Tipus Abstractes de Dades

i Disseny per Contracte

UOC (Universitat Oberta de Catalunya)
PFC (Projecte Final de Carrera)

Autor: Esteve Mariné i Gallisà
Carrera: Enginyeria en Informàtica
Consultor: Professor Jordi Cabot i Sagrera

Reus, 19 de juny de 2006

Dedicatòria:

A la meva família,
per la seva paciència

Agraïments:

Al personal docent de la UOC i,
en especial, als professors/a,
per ordre cronològic:

Xavier Franch,
Fatos Xhafa i
M. Elena Rodríguez,

per haver confiat en mi

Abstract

Biblioteca de tipus abstractes de dades (TADs) que incorpora la tecnologia de disseny per contracte (DbC) i consta bàsicament de dues parts. La primera, l'especificació, que és el punt d'entrada a l'API, és un conjunt d'interfícies, les quals introdueixen les operacions dels TADs i en fixen el comportament, així com defineixen els drets i obligacions dels usuaris. La segona, la implementació, determina una o més representacions dels valors del TAD i la codificació en J2SE 5 (*Tiger*) de les operacions, d'acord amb cadascuna d'aquestes representacions, que han de respectar les especificacions i estipulacions de la interfície respectiva. Les condicions del DbC s'introdueixen, com a comentaris del *javadoc*, precedides d'unes etiquetes especials. Consten d'unes sentències (invariants a nivell d'interfície i pre / postcondicions a nivell de mètodes) que segueixen la gramàtica del compilador estàndard i han de retornar un valor booleà. L'aplicació les llegeix en temps d'execució, mitjançant un *doclet*, en genera assercions (operador *assert*) i les situa en unes classes de nova creació que compila al vol. Un *Proxy* dinàmic, usant tècniques reflexives, s'encarrega d'interposar aquest codi de DbC entre el client i la implementació del TAD.

Keywords: *Abstract Data Types, Algebraic Specification, Design by Contract, Java, Doclet, Automated code generation, Reflection, Dynamic Proxy, Assertion, Runtime Contract Checker*

Àrea del PFC: Generació automàtica de programari

Índex de continguts

Índex de figures

1. Introducció

1.1 Justificació del PFC i context en el qual es desenvolupa: punt de partida i aportació del PFC

Es disposa d'una biblioteca de TADs (Tipus Abstractes de Dades), similar a la que utilitzen els alumnes de d'Estructura de la Informació de la UOC, implementada amb Java (JDK 1.4.x), de la qual es vol **eliminar el codi d'autodefensa**. Tal com es pot veure en els següents exemples, aquest codi salvaguarda la consistència del TAD de la biblioteca, però no fa palès l'error de l'aplicació client.

```
/**
 * Empila un element a continuació del que està al cim, si hi cap.
 * @see  java.util.Stack#push(Object)
 * @param elem  element que es vol afegir a la pila
 */
public void afegeix(Object elem)
{
   if ( ple() )  return;                                  // pila plena
   //...;
}
```

```
/**
 * Afegeix un element a la posició que li correspon, si hi cap.
 * Es deix com a exercici el tractament de l'error (cua plena, element
 * nul o no comparable amb els de la cua).
 * @see  CuaPrioritaria
 * @param elem  element comparable que es vol afegir a la cua
 */
public void afegeix(Object elem)
{
   if ( !(elem instanceof Comparable) )  return;  // elem no comparable
   //...;
}
```

Figura 1

En el primer cas, el client creu que està afegint elements, en un contenidor ple, i el TAD no fa res. En el segon, intenta afegir elements no comparables, en una cua prioritària, i el TAD tampoc fa res. En ambdós casos, les classes de la biblioteca no es veuen afectades, però l'aplicació client es desestabilitzarà sense haver rebut cap missatge d'error.

El codi d'autodefensa respon principalment a dos incompliments de contracte:

- Operacions no vàlides
  - Afegir un element en un contenidor fitat ple
  - Treure un element d'un contenidor buit
  - Etc.

- Paràmetres incorrectes
  - Tipus de dades inadequat
  - Valor de la dada incorrecte
  - Etc.

El PFC pretén aportar una solució consistent, didàctica, eficient i transparent per a l'usuari, basada en el disseny per contracte (DbC), implementat mitjançant generació automàtica de codi, a partir d'assercions introduïdes, com a etiquetes, als comentaris.

1.2 Objectius del PFC

El principal objectiu és garantir el compliment de les condicions del contracte, expressades mitjançant tres tipus d'assercions i una indicació, introduïdes com a comentaris del *javadoc* encapçalats amb les següents etiquetes:

- `@inv (booleà)`: Invariants a nivell d'interfícies
- `@pre (booleà)`: Precondicions a nivell de mètodes
- `@post (booleà)`: Postcondicions a nivell de mètodes
- `@pure`: Indica que la implementació del mètode no altera el contingut o estat del contenidor

Per aconseguir l'objectiu principal cal:

- **Reestructurar la biblioteca de TADs**

L'usuari només necessita consultar les interfícies de l'API, a la documentació generada pel *javadoc*. Allí veurà les funcionalitats que ofereix cadascuna, les signatures dels mètodes i les condicions del contracte. Un usuari expert consultarà, a més, la documentació de cada implementació de la interfície, per veure quina li convé, sospesant criteris d'eficiència espacial i temporal

- **Generar codi complementari de DbC**

Que es pugui interposar de forma transparent, entre l'aplicació client i els TADs, per verificar si es compleixen les condicions.

Per assolir els objectius es consideren adients els següents **requeriments**:

- Sintaxi de les assercions compatible amb el llenguatge Java

Es pretén que l'usuari no hagi d'aprendre un nou llenguatge per expressar o interpretar les assercions.

- Independència de la plataforma

La dóna el propi llenguatge de programació. L'aplicació ha de funcionar en qualsevol entorn on es pugui executar la màquina virtual de Java (JVM).

- Independència de la implementació concreta dels TADs

Atès que els TADs, a més de ser extensibles, es poden implementar de diverses formes, el contracte s'ha d'especificar a nivell d'interfícies.

- Independència de llibreries externes

L'aplicació només ha de necessitar la màquina virtual de Java (JVM) i el seu *runtime* estàndard (JRE).

- L'aplicació no ha d'afectar el codi font de la biblioteca

És a dir, s'ha de basar en el codi suplementari, sense modificar el codi font de la llibreria ni de les seves extensions.

- Autonomia de la biblioteca de TADs i de l'aplicació de DbC

La biblioteca de TADs s'ha de poder usar sense activar el DbC, així com l'aplicació de DbC ha de ser reutilitzable per verificar que qualsevol altra classe respecta les condicions declarades en una interfície que implementa.

- Compatibilitat amb les eines de Java (*javac*, *java* i *javadoc*)

El codi generat ha de ser Java pur, per poder ser documentat, compilat i executat amb les eines *javadoc*, *javac* i *java* estàndards, respectivament.

1.3 Enfocament i mètode seguit

Un dels incompliments de contracte esmentats, els paràmetres amb tipus de dades inadequats, ja està resolt amb el J2SE 5.0 de Sun. Els tipus genèrics permeten tenir un control del tipus de dades en temps de compilació i estalviar els *castings*. Per tant, un cop reestructurada la biblioteca de TADs, la primera mesura és actualitzar-la amb el JDK 1.5.x.

Des d'un enfocament teòric, les precondicions s'han de comprovar a l'entrada i les postcondicions a la sortida dels mètodes corresponents, així com els invariants a l'entrada i sortida de tots els mètodes de la classe:

- `{@pre} Constructor {@post & @inv}`
- `{@pre & @inv} Method {@post & @inv}`

Ara bé, si es parteix d'un estat consistent, es pot estalviar la comprovació dels invariants a l'entrada dels mètodes. Tot i amb això, és poc eficient fer la comprovació a la sortida de tots els mètodes d'una classe, quan n'hi ha molts que no afecten, per a res, el contingut o estat de l'objecte.

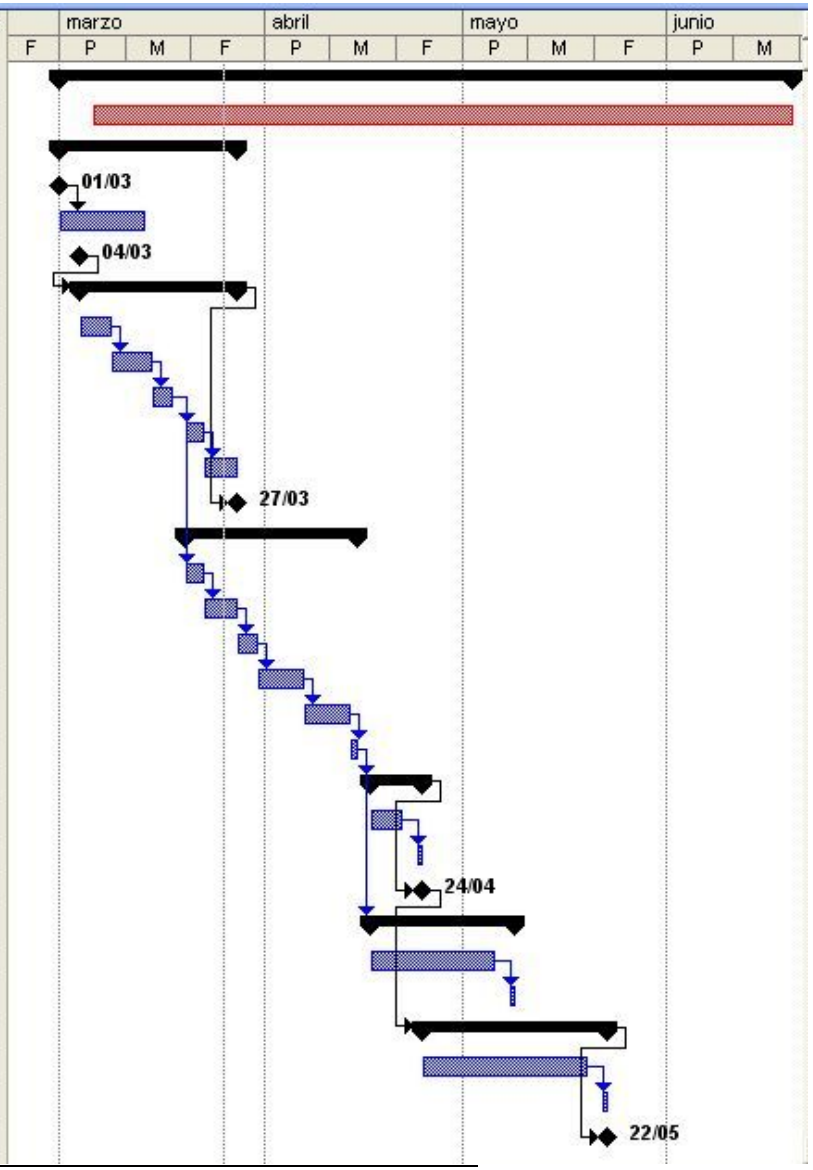
Aleshores, el projecte s'enfoca per etapes, amb tests incrementals:

1. Estudi preliminar de treballs relacionats
2. Estudi de la possibilitat d'adaptació de codi obert reusable
3. Generació d'assercions a partir de la informació del *doclet*
4. Generació de codi que permeti executar les assercions
5. Inserció del codi generat a l'entrada o sortida dels mètodes
6. Optimització per reducció del nombre de comprovacions
7. Integració amb les famílies de TADs de la biblioteca
8. Documentació, instal·lació i configuració

1.4 Planificació del projecte

Diagrama de Gantt

	Nombre de tarea	Duración	Comienzo	Fin	Predecesora:
1	**Implementació del projecte**	**80 días**	**mié 01/03/06**	**lun 19/06/06**	
2	Documentació i integració	76 días	lun 06/03/06	lun 19/06/06	
3	**Definició de Serveis**	**20 días**	**mié 01/03/06**	**lun 27/03/06**	
4	Inici del curs 2005/06-2	0 días	mié 01/03/06	mié 01/03/06	
5	Estudis preliminars	2 sem.	mié 01/03/06	lun 13/03/06	4
6	Trobada presencial d'inici de curs	0 días	sáb 04/03/06	sáb 04/03/06	
7	**Pla inicial del Projecte**	**17 días**	**sáb 04/03/06**	**lun 27/03/06**	**6**
8	Anàlisi de requeriments	4 días	sáb 04/03/06	mié 08/03/06	
9	Definició d'objectius	4 días	jue 09/03/06	mar 14/03/06	8
10	Arquitectura i funcionalitats	3 días	mié 15/03/06	vie 17/03/06	9
11	Estrategies d'implementació	3 días	lun 20/03/06	mié 22/03/06	10
12	Schedule del projecte	3 días	jue 23/03/06	lun 27/03/06	11
13	Lliurament PAC 1	0 días	lun 27/03/06	lun 27/03/06	7
14	**Generació d'assercions**	**20 días**	**lun 20/03/06**	**vie 14/04/06**	
15	Especificació matemàtica dels TADs	3 días	lun 20/03/06	mié 22/03/06	10
16	Gramàtica de les assercions	3 días	jue 23/03/06	lun 27/03/06	15
17	Inserció assercions a les interfícies	3 días	mar 28/03/06	jue 30/03/06	16
18	Captura de les assercions (doclet)	1 sem	vie 31/03/06	jue 06/04/06	17
19	Expansió funcions auxiliars	1 sem	vie 07/04/06	jue 13/04/06	18
20	Test	1 día	vie 14/04/06	vie 14/04/06	19
21	**Generació de codi font**	**6 días**	**lun 17/04/06**	**lun 24/04/06**	**20**
22	Generació classes d'interposició	1 sem	lun 17/04/06	vie 21/04/06	
23	Test	1 día	lun 24/04/06	lun 24/04/06	22
24	Lliurament PAC 2	0 días	lun 24/04/06	lun 24/04/06	21
25	**Interposició de codi DbC**	**16 días**	**lun 17/04/06**	**lun 08/05/06**	**20**
26	Implementació dynamic proxies	3 sem.	lun 17/04/06	vie 05/05/06	
27	Test	1 día	lun 08/05/06	lun 08/05/06	26
28	**Optimització verificació d'invariants**	**20 días**	**mar 25/04/06**	**lun 22/05/06**	**24**
29	Implementació visitors	19 días	mar 25/04/06	vie 19/05/06	
30	Test	1 día	lun 22/05/06	lun 22/05/06	29
31	Lliurament PAC 3	0 días	lun 22/05/06	lun 22/05/06	28



1.5 Productes obtinguts

Tant a la biblioteca de TADs (tads.jar) com a l'aplicació de DbC (dbc.jar) hi ha inclosos els tests i els exemples, així com la màxima informació pel *debugger*.

- tads.jar (112 KB)

Especificació i condicions del contracte, a nivell d'interfícies, de les principals famílies de TADs. Implementació d'alguna de les representacions de cadascuna.

- dbc.jar (19 KB)

Aplicació de generació i interposició de codi DbC, a partir de la informació recollida pel *javadoc*. Es pot habilitar o deshabilitar amb l'opció `java –ea ...`

- docs (6 MB)

Documentació dels dos anteriors generada automàticament pel *doclet* estàndard a partir dels fitxers font. Inclou tota la informació bàsica del *javadoc*, amb les assercions, més els diagrames UML, el codi dels fitxers font, els enllaços de totes les classes del JDK i un manual de *getting started* (`docs\index.html#Help`).

1.6 Breu descripció dels altres capítols de la memòria

2. **Treballs relacionats**. Descriu les aplicacions conegudes que d'alguna forma utilitzen el disseny per contracte. Després d'analitzar les seves característiques, s'arriba a la conclusió que no n'hi ha cap que s'adapti als requeriments considerats idonis.

3. **Especificació del contracte**. Després d'especificar dos tipus de dades primitius, d'us comú (naturals amb zero i booleans), es descriu, mitjançant l'exemple d'una Pila, la forma de traspassar l'especificació algebraica d'un TAD a codi de disseny per contracte.

4. **Redisseny de la biblioteca de TADs**. A més d'assenyalar els avantatges (especialment la cobertura de tipus de dades genèrics) d'actualitzar la implementació de la biblioteca de TADs a JDK 1.5, aquesta es redissenya perquè els usuaris disposin d'un API, basat en interfícies, on puguin veure les especificacions o funcionalitats dels tipus abstractes de dades, així com les estipulacions del contracte (DbC).

5. **Disseny de l'aplicació**. Dins del disseny de l'aplicació de DbC, en primer terme, es distingeixen dos paquets (tads i dbc). Del primer, que conté la biblioteca de TADs, se n'extreu una part complexa que serveix per aplicar-hi un prototip. El segon paquet recull, de manera incremental, els mòduls de generació i d'interposició del codi DbC que s'afegeixen al prototip.

6. **Generació d'assercions**. Entra en el detall de la generació d'assercions (precondicions, invariants i postcondicions), a partir de la informació recollida pel *javadoc*, que s'adapten a l'estructura de l'operador *assert*.

7. **Generació de codi executable**. Explica la generació i la compilació al vol de les tres classes que contenen les assercions (una pels invariants i les altres dues pels mètodes amb pre o postcondicions, respectivament).

8. **Interposició de codi DbC**. Detalla com un *proxy* dinàmic s'encarrega, mitjançant tècniques reflexives, d'interposar el codi DbC generat entre el client i el TAD, i com s'optimitza l'aplicació generant i executant les verificacions només quan és necessari.

9. **Joc de proves**. Descriu diferents tipus de proves que es complementen i la problemàtica que representa no poder fer proves unitàries, ja que el JUnit fa servir una tecnologia similar (*Reflection* i captura d'*Assertion Errors*) a la de l'aplicació de DbC, que les fan incompatibles.

2. Treballs relacionats

El Disseny per Contracte (DbC) fou introduït l'any 1992, per Bertrand Meyer, al llenguatge Eiffel, perquè les classes d'un sistema es poguessin comunicar, les unes amb les altres, sobre la base d'uns beneficis i obligacions definits amb precisió.

	Benefit	**Obligation**
Client	- no need to check output values - result guaranteed to comply to postcondition **(4)**	satisfy preconditions **(1)**
Provider	**(2)** - no need to check input values - input guaranteed to comply to precondition	**(3)** satisfy postconditions

Figura 3

La regla general del programari de DbC és verificar les assercions tal com s'indica a la taula següent, extreta del context dels *triggers* de JMScript:

Event		**Invariant**	**Pre-condition**	**Post-condition**
Method Entry	***Method***	Y	Y	-
	Constructor	N	Y	-
Method Exit	***Method***	Y	-	Y
	Constructor	Y	-	Y

Figura 4

2.1 iContract (iContract-jdk1_2.zip 02-06-2000 518 KB)

Va ser desenvolupat en Java l'any 1998 per Reto Kramer seguint la idea de Bertrand Meyer. Genera codi font amb verificacions dels invariants, així com de les pre i postcondicions associades als mètodes, declarades a les interfícies i classes. Els comentaris etiquetats (@pre, @post i @invariant) són interpretats per iContract i convertits en comprovacions d'assercions que s'insereixen al codi font.

El programari està obsolet i la web caiguda des de fa força temps. Sembla que el codi font no era lliure. Utilitzava un *lexer* i un *parser* propis, sense aprofitar les actuals potencialitats del *doclet* estàndard.

2.2 DBCProxy (dbcproxy.jar 21-01-2002 28.5 KB)

Afegeix DbC a Java fent servir proxys dinàmics i generació de codi. Les assercions, igual que a l'iContract, també es declaren com a comentaris del *javadoc*, amb les mateixes etiquetes, i és d'aplicació a nivell d'interfícies.

El codi no està actualitzat i té alguna incompatibilitat com la utilització d'una funció anomenada *assert*, perquè aquest mot es va declarar paraula reservada amb el J2SE 1.4. Existeix una altra versió del 2004 que utilitza la llibreria Javassist (*Java Programming Assistant*) per a manipular els *bytecode*, i la de programació orientada a aspectes (AOP) de JBoss.

2.3 JMSAssert (JMSAssert_1.02 19-02-2002 1246 KB)

Els programes de java que requereixen especificació rigorosa la poden introduir a nivell de codi font usant certes etiquetes entremig dels comentaris del *javadoc*. El següent pas és executar JMSAssert sobre el codi, el qual crea automàticament arxius de DbC que contenen codi en JMScript. Aquest codi generat representa senyals que són invocades per les assercions en temps d'execució per forçar explícitament les obligacions del contracte entre proveïdors i consumidors.

JMSAssert no és programari lliure i només es permet avaluar temporalment.

2.4 Barter (barter-0_2_0.zip 07-08-2002 27.5 KB)

És una eina per incrementar la qualitat de les aplicacions escrites en java. Permet al programador fer servir DbC i definir aspectes de desenvolupament exactament a les classes i interfícies que són rellevants, com a comentaris del *javadoc*. És essencialment un generador de codi per a AspectJ (veure el punt 2.8), implementat com a un *xDoclet* (extensió del *doclet* estàndard).

2.5 jContractor (jcontractor-0.1.tar.gz 02-02-2003 1.06 MB)

És una implementació del DbC pel llenguatge java escrita integrament en java. Els contractes són escrits com a mètodes que segueixen una convenció de nomenament. jContractor proporciona verificació del contracte en temps d'execució per instrumentació del codi binari de les classes que defineixen contractes. jContractor pot afegir codi de verificació a fitxers per executar posteriorment, o pot instrumentar classes en temps d'execució quan són carregades a memòria. Tots els contractes estan escrits en java estàndard, de tal forma que no cal aprendre un altre llenguatge d'especificació. jContractor es basa en la llibreria estàndard i no requereix preprocessat o modificacions de la JVM.

Utilitza BCEL (*Byte Code Engineering Library*) que facilita als usuaris una forma pràctica d'analitzar, crear i manipular fitxers precompilats de java (.class).

2.6 Jass (jass-2.0.12.jar 03-03-2005 850 KB)

Jass (*Java with assertions*) és un precompilador que suporta una extensió de les assercions pel java, transferides dels conceptes de DbC de Bertrand Meyer, ampliades amb noves funcionalitats. El precompilador està escrit en java pur al 100%.

Les pre/postcondicions s'introdueixen com a comentaris a l'interior dels mètodes, la qual cosa representa una pèrdua d'eficiència sense contribuir a clarificar el codi (es podrien posar directament assercions executables).

2.7 JML (JML.5.2.tar.gz 12-07-2005 13.56 MB)

JML (Java Modeling Language) és un llenguatge d'especificació del comportament que pot ser usat per a especificar el comportament dels mòduls de Java (com en el disseny per contracte, DbC). Té moltes eines per a implementar verificació d'assercions, tests unitaris, etc.

No té punt de comparació amb els programari DbC descrit anteriorment, pel volum de les llibreries, el llenguatge d'especificació singular, la declaració de les assercions com a comentaris no reconeguts pel *doclet* estàndard, etc.

Té l'inconvenient de tenir una llibreria molt voluminosa, desproporcionada en relació a la de la biblioteca de TADs, que ocupa uns 150 KB.

2.8 OEscudero (project.jar 02-01-2005 306 KB)

Projecte final de carrera de l'alumne Òscar Escudero Sànchez, dirigit pel professor Jordi Àlvarez Canal, basat en aspectes, amb una finalitat similar a l'actual, però que no compleix els requeriments que es consideren idonis per assolir els objectius que es plantegen:

PROJECTE ANTERIOR	PROJECTE ACTUAL
- Utilitza una llibreria molt voluminosa d'AOP (AspectJ)	- Només fa servir la llibreria estàndard del JDK (*Java Development Kit*)
- No considera que els TADs poden tenir diverses implementacions i, per tant, les condicions del contracte han d'estar a nivell d'interfícies	- L'especificació dels TADs es reflexa, a nivell de la jerarquia d'interfícies, amb assercions derivades de les equacions que defineix el model matemàtic
- Les especificacions tenen restriccions lèxiques i sintàctiques	- Les assercions només han de seguir la gramàtica del compilador estàndard
- Fa comprovacions sobre atributs de la llibreria, que tenien visibilitat *private* o *protected*, i s'han modificat a *public*, en contra de la doctrina general sobre OOP	- Les assercions consulten paràmetres i mètodes definits a les interfícies, sense afectar l'estat del contenidor i sense modificar per a res el codi font de la biblioteca de TADs
- Quan detecta un incompliment de contracte es limita a llançar l'excepció amb missatge "Violació del contracte", sense cap més explicació	- Especifica si s'ha violat un invariant o una pre/postcondició, quina és, on està declarada (classe o mètode i línia) i, a més, la informació del *stack trace*

Figura 5

Els treballs relacionats, descrits anteriorment, no s'ajusten a la majoria de requeriments (veure 1.2), que s'han considerat idonis per assolir els objectius.

3. Especificació del contracte

En un primer nivell, el TAD té una especificació matemàtica amb uns tipus de dades auxiliars, unes operacions, unes equacions i unes possibles situacions d'error (inconsistència). A l'hora de fer la programació d'un TAD, orientada a l'objecte, les operacions, amb els seus paràmetres, es distribueixen dins d'una jerarquia d'interfícies. El problema principal és assegurar que les diverses implementacions del TAD, així com els usuaris, respectin les equacions i evitin les situacions d'error.

La solució proposada és de tipus DbC (disseny per contracte). Les condicions esmentades es reflecteixen a l'API de la biblioteca de TADs, mitjançant invariants a nivell d'interfície i pre i postcondicions a nivell de mètodes.

Les assercions involucren tots els descendents de la interfície. Per exemple, un mètode sobreescrit pot afegir restriccions, però no treure'n.

3.1 Funcions auxiliars

Per derivar l'especificació matemàtica del TAD cap a la interfície s'usen quatre funcions auxiliars:

- `$obj()`: retorna l'objecte (instància de la classe) en el seu estat actual
- `$old()`: retorna l'objecte en l'estat anterior a l'execució d'un mètode
- `$oldparam(p.x)`: retorna el valor `x` obtingut, a partir d'un argument, abans de l'execució d'un mètode
- `$return()`: resultat retornat, un cop executat un mètode

La primera és d'ús general. En canvi, les tres darreres només es poden utilitzar dins de les postcondicions, perquè no tenen cap sentit abans de l'execució del mètode.

També s'usen les pròpies funcions definides al TAD, però hi ha la restricció de no fer servir operacions que alteren el contingut del contenidor (veure especificació algebraica), perquè els resultats serien indeterminats o impredictibles. Per evitar-ho, es marquen les operacions observadores o consultores amb l'etiqueta `@pure` i es sobreentén que les demés són constructores o modificadores.

Les assercions condicionals es representen per l'operador:

- `$implies`: booleà $\Rightarrow$ asserció

```
/**
 * Afegeix un element a la seqüència, si hi cap.
 * Sobreescriu Sequence.put() per afegir un parell d'assercions.
 * @param elem  element que es vol afegir a la seqüència
 *
 * @post  $old().isEmpty() $implies $obj().get() == elem
 * @post  !$old().isEmpty() $implies $obj().get() == $old().get()
 */
public void put(Elem elem);
```

Figura 6

és a dir, en aquesta operació de les cues (TAD **Queue**<Elem>), si es compleix la premissa aleshores s'avalua l'asserció.

3.2 Exemple d'una Pila

Abans de fer l'especificació del TAD d'exemple, es fa la d'un parell de tipus de dades primitius d'ús molt comú (lògics i naturals amb el zero).

```
univers Boolean especificació
  gènere boolean
  generadores
    true: ----> boolean
    false: ----> boolean
  operacions
    NOT : boolean ----> boolean
     OR : boolean boolean ----> boolean
     AND : boolean boolean ----> boolean
  equacions  b: boolean
    NOT true = false
    NOT false = true
    b OR true = true
    true OR b = true
    b OR false = b
    false OR b = b
    b AND true = b
    true AND b = b
    b AND false = false
    false AND b = false
funivers
```

```
univers Natural especificació
  usa Boolean
  gènere natural
  generadores
    0: ----> natural
    suc: natural ----> natural
  operacions
    pred: natural ----> natural
    isZero: natural ----> boolean
    _+_: natural natural ----> natural
    _≤_: natural natural ----> boolean
  errors  n: natural
    pred(n): isZero(n)
  equacions  n,m: natural
    isZero(0) = true
    isZero(suc(n)) = false
    n + 0 = n     n + suc(m) = suc(n + m)
    0 ≤ n = true     suc(n) ≤ 0 = false
    suc(n) ≤ suc(m) = n ≤ m
    pred(suc(n)) = n
funivers
```

Figura 7

Amb aquests tipus de dades auxiliars ja es pot especificar l'univers d'un TAD. S'escull la pila afitada per ser un dels més populars.

```
--/->     altera el contingut del TAD
---->     no altera el contingut del TAD
cursiva   operació derivada

univers Stack(Element) especificació
  usa Natural, Boolean
  gènere stack
  generadores
    new: ----> stack
    put: stack element --/-> stack
  operacions
    remove: stack --/-> stack
    get: stack ----> element
    count: stack ----> natural
    capacity: stack ----> natural
    isEmpty: stack ----> boolean
    isFull: stack ----> boolean
  errors  s: stack
    put(s): isFull(s)
    remove(s): isEmpty(s)
    get(s): isEmpty(s)
  equacions  s: stack, e: element
    remove(put(s,e)) = s
    get(put(s,e)) = e
    count(new) = 0
    count(put(s,e)) = suc(count(s))
    isEmpty(s) = isZero((count(s))
    isFull(s) = (count(s) = capacity(s))
funivers
```

```
@inv $obj().count() >= 0
@inv $obj().count() <= $obj().capacity()
public interface BoundedStack<Elem>
{
   @pure
   public int count();

   @pure
   @post $return() == ($obj().count() == 0)
   public boolean isEmpty();

   @pure
   public int capacity();

   @pure
   @post $return() == ($obj().count() == $obj().capacity())
   public boolean isFull();

   @post $obj().count() == $old().count() + 1
   @post $obj().get() == elem
   public void put(Elem elem);

   @pure
   @pre !$obj().isEmpty()
   @post $obj().count() == $old().count()
   public Elem get();

   @pre !$obj().isEmpty()
   @post $obj().count() == $old().count() – 1
   @post $return() == $old().get()
   public Elem remove();
}
```

Figura 8

L'especificació anterior recull l'herència de les interfícies antecessores **Sequence**, **BoundedContainer** i **Container**, tant pel que fa als seus mètodes com a les assercions allí definides.

Per exemple, l'invariant `@inv $obj().count() >= 0`, definit a **Container**, assegura que el nombre d'elements, qualsevol que sigui l'estat del contenidor, serà un natural (amb zero) i `@inv $obj().count() <= $obj().capacity()`, heretat de **BoundedContainer**, assegura que el nombre d'elements ha de ser igual o inferior a la capacitat i, de retruc, que la capacitat també ha de ser un natural.

En canvi, la postcondició `@post $return() == $old().get()`, que **Stack** afegeix al mètode esborrar de **Sequence**, diu que l'element retornat ha de ser igual que el del cim abans de fer l'operació i se sap que aquest element del cim és el darrer que s'ha empilat, per la postcondició expressada al mètode empilar `@post $obj().get() == elem`.

4. Redisseny i actualització de la biblioteca de TADs

L'objectiu principal del nou disseny és que els usuaris disposin d'un API, generat pel *javadoc*, on puguin veure les especificacions o funcionalitats dels tipus abstractes de dades (TADs), així com les estipulacions del contracte (DbC).

Les principals famílies de TADs, definides a l'API mitjançant interfícies, són:

• Sequence	• Function	• PositionalList	• BinaryRelation
• Stack	• Bag	• Tree	• Digraph
• Queue	• Set	• BinaryTree	• Graph

Figura 9

L'actualització de la biblioteca de TADs resulta convenient per aprofitar les noves prestacions del JDK 1.5, en especial, la dels tipus de dades genèrics.

4.1 Enfocament

Per optimitzar la biblioteca de TADs i adaptar-la al disseny per contracte, s'han adoptat les següents decisions:

- DbC independent de les implementacions

Les estipulacions a nivell d'interfícies són vàlides per a qualsevol implementació del TAD. Per exemple, un Arbre es pot implementar sobre un vector, per llistes fitades o encadenades de fills, per delegació en un arbre binari (*leftmost-child*, *right-sibling*) enfilat (*threaded tree*) o no, ... però a tots afecta, per exemple, l'asserció que si el pare és nul el fill ha de ser l'arrel, perquè tots els nodes d'un arbre, tret de l'arrel, han de tenir un pare.

- Compatibilitat de noms

La nomenclatura del codi es posa en anglès per estalviar problemes de traducció. Per una altra banda, els noms de les operacions bàsiques es comparteixen per les diferents famílies de TADs. Per exemple, *put*, *remove*, *get* i/o *exists* serveixen per definir el fet d'afegir, suprimir o obtenir un element de qualsevol contenidor, o comprovar-ne l'existència, respectivament. El DbC s'encarrega de distingir, per exemple, entre l'operació homònima d'afegir a una Cua, a una Pila o a una Llista.

- Propagació de les assercions

Les assercions definides en una interfície es propaguen a tota la descendència. És a dir, els descendents poden afegir restriccions, però no treure'n. Per exemple, alguns autors deriven el Sac del Conjunt i aquí es fa a l'inrevés afegint una postcondició a l'operació d'afegir del Conjunt, que assegura que el nombre d'ocurrències de l'element, rebut com a paràmetre, serà exactament igual a u després de la inserció, perquè el Conjunt no permet repeticions.

- Recorreguts innocus

Atès que hi ha recorreguts, basats en la interfície *Iteration*, a tots els nivells de la jerarquia de TADs, qualificats com a *@pure* (que no han d'afectar l'estat del contenidor) i deriven de la interfície *Iterable*/*Iterator*, s'ha fet una classe abstracta (*TIteration*) que buida el mètode heretat *remove*(). Tots els iteradors es retornen com a noves classes (protegides o anònimes) derivades d'aquella.

- Recorreguts eficients

Molts recorreguts es retornaven en una seqüència amb un cost inicial lineal, tant en espai com en temps. Per aconseguir que els recorreguts es puguin retornar amb cost asimptòtic temporal O(k) i sense cost espacial addicional, s'han transformat els recursius, com els dels arbres (*preorder*, *inorder* o *postorder*) en iteratius i, a més, es retorna directament un iterador. Fins i tot, els típics algorismes dels grafs (*BFS*, *DFS*, *TOS*, *Dijkstra* o *Prim*) s'han adaptat a la interfície *Iteration*, la qual no ha d'afectar, per a res, l'estat del contenidor.

- Extensibilitat del DbC

Les assercions del DbC estan pensades perquè siguin compatibles amb futures extensions tant de les pròpies interfícies com de les classes que les implementen. Els recorreguts implementats amb els esmentats iteradors innocus, accessibles a tots els nivells i compatibles amb el bucle *for-each* del JDK 1.5, permeten ampliar les funcions auxiliars de DbC amb el *@forall* i l'*@exists*.

4.2 Interfícies

Aquest primer nivell de l'API és basa únicament amb interfícies i, per tant, ha de ser independent de les diverses implementacions dels TADs. Per ampliar les funcionalitats o modificar les condicions del contracte, primer s'ha de fer una extensió d'una o més interfícies i seguidament implementar la nova interfície mitjançant una classe.

4.2.1 Contenidores i Iteradors

Al cim de la jerarquia es troba el Contenidor, que permet consultar si està buit i quants i quins elements hi ha emmagatzemats. Cap de les tres operacions altera el seu estat.

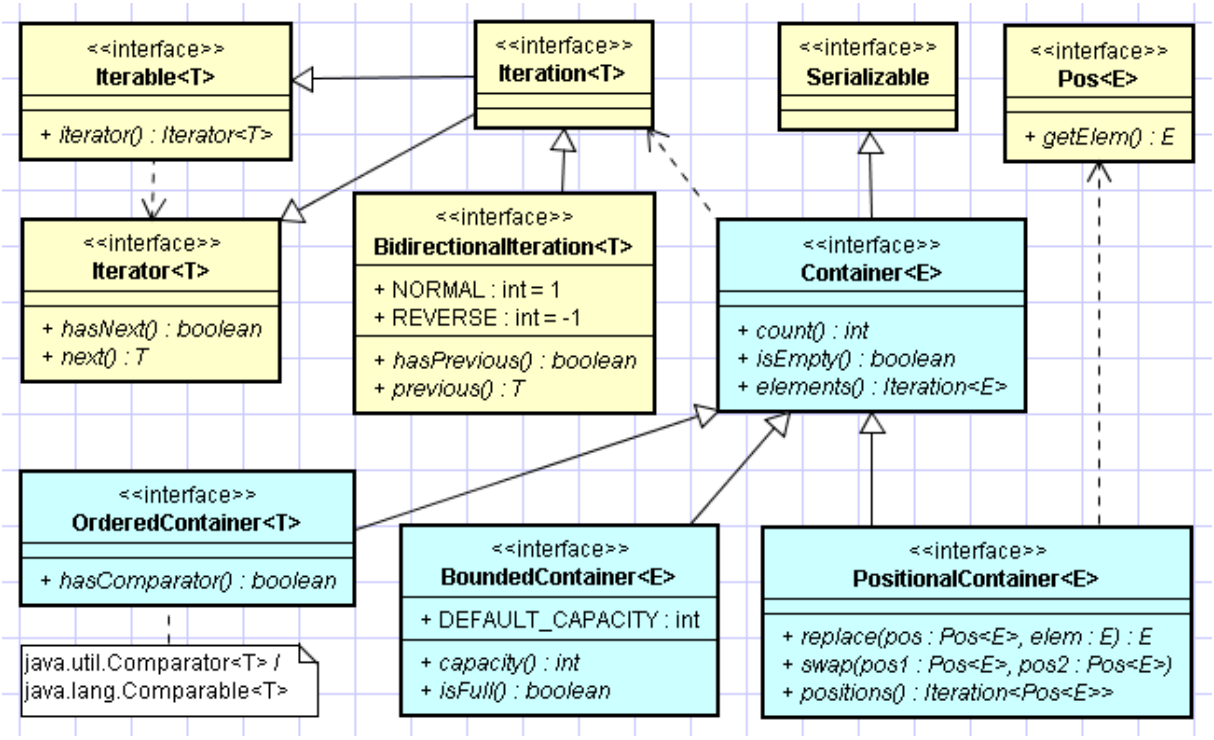


Figura 10

Estén la interfície *Serializable* per a poder convertir a cadenes o fluxos de bytes (*streams*) els objectes del contenidor i gravar-los o transmetre'ls.

Els elements dels contenidors es retornen com a iteradors. Aquests són compatibles amb els bucles *for-each* del JDK 1.5 i no alteren l'estat del contenidor.

El contenidor ordenat permet establir la precedència amb el criteri natural definit a la interfície *java.lang.Comparable* o bé utilitzar un comparador ad hoc, definit a la interfície *java.util.Comparator*. Facilita l'establiment d'una precondició a les operacions generadores, perquè si no hi ha comparador, les classes dels elements, rebuts com a paràmetres, han d'implementar *Comparable*.

4.2.2 Seqüències

En un segon nivell es troben les seqüències, que poden ser fitades, posicionals o combinacions d'elles, per permetre la màxima eficiència asimptòtica espacial i temporal. Seguidament es poden veure algunes combinacions del TAD Cua.

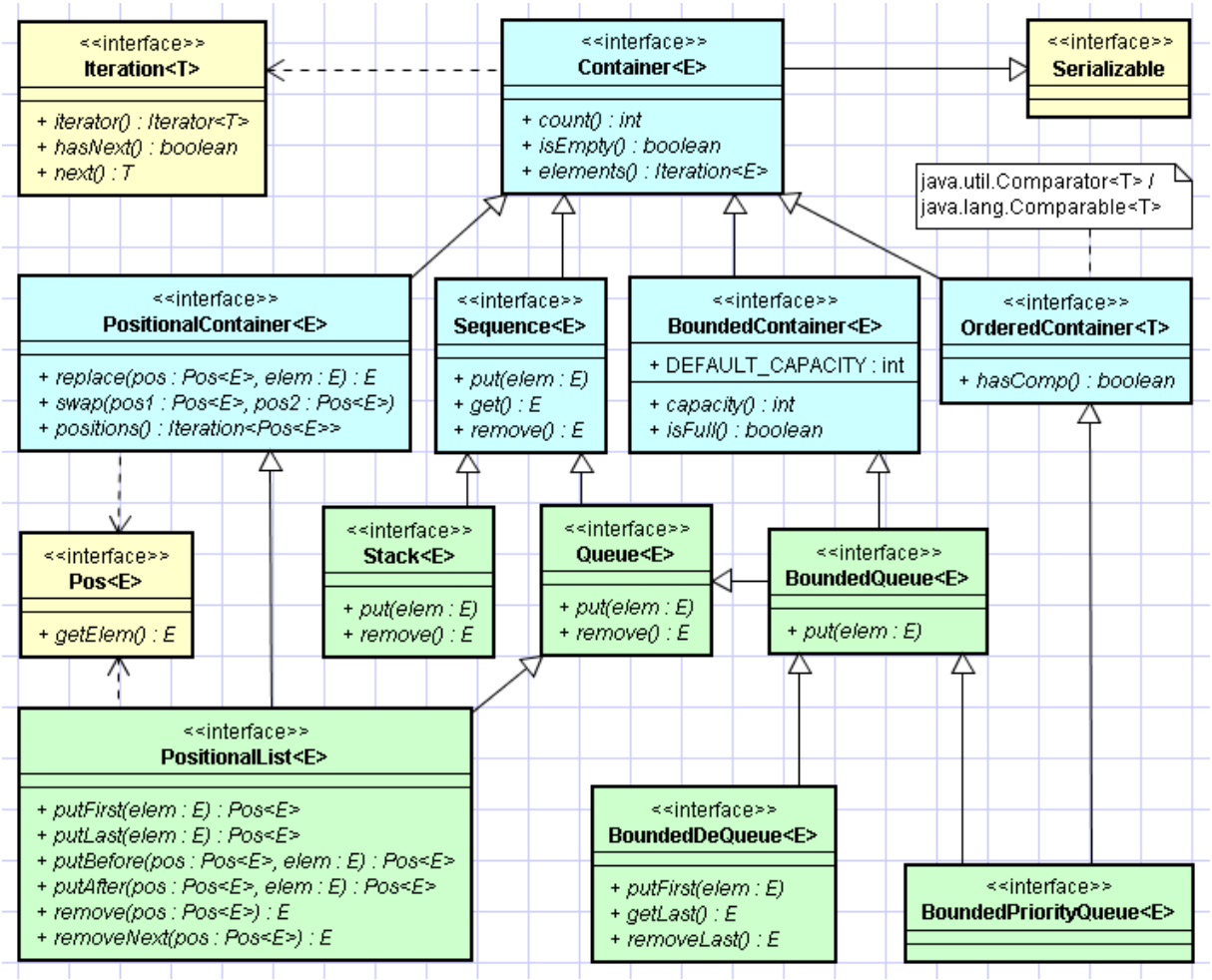


Figura 11

S'ha optat per una Llista-cua, que deriva de la Cua afegeix operacions posicionals a les pròpies de les seqüències. Una altra interfície defineix la Doble-Cua fitada, que permet implementacions de cost O(1) de totes les operacions (*LIFO* i *FIFO*).

4.2.3 Arbres

Una part molt interessant de la biblioteca és la jerarquia dels arbres, perquè permeten una munió d'implementacions. Ara bé, les dels arbres de cerca no han de permetre i, per tant, han de buidar de contingut les operacions posicionals modificadores heretades, perquè no garanteixen la consistència del TAD.

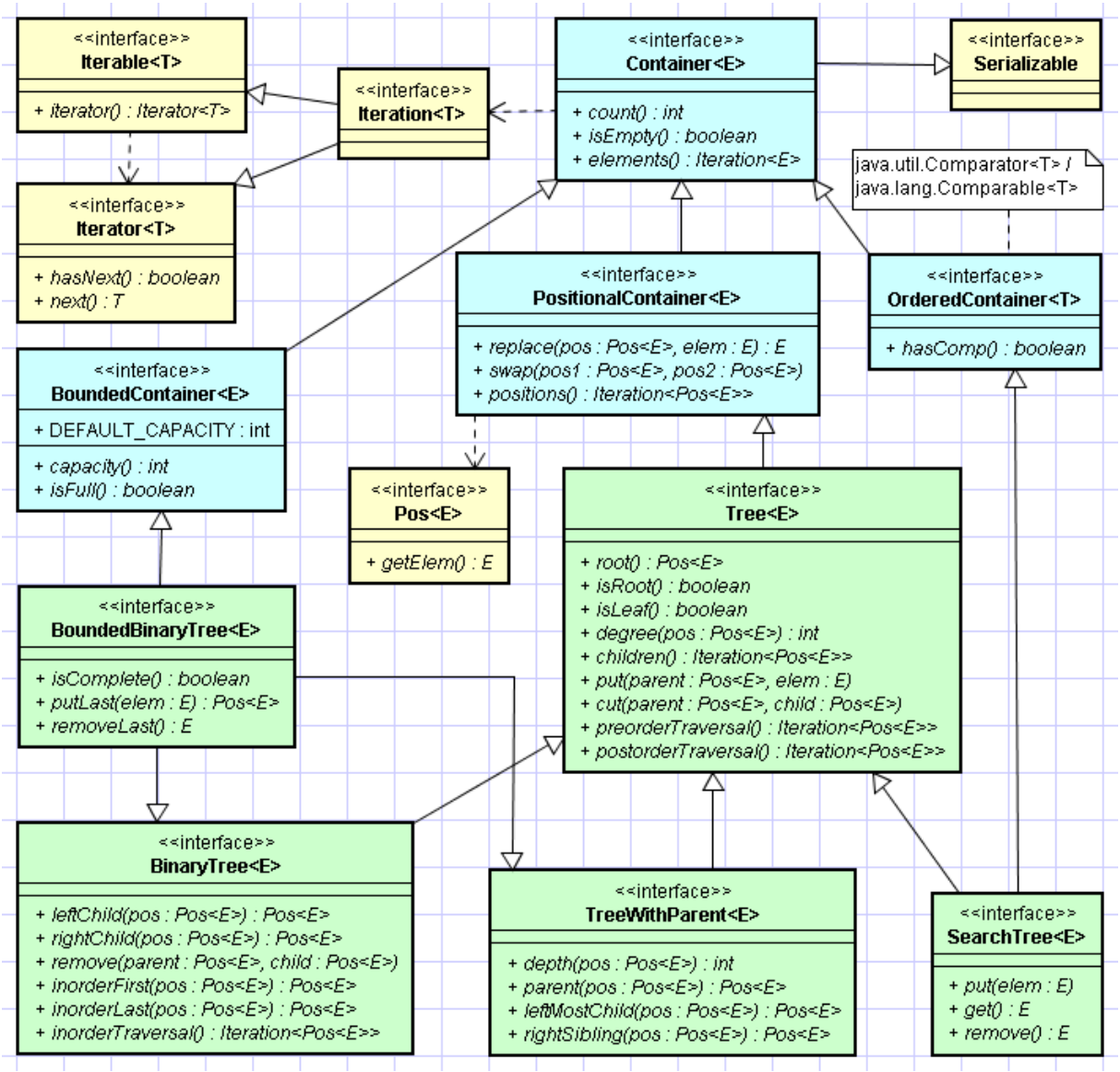


Figura 12

Els TADs disposen d'algunes operacions derivades que faciliten el DbC:

• *Tree.degree(node)*
Permet controlar el grau de sortida d'un node dels arbres n-aris

• *BinaryTree.inorderFirst(node)* i *BinaryTree.inorderLast(node)*
Permet comprovar la consistència de l'operació generadora `branch` als arbres binaris de cerca, perquè el darrer element inordre (el més gran) del subarbre esquerre ha de ser més petit que el contingut a la nova arrel i, anàlogament, el més petit del subarbre dret ha de ser més gran que la nova arrel.

• *BoundedBinaryTree.isComplete()*
Les operacions auxiliars d'afegir i de suprimir el darrer element d'un arbre binari, implementat sobre vector, només són vàlides quan l'arbre és complet. És a dir, si té l'altura mínima i totes les seves posicions buides estan al darrer nivell a la dreta.

4.2.4 Funcions, Sacs i Conjunts

Per una banda es defineixen les funcions i per l'altra els sacs i els conjunts. Els conjunts es presenten com una extensió dels sacs amb la restricció que el nombre d'ocurrències d'un element només pot ser zero o u.

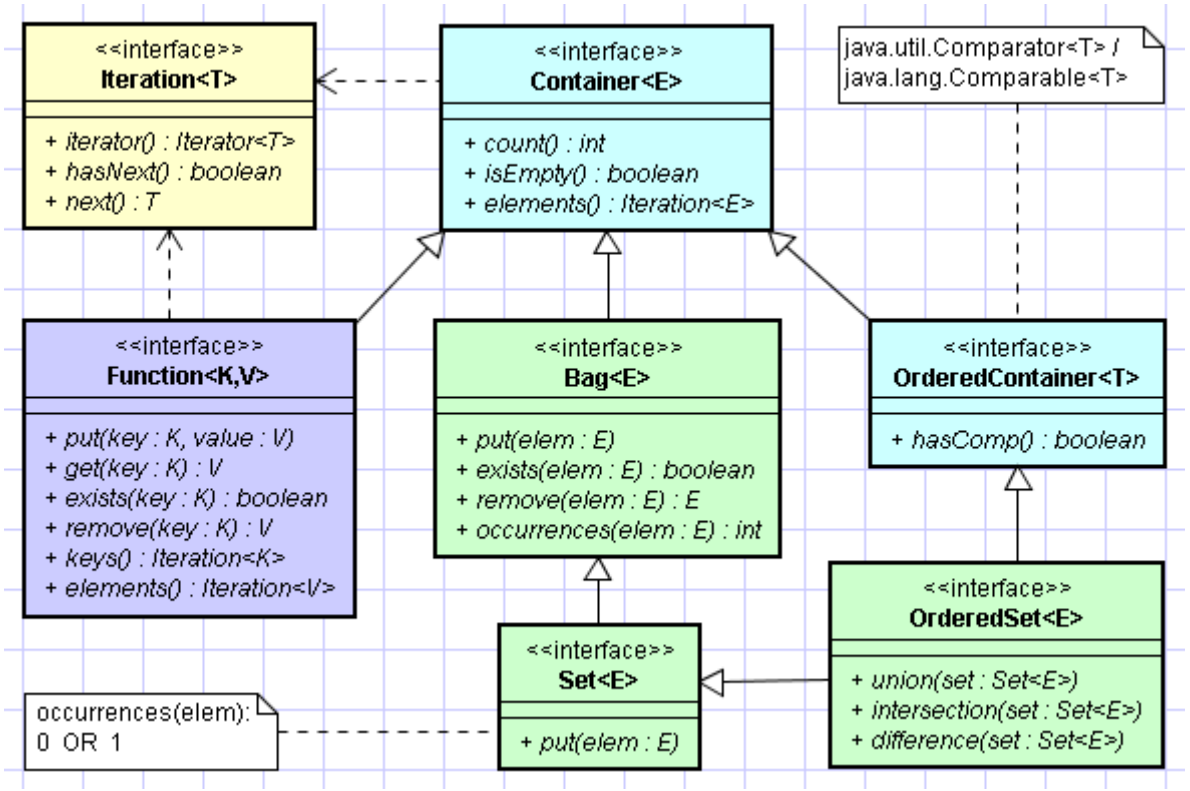


Figura 13

4.2.5 Relacions binàries i Grafs

Després dels sacs, els conjunts i les funcions, ve la jerarquia de les relacions binàries i els grafs. Atesa la proliferació d'extensions que permeten les relacions, s'ha optat per un parell de simplificacions:

- **Dominis no fixos**. Els vèrtexs <V> es donen d'alta o de baixa del seu domini, automàticament, quan formen part d'alguna interrelació o es queden isolats, respectivament
- **Relacions etiquetades**. Cada relació té un atribut <E> que, en ser de tipus genèric, permet nomenar-la (*String*) o assignar-li un pes (cas de les relacions valorades amb un *Number*) o, en general, emmagatzemar-hi un objecte qualsevol (*Object*) o bé deixar el seu valor per defecte (*null*)

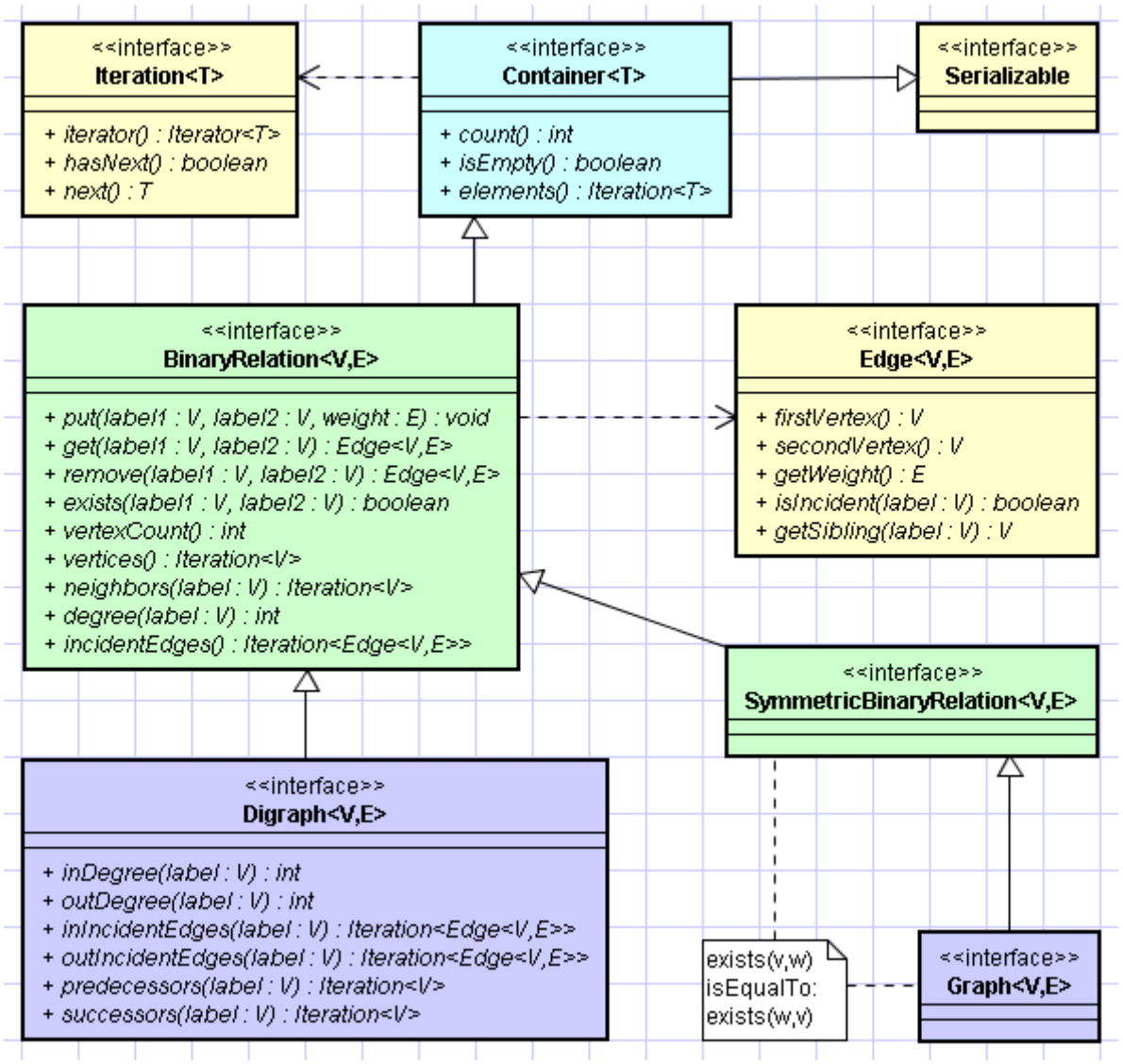


Figura 14

Les relacions simètriques deriven de les relacions generals i afegeixen una postcondició al mètode que estableix una relació, per assegurar la simetria. Així com en els digrafs la postcondició és que existeixi la nova relació, en els grafs s'exigeix que existeixi també la simètrica: `exists(v,w) == exists(w,v)`

4.3 JDK 1.5 (*Tiger*)

La nova implementació aprofita algunes de les millores tecnològiques introduïdes per la versió J2SE 5.0 de Sun, especialment:

- **Tipus genèrics**. Conceptualment i en relació a la solució anterior basada en la classe *Object*, permeten una millor representació dels tipus abstractes de dades, que es concreta en el moment de la utilització de la biblioteca. També estalvien moltes operacions i comprovacions de *casting* així com errors de *type mismatch*.
- **Bucles for-each**. En combinació amb la interfície **Iterator**, redueixen el volum del codi font i estalvien errors d' o*ut of bounds*.

- **Static import**. Permet, per exemple, la importació estàtica del màxim valor positiu dels enters, que es fa servir per retallar el bit de signe, a les funcions de dispersió, amb l'operador '&' (*bitwise AND*).

5. Disseny de l'aplicació

El disseny, emmarcat en un cicle de vida iteratiu i incremental, parteix d'una visió general de l'arquitectura, seguida d'una altra a nivell de paquets i finalment entra en el detall amb els diagrames estàtic i de seqüència.

5.1 Arquitectura

L'arquitectura d'una aplicació informàtica dóna una visió global del sistema. Serveix per identificar-ne els elements més importants i les seves relacions.

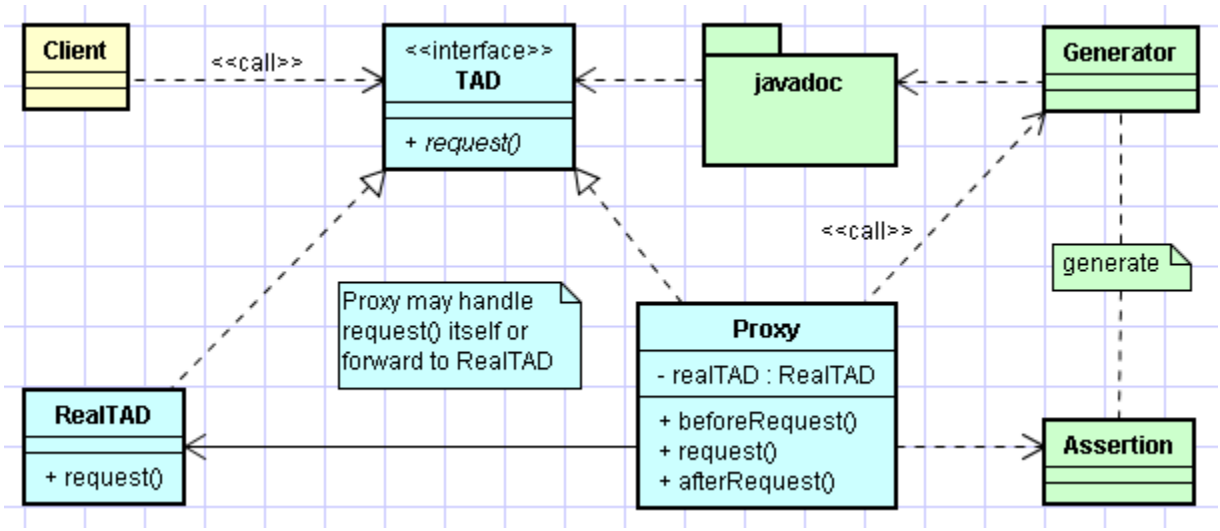


Figura 15

A grans trets, el client requereix els serveis d'un TAD representat per una jerarquia d'interfícies. El TAD recull les condicions del contracte, mitjançant invariants, pre i postcondicions, introduïdes com a comentaris del *javadoc* precedits de l'etiqueta corresponent (@inv, @pre i @post).

Un generador s'encarrega de transformar el contingut de les etiquetes, llegides pel *javadoc*, en assercions executables.

Entre la classe que implementa el TAD (RealTAD) i el client, s'interposa una classe intermediària (Proxy) que també l'implementa i, abans de servir la petició, executa una sèrie d'operacions consecutives:

- El Proxy captura la petició del client, mitjançant un *Invocation Handler*
- Abans d'executar el mètode de la petició, comprova les precondicions
- Executa el mètode de la petició, delegant en el RealTAD, i conserva el resultat
- Després d'executar el mètode, comprova les postcondicions i els invariants
- Si no s'ha violat cap asserció, retorna al client el resultat obtingut abans

5.2 Paquets

L'aplicació consisteix bàsicament en dos paquets (dbc i tads). El primer paquet, amb el programari que implementa el disseny per contracte, dependrà principalment dels paquets *tools* i *reflect* del JDK. El segon paquet, amb els TADs, depèn en certa mesura dels paquets *lang* i *util* del *runtime* de Java.

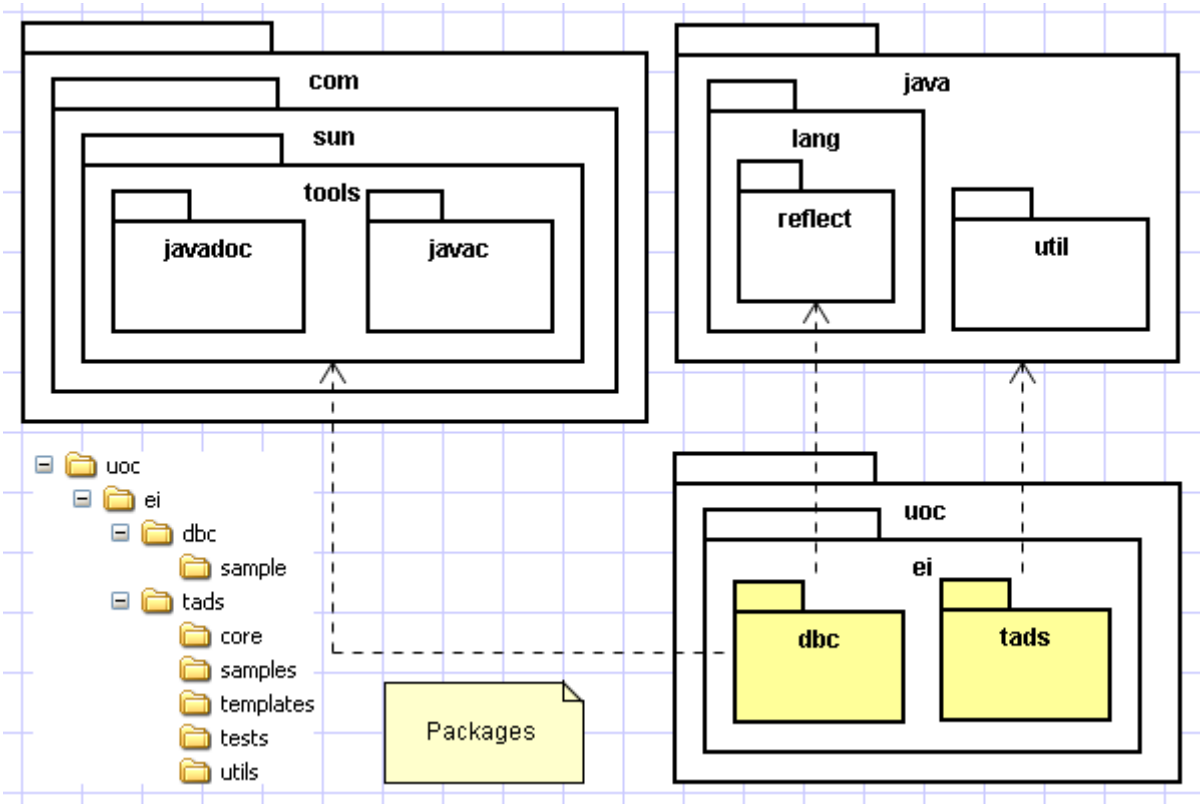


Figura 16

5.2.1 Paquet de disseny per contracte (dbc)

A grans trets, podem distingir les dependències del paquet dbc, d'acord amb les fases planificades, segons avança la implementació del projecte:

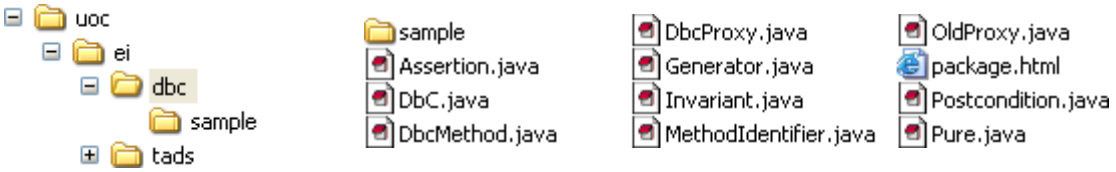


Figura 17

• Generació d'assercions
Es basa en la documentació generada pel *javadoc*

• Generació de codi
A més del paquet estàndard *java.lang* i de les utilitats de *java.util*, cal la compilació al vol (*on the fly*), amb el *javac*, del codi generat

• Inserció de codi
Principalment depèn del paquet *java.lang.reflect* (*Reflection*)

• Test
El paquet té una subcarpeta (*sample*) que presenta un exemple d'aplicació. Consisteix en l'extensió d'una interfície de la biblioteca, la seva implementació i un fitxer de prova de les assercions

5.2.2 Paquet de tipus abstractes de dades (tads)

El paquet tads consta de l'API amb les interfícies dels TADs i algunes subcarpetes que contenen fitxers agrupats per diferents conceptes:

Figura 18

• tads
Conjunt d'interfícies que representen les especificacions formals dels (TADs), i les estipulacions del contracte (DbC) introduïdes com a comentaris del *javadoc*

• core
Classes que implementen les interfícies

• doc-files
Imatges que utilitza el *javadoc* per il·lustrar la documentació generada

• samples
Exemples d'aplicació d'alguns mètodes de les principals classes

• templates
Classes abstractes que implementen codi comú dels seus descendents

• tests
Proves específiques, i proves unitàries mitjançant JUnit

• utils
Classes amb atributs constants i altres utilitats estàtiques

5.3 Diagrama de classes DbC

L'aplicació de DbC és vàlida per qualsevol classe que implementi una interfície, la qual pot estendre altres interfícies, també amb assercions declarades als comentaris del *javadoc*.

Paral·lelament al diagrama esquemàtic de l'arquitectura (veure 5.1), es poden distingir els tres mòduls de programari amb més detall:

• *Proxy* dinàmic (color blavós)
És creat mitjançant una operació constructora estàtica per la classe de prova. El *handler* l'activa cada vegada que l'esmentada classe de prova invoca un mètode de la interfície (veure 5.5)

• *Doclet* generador (color verdós)
És activat pel *proxy*. Mitjançant la informació que obté del *javadoc* p, construeix les assercions (veure 6), les introdueix a les classes d'interposició, les quals són generades i compilades al vol (veure 7).

• Assercions (color grogós)
La jerarquia d'assercions inclou el codi comú d'interposició (veure 6.5), que es heretat per les tres classes generades: una pels invariants i les altres dues pels mètodes amb pre o postcondicions, respectivament.

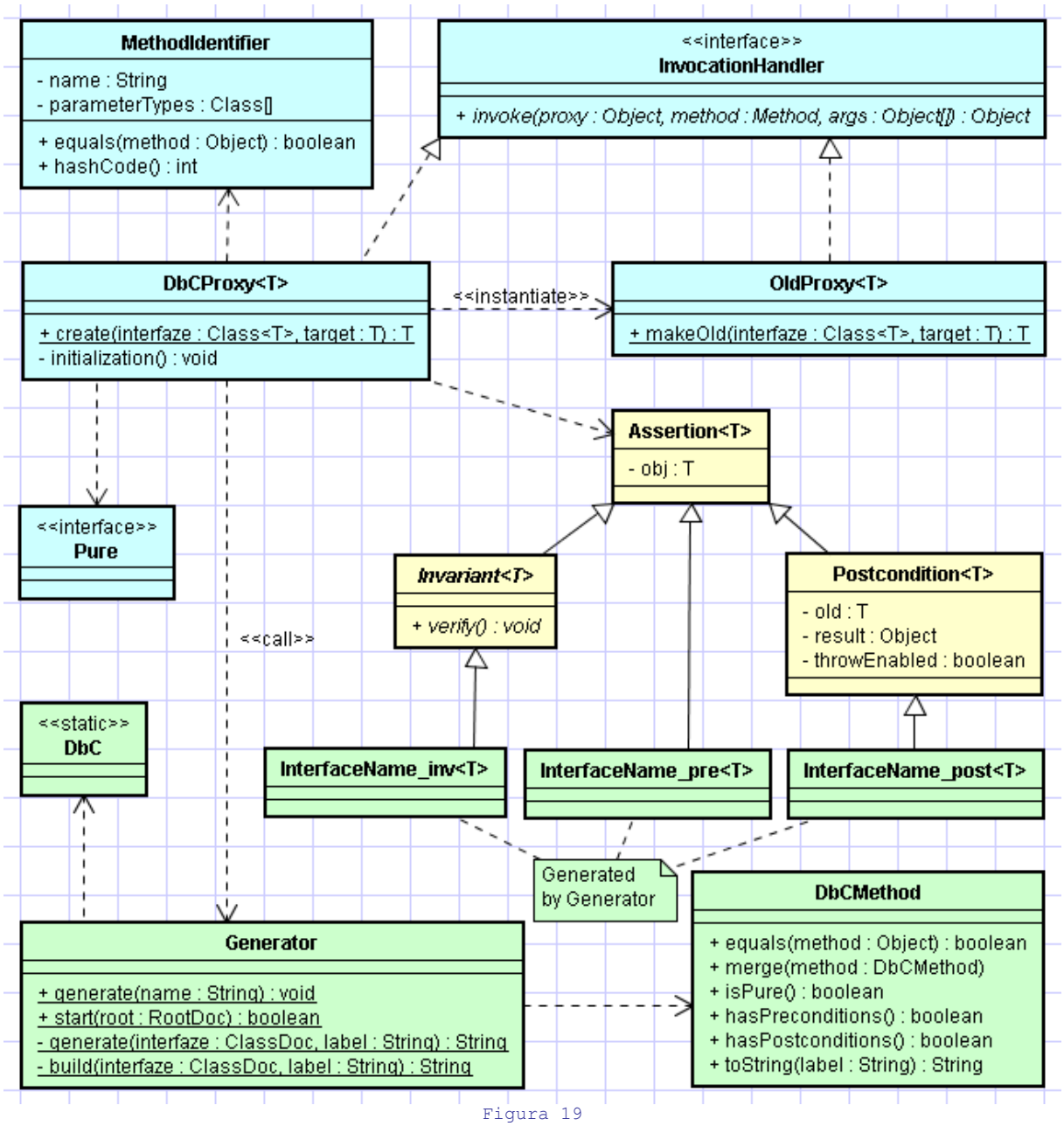


Figura 19

5.4 Prototip

Dins d'un cicle de vida iteratiu i incremental, s'ha optat per un sistema de programació exploratòria, que es diferencia del cicle de vida amb prototipatge, perquè el programari és real des del primer moment.

El punt de partida (veure apartat 1.3) de la implementació és un *doclet* (**Generator**), que recopila tota la informació proporcionada pel javadoc, una classe amb constants i altres utilitats estàtiques (**DbC**) i una extensió de la biblioteca de TADs.

Per aplicar el prototip de DbC, s'ha escollit una part complexa de la biblioteca de TADs i s'ha dissenyat l'extensió d'una interfície implementada per una classe (*BoundedStackPos* i ArrayStackPos). La complexitat ve donada per les següents característiques:

• Herència múltiple
BoundedStack hereta de dues interfícies

• Herència repetida
BoundedStack hereta dues vegades de *Container*

• Sobreescriptura
Stack sobreescriu el mètode *Sequence*.put(elem) per afegir una postcondició de l'empilar (@post $obj().get() == elem) i *BoundedStack* el torna a sobreescriure per afegir una precondició dels contenidors afitats (@pre !$obj().isFull())

• Sobrecàrrega
BoundedStackPos sobrecarrega el mètode *Sequence*.get() per afegir el paràmetre de l'índex dins de la seqüència

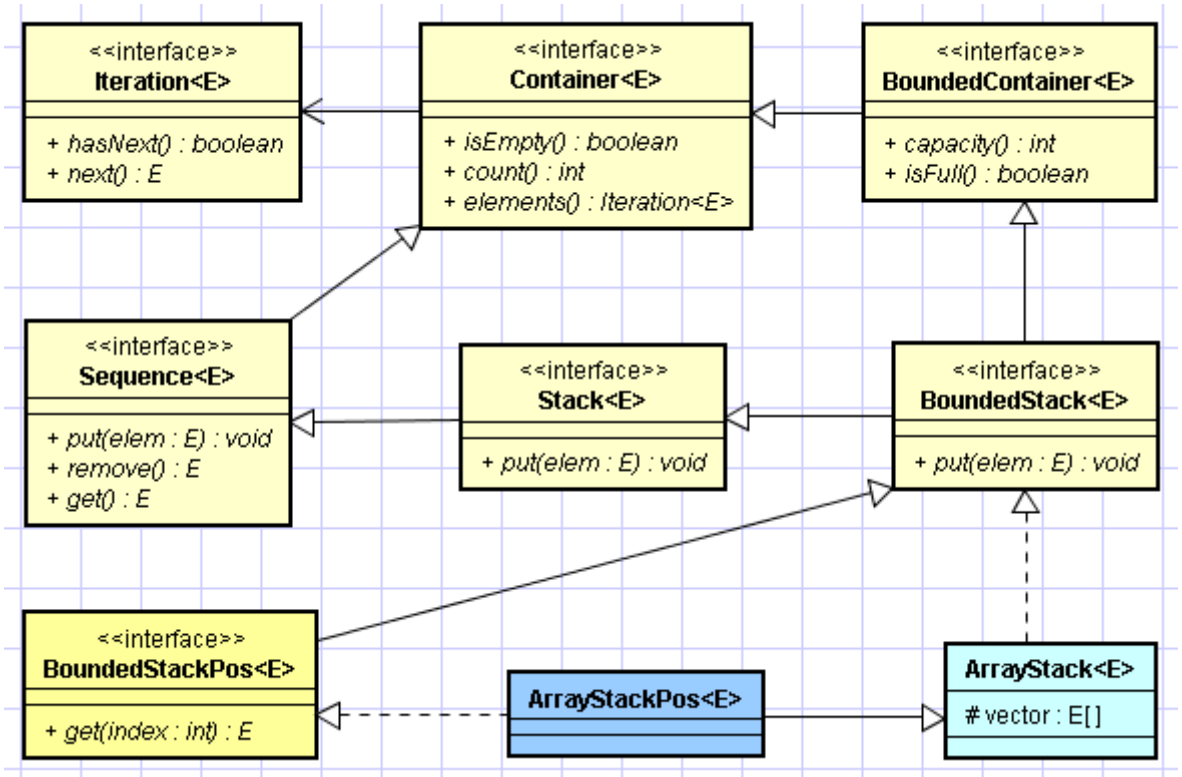


Figura 20

5.5 Diagrama de seqüència DbC

Diagrama de seqüència de creació de les classes centrat en els mètodes:

- `DbcProxy.create(Class<T> interfaze, T target)`
- `DbcProxy.inizialitation()`

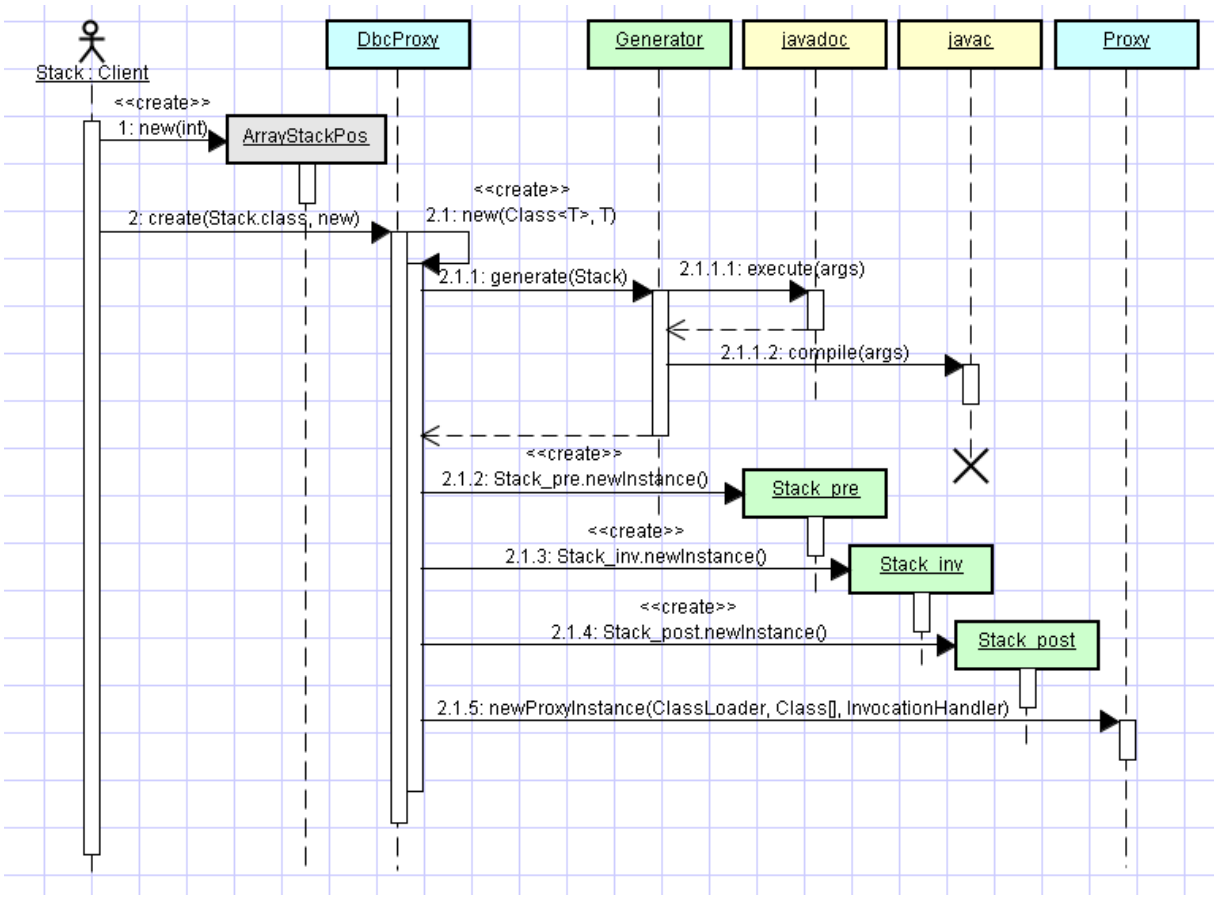


Figura 21

Diagrama de seqüència d'interposició de codi centrat en el mètode:

- `DbcProxy.invoke(Object proxy, Method method, Object[] args)`
- `Interface java.lang.reflect.InvocationHandler`

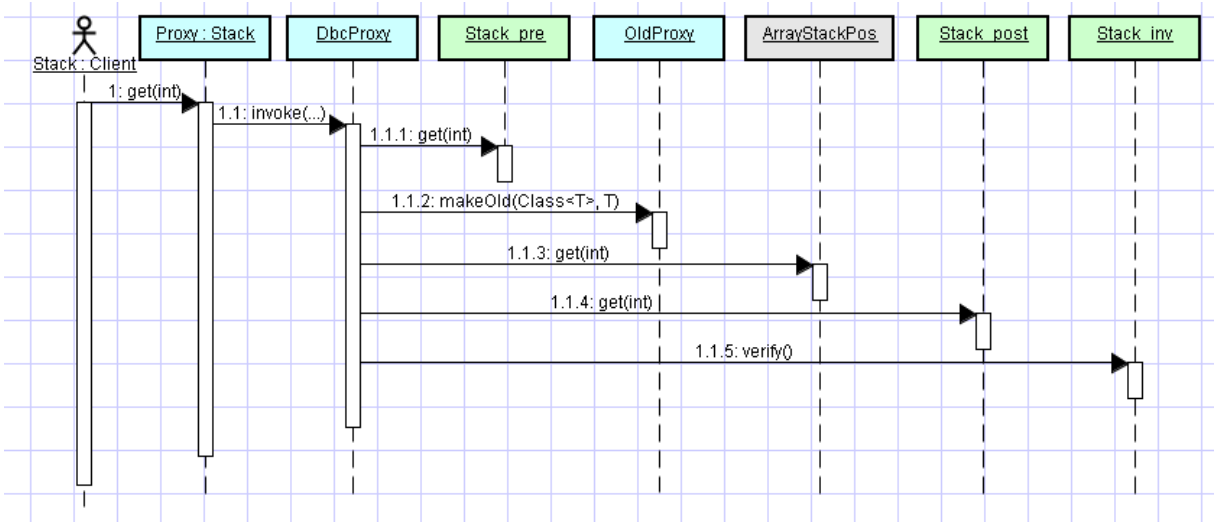


Figura 22

6. Generació d'assercions

Les assercions estan introduïdes, amb la seva etiqueta (@pre, @inv, @post), als comentaris del *javadoc*. Els invariants a les capçaleres de les interfícies i les pre i postcondicions a les de llurs mètodes:

```
21  *
22  * @inv  $obj().count() >= 0
23  */
24 public interface Container<Elem>  extends java.io.Serializable
25 {
26    /**
27     * Comprova si el contenidor està buit.
28     * @return  cert si el contenidor està buit; altrament retorna fals
29     *
30     * @pure  la implementació del mètode no altera l'estat del contenidor
31     * @post  $return() == ($obj().count() == 0)
32     */
33    public boolean isEmpty();
```

Figura 23

6.1 Invariants (@inv)

L'algorisme de la classe d'utilitats estàtiques **DbC** primer cerca les superclasses de la classe de prova:

```
DbC.superClassesOf(uoc.ei.dbc.test.ArrayStackPos):
[uoc.ei.dbc.test.ArrayStackPos, uoc.ei.tads.ArrayStack]
```

Figura 24

després obté un vector amb totes les interfícies implementades per la classe de prova, les seves superclasses i les antecessores d'aquelles, controlant l'herència múltiple i l'herència repetida:

```
DbC.superInterfacesOf(uoc.ei.dbc.test.ArrayStackPos):
[uoc.ei.tads.Sequence, uoc.ei.dbc.test.BoundedStackPos, uoc.ei.tads.Stack, uoc.
ei.tads.BoundedStack, uoc.ei.tads.BoundedContainer, uoc.ei.tads.Container]
```

Figura 25

finalment l'algorisme **DbC.buildAssertions** recull els invariants declarades amb l'etiqueta @inv als comentaris del *javadoc* de totes les interfícies implementades per la classe de prova i els hi dóna format d'asserció (operador *assert*):

```
    assert _obj().count() <= _obj().capacity():
    "Invariant violated:" + DbC.LS +
    "$obj().count() <= $obj().capacity()" + DbC.LS +
    "declared at" + DbC.LS +
    "uoc.ei.tads.BoundedContainer:16" + DbC.LS;
```

```
    assert _obj().count() >= 0:
    "Invariant violated:" + DbC.LS +
    "$obj().count() >= 0" + DbC.LS +
    "declared at" + DbC.LS +
    "uoc.ei.tads.Container:24" + DbC.LS;
```

Figura 26

6.2 Precondicions i postcondicions (@pre i @post)

L'algorisme **DbC.buildMethods** cerca els mètodes declarats a totes les interfícies implementades per la classe de prova **ArrayStackPos**, recull les precondicions i les postcondicions declarades, amb l'etiqueta @pre i @post, als comentaris del *javadoc* dels mètodes, fins i tot dels sobreescrits, i els hi dóna format d'asserció (operador *assert*), com en el cas dels invariants:

```
public void put(Object elem)  throws Throwable
{
    assert !_obj().isFull():
    "Precondition violated:" + DbC.LS +
    "!$obj().isFull()" + DbC.LS +
    "declared at" + DbC.LS +
    "uoc.ei.tads.BoundedStack.put(Elem):21" + DbC.LS;
}
```

```
public void put(Object elem)  throws Throwable
{
    assert _obj().count() == _old().count() + 1:
    "Postcondition violated:" + DbC.LS +
    "$obj().count() == $old().count() + 1" + DbC.LS +
    "declared at" + DbC.LS +
    "uoc.ei.tads.Sequence.put(Elem):23" + DbC.LS;

    assert _obj().get() == elem:
    "Postcondition violated:" + DbC.LS +
    "$obj().get() == elem" + DbC.LS +
    "declared at" + DbC.LS +
    "uoc.ei.tads.Stack.put(Elem):20" + DbC.LS;
}
```

Figura 27

6.3 Violació d'una asserció (*assert*)

Des de la classe de prova es simula la interposició d'una asserció (precondició) i s'intenta accedir a una posició fora del rang del vector, per comprovar el format del missatge d'error generat per la precondició.

Sense DbC saltaria un *java.lang.IndexOutOfBoundsException*

```
Exception in thread "main" java.lang.AssertionError: Precondition violated:
index >= 0 && index < $obj().count()
declared at
uoc.ei.dbc.test.BoundedStackPos.get(int):28

        at uoc.ei.dbc.test.BoundedStackPos_pre.get(BoundedStackPos_pre.java:78)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
        at java.lang.reflect.Method.invoke(Unknown Source)
        at uoc.ei.dbc.DbcProxy.invoke(DbcProxy.java:155)
        at $Proxy0.get(Unknown Source)
        at uoc.ei.dbc.test.ArrayStackPos.main(ArrayStackPos.java:52)
```

Figura 28

6.4 Generació de la documentació (*javadoc*)

Method Detail

get

`Elem` **`get`**`(int index)`

Accessor de lectura de l'element de la pila que ocupa la posició indicada. Sobrecarrega el mètode Sequence.get().

Parameters:

`index` - posició de l'element dins de la seqüència

Returns:

element que ocupa la posició índex

Pure:

la implementació del mètode no altera l'estat del contenidor

Precondition:

!$obj().isEmpty(), index >= 0 && index < $obj().count()

Postcondition:

$obj().count() == $old().count()

Figura 29

Com es pot observar, les assercions són compatibles amb el *doclet* estàndard i, per tant, queden reflectides a l'API de la biblioteca de TADs.

6.5 Detall de la generació d'assercions

S'ha de generar codi compilable, dins del seu àmbit, per les quatre funcions auxiliars. Pel que fa a les dues primeres (`$obj` i `$old`), la transformació és quasi immediata. En canvi, les altres (`$oldparam` i `$return`) requereixen un procés més complex.

També cal generar una estructura condicional derivada de l'operador `$implies`

6.5.1 $obj() i $old()

Atès que han de retornar, respectivament, l'objecte (instància de la classe) en el seu estat actual o anterior a l'execució d'un mètode, només cal canviar el caràcter '$' per un guió baix '_' per cridar la respectiva funció generada que fa el *casting*. Per exemple, en el cas d'un arbre binari:

```
public BinaryTree _obj()
{
   return  (BinaryTree)getObj();
}

public BinaryTree _old()
{
   return  (BinaryTree)getOld();
}
```

```
assert _obj().count() == _old().count() - 1:
"Postcondition violated:" + DbC.LS +
"$obj().count() == $old().count() - 1" + DbC.LS +
"declared at" + DbC.LS +
"uoc.ei.tads.BinaryTree.remove(uoc.ei.tads.Position, uoc.ei.tads.Position):27" + DbC.LS;
```

Figura 30

6.5.2 $oldparam(p.x)

Atès que ha de retornar el valor `x` obtingut, a partir d'un argument, abans de l'execució d'un mètode, genera una funció complementària que desa i retorna el valor obtingut del paràmetre la primera vegada que s'ha executat el mètode. Per evitar la indeterminació que pot suposar l'existència d'altres arguments homònims, genera un identificador únic dins del seu àmbit:

- `signaturaMètode + ':' + nomParàmetre + '.' + funció`

```
assert pos1.getElem() == _oldarg("swap(uoc.ei.tads.Position pos1,
       uoc.ei.tads.Position pos2):pos2.getElem()", pos2.getElem()):
"Postcondition violated:" + DbC.LS +
"pos1.getElem() == $oldparam(pos2.getElem())" + DbC.LS +
"declared at" + DbC.LS +
"uoc.ei.tads.PositionalContainer.swap(uoc.ei.tads.Position,
       uoc.ei.tads.Position):44" + DbC.LS;
```

Figura 31

6.5.3 $return()

Atès que només cal retornar el resultat, un cop executat un mètode, el principal problema es presenta quan aquest retorna un tipus de dades primitiu, perquè el compilador utilitza un embolcall (*wrapper*). En aquests casos és necessari generar codi complementari d'emmotllament (*casting*) per desfer-lo:

```
assert ((Boolean)result()).booleanValue() == (_obj().count() == 0):
"Postcondition violated:" + DbC.LS +
"$return() == ($obj().count() == 0)" + DbC.LS +
"declared at" + DbC.LS +
"uoc.ei.tads.Container.isEmpty():33" + DbC.LS;
```

Figura 32

6.5.4 $implies

L'aplicació genera una estructura condicional, de tal forma que, si es compleix la premissa, aleshores s'avalua l'asserció. Per exemple, en el cas d'un arbre:

```
if (parent == null)
assert _obj().isRoot(((uoc.ei.tads.Position)result())):
"Postcondition violated:" + DbC.LS +
"parent == null $implies $obj().isRoot($return())" + DbC.LS +
"declared at" + DbC.LS +
"uoc.ei.tads.Tree.put(uoc.ei.tads.Position, Elem):85" + DbC.LS;
```

Figura 33

7. Generació de codi executable

La generació de codi està a càrrec de la classe **Generator**, que disposa de quatre mètodes (veure punt 5.3). El primer recull la informació mitjançant les eines del *javadoc*, el segon gestiona la informació rebuda, el tercer construeix els fitxers font i el quart els grava a disc compilant-los al vol (*on the fly*).

7.1 Recull d'informació

El punt d'entrada de la classe **Generator** és el mètode **generate(String)** que rep com a paràmetre el nom qualificat (nom precedit del *package*) de la interfície considerada, la qual pot estendre múltiples interfícies.

Aquest mètode recull les variables, que defineixen el camí cap als fitxers del codi font i els paquets (carpetes) que els contenen, del fitxer **dbc.properties**.

A continuació executa el mètode **javadoc.Main.execute(String[])** amb els arguments recollits abans. Aquest mètode, un cop acabada l'estructuració de la documentació, activa el mètode **start(RootDoc)**.

7.2 Gestió de la classe generadora

El punt d'entrada a la documentació estructurada és el mètode **start(RootDoc)** activat directament pel *javadoc*. El paràmetre representa l'arrel de tota la informació recollida per una execució del *javadoc*. Des d'ella es pot treure tota la informació estructurada i els paràmetres passats per l'usuari (*packages*, classes i opcions).

El mètode extreu, de l'esmentada arrel, el document amb la informació de la interfície objecte del disseny per contracte. Després activa tres vegades els mètodes de generació del codi font de les tres classes encarregades de verificar les assercions (precondicions, invariants i postcondicions).

7.3 Construcció del codi font amb les assercions

El mètode **build(ClassDoc,String)**, amb la col·laboració de les classes **DbC** i **DbcMethod**, construeix el codi font amb els mètodes que contenen assercions i d'altres d'auxiliars. Com a mostra, s'inclou un fragment de la classe que gestiona les precondicions:

```
package uoc.ei.dbc.test;
import  uoc.ei.dbc.*;

/**
 * Classe generada per Generator.generate(...).
 *
 * @author   Jordi Cabot Sagrera (professor),
 *           Esteve Mariné Gallisà (alumne),
 *           Projecte final de carrera (PFC),
 *           Universitat Oberta de Catalunya (UOC)
 * @version  Curs 2005/06-2
 */
public class BoundedStackPos_pre  extends Assertion
{
   public BoundedStackPos _obj()
   {
      return  (BoundedStackPos)getObj();
   }

   /**
    * Method declared at:
    * C:/UOC/20062006/PFC/DbC.06/src/uoc/ei/tads/Sequence.java:24
    * C:/UOC/20062006/PFC/DbC.06/src/uoc/ei/tads/Stack.java:21
    * C:/UOC/20062006/PFC/DbC.06/src/uoc/ei/tads/BoundedStack.java:22
    */
   public void put(Object elem)  throws Throwable
   {
      assert !_obj().isFull():
      "Precondition violated:" + DbC.LS +
      "!$obj().isFull()" + DbC.LS +
      "declared at" + DbC.LS +
      "uoc.ei.tads.BoundedStack.put(Elem):22" + DbC.LS;
   }

   /**
    * Method declared at:
    * C:/UOC/20062006/PFC/DbC.06/src/uoc/ei/dbc/test/BoundedStackPos.java:29
    */
   @Pure
   public void get(int index)  throws Throwable
   {
      assert !_obj().isEmpty():
      "Precondition violated:" + DbC.LS +
      "!$obj().isEmpty()" + DbC.LS +
      "declared at" + DbC.LS +
      "uoc.ei.dbc.test.BoundedStackPos.get(int):29" + DbC.LS;

      assert index >= 0 && index < _obj().count():
      "Precondition violated:" + DbC.LS +
      "index >= 0 && index < $obj().count()" + DbC.LS +
      "declared at" + DbC.LS +
      "uoc.ei.dbc.test.BoundedStackPos.get(int):29" + DbC.LS;
   }
}
```

Figura 34

7.4 Generació de les classes

El mètode **generate(ClassDoc,String)** grava temporalment els fitxers amb el codi font de les classes (.java) i els compila al vol (*on the fly*) obtenint els fitxers precompilats (*bytecode*):

- `interfaceName + _ + DbC.PRE  + .class`
- `interfaceName + _ + DbC.INV  + .class`
- `interfaceName + _ + DbC.POST + .class`

dins de la carpeta indicada per la propietat **build.dir**

8. Interposició de codi DbC

Com s'ha vist als apartats 5.3 i 5.5, la primera acció és crear un Proxy que embolcalla la classe de prova i implementa la interfície o interfícies interessades.

```
051    public static <T> T create(Class<T> interfaze, T target)
052    {
053       return  (T) Proxy.newProxyInstance
054       (
055          target.getClass().getClassLoader(),
056          new Class<?>[] { interfaze },
057          new DbcProxy<T>(interfaze, target)
058       );
059    }
```

Figura 35

8.1 Dynamic Proxy

L'operació constructora, que implementa un `InvocationHandler`, activa la generació de codi i inicialitza totes les classes derivades d'`Assertion`.

```
public interface java.lang.reflect.InvocationHandler

InvocationHandler is the interface implemented by the invocation handler of a proxy instance.
Each proxy instance has an associated invocation handler. When a method is invoked on a proxy
instance, the method invocation is encoded and dispatched to the invoke method of its
invocation handler.
```

Aleshores, espera que el *handler* li derivi les crides, als mètodes de la interfície, que fa la classe de prova.

```
153    public Object invoke(Object proxy, Method method, Object[] args)
154       throws Throwable
155    {
156       T old;  Object result = null;
157       Method preMethod = null, postMethod = null;
158
159       try
160       {
161          // comprova les precondicions, si n'hi ha
162          preMethod = preMethods.get(new MethodIdentifier(method));
163          if (preMethod != null)  preMethod.invoke(pre, args);
164          // inicialitza el resultat del mètode de la postcondició
165          postMethod = postMethods.get(new MethodIdentifier(method));
166          if (postMethod != null)
167          {
168             // crea una còpia de l'objecte abans d'executar el mètode real
169             old = OldProxy.makeOld(interfaze, target);
170             post.setOld(old);
171             // inicialitza resultats de les funcions old() i oldparam(p.x)
172             post.setThrowStatus(Boolean.FALSE);
173             try { postMethod.invoke(post, args); }
174             catch (Throwable ignored) { }
175             finally { post.setThrowStatus(Boolean.TRUE); }
176          }
177          // fa una còpia del resultat retornat pel mètode real
178          result = method.invoke(target, args);
179          // comprova les postcondicions, si n'hi ha
180          if (postMethod != null)
181          {
182             post.setResult(result);
183             postMethod.invoke(post, args);
184          }
185          // comprova els invariants, si cal
186          if (method.getAnnotation(Pure.class) == null)  inv.verify();
187       }
188       catch (InvocationTargetException ite) { throw ite.getCause(); }
189
190       return  result;
191    }
```

Figura 36

8.2 Optimització de les verificacions

Si bé el *proxy* dinàmic implementa la mateixa interfície que la classe de prova i, per tant, s'interposa quan aquesta crida qualsevol dels seus mètodes, només es genera i s'executa codi d'interposició quan és estrictament necessari.

8.2.1 Pre i postcondicions

Sens perjudici que les pre i postcondicions, si n'hi ha, s'han de comprovar, respectivament, a l'entrada i sortida de tots els mètodes, quan un mètode de la classe de prova no té algun dels dos grups d'assercions, no es genera ni s'executa el codi d'interposició que li correspon.

```
159       try
160       {
161          // comprova les precondicions, si n'hi ha
162          preMethod = preMethods.get(new MethodIdentifier(method));
163          if (preMethod != null)  preMethod.invoke(pre, args);
```

Figura 37

Quan un mètode no té cap postcondició s'estalvia igualment la generació i l'execució de codi d'interposició, així com la inicialització dels valors de les funcions auxiliars `old()` i `oldparam(p.x)`

```
165          postMethod = postMethods.get(new MethodIdentifier(method));
166          if (postMethod != null)
167          {
168             // crea una còpia de l'objecte abans d'executar el mètode real
169             old = OldProxy.makeOld(interfaze, target);
170             post.setOld(old);
171             // inicialitza resultats de les funcions old() i oldparam(p.x)
172             post.setThrowStatus(Boolean.FALSE);
173             try { postMethod.invoke(post, args); }
174             catch (Throwable ignored) { }
175             finally { post.setThrowStatus(Boolean.TRUE); }
176          }
```

Figura 38

8.2.2 Invariants

Com s'indica a la introducció, des d'un enfocament teòric, els invariants s'han de comprovar a l'entrada i sortida de tots els mètodes de la classe:

- `{@pre} Constructor {@post & @inv}`
- `{@pre & @inv} Method {@post & @inv}`

Ara bé, si es parteix d'un estat consistent, es pot estalviar la comprovació dels invariants a l'entrada dels mètodes. Tot i amb això, és poc eficient fer la comprovació a la sortida de tots els mètodes d'una classe, quan n'hi ha molts que no afecten, per a res, l'estat de l'objecte.

Atès que l'estat d'una classe s'altera quan es modifica algun dels seus atributs, en una primera versió, es va pensar en utilitzar un *framework* lleuger, com el ObjectWeb ASM, que facilita *Visitors* a baix nivell, però anava en contra del requeriment de no incorporar llibreries externes. Posteriorment, per aconseguir l'accés als atributs que tenen visibilitat *friend*, *protected* o *private* es va fer un nou fitxer de permisos (**`dbc.policy`**) per afegir una autorització general durant la fase de proves:

```
grant {
       permission java.security.AllPermission;
};
```

```
java -ea -Djava.security.manager -Djava.security.policy=dbc.policy ...
```

Figura 39

La idea només dóna bon resultat quan hi ha pocs atributs i molts invariants. Per una altra banda, és conceptualment incorrecte accedir directament als atributs sense visibilitat.

Un cop descartades les idees anteriors, per optimitzar la comprovació d'invariants s'ha creat una interfície buida que permet traspassar, al codi generat, la indicació de l'etiqueta @pure:

```java
01 package uoc.ei.dbc;
02 import  java.lang.annotation.Retention;
03 import  java.lang.annotation.RetentionPolicy;
04
05 /**
06  * Interfície per anotar els mètodes purs, els que no alteren l'estat del
07  * contenidor (tag = @pure). L'anotació es grava al fitxer precompilat i
08  * pot ser usada per la màquina virtual en temps d'execució.
09  *
10  * @author   Jordi Cabot Sagrera (professor),
11  *           Esteve Mariné Gallisà (alumne),
12  *           Projecte final de carrera (PFC),
13  *           Universitat Oberta de Catalunya (UOC)
14  * @version  Curs 2005/06-2
15  */
16 @Retention(RetentionPolicy.RUNTIME)
17 public @interface Pure { }
```

Figura 40

Mitjançant una de les millores del JDK 1.5, en temps d'execució es pot esbrinar si un mètode porta l'anotació @Pure i, per tant, estalviar la comprovació d'invariants a la sortida, com es pot veure en el següent fragment de codi:

```java
185          // comprova els invariants, si cal
186          if (method.getAnnotation(Pure.class) == null)  inv.verify();
187       }
188       catch (InvocationTargetException ite) { throw ite.getCause(); }
```

Figura 41

Una senzilla consulta permet estalviar, en la majoria de casos, la verificació d'invariants.

9. Joc de proves

La primera intenció era fer un joc de proves unitari, mitjançant *JUnit*, però s'ha comprovat que aquesta utilitat presenta incompatibilitats, perquè fa servir una tecnologia, basada en *Reflection* i llançament d'*AssertionError*, similar a la utilitzada pel codi d'interposició de l'aplicació de DbC.

Aleshores, s'han implementat quatre tipus de proves, dues basades en el *JUnit*, unes altres en el *debugging* per pantalla i les demés amb DbC.

9.1 Proves unitàries amb JUnit (sense DbC)

S'han fet les proves amb la versió 4.1, que permet anotacions. Per exemple:

```java
@RunWith(Suite.class)
@SuiteClasses
({
   // ...;
})
```

```java
@Before
public void setUp()
{
   // ...;
}
```

```java
@Test
public void equations()
{
   // ...;
}
```

Figura 42

9.1.1 Verificació de l'especificació algebraica

Per poder implementar equacions niuades, que comparen l'estat del TAD abans i després de les operacions, es treballa amb dos TADs i s'utilitza el mètode *equals* dels iteradors que retorna cert, si l'objecte rebut és no nul, de la mateixa classe i, fent un recorregut paral·lel, els elements obtinguts són iguals dos a dos, d'acord amb el seu operador *equals*; fals, altrament. Això permet fer una comparació d'equivalència, per exemple, entre dos TADs de la classe **DeQueue**:

- `assertEquals(q1.elements(), q2.elements());`

Per una altra banda, la doble cua permet comprovar indirectament les especificacions de la seva superclasse cua.

```
--/->     altera el contingut del TAD
---->     no altera el contingut del TAD
cursiva   operació derivada

univers DeQueue(Element) especificació
  usa Natural, Boolean
  gènere dequeue
  generadores
    new: ----> dequeue
    put: dequeue element --/-> dequeue
    putFirst: dequeue element --/-> dequeue
  operacions
    remove: dequeue --/-> dequeue
    removeLast: dequeue --/-> dequeue
    get: dequeue ----> element
    getLast: dequeue ----> element
    count: dequeue ----> natural
    capacity: dequeue ----> natural
    isEmpty: dequeue ----> boolean
    isFull: dequeue ----> boolean
  errors  q: dequeue
    put(q): isFull(q)
    putFirst(q): isFull(q)
    remove(q): isEmpty(q)
    removeLast(q): isEmpty(q)
    get(q): isEmpty(q)
    getLast(q): isEmpty(q)
  equacions  q: dequeue, e: element
    remove(putFirst(q,e)) = q
    removeLast(put(q,e)) = q
    get(putFirst(q,e)) = e
    getLast(put(q,e)) = e
    isEmpty(q)      => get(put(q,e)) = e
    NOT isEmpty(q) => get(put(q,e)) = get(q)
    NOT isEmpty(q) => remove(put(q,e)) =
                      put(remove(q),e)
    count(new) = 0
    count(put(q,e)) = suc(count(q))
    count(putFirst(q,e)) = suc(count(q))
    isEmpty(q) = isZero((count(q))
    isFull(q) = (count(q) = capacity(q))
funivers
```

```
/** Inicialitza atributs abans de cada test. */
@Before
public void setUp()
{
   q1 = new ArrayDeQueue<String>(2);
   q2 = new ArrayDeQueue<String>(2);
   one = "1";   two = "2";
}

/** Comprova les equacions. */
@Test
public void equations()
{
   int n = q1.count();
   // count(new) = 0
   assertEquals(n, 0);
   // isEmpty(q) = isZero(count(q))
   assertTrue(q1.isEmpty());
   q1.put(one);
   // isEmpty(put(q,e)) = false
   assertFalse(q1.isEmpty());
   // isEmpty(q) => get(put(q,e)) = e
   assertEquals(q1.get(), one);
   // count(put(q,e)) = suc(count(q))
   assertEquals(q1.count(), n+1);
   // NOT isEmpty(q) => get(put(q,e)) = get(q)
   q2.put(one);
   q1.put(two);
   assertEquals(q1.get(), q2.get());
   // NOT isEmpty(q) => remove(put(q,e)) =
   //                   put(remove(q),e)
   q1.remove();
   q2.remove();   q2.put(two);
   assertEquals(q1.elements(), q2.elements());
   q2.put(one);
   // isFull(q) = (count(q) = capacity(q))
   assertTrue(q2.isFull());
   assertEquals(q2.count(), q2.capacity());
   // removeLast(put(q,e)) = q
   assertEquals(q2.removeLast(), one);
   assertEquals(q1.elements(), q2.elements());
   // remove(putFirst(q,e)) = q
   q2.putFirst(one);
   assertEquals(q2.remove(), one);
   assertEquals(q1.elements(), q2.elements());
}
```

Figura 43

Com es pot veure a la taula precedent, es poden resoldre totes les equacions, llevat de la comprovació de les situacions d'error, perquè les precondicions estan a nivell de DbC (com a comentaris del *javadoc* amb l'etiqueta `@pre`) i el *JUnit* opera només a la part del codi executable.

9.1.2 Proves singulars

El *JUnit* inicialitza el contenidor abans de provar cadascun dels mètodes. A continuació es pot veure una part de les proves en una llista doblement enllaçada que hereta de la Llista-cua.

```
public class DoublyLinkedListTest
{
   /** Interfície del TAD. */
   private PositionalList<String> list;

   /** Inicialitza els atributs abans de cada test. */
   @Before
   public void setUp()
   {
      list = new DoublyLinkedList<String>();
   }

   /**
    * Comprova les operacions d'afegir al principi, esborrar posició i
    * d'altres relacionades.
    */
   @Test
   public void testPutFirst()
   {
      assertEquals(list.putFirst("1").getElem(), "1");
      assertEquals(list.putFirst("2").getElem(), "2");
      assertEquals(list.putFirst("3").getElem(), "3");
      assertEquals(list.putFirst("4").getElem(), "4");
      assertEquals(list.count(), 4);
      assertEquals(Ks.elementsToString(list), "4 3 2 1");
      BidirectionalIteration<Position<String>> it = ((DoublyLinkedList)
                        list).positions(BidirectionalIteration.REVERSE);
      try { it.next();
            fail("Expected: java.util.NoSuchElementException");
      }     catch (java.util.NoSuchElementException ex) {}
      for (char ch = '1'; it.hasPrevious(); ch++)
      {
         assertEquals(list.remove(it.previous()), String.valueOf(ch));
      }
      try { it.previous();
            fail("Expected: java.util.NoSuchElementException");
      }     catch (java.util.NoSuchElementException ex) {}
      assertTrue(list.isEmpty());
      assertEquals(list.putFirst("Bye").getElem(), "Bye");
      assertFalse(list.isEmpty());
   }
   // ...;
}
```

Figura 44

9.2 *Debugging* per pantalla (sense DbC)

La prova té una doble finalitat, es tracta de comprovar directament a la consola si el TAD es comporta adequadament, i també veure quines funcionalitats hi ha disponibles i com s'executen.

```
Crea un arbre de cerca AVL i hi afegeix 8 primers mesos:

                        May
                March
                        June
                                July
        January
                        February
                August
                        April

count() = 8;  isEmpty() = false;  children(root):  August March
root() = January;  isLeaf(root()) = false;  degree(root()) = 2

INORDER:     April August February January July June March May
PREORDER:    January August April February March June July May
POSTORDER:   April February August July June May March January
LEVELORDER:  January August March April February June May July

Esborrar tot, put('bye') i put('Bye'):  Bye bye
```

```
Crea el conjunt  A = {0,1,2,3,4,5,6}  i  B = {3,4,5,6,7,8,9}:

A.elements() = 0 1 2 3 4 5 6
B.elements() = 3 4 5 6 7 8 9

A.count() = 7;  A.isEmpty() = false;  A.exists('9') = false
B.count() = 7;  B.isEmpty() = false;  B.exists('9') = true

B.intersection(A),  B.elements() = 3 4 5 6
A.union(B),         A.elements() = 0 1 2 3 4 5 6
A.difference(B),    A.elements() = 0 1 2

A.count() = 3;  A.isEmpty() = false;  A.exists('3') = false
B.count() = 4;  B.isEmpty() = false;  B.exists('3') = true

A.intersection(B), A.put('bye') i A.put('Bye'):  Bye bye
```

```
runGraph:

     [java] Crea un digraf llista etiquetat amb 7 interrelacions:

     [java] (A)----75--->(C)
     [java] (A)----20--->(D)
     [java] (A)----60--->(E)
     [java] (C)----200--->(B)
     [java] (D)----2--->(C)
     [java] (D)----30--->(E)
     [java] (E)----30--->(B)

     [java] count() = 7;  isEmpty() = false;  vertexCount() = 5

     [java] DFS:  A E B D C
     [java] BFS:  A C D E B
     [java] TOS:  A D C E B

     [java] Cami minim del node 'A' a la resta:

     [java] [A] 0.0
     [java] [D] 20.0
     [java] [C] 22.0
     [java] [E] 50.0
     [java] [B] 80.0

     [java] Esborrar tot i put('Bye', 'bye', null):  (Bye)------->(bye)
```

Figura 45

9.3 Prova de violació d'assercions amb DbC

S'ha de preparar un embolcall (*wrapper*) per la classe que es vol provar. La classe ha d'implementar la interfície com també ho fa el nou *proxy* retornat:

- `InterfaceName<T> adt = DbcProxy.create( InterfaceName.class, new ClassName<T>(...) );`
- `adt.doSomething();`

També s'han d'habilitar les assercions, quan s'executa la prova, mitjançant l'opció `-enableassertions` de `java` (*Java application launcher*), per exemple:

- `java -ea uoc.ei.tads.tests.OldParamTest`

Fa una extensió d'un arbre binari i sobreescriu un mètode heretat per provocar la violació d'una postcondició.

```
16 public class OldParamTest<Elem>  extends LinkedBinaryTree<Elem>
17 {
18    /**
19     * Reemplaça l'element contingut a la posició rebuda.
20     * @param elem  nou element
21     * @param pos  posició de referència
22     * @return  element que hi havia a la posició
23     */
24    public Elem replace(Position<Elem> pos, Elem elem)
25    {
26       Elem old = pos.getElem();
27       ((BinTreeNode)pos).setElem(elem);
28
29       // return  old; // return OK
30       return  elem;   // return KO
31    }
32
33    /** Permet fer el test. */
34    public static void main(String[] args)
35    {
36       final String one = "1", two = "2";
37       Position<String> pos;
38       BinaryTree<String> tree =  DbcProxy.create(
39          BinaryTree.class, new OldParamTest<String>() );
40       pos = tree.put(null, one);
41       tree.replace(pos, two);
42    }
43 }
```

Figura 46

S'ha de considerar que el mètode sobreescrit hereta les assercions declarades a la interfície `uoc.ei.tads.PositionalContainer`:

```
23     *
24     * @pre   pos != null
25     * @pre  !$obj().isEmpty()
26     * @post  $obj().count() == $old().count()
27     * @post  pos.getElem() == elem
28     * @post  $return() == $oldparam(pos.getElem())
29     */
30    public Elem replace(Position<Elem> pos, Elem elem);
```

Figura 47

La violació d'una asserció fa que l'aplicació DbC interrompi l'execució.

```
Violacio d'una precondicio:
===========================

Exception in thread "main" java.lang.AssertionError: Precondition violated:
$obj().isComplete()
declared at
uoc.ei.tads.BoundedBinaryTree.putLast(Elem):47

        at uoc.ei.tads.BoundedBinaryTree_pre.putLast(BoundedBinaryTree_pre.java:94)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
        at java.lang.reflect.Method.invoke(Unknown Source)
        at uoc.ei.dbc.DbcProxy.invoke(DbcProxy.java:163)
        at $Proxy0.putLast(Unknown Source)
        at uoc.ei.tads.tests.PreconditionTest.main(PreconditionTest.java:26)

Violacio d'un invariant:
========================

Exception in thread "main" java.lang.AssertionError: Invariant violated:
$obj().count() >= 0
declared at
uoc.ei.tads.Container:24

        at uoc.ei.tads.OrderedSet inv.verify(OrderedSet inv.java:29)
        at uoc.ei.dbc.DbcProxy.invoke(DbcProxy.java:186)
        at $Proxy0.count(Unknown Source)
        at uoc.ei.tads.tests.InvariantTest.main(InvariantTest.java:28)

Violacio d'una postcondicio:
============================

Exception in thread "main" java.lang.AssertionError: Postcondition violated:
$return() == $oldparam(pos.getElem())
declared at
uoc.ei.tads.PositionalContainer.replace(uoc.ei.tads.Position, Elem):30

        at uoc.ei.tads.BinaryTree_post.replace(BinaryTree_post.java:135)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
        at java.lang.reflect.Method.invoke(Unknown Source)
        at uoc.ei.dbc.DbcProxy.invoke(DbcProxy.java:183)
        at $Proxy0.replace(Unknown Source)
        at uoc.ei.tads.tests.OldParamTest.main(OldParamTest.java:42)
```

Figura 48

En cadascun dels casos anteriors el programa avortaria, però s'ha preparat el fitxer per lots perquè executi consecutivament una prova de violació de cadascuna de les assercions (precondició, invariant i postcondició).

Els resultats conjunts anteriors s'obtindrien individualment executant:

- `java -ea uoc.ei.tads.tests.PreconditionTest`
- `java -ea uoc.ei.tads.tests.InvariantTest`
- `java -ea uoc.ei.tads.tests.OldParamTest`

Glossari

• **ADT** (*Abstract Data Type*)
Veure TAD

• **AOP** (*Aspect-Oriented Programming*)
És una nova tècnica que tracta per separat algunes responsabilitats, la implementació de les quals està a priori escampada per tot el sistema, com és en el cas de la OOP. Aquests aspectes poden ser la sincronització, distribució, manipulació d'errors, gestió de la seguretat, etc.

• **API** (*Application Program Interface*)
Interfície de comunicació entre un programa i els seus usuaris. En aquest context, es refereix a la documentació de les interfícies, generada pel *javadoc*, on es poden consultar les funcionalitats dels tipus abstractes de dades (TADs) i les estipulacions del contracte (DbC)

• **Asserció**
Expressió que involucra algunes entitats de programari i que estableix una condició específica que aqueixes han de satisfer en certes etapes de l'execució

• **DbC** (*Design by Contract*)
Fou introduït l'any 1992, per Bertrand Meyer, al llenguatge Eiffel, perquè les classes d'un sistema es poguessin comunicar, les unes amb les altres, sobre la base d'uns beneficis i obligacions definits amb precisió

• **Invariant**
Asserció que expressa restriccions de consistència generals que caracteritzen la semàntica de la interfície o classe

• **J2SE** (*Java Standard Edition*)
Plataforma estàndard del llenguatge de programació Java. Consta del JDK, JRE, documentació i altres productes relacionats. Tot el programari del projecte és compatible amb la plataforma de Sun Microsystems, Inc

• **JDK** (*Java Development Kit*)
Llibreria i paquet d'eines que permeten desenvolupar programari Java

• **JRE** (*Java Runtime Environment*)
Entorn que permet executar programari Java

• **JVM** (*Java Virtual Machine*)
Programari que serveix d'intèrpret entre les classes Java precompilades i un sistema operatiu específic. Permet executar aplicacions Java en qualsevol plataforma sense haver de canviar res del codi.

• **LS** (*line separator*)
Representa el salt de línia de la plataforma. S'utilitza en lloc del '`\n`' (*new line*) i '`\r`' (*carriage return*) sobretot per obtenir compatibilitat entre les plataformes MS-DOS/Windows i Unix/Linux

• **OOP** (*Object Oriented Programming*)
Veure POO

• **PFC** (Projecte final de carrera)
La carrera d'Enginyer en Informàtica consta actualment de 10 cursos semestrals i un projecte de final de carrera, amb un total de 300 crèdits (180 corresponen a la carrera prèvia d'Enginyer Tècnic)

• **POO** (Programació Orientada a l'Objecte)
Mètode de programació d'objectes modular on cada objecte programat és una entitat independent. Dins del paradigma s'utilitzen conceptes generals com abstracció, encapsulació o reutilització i característiques com herència, polimorfisme, generacitat, etc.

• **Postcondició**
Predicat lògic que s'ha de complir en acabar l'execució d'una operació, sempre i quan s'hagi complert prèviament la precondició corresponent: {P} C {Q}

• **Precondició**
Predicat lògic que s'ha de complir en començar l'execució d'una operació

• **TAD** (Tipus Abstracte de Dades)
Domini de valors sobre els quals es poden aplicar una o més operacions, que també formen part del TAD. Consta de dues parts: l'especificació, que introdueix les operacions del TAD i en fixa el comportament, i la implementació, que determina una representació dels valors el TAD i la codificació de les operacions d'acord amb aquesta representació.

• **UML** (*Unified Modeling Language*)
És un llenguatge per especificar, dissenyar i construir sistemes, inicialment de programari orientat a objectes. L'UML intenta definir un llenguatge homogeni (un llenguatge gràfic) per modelar totes les fases del desenvolupament d'una aplicació, des de les especificacions del programa per part del client fins al disseny detallat per al programador.

Bibliografia

Tipus abstractes de dades (TADs)

- Estructures de dades. Especificació, disseny i implementació (4a ed.)
  Xavier Franch (1999)

- Tipos Abstractos de Datos (Apuntes E.T.S.I. Informàtica)
  Sergio Gálvez Rojas (2005/06)

- Data Structures and Algorithms in Java (4th edition)
  Michael T. Goodrich and Roberto Tamassia (2005)

- Objects, Abstraction, Data Structures and Design: Using Java
  Elliot B. Koffman and Paul A. T. Wolfgang (2005)

Especificacions algebraiques

- Software Engineering (5th edition)
  Chapter 10: “Algebraic Specification”
  Ian Sommerville (2000)

- Testing Implementations of Algebraic Specifications
  with Design-by-Contract Tools
  Isabel Nunes (2005)

Disseny per contracte (DbC)

- Object-Oriented Software Construction (Second Edition)
  Part C.11. “Design by Contract: Building reliable software”
  Bertrand Meyer (1997)

- Examples of Design by Contract in Java
  Reto Kramer (1999)

- Supporting Design by Contract in Java
  Martin Lackner (2002)

- Thinking in Java (3rd ed. Revision 4.0)
  15. Discovering Problems: “Using Assertions for Design by Contract”
  Bruce Eckel (2002)

Generació de codi

- Code generation using Javadoc
  Mark Pollack (2000)

- Automated Code Generation
  Matt Stephens (2002)

- Reflection vs. code generation
  Michael J. Rettig with Martin Fowler (2001)

Proxies

- Explore the Dynamic Proxy API
  Jeremy Blosser (2000)

- Java Reflection in Action
  Chapter 4: “Using Java's Dynamic Proxy”
  Ira R. Forman and Nate Forman (2004)

Assercions

- JDK 1.4. Tutorial
  Chapter 6: “Assertion facility”
  Greg Travis (2002)

- Using Assertions in Java Technology
  Qusay H. Mahmoud (2005)

Proves unitàries

- Tècniques de desenvolupament de programari:
  Tècniques de *testing* per a programari orientat a objectes
  Fatos Xhafa (2002)

- Java Development with Ant
  Chapter 4: “Testing with JUnit”
  Erik Hatcher and Steve Loughran (2002)

- JUnit in Action
  Chapter 3: Sampling JUnit
  Vincent Massol with Ted Husted (2003)

Annex

Es pot generar la distribució dels fonts i precompilats executant `ant dist` al directori base:

- emarine.zip
- emarine.tar.gz
- emarine.tar.bz2

Les distribucions comprimides apareixen al directori pare del directori base.

Esteve