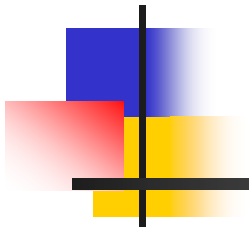
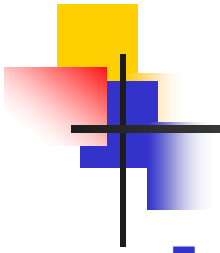


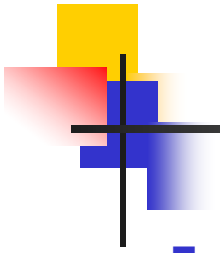
Introducción al Diseño con Patrones

Fernando Bellas Permuy
Departamento de Tecnologías de la Información y las Comunicaciones (TIC)
Universidad de A Coruña
http://www.tic.udc.es/~fbellas
fbellas@udc.es



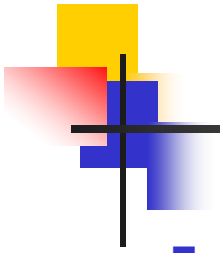
Índice

- Introducción
- Tipos de patrones
- Descripción de patrones
- Caso de estudio: diseño de una aplicación Web bancaria
- Unas palabras finales
- Referencias



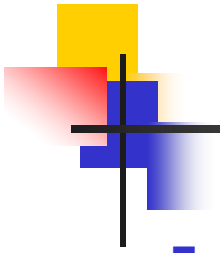
Introducción (1)

- Diseñar software orientado a objetos es difícil
- Diseñar software reusable orientado a objetos es todavía más difícil
- Es difícil que un diseñador inexperto sea capaz de hacer un buen diseño
  - Conoce los principios básicos de la orientación a objetos (herencia, polimorfismo, etc.) y un lenguaje orientado a objetos
  - Pero no sabe cómo usar esos conocimientos de manera eficaz



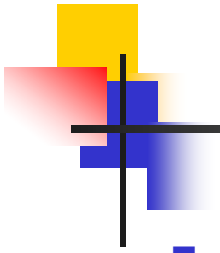
Introducción (2)

- Sin embargo, un diseñador experto es capaz de hacer buenos diseños
  - Reusan soluciones de diseño que les han funcionado bien en el pasado para resolver problemas similares
- Un **patrón** es una solución a un problema de diseño no trivial que es
  - Efectiva: ha valido para resolver el problema en diseños pasados
  - Reusable: la solución es la misma para problemas similares



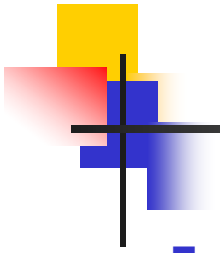
Introducción (3)

- Eric Gamma, Richard Helm, Ralph Johnson y John Vlissides publicaron el primer catálogo de patrones en el ámbito del software en 1994
  - *Design Patterns: Elements of Reusable Object-Oriented Software*
  - También conocido como *GoF* ("Gang of Four")
  - Y desde entonces, se han publicado muchos otros catálogos
  - Fue la primera vez que se documentó el conocimiento que usaban los expertos para resolver problemas de diseño
  - Se inspiraron en el trabajo de Christopher Alexander, que había documentado patrones en el ámbito de la arquitectura



Introducción (y 4)

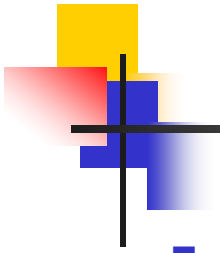
- Ventajas del diseño con patrones
  - Permiten reusar soluciones probadas
  - Facilitan la comunicación entre diseñadores
    - Los patrones tienen nombres estándar
  - Facilitan el aprendizaje al diseñador inexperto



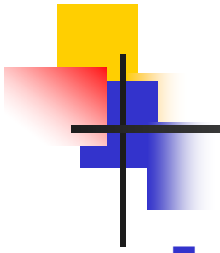
Tipos de patrones (1)

- Existen patrones para las distintas actividades que hay que realizar en un proyecto (análisis, diseño, implementación, etc.)
  - **Este seminario se centra en los relativos al diseño**
- Existen patrones independientes del dominio y otros específicos a un dominio concreto
  - Ej.: Los patrones del GoF son independientes del dominio
  - Ej.: Los patrones publicados en http://java.sun.com/blueprints/patterns/index.html son aplicables al desarrollo de aplicaciones empresariales (persistencia, tecnologías Web, etc.) con J2EE (la plataforma empresarial de Java)

Tipos de patrones (2)

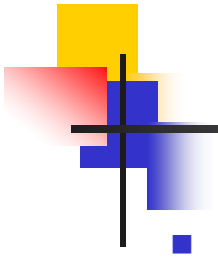


- *Pattern-Oriented Software Architecture: A System of Patterns*
  - También conocido como *The POSA Book*
  - Fue el primer libro en hacer una clasificación de patrones
    - Patrones arquitectónicos [Diseño]
    - Patrones de diseño [Diseño]
    - Idiomas [Implementación]
- Patrones arquitectónicos
  - Aconsejan la arquitectura global que debe seguir una aplicación
  - Ej.: El patrón Model-View-Controller (MVC) aconseja la arquitectura global que debe tener una aplicación interactiva



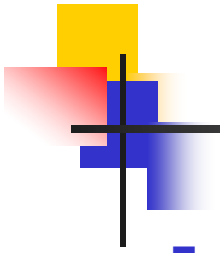
Tipos de patrones (y 3)

- Patrones de diseño
  - Explican cómo resolver un problema concreto de diseño
  - El patrón "Data Access Object" (DAO) permite abstraer y encapsular los accesos a un repositorio de datos (ej.: BD relacional, BD orientada a objetos, ficheros planos, etc.)
- Idiomas
  - Explican cómo resolver un problema particular de implementación con una tecnología concreta
  - Ej.: ¿Cómo comparar objetos correctamente en Java?
    - Correcta redefinición de **`equals()`** y **`hashCode()`**



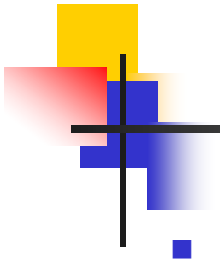
Descripción de patrones

- Cada catálogo de patrones utiliza una plantilla para describir sus patrones
  - No existen plantillas estándar
- Ejemplo: plantilla para patrones de diseño [GoF]
  - Nombre y clasificación (creación, estructura o comportamiento)
  - Intención
  - También conocido como
  - Motivación
  - Aplicabilidad
  - Estructura
  - Participantes
  - Colaboraciones
  - Consecuencias
  - Implementación
  - Código de ejemplo
  - Usos conocidos
  - Patrones relacionados



Caso de estudio: diseño de una aplicación Web bancaria (1)

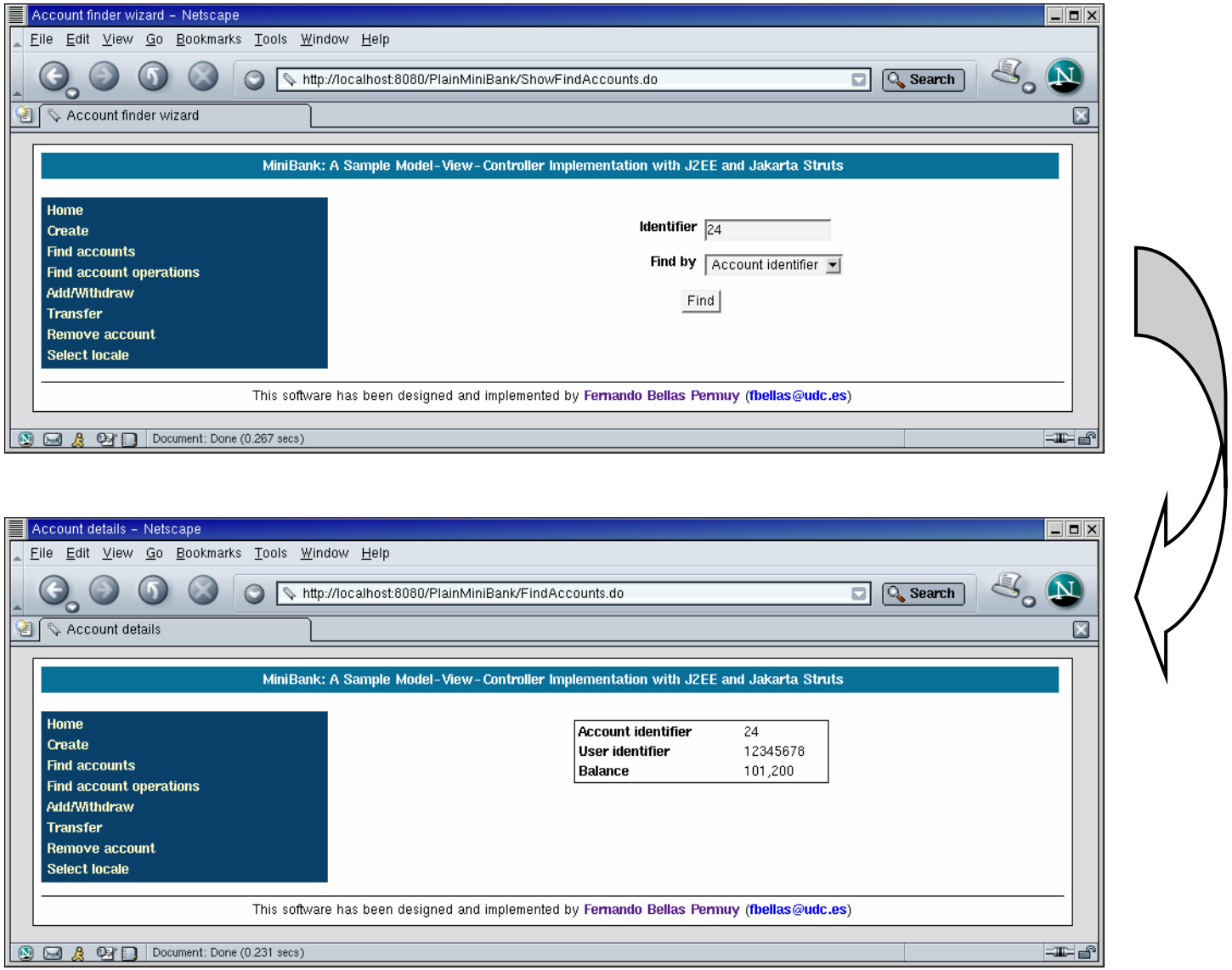
- Caso de estudio: MiniBank
  - Un sencillo ejemplo de una aplicación Web bancaria
- ¿Cuáles son los objetivos?
  - Ver ejemplos de patrones aplicados a un caso
    - Patrones arquitectónicos: MVC
    - Patrones de diseño: Transfer Object, DAO, Factory, Facade y Command
  - Obtener una visión global de los principios básicos de diseño de una aplicación Web con J2EE y Jakarta Struts
  - **Lo más importante: captar la idea de lo que significa diseñar con patrones**
- ¿Cuáles **no** son los objetivos?
  - Estudiar el diseño **completo** de MiniBank
  - Obtener una visión **detallada** de los principios básicos de diseño de una aplicación Web con J2EE y Jakarta Struts



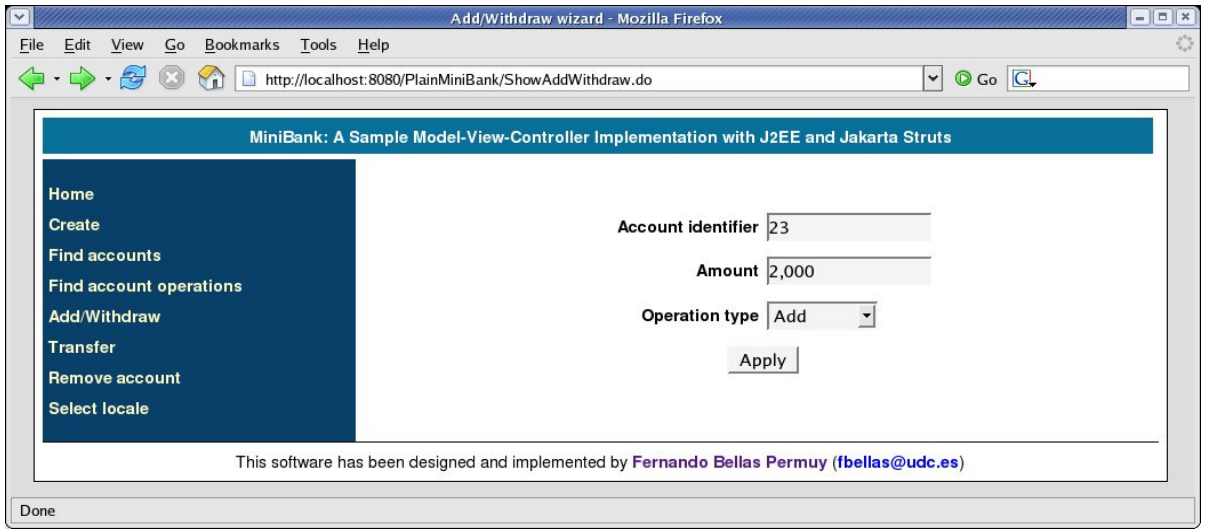
Caso de estudio: diseño de una aplicación Web bancaria (2)
Casos de uso implementados en MiniBank

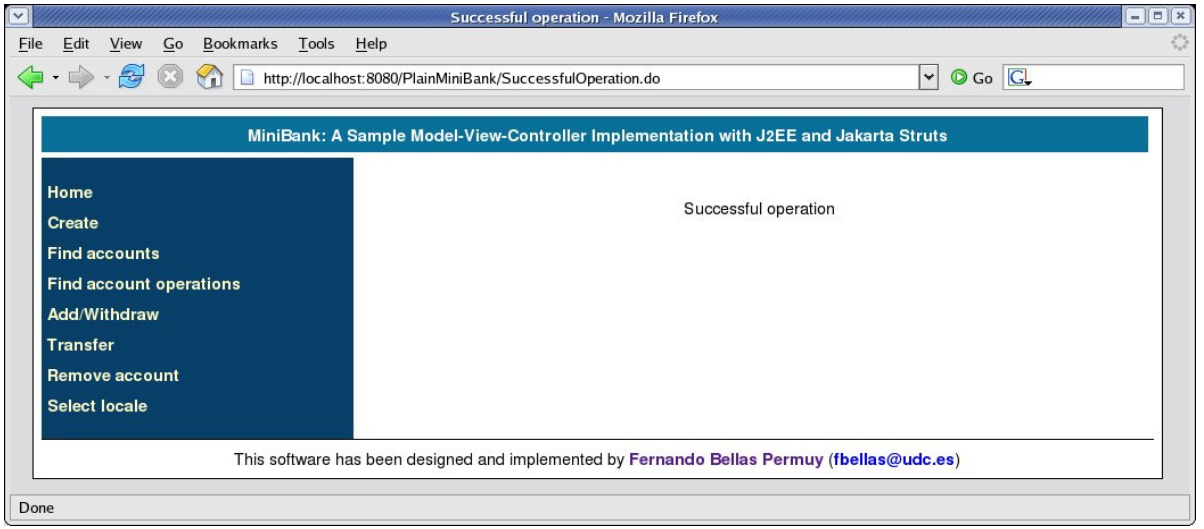
- Crear una cuenta bancaria
  - Datos de una cuenta: identificador de la cuenta, identificador del usuario, balance
  - El identificador de la cuenta se genera automáticamente
- Buscar una cuenta a partir de su identificador
- Añadir una cantidad de dinero a una cuenta
- Retirar una cantidad de dinero de una cuenta
- Buscar todas las cuentas de un usuario
- Eliminar una cuenta
- Hacer una transferencia de dinero de una cuenta a otra
- Buscar todas las operaciones que se han hecho sobre una cuenta entre dos fechas
  - Datos de una operación: identificador de la operación, identificador de la cuenta, fecha, tipo (añadir/retirar), cantidad de dinero

Caso de estudio: diseño de una aplicación Web bancaria (3)
MiniBank: búsqueda de una cuenta a partir de su identificador

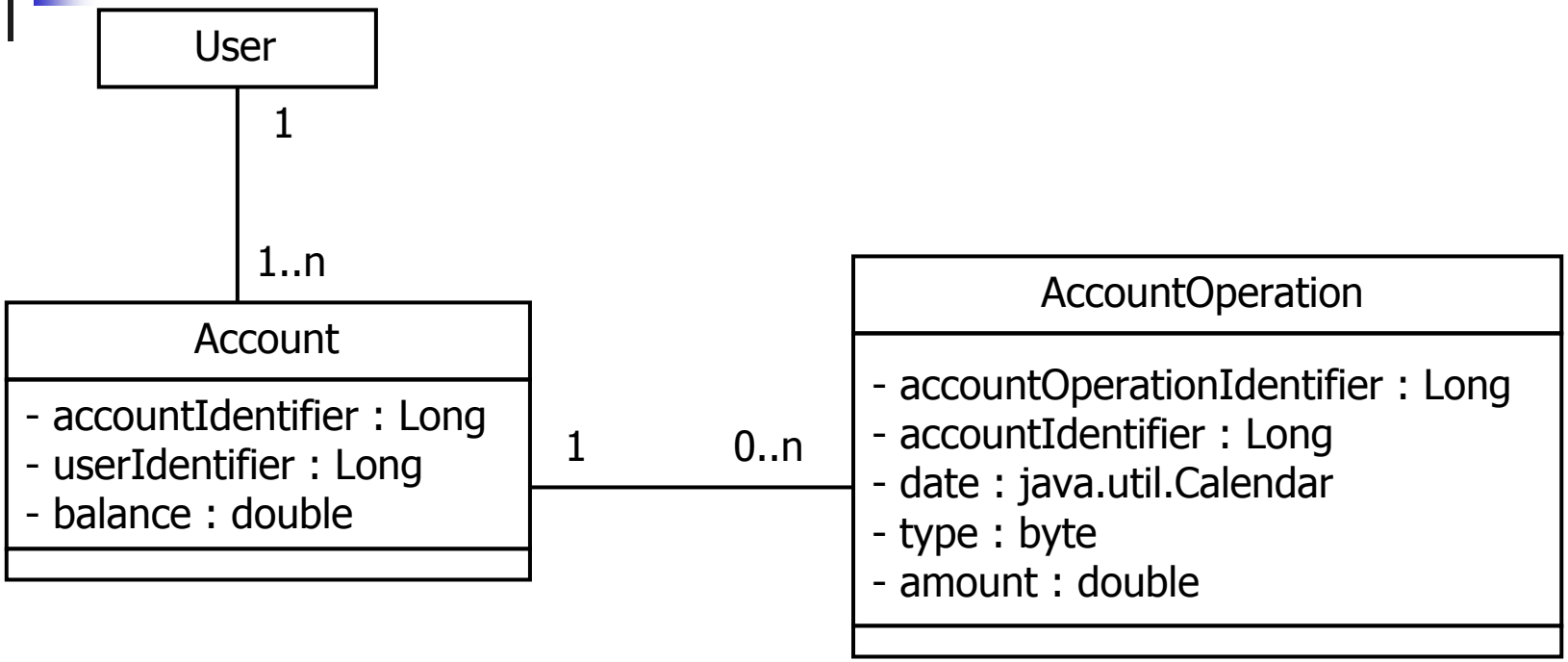


Caso de estudio: diseño de una aplicación Web bancaria (4)
MiniBank: añadir dinero a una cuenta

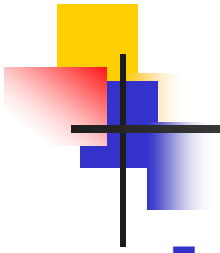




Caso de estudio: diseño de una aplicación Web bancaria (5)
Objetos del dominio



- NOTA:
  - Un objeto del dominio es uno que corresponde a un concepto del mundo real

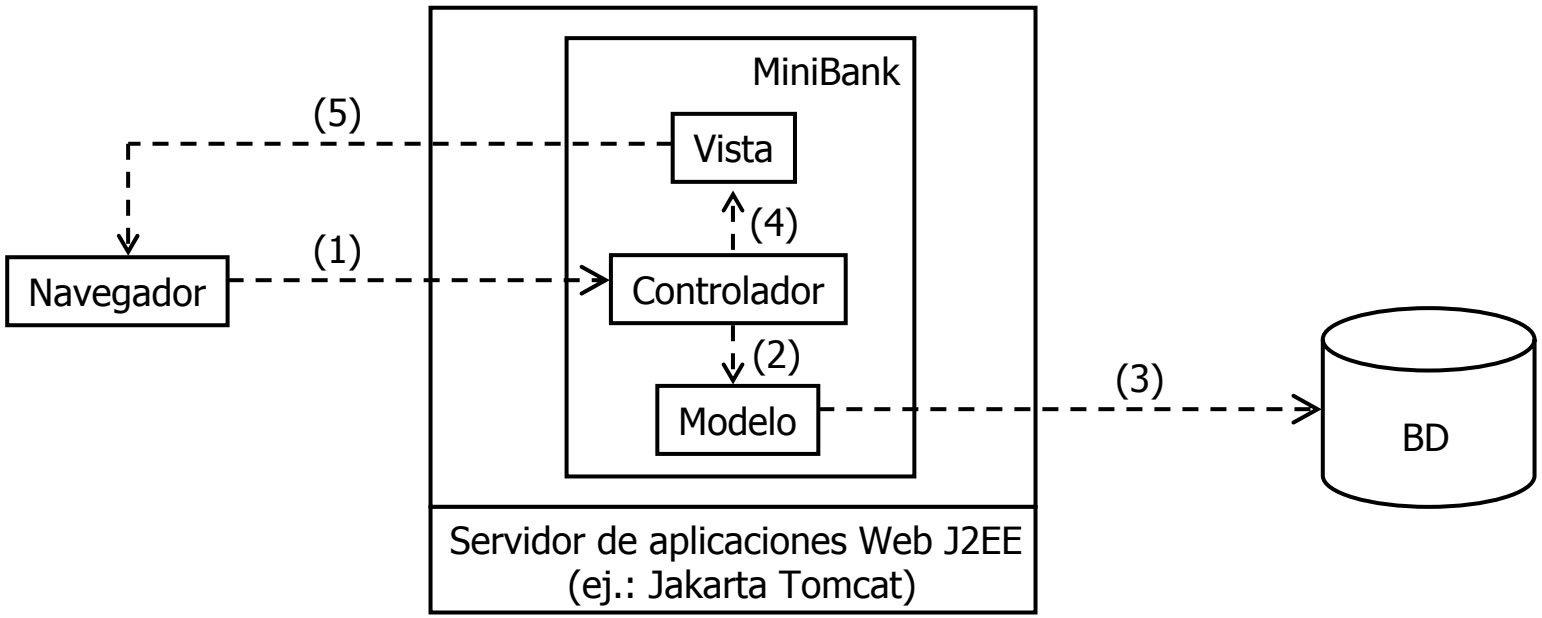


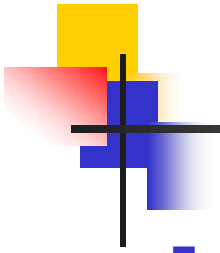
Caso de estudio: diseño de una aplicación Web bancaria (6)
Arquitectura global de la aplicación

- Problema
  - Deseamos que haya una separación entre la interfaz de usuario y la lógica de negocio
    - Interfaz de usuario: las clases y/o artefactos que reciben los eventos del usuario y generan las respuestas visuales
    - Lógica de negocio: las clases que implementan los casos de uso de manera independiente de la interfaz de usuario
  - Ventajas
    - Si se decide cambiar de tipo de interfaz de usuario en un futuro, la lógica de negocio se puede reusar
      - La lógica de negocio es independiente de la interfaz de usuario
    - Se favorece la separación de roles en el equipo de desarrollo
      - Personas que desarrollan la interfaz de usuario
      - Personas que desarrollan la lógica de negocio

Caso de estudio: diseño de una aplicación Web bancaria (7)
Arquitectura global de la aplicación

- Solución
  - Patrón Model-View-Controller (MVC)
    - Estructurar el software en 3 capas
      - Modelo: lógica de negocio
      - Vista (interfaz de usuario): las clases y/o artefactos que generan las repuestas visuales
      - Controlador (interfaz de usuario): las clases que reciben los eventos del usuario, invocan al modelo, y finalmente a la vista



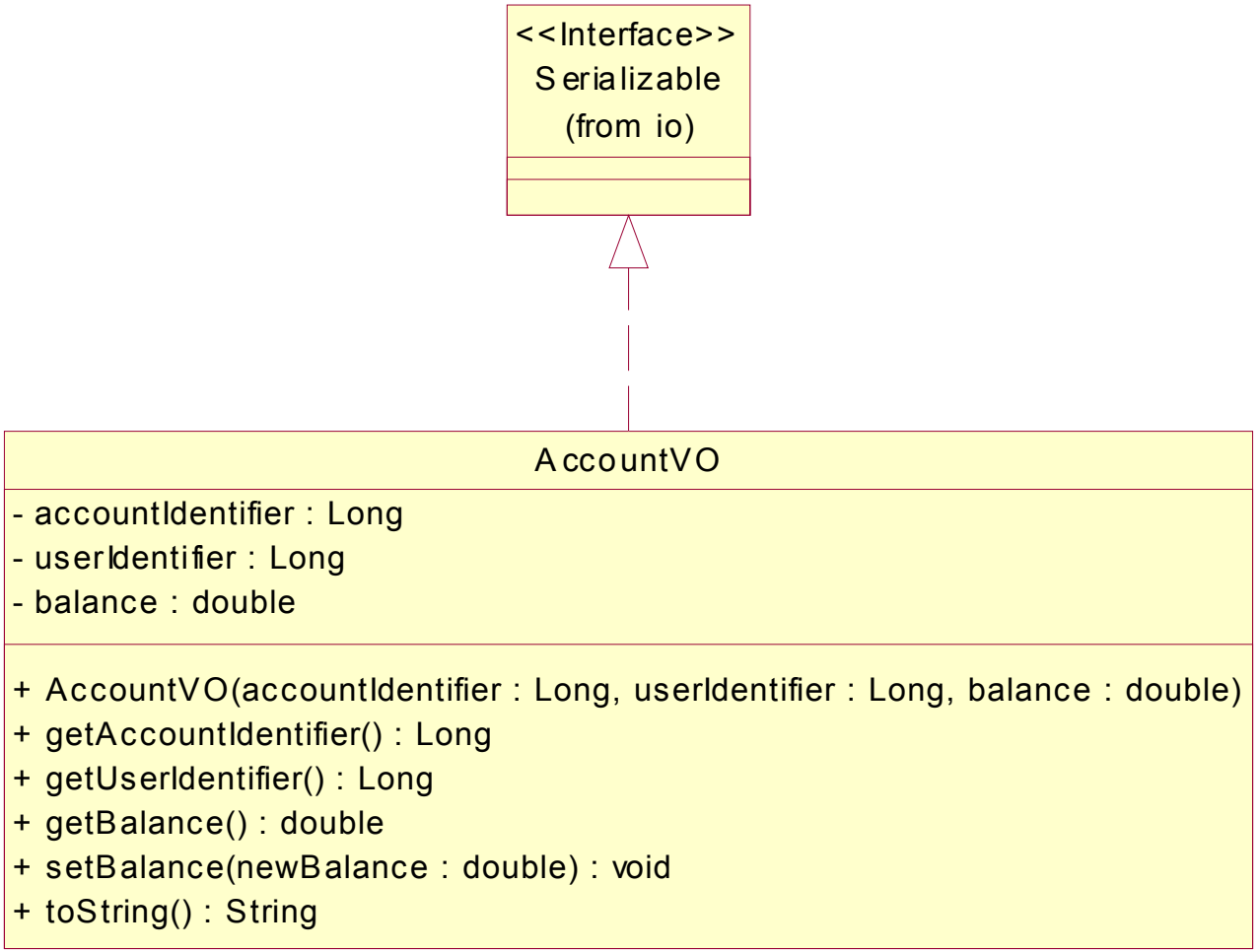


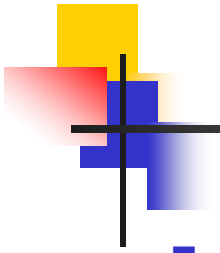
Caso de estudio: diseño de una aplicación Web bancaria (8)
Diseño de la capa modelo - Problemas

- Problema 1: ¿Cómo representar el estado de los objetos del dominio (persistentes)?
- Problema 2: ¿Cómo implementar la persistencia de los objetos del dominio de forma independiente del repositorio de datos (BD relacional, BD orientada a objetos, etc.)?
- Problema 3: ¿Cómo proporcionar una vista sencilla de la capa modelo a la capa controlador?

Caso de estudio: diseño de una aplicación Web bancaria (9)
Diseño de la capa modelo – Solución al problema 1

- Ejemplo para el caso de los datos de una cuenta bancaria



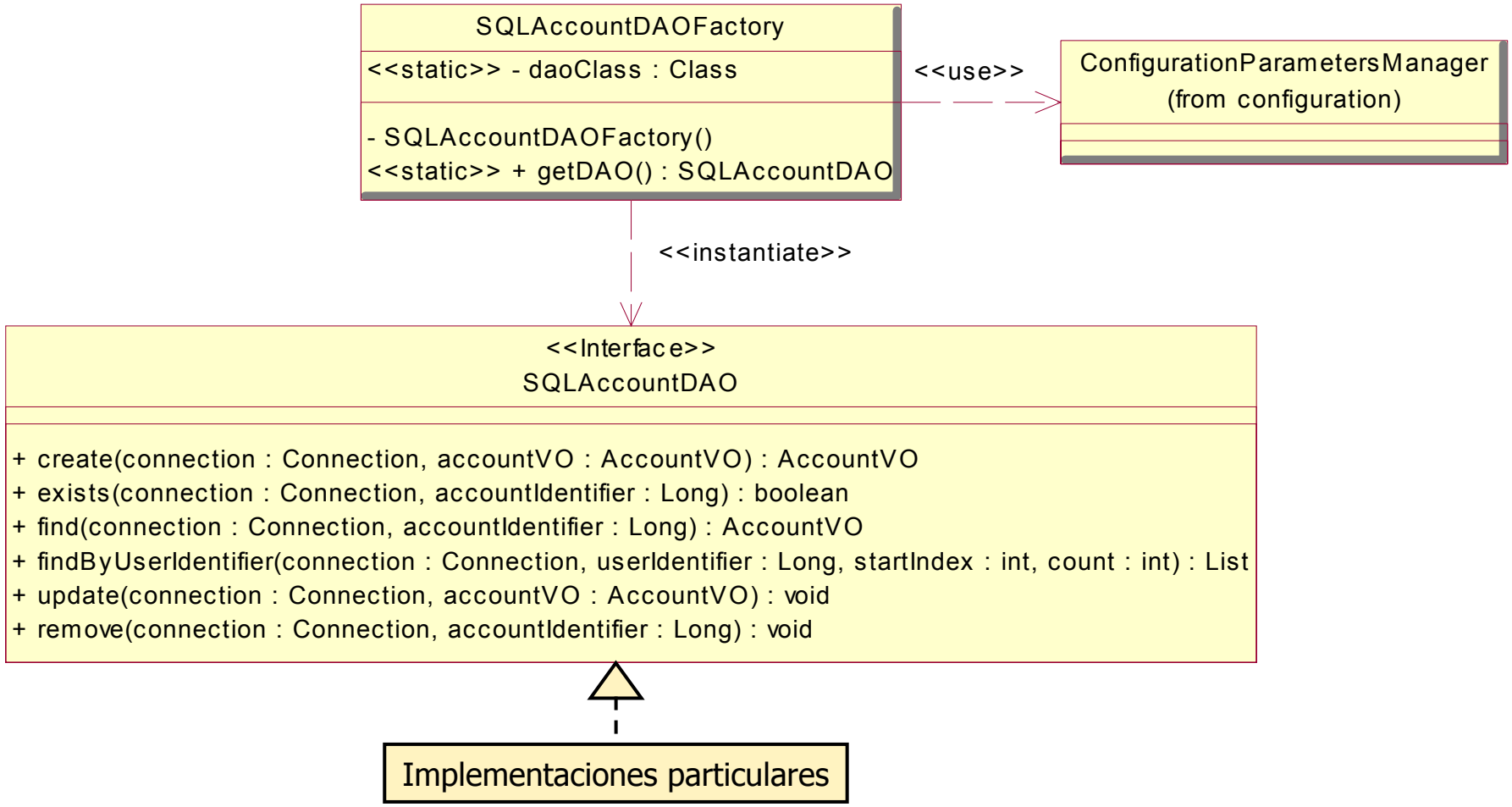


Caso de estudio: diseño de una aplicación Web bancaria (10)
Diseño de la capa modelo – Solución al problema 1

- Patrón Transfer Object (TO)
  - Anteriormente conocido como Value Object (VO)
  - Representa estado/valor
  - Atributos privados => estado
  - Métodos **`get`** para acceder a los atributos
  - Métodos **`set`** para los atributos modificables
  - Quizás un método **`toString`** para imprimir el estado
    - Útil para depuración y pruebas de unidad
  - Implementa **`java.io.Serializable`** si se precisa enviar por la red
  - Fuente: http://java.sun.com/blueprints/patterns/index.html

Caso de estudio: diseño de una aplicación Web bancaria (11)
Diseño de la capa modelo – Solución al problema 2

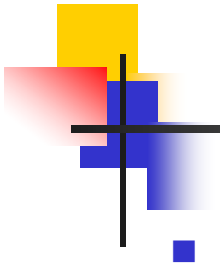
- Ejemplo para el caso de los datos de una cuenta bancaria



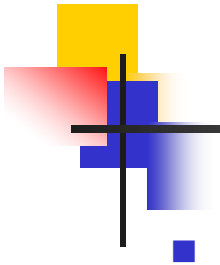
Caso de estudio: diseño de una aplicación Web bancaria (12)
Diseño de la capa modelo – Solución al problema 2

- Recuperación de los datos de una cuenta a partir de su identificador

```
/* Get a connection. */
Connection connection = ...

/* Get dao. */
SQLAccountDAO dao = SQLAccountDAOFactory.getDAO();

/*
 * Get the AccountVO corresponding to accountIdentifier
 * 12345.
 */
Long accountIdentifier = new Long(12345);
AccountVO accountVO = dao.find(connection,
    accountIdentifier);
```

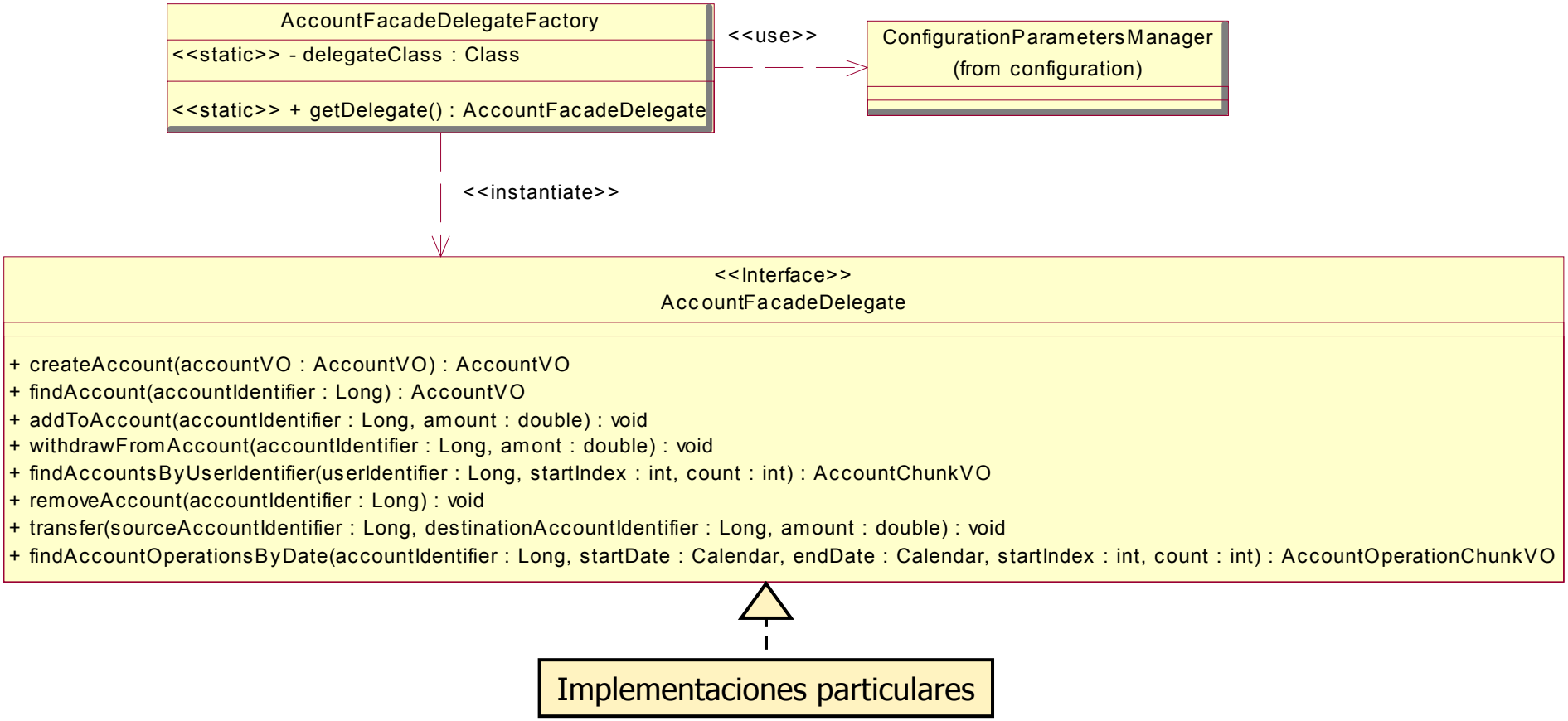


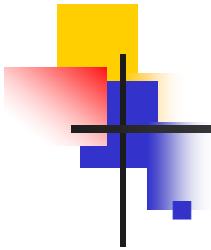
- Patrón Data Access Object (DAO)
  - Ejemplo: **`SQLAccountDAO`**
  - Define un interfaz con operaciones para insertar, borrar, actualizar y recuperar los objetos persistentes de un tipo
  - El interfaz oculta el tipo de repositorio de datos
    - NOTAS
      - Por motivos de sencillez (se explican más adelante), el interfaz **`SQLAccountDAO`** deja claro en su nombre (prefijo **`SQL`**) que las implementaciones trabajarán contra una BD relacional (porque las operaciones reciben un objeto **`java.sql.Connection`**)
      - Aún así, oculta la base de datos relacional concreta que se use (Oracle, SQL Server, PostgreSQL, MySQL, etc.) , que para ciertos aspectos pueden usar dialectos de SQL diferentes (ej.: generación de identificadores numéricos)
  - Se proporcionan tantas implementaciones del interfaz como repositorios de datos distintos se quieran soportar
  - Fuente: http://java.sun.com/blueprints/patterns/index.html
    - Es un caso particular del patrón Strategy [GoF]



- Patrón Factory
  - Ejemplo: **`SQLAccountDAOFactory`**
  - Permite crear objetos de una misma familia (implementan el mismo interfaz) sin que el código dependa de los nombres concretos de las clases
  - **`SQLAccountDAOFactory`**
    - **`getDAO()`**
      - Lee de la configuración (mediante **`ConfigurationParametersManager`**) cuál es nombre de la clase concreta que implementa **`SQLAccountDAO`**
      - Usando el API de Java (**`java.lang.Class`**) carga la clase en memoria y crea una instancia
    - Si se decide cambiar de base de datos, sólo es preciso crear una nueva implementación de **`SQLAccountDAO`** (en caso de que no sea posible dar una implementación genérica) y modificar el fichero de configuración
      - **Plug-n-play**
    - Fuente: GoF

Caso de estudio: diseño de una aplicación Web bancaria (15)
Diseño de la capa modelo – Solución al problema 3



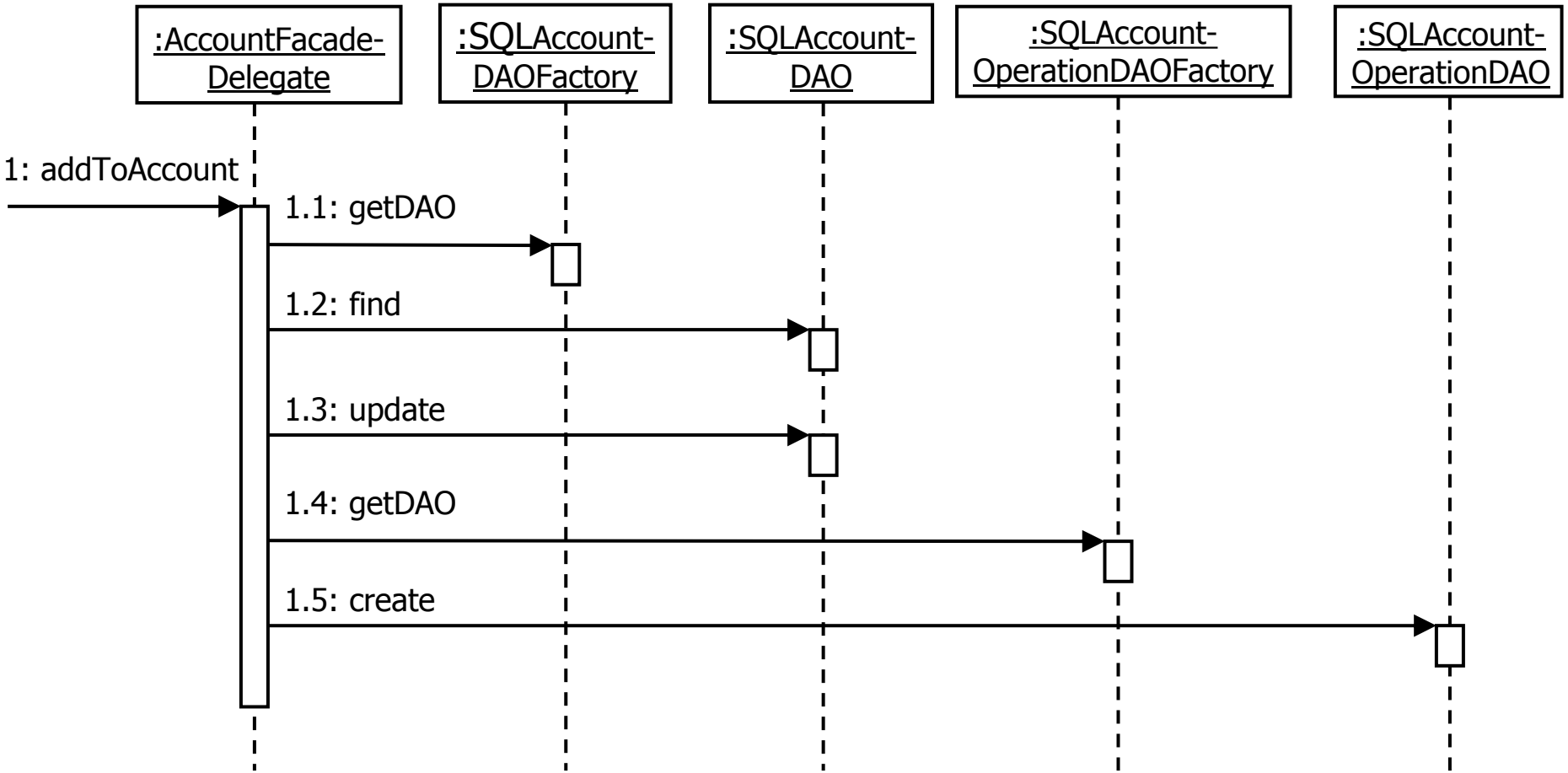


Caso de estudio: diseño de una aplicación Web bancaria (16)
Diseño de la capa modelo – Solución al problema 3

- Patrón Facade
  - Define una interfaz de alto nivel que simplifica el uso de un subsistema
  - Fuentes: GoF
  - En el caso de la capa modelo
    - Session Facade + Business Delegate (http://java.sun.com/blueprints/patterns/index.html)
    - Una fachada representa un conjunto de casos de uso lógicamente relacionados
    - Cada operación corresponde a un caso de uso
    - La declaración de las operaciones oculta la tecnología usada en su implementación
    - Puede usarse una factoría para crear instancias
      - Permite seleccionar implementaciones alternativas
    - En el caso de MiniBank, sólo hay una fachada (**`AccountFacadeDelegate`**)
    - En una aplicación real normalmente hay más de una fachada
    - Ej.: si ampliásemos MiniBank para contemplar la consulta del catálogo de productos que vende el banco, podríamos tener un **`ProductCatalogFacade`** (que ofrece operaciones para: encontrar productos por identificador, categoría, palabras clave, etc.; recuperar todas las categorías, etc.)

Caso de estudio: diseño de una aplicación Web bancaria (17)
Diseño de la capa modelo – Solución al problema 3

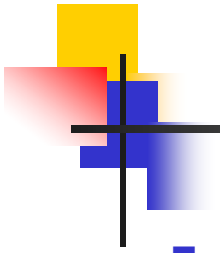
- Implementación de **`addToAccount`**



Caso de estudio: diseño de una aplicación Web bancaria (18)
Diseño de la capa modelo – Solución al problema 3

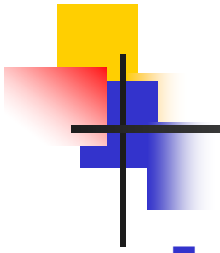
- Implementación de **`addToAccount`** (cont)

```
<< Get connection >>
<< Start transaction >>

/* Find account. */
SQLAccountDAO accountDAO = SQLAccountDAOFactory.getDAO();
AccountVO accountVO = accountDAO.find(connection, accountIdentifier);

/* Set new balance. */
double currentBalance = accountVO.getBalance();
double newBalance = currentBalance + amount;
accountVO.setBalance(newBalance);
accountDAO.update(connection, accountVO);

/* Register account operation. */
SQLAccountOperationDAO accountOperationDAO =
    SQLAccountOperationDAOFactory.getDAO();
AccountOperationVO accountOperationVO =
    new AccountOperationVO(new Long(-1), accountIdentifier,
        Calendar.getInstance(), ADD_OPERATION, amount);
accountOperationDAO.create(connection, accountOperationVO);

<< Commit transaction >>
<< Close connection >>
```



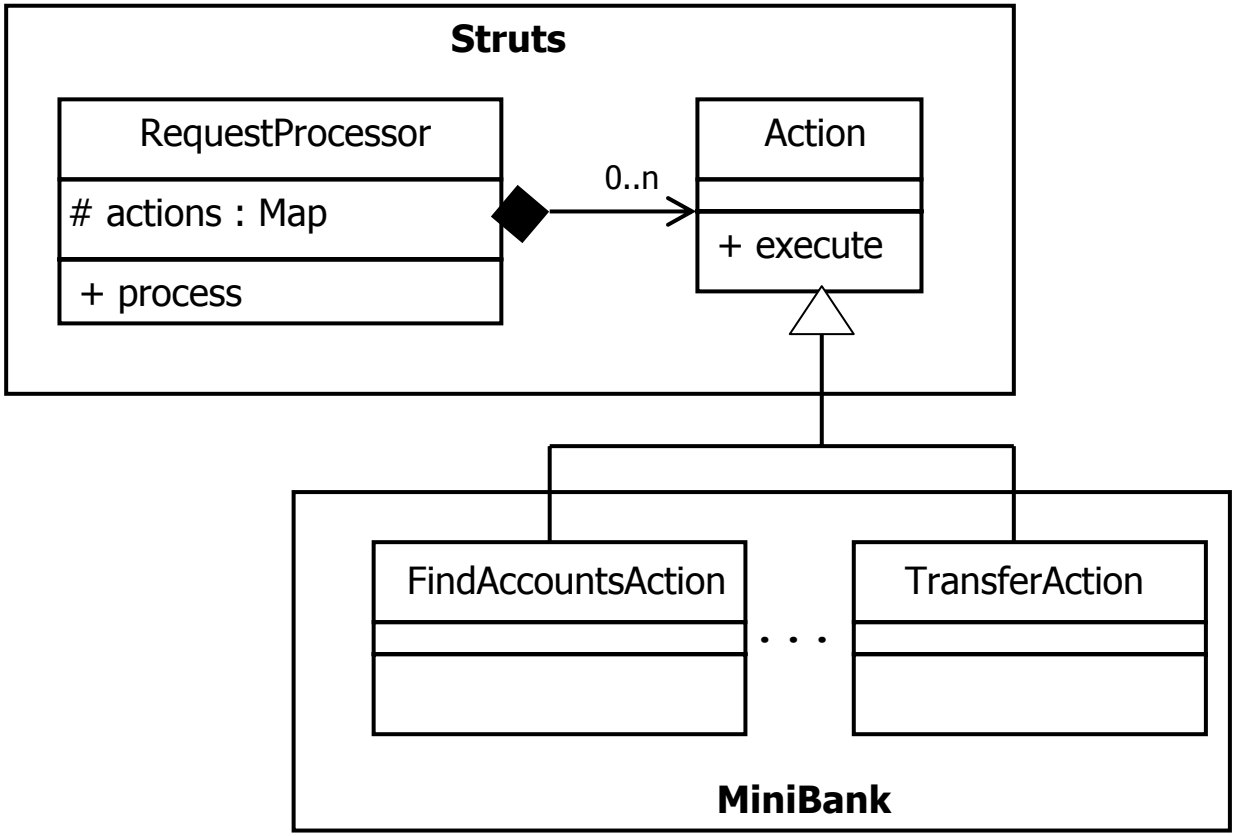
Caso de estudio: diseño de una aplicación Web bancaria (19)
Diseño de la capas controlador y vista en MiniBank

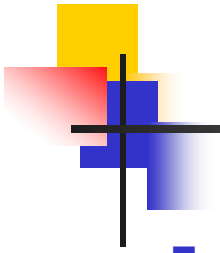
- Capa vista
  - JavaServer Pages
    - Una página JSP permite generar el markup (en este caso, HTML) que se le va a enviar al navegador (un formulario, el resultado de un caso de uso, etc.)
  - Las librerías de tags JSP estándar y la de HTML de Struts
    - Con estas librerías y una buena arquitectura MVC, no se requieren conocimientos de programación para crear las páginas JSP (sólo conocimientos de HTML y de los tags de estas dos librerías)
- Capa controlador
  - Jakarta Struts
    - Proporciona un **framework** para implementar la capa controlador de la aplicación Web
    - Proporciona la mencionada librería de tags de HTML (capa vista)



- Diferencia entre un framework y una librería
  - Librería
    - Conjunto de clases a las que el desarrollador invoca directamente (normalmente a un subconjunto de ellas, las que definen el API de la librería)
      - **El código de la aplicación invoca a la librería**
  - Framework
    - Conjunto de clases que implementan de manera parcial la arquitectura de una aplicación (o parte de la arquitectura)
      - Implementan una colección de patrones
    - Para completar la arquitectura, el desarrollador tiene que implementar algunos interfaces y/o extender algunas clases abstractas
      - **El framework invoca al código de la aplicación**
    - También suelen tener una parte que actúa como librería

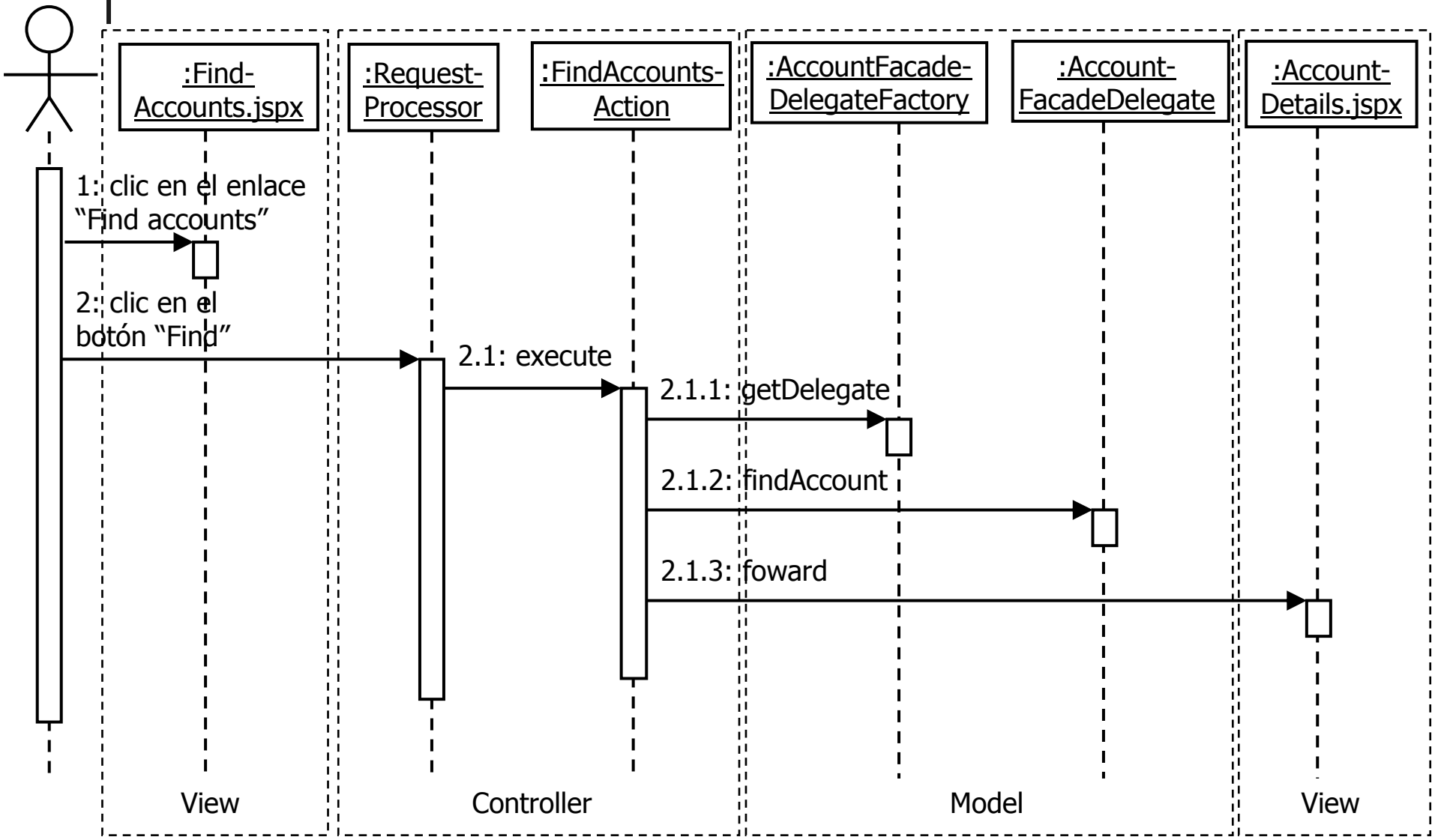
Caso de estudio: diseño de una aplicación Web bancaria (21)
Diseño de la capa controlador





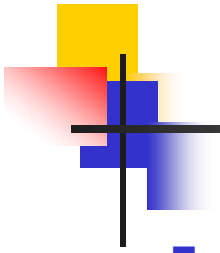
- El anterior esquema corresponde a una aplicación del patrón Command
  - Fuente: GoF
- **`RequestProcessor`**
  - Recibe una petición HTTP
  - Invoca al método **`execute`** de la acción correspondiente (todas extienden de **`Action`**), cuya implementación:
    - Accede a los parámetros de la petición HTTP (ej.: los campos de un formulario)
    - Invoca una operación de la fachada del modelo
    - Selecciona una página JSP para generar la respuesta y le pasa el resultado de la operación
  - Pasa el control a la página JSP seleccionada por la acción
    - La página JSP genera la respuesta

Caso de estudio: diseño de una aplicación Web bancaria (23)
Ejecución de un caso de uso



Caso de estudio: diseño de una aplicación Web bancaria (24)
Ejemplo de página JSP (capa vista): AccountDetails.jspx

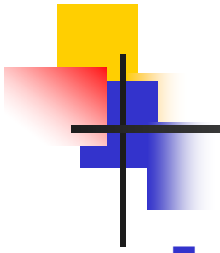
```
<table xmlns="http://www.w3.org/1999/xhtml"
       xmlns:jsp="http://java.sun.com/JSP/Page"
       xmlns:fmt="http://java.sun.com/jsp/jstl/fmt"
       xmlns:c="http://java.sun.com/jsp/jstl/core"
       class="accountDetails">

    <jsp:output omit-xml-declaration="true"/>

    <tr>
        <th><fmt:message key="AccountAttributes.accountIdentifier"/></th>
        <td><c:out value="${requestScope.account.accountIdentifier}"/></td>
    </tr>

    <tr>
        <th><fmt:message key="AccountAttributes.userIdentifier"/></th>
        <td><c:out value="${requestScope.account.userIdentifier}"/></td>
    </tr>

    <tr>
        <th><fmt:message key="AccountAttributes.balance"/></th>
        <td><fmt:formatNumber value="${requestScope.account.balance}"/></td>
    </tr>

</table>
```



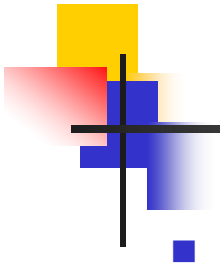
Caso de estudio: diseño de una aplicación Web bancaria (y 25)
Resumen del método de diseño seguido en MiniBank

- Modelo
  - Un TO por cada objeto persistente
  - Un DAO (+ factoría) por cada objeto persistente
  - Definir fachadas (+ factorías) del modelo
    - Cada fachada agrupa a un conjunto de casos de uso relacionados
- Controlador
  - Una acción (comando) por cada caso de uso
- Vista
  - Por cada caso de uso: una página JSP para el formulario de entrada y/o una página JSP para generar la salida

Unas palabras finales



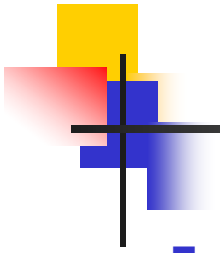
- Diseñar con patrones **NO ES**
  - "Aplico todos los patrones que yo sé en el diseño de mi software"
- Diseñar con patrones **ES**
  - "Resuelvo cada problema de diseño que me surge con un patrón apropiado"
- **OJO**, un uso excesivo de patrones fácilmente terminará en una arquitectura excesivamente compleja
  - Hay que aplicar patrones para resolver "problemas de verdad" y no problemas "filosóficos"
  - En resumen: KISS (Keep It Simple Stupid!)



Referencias (1)

- Libros
  - E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addisson-Wesley, 1994
  - J. Crupi, D. Alur, D. Malks, *Core J2EE Patterns, 2nd edition*, Prentice Hall, 2003
  - F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*, Wiley & Sons, 1996
  - J. W. Cooper, *Java Design Patterns: A Tutorial*, Addison-Wesley, 2000
  - M. Grand, *Patterns in Java: Catalogue of Reusable Design Patterns Illustrated with UML - Vol 1*, Wiley & Sons, 2002
- Sitios Web
  - http://hillside.net/patterns
  - http://java.sun.com/blueprints/patterns/index.html

Referencias (y 2)



- Asignaturas en la Facultad de Informática de la UDC
  - Diseño de Sistemas Informáticos
    - http://www.lfcia.org/dsi
    - 4º Ingeniería Informática
    - Se centra en los patrones del GoF
    - Transparencias y código disponibles
  - Integración de Sistemas
    - http://www.tic.udc.es/~fbellas/teaching/is
    - 5º Ingeniería Informática
    - Se centra en el diseño e implementación de aplicaciones empresariales con J2EE
    - Transparencias y código disponibles
- Transparencias de esta charla disponibles en
  - http://www.tic.udc.es/~fbellas/teaching/pfc3