PROGRAMACION CONCURRENTE

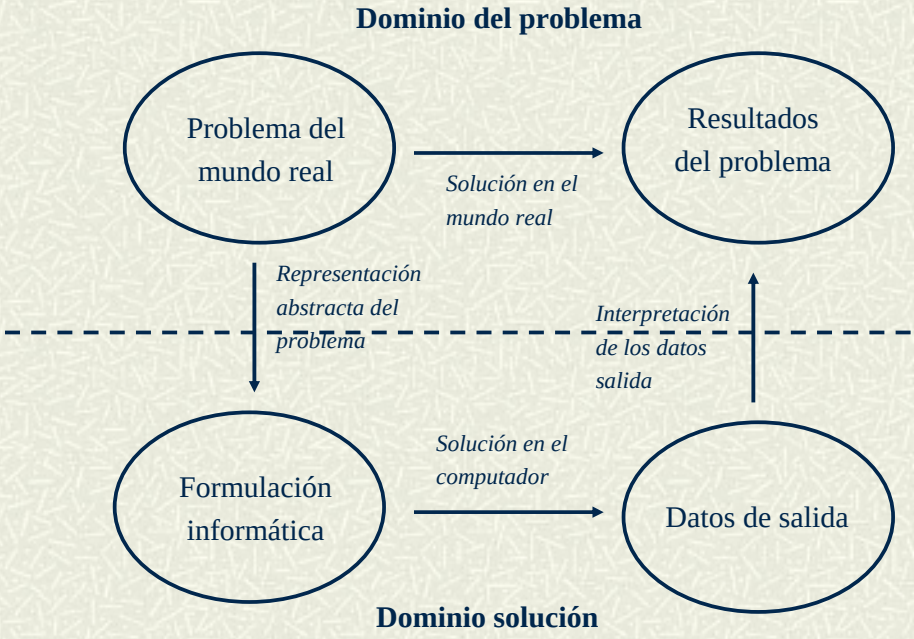
VI.3: Objetos distribuidos

José M. Drake

Notas:

Posibilidades que ofrece Java para la comunicación en red: Socket,RMI y URL.

Libertad en la representación abstracta.



Notas:

La complejidad de los problemas que se resuelven es consecuencia del desacoplo entre el lenguaje y los conceptos del dominio del problema que se resuelve y los del dominio informático.

El cliente que plantea el problema y el ingeniero que lo resuelve mediante una aplicación software, manejan diferentes interpretaciones del problema:

- Tiene diferente visión de su naturaleza.
- Conciben soluciones diferentes.
- Intercambian entre ellos especificaciones muy complejas.

Pequeños cambios en las especificaciones del problema pueden requerir cambios muy importantes (cualitativamente y cuantitativamente) en la correspondiente formulación informática.

Objetos y componentes

- Los objetos tradicionales son módulos de código que encapsulan operaciones e información y que son capaces de modelar conceptos del dominio de aplicación.
- La metodología orientada a objetos es una estrategia de descomposición del código que facilita:
  - **Descomponibilidad modular**: Permita descomponer sucesivamente cada módulo en otros mas simples que puedan ser abordados por sí.
  - **Componibilidad modular**: Genere módulos que puedan ser libremente combinados para generar módulos mas complejos.
  - **Comprensibilidad modular**: Cada módulo que se genera puede ser descrito y comprendido por sí y con independencia de otros.
  - **Continuidad modular**: La descomposición debe ser tal que pequeñas modificaciones de las especificación del problema induzca cambios en poco módulos y en proporción de las modificaciones introducidas.
  - **Protección modular**: Los errores que se produzcan en un módulo queden confinados y puedan tratarse en él.
  - **Soporte inherente de la concurrencia**: La concurrencia propia de los dominios reales se transfiere de forma natural a la aplicación.
- Los objetos tradicionales son elementos incrustados en la aplicación que sólo conocen los diseñadores y programadores.

Notas:

Objetos distribuidos: Componentes

- Un objeto distribuido es un módulo de código con plena autonomía que se puede instanciar en cualquier nudo de la red y a cuyos servicios pueden acceder clientes ubicados en cualquier otro nudo.
- Un componente es un módulo autocontenido concebido para la reutilización, que tiene especificado:
  - Los servicios que ofrece (como los objetos).
  - Los requerimientos que requieren para poder ser instalados en un nudo.
  - La posibilidades de configuración que ofrecen.
  - Información instrospectiva que describe su naturaleza y funcionalidad.
- Un componente es un objetos distribuido que no está ligado a ninguna aplicación, que se puede instanciar en cualquier nudo y ser gestionado por herramientas automáticas.

Notas:

Ventajas de los objetos distribuidos:

- Los objetos encapsulan la funcionalidad y la información de negocio en elementos que pueden ubicarse en cualquier punto del sistema distribuido.
- Los sistemas cliente/servidor construidos mediante objetos permiten adaptar la granularidad de la descomposición al nivel que se quiera.
- Ofrecen una alta capacidad de dar información de si mismo, y en consecuencia pueden ser manejados por herramientas visuales que facilitan su localización y manejo por operadores, o por aplicaciones de gestión automática del sistema.
- La interfaz y la implementación de un componentes son independientes, por lo que especialmente facilitan su mantenimiento y sustitución.
- Los componentes son especialmente idóneos para encapsular aplicaciones legadas. Remozándolas con una interfaz de componente se les proporciona nuevas opciones de conectividad e integración.
- Existen middlewares de distribución estandarizados que facilitan el ensamblado de componentes.

Notas:

Perspectivas de la tecnologías de componentes

- Va a modificar el papel de los diseñadores de aplicaciones. Dejaran de ser programadores y pasarán a ser ensambladores.
- Los pequeños desarrolladores y los ISV (Independent Software Vendors) pueden crear productos que se integren de forma sencilla con el software existente.
- Los grandes desarrolladores y los fabricantes de ERP (Enterprise Resource Planning) simplificaran la complejidad de sus aplicaciones utilizando componentes como cajas negras.
- Se generará un mercado de software orientados a mercados específicos que ofrecerá catálogos de componentes que podrán integrarse para construir nuevas aplicaciones o para enriquecer las existentes.
- La metodología de componentes, reducen la complejidad de las aplicaciones, el costo de desarrollo y el tiempo de acceso al mercado. Las aplicaciones serán mas mantenibles, mas independiente de las plataformas, y permitirán un mayor nivel de calidad.

Notas:

¿Que es un componente?

- Un componente es una pieza de software suficientemente pequeña para crearla y mantenerla con un esfuerzo reducido, y los suficientemente grande para ser distribuida y soportada.
- Características propias de un componente son:
  - Es una unidad distribuible comercialmente. Es una pieza de código (ejecutable o fuente) autocontenida y que habitualmente se adquiere en el mercado de software.
  - No es una aplicación completa: Es un módulo que realiza una funcionalidad limitada concebida para ser ensamblada con otros.
  - Concebida para ser combinada de forma impredecible: El diseñador debe prever que puede ser usada de formas no prevista por él.
  - Tiene unas interfaces bien especificadas: La interfaz es independiente de su implementación, y constituye el contrato que ofrece el componente frente al mundo externo.
  - Dispone de recursos para ser gestionado mediante herramientas: Un componente puede ser incorporado a una caja de herramientas estándar y sus características ser visualizadas mediante ventanas de configuración.

Notas:

Cracterísticas de los componentes (2)

- Notificación de eventos: Los componentes tienen capacidad de generar eventos que son recibidos asíncronamente por clientes o servidores que se hayan declarados suscriptores del mismo.
- Administración de configuración y propiedades: Un propiedad es un atributo con nombre cuyo valor puede ser leído y que al asignársele valores cambian su estado y modifican su comportamiento.
- Metadatos e instroinspección: A un componente se le puede requerir información de naturaleza y funcionalidad que ofrece. Las herramientas deben tener capacidad de gestionar componentes desconocidos.
- Interoperatibidad: Un componente puede ser accedido de acuerdo con protocolos bien definidos a través de punteros, redes, lenguajes sistemas operativos y herramientas.
- Facilidad de uso: Un componente ofrece un conjunto de operaciones estándar para su gestión: Creación, destrucción, activación, pasivación, conexión, desconexión, etc.

Notas:

Características de los componentes servidores

- Los componentes específicamente destinados a constituir servidores de carácter público, tienen mas características:
  - Seguridad: Un componente debe tener defensas frente a amenazas externas: Capacidad de autentificarse, autentificar a los clientes, control de acceso, registros de auditoría.
  - Licencias: Un componente de ofrecer mecanismos de gestión de licencias, y de auditoria de su uso para repercutir su costo.
  - Versiones: Debe ofrecer control de su versión, así como garantizar que los clientes encuentran la versión que necesitan.
  - Administración del ciclo de vida: Debe controlar gestionar su creación, destrucción, almacenamiento, así como clonarse y migrar.
  - Control de transacciones: Debe tener capacidad de protegerse frente a una utilización abusiva.
  - Persistencia: Debe tener capacidad de guardar transitoriamente su estado en un repositorio para poder posteriormente recuperarlo.
  - Relaciones: Debe tener capacidad de asociarse dinámicamente y estáticamente con otros componentes. Por ejemplo, como contenedor.
  - Autoprueba: Debe tener capacidad de autodiagnosticarse y de notificar sus problemas.
  - Autoinstalación: Debe tener capacidad de autoinstalarse y darse de alta en el Registro de objetos.

Notas:

Ejemplo componentes Java Bean

- JavaBeans es una tecnología de componentes basada en el lenguaje Java que posibilita construir aplicaciones mediante ensamblado de módulos (componentes) reusables.
- Los JavaBeans son clases 100% Java solo que son formuladas con unos patrones de diseño y de convenios de identificadores que facilitan su gestión:
  - Escribir módulos que pueden reusarse en multitud de aplicaciones.
  - Desarrollar entornos de trabajo que facilitan la construcción de las aplicaciones.
- Los Beans hacen públicas sus propiedades (métodos, eventos, atributos, etc.) de forma que las herramientas gráficas de diseño pueden mantenerlos en toolboox y mostrar al diseñador sus características para que las gestiones durante el diseño.

Notas:

Características de los JavaBeans

- Son siempre clases públicas
- Tiene que tener un constructor público del tipo por defecto (sin argumentos) aunque puede tener otros mas específicos.
- Tienen que implementar la interfaz “Serializable” que asegura su codificación (mediante un texto).
- Tiene métodos de acceso públicos a los atributos que siguen la convención de la interfaz BeanInfo.
  - get<PropertyName> y set<PropertyName> e is<BooleanProperty Name>
- Si los beans generan eventos debe implementar seguir el patrón Listener y ofrecer métodos para que se registren y den de baja los objetos que deseen recibirlos.

Notas:

The JavaBeans API makes it possible to write component software in the Java programming language. Components are self-contained, reusable software units that can be visually composed into composite components, applets, applications, and servlets using visual application builder tools. JavaBean components are known as *Beans*. Components expose their features (for example, public methods and events) to builder tools for visual manipulation. A Bean's features are exposed because feature names adhere to specific *design patterns*. A "JavaBeans-enabled" builder tool can then examine the Bean's patterns, discern its features, and expose those features for visual manipulation. A builder tool maintains Beans in a palette or toolbox. You can select a Bean from the toolbox, drop it into a form, modify it's appearance and behavior, define its interaction with other Beans, and compose it and other Beans into an applet, application, or new Bean. All this can be done without writing a line of code.

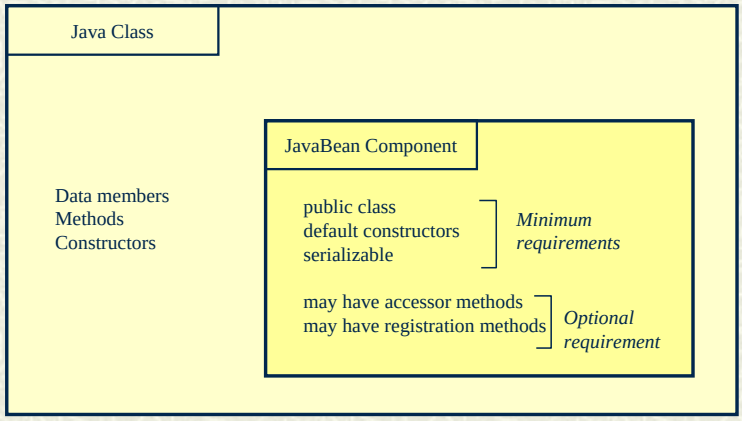
The following list briefly describes key Bean concepts, and gives the chapter in the JavaBeans specification where you can read a complete description.

•Builder tools discover a Bean's features (that is, its properties, methods, and events) by a process known as *introspection*. Beans support introspection in two ways:

•By adhering to specific rules, known as *design patterns*, when naming Bean features. The Instropector class examines Beans for these design patterns to discover Bean features. The Introspector class relies on the *core reflection* API.

•By explicitly providing property, method, and event information with a related **Bean Information** class. A Bean information class implements the BeanInfo interface. A BeanInfo class explicitly lists those Bean features that are to be exposed to application builder tools.

Requerimientos mínimos y opcionales de los JavaBeans



Notas:

•***Properties*** are a Bean's appearance and behavior characteristics that can be changed at design time. Builder tools introspect on a Bean to discover its properties, and expose those properties for manipulation. Chapter 7 of the JavaBeans API Specification discusses properties.

•Beans expose properties so they can be **customized** at design time. Customization is supported in two ways: By using property editors, or by using more sophisticated Bean customizers. Chapter 9 of the JavaBeans API Specification discusses Bean customization.

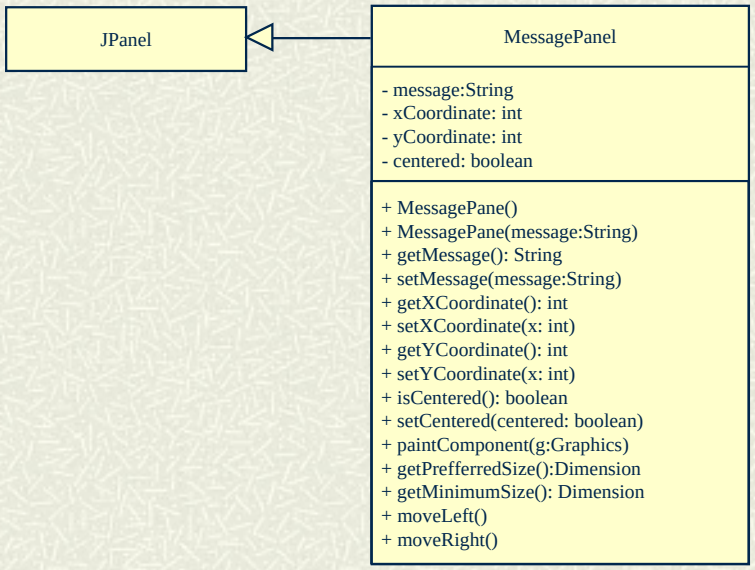
•Beans use ***events*** to communicate with other Beans. A Bean that wants to receive events (a listener Bean) registers its interest with the Bean that fires the event (a source Bean). Builder tools can examine a Bean and determine which events that Bean can fire (send) and which it can handle (receive). Chapter 6 of the JavaBeans API Specification discusses events.

•***Persistence*** enables Beans to save and restore their state. Once you've changed a Beans properties, you can save the state of the Bean and restore that Bean at a later time, property changes intact. JavaBeans uses Java Object Serialization to support persistence. Chapter 5 of the JavaBeans API Specification discusses persistence.

•A Bean's ***methods*** are no different than Java methods, and can be called from other Beans or a scripting environment. By default all public methods are exported.

Although Beans are designed to be understood by builder tools, all key APIs, including support for events, properties, and persistence, have been designed to be easily read and understood by human programmers as well.

Ejemplo de un Java Bean: MessagePanel

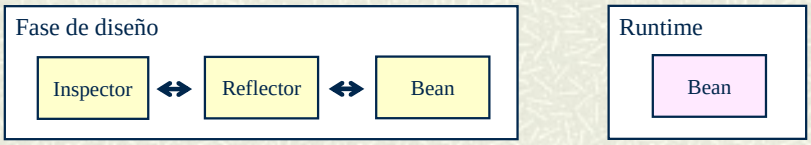


Notas:

En este ejemplo analizamos la estructura de un sencillo componente Java Bean que denominamos MessagePane cuya función es representar un mensaje sobre un JPanel.

Java Bean y la herramientas gráficas de desarrollo.

- Puesto que un componente Java Bean es una clase Java, podrá utilizarse en cualquier sitio donde esta pueda utilizarse.
- Al estar construidos de acuerdo con un patrón específico puede ser visualizado y gestionado por las herramientas gráficas Eclipse, JBuilder, Visual Café, etc. (aún cuando haya sido desarrollado por el usuario)
- Existe un componente denominado componente **Inspector** que permite cambiar y visualizar las propiedades y eventos durante la fase de diseño.
- El componente **Reflector** mantiene la actualización entre el código del componente y la información del Inspector.



Notas:

The Beans Development Kit

The BDK is delivered separately from the JDK. You can download freely from the JavaBeans web site.This site contains instructions for installing the BDK on your system. Here is a general description of the BDK files and directories:

•README.html contains an entry point to the BDK documentation

•LICENSE.html contains the BDK license agreement

•GNUmakefile and Makefile are Unix and Windows makefiles (.gmk and .mk suffixes) for building the demos and the BeanBox, and for running the BeanBox

•beans/apis contains:

- a java directory containing JavaBeans source files
- a sun directory containing property editor source files

•beans/beanbox contains

- makefiles for building the BeanBox
- scripts for running the BeanBox
- a classes directory containing the BeanBox class files
- a lib directory containing a BeanBox support jar file used by MakeApplet's produced code

sun and sunw directories containing BeanBox source (.java) files

- a tmp directory containing automatically generated event adapter source and class files, .ser

files, and applet files automatically generated by MakeApplet.

• ....