Persistència en fitxers

Josep Cañellas Bornas

Accés a dades

3.5 Binding

El *Binding* és una tècnica que consisteix a vincular classes Java amb formats específics d'emmagatzematge de manera automatitzada. Sovint un conjunt idèntic de classes pot donar lloc a múltiples esquemes XML. Això significa que normalment l'automatització de l'escriptura de dades en format XML, que en termes tècnics s'anomena *marshalling*, requereix de certa ajuda per tal de decidir quin format s'automatitzarà d'entre els molts possibles. D'això se'n diu *mapar*, perquè ve a ser com si configuréssim un mapa on s'indiquen els vincles entre les classes i els seus atributs, i els elements i atributs XML.

En Java existeixen diverses biblioteques per gestionar el *binding*, com per exemple *JAXB*, *JiBX*, *XMLBinding*, etc. Des de la versió 6.0 s'ha incorporat en el JDK estàndard *JAXB,* unapotent biblioteca.

3.5.1 Configuració amb anotacions

JAXB utilitza *Anotacions* per aconseguir la informació extra necessària per *mapar* el *binding*. Les *Anotacions* són unes interfícies o classes de Java molt especials. Serveixen per associar informació i funcionalitat als objectes sense interferir en l'estructura del model de dades. Abans que apareguessin les *Anotacions* era necessari fer servir l'herència per poder afegir funcionalitat a una classe sense haver de codificar-la. L'herència, però, interfereix directament en el model de dades, ja que en Java no és possible l'herència múltiple. Si la classe `Rectangle`, per exemple, en correspondència al model de dades, cal que hereti de `FiguraSimple`, però a la vegada necessitem afegir a la classe rectangle una funcionalitat extra, si no fem servir *Anotacions* serà necessari escriure codi extra o fins i tot haver de modificar el model final de dades per aconseguir ambdós objectius.

Si, per contra, fem servir Anotacions, els objectes disposaran d'informació o de funcionalitat extra sense que el model de dades quedi modificat, ja que les *Anotacions* no són visibles des dels objectes, malgrat que sí són accessibles via reflexió. Les *Anotacions* poden associar-se a un paquet, a una classe, a un atribut o fins i tot a un paràmetre. Aquestes classes especials es declaren en el codi de l'aplicació anteposant el símbol @ al nom de l'*Anotació*. Quan el compilador de Java detecta una *Anotació* crea una instància i la injecta dins l'element estructural afectat (paquet, classe, mètode, atribut, etc.). Això fa que aquestes no apareguin com a atributs o mètodes propis de l'objecte, i per això diem que no interacciona amb el model de dades, però les aplicacions que ho necessitin poden obtenir la instància injectada i fer-la servir.

Les *Anotacions* poden declarar-se amb paràmetres o sense. En cas que tinguin paràmetres, aquests poden ser altres Anotacions, valors constants o valors literals, de manera que estiguin disponibles en temps de compilació (que és quan el compilador realitza la injecció).

Abans de començar a estudiar algunes de les *Anotacions* de JAXB, veurem que aquesta biblioteca és capaç també de generar, de forma automàtica, les classes necessàries per suportar la informació descrita per un Schema XML, el que venim anomenant model de l'aplicació. Les classes generades es creen amb les *Anotacions* necessàries per establir correctament els vincles entre els atributs de les classes i els elements XML.

3.5.2 Generació automàtica del model de dades

En XML, els *Schema* són documents XML que permeten definir l'estructura d'altres documents. L'estudi dels *Schema* cau per complet fora dels objectius d'aquest mòdul, però com que són una peça important en el *Binding* farem un petit repàs com a recordatori dels conceptes més importants. Els *Schema* permeten definir tipus de dades que es classifiquen en tipus de dades simples i

tipus de dades complexes. Els tipus de dades simples són valors unitaris i no es poden descompondre perquè perdrien totalment el seu sentit. Generalment, es corresponen amb tipus de dades primitius, però s'amplien amb tipus específics com *dates*, *text* o classes que representin conceptualment el mateix que els valors primitius (classe `Integer`, `BigInteger`, `Float`, etc.).

Els tipus complexos es defineixen com una composició d'altres tipus (ja siguin simples o complexes), els quals aporten un sentit extra en ser tractats en conjunt. El símil per excel·lència en termes de Java són les classes, i per especificar un tipus complex es defineixen elements, cadascun dels quals representaria un dels tipus en què es descompondria el tipus complex que s'estigui definint. És a dir, podem establir una correspondència entre els elements d'un tipus complex i els atributs d'una classe. El problema és que, al mateix nivell que els elements, es poden definir també atributs XML amb una correspondència també directa amb atributs de les classes Java. Els elements XML que configurin un tipus complex es poden definir com una seqüència ordenada d'elements (*sequence*), com una tria opcional (*choice*) o bé com una combinació d'ambdues.

XMLTool4NetBeans

XMLTool4NetBeans es distribueix a la versió 7.1 de l'IDE. Si teniu una versió anterior caldrà que us l'actualitzeu usant l'eina pròpia de l'IDE o baixant-vos-el directament del portal de plugins de NetBeans (http://plugins.netbeans.org/PluginPortal/) i cercant pel nom, o accedint directament a la pàgina http://plugins.netbeans.org/plugin/40292/xmltools4netbeans. Podeu trobar informació d'ús a la pàgina de l'autor: http://blogs.oracle.com/geertjan/entry/xml_schema_editor_in_netbeans.

NetBeans disposa d'un *plugin*, anomenat **XMLTools4NetBeans**, on podem trobar un conjunt d'eines XML, entre les quals destaquem un editor d'esquemes molt potent.

El format XML que hem triat és un format còmode per fer el tractament de dades usant un DOM, ja que les dades dels diferents nivells de la jerarquia d'una figura queden totalment encapsulades en nodes independents. Aquest format, però, presenta certa dificultat d'interpretació als ulls humans en llegir el contingut XML generat, ja que les dades de figura o figura simple es tracten com a components interns de les figures específiques (`Rectangle`, `Cercle` o `Grup`), més que no pas tractar-se com a contenidors genèrics que continguin casos específics.

Aquí, en tractar-se d'un procediment automàtic de transferència de dades, podem deixar de banda l'elecció del tipus d'estructura en funció del tractament i centrar-nos més en la part conceptual. Dissenyarem un format senzill més fàcilment interpretable en què no cal plasmar la jerarquia. D'aquesta manera, independitzem el format XML de la implementació Java finalment realitzada. Així, si algun dia decidim reestructurar la jerarquia, podem continuar amb el mateix format.

Podeu fer servir el *plugin* XMLTools4NetBeans per obtenir un *Schema* que ens defineixi un format per emmagatzemar les figures del dibuix definides a l'apartat "Implementació d'un exemple" de la Secció "Parser o analitzador sintàctic".

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3      targetNamespace="http://xml.netbeans.org/schema/dibuix"
4      xmlns:tns="http://xml.netbeans.org/schema/dibuix"
5      elementFormDefault="qualified">
6      <!--Tipus Punt per definir les coordenades bidimensional-->
7      <xsd:complexType name="Point">
8          <!--Les coordenades x i y és defineixen com atributs, no pas com
               elements-->
9          <xsd:attribute name="x" type="xsd:integer"/>
10         <xsd:attribute name="y" type="xsd:integer"/>
11     </xsd:complexType>
12     <!--Tipus Rectangle amb totes les seves caracteristiques associades-->
13     <xsd:complexType name="Rectangle">
14         <xsd:sequence>
15             <xsd:element name="posicio" type="tns:Point"></xsd:element>
16             <xsd:element name="escala" type="xsd:integer"/></xsd:element>
17             <xsd:element name="rotacio" type="xsd:float"/></xsd:element>
18             <xsd:element name="colorlinia" type="xsd:integer"></xsd:element>
```

```
            <xsd:element name="colorfigura" type="xsd:integer"></xsd:element>
            <xsd:element name="alcada" type="xsd:integer"></xsd:element>
            <xsd:element name="amplada" type="xsd:integer"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <!--Tipus Cercle amb totes les seves caracteristiques associades-->
    <xsd:complexType name="Cercle">
        <xsd:sequence>
            <xsd:element name="posicio" type="tns:Point"></xsd:element>
            <xsd:element name="escala" type="xsd:integer"/></xsd:element>
            <xsd:element name="rotacio" type="xsd:float"/></xsd:element>
            <xsd:element name="colorlinia" type="xsd:integer"></xsd:element>
            <xsd:element name="colorfigura" type="xsd:integer"></xsd:element>
            <xsd:element name="radi" type="xsd:integer"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <!--Tipus que representarà una llista de figures. La llista podrà estar
        definida sense cap figura (buida) o be contenir un nombre indeterminat
         de figures-->
    <xsd:complexType name="LlistaFigures">
         <xsd:choice maxOccurs="unbounded" minOccurs="0">
             <xsd:element name="rectangle" type="tns:Rectangle"></xsd:element>
             <xsd:element name="cercle" type="tns:Cercle"></xsd:element>
             <xsd:element name="grup" type="tns:Grup"></xsd:element>
        </xsd:choice>
    </xsd:complexType>
    <!--Tipus Grup amb totes les seves caracteristiques associades-->
    <xsd:complexType name="Grup">
        <xsd:sequence>
            <xsd:element name="posicio" type="tns:Point"></xsd:element>
            <xsd:element name="escala" type="xsd:integer"/></xsd:element>
            <xsd:element name="rotacio" type="xsd:float"/></xsd:element>
            <!--L'element ·colleccio es defineix com una llista de figures-->
            <xsd:element name="colleccio" type="tns:LlistaFigures"></
                xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <!--L'element arrel cal definir-lo sense especificar l'atribut tipus per
        tal que JAXB el detecti com a tal. El tipus caldrà definir-lo com un
        element complex amb un únic component o la llista de figures-->
    <xsd:element name="dibuix">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension xmlns:tns="http://xml.netbeans.org/schema/dibuix
                    " base="tns:LlistaFigures"></xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Un esquema com aquest validaria documents XML semblants a:

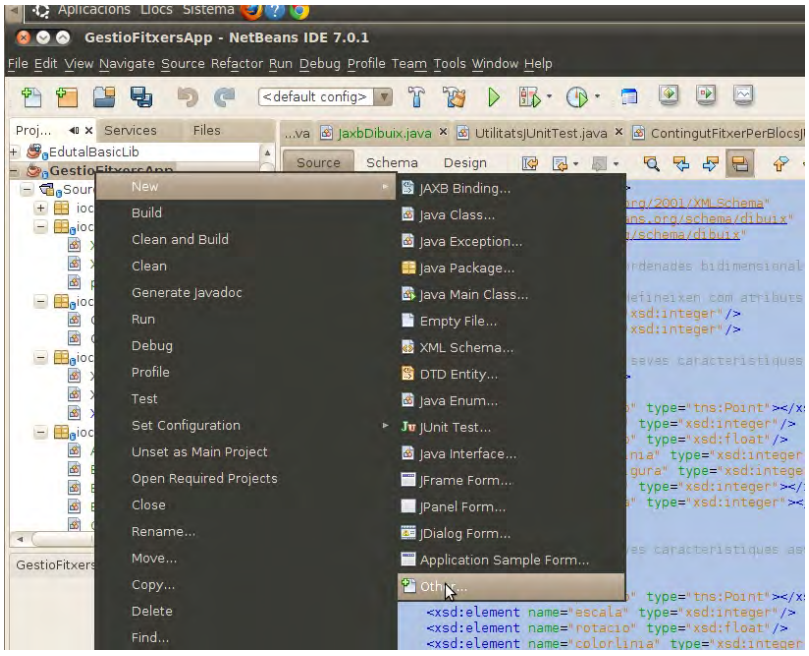
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dibuix xmlns="http://xml.netbeans.org/schema/dibuix">
    <rectangle>
        <posicio x="0" y="5"/>
        <escala>1</escala>
        <rotacio>0.0</rotacio>
        <colorLinia>-16777216</colorLinia>
        <colorFigura>-1</colorFigura>
        <alcada>10</alcada>
        <amplada>10</amplada>
    </rectangle>

    <cercle>
        <posicio x="50" y="32"/>
        <escala>1</escala>
        <rotacio>0.0</rotacio>
        <colorLinia>-16777216</colorLinia>
```

```
        <colorFigura>-1</colorFigura>
        <radi>24</radi>
    </cercle>

    <grup>
        <posicio x="0" y="0"/>
        <escala>1</escala>
        <rotacio>0.0</rotacio>
        <colleccio>
            <rectangle>
                <posicio x="1" y="1"/>
                <escala>1</escala>
                <rotacio>0.0</rotacio>
                <colorLinia>-16777216</colorLinia>
                <colorFigura>-1</colorFigura>
                <alcada>100</alcada>
                <amplada>100</amplada>
            </rectangle>
            <rectangle>
                <posicio x="100" y="100"/>
                <escala>1</escala>
                <rotacio>0.0</rotacio>
                <colorLinia>-16777216</colorLinia>
                <colorFigura>-1</colorFigura>
                <alcada>10</alcada>
                <amplada>10</amplada>
            </rectangle>
            <cercle>
                <posicio x="50" y="55"/>
                <escala>1</escala>
                <rotacio>0.0</rotacio>
                <colorLinia>-16777216</colorLinia>
                <colorFigura>-1</colorFigura>
                <radi>30</radi>
            </cercle>
        </colleccio>
    </grup>
</dibuix>
```

Figura 3.6. Menú per activar el Wizard de JAXB

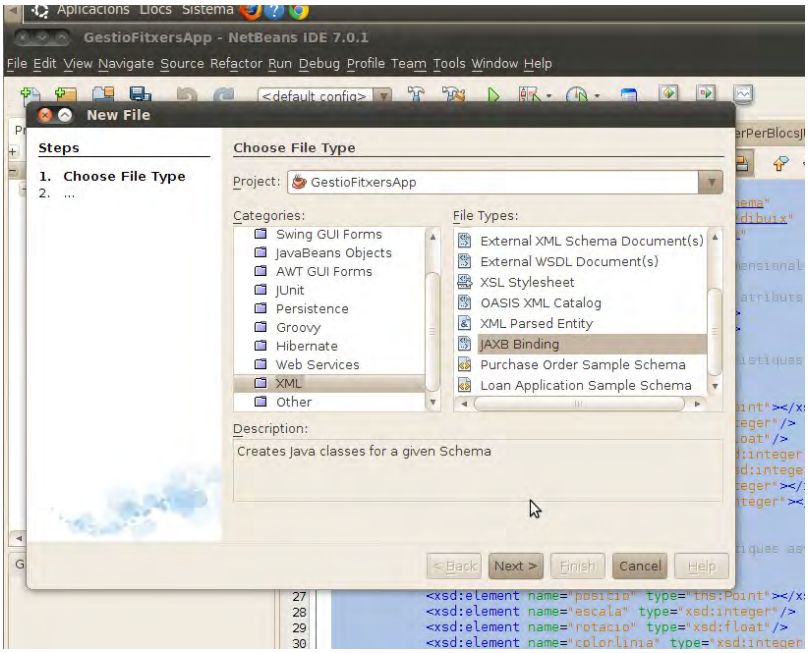


JaxB és capaç de generar el model de dades automàticament (classes `Rectangle`, `Cercle`, `Grup` o `Dibuix`) realitzant un tractament anomenat compilació JAXB

a partir de l'esquema definit més amunt. Aquest procés es pot realitzar usant comandes de consola o fent servir amb l'ajuda de l'IDE, *JAXBWizard* (figura 3.6). Sobre el projecte, cliqueu el botó dret del ratolí, seleccioneu el menú *New* i escolliu l'opció *Others*.

Això us obrirà un quadre de diàleg on caldrà que cerqueu l'opció *XML* a l'esquerra i el tipus *JAXB Binding* a la dreta (figura 3.7).

Figura 3.7. Quadre de diàleg per escollir la creació dels fitxers de configuració JAXB

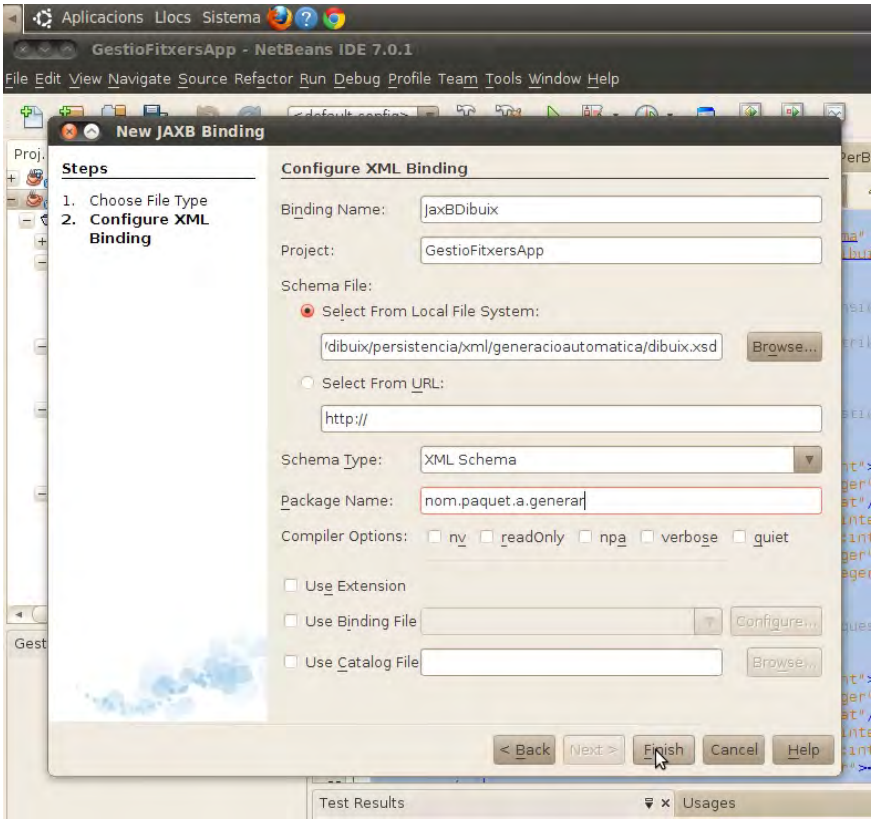


Cliqueu *Next* un cop hagueu fet la selecció correcta. D'aquesta manera, se us obrirà el darrer quadre de diàleg on caldrà que introduïu el nom de la configuració (ja que podeu generar-ne més d'una) i escolliu el fitxer esquema a partir del qual voleu fer la generació. Si cal, podeu navegar pel sistema de fitxers. Un cop escollit l'esquema, cal que introduïu el nom del paquet on desitgeu ubicar les classes generades, i finalment polseu *Finish*.

Aquest procés crearà les classes del model (les figures i el dibuix), més una classe anomenada `ObjectFactory`, la qual tindrà la funció d'instanciar els objectes de cada una de les classes del model durant el procés d'hidratació dels objectes o de traspàs de la informació des de l'XML. Si mireu el contingut, veureu també un altre fitxer anomenat *package-info.java*. No es tracta pas d'una classe, sinó d'un tipus de fitxer especial introduït a partir de la versió 5.0 de Java que permet recollir informació diversa sobre un paquet concret. Entre d'altres, admet una descripció textual sobre el propòsit general del fitxer, comentaris específics del paquet per afegir al *javadoc* i, el més important, les anotacions a nivell de paquet.

Per defecte, JAXB crearà una única *Anotació* a nivell de paquet especificant l'espai de noms que figuri al document *Schema* utilitzat per a la generació just abans de la sentència Java que identifica un paquet (figura 3.8).

Figura 3.8. Quadre de diàleg per configurar la generació automàtica de les classes del model a partir d'un Schema




```
@javax.xml.bind.annotation.XmlSchema(namespace =
    "http://xml.netbeans.org/schema/dibuix", elementFormDefault =
    javax.xml.bind.annotation.XmlNsForm.QUALIFIED)
package ioc.dam.m6.exemples.dibuix.persistencia.xml.generacioautomatica.schema;
```

Podeu observar el símbol @ com a element identificador de les *Anotacions*. Aquesta anotació, concretament, conté dos paràmetres, l'espai de noms de l'esquema i la indicació que les etiquetes descrites a l'*Schema* són qualificades i, per tant, pertanyents a l'espai de noms indicat.

La informació associada al paquet, junt amb la classe `ObjectFactory`, servirà a JAXB per crear el context de treball: una instància de *JAXContext* que usarà aquesta informació per crear uns objectes `Marshaller` o `UnMarshaller` específics per treballar amb el nostre model. La instrucció per crear el context és la següent:

```
JAXBContext context = JAXBContext.newInstance(nomPaquet);
```

L'objecte `Marshaller` l'instancia el context, fent:

```
Marshaller marshaller = context.createMarshaller();
```

Per realitzar transvasament d'informació del model al document XML serà necessari l'objecte del model corresponent a l'objecte contenidor principal de tota la informació (és a dir, l'equivalent al qual serà l'arrel del document XML) i un OutputStream associat al fitxer d'emmagatzematge.

```
marshaller.marshal(objecteRoot, fitxerOutputStream);
```

En el nostre model de figures, l'objecteRoot seria el dibuix.

La instància *UnMarshaller* també la crearà el context:

```
1 Unmarshaller unmarshaller = context.createUnmarshaller();
```

Per obtenir un objecte contenidor a partir de les dades emmagatzemades en un XML caldrà passar per paràmetre un `FileInputStream` del fitxer. El mètode `unmarshal`gestionarà tota la creació d'objectes necessaris per realitzar el transvasament d'informació cap al model. La instanciació dels objectes es realitzarà fent servir la classe `ObjectFactory`. El mètode `unmarshal`retornarà un objecte corresponent al contenidor principal i l'equivalent a l'arrel del document XML.

```
1 objectRoot = unmarshaller.unmarshal(fitxerInputStream);
```

Vigileu que les instàncies siguin realment de les classes generades per JAXB i no pas les usades a l'apartat "Implementació d'un exemple" de la Secció "Parser o analitzador sintàctic" i que formen part de la biblioteca del mòdul. Recordeu que, de moment, el vincle s'ha establert amb les classes generades de forma automàtica.

Podeu fer la prova instanciant un objecte Dibuix al qual li afegiu unes quantes figures.

3.5.3 Ús de JAXB amb un model de disseny propi

Normalment, el models de dades solen ser la peça clau de les aplicacions, i generar-los automàticament a partir d'un Schema no és, de ben segur, la millor manera d'optimitzar-lo. Es pot modificar l'Schema per aconseguir millors resultats, però a vegades és preferible plantejar les coses al revés. És a dir, partint de les classes, fem les anotacions oportunes per aconseguir generar XML que validin un determinat Schema.

Segurament, la generació automàtica pot ser útil quan sigui necessari disposar ràpidament d'un model de classes Java a partir d'un model XML existent. Però si el punt de partida són les classes Java, la solució passa per escriure notacions sobre el model o sobre alguna classe derivada.

Cal que entengueu la derivació com a herència.

Si partim d'un model dissenyat totalment per nosaltres caldrà crear, a banda de les classes del model, la classe `ObjectFactory`i el fitxer *package-info.java*. L'`ObjectFactory`tindrà almenys un mètode de creació per a cada classe del model. Si fos necessari, una classe podria disposar de més d'un mètode d'instanciació, de la mateixa manera que fos necessari tenir diversos constructors.

Si, en canvi, partíssim d'un model derivat d'una altre al qual no hi tinguéssim accés, perquè fos un biblioteca de tercers, perquè no disposéssim de les fonts o per qualsevol altre motiu, només podríem anotar les classes derivades. Com ja hem vist, les *Anotacions* tenen diferents àmbits d'influència segons el tipus. Hi ha anotacions de paquets que afecten totes les seves classes, però hi ha també anotacions de classes especifiques, el rang de les quals afectaria només a la classe on s'ha definit i als seus components, mètodes, atributs, etc.

Un problema que trobarem quan treballem amb herència usant JAXB és que les *Anotacions* de classes afecten només la classe específica on s'han escrit les

anotacions, i no serveixen per a les superclasses. No és possible, per exemple, anotar en la classe `Cercle` un atribut de `Figura`. Si necessitem anotar l'atribut caldrà fer-ho dins la mateixa classe `Figura`. Però si no hi tinguéssim accés, existeixen diverses solucions. Aquí plantejarem la més laboriosa, però també la que permet un major grau de configuració. La tècnica consisteix, d'una banda, a evitar que JAXB faci cap exploració de les classes que no puguem anotar. De l'altra, a dotar a les classes derivades d'accessors (mètodes `geti set`) propis per accedir als atributs heretats.

És possible optar per solucions intermèdies si la seriació per defecte que fa JAXB ja ens anés bé. En aquest cas, només caldria derivar la classe que representi l'etiqueta arrel per poder anotar-la com a tal.

Anotacions bàsiques

Per introduir les anotacions bàsiques de JAXB continuarem amb l'exemple del dibuix de figures geomètriques. Implementarem dues versions: en una, crearem el model des de zero i escriurem *anotacions* en qualsevol de les classes creades. A la segona versió, partirem del model que trobem a la biblioteca de manera que per anotar-lo ens calgui crear un model derivat. Farem les implementacions en paral·lel, i així copsarem millor les diferències.

Preparació dels paquets del model

Pel que fa a la implementació des de zero, en primer lloc caldrà crear el paquet. Per exemple:

```
ioc.dam.m6.exemples.dibuix.persistencia.xml.model
```

Per al model derivat, el paquet podria ser:

```
ioc.dam.m6.exemples.dibuix.persistencia.xml.modelderivat
```

També caldrà evitar que JAXB explori les classes originals de la biblioteca impedint-ne la seriació a l'XML. En el nostre projecte crearem un paquet amb el mateix nom que el paquet on es trobi el model original (el de la biblioteca). Un cop creat, hi ubicarem únicament un fitxer *package-info.java,* en el qual anotarem la deshabilitació dels accessors fent servir `@XmlAccessorType`, al qual li passarem per paràmetre el valor de la constant `@XmlAccessType.NONE`, la qual indicarà que les classes del paquet associat no seran accessibles des de JAXB.

```
@javax.xml.bind.annotation.XmlAccessorType(value=
    javax.xml.bind.annotation.XmlAccessType.NONE)
package ioc.dam.m6.exemples.dibuix;
```

Podreu crear un fitxer *package-info.java* clicant *new File*, escollint la categoria Java i seleccionant el tipus de fitxer *Java Package Info*.

Per a la creació del model, en la versió de la implementació total (des de zero) caldrà crear les mateixes classes (`Cercle`, `Rectangle`, `Grup`, `Figura`, `Dibuix`, etc.) que el model de la biblioteca.

Per al model derivat, caldrà implementar `XmlDibuix`, `XmlCercle`, `XmlRectangl` i `XmlGrup` heretant de les classes corresponents i implementant els accessors

adequats. Veiem com a exemple `XmlCercle`:

```java
public class XmlCercle extends Cercle {

    public XmlCercle() {
    }

    public XmlCercle(int x, int y, int radi) {
        super(x, y, radi);
    }

    @Override
    public Point getPosicio() {
        return super.getPosicio();
    }

    @Override
    public void setPosicio(Point posicio) {
        super.setPosicio(posicio);
    }

    @Override
    public Color getColorFigura() {
        return super.getColorFigura();
    }

    @Override
    public void setColorFigura(Color colorFigura) {
        super.setColorFigura(colorFigura);
    }

    @Override
    public Color getColorLinia() {
        return super.getColorLinia();
    }

    @Override
    public void setColorLinia(Color colorLinia) {
        super.setColorLinia(colorLinia);
    }

    @Override
    public int getEscala() {
        return super.getEscala();
    }

    @Override
    public void setEscala(int escala) {
        super.setEscala(escala);
    }

    @Override
    public float getRotacio() {
        return super.getRotacio();
    }

    @Override
    public void setRotacio(float rotacio) {
        super.setRotacio(rotacio);
    }

    @Override
    public int getRadi() {
        return super.getRadi();
    }

    @Override
    public void setRadi(int radi) {
        super.setRadi(radi);
    }
```

```
}
```

De forma semblant, caldrà implementar `XmlRectagle` i `XmlGrup`. D'aquesta darrera cal comentar que l'atribut `elements`necessita també accessors per tal que JAXB hi pugui accedir.

```
public class XmlGrup extends Grup{
    public ArrayList<Figura> getElements(){
        return elements;
    }

    public void setElements(ArrayList<Figura> list){
        elements=list;
    }

    ...
}
```

Finalment, a la `classeXmlDibuix` hi afegirem un nou mètode per copiar la llista de figures d'un dibuix a un altre.

```
public class XmlDibuix extends Dibuix {
    public XmlDibuix() {
    }

    public List<Figura> getFigures(){
        return this;
    }

    public void copiarDesDe(Dibuix dibuix){
        this.clear();
        this.addAll(dibuix);
    }
}
```

Per a ambdues versions, i amb l'objectiu que JAXB reconegui les classes que ha de manipular i *mapar*, caldrà crear un arxiu *package-info.java* i un `ObjectFactory`a cada paquet.

Al fitxer amb la informació del paquet hi escriurem l'anotació `@XmlSchema`, que ens permetrà definir un espai de noms (`http://xml.ioc.edu/schema/dibuix`, per exemple) i indicar si els noms són o no qualificats. També indicarem de forma exclusiva en la versió del model derivat, que l'accés a les dades vinculades amb l'XML es realitzarà per mitjà d'accessors (mètodes `getiset`) usant el valor de la constant *XmlAccessType.PROPERTY*, mentre que el tipus d'accés per al model propi serà directe als atributs. Usarem el valor `XmlAccessType.FIELD`.

El contingut del *package-info.java* del model derivat és el següent:

```
@javax.xml.bind.annotation.XmlSchema(namespace =
    "http://xml.ioc.edu/schema/dibuix", elementFormDefault =
    javax.xml.bind.annotation.XmlNsForm.QUALIFIED)
@javax.xml.bind.annotation.XmlAccessorType(value=
    javax.xml.bind.annotation.XmlAccessType.PROPERTY)
package ioc.dam.m6.exemples.dibuix.persistencia.xml.modelderivat;
```

I el del *package-info.java* del model propi, aquest:

```
@javax.xml.bind.annotation.XmlSchema(namespace =
```

```
    "http://xml.ioc.edu/schema/dibuix", elementFormDefault =
    javax.xml.bind.annotation.XmlNsForm.QUALIFIED)
@javax.xml.bind.annotation.XmlAccessorType(value=
    javax.xml.bind.annotation.XmlAccessType.FIELD)
package ioc.dam.m6.exemples.dibuix.persistencia.xml.model;
```

Seguidament, caldrà crear i adaptar les classes *factory* que per defecte usa JAXB. Crearem en cada paquet una classe `ObjectFactory` que anotarem amb `@XmlRegistry`. Aquesta anotació indica a JAXB que es tracta d'una classe instanciadora i que, per tant, analitzant els seus mètodes es poden obtenir totes les classes usades durant la vinculació.

En el paquet del model propi, la classe `ObjectFactory` tindrà mètodes per generar rectangles, cercles, grups o dibuixos. Els mètodes instanciadors es compondran del prefix "*create*" i d'un sufix amb el nom que l'XML usarà com a referència d'aquesta classe:

```
@XmlRegistry
public class ObjectFactory {

    public ObjectFactory() {
    }

    public Grup createGrup() {
        return new Grup();
    }

    public Rectangle createRectangle() {
        return new Rectangle();
    }

    public Rectangle createRectangle(int x, int y, int amplada, int alcada) {
        return new Rectangle(x, y, amplada, alcada);
    }

    public Cercle createCercle() {
        return new Cercle();
    }

    public Cercle createCercle(int x, int y, int radi) {
        return new Cercle(x, y, radi);
    }

    public Dibuix createDibuix() {
        return new Dibuix();
    }
}
```

Per facilitar la codificació de crear objecte de tipus rectangle i cercle, afegirem dos mètodes més que acceptin paràmetres d'inicialització. Aquest mètodes no podran substituir els instanciadors per defecte, ja que JAXB usa sempre instanciadors sense paràmetres.

De forma semblant, en el paquet del model derivat:

```
@XmlRegistry
public class ObjectFactory {

    public ObjectFactory() {
    }

    public XmlGrup createGrup() {
```

```
8           return new XmlGrup();
9       }
10
11      public XmlRectangle createRectangle() {
12          return new XmlRectangle();
13      }
14
15      public XmlRectangle createRectangle(int x, int y, int amplada,
16          int alcada) {
17          return new XmlRectangle(x, y, amplada, alcada);
18      }
19
20      public XmlCercle createCercle() {
21          return new XmlCercle();
22      }
23
24      public XmlCercle createCercle(int x, int y, int radi) {
25          return new XmlCercle(x, y, radi);
26      }
27
28      public XmlDibuix createDibuix() {
29          return new XmlDibuix();
30      }
31  }
```

Anotacions de classe

L'anotació `@XmlElementRoot` serveix per indicar quina classe o quines classes són candidates a usar-se com a node arrel del document XML. Les instàncies d'aquestes classes poden accedir de manera directa o indirecta a tota la informació que s'emmagatzemarà al'XML. Cal recordar que XML no admet documents amb arrels múltiples i, per tant, no és possible seriar les dades a partir de dos o més objectes. En el nostre cas, la classe `arrel` correspon a `Dibuix`. Afegiu sobre la declaració de la classe l'`Anotació`.

```
1  @XmlRootElement(name="dibuix")
2  public class Dibuix extends ArrayList<Figura>{
3      ...
```

@XmlElementRoot accepta el paràmetre `nom`, el qual indica el nom que tindrà l'etiqueta arrel en el document XML. Feu el mateix a la classe `XmlDibuix`.

Emmagatzemeu els canvis i obriu la classe `Figura`. Afegirem una notació per indicar l'ordre en què voldrem escriure els seus atributs.

```
1  @XmlType(propOrder={"posicio","escala","rotacio"})
2  public abstract class Figura {
3      ...
```

L'Anotació `@XmlType` permet, entre d'altres coses, especificar l'ordre en què s'escriuran els seus camps. Observeu a la sintaxi com les Anotacions reben un conjunt de dades: el conjunt de dades es tanca entre claus, i per separar cada una de les dades s'intercala una coma.

Aquesta *Anotació* és opcional. De fet, si no la poséssim, tant el *marshaller* com l'*unmarshaller* funcionarien sense cap problema. De tota manera, es tracta d'una

anotació important perquè sovint els XML han de complir DTD o Schemes força restrictius que requereixen un ordre determinat de les etiquetes.

Farem quelcom semblant a la classe `FiguraSimple` i `Rectangle`.

```
@XmlType(propOrder = {"colorLinia","colorFigura"})
public class FiguraSimple extends Figura{
    ...

@XmlType(propOrder = {"alcada","amplada"})
public class Rectangle extends FiguraSimple{
    ...
```

No és necessari definir l'ordre dels atributs de la classe `Cercle` ni la classe `Grup`, perquè només en tenen un.

Les operacions equivalents al paquet del model derivat haurien de poder definir el mateix ordre, però com que no tenim accés a les classes derivades caldrà fer les anotacions a la pròpia classe, que és on hi trobarem els accessors (`get`si `set`s). A més, caldrà indicar que el tipus XML que aquesta classe representa no coincideix amb el seu nom.

```
@XmlType(name="Rectangle", propOrder={"posicio","rotacio","escala",
    "colorFigura","colorLinia","alcada","amplada"})
public class XmlRectangle extends Rectangle{
    ...
```

Cal realitzar la mateixa operació amb totes i cada una de les classes derivades:

```
@XmlType(name="Cercle", propOrder={"posicio","rotacio","escala",
    "colorFigura","colorLinia","radi"})
public class XmlCercle extends Cercle {
    ...

@XmlType(name="Grup", propOrder={"posicio","rotacio","escala","elements"})
public class XmlGrup extends Grup{
    ...
```

Anotacions de camps, mètodes o accessors

De vegades caldrà matisar vincles específics pels elements de les classes. Podem, per exemple, evitar que el valor d'un determinat camp es traspassi a l'XML. Això ho aconseguirem marcant-lo amb la notació `@XmlTransient`. És el que farem amb el camp TipusFigura de la classe `Figura` del model propi.

```
@XmlType(propOrder={"posicio","escala","rotacio"})
public abstract class Figura {
    @XmlTransient
    private TipusFigura tipusFigura=TipusFigura.INDETERMINADA;
    private Point posicio=null;
    ...
```

En el model derivat no cal fer-ho, perquè ja hem impedit l'accés a nivell del paquet. Malgrat tot, caldrà definir altres anotacions. Com que no hi ha accés directe als camps, JAXB no sabrà quin nom posar a les etiquetes de l'XML i, per defecte, JAXB usarà sempre el nom del camp associat. En el model derivat, caldrà indicar

a l'Anotació el nom desitjat a l'etiqueta. Les Anotacions s'han de fer només a un dels dos accessors, i per convenció solen posar-se en els mètodes de lectura (`get`), però s'accepten a qualsevol dels dos.

Les propietats comunes caldrà repetir-les a totes les classes:

```
...

    @XmlElement(name="posicio")
    @Override
    public Point getPosicio() {
        return super.getPosicio();
    }
    ...

    @XmlElement(name="escala")
    @Override
    public int getEscala() {
        return super.getEscala();
    }
    ...

    @XmlElement(name="rotacio")
    @Override
    public float getRotacio() {
        return super.getRotacio();
    }
    ...

    @XmlElement(name="colorFigura")
    @Override
    public Color getColorFigura() {
        return super.getColorFigura();
    }
    ...

    @XmlElement(name="colorLinia")
    @Override
    public Color getColorLinia() {
        return super.getColorLinia();
    }
    ...
```

I les específiques, a les classes corresponents:

```
@XmlElement(name="alcada")
    @Override
    public int getAlcada() {
        return super.getAlcada();
    }
    ...

    @XmlElement(name="amplada")
    @Override
    public int getAmplada() {
        return super.getAmplada();
    }
    ...
```

El tractament de col·leccions també s'especifica a nivell d'element o de propietat. Per defecte, les col·leccions Java es plasmaran al'XML com una seqüència d'elements descendents directes de la classe contenidora. És a dir, en el cas dels *grups*, que contenen una llista de *figures*, si no s'indica res més, la seqüència de figures es penjarà directament de l'etiqueta *grup*. No és això el que volem. De fet,

es vol fer dependre la seqüència de *figures* d'un element extra anomenat *colleccio*, el qual formaria part de l'element grup i contindria les *figures* de forma directa. JAXB disposa de l'*Anotació* `@XmlElementWrapper` per indicar la necessitat d'una etiqueta contenidora extra.

D'altra banda, els elements de la llista de figures a vegades seran rectangles, d'altres cercles i d'altres grups. La manera d'associar un tipus de dada amb una etiqueta es pot especificar aquí usant l'Anotació `@XmlElements`, a la qual se li indicarà usant una col·lecció de `@XmlElement` quines classes s'associaran a quines etiquetes.

```
public class Grup extends Figura{
    @XmlElementWrapper(name="colleccio", required=true)
    @XmlElements({
        @XmlElement(name="rectangle",type=Rectangle.class),
        @XmlElement(name="cercle",type=Cercle.class),
        @XmlElement(name="grup",type=Grup.class)
    })
    ArrayList<Figura> elements = new ArrayList<Figura>();
    ...
```

El mateix cal fer a la classe `XmlGrup`, però associant les classes corresponents.

```
@XmlElementWrapper(name="colleccio", required=true)
    @XmlElements({
        @XmlElement(name="rectangle",type=XmlRectangle.class),
        @XmlElement(name="cercle",type=XmlCercle.class),
        @XmlElement(name="grup",type=XmlGrup.class)
    })
    public ArrayList<Figura> getElements(){
        return elements;
    }
```

La classe `Dibuix` és també una llista de característiques molt similars a la classe `Grup`. Caldrà, doncs, una anotació semblant. Abans, però, haurem de salvar un problema: l'Anotació `@XmlElements` només pot ser definida a nivell de mètode o d'atribut i no pas de classe. Com que `Dibuix` hereta d'`ArrayList`, no disposa de cap atribut contenidor, la solució passa per crear un accessor de lectura on es pugui especificar la notació:

```
@XmlRootElement(name="dibuix")
public class Dibuix extends ArrayList<Figura>{
    @XmlElements({
        @XmlElement(name="rectangle",type=Rectangle.class),
        @XmlElement(name="cercle",type=Cercle.class),
        @XmlElement(name="grup",type=Grup.class)
    })
    public ArrayList<Figura> getFigures(){
        return this;
    }
    ...
```

Cal, també, realitzar una anotació idèntica a la classe `XmlDibuix`, substituint les classes originals per les derivades corresponents.

```
@XmlElements({
    @XmlElement(name="rectangle",type=XmlRectangle.class),
    @XmlElement(name="cercle",type=XmlCercle.class),
    @XmlElement(name="grup",type=XmlGrup.class)
```

```
})
public List<Figura> getFigures(){
    return this;
}
```

Rectificació de les anotacions a nivell de paquet

Malauradament, un cop s'han realitzat totes aquestes modificacions, si fem la prova intentant seriar una instància de Dibuix ens saltarà una excepció. Això és degut al fet que el nostre model utilitza dues classes pròpies de Java que JAXB no sap *mapar*. Ens referim a `Point` i a `Color`. JAXB necessita que totes les classes del model tinguin un constructor per defecte (constructor sense paràmetres) i que els atributs que calgui seriar tinguin accessors de lectura i escriptura (`get` i `set`). La classe `Color` no disposa de constructor per defecte, i la classe `Point` té un accessor de tipus *double* corresponent a un atribut de tipus enter i un accessor anomenat `getLocation` que no es correspon a cap atribut.

Per solucionar aquests problemes, JAXB disposa d'unes anotacions que permeten declarar classes o tipus alternatius, així com la forma de passar d'un tipus a un altre. Ens referim a `@XmlJavaTypeAdapter` i a la classe `XmlAdapter`. La primera és l'Anotació amb la qual s'informarà de quines classes han de fer d'adaptadors. La segona és una classe abstracta i parametritzada amb els tipus que caldrà intercanviar, d'on qualsevol adaptador haurà de derivar.

Per exemple, en passar al'XML, volem que les figures escriguin el color com si fos un valor numèric. Crearem una Classe que hereti de `XmlAdapter`, que haurà d'implementar els seus dos mètodes abstractes anomenats `unmarshal` i `marshal`. El mètode `unmarshal`permet convertir el valor obtingut des del'XML a un valor de tipus vàlid en el llenguatge Java. En el nostre cas, rebrà un enter per paràmetre i el transformarà a `Color`.

```
public class ColorAdapter extends XmlAdapter<Integer, Color> {

    @Override
    public Color unmarshal(Integer v) throws Exception {
        return new Color(v.intValue());
    }

    @Override
    public Integer marshal(Color v) throws Exception {
        return new Integer(v.getRGB());
    }

}
```

El mètode `marshal` fa l'operació inversa: rep per paràmetre el valor provinent de la classe Java i el converteix en un valor adequat per emmagatzemar al'XML.

Farem el mateix amb la classe `Point`. La diferència és, però, que els punts no poden transformar-se en un tipus simple, ja que sempre es necessiten dues coordenades. La classe alternativa la crearem nosaltres mateixos assegurant-nos que compleix els requisits de JAXB:

```
public class Posicio implements Serializable{
```

```
    private int coordenadaX;
    private int coordenadaY;
    public Posicio() {
    }

    public Posicio(Point p){
        this.coordenadaX=p.x;
        this.coordenadaY=p.y;
    }

    @XmlAttribute(name="x")
    public int getCoordenadaX() {
        return coordenadaX;
    }

    public void setCoordenadaX(int coordenadaX) {
        this.coordenadaX = coordenadaX;
    }

    @XmlAttribute(name="y")
    public int getCoordenadaY() {
        return coordenadaY;
    }

    public void setCoordenadaY(int coordenadaY) {
        this.coordenadaY = coordenadaY;
    }
}
```

També implementarem el seu adaptador:

```
public class PointAdapter extends XmlAdapter<Posicio, Point> {

    @Override
    public Point unmarshal(Posicio v) throws Exception {
        return new Point(v.getCoordenadaX(), v.getCoordenadaY());
    }

    @Override
    public Posicio marshal(Point v) throws Exception {
        return new Posicio(v);
    }

}
```

Finalment, per tal que els canvis tinguin efecte, caldrà declarar-ho en forma d'Anotacions. Això ho farem a nivell de paquet. La notació rep dos paràmetres: el paràmetre *type* identifica la classe incompatible, i el paràmetre *value* especificarà per quina classe caldrà substituir-la. Per tal que es puguin definir tants adaptadors com sigui necessari, aquests es passaran per paràmetre de l'anotació `XmlJavaTypeAdapters`.

```
@javax.xml.bind.annotation.XmlSchema(
    namespace = "http://xml.netbeans.org/schema/dibuix",
    elementFormDefault = javax.xml.bind.annotation.XmlNsForm.QUALIFIED)

@javax.xml.bind.annotation.adapters.XmlJavaTypeAdapters({
    @javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter(
        value=ioc.dam.m6.exemples.dibuix.persistencia.xml.binding.
            ColorAdapter.class,
        type=java.awt.Color.class),
    @javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter(
        value=ioc.dam.m6.exemples.dibuix.persistencia.xml.binding.
            PointAdapter.class,
        type=java.awt.Point.class)
```

```
})

package ioc.dam.m6.exemples.dibuix.persistencia.xml.binding.model;
```

Implementació del controlador

Tal com hem fet en anteriors ocasions, crearem una classe `Controlador` que aglutini un conjunt d'utilitats per facilitar les crides a la seriació i deseriació en format XML usant les eines JAXB. Bàsicament, la Utilitat que construirem ha de facilitar la creació d'un context específic, així com l'escriptura des del model al'XML i la lectura des del'XML al model.

Aquesta classe l'anomenarem `BindingCtrl`. Contindrà una instància d'Utilitats, un `File` referenciant el fitxer d'emmagatzematge XML i una cadena representant el nom del paquet on es trobaran les classes del model de dades. Crearem un mètode *init* que inicialitzarà el controlador creant un JAXBContext. També disposarà d'utilitats per realitzar la seriació/deseriació a l'XML.

```
public class BindingCtrl {
    protected Utilitats utilitats = new Utilitats();
    protected File file = null;
    protected boolean init = false;
    protected JAXBContext context;
    protected String nomPaquet=null;

    public BindingCtrl() {
    }

    public BindingCtrl(String ctx) {
        nomPaquet = ctx;
    }

    public File getFile() {
        return file;
    }

    public void init() throws JAXBException {
        if(nomPaquet!=null){
            context = JAXBContext.newInstance(nomPaquet);
        }else{
            context = JAXBContext.newInstance(classFactory);
        }
        init=true;
    }

    public void setFile(File file) {
        this.file = file;
    }

    protected void escriu(Object object) throws BindingCtrlException{
        FileOutputStream out = null;
        try {
            if(!init){
                init();
            }
            Marshaller marshaller = context.createMarshaller();
            marshaller.setProperty(Marshaller.JAXB_ENCODING, "UTF-8");
            marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
            out = new FileOutputStream(file);
            marshaller.marshal(object, out);
        } catch (FileNotFoundException ex) {
            throw new BindingCtrlException(ex);
```

```
        } catch (JAXBException ex) {
            throw new BindingCtrlException(ex);
        } finally {
            utilitats.intentarTancar(out);
        }
    }

    protected Object llegeix() throws BindingCtrlException {
        Object object = null;
        FileInputStream in=null;
        try {
            if(!init){
                init();
            }
            Unmarshaller unmarshaller = context.createUnmarshaller();
            in = new FileInputStream(file);
            object = unmarshaller.unmarshal(in);
        } catch (FileNotFoundException ex) {
            throw new BindingCtrlException(ex);
        } catch (JAXBException ex) {
            throw new BindingCtrlException(ex);
        } finally {
            utilitats.intentarTancar(in);
        }
        return object;
    }
}
```

A partir d'aquesta classe, per herència podem crear controladors específics per cada model concret. Per exemple, la del model derivat s'implementarà fent el següent:

```
public class ModelDerivatCtrlDibuix extends BindingCtrl{
    Dibuix root;

    public ModelDerivatCtrlDibuix(){
        this(new Dibuix());
    }

    public ModelDerivatCtrlDibuix(Dibuix dibuix){
        super(dibuix.getClass().getName().substring(0,dibuix.getClass().
            getName().lastIndexOf(".")));
        this.root = dibuix;
    }

    public ModelDerivatCtrlDibuix(Dibuix dibuix, File file){
        this(dibuix);
        this.file=file;
    }

    public void emmagatzemar() throws xmlDibuixException{
        try {
            escriu(root);
        } catch (BindingCtrlException ex) {
            throw new xmlDibuixException(ex);
        }
    }

    public void recuperar() throws xmlDibuixException {
        try {
            root.clear();
            root.addAll((Dibuix) this.llegeix());
        } catch (BindingCtrlException ex) {
            throw new xmlDibuixException(ex);
        }
    }

    public Dibuix getDibuix() {
```

```
37          return root;
38      }
39
40  }
```