UNIVERSITAT OBERTA DE CATALUNYA

Enginyeria Tècnica Informàtica de Gestió

GENERACIÓ AUTOMÁTICA DE L'ESQUEMA DE BASES DE DADES OBJECTE-RELACIONAL D'UNA APLICACIÓ A PARTIR DEL SEU DIAGRAMA DE CLASSES.

Alumne: PÉREZ VIDAL, JOSÉ ANTONIO

Dirigit per: JORDI CABOT SAGRERA

CURS 2003-04 (Setembre-Gener)

GENERACIÓ AUTOMÁTICA DE L'ESQUEMA DE BASES DE DADES OBJECTE-RELACIONAL D'UNA APLICACIÓ A PARTIR DEL SEU DIAGRAMA DE CLASSES.

L'objectiu d'aquest projecte de fi de carrera és obtenir una aplicació en llenguatge JAVA amb capacitat de generar a partir d'un diagrama de classes la seva representació en un esquema d'una base de dades objecte-relacional Oracle 9i. Per aconseguir-ho s'aprofita la capacitat de les eines de disseny UML (eines CASE) per a generar fitxers XMI, que són subconjunts del llenguatge XML per a la descripció de models UML i que contenen la informació del diagrama de classes dintre de les marques típiques dels fitxers XML. L'aplicació JAVA rep com a entrada aquest fitxer XMI. Manté en memòria el fitxer en forma d'un arbre de nodes aplicant un model de processament DOM per a XML del qual disposa el llenguatge JAVA mitjançant el seu API JAXP. Així l'aplicació es capaç de recórrer tot el document recollint unes dades determinades. Un cop recorregut tot el document s'analitzen les dades recollides i es classifiquen segons el seu tipus. Quan ja es tenen totes les dades classificades es passa a reconstruir la informació del document analitzat dins d'un fitxer script amb les sentències SQL necessàries per a la creació de l'esquema de la base de dades. Per tant, l'aplicació en llenguatge JAVA rep d'entrada un fitxer XMI i genera de sortida un fitxer pla al que se li dona extensió SQL. Aquest fitxer SQL està capacitat per a plasmar els tipus objecte, relacions d'herència simple i relacions d'associació d'agregació i de referència que s'hagin descrit al diagrama de classes. L'aplicació tracta fitxers XMI generats pel programa Poseidon for UML, abans de provar-ho amb altres eines CASE caldrà comprovar si els fitxers XMI que generen aquestes eines contenen les mateixes marques.

GENERACIÓN AUTOMÁTICA DEL ESQUEMA DE BASE DE DATOS OBJETO-RELACIONAL DE UNA APLICACIÓN A PARTIR DE SU DIAGRAMA DE CLASES.

El objetivo de este proyecto de fin de carrera es obtener una aplicación en lenguaje JAVA con capacidad para generar a partir de un diagrama de clases su representación en un esquema de una base de datos objeto-relacional Oracle 9i. Para conseguirlo se aprovecha la capacidad de las herramientas de diseño UML (herramientas CASE) para generar ficheros XMI, que son subconjuntos del lenguaje XML para la descripción de modelos UML, los cuales contienen la información del diagrama de clases dentro de las marcas típicas de los ficheros XML. La aplicación JAVA recibe como entrada este fichero XMI. Mantiene en memoria el fichero en forma de un árbol de nodos aplicando un modelo de procesamiento DOM para XML, del cual dispone el lenguaje JAVA gracias a su API JAXP. De este modo la aplicación es capaz de recorrer todo el documento recogiendo unos datos determinados. Una vez recorrido todo el documento se analizan los datos recogidos y se clasifican según su tipo. Cuando ya se tienen todos los datos clasificados se pasa a reconstruir la información del documento analizado dentro de un fichero script con las sentencias SQL necesarias para crear el esquema de la base de datos. Por lo tanto, la aplicación en lenguaje JAVA recibe de entrada un fichero XMI y genera de salida un fichero plano al cual se le da extensión SQL. Este fichero SQL está capacitado para plasmar los tipos objeto, relaciones de herencia simple y relaciones de asociación de agregación y de referencia que se hayan descrito en el diagrama de clases. La aplicación trata ficheros XMI generados por el programa Poseidon for UML, antes de probarlo con otras herramientas CASE será necesario comprobar si los ficheros XMI que generan estas herramientas contienen las mismas marcas.

AUTOMATIC GENERATION OF THE OBJECT-RELATIONAL DATABASE SCHEMA OF AN APPLICATION FROM ITS CLASS DIAGRAM.

The aim of this dissertation is to get a JAVA application which will be able to generate a representation of an Oracle 9i Object-Relational database schema. For this purpose, it takes advantage of the capabilities of UML design tools (CASE tools) to generate XMI files, which are a subclass of XML language for describing UML model information about class diagram between XML typical tags. The JAVA application loads the file in memory on the way of a nodes tree applying a DOM processing model for XML, which is provided by JAVA programming language thanks to its JAXP API. In this way application is able to walk through all document collecting specific data. Once this check is over, application analyses the whole collected data and it classifies these information depending on its nature. When all the information is classified, it can be used for rebuilding the document information into an SQL script file, which will contain the necessary SQL sentences to create the database schema. So, this java application receives an input as a XMI file and then it generates an output as flat file with a SQL extension. This SQL file is able to represent object types, single inheritance relations, aggregation and reference association relations which could be described in the diagram class. Application works with XMI files generated by Poseidon for UML application; it will be necessary to check if the XML files generated by other CASE tools contain the same tags before testing with them.

INDEX

1. Glossari 6
2. Introducció 7
   2.1 Què és l'UML ? 7
   2.2 Què és un SGBDOR ? 8
   2.3 Què són l'XML i l'XMI ? 8
   2.4 Models de processament d'XML 9
   2.5 Quin és l'objectiu d'aquest treball ? 10
   2.6 Quin és l'abast d'aquest treball ? 10
3. Anàlisi de l'aplicació 11
   3.1 Diagrama de classes de l'aplicació 11
   3.2 Classes de l'aplicació 12
      3.2.1 AnalitzadorXMI 12
      3.2.2 Classe 13
      3.2.3 Atribut 15
      3.2.4 TipusDada 16
      3.2.5 Herencia 17
      3.2.6 Associacio 18
      3.2.7 AssociacioNN 20
      3.2.8 Utils 21
   3.3 Anàlisi del funcionament de l'aplicació 23
   3.4 Com executar l'aplicació 41
   3.5 Línies futures de treball 42
4. Agraïments 43
5. Bibliografia 43
6. Codi dels programes principals de l'aplicació 45
   6.1 AnalitzadorXMI.java 45
   6.2 Classe.java 57
   6.3 Utils.java 60

GLOSSARI

API : Acrònim de Application Programming Interface.

CASE : Acrònim de Computer-Aided Software Engineering (enginyeria de software assistida per computador).

DOM : Acrònim de Document Object Model.

JAVA : Llenguatge de programació orientat a objectes.

JAXP : Acrònim de JAVA API for XML Processing.

SAX : Acrònim de Simple API for XML.

SGBDOO : Acrònim de Sistema Gestor de Bases de Dades Orientat a Objectes.

SGBDOR : Acrònim de Sistema Gestor de Bases de Dades Objecte-Relacional.

SGBDR : Acrònim de Sistema Gestor de Bases de Dades Relacional.

Script : Fitxer de text que conté un seguit d'ordres que s'executen seguint un ordre seqüencial.

SQL : Acrònim de Structured Query Language (llenguatge de consulta estructurat).

UML : Acrònim de Unified Modeling Language (Llenguatge de modelatge unificat).

XMI : Acrònim de XML Metadata Interchange.

XML : Acrònim de eXtensible Markup Language.

2. Introducció

Abans d'aprofundir en el funcionament de l'aplicació desenvolupada cal explicar alguns conceptes que són imprescindibles entendre :

2.1 Què es l'UML ?

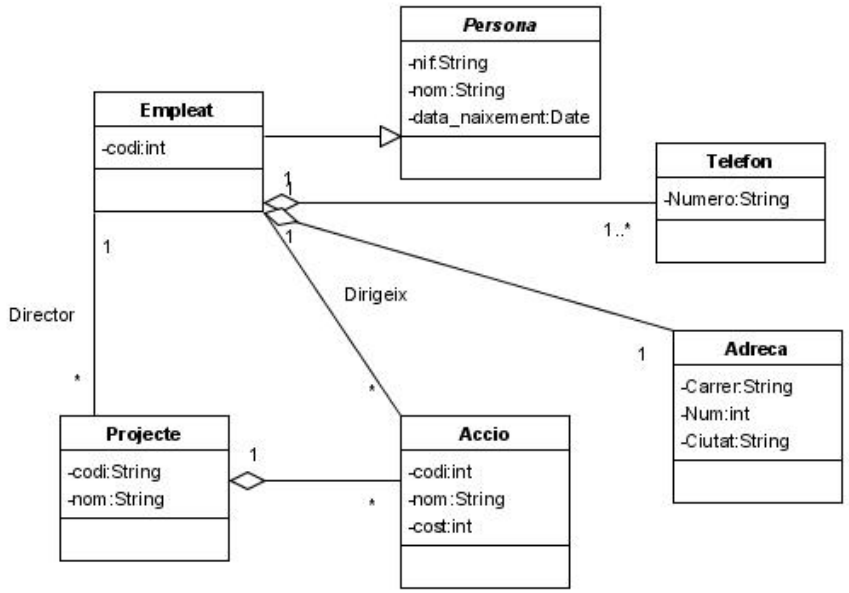
L'UML és un llenguatge de modelatge visual que es fa servir per a especificar, visualitzar, construir i documentar elements d'un sistema de software. Permet entendre, dissenyar, configurar, mantenir i controlar la informació sobre els sistemes a construir. No es tracta de cap llenguatge de programació, si no d'un llenguatge de propòsit general per al modelatge orientat a objectes.

No és un llenguatge propietari. Va ser creat sota els auspicis de l'OMG (Object Management Group) davant la necessitat d'un estàndard en el modelatge de sistemes de software. És actualment el llenguatge de modelatge més emprat dins la indústria per a l'especificació d'aplicacions informàtiques.

Entre les diferents especificacions que permet l'UML trobem la de la part estàtica de l'aplicació (**diagrama de classes**), la qual representa el conjunt de classes de l'aplicació i les seves interrelacions, permetent guardar la informació necessària per a facilitar un correcte desenvolupament de l'aplicació.

La majoria de les eines CASE per al disseny UML inclouen capacitats de generació de codi que, a partir del diagrama de classes, permeten generar l'estructura de la base de dades de l'aplicació. No obstant, aquest procés és un procés propietari (depèn de cada eina) i només serveix per a bases de dades relacionals; així, tenim els exemples de les aplicacions WMEE (Wonder Machine Enterprise Edition de Sygel), FL (Foundation Layer de Novosoft) o VisualUML (de Visual Object Modelers) entre moltes altres. Aquest factor limita les possibilitats d'utilitzar aquest tipus de generació automàtica, ja que la tendència actual és utilitzar cada cop més les funcionalitats que proporcionen els SGBDOR.

Exemple de diagrama de classes :



2.2 Què és un SGBDOR ?

El món de les bases de dades ha anat evolucionant des de l'emmagatzematge de les dades en fitxers fins a l'emmagatzematge en bases de dades orientades a objectes. Pel camí han quedat models com el jeràrquic o el model en xarxa. Actualment ens trobem en una situació pont entre el model relacional i el model orientat a objectes. En mig d'aquestes dues solucions trobem el model objecte-relacional.

Un SGBDR és un model de persistència de la informació on les dades s'organitzen en forma de taules amb files i columnes.

Un SGBDOO és un model de persistència de la informació que permet guardar les dades en forma d'objectes amb diferents propietats.

Un **SGBDOR** permet guardar les dades en forma d'objectes que al seu temps es poden guardar en taules especials. Ens dona els avantatges dels SGBDOO i, al mateix temps, els avantatges dels SGBDR. El seu principal valor consisteix en que ens permet aprofitar el millor de tots dos models sense haver de prescindir de cap d'ells.

2.3 Què són l'XML i l'XMI ?

L'XML és un llenguatge que permet la creació de llenguatges de marca (és un Meta-llenguatge de marca) orientats cap a un tipus específic de contingut.

Un llenguatge de marca fa servir etiquetes incrustades al text per a descriure les seves diferents parts.

En sí mateix, XML és també un llenguatge de marca que pot executar-se des de qualsevol plataforma, sistema operatiu o entorn de treball, i està dissenyat per a proporcionar als programadors un mecanisme que pugui descriure la informació. Originalment va ser dissenyat per a projectes de publicació, però ha estat desenvolupat per a facilitar i millorar l'intercanvi de dades dins la xarxa. Permet als programadors escriure les seves pròpies definicions del tipus de document (DTDs) descrivint conjunts d'etiquetes i atributs que defineixen tipus específics de contingut.

Es desenvolupa sota el control del World Wide Web Consortium's (W3C) XML Working Group per a assegurar que els seus mecanismes siguin regulars i no afavoreixin a cap fabricant en particular. Mentre que cada vocabulari tindrà un propòsit i un tipus de contingut específics, tots han de respectar les regles XML, tal que siguin subconjunts del propi XML.

L'XMI és un subconjunt o vocabulari XML que conté la informació d'un model especificat en UML.

2.4 Models de processament d'XML

Per a la manipulació del fitxer XMI existeix diferents analitzadors de sintaxi (*parsers)* java, els quals només es diferencien en com fan el recorregut per la informació. En principi es va haver de triar entre aquestes tres metodologies : SAX, DOM i JDOM.

SAX és un *parser* que manipula només la informació que se l'indica, és un mètode selectiu (funciona per events que es llencen quan l'analitzador es troba amb un element determinat) on cal dir-li quins nodes ens interessen i ell els agafarà. El seu principal avantatge és que és molt ràpid i sense despesa de memòria.

DOM és un *parser* de recorregut total del document XML en forma d'arbre. Així, DOM veu el document com un arbre amb un node arrel i altres nodes que van baixant depenent uns dels altres fins arribar a les fulles; fent servir aquest *parser* obtenim un recorregut on a partir d'un node principal arribem fins les fulles d'aquest i, un cop fet el recorregut sobre aquest node principal fins al seu final, es passa a fer el recorregut del següent node principal fins a la seves fulles, en cas d'existir aquest node. El recorregut no acaba fins que no s'ha recorregut tot l'arbre sencer. Aquest mètode té el desavantatge que cal tenir tot l'arbre en memòria, però es podrà recórrer totes les vegades que calgui.

JDOM és un altre *parser* que fa recorregut del document en forma d'arbre però pensat especialment per a metodologia JAVA (els anteriors no són específics per a JAVA). El seu desavantatge és que encara està en una etapa massa inicial, però en un futur serà molt més òptim que no pas SAX i DOM, quan es vulgui manipular documents XML des de JAVA.

El llenguatge JAVA també disposa d'unes llibreries de classes agrupades dins un API que es diu **JAXP.** JAXP per tant no és cap parser**,** si no un API o grup de classes JAVA que aglutina en diferents llibreries les metodologies SAX i DOM. Per tant amb JAXP tenim el millor de ambdues metodologies.

A l'haver d'escollir entre aquestes metodologies, un cop analitzats els diferents punts positius i negatius de cadascuna d'elles, sense dubtar-ne vam triar la metodologia JAXP, que ens permetrà fer servir la metodologia DOM (la que més ens interessa doncs ens permet agafar allò que volem tal com ho anem trobant al recorregut de l'arbre en tants recorreguts com calgui del document) i també podríem fer servir llibreries SAX de JAVA, com per exemple per a controlar els errors de manipulació dins la nostra aplicació (amb la classe org.xml.sax.ErrorHandler). La versió de JAXP que farem servir és la JAXP 1.2 .

2.5 Quin és l'objectiu d'aquest treball ?

El que es pretenia en aquest projecte era construir una aplicació que permetés la generació d'una base de dades *objecte-relacional* a partir d'un diagrama de classes qualsevol. L'aplicació havia de poder treballar amb models especificats utilitzant qualsevol de les eines CASE actuals i per tant llegir el diagrama de classes a partir del seu corresponent fitxer XMI. A l'ésser l'XMI un format estàndard, un mateix model especificat utilitzant eines CASE diferents hauria de tenir la mateixa representació en XMI; això no és del tot cert doncs sovint trobem dues eines de modelatge UML que permeten generar fitxers XMI on els noms de les marques poden no coincidir exactament; un exemple d'aquestes diferències el trobem en el fitxer XMI generat per les aplicacions *Poseidon for UML* i *VisualUML*.

Per tant, l'aplicació desenvolupada té com a entrada un fitxer XMI i genera com a sortida a partir d'aquest un script amb les sentències de creació de la base de dades *objecte-relacional* corresponent. Degut a les possibilitats que ofereixen els SGBDOR, la transformació del diagrama de classes és molt més fàcil que si s'hagués de fer la transformació per a un sistema gestor relacional. En utilitzar XMI com a format d'entrada, l'aplicació podria utilitzar-se per estendre la funcionalitat de qualsevol eina CASE que hi hagi al mercat (quasi totes les eines CASE actuals permeten generar models en XMI).

2.6 Quin és l'abast d'aquest treball ?

Aquesta aplicació ha de permetre, a partir de qualsevol eina CASE del mercat amb capacitat de generar models en XMI, obtenir un fitxer script d'operacions DDL (operacions de modificació en l'estructura d'una base de dades, com pot ser la creació o modificació de tipus objecte o de taules d'aquests tipus objecte) que a l'executar-se en entorn d'Oracle SQLPLUS (o en general en qualsevol entorn d'execució de sentències SQL) representi fidelment a la base de dades tots els elements plasmats a un diagrama de classes qualsevol.

L'aplicació ha estat desenvolupada per a funcionar amb *Poseidon for UML*, que és una entre moltes de les eines que existeixen al mercat per al disseny de models UML, i com ja hem comentat, encara que l'objectiu era que pogués funcionar amb qualsevol eina CASE que generés un fitxer XMI, les marques d'aquest fitxer poden no correspondre's exactament entre diverses eines. De totes formes, l'adaptació al XMI que faci servir una o altra eina no és gens difícil, afectant a un grup molt reduït de línies de codi de l'aplicació.

Per limitacions de temps, el desenvolupament d'aquesta aplicació afecta a un número limitat d'elements del diagrama de classes :

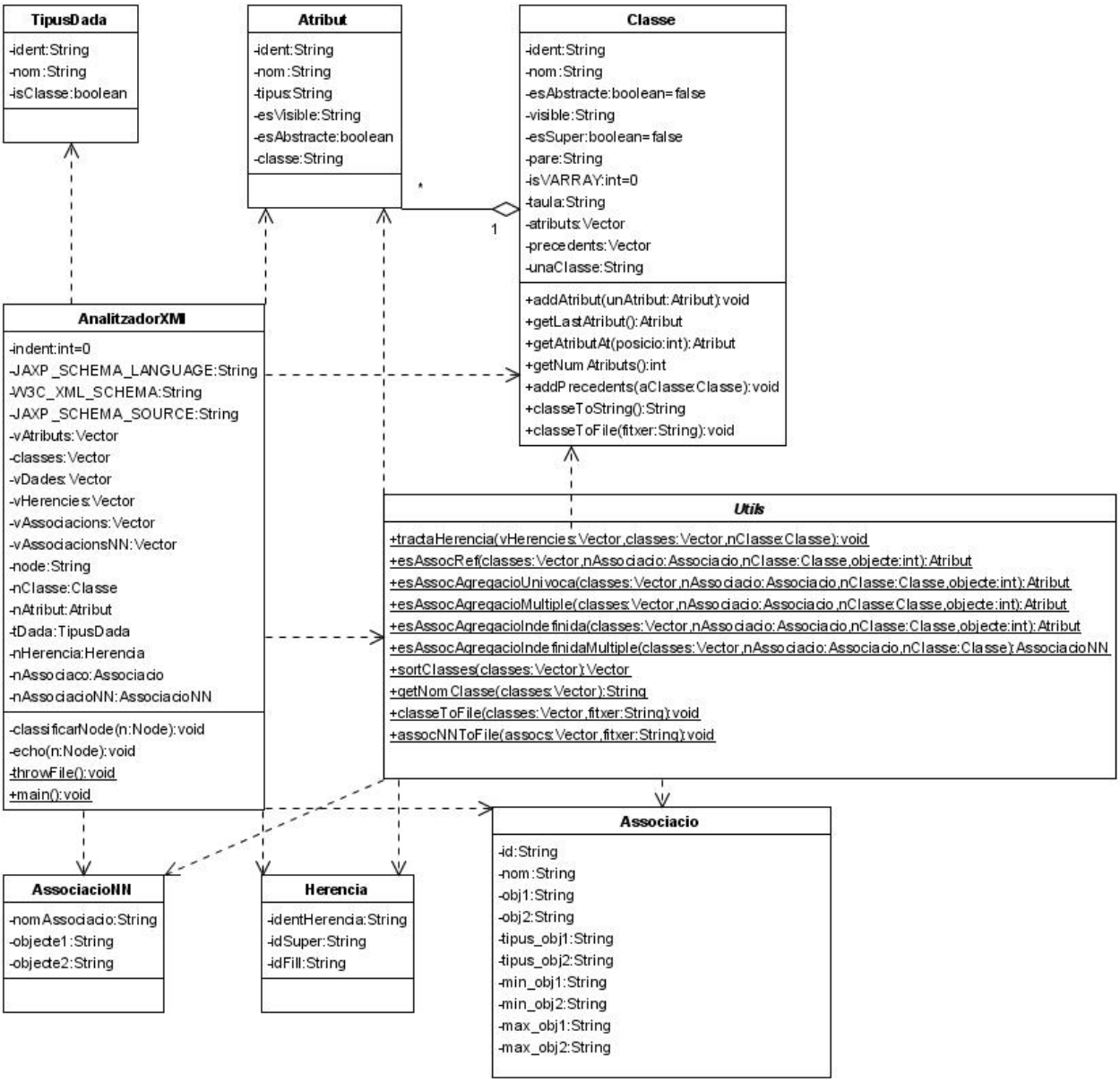
- Classes amb els seus atributs i els tipus d'aquests atributs.
- Relacions d'herència simple entre classes.
- Associacions de referència entre classes.
- Associacions d'agregació entre classes.

En combinació amb altres projectes de fi de carrera que s'estan duent a terme es podran generar també les restriccions que formen part d'un diagrama de classes en UML.

3. Anàlisi de l'aplicació.

Abans de res, es farà una presentació de les classes que composen el producte indicant a grans trets per a que serveix cadascuna, encara que aquesta funcionalitat es veurà més clarament després amb un exemple en el que anirem incrementant una per una cadascuna de les diferents necessitats amb les que ens anirem trobant en el nostre camí que ens ha de portar a l'obtenció d'un script el més semblant possible al diagrama UML inicial.

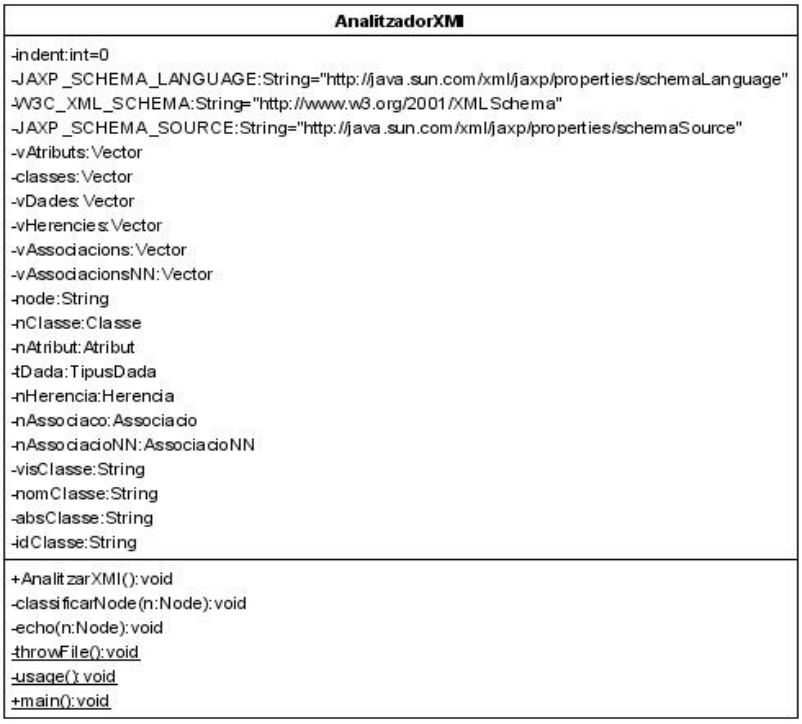
3.1 Diagrama de classes de l'aplicació



3.2 Classes de l'aplicació

3.2.1 AnalitzadorXMI

Classe que s'encarrega d'engegar el procés cridant primer al *parser* **DOM** i durant el recorregut pels diferents nodes anirà analitzant amb l'ajuda de la resta de classes el document XMI passat per paràmetre. A l'analitzar el document, en el mètode **echo( )**, va agafant tots els nodes del document i només analitzarà a un nivell més exhaustiu aquells que pertanyin als tipus que se li indiquin : actualment seran Class, Attribute, DataType, Generalitzation, Association, AssociationEnd i MultiplicityRange. Si són de qualsevol d'aquests tipus aleshores caldrà un recorregut de les propietats dels nodes on per cada una d'elles es cridarà el mètode **classificarNode(Node)**, el qual quan té les propietats necessàries d'un element crea la instancia de la classe determinada pel node i les seves propietats, podent ser Classe, Atribut, TipusDada, Herència o Associació, i guarda aquesta instància en un vector d'objectes del mateix tipus (classes, atributs, tipusdades, herències i associacions). Un cop fet el recorregut complet de l'arbre que representa el document XMI llença un mètode (**throwFile**()) on agrupa i reconstrueix cadascú dels objectes obtinguts del recorregut, seguidament crida un altre mètode (**classeToFile**()) el qual pertany a la classe **Utils** i que permet l'escriptura dels objectes recuperats dins un fitxer pla amb l'extensió sql, basant-se en unes regles definides en aquest darrer mètode. Per últim, després de gravar els tipus objectes amb els seus atributs dins aquest fitxer pla, es crida un mètode (**assocNNToFile**()), el qual també pertany a la classe **Utils** i permet crear les taules necessàries per a representar associacions d'agregació n a n entre dos objectes dins el mateix fitxer pla d'abans, amb extensió sql. Si no es passen els paràmetres correctes es dispararà el mètode **usage()** mostrant un missatge indicant l'ús correcte de l'aplicació. Aquest darrer mètode d'ajuda també es pot cridar des de línia de comandes executant "java AnalitzadorXMI –us" o "java AnalitzadorXMI –help".



3.2.2 Classe

Aquesta classe representa els tipus objecte d'una base de dades OO. Li hem dit Classe per que va ser el primer nom triat, encara que també se li hagués pogut dir TipusObjecte.class, doncs pot representar ambdues coses. És instanciada per la classe AnalitzadorXMI, al mètode classificarNode(Node).

Es composa d'aquests **atributs** :

ident : representa un número identificatiu del tipus objecte en qüestió i que el permetrà identificar quan participi en qualsevol tipus d'associació.

nom : representa el nom del tipus objecte.

esAbstracte: atribut booleà que ens indica si aquest tipus objecte és instanciable.

visible : atribut que ens indica si el tipus objecte és públic o privat

esSuper: atribut booleà que ens indica si d'aquest tipus objecte existeix algun altre que hereta.

pare : atribut que ens indica quin és el tipus objecte (si existeix) del qual aquest hereta. Si no hereta de cap aleshores aquest camp és buit.

isVARRAY : atribut enter que indica quants components tindrà el VARRAY d'objectes d'aquest tipus objecte que caldrà crear. Si és igual a zero aleshores no caldrà crear cap VARRAY.

taula : atribut que guarda el nom de la taula que cal crear del tipus objecte actual. Si la cadena és buida aleshores voldrà dir que no cal crear aquesta taula.

atributs : vector de tipus Atribut que guarda tots els atributs que té aquest tipus objecte i que caldrà indicar a l'hora de fer l'script.

precedents : vector de tipus Classe que guarda tots aquells tipus objecte que cal crear a l'script abans que l'actual per tal que no hi hagi errors a l'hora de la creació.

unaClasse : atribut que guarda la cadena de creació del tipus objecte actual en relació a les característiques del mateix. Aquesta cadena és la que s'afegeix al fitxer pla que serà l'script.

Respecte als seus **mètodes**, a banda dels constructors, escriptors i lectors de cada atribut, tenim :

addAtribut(**Atribut a**) : afegeix un atribut al vector atributs

getLastAtribut() : retorna el darrer atribut del vector atributs

addPrecedents(**Classe c**): afegeix un tipus objecte al vector de precedents

classeToString() : maqueta una cadena amb les sentències necessàries per a crear un tipus objecte a partir de les característiques de l'objecte instanciat actual. Aquesta cadena la guarda a l'atribut unaClasse.

classeToFile(**String fitxer**): introdueix la cadena de l'atribut unaClasse dins el fitxer pla de l'script, la ruta i el nom del qual li passem per paràmetre.

equals(**Classe c**): mètode que comprova si dos tipus objectes són iguals (l'actual objecte amb el passat per paràmetre), retornant un booleà a cert si ambdós tenen el mateix identificador, fals en cas contrari.

3.2.3 Atribut

Aquesta classe representa els atributs o propietats dels tipus objecte. És instanciada per la classe AnalitzadorXMI, al mètode classificarNode(Node).

Es composa d'aquests **atributs** :

ident : representa un número identificatiu de l'atribut en qüestió i que el permetrà identificar dins del tipus objecte al que pertany.

nom : representa el nom de l'atribut.

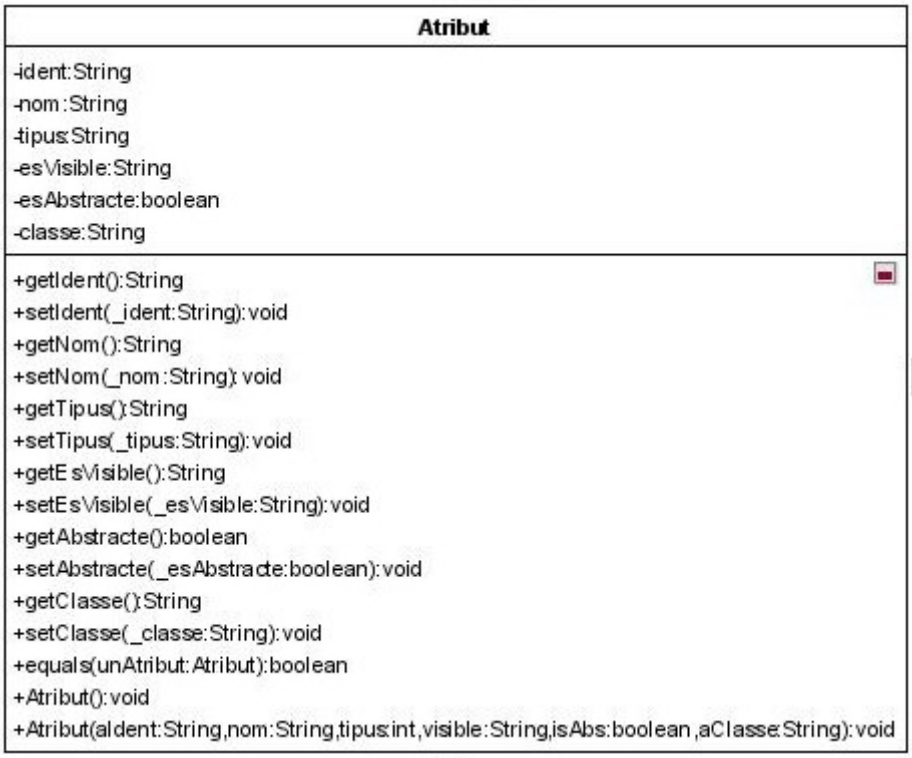
tipus : cadena que representa el tipus de dada que és l'atribut. Està directament relacionat amb el nom d'un objecte TipusDada, el qual analitzarem seguidament. Un tipus pot ser, per exemple, Integer o String.

visible : atribut que ens indica si l'atribut és públic o privat respecte al tipus objecte al que pertany.

esAbstracte: atribut booleà que ens indica si aquest és abstracte o no.

Classe: cadena que indica el nom del tipus objecte al que pertany l'atribut.

L'unic **mètode** que conté el qual no és constructor, escriptor ni lector de cap dels seus atributs és **equals**(**Atribut a**), el qual comprova si dos atributs són iguals si tenen el mateix identificador i el mateix nom, en aquest cas retorna un booleà a cert, en cas contrari retornaria fals.



3.2.4 TipusDada

Aquesta classe representa els tipus als que pertanyen els diferents atributs d'un tipus objecte. És instanciada per la classe AnalitzadorXMI, al mètode classificarNode(Node).

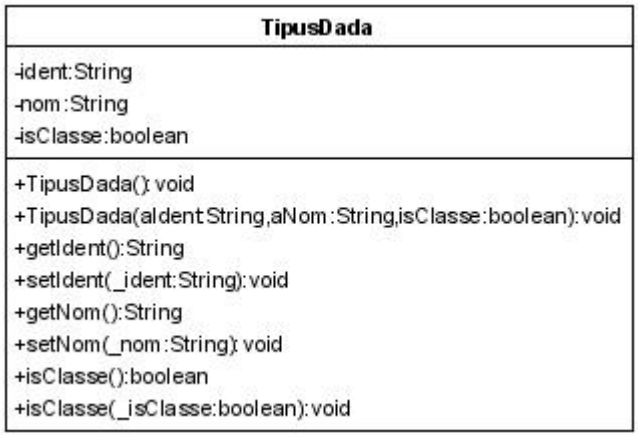
Es composa d'aquests **atributs** :

ident : representa un número identificatiu del tipus de dada en qüestió i que el permetrà identificar per a cada atribut d'aquest tipus. Així, per exemple, tots els atributs String tindran per al seu tipus de dada el mateix identificador.

nom : representa el nom de l'atribut.

isClasse : indica si un tipus de dada és una Classe o no. Així, String o Empleat (si es fes servir com atribut), retornarien cert, mentre que int o float retornarien fals.

Aquesta classe només té mètodes constructors, de lectura i escriptura.



3.2.5 Herencia

Aquesta classe representa una associació d'herència entre dos tipus objecte. És instanciada per la classe AnalitzadorXMI, al mètode classificarNode(Node).

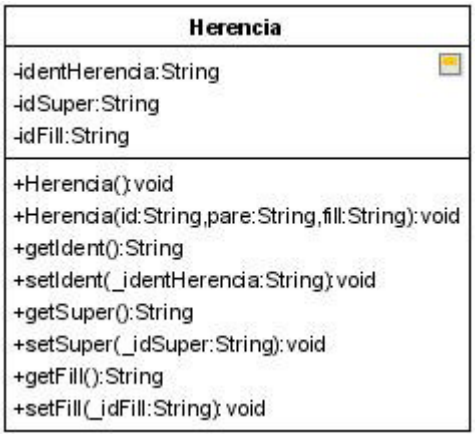
Es composa d'aquests **atributs** :

identHerencia : representa un número identificatiu de l'associació en qüestió i que la permetrà identificar dins de totes les relacions d'herència del diagrama.

idSuper : aquest atribut conté l'identificador del tipus objecte del qual s'hereta.

idFill : aquest atribut conté l'identificador del tipus objecte que hereta.

Aquesta classe només té mètodes constructors, de lectura i escriptura.



3.2.6 Associació

Aquesta classe representa una associació d'agregació o de referència entre dos tipus objecte. És instanciada per la classe AnalitzadorXMI, al mètode classificarNode(Node).

Es composa d'aquests **atributs** :

Id : representa un número identificatiu de l'associació en qüestió.

nom: representa el nom de l'associació en sí mateixa.

obj1: representa l'identificador del primer tipus objecte que apareix en l'associació.

obj2: representa l'identificador del segon tipus objecte de l'associació.

isNavObj1: representa el sentit de la navegació de l'associació. Si l'agregació es dirigeix cap a un dels dos sentits aleshores aquest atribut té valor cert.

isNavObj2: com a l'anterior, si l'agregació es dirigeix cap a un dels dos sentits aleshores aquest atribut té valor cert, en cas contrari tindrà valor fals.

tipus_obj1: aquest atribut indica el tipus d'associació que té l'objecte 1, en una de referència sempre serà "none", mentre que en una d'agregació podrà ser "none" o "aggregate".

tipus_obj2: tal com a l'anterior.

min_obj1: l'atribut conté el valor mínim de les participacions que l'objecte tindrà en l'associació, i que poden ser des de zero a n.

min_obj2: l'atribut conté el valor mínim de les participacions que l'objecte tindrà en l'associació, i que poden ser des de zero a n.

max_obj1: l'atribut conté el valor màxim de les participacions que l'objecte tindrà en l'associació, i que poden ser des d'1 a n.

max_obj2: l'atribut conté el valor màxim de les participacions que l'objecte tindrà en l'associació, i que poden ser des d'1 a n.

Aquesta classe, a banda dels mètodes constructors, escriptors i lectors, només té un **mètode** que retorna una cadena amb tots els valors de l'associació, **getAssociacio**().

Associacio
-id:String -nom:String -obj1:String -obj2:String -isNavObj1:boolean -isNavObj2:boolean -tipus_obj1:String -tipus_obj2:String -min_obj1:String -min_obj2:String -max_obj1:String -max_obj2:String
+Associacio():void +Associacio(id:String,nom:String,obj1:String,obj2:String,isNavObj1:boolean,isNavObj2:boolean,tObj1:String,tObj2:String,minObj1:String,minObj2:String,maxObj1:String,maxObj2:String):void +operation_10():void +getId():String +setId(_id:String):void +getNom():String +setNom(_nom:String):void +getObj1():String +setObj1(_obj1:String):void +getObj2():String +setObj2(_obj2:String):void +isNavObj1():boolean +setNavObj1(_isNavObj1:boolean):void +isNavObj2():boolean +setNavObj2(_isNavObj2:boolean):void +getTipus_obj1():String +setTipus_obj1(_tipus_obj1:String):void +getTipus_obj2():String +setTipus_obj2(_tipus_obj2:String):void +getMin_obj1():String +setMin_obj1(_min_obj1:String):void +getMin_obj2():String +setMin_obj2(_min_obj2:String):void +getMax_obj1():String +setMax_obj1(_max_obj1:String):void +getMax_obj2():String +setMax_obj2(_max_obj2:String):void +getAssociacio():String

3.2.7 AssociacioNN

Aquesta classe representa una associació d'agregació n a n entre dos tipus objecte. En aquest cas, es guarden el nom de l'associació i els identificadors dels dos objectes, doncs caldrà crear una taula de nom el de l'associació, i on les columnes, en aquest cas, seran l'objecte 1 i l'objecte 2. És instanciada per la classe AnalitzadorXMI però, a diferència de les anteriors classes, al mètode throwFile( ), quan es troben les associacions d'aquest tipus en el recorregut del vector d'associacions .

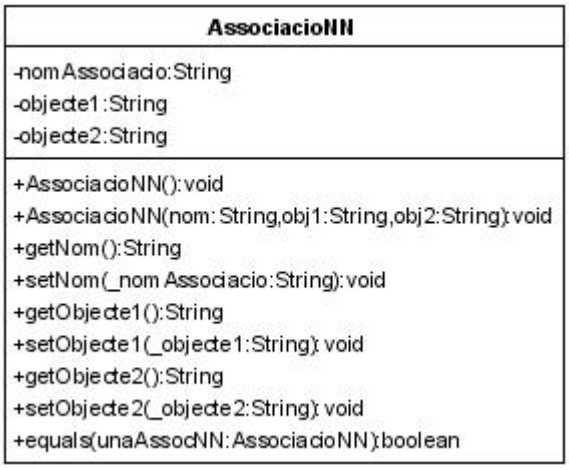
Es composa d'aquests **atributs** :

nomAssociacio : atribut que conté el nom de l'associació n a n.

objecte1 : aquest atribut conté l'identificador del primer tipus objecte de l'associació.

objecte2 : aquest atribut conté l'identificador del segon tipus objecte de l'associació.

Aquesta classe només conté un **mètode** que no sigui constructor, escriptor o lector, es tracta d'**equals**(**AssociacioNN a**), el qual comprova si dos associacions n a n (l'actual instància i la passada per paràmetre) són la mateixa, retornant cert si ambdues tenen el mateix nom, en cas contrari retorna fals.



3.2.8 Utils

Classe abstracta on els seus mètodes són eines bàsiques que permeten alleugerar i fer més clar el codi d'AnalitzarXMI. Es tracta de mètodes estàtics que permeten treballar amb tota la resta de classes. Al ser una classe abstracta no és instanciada si no que els seus mètodes son tots cridats directament des de la classe AnalitzadorXMI o des de la mateixa classe Utils.

Es composa d'aquests **mètodes** :

tractaHerencia(Vector vHerencies, Vector classes, Classe nClasse) :
Aquest mètode fa un recorregut pel vector d'associacions d'herència buscant si una classe determinada és dins d'alguna d'aquestes. En aquest cas marca aquesta classe i l'altre que participa en la relació com pare i com fill, cada rol segons indiqui aquest objecte instància d'Herencia.

esAssocRef(Vector classes, Associacio nAssociacio, Classe nClasse, int objecte) :
Aquest mètode comprova si una classe determinada pertany a una relació de referència determinada. En aquest cas crearia un atribut per a aquest tipus objecte del tipus de l'objecte al que fa referència, amb la clàusula REF davant.

esAssocAgregacioUnivoca(Vector classes, Associacio nAssociacio, Classe nClasse, int objecte) :
Aquest mètode comprova si una classe determinada pertany a una relació d'agregació 1 a 1. En aquest cas crearia un atribut per a aquest tipus objecte del tipus objecte que agrega.

esAssocAgregacioMultiple(Vector classes, Associacio nAssociacio, Classe nClasse, int objecte) :
Aquest mètode comprova si una classe determinada pertany a una relació d'agregació 1 a x, on x >1 i x està definit. En aquest cas crearia un atribut per a aquest tipus objecte d'un VARRAY del tipus objecte que agrega. A més posaria a cert l'atribut isVARRAY d'aquest tipus objecte agregat, això a l'hora de crear l'script faria crear una taula de VARRAY d'aquest tipus objecte.

esAssocAgregacioIndefinida(Vector classes, Associacio nAssociacio, Classe nClasse, int objecte) :
Aquest mètode comprova si una classe determinada pertany a una relació d'agregació 1 a n. En aquest cas crearia un atribut per a aquest tipus objecte d'una taula del tipus objecte que agrega. A més ompliria l'atribut taula d'aquest tipus objecte agregat, això a l'hora de crear l'script faria crear una taula de d'aquest tipus objecte.

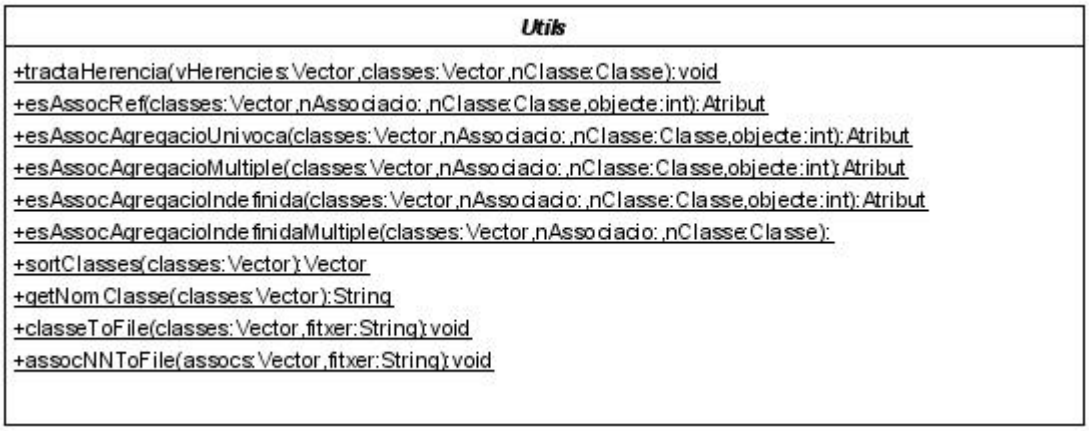
esAssocAgregacioIndefinidaMultiple(Vector classes, Associacio nAssociacio, Classe nClasse) :
Aquest mètode comprova si una classe determinada pertany a una relació d'agregació n a n. En aquest es retorna un objecte AssociacioNN que es guardarà en un vector. Aquest vector es recorre després al mètode Utils.assocNNToFile(), que a partir d'ell construirà les taules de les associacions n a n que calgui.

sortClasses(Vector classes) :
Aquest mètode ordena el vector de tipus objecte abans de crear l'script segons els tipus objecte precedents de cadascú d'ells. Així evitem que es pugui referenciar algun tipus objecte que encara no s'hagi creat, cosa que donaria error a l'hora d'executar l'script. Retorna el mateix vector ordenat.

getNomClasse(Vector classes) :
Aquest mètode retorna el nom del darrer tipus objecte del vector dels tipus objecte. Es cridat des d'AnalitzarXMI per a obtenir el nom de la classe a la que cal assignar-li a un Atribut.

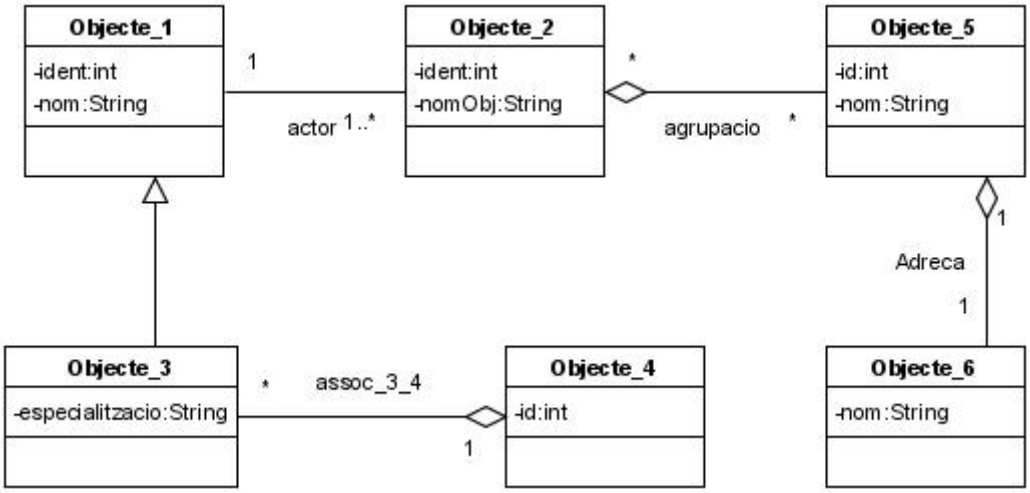
classeToFile(Vector classes, String fitxer) :
Aquest mètode fa una crida al mètode Utils.sortClasses() per a poder ordenar els tipus objectes segons els precedents de cadascú d'ells. Un cop ordenat el vector es fa un recorregut del mateix i per cadascú dels tipus objecte del vector es crida el mètode classeToFile() de cada instància de Classe que s'encarrega de afegir a l'script el tipus objecte en qüestió.

assocNNToFile(Vector assocs, String fitxer) :
Aquest mètode fa un recorregut pel vector assocs, que és un vector d'AssociacioNN. S'executa després del classeToFile, i per cada element que troba ho inclou a l'script en forma de taula amb el nom de l'associació i els tipus objecte que participen en l'associació n a n com a columnes de la mateixa taula.



3.3 ANÀLISI DEL FUNCIONAMENT DE L'APLICACIÓ

Desenvoluparem un exemple de com funciona l'aplicació per a un diagrama de classes senzill :



Aquest diagrama conté sis classes amb les següents característiques :

a) Tots els objectes tenen atributs.
b) Objecte_3 hereta d'Objecte_1.
c) Objecte_2 tindrà un atribut que es dirà actor i que serà del tipus de l'Objecte_1.
d) Hi haurà una taula que representi l'associació n a n de l'Objecte_2 amb l'Objecte_5.
e) Objecte_3 tindrà un atribut que serà una taula del tipus de l'Objecte_4.
f) Objecte_5 tindrà un atribut Adreca que serà del tipus Objecte_6.

Aquest senzill diagrama de classes UML es pot exportar a un document XMI que contindrà unes 3000 línies però on només unes determinades etiquetes ens interessaran.

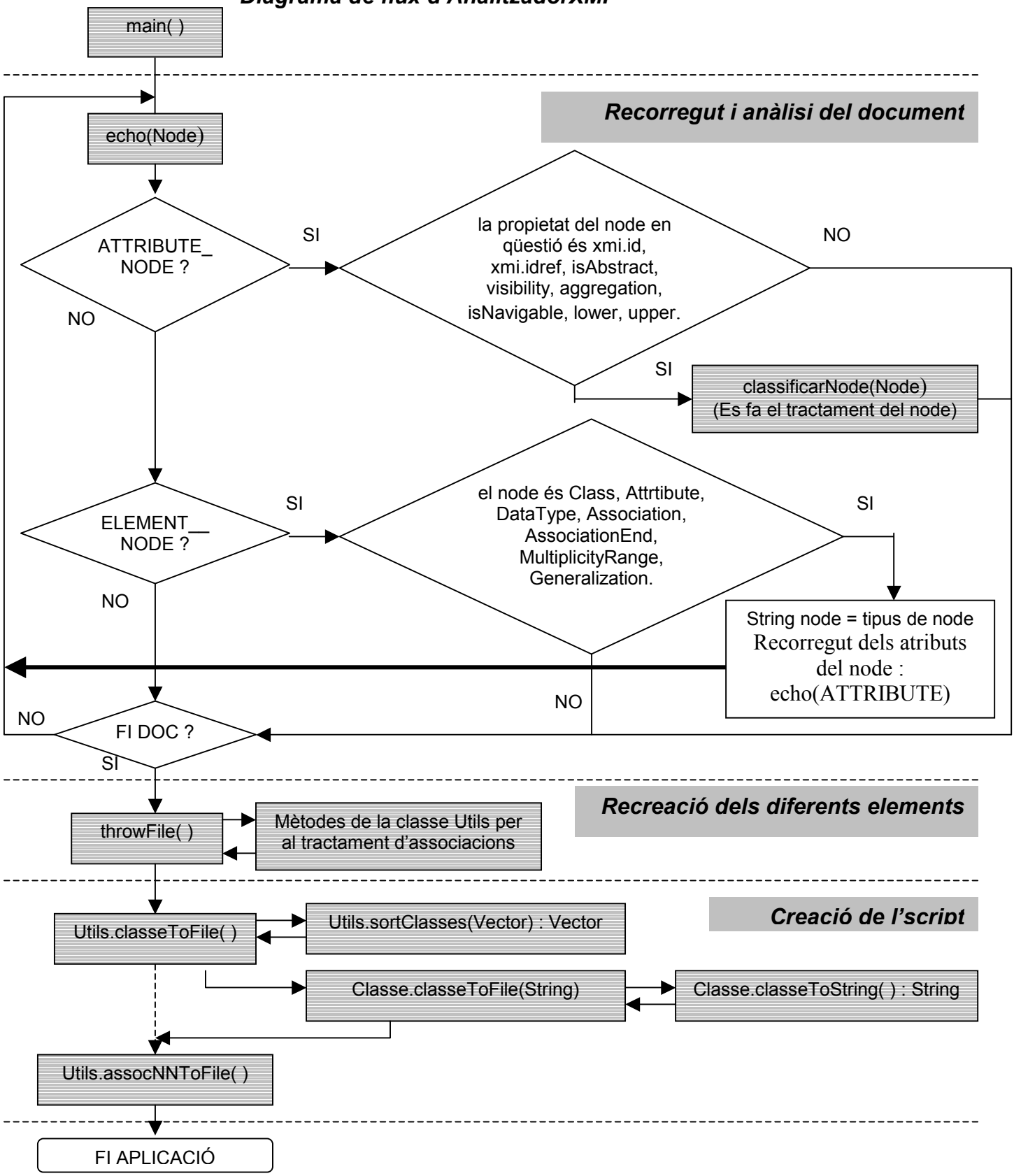
El nostre objectiu és aconseguir plasmar els diferents elements del diagrama de classes amb les seves relacions dins d'un script que ens permeti crear sense cap esforç la mateixa estructura del diagrama per a una base de dades objecte-relacional. Per a obtenir aquest resultat cal fer :

a) **Recorregut i anàlisi** : Anar analitzant les etiquetes del fitxer XMI en un recorregut sencer del mateix i tractar les dades que ens proporcionin algunes d'aquestes etiquetes.

b) **Recreació dels elements** : Construir tots els elements de l'script a partir de les dades recollides.

c) **Creació de l'script** : Passar tots els elements obtinguts a un fitxer sql en l'ordre i l'estructura sintàctica necessaris per a que l'execució del mateix fitxer dins de l'entorn de Oracle 9i SQLPLUS es doni amb èxit.

Diagrama de flux d'AnalitzadorXMI



Per a poder fer un anàlisi exhaustiu de com aquesta aplicació aconsegueix el seu objectiu anirem observant com es succeeixen aquestes tres fases citades sobre l'exemple anterior.

A) RECORREGUT I ANÀLISI DEL FITXER XMI

La classe que engega l'aplicació i és la columna vertebral de la mateixa és **AnalitzadorXMI**; en aquesta classe es fa la transformació del fitxer XMI en un arbre DOM (Document Object Model) que es pot recórrer baixant per l'entramat dels seus nodes fins arribar a les fulles (nodes sense fills). Primerament, al mètode **main** (el que primer s'executa quan s'engega l'aplicació) es crea en memòria aquest arbre DOM i es guarda a la variable doc :

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(new File(filename));
```

Un cop tenim el document en forma d'arbre ja podem recórrer els seus nodes en busca dels que ens interessin. No oblidem que el recorregut es fa agafant un node i arribant fins les fulles d'aquest, aleshores s'agafa el següent node del mateix nivell i també s'aprofundeix fins les seves fulles, i així fins acabar amb tots els nodes que hi hagi. El recorregut es llença mitjançant el mètode **echo(Document)** del mateix **AnalitzadorXMI** :

```
new AnalitzadorXMI().echo(doc);
```

Dins d'aquest mètode s'analitza si l'element actual és un node (ELEMENT_NODE) o una propietat d'un node (ATTRIBUTE_NODE), en cas que no fos ni un ni l'altre aleshores no ens interessa per a res i podem passar a un altre node. Podem considerar que el que nosaltres entenem com a node és el tipus ELEMENT_NODE, i el tipus ATTRIBUTE_NODE només són les propietats d'aquest node. Internament, l'aplicació tracta aquestes propietats com si fossin també nodes que pengen directament d'un node Element, indicant que són del tipus ATTRIBUTE_NODE (atribut de l'element).
Així, per exemple, si agafem un tros de codi del fitxer XMI de l'exemple trobem :

```
<UML:Class xmi.id = 'b1b14:f9ccf03f0a:-7fea' name = 'Objecte_4' visibility = 'public'
  isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'
  isActive = 'false'>
    ....
</UML:Class>
```

Tots els nodes ELEMENT_NODE porten davant la clàusula UML, així, en aquest exemple, el node ELEMENT_NODE seria Class, mentre que els nodes ATTRIBUTE_NODE serien xmi.id, name, visibility, etc..

Un cop sabem si l'element a analitzar és un ELEMENT_NODE o un ATTRIBUTE_NODE hem de comprovar si pertany a algun dels tipus que ens interessen doncs contindran valors imprescindibles per a poder crear un script fidel al diagrama UML . AnalitzadorXMI conté un atribut que es diu node i és de tipus String, el qual cal informar amb un valor o un altre segons el tipus de node ELEMENT_NODE que estiguem tractant, doncs després ens basarem en aquest valor a l'hora de tractar la dada.

Si el node és tipus **ELEMENT_NODE** ens interessa qualsevol d'aquests tipus (indiquem el valor que agafarà l'atribut node):

Attribute : identifica els atributs o propietats dels tipus objecte o classes, per exemple nom a Objecte_1. L'atribut node serà igual a "atribut".

```
<UML:Attribute xmi.id = 'b1b14:f9ccf03f0a:-7eac' name = 'nom' visibility = 'private'
  isSpecification = 'false' ownerScope = 'instance'>
  <UML:StructuralFeature.type>
    <UML:Class xmi.idref = 'b1b14:f9ccf03f0a:-7ff4'/>
  </UML:StructuralFeature.type>
</UML:Attribute>
```

DataType : identifica el tipus al que pertany cada atribut, per exemple l'atribut ident de la classe Objecte_1 és de tipus int. Els tipus de dada que són classes, com String, no els trobarem amb aquesta etiqueta, per això quan es tracti un tipus objecte, a més d'afegir-ho a un vector dels tipus objecte, ho afegirem al vector de tipus de dada, doncs qualsevol objecte pot ser un tipus de dada. L'atribut node serà igual a "tipusDada".

```
<UML:DataType xmi.id = 'b1b14:f9ccf03f0a:-7ff6' name = 'int' isSpecification = 'false'
  isRoot = 'false' isLeaf = 'false' isAbstract = 'false'/>
<UML:DataType xmi.id = 'b1b14:f9ccf03f0a:-7ff1' name = 'void' isSpecification =
'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'/>
<UML:Class xmi.id = 'b1b14:f9ccf03f0a:-7ff4' name = 'String' isSpecification = 'false'
  isRoot = 'false' isLeaf = 'false' isAbstract = 'false' isActive = 'false'/>
```

Generalization : identifica una associació d'herència, contenint un identificador, una classe filla i una classe pare de la que hereta la filla. L'atribut node serà igual a "herencia".

```
<UML:Generalization xmi.id = 'b1b14:f9ccf03f0a:-7e6b' isSpecification = 'false'>
  <UML:Generalization.child>
    <UML:Class xmi.idref = 'b1b14:f9ccf03f0a:-7fed'/>
  </UML:Generalization.child>
  <UML:Generalization.parent>
    <UML:Class xmi.idref = 'b1b14:f9ccf03f0a:-7e6a'/>
  </UML:Generalization.parent>
</UML:Generalization>
```

Association : identifica una associació entre dues classes (ens dona l'identicador i el nom d'aquesta associació) . L'atribut node serà igual a "assoc".

AssociationEnd : defineix les característiques d'una associació entre dues classes (identificadors de les classes participants) . L'atribut node serà igual a "endAssoc".

MultiplicityRange: defineix, també per a una associació entre dues classes, la participació de cadascuna de les parts de l'associació . L'atribut node serà igual a "rangAssoc".

A continuació es mostra un exemple de com està reflectit al fitxer XMI una associació del nostre exemple de diagrama de classes , es pot observar com hi són els nodes Association, AssociationEnd i MultiplicityRange, i les dades que contenen:

```
<UML:Association xmi.id = 'b1b14:f9ccf03f0a:-7e67' name = 'actor' isSpecification
      = 'false'  isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
  <UML:Association.connection>
    <UML:AssociationEnd xmi.id = 'b1b14:f9ccf03f0a:-7e66' name = '' visibility =
      'public' isSpecification = 'false' isNavigable = 'false' ordering = 'unordered'
      aggregation = 'none'   targetScope = 'instance' changeability = 'changeable'>
      <UML:AssociationEnd.multiplicity>
        <UML:Multiplicity xmi.id = 'b1b14:f9ccf03f0a:-7e65'>
          <UML:Multiplicity.range>
            <UML:MultiplicityRange xmi.id = 'b1b14:f9ccf03f0a:-7e64' lower = '1' upper
        = '1'/>
          </UML:Multiplicity.range>
        </UML:Multiplicity>
      </UML:AssociationEnd.multiplicity>
      <UML:AssociationEnd.participant>
        <UML:Class xmi.idref = 'b1b14:f9ccf03f0a:-7e6a'/>
      </UML:AssociationEnd.participant>
    </UML:AssociationEnd>
    <UML:AssociationEnd xmi.id = 'b1b14:f9ccf03f0a:-7e63' visibility = 'public'
      isSpecification = 'false' isNavigable = 'false' ordering = 'unordered' aggregation
        = 'none'   targetScope = 'instance' changeability = 'changeable'>
      <UML:AssociationEnd.multiplicity>
        <UML:Multiplicity xmi.id = 'b1b14:f9ccf03f0a:-7e62'>
          <UML:Multiplicity.range>
            <UML:MultiplicityRange xmi.id = 'b1b14:f9ccf03f0a:-7e61' lower = '1' upper
        = '-1'/>
          </UML:Multiplicity.range>
        </UML:Multiplicity>
      </UML:AssociationEnd.multiplicity>
      <UML:AssociationEnd.participant>
        <UML:Class xmi.idref = 'b1b14:f9ccf03f0a:-7ff0'/>
      </UML:AssociationEnd.participant>
    </UML:AssociationEnd>
  </UML:Association.connection>
</UML:Association>
```

Class : identifica tipus objecte, l'equivalent de les classes de la programació orientada a objectes. En aquest cas poden ser tant els objectes creats al diagrama (Objecte_1, Objecte_2,...,Objecte_6) com per exemple la classe String.
Aquí l'atribut node agafarà un valor o un altre depenent del seu valor anterior. Així, si node abans era igual a "herencia", ara serà igual a "fill" (indicant que la classe que tractem està dins d'una associació d'herència com a filla d'una altra). Si abans era igual a "filla" ara serà igual a "super",

indicant que, si abans hem tractat la classe filla dins l'herència, ara tractarem la classe de la que hereta l'anterior. Si abans era "rangAssoc" voldrà dir que l'actual node Class penja d'un node MultiplicityRange i ens indicarà les participacions mínimes i màximes de la classe que referencia dins l'associació, per lo que ara node serà igual a "objAssoc". Si node era igual a "atribut" aleshores significa que l'actual node Class penja d'un node "Attribute", això vol dir que aquesta classe fa referència a una classe que participa com a tipus de dada d'un atribut i caldrà canviar el valor de node a "classeTipusDada". En qualsevol altre cas, node serà igual a "classe".

```
<UML:Class xmi.id = 'b1b14:f9ccf03f0a:-7ff0' name = 'Objecte_2' visibility = 'public'
  isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'
  isActive = 'false'>
  <UML:Classifier.feature>
    <UML:Attribute xmi.id = 'b1b14:f9ccf03f0a:-7fef' name = 'ident' visibility = 'private'
      isSpecification = 'false' ownerScope = 'instance'>
      <UML:StructuralFeature.type>
        <UML:DataType xmi.idref = 'b1b14:f9ccf03f0a:-7ff6'/>
      </UML:StructuralFeature.type>
    </UML:Attribute>
    <UML:Attribute xmi.id = 'b1b14:f9ccf03f0a:-7fee' name = 'nomObj' visibility = 'private'
      isSpecification = 'false' ownerScope = 'instance'>
      <UML:StructuralFeature.type>
        <UML:Class xmi.idref = 'b1b14:f9ccf03f0a:-7ff4'/>
      </UML:StructuralFeature.type>
    </UML:Attribute>
  </UML:Classifier.feature>
</UML:Class>
```

Observem que quan entrem a tractar un ELEMENT_NODE, el que fem és donar un valor a la variable node. Després caldrà fer un recorregut pels atributs d'aquest element cridant de forma recurrent el mètode echo per cadascun d'aquests atributs. És aquí quan el node passat per paràmetre al mètode echo és de tipus ATTRIBUTE_NODE.

Si el node és tipus **ATTRIBUTE_NODE** ens interessa qualsevol d'aquestes propietats :

xmi.id : és l'identificador de l'element.

xmi.idref : defineix una referència a un element amb un xmi.id igual al valor d'aquesta propietat.

name : nom de l'element

isAbstract : indica si l'element és abstracte o no.

visibility : indica la visibilitat de l'element (privat o públic).

aggregation : indica per a una associació si és d'agregació.

isNavigable : indica el sentit d'una associació.

lower : indica el número mínim de participacions d'un element en una associació.

upper : indica el número màxim de participacions d'un element en una associació.

Si l'ATTRIBUTE_NODE és de qualsevol d'aquests tipus, el següent pas és cridar el mètode **classificarNode(Node)** ; aquesta selecció ens permet obviar totes aquelles propietats que no ens interessen.
En classificarNode() es fa l'anàlisi o tractament d'un node tipus ATTRIBUTE _NODE n passat com a paràmetre. Segons el valor de l'atribut String node, n serà una propietat d'un element o d'un altre. Aquestes seran les diferents opcions possibles :

a) Si node = "classe", anirem carregant l'identificador, el nom, si és abstracte i la visibilitat (privat o públic).
Si ja hem carregat les tres primeres (totes excepte visibilitat) aleshores podem carregar la classe al vector de tipus de dada, doncs totes les classes poden ser utilitzades com atribut en qualsevol altra classe del diagrama. Així, al nostre exemple, si haguéssim posat per a l'Objecte_4 un atribut del tipus de l'Objecte_6, ens caldria afegir aquesta classe com a tipus de dada , tal com també afegim la classe String.
Quan hàgim carregat les quatre propietats aleshores podrem crear una instància de la classe Classe i afegir-la al vector de classes. Fem aquesta distinció entre carregar la propietat de visibilitat o no doncs les classes que no hàgim creat al diagrama de classes aleshores no tindran la propietat de visibilitat. Aquestes classes només intervenen com a tipus de dades d'atributs (tal com passa amb String, una classe que nosaltres no creem si no que la importem de la llibreria java.lang) per tant no les hem de crear al nostre model. A més d'afegir les classes que nosaltres creem al vector de classes, també canviarem els seus noms al vector de tipus de dades (on fa un moment les hem afegit) afegint-ne "_t", doncs aquest serà el nom del tipus objecte que es crearà a l'script.
Un cop hem fet aquest pas hem de posar tots els valors auxiliars, que ens han servit per a carregar la classe, a nul, doncs si no mantindrien el seu valor per al següent node d'aquest tipus provocant un funcionament erroni de l'aplicació.

b) Si node = "atribut", primer, i si no existeix ja, crearem una instància de la classe Atribut i li afegirem el nom de la classe a la que pertany l'atribut. Aquest nom el traiem del darrer element del vector de classes (sabem que l'atribut pertany a aquesta classe gràcies al recorregut en forma d'arbre que estem fent, així l'ELEMENT_NODE Attribute és un nivell per sota d'una altra ELEMENT_NODE Class que haurem introduït just abans). Un cop tenim el nom de la classe obtenim l'identificador i la visibilitat. Quan tenim totes aquestes propietats informades aleshores podem introduir l'atribut al vector d'atributs. Encara ens quedarà per saber a quin tipus de dada pertany aquest atribut (si és int, String, boolean,..), per a poder inserir-ne aquesta propietat caldrà baixar encara un nivell més a l'arbre del document trobant-ho dins una etiqueta DataType o Class, tal com veurem ara a continuació.

c) Si node = "classeTipusDada", ens trobem amb el tipus de dada d'un atribut. Aquest tipus de dada fa referència a una classe (el tipus de dada String és l'exemple més clar) i només conté una propietat que ens interessi, xmi.idref, la qual conté el valor de l'identificador de la classe en qüestió. Un cop obtenim aquesta propietat la inserim al darrer atribut del vector d'atributs, doncs sabem que li pertany gràcies al recorregut que estem fent del document.

d) Si node = "tipusDada" pot passar que contingui una referència al tipus de dada (aleshores només trobaríem la propietat xmi.idref) o que contingui la descripció sencera del tipus de dada (amb un nom, i un identificador). En el primer cas actuarem igual que si node = "classeTipusDada", guardant al darrer element del vector d'atributs el valor del tipus de dada com una referència a l'identificador d'aquest tipus. Si ens trobéssim en el segon cas aleshores cal crear una instància de la classe TipusDada, amb un nom i un identificador, i guardar aquest objecte al vector de tipus de dades (recordem que aquí també guardàvem representacions de les classes que també intervenien com a tipus de dades dels diferents atributs).

e) Si node = "assoc" aleshores estem tractant una propietat que pertany a un node que defineix una associació. Aquest tipus de node només conté un identificador i un nom de l'associació. Si la propietat és xmi.id aleshores cal crear un objecte Associació, inserir-li aquest identificador i guardar-ho en un vector d'associacions. Si la propietat és name aleshores inserim el nom de l'associació al darrer objecte del vector d'associacions, doncs sabem que aquest objecte és al que li pertany aquesta propietat gràcies al recorregut que s'està fent del document.

f) Si node = "endAssoc", sabem que el node al que pertany la propietat a guardar és un nivell per sota d'un node Association. Per tant, cal recuperar el darrer element del vector d'associacions. En aquest tipus de node podem trobar més propietats de l'associació. Abans hem recuperat l'identificador i el nom. Aquí obtindrem el tipus d'associació (si és una associació d'agregació o no), si és navegable (és a dir, si l'associació es dirigeix cap algun sentit dels dos possibles, si no es dirigeix cap a cap sentit aleshores estem parlant d'una associació de referència).

g) Si node = "rangAssoc" obtenim les participacions mínimes i màximes de cadascú dels objectes de l'associació (dos objectes doncs l'associació ha de ser binària), guardant per al darrer element del vector d'associacions, com hem fet al punt anterior, els valors setMinObj_1 i setMinObj_2 si la propietat és "lower", o setMaxObj_1 i setMaxObj_2 si la propietat és "upper". Aquestes propietats són molt importants per a la posterior construcció correcta del fitxer script.

h) Si node = "objAssoc" obtenim els identificadors dels dos tipus objecte associats, és a dir, per a un node ELEMENT_NODE Class que penja a un nivell inferior d'un altre ELEMENT_NODE Association, recollim la propietat xmi.idref que fa referència a una classe. Tindrem dos elements Class que ens reportaran dos identificadors, els quals introduirem al darrer element del vector d'associacions, com hem fet als punts anteriors e, f i g.

i) Si node = "herencia" aleshores estem tractant una propietat que pertany a un node que defineix una associació d'herència simple (entre dos objectes). Aquest tipus de node només conté un identificador de l'associació. Caldrà crear un objecte Herencia, inserir-li aquest identificador i guardar-ho en un vector d'herències.

j) Si node = "super" aleshores només podem tenir la propietat de xmi.idref que fa referència a una classe la qual és de la que hereta una altra. Cal recuperar el darrer element del vector d'herències i inserir a l'atribut super l'identificatiu d'aquesta classe.

k) Si node = "fill" estarem obtenint el valor de la propietat xmi.idref que fa referència a la classe que hereta d'una altra. Guardarem aquest valor també a l'atribut fill del darrer objecte Herencia del vector d'herències.

Aquest serà el tractament sencer de les propietats durant el recorregut complet de l'arbre que representa el fitxer XMI. Cada cop que un ELEMENT_NODE ha estat analitzat (s'han recuperat els seus atributs o propietats) es comprova si té fills, en cas afirmatiu s'aniria baixant per aquesta branca fins arribar a les fulles, i en cas contrari doncs es pujaria al nivell superior per a tornar a baixar pel següent node de l'arbre.
Un cop s'ha recorregut tot l'arbre sencer, aleshores ja podem passar a la següent fase : la reconstrucció de tots els elements que hem obtingut durant aquest recorregut.

B) RECONSTRUCCIÓ DELS DIFERENTS ELEMENTS

Un cop el mètode echo(Node) ha fet el recorregut sencer del document, aleshores retorna el control al mètode main de la classe AnalitzarXMI, el qual ho havia cridat. El següent mètode que cridarà el mètode principal serà **throwFile()** el qual s'encarrega de gestionar tota aquesta segona fase de reconstrucció o reagrupament de les dades obtingudes a partir del recorregut fet a la primera fase de l'aplicació.
Aquest mètode s'encarrega d'afegir els atributs als tipus objecte respectius i crear les estructures necessàries per a reproduir fidelment tots els elements representats al diagrama de classes del qual partim.
La majoria dels processos no els executa ell si no que delega en la classe abstracta **Utils**, la qual només ens proporciona eines en forma de mètodes estàtics que permeten alleugerar el codi de la classe AnalitzarXMI, doncs si no aquesta classe a la pràctica no seria manejable. Gràcies a la classe Utils tenim un codi ordenat i net, el que es tradueix en facilitats a l'hora de comprendre els passos que es fan.

El mètode throwFile() parteix del recorregut del vector de classes que s'ha anat omplint durant la fase anterior. Aleshores, per a cada classe fa aquest recorregut que podem dividir en 4 passos :

a) **comprovació d'herència** :
es tracta d'analitzar si la classe actual participa en una associació d'herència. Per a poder fer aquest anàlisi es fa una crida del mètode **Utils.tractaHerencia()** al que se li passa per paràmetre el vector de classes, el d'herències i la classe que estem analitzant. Es fa un recorregut del vector d'herències cercant, primer, si l'atribut super de l'element actual del vector d'herències és igual a l'identificador de la classe actual; si no és així aleshores comprova el mateix amb l'atribut fill. Si fos igual a super caldria marcar a la classe actual l'atribut esSuper com a cert. Si fos igual a fill, aleshores caldria trobar el nom de la classe de la que hereta (mitjançant un altre recorregut de les classes en el que es compararia el valor del camp super de l'element Herencia amb l'identificador de cada classe) i aquest, un cop trobat, es guardaria a l'atribut pare de la classe actual. Per ho tant, si trobem que una classe hereta d'una altra, o d'ella hereta una altra, aleshores cal informar els camps esSuper o pare d'aquesta classe. Amb aquesta informació ja tenim prou per a representar les diferents associacions d'herència que es puguin donar al nostre diagrama de classes.
Al nostre exemple l'aplicació hauria de detectar al vector d'herències un element, el qual contindria a l'atribut super l'identificador que fa referència a l'identificador de la classe Objecte_1 (per a aquest element marcaria el camp esSuper com cert), mentre que a l'atribut fill trobaria l'identificador de la classe Objecte_3, a la qual l'afegirà al seu atribut pare el valor del nom de la classe de la que hereta, Objecte_1.

b) **comprovació d'associació** :
es tracta d'analitzar si la classe actual participa en qualsevol altre tipus d'associació que no sigui pas la d'herència, la qual hem tractat al punt anterior. En aquest punt cal diferenciar entre cinc tipus diferents d'associacions :

1) associacions de referència
2) associacions d'agregació 1 a 1
3) associacions d'agregació 1 a x, on x > 1, x està definit.
4) Associacions d'agregació 1 a n.
5) Associacions d'agregació x a x, on x > 1.

Cal fer un recorregut de tot el vector d'associacions i per cada tipus d'associació comprovar si la classe actual participa, podent no participar en cap, participar en un tipus o en tots.

1) Per a comprovar el cas 1 (**associacions de referència**) es crida al mètode **Utils.esAssocRef()**, el qual retorna un atribut en cas de que la classe actual participi en una associació d'aquest tipus, o retornarà nul en cas contrari. Aquest mètode comprova si la classe participa en l'actual associació, si és així aleshores comprova si l'associació és de referència mitjançant comparar si ambdós atributs, tipus_obj1 i tipus_obj2, són iguals a "none" (no tenen cap d'ells el valor "aggregate" d'una associació d'agregació). Si és així comprova que el màxim de participacions de la classe actual siguin n dins l'associació. Si també aquesta resposta és afirmativa aleshores recupera els valors de l'altra classe participant i crea un objecte Atribut que pertanyerà a la classe actual , indicant també que el tipus de dada té de nom el nom de l'associació i fa referència a l'altra classe participant (amb la clàusula REF); a més, també caldrà afegir al vector de classes precedents de la classe actual a la classe referenciada, doncs, si no, podem tenir problemes a l'hora d'executar l'script. La comprovació cal fer-la buscant primer si la classe actual és el primer participant de l'associació , i després buscant si és el segon participant, per tal que no es pugui passar per alt aquesta associació. Si el mètode retorna un atribut no nul aleshores caldrà afegir-ho al vector d'atributs.
Al nostre exemple, aquesta associació és la que es dona entre les classes Objecte_1 i Objecte_2. Observem que és una associació sense cap sentit de navegació i on un dels participants, l'Objecte_1, té un número màxim i mínim de participacions igual a 1, mentre que l'Objecte_2 pot participar d'1 a n vegades; aquesta classe és la que haurà de tenir un atribut de nom "actor" (nom de l'associació) fent referència al tipus objecte Objecte_1 amb la clàusula "_t" :

actor REF Objecte_1_t

aquesta clàusula representarà una associació de referència en una base de dades Objecte-relacional.

2) Per a comprovar si existeix una associació d'**agregació 1 a 1** es crida al mètode **Utils.esAssocAgregacioUnivoca()**, que també retornarà un atribut si troba una associació d'aquest tipus per a la classe actual i l'associació actual o nul es cas contrari. Aquest mètode comprova si la classe participa en l'actual associació, si és així aleshores comprova si l'associació és d'agregació verificant que un dels dos objectes participants és del tipus "aggregate", en aquest cas comprova si el màxim de participacions de l'altre objecte és 1 (no ens interessen les participacions de la classe actual); si tot això es compleix aleshores es recupera els valors de l'altra classe participant i es crea un objecte Atribut que pertanyerà a la classe actual , de nom el de la classe agregada, i el seu tipus de nom també el de la classe agregada però amb la clàusula "_t", que serà el nom d'aquest tipus objecte a la base de dades; a més, també caldrà afegir la classe agregada al vector de classes precedents de la classe actual, doncs si no podem tenir problemes a l'hora d'executar l'script. La comprovació cal fer-la buscant primer si la classe actual és el primer participant de l'associació , i després buscant si és el segon participant, per tal que no es pugui passar per alt aquesta associació. Si el mètode retorna un atribut no nul aleshores caldrà afegir-ho al vector d'atributs.
Al nostre exemple, aquesta associació és la que es dona entre les classes Objecte_5 i Objecte_6; dins del fitxer xmi, la propietat aggregation='aggregate" s'aplica sobre el participant Objecte_5, indicant-ne que és l'element al que s'agrega l'altre element. Així, caldrà crear un atribut per a Objecte_5 així :

Objecte_6 Objecte_6_t

, és a dir, es crea un atribut de tipus Objecte_6_t per al tipus objecte Objecte_5.

3) Per a comprovar si existeix una associació d'**agregació 1 a x (on x és més gran d'1 i és un número definit)** es crida al mètode **Utils.esAssocAgregacioMultiple()**, que també retornarà un atribut si troba una associació d'aquest tipus per a la classe actual i l'associació actual o nul es cas contrari. Aquest mètode comprova si la classe participa en l'actual associació, si és així aleshores comprova si l'associació és d'agregació verificant que un dels dos objectes participants és del tipus "aggregate", en aquest cas comprova si el màxim de participacions de l'altre objecte és un número més gran d'1 (no ens interessen les participacions de la classe actual); si tot això es compleix aleshores es recupera els valors de l'altra classe participant i es crea un objecte Atribut que pertanyerà a la classe actual , de nom el de la classe agregada, i el seu tipus de nom també el de la classe agregada però amb la clàusula "_v", que serà el nom del varray (array variable) que caldrà crear a la base de dades a partir de la classe agregada (caldrà inserir en l'atribut isVARRAY d'aquesta classe el valor del màxim de

participacions que té dins l'associació); a més, també caldrà afegir la classe agregada al vector de classes precedents de la classe actual, doncs si no podem tenir problemes a l'hora d'executar l'script. La comprovació cal fer-la buscant primer si la classe actual és el primer participant de l'associació , i després buscant si és el segon participant, per tal que no es pugui passar per alt aquesta associació. Si el mètode retorna un atribut no nul aleshores caldrà afegir-ho al vector d'atributs.
A l'exemple no tenim cap associació d'agregació d'aquest tipus. En principi l'entorn de l'aplicació per a la construcció de diagrames UML no permet l'opció d'indicar un valor limitat de participacions diferent d'1 o zero, caldria manipular el fitxer XMI directament i allà indicar per a aquesta associació i el participant en qüestió que la seva propietat upper és igual a un valor limitat (5,10,...). Si es fa això, quan tornem a carregar el diagrama de classes d'aquest fitxer XMI, ens mostrarà el nou valor de participacions màximes. Si en lloc de fer servir Poseidon for UML es fa servir una eina CASE com VisualUML aleshores podem obtenir aquests valors limitats majors que 1.

4) Per a comprovar si existeix una associació d'**agregació 1 a n (n és un número il·limitat)** el mètode throwFile farà una crida al mètode **Utils.esAssocAgregacioMultiple()**, que també retornarà un atribut si troba una associació d'aquest tipus per a la classe actual i l'associació actual o nul es cas contrari. Aquest mètode comprova si la classe participa en l'actual associació, si és així aleshores comprova si l'associació és d'agregació verificant que un dels dos objectes participants és del tipus "aggregate", en aquest cas comprova si el màxim de participacions de l'altre objecte és igual a –1 i el de la classe actual és igual a 1; si tot això es compleix aleshores es recupera els valors de l'altra classe participant i es crea un objecte Atribut que pertanyerà a la classe actual , de nom el de la classe agregada, i el seu tipus de nom també el de la classe agregada però amb la clàusula "_nt", que serà el nom de la taula de tipus objectes que caldrà crear a la base de dades a partir de la classe agregada (caldrà inserir en l'atribut taula d'aquesta classe el nom de la mateixa classe agregada); a més, també caldrà afegir la classe agregada al vector de classes precedents de la classe actual, doncs si no podem tenir problemes a l'hora d'executar l'script. La comprovació cal fer-la buscant primer si la classe actual és el primer participant de l'associació , i després buscant si és el segon participant, per tal que no es pugui passar per alt aquesta associació. Si el mètode retorna un atribut no nul aleshores caldrà afegir-ho al vector d'atributs.
A l'exemple aquesta associació és la que es dona entre les classes Objecte_3 i Objecte_4. En aquest cas, caldrà crear una taula de tipus objecte de l'Objecte_3 (amb la clàusula final "_nt, que és el nostre estàndard per a la creació de taules) i l'Objecte_4 tindrà un atribut de nom Objecte_3 i de tipus aquesta taula :

EXECUTE IMMEDIATE 'CREATE OR REPLACE TYPE Objecte_3_nt AS TABLE OF Objecte_3_t ' ;

```
EXECUTE IMMEDIATE 'CREATE OR REPLACE TYPE Objecte_4_t
AS OBJECT (
                id NUMBER(10) ,
                Objecte_3 Objecte_3_nt
        ) ' ;
```

5) Per a comprovar si existeix una associació d'**agregació n a n (on n és un número il·limitat)** es crida al mètode de la classe **Utils esAssocAgregacioIndefinidaMultiple()**, que, en aquest cas, retornarà un objecte AssociacioNN si troba una associació d'aquest tipus per a la classe actual i l'associació actual o nul es cas contrari. Aquest mètode comprova si la classe participa en l'actual associació, si és així aleshores comprova si l'associació és d'agregació verificant que un dels dos objectes participants és del tipus "aggregate", en aquest cas comprova si el màxim de participacions de l'altre objecte és igual a –1 i el de la classe actual és igual a 1; si tot això es compleix aleshores es recupera els valors de l'altra classe participant i es crea un objecte AssociacioNN de nom el nom de l'associació, i amb els valors dels noms dels dos tipus objecte o classes que participen en l'associacio. La comprovació cal fer-la buscant primer si la classe actual és el primer participant de l'associació , i després buscant si és el segon participant, per tal que no es pugui passar per alt aquesta associació. Si el mètode retorna una AssociacioNN no nul aleshores caldrà afegir-la al vector d'associacions n a n.
   Al nostre exemple, aquesta associació queda reflectida en l'associació *agrupació* entre les classes Objecte_2 i Objecte_5. Per a aquest cas caldrà crear una taula d'objectes amb el nom de l'associació i de camps els dos objectes :

```
EXECUTE IMMEDIATE 'CREATE TABLE agrupacio_nt(
                        Objecte_2 Objecte_2_t
                        Objecte_5 Objecte_5_t
                        )';
```

Segurament han quedat possibles casos per implementar per causa de les limitacions que posa el temps per a arribar a totes les possibilitats, però sembla fàcil poder implementar més possibilitats en el moment que aquestes siguin necessàries.

c) **Canvi de nom de tipus de dada als atributs** :
Existeixen tipus de dada que ens caldrà canviar el seu nom a l'hora de fer l'script, així, el tipus String en entorn de base de dades es sustitueix per VARCHAR2, i el tipus int per NUMBER. Per a poder fer aquest canvi caldrà fer un recorregut del vector d'atributs i per a cada atribut es farà un nou recorregut del vector de tipus de dades; quan l'identificador del tipus de l'atribut coincideixi amb el del tipus de dada significarà que són el mateix tipus (recordem que a l'atribut no guardàvem el nom del tipus de dada que eren classe si no el seu identificador); en aquest moment, si el tipus de dada té un nom que volem canviar a l'atribut només caldrà

comparar i si es compleix que el tipus de dada en qüestió és el que volem canviar aleshores només cal que canviem a l'atribut el valor de la variable tipus. Cal dir que en cap lloc del diagrama de classes o del fitxer XMI es pot limitar el número de literals que composen un atribut (és a dir, no es pot indicar que String estigui limitat a 100 caràcters, per exemple), per això s'ha triat per indicar un valor determinat.
Aquesta funció es fa al mateix mètode throwFile() un cop s'ha enllestit el processament de les associacions.

d) **assignació d'atributs a la classe actual** :
com a darrer punt dins d'aquesta fase només quedarà afegir els atributs a les classes. Dins del marc de l'anterior recorregut del vector d'atributs, no hem sortit d'ell. Es compara si l'actual atribut pertany a l'actual element del vector classes (no oblidem que encara estem dins el recorregut del vector de classes), si fos així aleshores s'afegiria l'actual atribut al vector d'atributs de la classe actual.

Aquest és el procés sencer del mètode throwFile() , en el qual es basa l'estructura de la fase de reconstrucció dels diferents elements del diagrama de classes.

C) CREACIÓ DE L'SCRIPT

Un cop ja tenim tots els elements ben definits ja podem passar a construir el fitxer SQL que es podrà executar des d'un entorn de base de dades objecte-relacional, en aquest cas Oracle 9i.

Aquesta fase comença un cop el mètode throwFile() retorna el control al mètode main de la classe AnalitzadorXMI, el qual seguidament fa una crida a als dos mètodes que componen aquesta fase **:**

Utils.classeToFile(Vector classes, String nomFitxer)
Utils.assocNNToFile(Vector vAssociacionsNN, String nomFitxer).

Utils.classeToFile té dos funcions :

1) ordenar el vector de classes segons l'ordre de precedències. Recordem que això ens permet crear un script vàlid on determinats objectes no facin referència a objectes que encara no s'han creat; aquesta operació l'executa cridant al mètode **sortClasses(Vector classes)** de la mateixa classe **Utils**. SortClasses fa diversos recorreguts del vector classes recolzant-se en un vector auxiliar en el qual va deixant les classes que troba en un primer recorregut sense cap precedència eliminant-les del vector classes (les precedències les obté del mètode **getPrecedents( )** de cada classe, que retorna un vector amb les classes que aquesta té precedents, és a dir, que s'han de crear abans que ella; si aquest vector està buit aleshores la classe no depèn de cap altra); un cop té les classes que no depenen de cap altre fora del vector classes torna a fer un altre recorregut d'aquest vector eliminant ara aquelles classes on totes les precedents estiguin al vector auxiliar (si és així, a més, afegeix aquesta classe al mateix vector auxiliar). Els recorreguts del vector classe seguiran fins que no hi hagi cap classe al mateix. Quan totes les classes estiguin al vector auxiliar es retorna aquest vector al mètode Utils.classeToFile( ).

2) Un cop tenim el vector de classes ordenat, aleshores comença el recorregut del mateix, on per a cada classe es crida el seu mètode **Classe.classeToFile(String fitxer).** Aquest mètode de la classe instanciada crida un altre mètode de la mateixa classe, **classeToString()** per a que li retorni l'objecte Classe actual adaptat a un guió de creació d'un tipus objecte d'una base de dades objecte-relacional amb totes les propietats de la classe necessàries per a la construcció d'aquest tipus objecte; classeToString retorna un String el qual Classe.classeToFile( ) ho inserirà al fitxer de nom indicat com a paràmetre d'aquest mètode. Cal dir que aquest fitxer ja ha estat creat abans al constructor de la classe AnalitzadorXMI, on s'han afegit una línia de comentari i una altra de inici de procediment anònim pl/sql, és a dir, la clàusula BEGIN.

En tota aquesta operació el més interessant és observar que classeToString() agafa totes les dades de la classe pròpia i, segons el valor d'aquestes, li dona un nom a un tipus objecte, uns atributs d'un tipus determinat, i, fins i tot, indica que aquell tipus objecte hereta d'un altre objecte (clàusula UNDER), que ell mateix no és un objecte final doncs d'ell poden heretar uns altres (NOT FINAL), que potser sigui abstracte (NOT

INSTANTIABLE), o fer la creació seguidament d'un VARRAY o una TAULA del tipus objecte creat just abans, depenent dels atributs isVARRAY i taula de la classe. És important indicar que per a totes les sentències de creació d'objectes caldrà posar la clàusula

EXECUTE IMMEDIATE ' sentència de creació ' ;

Això és important doncs sense aquesta mesura Oracle ens donaria un error pl-00103, ja que no es permet executar operacions ddl des de dins d'un procediment pl/sql o des d'un fitxer d'script.

Utils.assocNNToFile és el mètode encarregat d'adaptar les associacions d'agregació n a n a un entorn de bases de dades Objecte-relacional. Aquestes associacions tenen la particularitat de que cal crear una taula d'objectes amb dues columnes, una per cada tipus d'objecte participant. Per ho tant, el que fa aquest mètode és un recorregut del vector d'associacions n a n, i per a cadascuna d'elles genera un guió de creació d'una taula d'aquest tipus, on el nom de la taula serà el nom de la pròpia associació (agregant "_nt" per a indicar que es tracta d'un objecte taula), els noms dels atributs seran els noms dels objectes participants i el tipus d'aquests seran els mateixos noms agregant la clàusula "_t" doncs és la que indica al nostre estàndard que estem parlant d'un tipus objecte. Cada taula s'afegeix al fitxer de l'script que es va crear al mètode classeToFile(). Tant si s'ha afegit com si no alguna taula d'associacions n a n aquest mètode afegirà les següents clàusules al final del fitxer script :

```
END;
\
show errors;
```

Aquestes línies permeten tancar el procediment, la seva execució des de línia de comandes de l''Oracle SQLPLUS i mostrar els possibles errors d'execució de l'script.

Un cop hem creat el fitxer script amb les sentències de creació de tots els tipus objectes (amb les possibles varrays i taules de tipus objectes que calgui crear) i totes les taules de les associacions d'agregació n a n l'aplicació conclou mostrant per línia de comandes el missatge de "El fitxer script.sql ha estat creat en aquest directori." .

Aplicant tota aquesta teoria al nostre exemple, el fitxer d'script quedaria així un cop s'ha executat l'aplicació :

```
-- script creat a partir de fitxer xml

BEGIN

EXECUTE IMMEDIATE 'CREATE OR REPLACE TYPE Objecte_6_t AS
OBJECT (
nom VARCHAR2(200)
)  ';
```

```
EXECUTE IMMEDIATE 'CREATE OR REPLACE TYPE Objecte_5_t AS OBJECT (
id NUMBER(10) ,
nom VARCHAR2(200) ,
Objecte_6 Objecte_6_t
)  ';


EXECUTE IMMEDIATE 'CREATE OR REPLACE TYPE Objecte_1_t AS OBJECT (
ident NUMBER(10) ,
nom VARCHAR2(200)
)  NOT FINAL ';


EXECUTE IMMEDIATE 'CREATE OR REPLACE TYPE Objecte_2_t AS OBJECT (
ident NUMBER(10) ,
nomObj VARCHAR2(200) ,
actor REF Objecte_1_t
)  ';


EXECUTE IMMEDIATE 'CREATE OR REPLACE TYPE Objecte_3_t UNDER Objecte_1_t (
especialitzacio VARCHAR2(200)
)  ';

EXECUTE IMMEDIATE 'CREATE OR REPLACE TYPE Objecte_3_nt AS TABLE OF Objecte_3_t ';


EXECUTE IMMEDIATE 'CREATE OR REPLACE TYPE Objecte_4_t AS OBJECT (
id NUMBER(10) ,
Objecte_3 Objecte_3_nt
)  ';

EXECUTE IMMEDIATE 'CREATE TABLE agrupacio_nt(
Objecte_2 Objecte_2_t,
Objecte_5 Objecte_5_t
)';
END;
/
show errors;
```

3.4 COM EXECUTAR L'APLICACIÓ

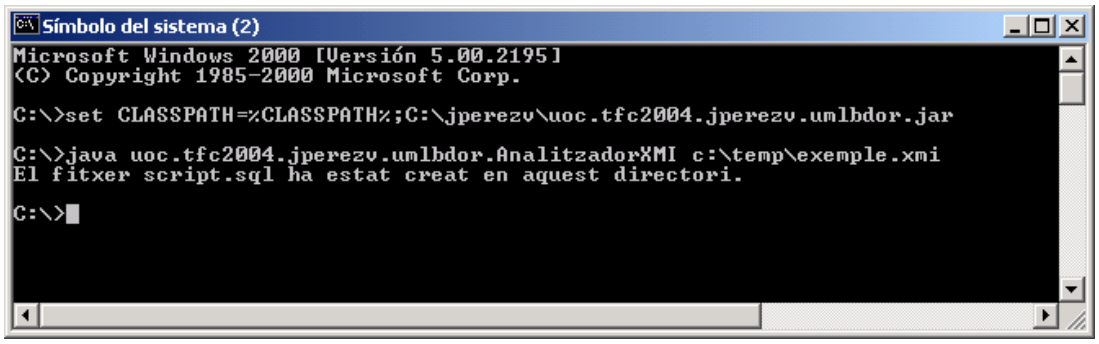
A continuació indicarem quin és el mètode que cal seguir per a la correcta execució de l'aplicació. L'aplicació està guardada en una llibreria amb el nom uoc.tfc2004.jperezv.umlbdor.jar; cal posar aquesta llibreria en un directori del disc dur i afegir-la al classpath; per exemple :

- desem el fitxer uoc.tfc2004.jperezv.umlbdor.jar al directori c:\temp;
- obrim una pantalla de MS-DOS
- executem la línia següent :

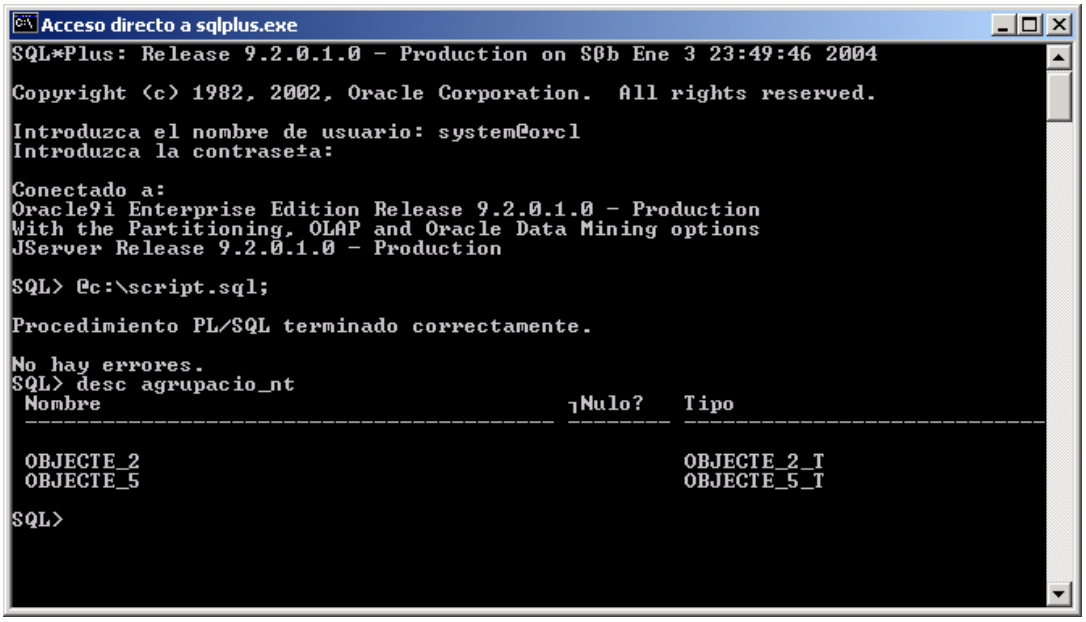
C:\>set CLASSPATH=%CLASSPATH%;C:\temp\uoc.tfc2004.jperezv.umlbdor.jar

- Per a executar l'aplicació ens cal anar al directori on es trobi el fitxer XMI que volem passar a l'script; en aquest cas l'execució seria així:

C:\>java uoc.tfc2004.jperezv.umlbdor.AnalitzadorXMI ruta\nomFitxerXMI.xmi



- Això dona com a resultat un missatge el qual pot ser d'èxit ("El fitxer script.sql ha estat creat en aquest directori.") o un missatge d'error. En cas que aparegui el missatge d'èxit trobarem el fitxer amb nom script.sql al mateix directori en el que hem executat l'aplicació.
- script.sql es pot executar des d'Oracle 9i SQLPLUS, sempre que l'usuari que accedeixi tingui permisos de creació d'objectes a la base de dades.



3.5 Línies futures de treball.

Tal com es va comentar anteriorment, aquesta aplicació ve a omplir un buit en el tractament dels diagrames UML. Fins ara havien aparegut eines CASE que permetien a partir d'un diagrama de classes obtenir l'esquema equivalent per a una base de dades relacional, però, fins ara, no existia cap eina que permetés aquesta transformació aplicada a bases de dades objecte-relacionals o orientades a objectes. Aquest treball ha de servir com a punt de partida per a nous treballs que permetin estendre el que aquí s'ha realitzat; els següents punts indiquen quines possibilitats hi ha per a millorar el treball aquí fet :

- Creació de restriccions a partir dels elements i les relacions del diagrama de classes.
- Afegir més elements en l'anàlisi del diagrama de classes fins arribar a poder plasmar qualsevol element del mateix dins l'script.
- Afegir funcions als tipus objecte de la base de dades a partir dels possibles mètodes que es puguin mostrar per a les classes del diagrama UML.
- Ampliar l'abast de l'aplicació a qualsevol eina CASE amb capacitat de generar fitxers XMI, és a dir, salvar l'obstacle que representa la generació de fitxers XMI amb diferent vocabulari de marques.

A banda d'aquestes noves funcionalitats, es podria millorar en lo que ja existeix un parell de detalls importants i que no s'han implementat en aquesta ocasió doncs no eren funcionalitats imprescindibles i el temps era força limitat. Aquestes millores serien :

- Creació d'una interfície gràfica en java que permetés seleccionar el fitxer XMI del disc dur de l'usuari i executés directament l'aplicació, sense necessitat d'entrar per línia de comandes.
- Aquesta interfície gràfica també podria permetre a l'usuari la interacció amb l'aplicació, amb lo que es podrien reajustar diferents paràmetres com, per exemple, la definició de longitud de literals que ha de tenir una variable (que ara mateix s'ha d'indicar de manera predeterminada amb tot el que això comporta en malbaratament d'espai d'emmagatzematge).

4. Agraïments.

Vull agrair a Silvia Saludes Palacino l'ajuda rebuda per la seva part amb el Power Point que m'ha ajudat a acabar a temps i no perdre massa temps en el disseny de la presentació.

5. Bibliografia

UML

Cre@ngel
Qué és UML.
[http://www.creangel.com/uml/, 6 de gener 2004]

Object Management Group
Introduction to OMG's Unified Modeling Language™ (UML™)
[http://www.omg.org/gettingstarted/what_is_uml.htm, 6 de gener 2004]

Steve Ash, DSDM Consortium
Selecting Support Tools for DSDM and UML Projects
[http://www.dsdm.org/timebox/newsletter2/umltools.asp,6 de gener 2004]

Eines CASE

Gentleware
Poseidon for UML
[http://www.gentleware.com/ 4 de gener 2004]

Novosoft, Software Development Service
Foundation Layer (FL)
[http://tech.novosoft-us.com/product_fl.jsp 4 de gener 2004]

Sygel
Wonder Machine Enterprise Edition (WMEE)
[http://www.sygel.com/en/Products/index.html, 4 de gener 2004]

Visual Object Modelers
Visual UML, eina CASE per al modelatge d'objectes
[http://www.visualuml.com, 4 de gener 2004]

XML

Natanya Pitts, Editorial Anaya, 1999
XML

Object Management Group
XML Metadata Interchange (XMI)
[http://www.omg.org/technology/documents/formal/xmi.htm, 06/01/2004]

W3C Organization
Extensible Markup Language (XML)
[http://www.w3.org/XML/, 6 de gener 2004]

Models de processament d'XML

Java en Castellano
Comparación de las Tecnologías Java para XML
[http://www.programacion.com/java/tutorial/javaxml/1/, octobre 2003]

Osmosis Latina
XML: DOM,SAX,JDOM y JAXP
[http://www.osmosislatina.com/xml/domsax.htm, octobre 2003]

SAX Project
SAX
[http://www.saxproject.org/, octubre 2003]

SUN Microsystems
Java API for XML Processing (JAXP) Downloads & Specifications
[http://java.sun.com/xml/downloads/jaxp.html, octubre 2003]

W3C Organization
W3C Document Object Model (DOM)
[http://www.w3.org/DOM/, octubre 2003]

6. Codi dels programes principals de l'aplicació

6.1 AnalitzadorXMl.java

```
package uoc.tfc2004.jperezv.umlbdor;
/* JAXP packages  */
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Vector;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
/**
 * Classe que permet obtenir els valors essencials per a
 * crear un script sql d'oracle a partir d'un fitxer xml. El path
 * d'aquest fitxer xml s'ha de passar com a argument de la classe.
 * Amb les llibreries JAXP es tracta aquest fitxer XML i creem un
 * arbre DOM. Un cop l'aplicació obté aquest arbre del document DOM,
 * va saltant pels nodes a l'arbre associant per a cadascun d'aquests
 * nodes els seus atributs.
 * Darrera Actualització 01-01-2004
 *
 * @author José Antonio Pérez Vidal; TFC-2003/04 Bases de Dades
 */
public class AnalitzadorXMl {
   /** Identificador de nivell a l'arbre */
   private int indent = 0;
   /* Constants utilitzades per JAXP 1.2 */
   static final String JAXP_SCHEMA_LANGUAGE =
      "http://java.sun.com/xml/jaxp/properties/schemaLanguage";
   static final String W3C_XML_SCHEMA =
      "http://www.w3.org/2001/XMLSchema";
   static final String JAXP_SCHEMA_SOURCE =
      "http://java.sun.com/xml/jaxp/properties/schemaSource";

   /* variables utilitzades per a seleccionar la informació dels  nodes    */
         /** vector que guarda les dades dels nodes tipus Atribut */
         private static Vector vAtributs = new Vector();
         /** vector que guarda les dades dels nodes tipus objecte */
         private static Vector classes = new Vector();
         /** vector que guarda els tipus dels objectes tipus Atribut */
         private static Vector vDades = new Vector();
         /** vector que guarda les dades dels nodes tipus Herència */
         private static Vector vHerencies = new Vector();
         /** vector que guarda les dades del tipus Associacio */
         private static Vector vAssociacions = new Vector();
         /** vector que guarda les dades del tipus Associacio n a n */
         private static Vector vAssociacionsNN = new Vector();
         /** cadena que indica què tipus d'element conté el node analitzat */
     private static String node = "";
         /** contenidor dels tipus Classe */
         private static Classe nClasse;
         /** contenidor dels tipus Atribut */
         private static Atribut nAtribut;
         /** contenidor dels tipus TipusDada que guarden el tipus dels objectes Atribut */
         private static TipusDada tDada;
         /**contenidor dels tipus Herencia, classe que guarda una relació on un objecte hereda d'altre */
         private static Herencia nHerencia;
         /** contenidor dels tipus Associacio, classe que guarda una associació entre dos objectes */
         private static Associacio nAssociacio;
         /** contenidor dels tipus AssociacioNN,classe que guarda una associació n a n entre 2 objectes */
         private static Associacio nAssociacioNN;

   // atributs que permeten guardar els valors que passarem als tipus Atribut i Classe
```

```
    /** variable auxiliar que ens permet guardar el nom d'una classe del diagrama en un objecte Classe */
    private static String nomClasse = null;
    /** variable auxiliar que ens permet guardar el tipus d'una classe del diagrama en un objecte Classe */
    private static String absClasse = null;
/** variable auxiliar que permet guardar l'identificador d'una classe del diagrama en un  objecte Classe */
    private static String idClasse = null;
/** variable auxiliar que ens permet guardar la visibilitat d'una classe del diagrama en un objecte Classe */
    private static String visClasse = null;

    /**
     * mètode constructor de la classe
     *
     */
    AnalitzadorXMI() {
        /*
         * abans de res eliminem el fitxer que volem crear en cas de que existeixi
         * això ho fem inicialitzant el fitxer.
         */
        try{
                String comentari = "-- FITXER SQL GENERAT A PARTIR DE DIAGRAMA DE "+
                                " CLASSES\n\nBEGIN";
                File fitxer = new File("script.sql");
                FileOutputStream f = new FileOutputStream(fitxer);
                f.write(comentari.getBytes());
                f.close();
        }catch(IOException e){
                        System.out.println("Error al constructor : "+e.getMessage()) ;
        }
    }

    /**
     *  mètode per a obtenir els atributs comuns d'un node DOM2 . És cridat desde echoNode(Node n)
     * @param Node n
     */
    private void classificarNode(Node n) {
                String auxAbs = null;
                if ( node.equals("classe") ){  // agafem les propietats de la classe
                        //  si la propietat de la classe és name vol dir que parlem del nom de l'objecte
                         // Classe
                        if ( ( n.getNodeName().equals("name") ) ) {
                                nomClasse =  n.getNodeValue();
                        }
                        // si la propietat de la classe és xmi.id vol dir que parlem de l'identificador de
                        // l'objecte Classe
                        if ( n.getNodeName().equals("xmi.id") ) {
                                idClasse =          n.getNodeValue();
                        }
                        // si la propietat de la classe és isAbstract ens indica si es tracta d'una classe
                        // abstracta o no
                        if ( n.getNodeName().equals("isAbstract") ) {
                                // aquesta condició és indispensable per a poder carregar
                                // correctament l'atribut absClasse
                                if (visClasse != null){
                                        absClasse = n.getNodeValue(); // nomes el carregarem quan
                                                                        //sigui una classe no tipus
                                }
                        //ens permet carregar aquest valor per a un tipus de dada Classe com un String
                                auxAbs = n.getNodeValue();
                        }

                        // si la propietat de la classe és isAbstract ens indica si es tracta d'una classe
                        // visible o no
                        if ( n.getNodeName().equals("visibility") ) {
                                visClasse = n.getNodeValue();
                        }

                        // quan tenim les mínimes dades podem crear un DataType especial tipus
                        //classe, tal com String
                        if ( (nomClasse != null) && ( idClasse != null ) && ( auxAbs != null )  ) {
```

```
                    tDada = new TipusDada(idClasse,nomClasse,true);
                    vDades.add(tDada);
                    tDada = null;
            }
            // quan tenim totes les propietats de la classe ja podem crear-la. Les classes
            // que pertànyen només a la referència dels tipus de dades, com String, no
            // entraran aquí doncs no tenen la propietat de visibilitat, aquesta propietat és
            // pròpia de les classes que nosaltres creem al diagrama.
            if ( (nomClasse != null) && ( idClasse != null ) && ( absClasse != null ) &&
                    (visClasse != null ) ) {
                    // ja podem crear la classe amb els paràmetres indicats
                    nClasse = new Classe(nomClasse, idClasse,
                                            absClasse.equals("true"), visClasse, null);
                    // afegim la classe al vector, encara ens faltara afegir els atributs
                    classes.add(nClasse);
                    // ara hem de canviar el ´nom del tipus de dada que abans havíem
                    // creat afegint _t doncs aquest serà el nom que tindrà a l'script

                    tDada = (TipusDada)vDades.lastElement();
                    tDada.setNom(nomClasse+"_t");
                    // si totes les propietats han estat complertes hem de tornar a posar a
                    // nul totes de nou
                    nomClasse = null;
                    idClasse = null;
                    absClasse = null;
                    visClasse = null;
            }
    }
    //      ara comprovem si el node a analitzar és d'atribut
    if ( node.equals("atribut") ){
            /*
             * aquí tractem els nodes de tipus atribut
             */
            if ( nAtribut == null){
            // si no existeix l'atribut cal crear-lo instanciant la classe
                    nAtribut = new Atribut();
            }
            // fiquem el nom de la classe a la que pertany l'atribut dins de l'atribut, així
            // queden relacionats
            nAtribut.setClasse(Utils.getNomClasse(classes));
            // si la propietat del node és el nom de l'atribut ho introduim a l'atribut

            if ( n.getNodeName().equals("name") ) {
                    nAtribut.setNom( n.getNodeValue());
            }
            // si la propietat del node és l'identificador ho introduim a l'atribut

            if ( n.getNodeName().equals("xmi.id") ) {
                    nAtribut.setIdent( n.getNodeValue());
            }
            // si la propietat del node és la visibilitat ho introduim a l'atribut

            if ( n.getNodeName().equals("visibility") ) {
                    nAtribut.setVisible( n.getNodeValue());
            }
            // si ja tenim tots els valors que necessitem de l'atribut ja ho podem introduir al
            // vector d'atributs
            if ( (nAtribut.getIdent()!= null )
                            && (nAtribut.getNom()!= null )
                            && (nAtribut.getVisible()!= null )
                             ){
                    vAtributs.addElement(nAtribut);
                    // un cop passat l'atribut el tornem a posar a nul per a que puguir
                    // tornar-se a fer servir
                    nAtribut = null;
            }
    }

    /*
```

```
 * aquí tractem els atributs que pertanyen a un tipus que és una classe, tal com passa
 * amb String
 */
if ( node.equals("classeTipusDada") ){

        //si la propietat és xmi.idref significa que es tracta d'una refència a un
        //  tipus Classe d'un atribut, com podría ser String, per exemple
        if (n.getNodeName().equals("xmi.idref")){
          // sabem que aquest tipus pertany al darrer atribut afegit al vector d'atributs
          // doncs el recorregut dels nodes de l'arbre xmi és seqüencial, una de les
          // raons per les que fem servir el parser DOM.
          Atribut unAtribut = (Atribut)vAtributs.lastElement();
          if ( unAtribut.getTipus() == null ){
                // afegim l'atribut tipus a l'objecte tipus Atribut en darrera posició del
                // vector d'atributs
                unAtribut.setTipus(n.getNodeValue());
          }
        }
}


/*
 *  el tipusDada és una etiqueta que pot significar un tipus de dada o una referència a
        *  aquest tipus de dada, en aquest darrer cas, caldrà afegir aquest valor a
l'atribut per a        *  que després se li pugui enllaçar aquest tipus.
 *  Distinguim els dos significats mitjançant la propietat del node, si és idref voldrà dir que
 *  és una  referència; si és xmi.id aleshores s'està definint un tipus int, boolean,...,
        *  encara que mai, seràn tipus instanciables, com pot ser l'String, el qual està
definit        *  sota l'etiqueta UML:Class i el qual el tractem als nodes tipus esClasse, com
ja hem
 *  vist abans
 */
if ( node.equals("tipusDada") ){
        if (n.getNodeName().equals("xmi.idref")){
                Atribut unAtribut = (Atribut)vAtributs.lastElement();
                if ( unAtribut.getTipus() == null ){
                        // afegim l'atribut tipus a l'objecte tipus Atribut.
                        unAtribut.setTipus(n.getNodeValue());
                }
        }else{
                if (tDada == null){
                        tDada = new TipusDada();
                }
                if (n.getNodeName().equals("xmi.id")){
                  tDada.setIdent(n.getNodeValue());
                }
                if (n.getNodeName().equals("name")){
                        tDada.setNom(n.getNodeValue());
                }
                if ( (tDada.getIdent()!= null )
                                && (tDada.getNom()!= null) ){
                        //afegim un nou TipusDada al vector de tipus de dades.
                        vDades.add(tDada);
                        // tornem a posar l'atribut tDada a nul per  tornar a fer-la servir
                        tDada = null;
                }
        }
}

// Ara analitzem si és tracta d'un node que indica una associació
if ( node.equals("assoc") )  {
        /*
         *  generem una associacio binària entre dos tipus diferents. Aquesta relació es
         *  guarda a un vector amb un tipus Associacio que té un nom i un identificador
         */
        if ( ( n.getNodeName().equals("xmi.id") ) ) {
                nAssociacio = new Associacio();
                nAssociacio.setId(n.getNodeValue());
                vAssociacions.add(nAssociacio);
```

```
                }
                if ( ( n.getNodeName().equals("name") )&& (!vAssociacions.isEmpty()) )

                {
                        nAssociacio = (Associacio)vAssociacions.lastElement();
                        nAssociacio.setNom(n.getNodeValue());
                }

        }

        // analitzem aquí els nodes que indiquen un participant en una associació
        if ( node.equals("endAssoc") ) {
                /*
                 *  obtenim d'aquest node el tipus d'agregació d'un objecte dins una associacio
                 *  entre dos objectes . A més, també obtenim si és navegable.
                 */
                if ( ( n.getNodeName().equals("aggregation") ) && (!vAssociacions.isEmpty())) {

                        nAssociacio = (Associacio)vAssociacions.lastElement();
                        /*
                         * com passava amb els tipus dada, en aquest cas sabem que els
                         * participants d'una associació es defineixen al fitxer xmi justament
                         * darrera de l'associació, per lo tant  podem trobar l'associació al
                         * darrer element del vector d'associacions. Li afegim a aquest
                                 * objecte associació primer el tipus del participant 1 i després
el                               * segón.
                         */
                        if ( nAssociacio.getObj1().equals("")){
                                nAssociacio.setTipus_obj1( n.getNodeValue());
                        }else{
                                nAssociacio.setTipus_obj2( n.getNodeValue());
                        }
                }

                if ( ( n.getNodeName().equals("isNavigable") ) && (!vAssociacions.isEmpty())) {

                        /*
                         * seguint les pautes anteriors fem el mateix per a indicar el sentit de la
                                 *  navegació de l'associació.
                         */
                        nAssociacio = (Associacio)vAssociacions.lastElement();
                        if ( nAssociacio.getObj1().equals("")){
                                nAssociacio.setNavObj1( n.getNodeValue().equals("true"));
                        }else{
                                nAssociacio.setNavObj2( n.getNodeValue().equals("true"));
                        }

                }
        }

        /*
         *  obtenim d'aquest node les participacions  d'un objecte participant
         *  dins una associacio entre dos objectes.
         *  existeix un número mínim(0..n) i un de màxim (1..n) de participacions
         */
        if ( node.equals("rangAssoc") ) {

                if ( !vAssociacions.isEmpty()){
                        // si la propietat analitzada es lower afegim el mínim de participacions
                                // de ambdós objectes
                        if ( ( n.getNodeName().equals("lower") ) ) {
                                nAssociacio = (Associacio)vAssociacions.lastElement();
                                if ( nAssociacio.getObj1().equals("")){
                                        nAssociacio.setMin_obj1(n.getNodeValue());
                                }else{
                                        nAssociacio.setMin_obj2(n.getNodeValue());

                                }
                        }
```

```
                        // si la propietat analitzada es upper afegim el màxim de participacions
                                // de ambdós objectes
                        if ( ( n.getNodeName().equals("upper") ) ) {
                                nAssociacio = (Associacio)vAssociacions.lastElement();
                                if ( nAssociacio.getObj1().equals("")){
                                        nAssociacio.setMax_obj1(n.getNodeValue());
                                }else{
                                        nAssociacio.setMax_obj2(n.getNodeValue());

                                }
                        }
                }
        }

        /*
         *  obtenim del node objAssoc els identificadors dels objectes
         *  que participen en una relació d'associació
         */
        if ( node.equals("objAssoc") )  {

                if ( ( n.getNodeName().equals("xmi.idref") ) ) {
                  if ( !vAssociacions.isEmpty()){
                        nAssociacio = (Associacio)vAssociacions.lastElement();
                        if ( nAssociacio.getObj1().equals("")){
                                nAssociacio.setObj1(n.getNodeValue());
                        }else{
                                nAssociacio.setObj2(n.getNodeValue());
                        }
                  }
                }
        }

        /*
         * si el node analitzat és un node d'herència
         *  generem una associacio binària entre dos tipus diferents
         *  on un d'ells hereta de l'altre. Es tracta d'herencia simple.
         *  Aquesta relació es guarda a un vector con un tipus Herencia
         */
        if ( node.equals("herencia") )  {

                // agafem la propietat d'herència del node dins un vector d'associacions
                        // d'herències
                if ( ( n.getNodeName().equals("xmi.id") ) ) {
                        nHerencia = new Herencia();
                        nHerencia.setIdent(n.getNodeValue());
                        vHerencies.add(nHerencia);
                }
        }

        /*
         * dintre de la mateixa associació d'herència cal que
         *  identifiquem quina és la classe pare de l'associació anterior
         */
         if ( node.equals("super") ){
                // guardem al darrer element del vector l'identificador de l'objecte pare
                if ( n.getNodeName().equals("xmi.idref") ) {
                        ((Herencia)vHerencies.lastElement()).setSuper(n.getNodeValue());
                }
        }

        /*
         * dintre de la mateixa associació d'herència cal que
         *  identifiquem quina és la classe filla de l'associació anterior
         */
        if ( node.equals("fill") ){
                // guardem al darrer element del vector l'identificador de l'objecte fill
                if ( n.getNodeName().equals("xmi.idref") ) {
                        ((Herencia)vHerencies.lastElement()).setFill(n.getNodeValue());
```

```
                        }
                    }
    } // fi classificarNode(Node n)

    /**
     * mètode que prové una rutina recursiva que obté els nodes de l'arbre DOM
     * @param Node n
     */
    private void echo(Node n) {
        int type = n.getNodeType();
            switch (type) {
                // si l'element que estem analitzant és una propietat d'un node només ens interessen
                        // aquestes propietats
                case Node.ATTRIBUTE_NODE:
                                if ( ( n.getNodeName().equals("name") ) ||
                                        ( n.getNodeName().equals("xmi.id") ) ||
                                ( n.getNodeName().equals("xmi.idref") ) ||
                                ( n.getNodeName().equals("isAbstract") ) ||
                                ( n.getNodeName().equals("visibility") ) ||
                                ( n.getNodeName().equals("aggregation") ) ||
                                ( n.getNodeName().equals("isNavigable") ) ||
                                ( n.getNodeName().equals("lower") ) ||
                                ( n.getNodeName().equals("upper") )
                                ) {
                    classificarNode(n);
                                }
                break;
                // si l'element analitzat és un element ( branca principal ) ens interessen aquestes etiquetes
                case Node.ELEMENT_NODE:
                                /*
                                 * abans de res cal reiniciar els valors dels atributs.
                                 * Tenim també un problema doncs les classes implicades dins una
                                        * herència venen dins d'etiquetes UML:Class on el node
superior és
                                 * UML:Generalization. * Quan tinguem un node Class
                                ens caldrà comprovar si l'anterior node llegit és Generalization
                                 * el qual marquem com node='herencia', o és del tipus fill, ja que és el
                                 * següent al herencia i anterior al node='super'. Per aquesta raò no
                                 * reiniciem un node que té la marca herencia o fill.
                                 * El mateix fem quan una etiqueta classe penja de una altre superior
                                        * que sigui MultiplicityRange o Attribute, doncs en aquestes
                                 * etiquetes es troben les referències a les classes que intervenen
                                 * en una associació o són atributs d'un tipus Classe, com pot ser un
                                        * atribut de tipus String.
                                 */
                                if (node != ""){
                                        if (!(node.equals("herencia")) &&
                                                !(node.equals("fill")) &&
                                                !(node.equals("rangAssoc")) &&
                                                !(node.equals("atribut"))
                                                ){
                                                node = "";
                                        }
                                }
                    if ( ( n.getNodeName().equals("UML:Class") ) ||
                            ( n.getNodeName().equals("UML:Attribute") ) ||
                                ( n.getNodeName().equals("UML:DataType") ) ||
                                        ( n.getNodeName().equals("UML:Association") ) ||
                                        ( n.getNodeName().equals("UML:AssociationEnd") ) ||
                                        ( n.getNodeName().equals("UML:MultiplicityRange") ) ||
                                        ( n.getNodeName().equals("UML:Generalization") )
                                    ){
                        if( n.getNodeName().equals("UML:Class") ){
                                if(node.equals("")){
                                                node = "classe";  // es defineix un tipus
                                                                        // Classe
                                }else{
                                                if (node.equals("fill")){
```

```
                                                    // es defineix que un tipus Classe
                                                            // no és final (es heredable)
                                                            node = "super";
                                                    }
                                                    if (node.equals("herencia")){
                                                    // es defineix que un tipus Classe hereda
                                                            //d'un altre
                                                                    node = "fill";
                                                    }
                                                    if (node.equals("rangAssoc")){
                                                    // es defineix que un tipus Classe forma part
                                            // d'una associacio
                                                                    node = "objAssoc";
                                                    }
                                                    if (node.equals("atribut")){
                                                    // es defineix que aquesta classe està
                                                            // referenciada com a tipus de dada
                                                                    node =

"classeTipusDada";
                                                    }
                                            }
                                    }
                                    if( n.getNodeName().equals("UML:Attribute") ){
                                    // es defineix un tipus Atribut
                                            node = "atribut";
                                    }
                                    if( n.getNodeName().equals("UML:DataType") ){
                                    // es defineix un tipus TipusDada que és dins un  tipus Atribut
                                            node = "tipusDada";
                                    }
                                            if( n.getNodeName().equals("UML:Association") ){
                                            // es defineix una relació d'associacio
                                                    node = "assoc";
                                            }
                                            if( n.getNodeName().equals("UML:AssociationEnd") ){
                                            // es defineix característiques de una relació  d'Associacio
                                                    node = "endAssoc";
                                            }
                                            if( n.getNodeName().equals("UML:MultiplicityRange") ){
                                            // es defineix més característiques ( les participacions ) d'una
                                                    // relació d'Associació
                                                    node = "rangAssoc";
                                            }
                                            if( n.getNodeName().equals("UML:Generalization") ){
                                            // es defineix una relació d'Herència simple entre dos objectes
                                            // de tipus Classe
                                                    node = "herencia";
                                            }
                                            // un cop hem recollit el tipus de l'ELEMENT_NODE anem
                                                    // recorrent el node (indent += 2)
                                            // per tal de recollir les propietats que ens interessen.
                                            // Aquestes propietats pertanyen al tipus ATTRIBUTE_NODE.
                                    NamedNodeMap atts = n.getAttributes();
                                    indent += 2;
                                    for (int i = 0; i < atts.getLength(); i++) {
                                        Node att = atts.item(i);
                                        echo(att);
                                    }
                                    indent -= 2;
                            }
                        break;

                        default:
                        break;
                    }

        // En cas que aquest node tingui fills baixarem fins les fulles per a obtenir la resta de
        // nodes que pengen d'aquest
```

```
        indent++;
        for (Node child = n.getFirstChild(); child != null;
            child = child.getNextSibling()) {
            echo(child);
        }
        indent--;
    } // fi echo(Node n)

        /**
         * mètode que enllaça els tipus objecte amb els seus atributs. Es cridat directament des del
mètode main()
         * d'AnalitzarXMI.
         */
        private static void throwFile(){

                // contenidor de cada classe del vector classes
                Classe nClasse ;
                // contenidor de cada atribut que es pugui crear per les relacions d'associació
                Atribut unAtribut ;
                // contenidor de cada associacio n a n
                AssociacioNN nAssociacioNN;

                // fem un recorregut de les classes
                for (int n = 0; n < classes.size(); n++) {
                        // agafem una classe del vector de classes
                        nClasse = (Classe)classes.elementAt(n);

                        // analitzem si hi ha una relació d'herència entre objectes
                        Utils.tractaHerencia(vHerencies,classes,nClasse);

                        // entrem en un recorregut de les relacions d'associació trobades
                        for ( int i=0; i<vAssociacions.size(); i++){

                                Associacio nAssociacio = (Associacio)vAssociacions.elementAt(i);

                                // comprovem si és una associació de referència, en aquest cas creem
                                // un atribut de la referència
                                unAtribut = Utils.esAssocRef(classes,nAssociacio,nClasse,1);
                                if ( unAtribut != null){
                                        vAtributs.add(unAtribut);
                                }
                                // tornem a comprovar si és una associació de referència, però amb el
                                // rol de les classes canviat
                                unAtribut = Utils.esAssocRef(classes,nAssociacio,nClasse,2);
                                if ( unAtribut != null){
                                        vAtributs.add(unAtribut);
                                }
                                // comprovem si és una associació d'agregació 1 a 1, com la de una
                                        // persona amb una adreça agregada
                                unAtribut = Utils.esAssocAgregacioUnivoca
                                                        (classes,nAssociacio,nClasse,1);
                                if ( unAtribut != null){
                                        vAtributs.add(unAtribut);
                                }
                                // tornem a comprovar si és una associació d'agregació, però amb el
                                        // rol  de les classes canviat
                                unAtribut = Utils.esAssocAgregacioUnivoca
                                                        (classes,nAssociacio,nClasse,2);
                                if ( unAtribut != null){
                                        vAtributs.add(unAtribut);
                                }
                                // comprovem si és una associació d'agregació múltiple i limitada,
                                // aleshores caldrà crear un vArray del tipus Classe indicat
                                unAtribut = Utils.esAssocAgregacioMultiple
                                                        (classes,nAssociacio,nClasse,1);
                                if ( unAtribut != null){
                                        vAtributs.add(unAtribut);
                                }
                                // tornem a comprovar si és una associació d'agregació múltiple i
```

```
                // limitada  però amb el rol de les classes canviat
                unAtribut = Utils.esAssocAgregacioMultiple
                                                (classes,nAssociacio,nClasse,2);
                if ( unAtribut != null){
                        vAtributs.add(unAtribut);
                }
                //comprovem si és una associació d'agregació múltiple no determinada
                // ( 1 a n) aleshores caldrà crear una taula del tipus Classe indicat
                unAtribut = Utils.esAssocAgregacioIndefinida
                                                (classes,nAssociacio,nClasse,1);
                if ( unAtribut != null){
                        vAtributs.add(unAtribut);
                }
                // tornem a comprovar si és una associació d'agregació múltiple no
                        // determinada però amb el rol de les classes canviat
                unAtribut = Utils.esAssocAgregacioIndefinida
                                                (classes,nAssociacio,nClasse,2);
                if ( unAtribut != null){
                        vAtributs.add(unAtribut);
                }

                // comprovem si és associació d'agregació múltiple indefinida(n a n)
                // aleshores caldrà crear una taula del tipus del nom de l'associació;
                // en aquest cas no s'afegeixen atributs sino associacionsNN que
                // després es tractaran  un cop s'han tractat els objectes.
                nAssociacioNN = Utils.esAssocAgregacioIndefinidaMultiple
                                                (classes,nAssociacio,nClasse);
                if ( nAssociacioNN != null){
                        boolean existeix = false;
                        for(int j = 0; j < vAssociacionsNN.size(); j++){
                                if ( nAssociacioNN.equals(
                                (AssociacioNN)vAssociacionsNN.elementAt(j) ) ){
                                        existeix = true;
                                }
                        }
                        if ( !existeix){
                                vAssociacionsNN.add(nAssociacioNN);
                        }
                }

        } // fi del bucle de comprovació de classes

        /*
         * fem un recorregut dels atributs i els comparem pel seu atribut classe amb el
         * nom de la classe que estem analitzant. Si la comparació es true, aleshores
                 * afegim l'atribut a la classe
         */
        for ( int i = 0; i < vAtributs.size(); i++){
                nAtribut = (Atribut)vAtributs.elementAt(i);
                for(int j = 0 ; j < vDades.size();j++){
                        tDada = (TipusDada)vDades.elementAt(j);
                        // afegim el tipus de la dada a cada atribut
                        // en cas de int i de String el passem als seus respectius tipus
                        // en bases de dades OO, com no tenim
                        // un delimitador de literals caldrà indicar un per defecte
                        if (nAtribut.getTipus()!= null){
                                if (nAtribut.getTipus().equals( tDada.getIdent()) ){
                                        if(tDada.getNom().equals("int"))
                                                tDada.setNom("NUMBER(10)");
                                        if(tDada.getNom().equals("String"))
                                                tDada.setNom("VARCHAR2(200)");
                                        nAtribut.setTipus( tDada.getNom() );

                                }
                        }
                }

                if(nClasse.getNom().equals(nAtribut.getClasse())){
                        nClasse.addAtribut(nAtribut);
```

```
                                        }
                                }
                        }
                } // fi throwFile()
/**
 * mètode principal de la classe que permet engegar e autoinstanciar la classe AnalitzarXMI
 */
    public static void main(String[] args) throws Exception {
        String filename = null;
        boolean dtdValidate = false;
        boolean xsdValidate = false;
        String schemaSource = null;
        boolean ignoreWhitespace = false;
        boolean ignoreComments = false;
        boolean putCDATAIntoText = false;
        boolean createEntityRefs = false;
        for (int i = 0; i < args.length; i++) {
            if (args[i].equals("-us")) {
                usage();
            } else if (args[i].equals("-help")) {
                usage();
            } else {
                filename = args[i];
                // és obligatori almenys un paràmetre, el fitxer xml a analitzar
                if (i != args.length - 1) {
                    usage();
                }
            }
        }
        if (filename == null) {
            usage();
        }
        // Pas 1: crear un DocumentBuilderFactory i configurar-ho
        DocumentBuilderFactory dbf =
            DocumentBuilderFactory.newInstance();
        // posem namespaceAware a true per a obtenir un arbre DOM de nivell 2 amb nodes
        // contenint informació del namespace. Això és necessari perquè el valor per defecte
        // del JAXP 1.0 va ser definit a false.
        dbf.setNamespaceAware(true);
        // indiquem el mode de validació a algú d'aquests : no validation, DTD validation
        // o XSD validation
        dbf.setValidating(dtdValidate || xsdValidate);
        if (xsdValidate) {
            try {
                dbf.setAttribute(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
            } catch (IllegalArgumentException x) {
                // això pot passar si el parser no soporta JAXP 1.2
                System.err.println(
                    "Error: Atribut JAXP DocumentBuilderFactory no reconegut: "
                    + JAXP_SCHEMA_LANGUAGE);
                System.err.println(
                    "Comprova que el parser pertany a la especificacio JAXP 1.2");
                System.exit(1);
            }
        }
        // indiquem l'schema source, si existeix.
        if (schemaSource != null) {
            dbf.setAttribute(JAXP_SCHEMA_SOURCE, new File(schemaSource));
        }
        // Opcional: indicar diferents opcions de configuració
        dbf.setIgnoringComments(ignoreComments);
        dbf.setIgnoringElementContentWhitespace(ignoreWhitespace);
        dbf.setCoalescing(putCDATAIntoText);
        // el contrari a crear entity ref nodes és expandir-n'els inline.
        dbf.setExpandEntityReferences(!createEntityRefs);
        // Pas 2: crear un DocumentBuilder que satisfagui les restriccions
        // especificades pel DocumentBuilderFactory
        DocumentBuilder db = dbf.newDocumentBuilder();
        // Pas 3: parsear el fitxer d'entrada
```

```
        Document doc = db.parse(new File(filename));
        // instanciem la classe la qual s'encarregarà del recorregut pel document en forma d'arbre
        new AnalitzadorXMI().echo(doc);
        // un cop instanciada i executada la classe AnalitzadorXMI tenim les dades parsejades i només
        // falta agrupar-les en objectes i els seus atributs ( reconstrucció dels diferents elements)
        throwFile();
                        // ara ja tenim tots els objectes preparats, només falta crear el fitxer cridant al mètode
                        // classeToFile de la classe Utils.
                        Utils.classeToFile(classes,"script.sql");
                        // un cop creats passats els objectes podem passar les associacions especials com N a
N cridant
                        // al mètode assocNNToFile de la classe Utils.
                        Utils.assocNNToFile(vAssociacionsNN,"script.sql" );
                        // si no s'ha produit cap incidència mostrem un missatge indicant que l'aplicacio ha
concluit amb èxit.
                        System.out.println("El fitxer script.sql ha estat creat en aquest directori.");
    } // fi main()
}
```

6.2 Classe.java

```
package uoc.tfc2004.jperezv.umlbdor;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Vector;
/**
 * Classe que encapsula tipus objecte amb un nom, identificador,
 * un boleà indicant si aquest objecte és abstracte i uns atributs
 * dintre d'un vector doncs no sabem quants atributs pot arribar
 * a tenir.
 *
 * Darrera actualització : 09/12/2003
 *
 * @author José Antonio Pérez Vidal
 */
public class Classe {
	/** nom de l'objecte */
	private String nom="";
	/** identificador de l'objecte */
	private String ident="";
	/** atribut que indica si l'objecte és abstracte */
	private boolean esAbstracte = false;
	/** atribut que indica la visibilitat de l'objecte */
	private String visible="";
	/** atribut que indica si de l'objecte hereda algun altre
objecte en una associació d'herència */
	private boolean esSuper=false;
	/** atribut que indica si l'objecte hereda d'algun altre en una
associació d'herència */
	private String pare="";
	/** atribut que indica si existeix un VARRAY d'aquest tipus
objecte i de quants elements; si és zero no existeix */
	private int isVARRAY = 0;
	/** atribut que indica si existeix una taula d'aquest tipus
objecte */
	private String taula = "";
	/** vector d'atributs que pertànyen a l'objecte */
	private static Vector atributs= null;
	/** vector d'objectes dels que depén aquest objecte */
	private Vector precedents = null;
	/** atribut que guarda la cadena amb la que es pot crear un
script d'un tipus objecte d'una basse de dades */
	private String unaClasse="";

	/**
	 * mètode constructor de la classe.
	 * No té paràmetres.
	 *
	 */
	public Classe(){
		nom = "";
		ident = "";
		esAbstracte = false;
		visible = "";
		esSuper = false;
		pare = "";
		isVARRAY = 0;
		taula = "";
		atributs = new Vector();
```

```
            precedents = new Vector();
      }
      /**
       * mètode constructor de la classe
       * amb uns quants paràmetres.
       * @param aNom
       * @param aIdent
       * @param esAbs
       * @param visible
       * @param vAtributs
       */
      public Classe(String aNom, String aIdent, boolean esAbs,String
aVisible, Vector vAtributs){
            nom = aNom;
            ident = aIdent;
            esAbstracte = esAbs;
            visible = aVisible;
            atributs = vAtributs;
            esSuper = false;
            isVARRAY = 0;
            taula = "";
            pare = "";
            precedents = new Vector();
      }
      /**
       * mètode que converteix l'objecte a string.
       * Aquest string crea un script sql a partir
       * dels paràmetres de l'objecte. Es cridat des de classeToFile
       * @return String
       */
      public String classeToString(){
            // creem la capçalera de la creació del tipus objecte
            String unaClasse="\nEXECUTE IMMEDIATE 'CREATE OR REPLACE"+
                        " TYPE "+nom+"_t";
            // si la classe té un pare ho hem d'indicar, sino li posem
            // la clàusula AS OBJECT
            if (this.getPare().equals("")){
                  unaClasse += " AS OBJECT (\n";
            }else{
                  unaClasse += " UNDER "+this.getPare()+"_t (\n";
            }
            Atribut vAtribut = null;
            // aquest proces ens permet afegir a l'String els atributs
            // de l'objecte
            if ( atributs != null){
                  int cont=0;
                  for( int i=0;i<atributs.size();i++){
                        vAtribut = (Atribut)atributs.elementAt(i);
                        if (nom.equals(vAtribut.getClasse())){
                        // si ja tenim un atribut li posem una coma al
                        //final de la línia d'atribut i pasem línia
                              if ( cont > 0){
                                    unaClasse += " , \n" ;
                              }
                              // afegim un nou atribut amb el seu tipus
                              unaClasse =
                       unaClasse+vAtribut.getNom()+""+vAtribut.getTipus();
                              cont ++;
                        }
                  }
            }else{
                  System.out.println("no grava atributs doncs el"+
                                    " vector es nul");
```

```
            }
            // un cop hem acabat amb els atributs tanquem paréntesi
            unaClasse += "\n) ";
            // si l'objecte és abstracte posem  NOT INSTANTIABLE
            if (this.esAbstracte){
                  unaClasse += " NOT INSTANTIABLE";
            }
            // si l'objecte és pare cal posar NOT FINAL indicant que
            // d'ell es pot heredar
            if (this.esSuper()){
                  unaClasse += " NOT FINAL";
            }
            // ara ja només queda saltar línia
            unaClasse += " ';\n";
            // si l'objecte té un tipus VARRAY referenciat cal crear-lo
            if (this.isVARRAY() > 0){
              unaClasse +="\nEXECUTE IMMEDIATE 'CREATE OR REPLACE "+
            "TYPE "+this.getNom()+"_v IS VARRAY("+this.isVARRAY()+
            ")OF" +this.getNom()+"_t ';\n";
            }
            // si existeix una taula de tipus de l'objecte cal crear-la
            if ( this.getTaula().equals(this.getNom())){
              unaClasse +="\nEXECUTE IMMEDIATE 'CREATE OR REPLACE "+
                  " TYPE "+this.getTaula()+"_nt AS TABLE OF "+
                  this.getTaula()+"_t ';\n";
            }
            // ja tenim complet l'objecte
            return unaClasse;
      }

      /**
       * mètode que construeix un fitxer a partir
       * del script que crea el mètode classeToString.
       * @param String fitxer
       */
      public void classeToFile(String fitxer){
            // agafem la cadena i fem un salt de línia, afegint la
            //cadena que retorna el mètode classeToString
            unaClasse = unaClasse+"\n"+this.classeToString();
            // un cop tenim la cadena la passem al fitxer indicat
            try{
                  File sqlfile = new File(fitxer);
                  FileOutputStream sortida = new
                                    FileOutputStream(sqlfile, true);
                  sortida.write(unaClasse.getBytes());
                  sortida.close();
            }catch(IOException e){
                  System.out.println("Error 1 :"+ e.getMessage());
            }
      }
}
```

6.3 Utils.java

```
package uoc.tfc2004.jperezv.umlbdor;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Vector;
/**
 * Classe que conté els mètodes necessaris per a treballar amb
AnalitzadorXMI.java
 *
 * Darrera actualització : 10/12/2003
 *
 * @author José Antonio Pérez Vidal
 */
public abstract class Utils {
	/**
	 * mètode que permet classificar les classes
	 * segons l'ordre de precedències,
	 * així es permet una correcta execució del script sql
	 *
	 * @param Vector classes
	 * @return Vector
	 *
	 */
	public static Vector sortClasses(Vector classes){
		Vector intercanvi = new Vector();
		// fem un primer bucle que seleccioni les classes sense
		// precedències
		for (int i =0; i < classes.size(); i++){
			Classe unaClasse = (Classe)classes.elementAt(i);
			if ( unaClasse.getPrecedents().isEmpty()){
				intercanvi.add(unaClasse);
				classes.remove(unaClasse);
			}
		}
		while (!classes.isEmpty()){
			// ara ja fem una cerca per a seleccionar els que
			// tenen precedencies
			for (int l =0; l < classes.size(); l++){
				boolean prec = false;
				Classe classe1 = (Classe)classes.elementAt(l);
				Vector precedents = classe1.getPrecedents();
				for (int j = 0; j < precedents.size(); j++){
					Classe classe2 =
						(Classe)precedents.elementAt(j);
					for (int k = 0; k < classes.size(); k++){
						Classe classe3 =
						(Classe)classes.elementAt(k);
						try{
							if (classe2.equals(classe3)){
								prec = true;
							}
						}catch(Exception e){
						}
					}
				}
				if(prec == false){
					intercanvi.add(classe1);
					classes.remove(classe1);
				}
```

```
                }
            }
            classes = intercanvi;
            return      classes;
        }
        /**
         * mètode que envia les classes al fitxer script sql.
         * Es cridat al main() un cop les classes han estat preparades
         * al throwFile()
         * @param Vector classes
         * @param String fitxer
         */
        public static void classeToFile(Vector classes, String fitxer){
                    // primer ordenem les classes al vector segons les
                    // depèndencies, això evita crear a l'script
                    // tipus objecte referenciant a altres tipus que
                    // encara no s'hàgin creat
                    classes = Utils.sortClasses(classes);
                    // fem un recorregut per totes les classes del vector
                    // construït de les classes o objectes
                    // i els passem al mètode d'una instància del mateix
                    // objecte Classe per a que ho formi al fitxer script
                    for (int j=0; j < classes.size(); j++){
                          Classe nClasse = (Classe)classes.elementAt(j);
                          nClasse.classeToFile(fitxer);
                    }
        }
        /**
         * mètode que construeix un fitxer a partir
         * del script que crea el mètode classeToString.
         *
         * @param Vector assocs
         * @param String fitxer
         */
        public static void assocNNToFile(Vector assocs, String fitxer){
            String unaAssocNN = "";
            for ( int i=0; i< assocs.size(); i++){
              AssociacioNN nANN=(AssociacioNN)assocs.elementAt(i);
              unaAssocNN += "\nEXECUTE IMMEDIATE 'CREATE TABLE "+
                  nANN.getNom()+"_nt(\n"+ nANN.getObjecte1()+"
                  "+nANN.getObjecte1()+"_t,\n"+ nANN.getObjecte2()+
                  " "+nANN.getObjecte2()+"_t\n)';\n";
            }
            unaAssocNN += "END;\n/\nshow errors;";
            try{
                  File sqlfile = new File(fitxer);
                  FileOutputStream sortida = new
                              FileOutputStream(sqlfile, true);
                  sortida.write(unaAssocNN.getBytes());
                  sortida.close();
            }catch(IOException e){
                  System.out.println("Error 1 :"+ e.getMessage());
            }
        }
}
```