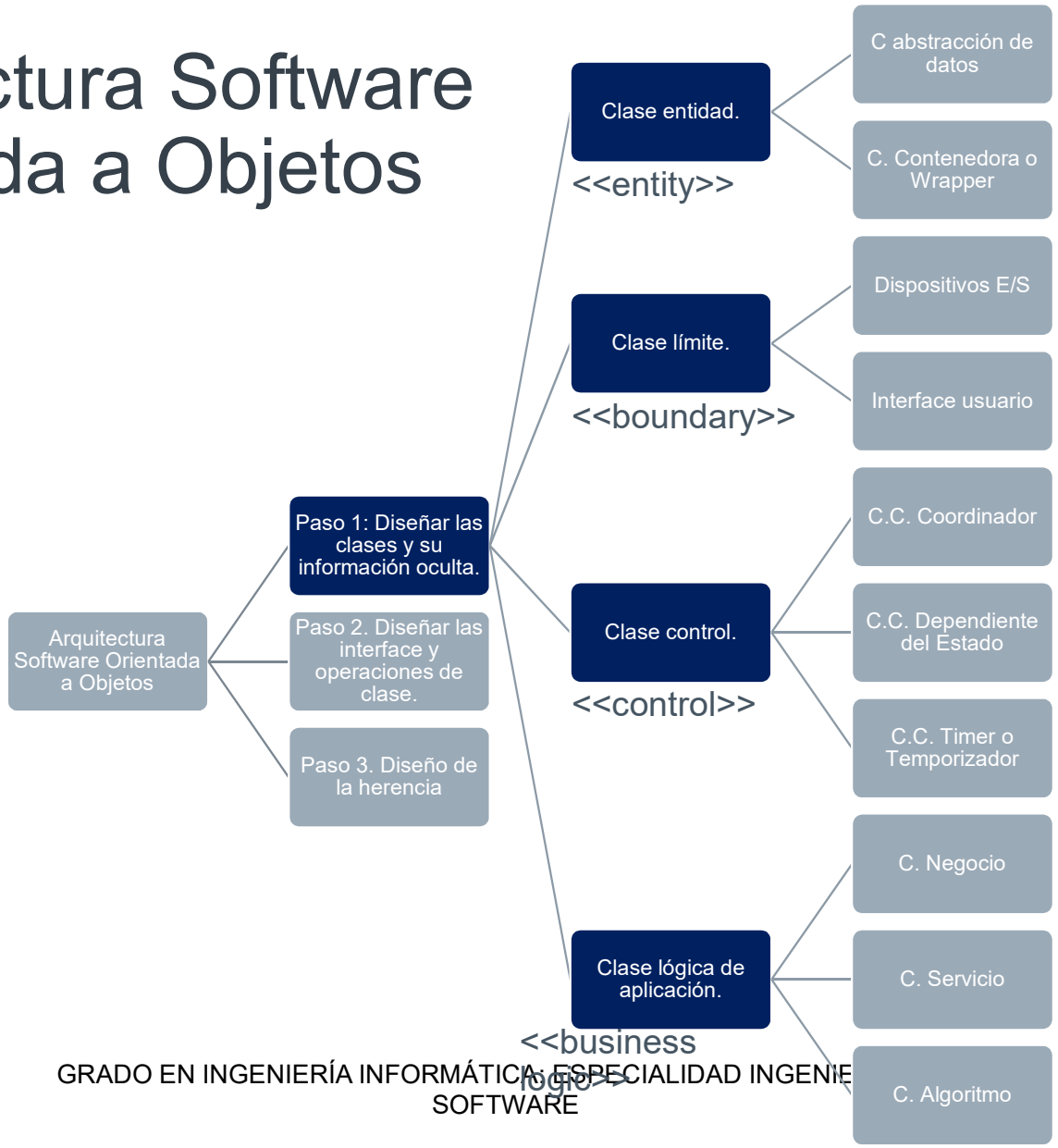
Paso 2. Diseñar las interface y operaciones de clase.

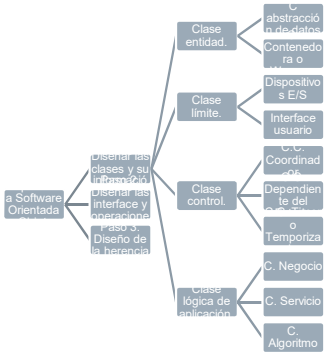
Paso 3. Diseño de la herencia

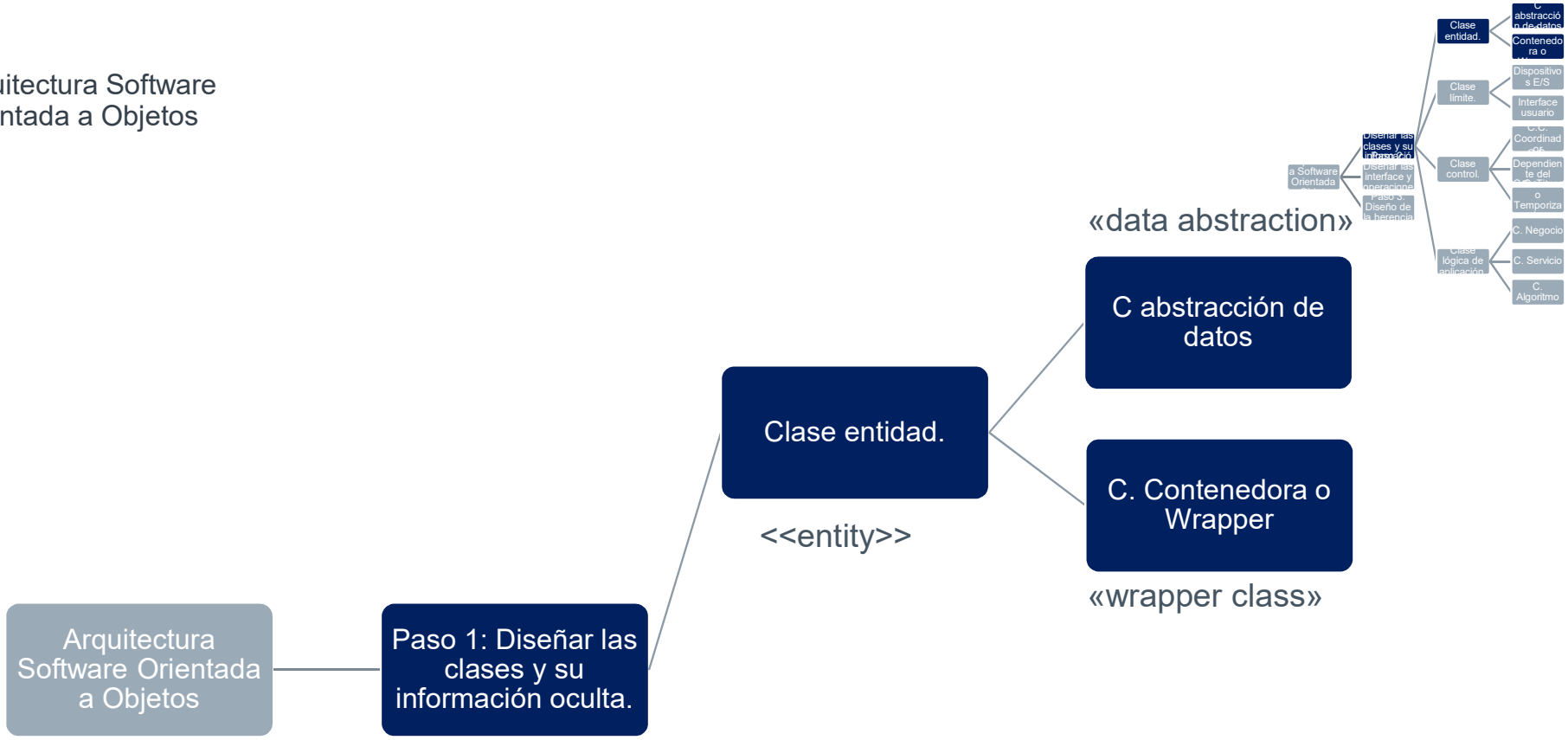
Diseño de Arquitectura Software Orientada a Objetos

- Paso 1. Diseñar las clases y su información oculta.
- Paso 2. Diseñar las interface y operaciones de clase.
- Paso 3. Diseño de la herencia

Arquitectura Software Orientada a Objetos



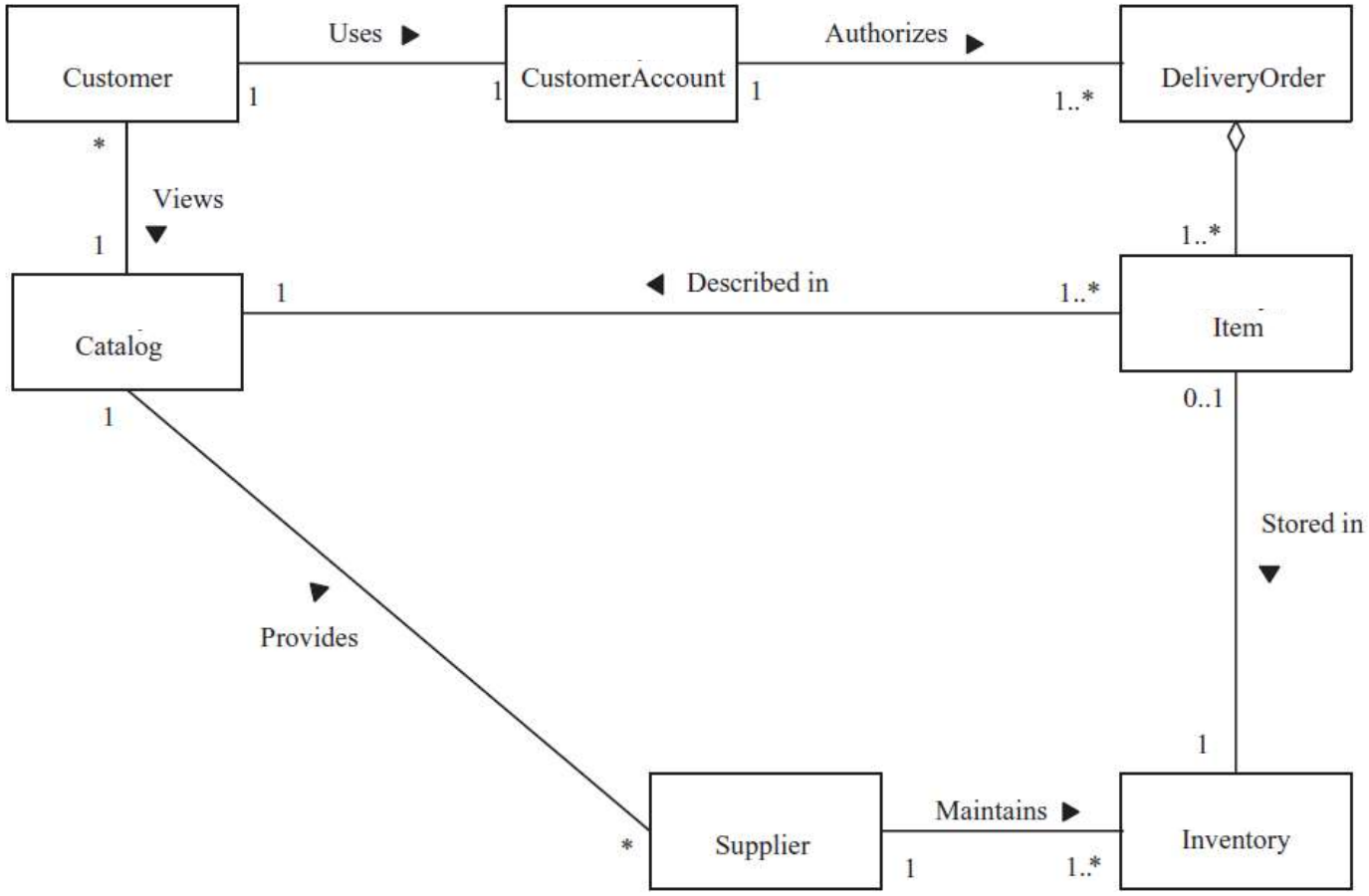


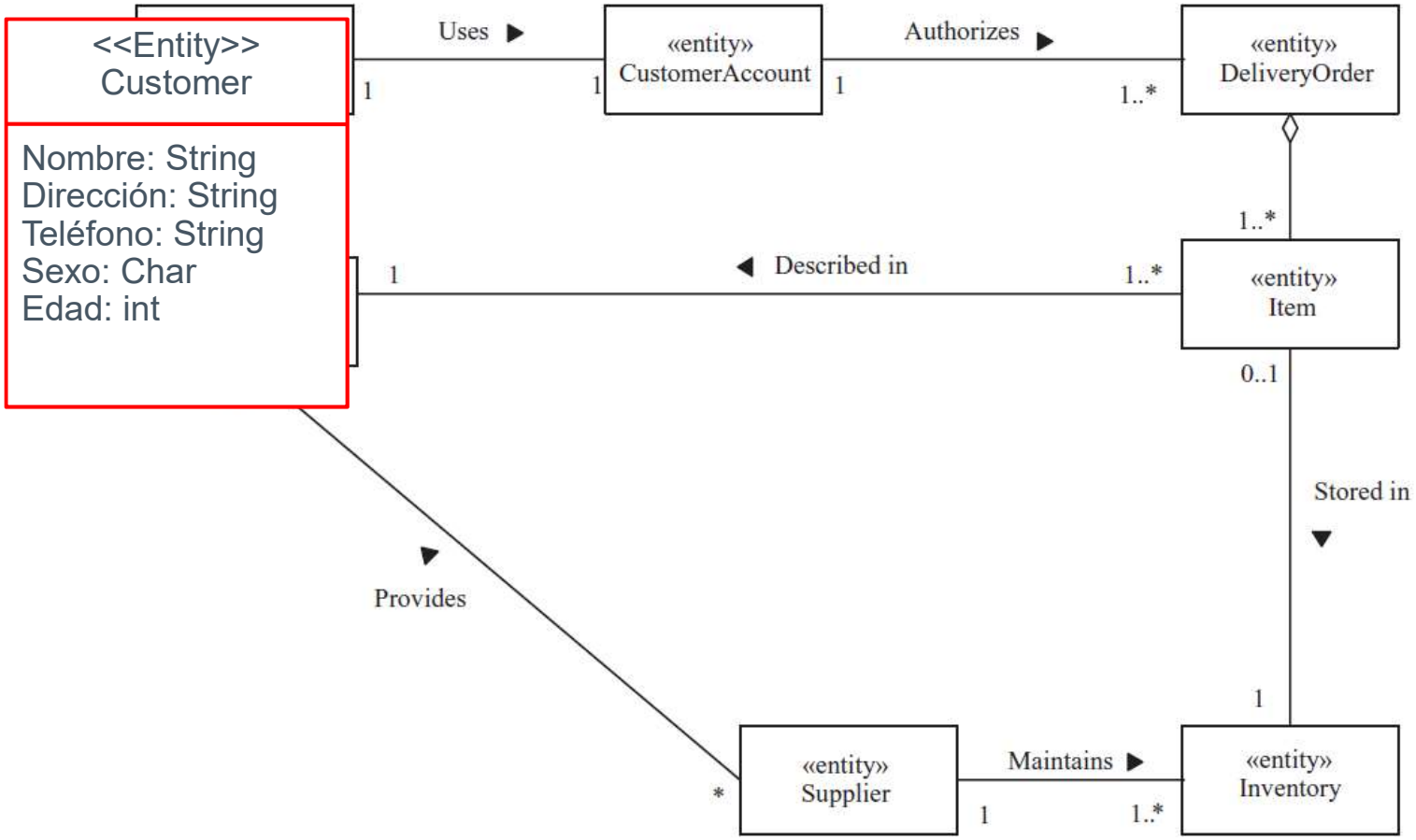

Arquitectura Software Orientada a Objetos
Paso 1: Diseñar las clases y su información oculta.
Clase entidad.
<<entity>>
C abstracción de datos
«data abstraction»
C. Contenedora o Wrapper
«wrapper class»

Diseño de Arquitectura Software Orientada a Objetos

- Paso 1. Diseñar las clases y su información oculta.
  - Clase entidad. Son determinadas en el modelo de análisis y encapsulan los datos. Se representan con el estereotipo <<entity>>.

Diseño de Arquitectura Software Orientada a Objetos



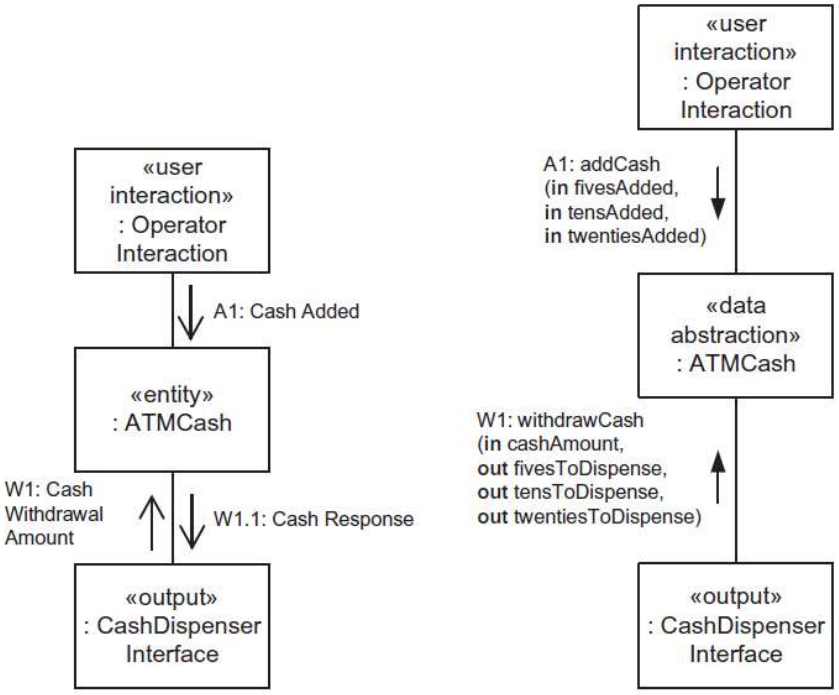

<<Entity>>
Customer
Nombre: String
Dirección: String
Teléfono: String
Sexo: Char
Edad: int
Uses
1
1
«entity»
CustomerAccount
Authorizes
1
1..*
«entity»
DeliveryOrder
1..*
1
Described in
1..*
«entity»
Item
0..1
Stored in
1
Provides
*
«entity»
Supplier
Maintains
1
1..*
«entity»
Inventory

Diseño de Arquitectura Software Orientada a Objetos

- Paso 1. Diseñar las clases y su información oculta.
- Clases entidad. Estas clases se pueden categorizar en clases de abstracción de datos y en Wrapper clases o clases contenedoras.
  - Clase de abstracción de datos. Encapsulan la estructura de los datos.
  - Wrapper clase. (clase contenedora). Oculta los detalles de como interactúa con un sistema existente o con un sistema heredado, que podría ser para acceder a datos almacenados en un sistema de gestión de archivos o un sistema de gestión de base de datos.

Diseño de Arquitectura Software Orientada a Objetos

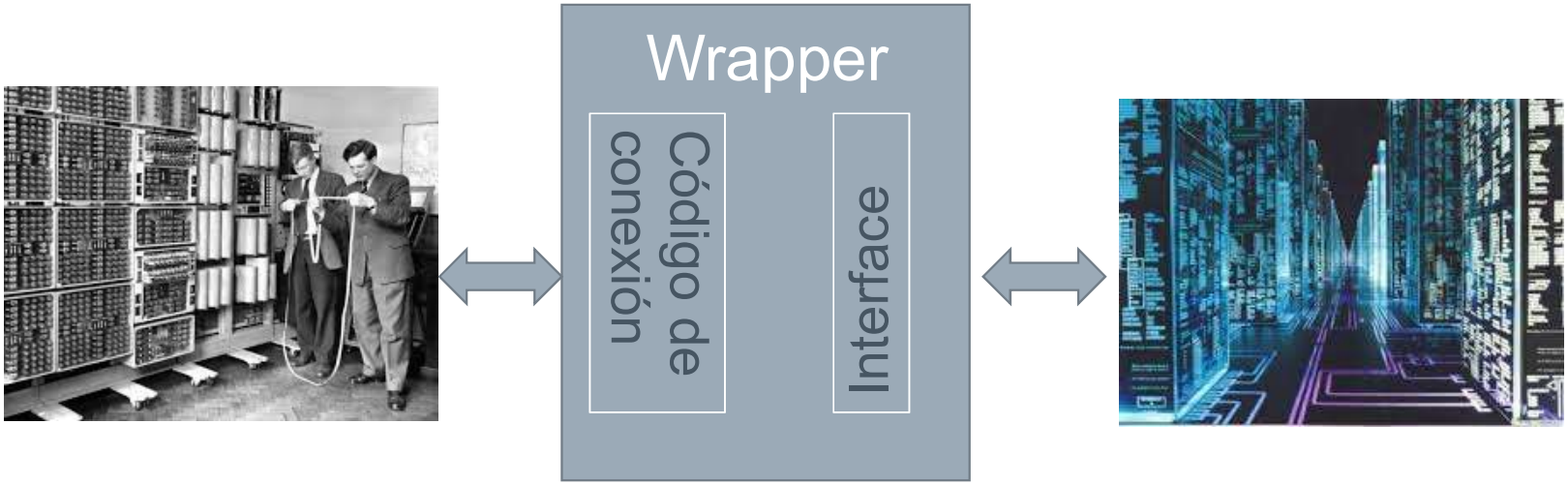
– Clase de abstracción de datos. Encapsulan la estructura de los datos. Ejemplo.



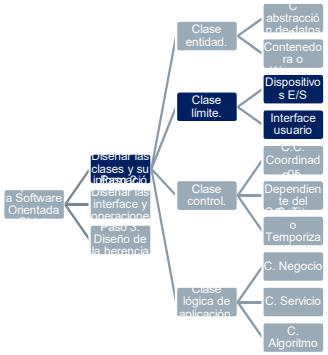
<<data abstraction>> ATMCash
Efectivo Int=0 Cinco: Int=0 Diez: Int=0 Veinte: Int=0 Cicuenta: Int=0
AñadirEfectivo(int cinco, int diez, int veinte, int cincuenta) RetirarEfectivo((int Cantidad, int cinco, int diez, int veinte, int cincuenta)

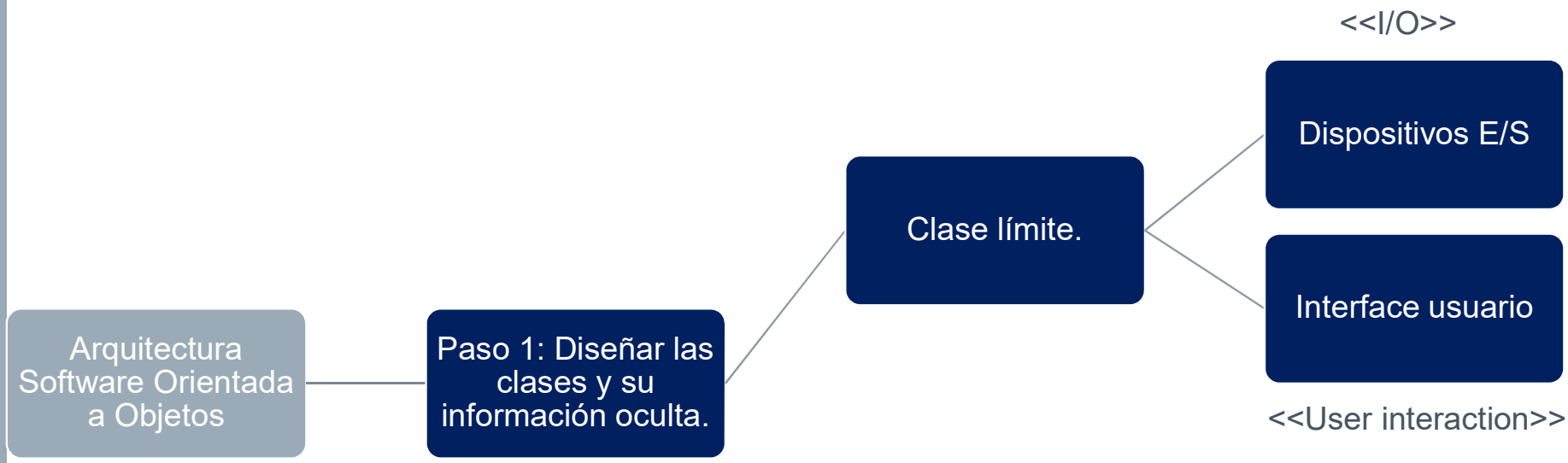
Diseño de Arquitectura Software Orientada a Objetos

– Wrapper clase. (clase contenedora). Oculta los detalles de como interactúa con un sistema existente o con un sistema heredado.



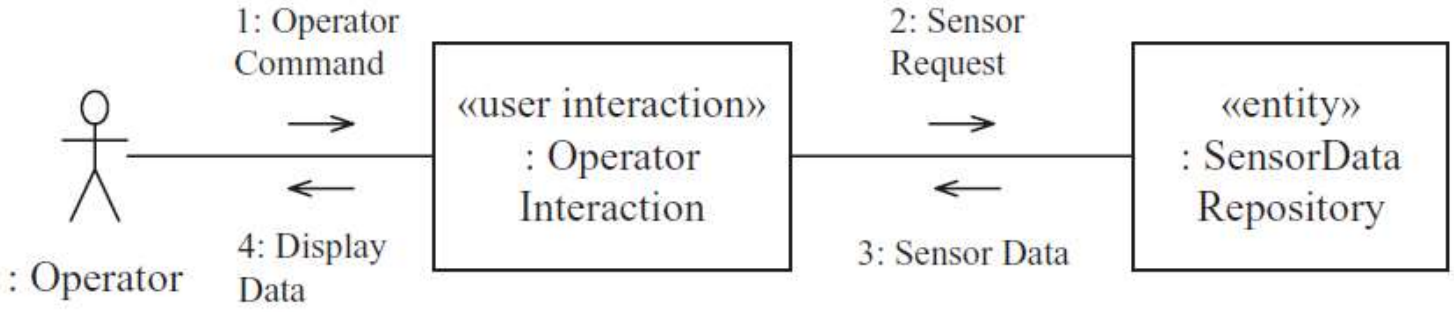
Arquitectura Software Orientada a Objetos





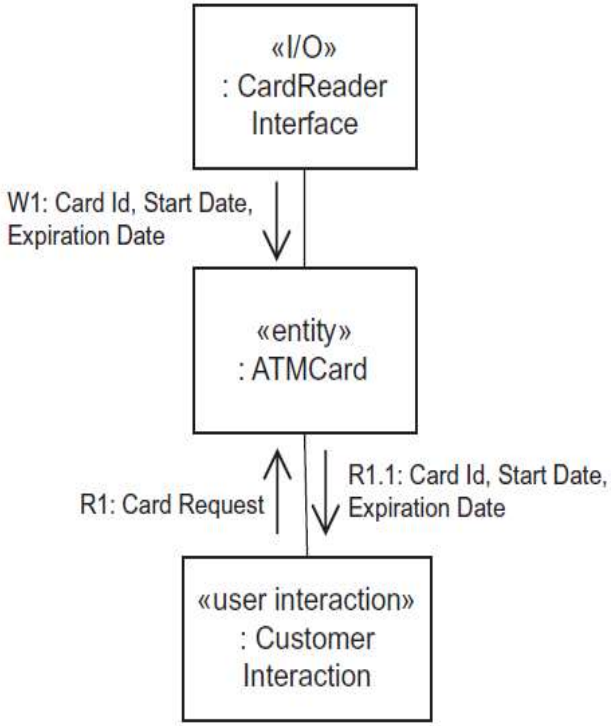
Diseño de Arquitectura Software Orientada a Objetos

- Paso 1. Diseñar las clases y su información oculta.
  - Clase límite. Sirven para comunicarse mediante una interfaz con el medio exterior a la aplicación. Ejemplos, son las que acceden a dispositivos de E/S o las de interacción gráfica con el usuario.

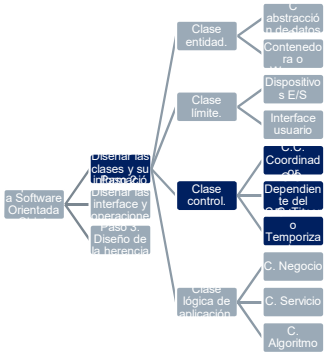


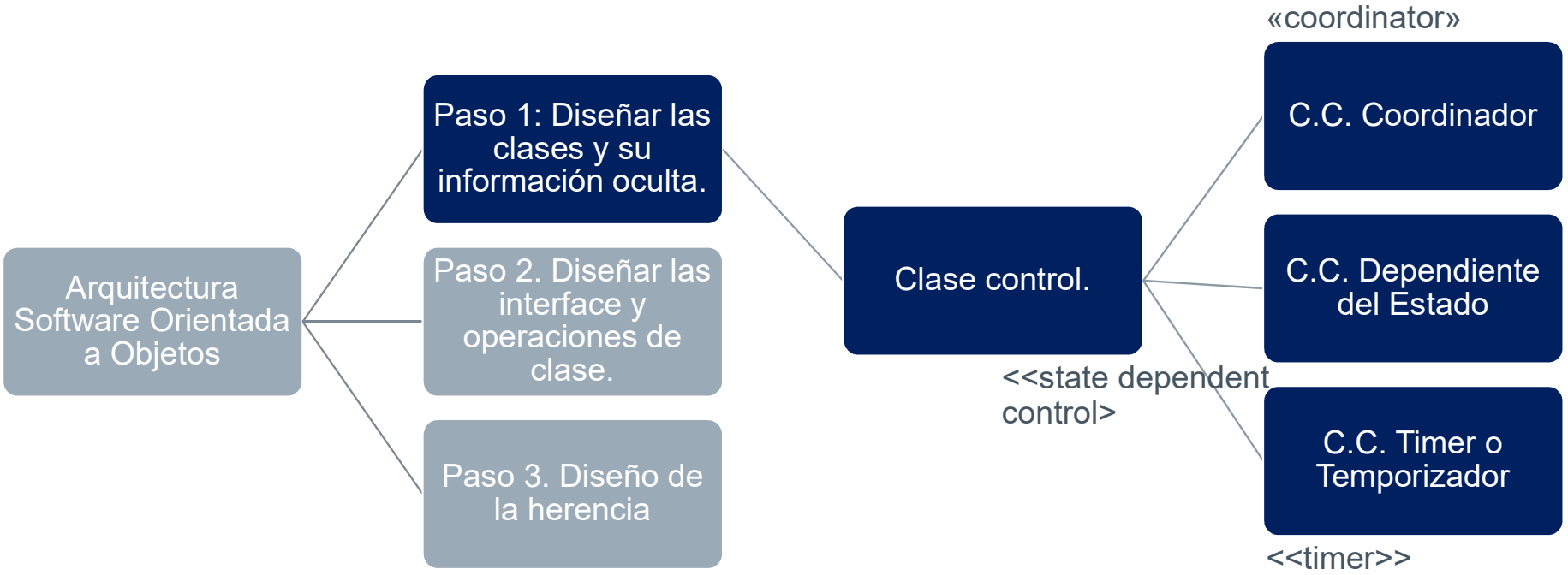
Diseño de Arquitectura Software Orientada a Objetos

- Paso 1. Diseñar las clases y su información oculta.
  - Dispositivos de I/O.
  - Interface de usuario.



Arquitectura Software Orientada a Objetos



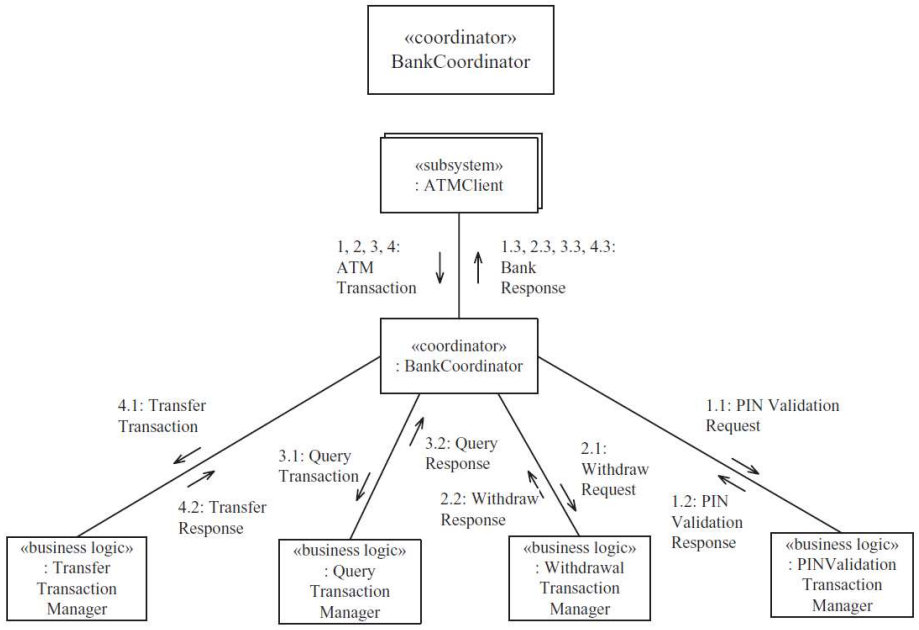


Diseño de Arquitectura Software Orientada a Objetos

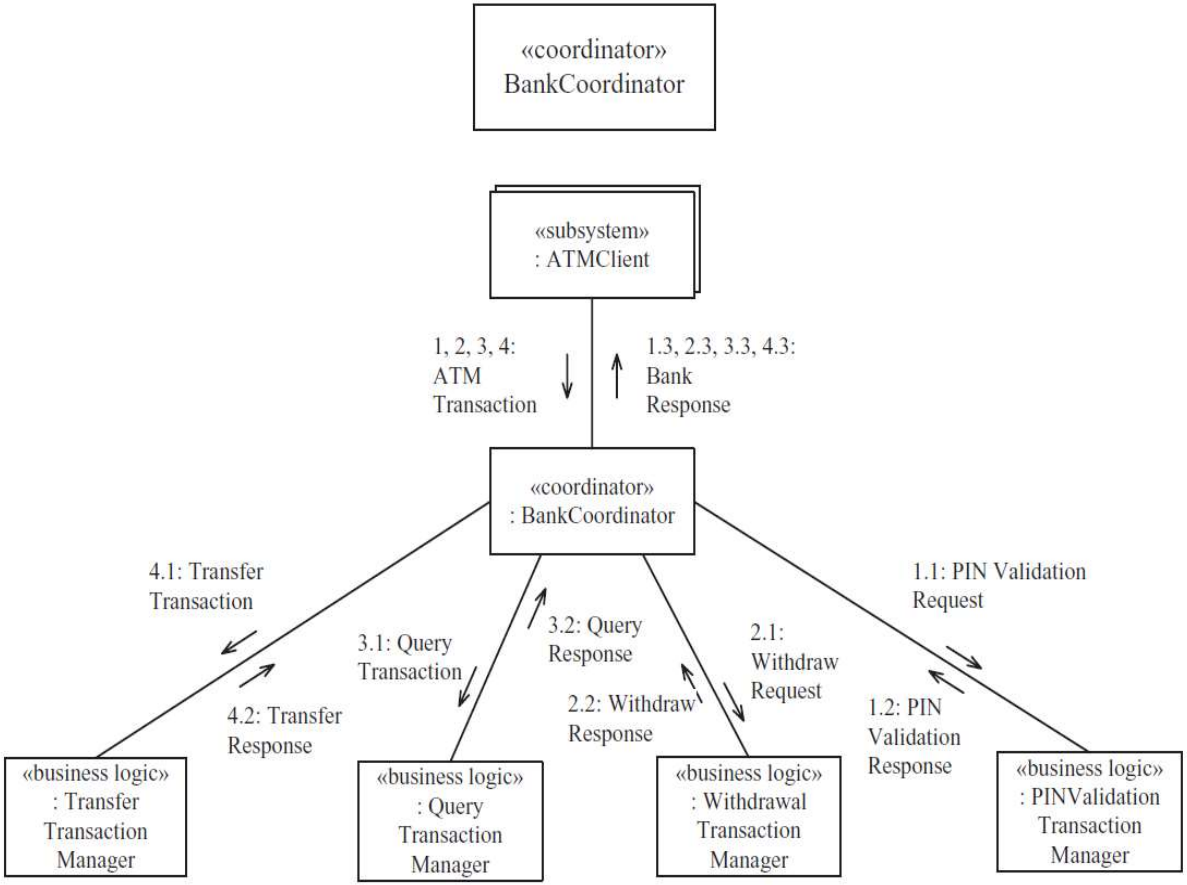
- Paso 1. Diseñar las clases y su información oculta.
  - Clase control. Proporcionar la coordinación general de una colección de objetos.

Diseño de Arquitectura Software Orientada a Objetos

- Paso 1. Diseñar las clases y su información oculta.
  - Clase control. Coordinador. Proporcionar la coordinación general de una colección de objetos. Ejemplo coordinador

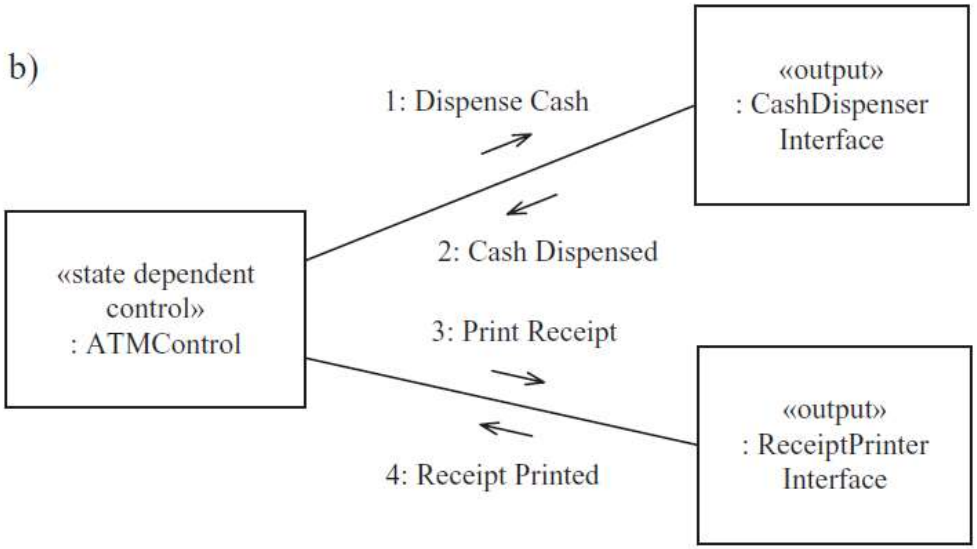


Diseño de Arquitectura Software Orientada a Objetos



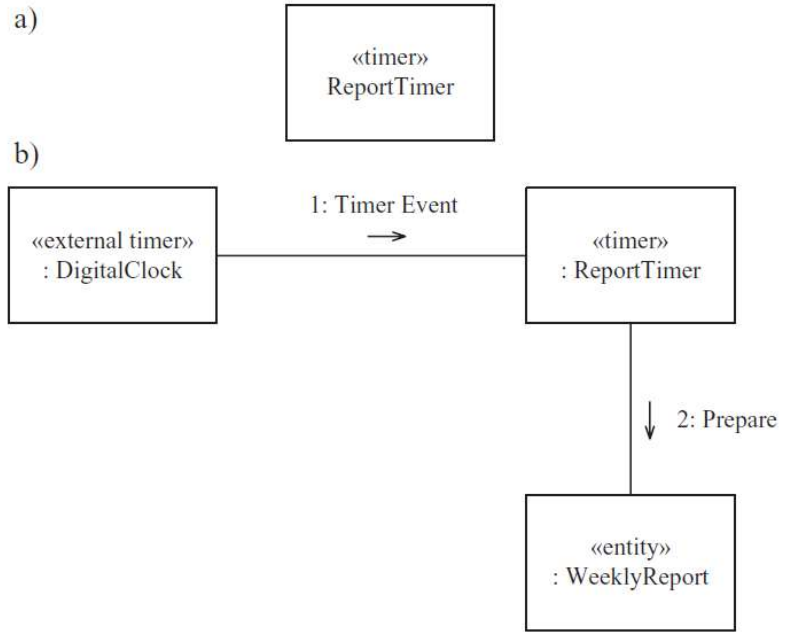
Diseño de Arquitectura Software Orientada a Objetos

- Paso 1. Diseñar las clases y su información oculta.
  - Clase control. Dependiente del estado. Proporcionar la coordinación general de una colección de objetos. Ejemplo Dependiente del estado

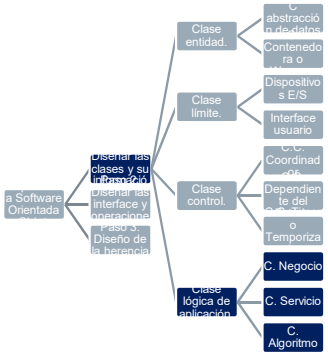


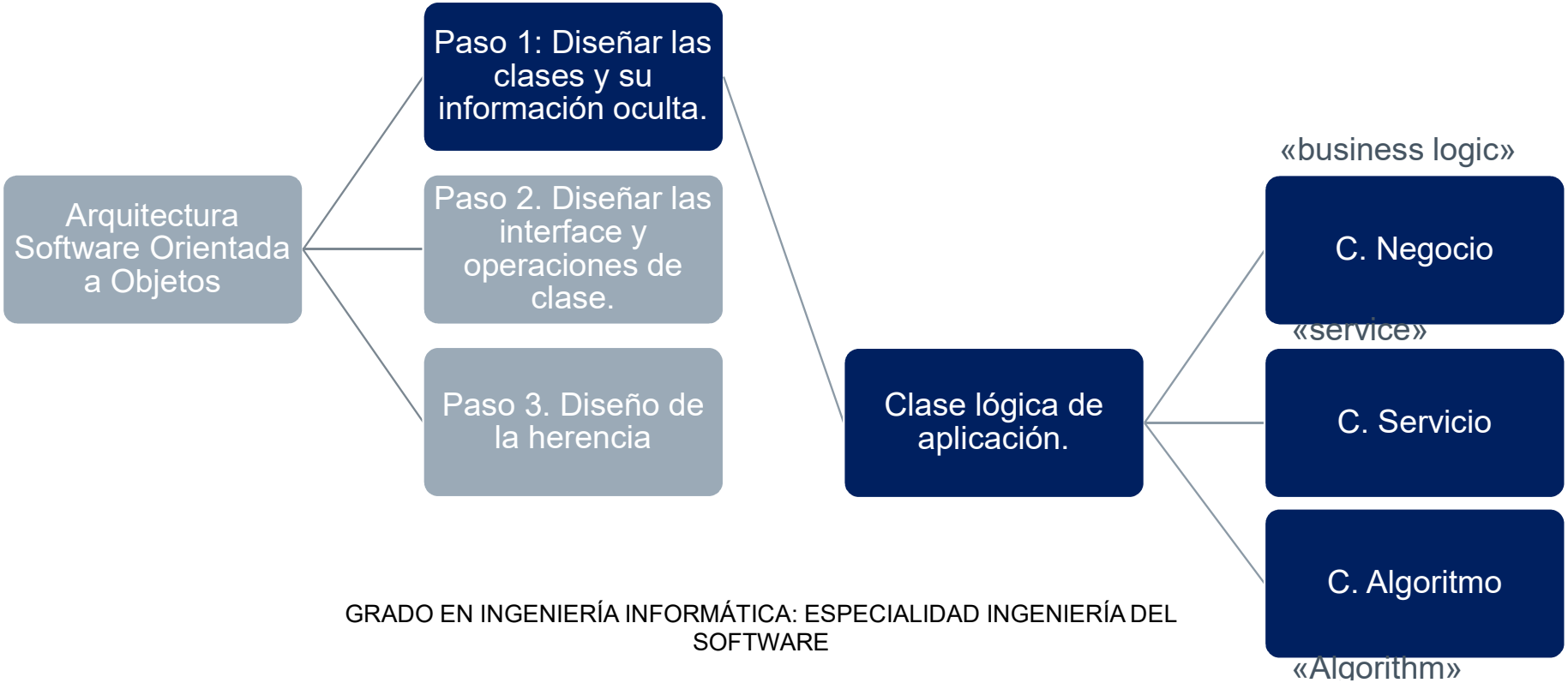
Diseño de Arquitectura Software Orientada a Objetos

- Paso 1. Diseñar las clases y su información oculta.
  - Clase control. Timer. Proporcionar la coordinación general de una colección de objetos. Ejemplo Timer o temporizador



Arquitectura Software Orientada a Objetos



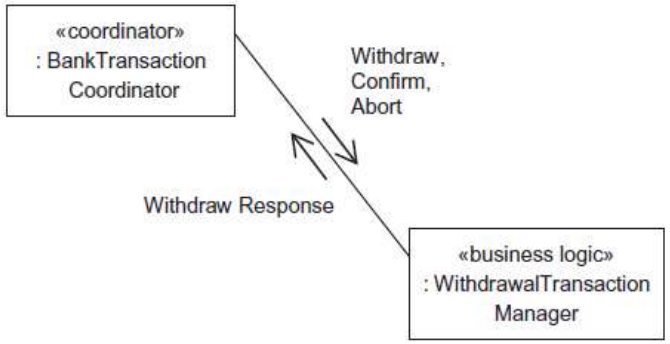


Diseño de Arquitectura Software Orientada a Objetos

- Paso 1. Diseñar las clases y su información oculta.
  - Clases de lógica de aplicación. Encapsular la lógica y algoritmos para aplicaciones específicas. Categorizado en clases de lógica de negocios, clases de servicio, o clases de algoritmos.

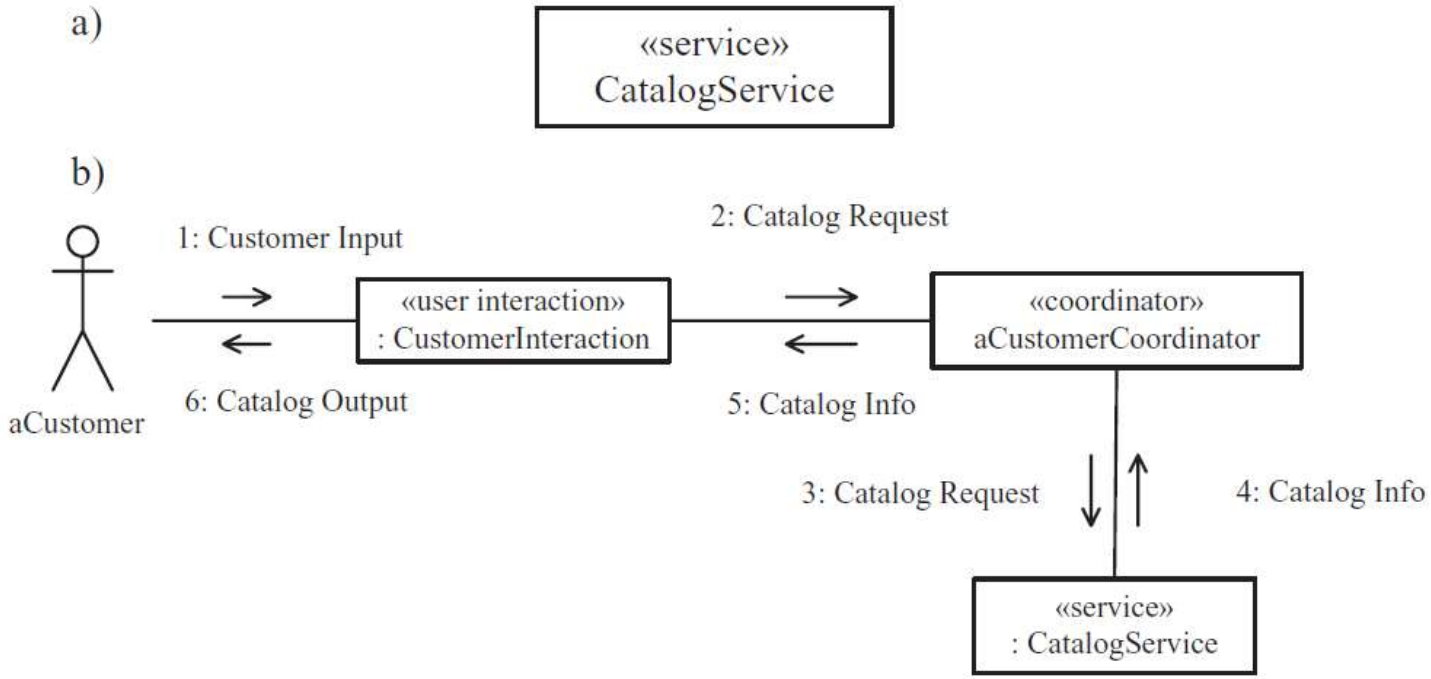
Diseño de Arquitectura Software Orientada a Objetos

– Clase de negocio. Una clase de lógica de negocio define la toma de decisiones, la lógica específica del negocio para procesar una petición de un cliente.



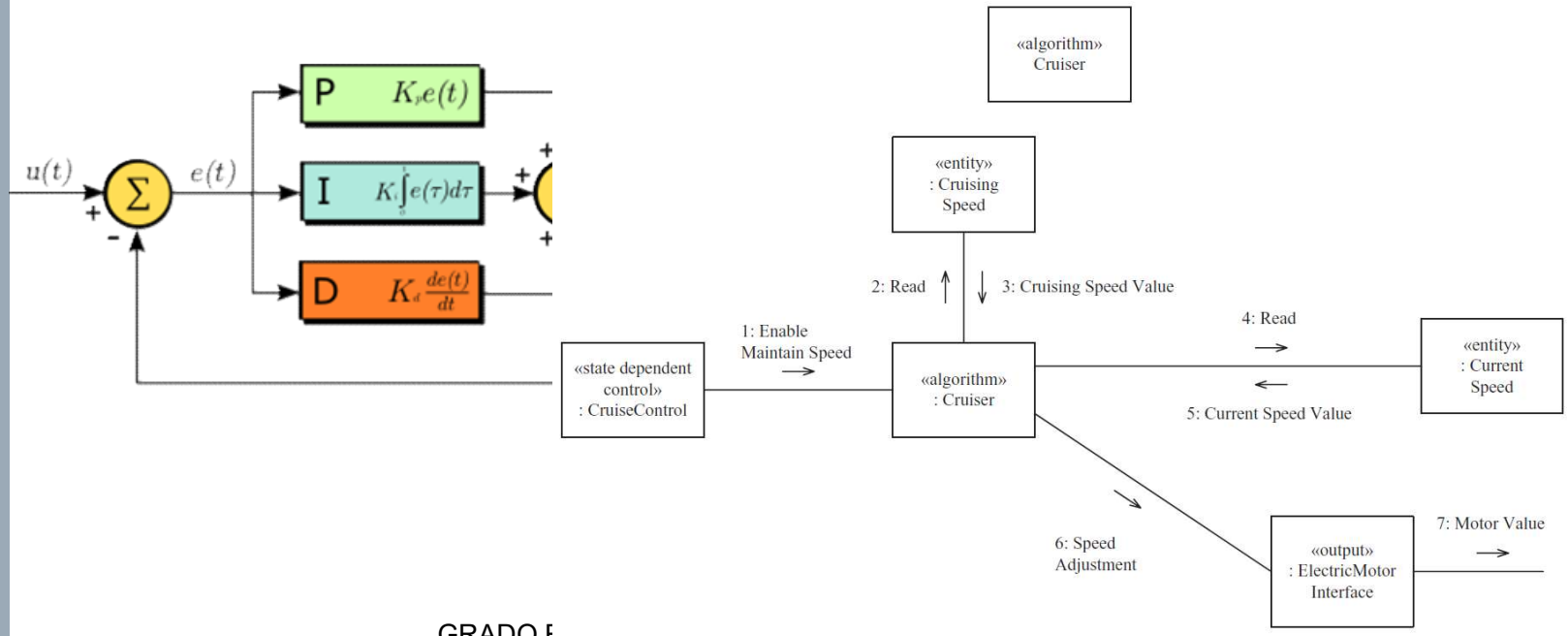
Diseño de Arquitectura Software Orientada a Objetos

– Clase servicio. Es una clase que proporciona un servicio a uno o varios objetos cliente.



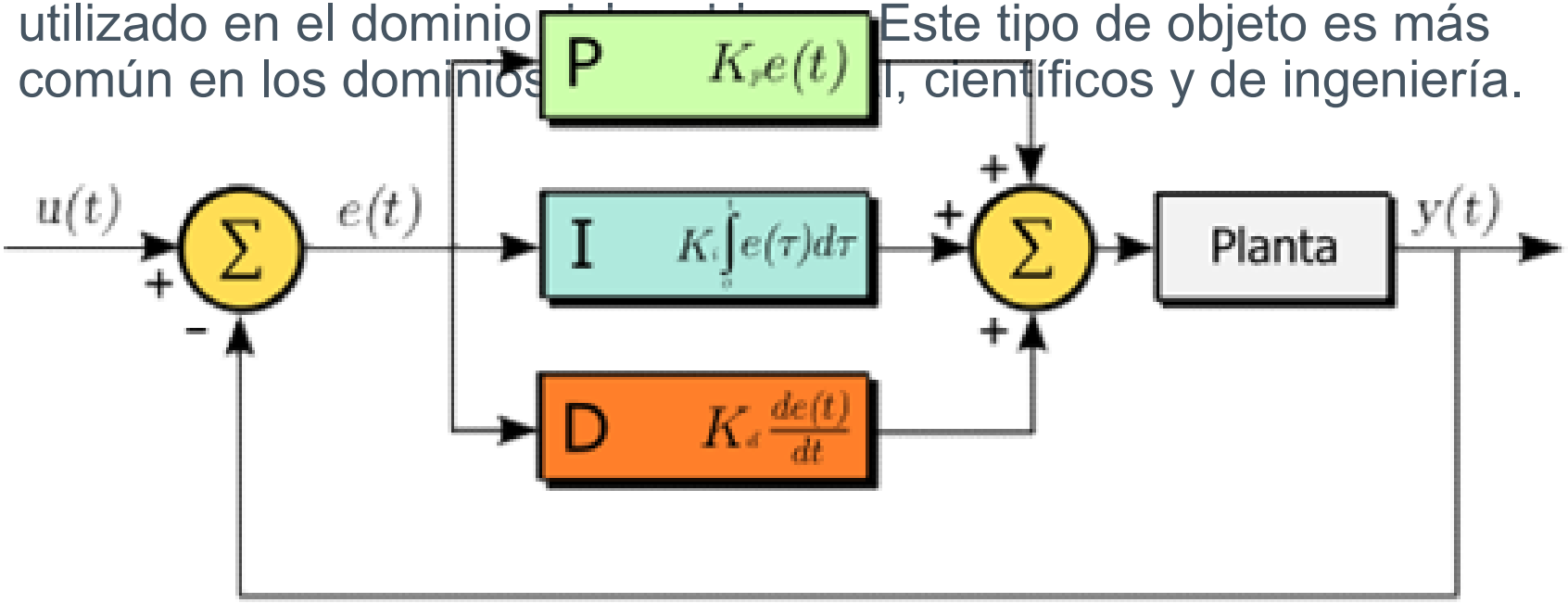
Diseño de Arquitectura Software Orientada a Objetos

– Clase algoritmo. Un objeto algoritmo encapsula un algoritmo utilizado en el dominio del problema. Este tipo de objeto es más común en los dominios de tiempo real, científicos y de ingeniería.

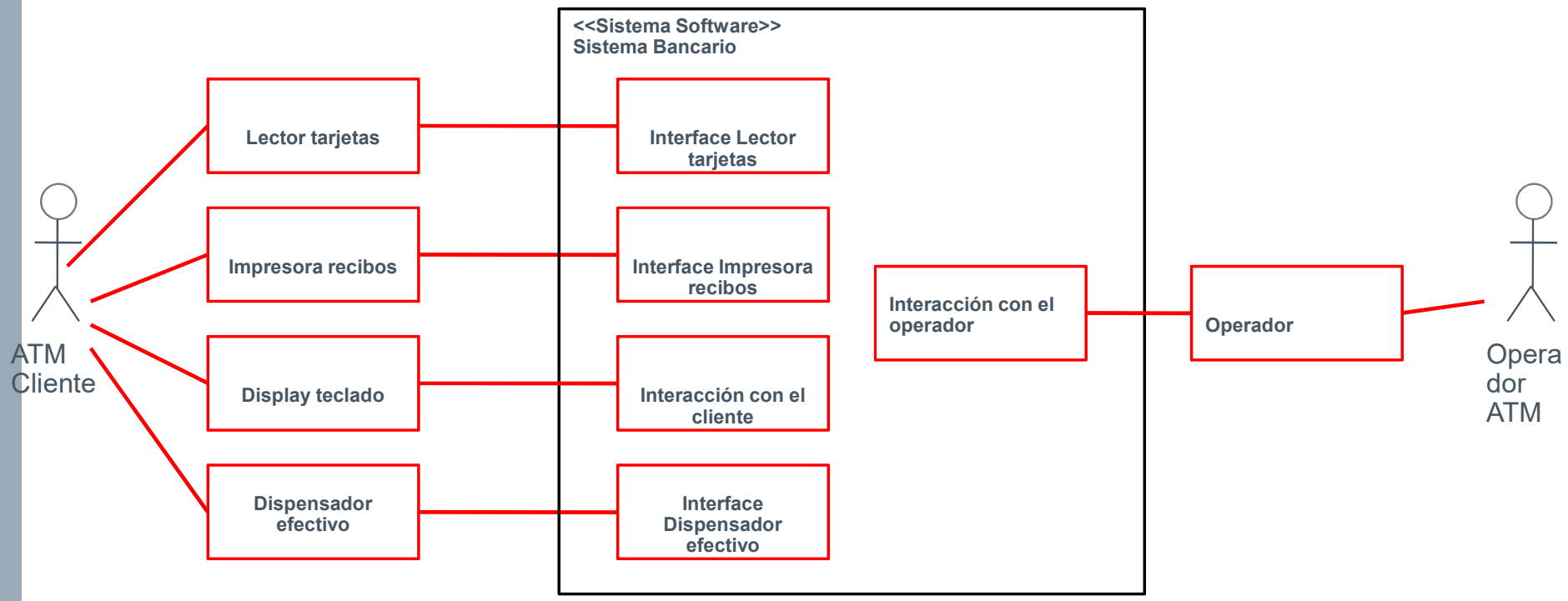


Diseño de Arquitectura Software Orientada a Objetos

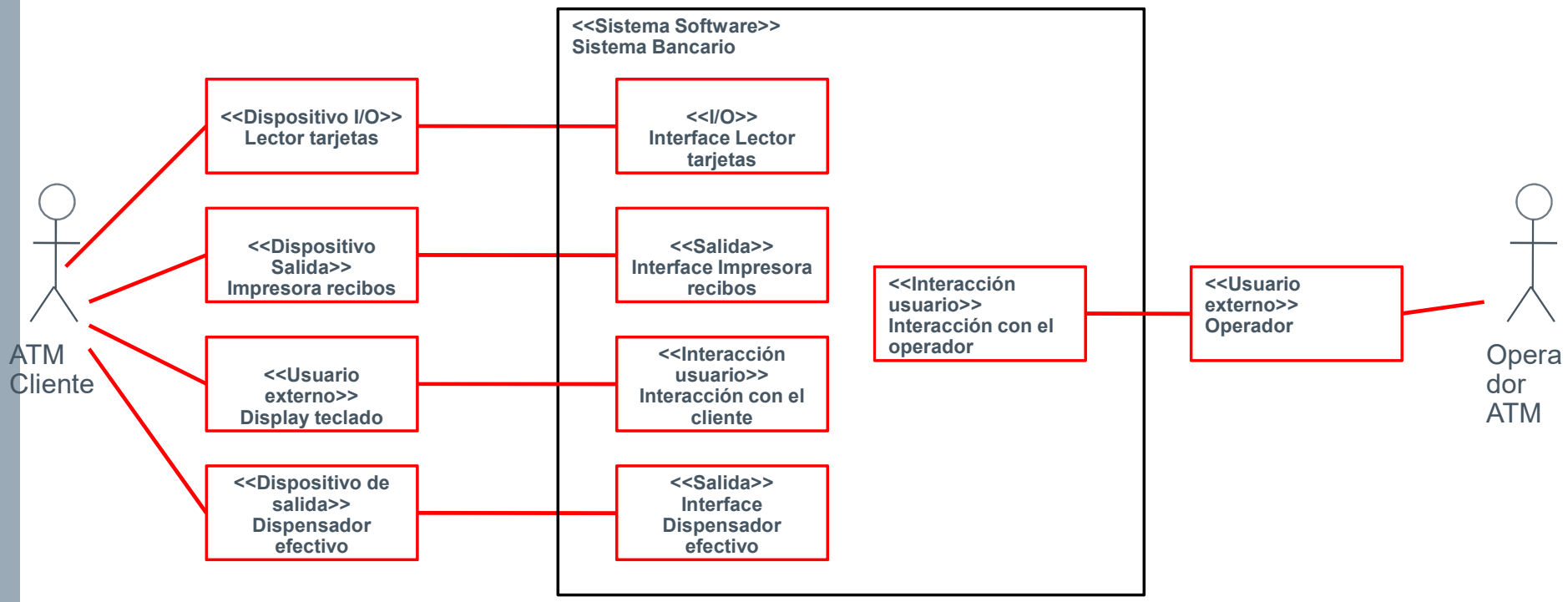
– Clase algoritmo. Un objeto algoritmo encapsula un algoritmo utilizado en el dominio [illegible]. Este tipo de objeto es más común en los dominios [illegible], científicos y de ingeniería.



Diseño de Arquitectura Software Orientada a Objetos

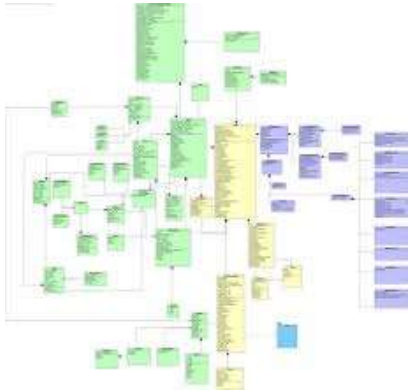


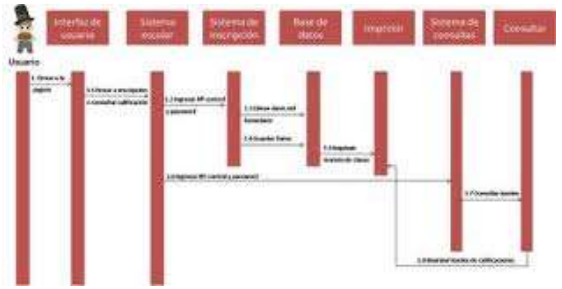
Diseño de Arquitectura Software Orientada a Objetos



Diseño de Arquitectura Software Orientada a Objetos

› Paso 2. Diseñar las interface y operaciones de clase.



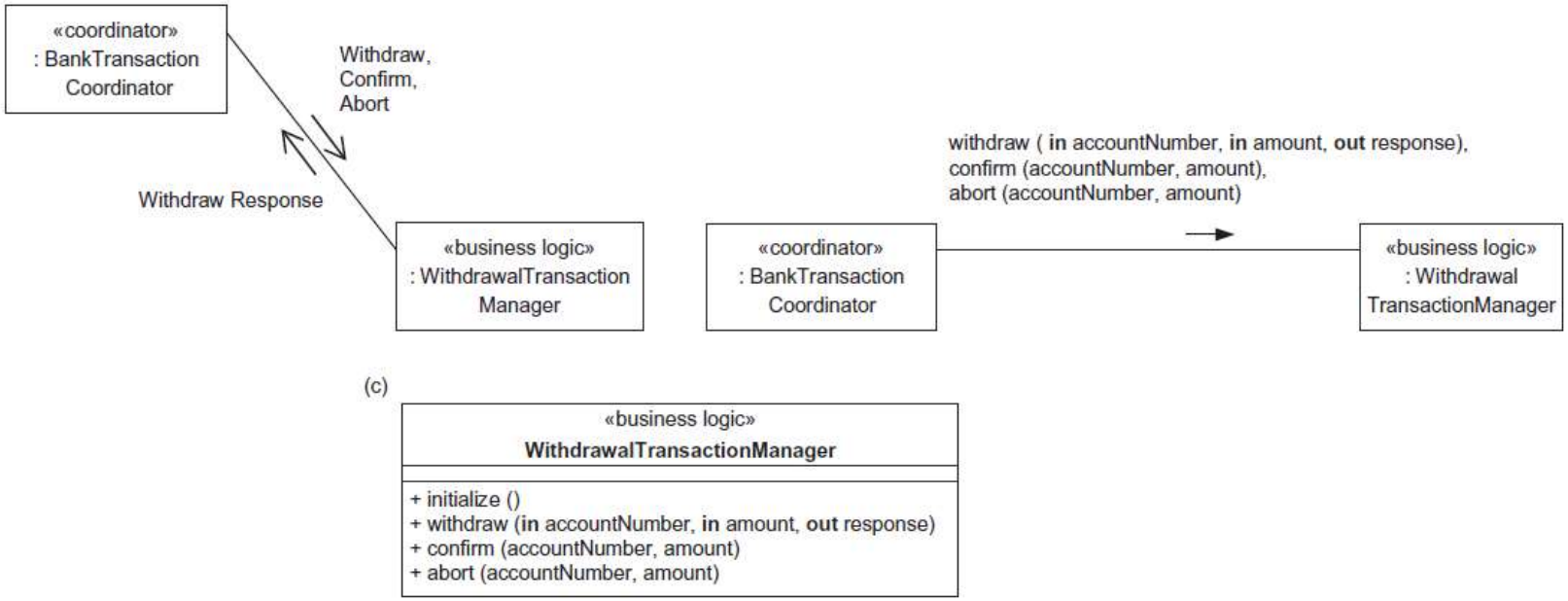


Diseño de Arquitectura Software Orientada a Objetos

- Paso 2. Diseñar las interface y operaciones de clase.
- Determinar operaciones a partir del modelo dinámico. Si los objetos son mapeados a un programa secuencial en los diagramas de comunicación el objeto emisor invoca una operación en el objeto receptor. El nombre del mensaje se asigna al nombre de la operación y los parámetros del mensaje se asignan a los parámetros de la operación. Es necesario determinar si las operaciones tienen parámetros de E/S.

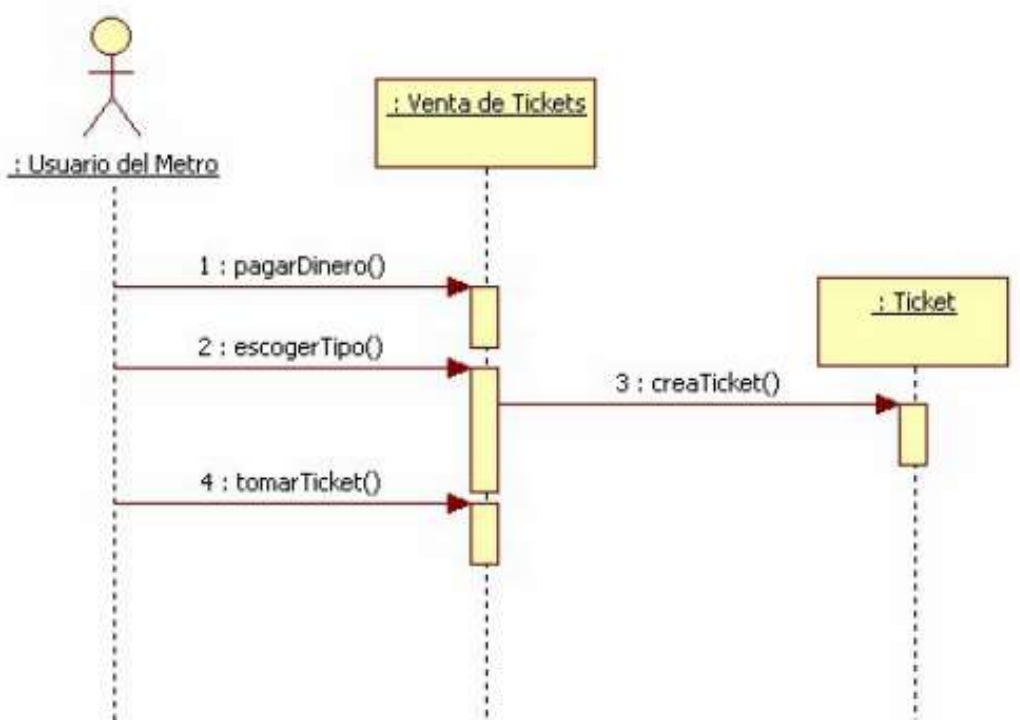
Diseño de Arquitectura Software Orientada a Objetos

› Paso 2. Diseñar las interface y operaciones de clase.



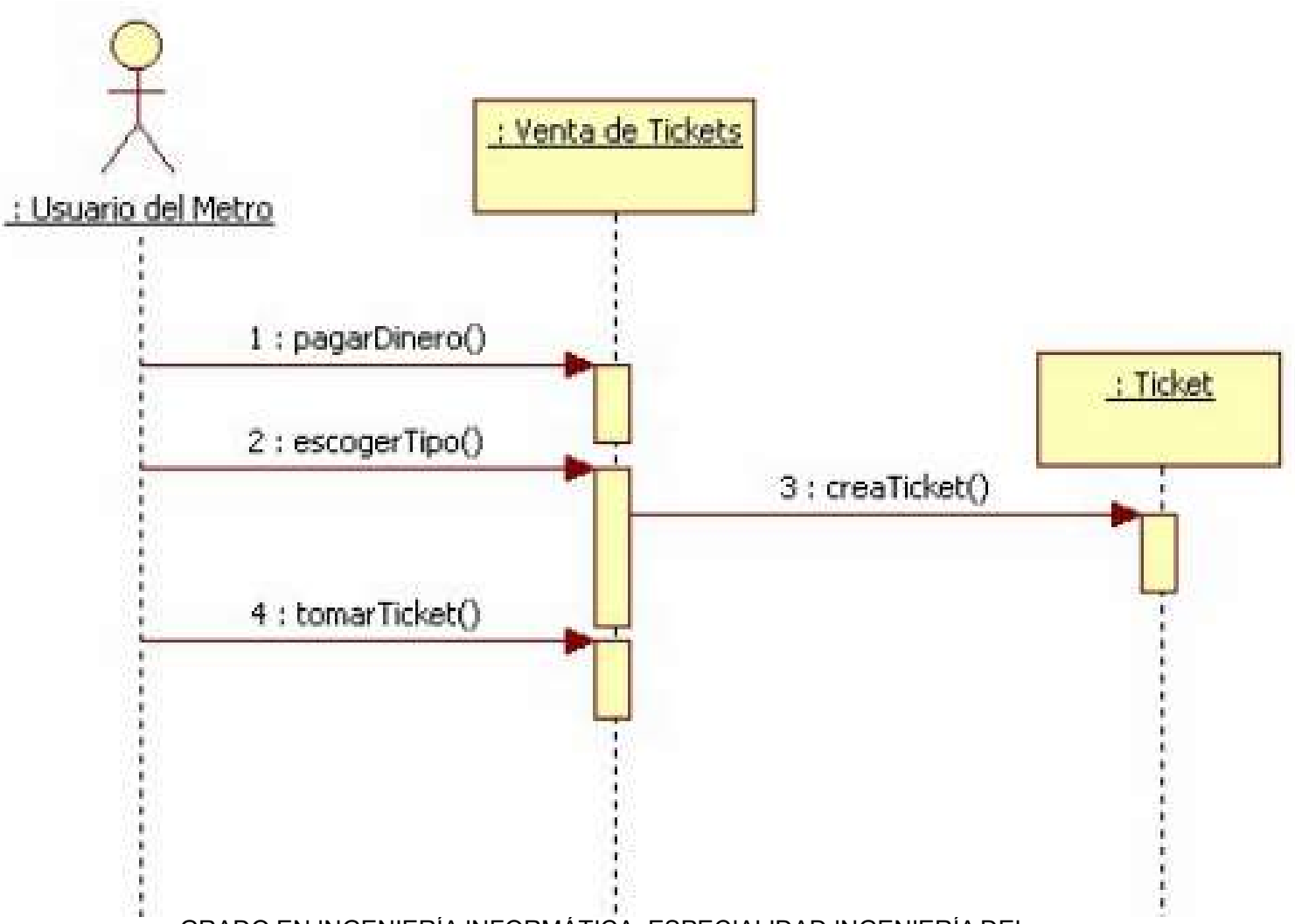
Diseño de Arquitectura Software Orientada a Objetos

› Paso 2. Diseñar las interface y operaciones de clase.



Dis… Ob…

› Pa…


: Usuario del Metro
: Venta de Tickets
: Ticket
1 : pagarDinero()
2 : escogerTipo()
3 : creaTicket()
4 : tomarTicket()

Diseño de Arquitectura Software Orientada a Objetos

- Paso 2. Diseñar las interface y operaciones de clase. Ejercicio: Realizar el diagrama de secuencia de una identificación de usuario.

Diseño de Arquitectura Software Orientada a Objetos

- Paso 2. A partir del modelo estático también se pueden diseñar las operaciones de clases, en particular las de actualización (crear, eliminar, modificar)

Diseño de Arquitectura Software Orientada a Objetos

- Paso 3. Diseño de la herencia. La herencia puede ser utilizada en el diseño de dos clases similares, pero no idénticas, - en otras palabras, las clases que comparten muchas, pero no todas, las características.
- Las jerarquías de clase (también denominadas como jerarquías de generalización/especialización y jerarquías de herencia) se pueden desarrollar ya sea de forma top-down o botton-up, o por alguna combinación de los dos enfoques.
- Cabe señalar que cuando se diseña con la herencia, la parte interna de las clases padres son visibles para las subclases.

Diseño de Arquitectura Software Orientada a Objetos

- Clases abstractas. Clase sin instancias. Por lo tanto, sólo se usa como una superclase y define una interfaz común para sus subclases.
- Una operación abstracta es una operación que se declara en una clase abstracta, pero no se ha implementado. Una clase abstracta debe tener al menos una operación abstracta.
- Una clase abstracta difiere en todas o algunas de las implementaciones de operación con respecto a las subclases. Las diferentes subclases de la misma clase abstracta puede definir implementaciones distintas de la misma operación abstracta.

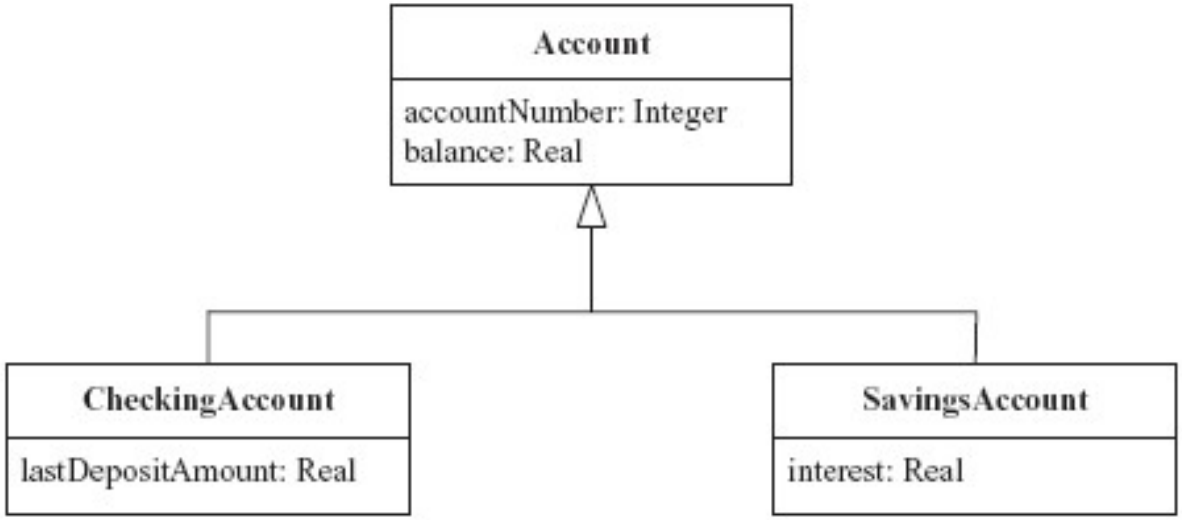
Diseño de Arquitectura Software Orientada a Objetos

- Algunas de las operaciones se pueden implementar en la clase abstracta, especialmente en los casos en los que algunas o todas las subclases necesiten utilizar la misma operación.
- **Polimorfismo y vinculación dinámica.** El polimorfismo se utiliza para indicar que diferentes clases pueden tener operaciones con el mismo nombre. La especificación de la operación es idéntica para cada clase, sin embargo, las clases implementan las operaciones de manera diferente. Esto permite que los objetos con interfaces idénticas puedan ser sustituidos uno por el otro en tiempo de ejecución.
- **Vinculación dinámica o enlace dinámico** se utiliza en conjunción con el polimorfismo y es la asociación en tiempo de ejecución de una petición a un objeto y una de sus operaciones. La vinculación dinámica consiste en que la asociación de una solicitud con un objeto se realiza en tiempo de ejecución y por lo tanto puede cambiar entre una invocación y la siguiente.
-

Ejemplo

› Este ejemplo es de un sistema bancario en el que existen distintas cuentas: En un principio, cuentas corrientes y cuentas de ahorro, podría diseñarse una clase abstracta llamada Cuenta, que encapsula los dos atributos generales que son necesarios para todas las cuentas: AccountNumber y balance.

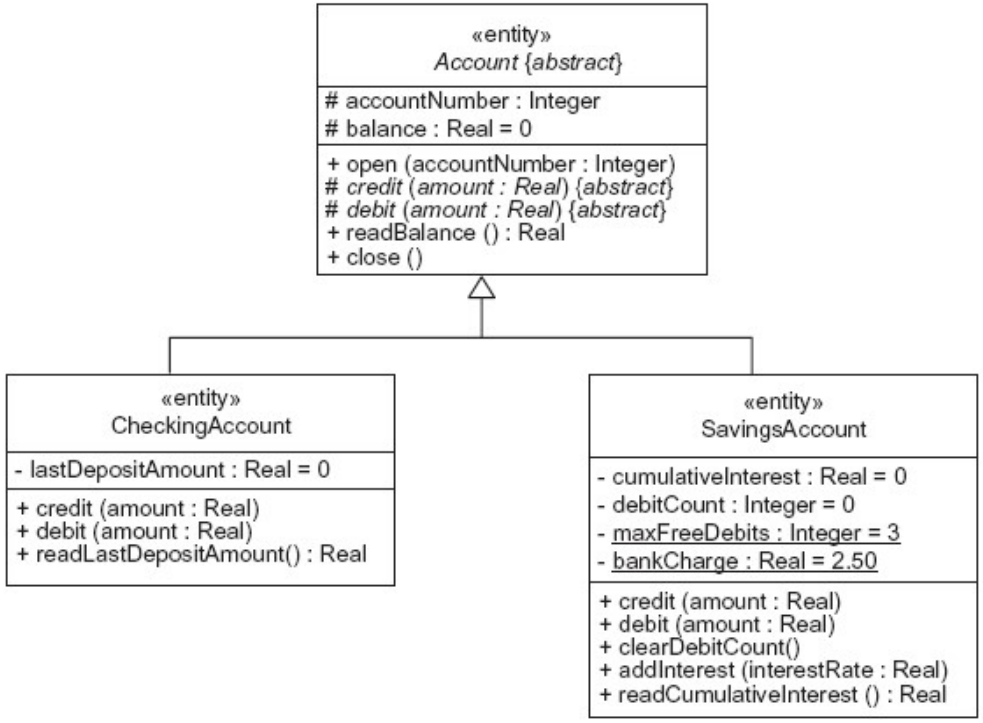
Ejemplo


Account
accountNumber: Integer
balance: Real
CheckingAccount
lastDepositAmount: Real
SavingsAccount
interest: Real

Ejemplo

› Además como en ambos casos será necesario abrir, cerrar cuentas, leer saldo de la cuenta, crédito y débito de las cuentas, se pueden generalizar dichas operaciones.

Ejemplo


«entity»
Account {abstract}
accountNumber : Integer
balance : Real = 0
+ open (accountNumber : Integer)
# credit (amount : Real) {abstract}
# debit (amount : Real) {abstract}
+ readBalance () : Real
+ close ()
«entity»
CheckingAccount
- lastDepositAmount : Real = 0
+ credit (amount : Real)
+ debit (amount : Real)
+ readLastDepositAmount() : Real
«entity»
SavingsAccount
- cumulativeInterest : Real = 0
- debitCount : Integer = 0
- maxFreeDebits : Integer = 3
- bankCharge : Real = 2.50
+ credit (amount : Real)
+ debit (amount : Real)
+ clearDebitCount()
+ addInterest (interestRate : Real)
+ readCumulativeInterest () : Real