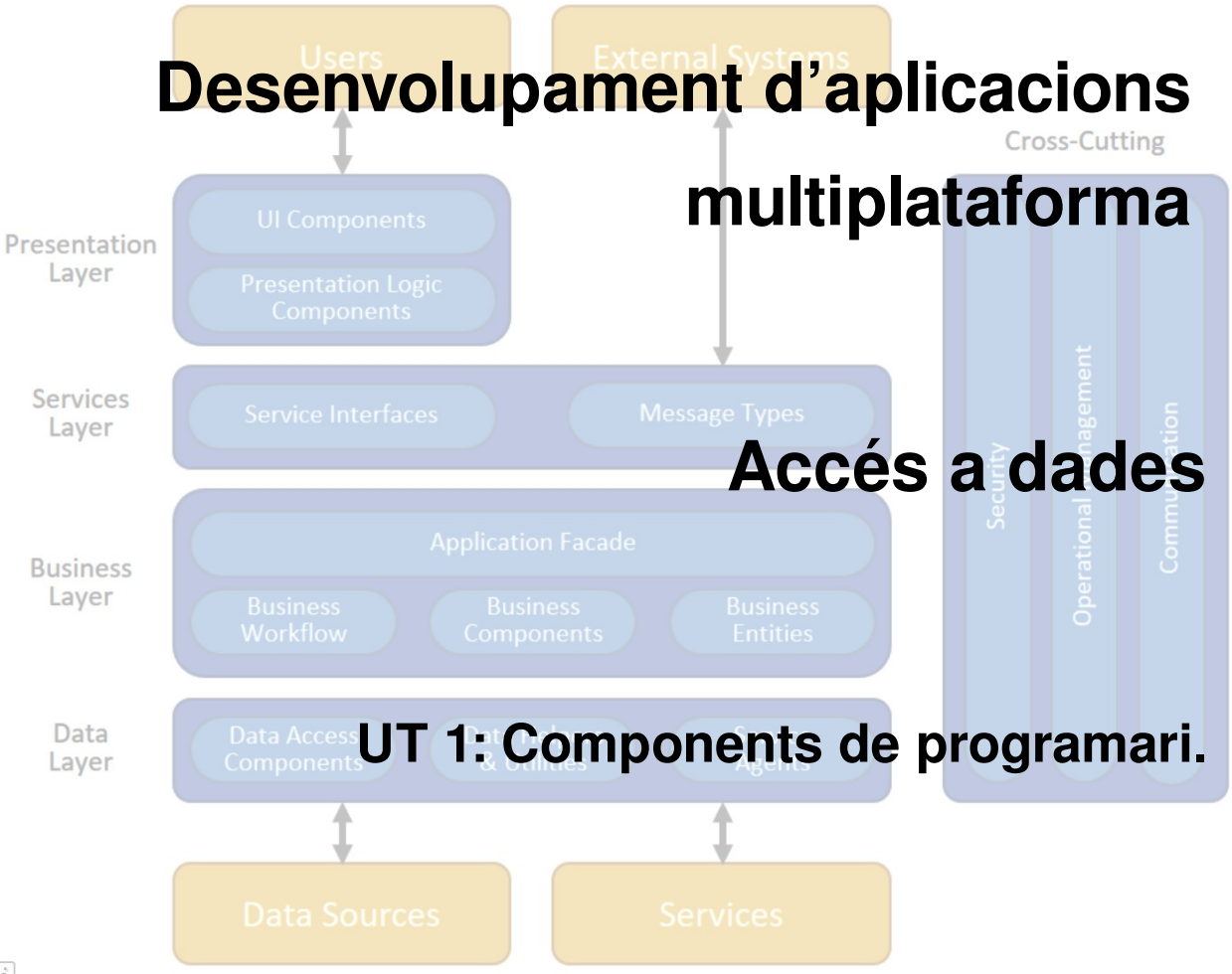
PAU CASESNOVES
Centre Integrat de Formació Professional



Desenvolupament d'aplicacions multiplataforma

Accés a dades

UT 1: Components de programari.

Joan Pons Tugores

Índex de continguts

Introducció....3
Definició de component....3
Fases de construcció dels components de programari....4
Especificació de components de programari....5
Implementació de components de programari....6
Empaquetatge dels components....6
Desplegament dels components de programari....7
Consells pràctics....7
Model-Vista-Controlador....7
Components....8
Paquets....8
Com fer les pràctiques....10

Introducció

Quan hem de dissenyar una aplicació tenim diverses opcions segons les seves característiques. Per exemple, si és una aplicació que no necessita recursos externs a la pròpia aplicació no necessitam més que un sol programa que realitzi les totes les tasques de l'aplicació. Un exemple podria ser el bloc de notes.

A mida que ens calgui enfrontar-nos a aplicacions cada cop més complexes, veurem que el nombre de classes començarà a créixer. Per intentar facilitar la integració de totes aquestes classes va aparèixer el concepte de component de programari. La idea és tractar el programari com si fos un producte mecànic:

Un cotxe es dissenya per parts de manera que cada una d'aquestes parts encaixi amb les altres i es pugui canviar fàcilment.

De la mateixa manera, s'intenta organitzar tot el conjunt de classes que componen una aplicació en components independents, de manera que sigui possible la creació independent de cada peça i s'aconsegueixi un assemblatge final fàcil i eficient.

Definició de component

Seguint amb l'exemple del cotxe, amb el qual hem introduït el concepte de component, aprofitarem per discutir quan una agrupació de peces podem considerar-la component i quan no.

Direm que els components de programari són aquelles unitats executables i independents que componen una aplicació, les quals tenen una funcionalitat i una forma d'interactuar perfectament definides, de manera que el seu assemblatge amb la resta de components es pugui fer sense haver de modificar el codi intern de cap d'ells i, a més, en qualsevol moment sigui possible la substitució per un altre component equivalent (amb una funcionalitat definida de forma idèntica).

Segurament, la forma més evident de components de programari que podeu reconèixer a primer cop d'ull són els anomenats plugins o extensions. Qui no ha instal·lat un diccionari al navegador, o no ha afegit un connector multimèdia per visualitzar determinats formats de vídeo?

Els plugins són unitats executables i independents que poden formar part d'una aplicació amb la qual interaccionen. La seva incorporació no requereix cap modificació del codi de l'aplicació ni de la resta de components. La instal·lació és molt fàcil de dur a terme i en qualsevol moment podem substituir el plugin per un altre de més eficient o per una versió superior del mateix.

Els components no tenen perquè ser tan independents, de fet les aplicacions no funcionarien sense alguns d'ells.

Si mirem enrere, ens podem fixar en els Drivers JDBC com un clar exponent. És tracta de components de baix nivell a partir dels quals construïm altres components més específics de cada aplicació, però són components al cap i a la fi que compleixen tots els requisits exposats. Són unitats executables independents que formen part de les aplicacions, que responen a una funcionalitat perfectament definida i es poden assemblar i intercanviar fàcilment per altres d'equivalents.

Fases de construcció dels components de programari

La definició que fins ara hem donat de component de programari pot servir per fer-nos una idea del concepte, però no ens ajuda gaire a saber què hem de fer per construir en últim terme un component.

Com qualsevol peça de programari, els components s'hauran d'especificar, implementar, empaquetar i desplegar en alguna aplicació i sota alguna plataforma concretes. Cada un d'aquests aspectes comporta característiques específiques que val la pena tenir en compte si volem construir un producte de qualitat.

Especificació de components de programari

El primer que haurem de fer és definir de manera molt clara les operacions que el component implementarà, quines dades utilitzaran, ...

Aquestes descripcions es poden fer o bé utilitzant llenguatges de disseny com UML o bé utilitzant els comentaris del programari, Javadoc, per exemple.

Si hem de permetre diferents implementacions del component, per exemple una que ataqui una base de dades relacional com Postgresql i una altra que ataqui una base de dades documental com Mongodb pot ser una bona idea definir una o més interfícies amb les operacions que oferirà el component i fer classes que implementin aquestes interfícies cada una atacant un tipus de bases de dades diferents.

Evidentment, en molts casos no serà necessari definir aquestes interfícies i podrem implementar directament les classes.

Sempre és important, però en el cas dels components encara més, documentar el codi de l'aplicació. És imoprtant:

- Descriure detalladament l'entrada de dades de cada operació.
- Descriure detalladament el tipus de resultat i com s'obté o es calcula a partir de les dades inicials. No es tracta de fer una codificació a la documentació, sinó una descripció detallada que no doni peu a interpretacions errònies.
- Descriure les condicions sota les quals es produirà algun error i explicarem quin tipus d'excepció s'obtindrà en cada cas.
- Descriurem quins requeriments són necessaris per assegurar el bon funcionament del de cada operació i si es coneixen, descriurem també aquelles condicions adverses que no siguin capaces d'assegurar que la resposta del component sigui la correcta.

Imaginem que un component disposa d'una operació per calcular el nombre de vendes que un comercial hagi fet durant un cert període de temps. Imaginem que, si no es donen unes dates correctes, l'operació no fos capaç de donar una resposta fiable. Caldria indicar a la documentació que el component funcionarà correctament si les dates són correctes i coherents. Aquesta documentació permetrà als desenvolupadors que facin servir el component en el desenvolupament d'alguna aplicació prestar especial atenció a les dates que es passin a la operació.

Implementació de components de programari

La implementació dels components fa referència a la codificació d'una especificació. Com ja s'ha comentat, una mateixa especificació pot ser implementada de diverses maneres.

En Java, la implementació s'haurà de fer amb una o més classes que implementin totes les interfícies o les operacions definides en el component. Encara que és possible implementar-ho tot en una sola classe, hem d'anar alerta de no crear megaclasses que absorbeixin tota la funcionalitat del component. És important repartir-la entre diverses classes per tal de rebaixar la complexitat.

Empaquetatge dels components

En crear un component estarem pensant en la seva reutilització. Una manera fàcil de fer-ho en Java és empaquetant tots els arxius que formen el component dins d'un arxiu *.jar. D'aquesta manera bastarà incorporar aquest *jar* al classpath de la nostra aplicació per poder-lo utilitzar.

Recordau que un JAR es un fitxer de format .zip constituït almenys per les classes compilades i per un fitxer descriptiu anomenat manifest.mf i ubicat a la carpeta interna del fitxer JAR anomenada meta-inf.

El fitxer manifest.mf és útil perquè conté informació sobre el component, fins i tot és possible indicar on es troben els altres components i biblioteques requerides per aquest (classpath necessari durant l'execució).

Els fitxers JAR poden contenir també fitxers de recursos que no siguin específicament de codi, per exemple d'imatges, o altres formats que siguin necessaris durant l'execució de les classes dels components.

Desplegament dels components de programari

En aplicacions Java, desplegar el component és tan senzill com incloure el fitxer JAR al classpath de l'aplicació. La majoria d'IDEs tenen alguna opció als seus menús per fer-ho.

En altres llenguatges desplegar el component pot ser més complicat, de vegades s'han de fer instal·ladors que depenen de la plataforma on s'ha d'executar.

Consells pràctics

Model-Vista-Controlador

A l'hora de dissenyar una aplicació convé tenir en ment que la separació del codi en parts més petites facilita la codificació i el manteniment del codi.

Un bon exemple és el patró de disseny **MVC, Model – Vista – Controlador**, que diu que el codi de l'aplicació s'hauria de dividir en tres capes o parts:

- Model: Les classes que implementen les dades de l'aplicació, per exemple les classes que mantenen en memòria les dades d'una fila d'una taula de la base de dades.
- Vista: Les classes que implementen la interacció amb l'usuari: Les finestres de l'aplicació d'escriptori, les pàgines de l'aplicació web, ...
- Controlador: Les classes que implementen la lògica de l'aplicació, normalment les accions de l'usuari a la vista acaben executant mètodes del controlador, que interactua amb el model per accedir a les dades persistents de l'aplicació.

Les aplicacions que desenvoluparem en aquest mòdul seguiran aquest patró de disseny: Hi haurà classes que implementin l'accés a dades, el nostre controlador, que simplement tornaran el resultat de les operacions que facin, mai les escriuran a la consola ni a cap finestra. D'aquesta manera serà més fàcil reutilitzar aquest codi.

Per altra banda les classes que implementin la vista no atacaran directament la base de dades, per exemple, si no que es limitaran a cridar mètodes del controlador per fer-ho. Si hem de fer canvis al controlador, mentre no canviem la seva interfície, les crides als seus mètodes, ho podrem fer sense haver de modificar la vista.

Components

Cada una d'aquestes parts pot estar formada per un o més components. Si aquests components poden ser reutilitzats en altres convendria desenvolupar-los en projectes a part i distribuir-los en forma de JAR.

Si aquests components són molt particulars de l'aplicació que estam fent, és a dir, difícilment reutilitzables els podrem desenvolupar al mateix projecte de l'aplicació.

Paquets

Els projectes, tant si són per a tota l'aplicació com si són per a un sol component, poden créixer molt i contenir moltes classes, interfícies, ...

És important organitzar el codi en paquets, de manera que sigui fàcil localitzar les classes. A més els paquets generen espais de noms, el nom complet d'una classe és el nom de la classe amb el nom del paquet com a prefixe. D'aquesta manera ens estalviam conflictes pel fet que dues classes tenguin el mateix nom.

L'estàndard de nomenclatura diu que el nom dels paquets d'una empresa o organització comença amb el nom del domini d'internet al revés. Després el format és lliure, però es sol posar el nom del projecte i a partir d'aquí cada paquet amb un nom que ajudi a identificar el que conté.

Per exemple, si el domini de la nostra empresa és

```
programadors.net
```

tots els paquets de les aplicacions de la nostra organització començaran amb

```
net.programadors
```

Si el nostre projecte es diu *bloc de notes*, tots els paquets del nostre projecte començaran:

```
net.programadors.blocnotes
```

Dins del projecte, la divisió en paquets és la que vulgui l'organització. Per exemple:

```
net.programadors.blocnotes.model
net.programadors.blocnotes.vista
net.programadors.blocnotes.controlador
```

O

```
net.programadors.gestio.personal
net.programadors.gestio.clients
net.programadors.gestio.inventari
```

Com fer les pràctiques

Si no es diu el contrari farem un sol projecte per a cada pràctica, dividint el codi en paquets. En alguns casos es demanarà que es faci un component en un projecte a part per poder-lo reutilitzar.