Disseny modular II

Ricard Gavaldà

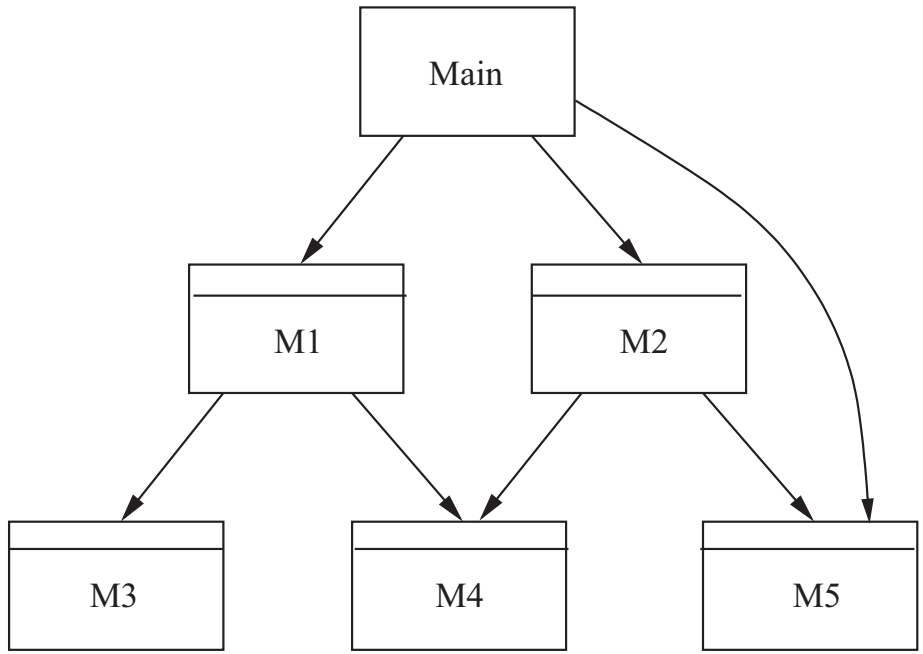
Programació 2
Facultat d'Informàtica de Barcelona, UPC
Primavera 2019

Aquesta presentació no substitueix els apunts

Contingut

Dependències entre mòduls i diagrames modulars

Diagrames modulars



"Jerarquia de classes" en programació OO
Diferents tipus d'arcs per a diferents tipus de relacions
En aquest curs, només relacions *d'ús*

Relacions entre mòduls

Programa = Conjunt de mòduls relacionats / dependents

Un mòdul pot:

- *definir* un nou tipus de dades, a partir d'altres
- *ampliar/enriquir* un tipus amb noves ops

Les relacions d'ús poden ser:

- visibles, en especificació
- ocultes, per una implementació concreta

Exercici: Conjunt d'Estudiants

Volem definir un nou tipus de dades, `Cjt_estudiants`, per gestionar conjunts d'estudiants

Relació d'ús: Dins de `Cjt_estudiants.doc`

```
#include "Estudiant.hh"
```

Important: per especificar `Cjt_estudiants` no cal saber la implementació de la classe `Estudiant`

Especificació de la classe `Cjt_estudiants`

Vegeu `Cjt_estudiants.doc`

Implementació de classes

Implementació d'una classe

Fases:

- Implementar el tipus: Triar una representació. Definir els atributs amb tipus ja existents
- Implementar les operacions: Codificar les seves operacions en termes d'instruccions
- Pot ser convenient definir operacions auxiliars (privades, no visibles des de fora)

Fitxers

Com m'agradaria a mi: Separació completa públic i privat

- .hh: Capçaleres de les ops públiques
- .cc: Atributs i codi de totes les ops.

Com es fa:

- .hh, part `private`: Atributs i capçaleres de ops. privades
- .hh, part `public`: Capçaleres de les ops. públiques
- .cc: Codi de les ops. públiques i privades

Exemple: Implementació de la classe `Estudiant`

Fitxers `Estudiant.hh` i `Estudiant.cc`

En aquest cas no hi ha mètodes privats, només els públics

Observeu la constant estàtica `MAX_NOTA`

Exercici: implementació alternativa del tipus `Estudiant`

Objectiu: demostrar la independència de la implementació

- Eliminem l'atribut booleà
- Si l'atribut `nota` és `-1`, l'estudiant no té nota

No canviem l'especificació $\rightarrow$ no cal revisar classes que l'usen

Atributs (al fitxer `.hh`)

- Variables o constants de tipus previs
- Sempre els declarem a la part `private`

- `const` = és una constant (no modificable)
- `static` = és *compartit* per tots els objectes de la classe
  - `classe.atribut` i no `objecte.atribut`

Operacions auxiliars privades al `.hh`

- Útils per a implementar les públiques
- Capçalera a la part `private`: No usables des de fora
- Poden referir-se als atributs de la classe
- Poden ser normals (tenen p.i.) o `static` (sense p.i.)

Codi dels mètodes (al fitxer `.cc`)

Notació: `nom_classe::nom_operacio(...)`

- :: vol dir "No estem implementant una op. nova, sinó la que ja haviem declarat abans"
- lliguem cada operació amb les seva capçalera al corresponent arxiu `.hh`
- dóna el dret d'accedir a la part private de la classe
- tant per a ops. públiques com privades
- a les `static`, no es repeteix l'`static`

Accés a un camp/atribut

Forma general: `nom_objecte.nom_atribut`

Exemple: `x.c`

Si `x` és un objecte de tipus `T`, llavors `c` ha de ser un atribut de `T`

Accés a un camp/atribut

Casos particulars:

- Podem ometre `nom_objecte.` per referir-nos al paràmetre implícit
  - Ex: `dni` sols es refereix al camp `dni` del paràmetre implícit

- `this`: el paràmetre implícit, l'objecte propietari. Usos:
  - Desambiguar quan en aquell àmbit de visibilitat hi ha una variable amb el mateix nom que un atribut.
  - Per pasar el paràmetre implícit com a paràmetre explícit d'una operació
  - En realitat, cal escriure `(*this)` i `this->atribut`: Tema 7

Exemple: Implementació de `Cjt_estudiants`

Decisions:

- Constant estàtica `MAX_NEST`, no era a l'especificació
- Decisió: el vector `vest` estarà ordenat per dni en tot moment
  - penalitza `afegir_estudiant` i `llegir_cjt_estudiants`
  - afavoreix la cerca (perquè es pot fer dicotòmica)
  - se suposarà inclosa a totes les Pres i les Posts
  - Atenció: no es "visible" des de l'especificació!
- Operacions privades
  - `ordenar_cjt_estudiants` (amb p.i.)
  - `cerca_dicot` (`static`, sense p.i.)

Vegeu fitxers `Cjt_estudiants.hh` i `Cjt_estudiants.cc`

Invariant de la representació

- Propietats dels atributs que ens comprometem a mantenir en la implementació de les operacions
- Queda garantit si només es manipula la representació amb ops. constructores i modificadores
- Implícit com a Pre i Post a totes les operacions
- A mi m'agrada escriure'l junt amb la representació: molt bona documentació!

Invariant de la representació (2)

Estudiant:

si (amb_nota) llavors (0 <= nota <= MAX_NOTA)

o be (implementacio sense booleà)

(nota == -1) o bé (0 <= nota <= MAX_NOTA)

Cjt_estudiants:

- 0 <= nest <= vest.size() = MAX_NEST,
- tots els estudiants de vest[0..nest-1] tenen dnis diferents,
- vest[0..nest-1] està ordenat creixentment pels DNI dels estudiants

Observacions

Algunes implementacions poden afegir *restriccions d'implementació*. Exemple:

- `Cjt_estudiants` amb una mida màxima
- Afecta a algunes precondicions, "no està ple"
- No és inherent a `Cjt_estudiants` sinó a una implementació seva

Observacions

Algunes implementacions poden afegir *anotacions sobre eficiència*

- A una implementació de `Cjt_estudiants`,
  - `afegir_estudiant`: "tarda temps proporcional a la mida del conjunt"
  - `existeix_estudiant`: "tarda temps logarítmic en la mida del conjunt"
- En una altra,
  - `afegir_estudiant`: "tarda temps constant"
  - `existeix_estudiant`: "tarda temps lineal en la mida del conjunt"

No són anotacions inherents a `Cjt_estudiants` sinó a implementacions seves

Ampliacions de tipus de dades: mòduls funcionals i llibreries

Ampliacions de tipus de dades

Ampliar un TAD: afegir noves funcionalitats
Mecanismes d'ampliació en OO

1. *Modificar la classe* existent per afegir els nous mètodes
2. *Enriquiment* = definir les noves operacions *fora de la classe*
3. *Herència* amb mecanismes del llenguatge (no a PRO2)

Solució 1: Modificar la classe

A .doc i .hh: Afegir capçaleres dels nous mètodes
A .cc: Afegir codi dels nous mètodes

Pro : Sovint més eficient

Con : Cal tenir accés *i entendre* la implementació original

Addicionalment, pot modificar-se la representació del tipus (per eficiència) i això pot significar modificar el codi de les operacions ja existents

Solució 2: Definir operacions *fora de la classe*

- No es modifica ni l'especificació ni la implementació de la classe original
- En un mòdul funcional nou (.hh i .cc), o en la classe que les usa
- Accions i funcions convencionals, paràmetre explícit i no implícit

Solució 2: Definir operacions *fora de la classe* (2)

Avantatges:

- No cal tenir permís per modificar classe original
- No engreixa la classe original amb mètodes d'ús puntual
- No cal canviar el codi de les ops si canviés la implementació (només s'usen els mètodes públics)

Inconvenients:

- Possible ineficiència
- Incongruència amb altre disseny OO

Com triar entre Solució 1 i Solució 2?

Solució 1:

- Si són ops. essencials al significat del tipus, generals i potencialment útils en moltes situacions (reusables)
- Quan solucioni problemes d'eficiència de la Solució 2

Solució 2:

- Quan només s'apliquen a un problema particular
- No fer-la més complexa sense necessitat
- No multiplicar versions

Solució 3: Extensió usant l'*herència* del llenguatge

Herència en OO: definir nous tipus de dades que hereden atributs i mètodes d'un tipus definit per una classe existent

- Subclasse de la classe original. Hereda tots els mètodes existents i n'afegeix de nous.
- Nou tipus de dades
- Ampliem classe original, sense modificar-la
- Nou tipus de dependència entre classes/mòduls (no d'ús)
- Cursos posteriors. Prohibida a PRO2

Exemple: Ampliació de `Cjt_estudiants`

Volem afegir a `Cjt_estudiants` operacions per a:

- donat el DNI d'un estudiant *que sabem que és al conjunt*, esborrar-lo del conjunt
- sabent que el conjunt no és buit, obtenir l'estudiant de nota màxima

Solució 1: Modificar la classe. Nou .doc

```
class Cjt_estudiants {

private:
...
public:
...
// Modificadores
...
void esborrar_estudiant(int dni);
/* Pre: existeix un estudiant al paràmetre implícit amb DNI dni */
/* Post: el paràmetre implícit conté els mateixos estudiants que
   l'original menys l'estudiant amb DNI dni */

// Consultores
...
Estudiant estudiant_nota_max() const;
/* Pre: el paràmetre implícit conté almenys un estudiant amb nota */
/* Post: el resultat és l'estudiant del paràmetre implícit amb
   nota màxima; si en té més d'un, és el de dni més petit */

};
```

Solució 1: Modificar la classe. Nou .hh

```
...
  private:
     vector<Estudiant> vest;
     int nest;
     static const int MAX_NEST = 60;
     int imax;  /* Aquest és el nou atribut */

     /* Invariant de la representacio:
     ...
     imax val -1 si cap estudiant te nota, i altrament
     conte l'index en vest de l'estudiant amb nota maxima
     */

  public:
...
void esborrar_estudiant(int dni);
Estudiant estudiant_nota_max( ) const;
...
```

Solució 1: Modificar la classe. Nou .cc

- Creadores: `imax` s'inicialitza a -1
- segueix sent -1 mentre cap estudiant del conjunt té nota
- `afegir_estudiant`: s'actualitza `imax`, si cal
- `estudiant_nota_max()`: retorna `vest[imax]`
- tot temps constant

Solució 1: Modificar la classe. Nou .cc

```
/* Pre: hi ha un estudiant al paràmetre implícit amb DNI dni */
/* Post: el paràmetre implícit conté els mateixos estudiants
   que l'original menys l'estudiant amb amb DNI dni */
void Cjt_estudiants::esborrar_estudiant(int dni) {
        // la pre garanteix que trobarem un estudiant
        // amb DNI dni
    int i = cerca_dicot(vest,0,nest-1,dni);

    for (int j = i; j < nest-1; ++j) vest[j] = vest[j+1];
    --nest;
    if (i == imax) recalcular_posicio_imax();
    else if (imax > i) --imax;
}
```

Solució 2: Definir mètodes *fora de la classe*

Nou fitxer `E_Cjt_estudiants.hh`:

```
#include "Estudiant.hh"
#include "Cjt_estudiants.hh"

void esborrar_estudiant(Cjt_estudiants &Cest, int dni);
/* Pre: existeix un estudiant a Cest amb DNI dni */
/* Post: Cest conté els mateixos estudiants que el seu valor original
   menys l'estudiant amb DNI dni */

Estudiant estudiant_nota_max(const Cjt_estudiants &Cest);
/* Pre: Cest conté almenys un estudiant amb nota */
/* Post: el resultat és l'estudiant de Cest amb nota màxima;
   si en té més d'un, és el de dni més petit */
```

Solució 2: Definir mètodes *fora de la classe*

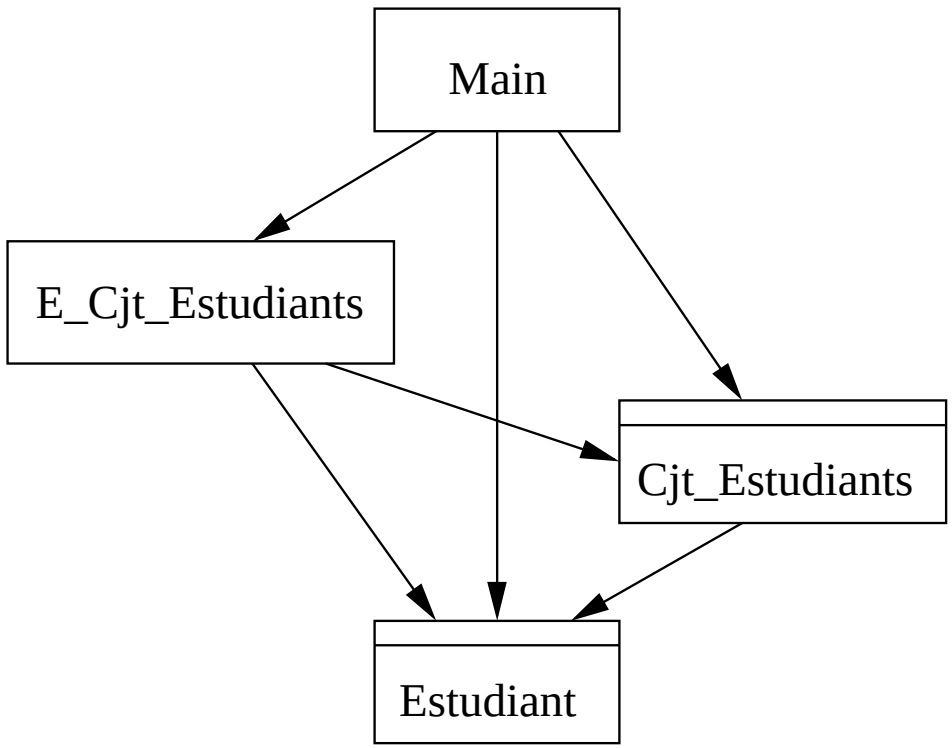
Vegeu `E_Cjt_estudiants.cc`

Observació:

- `estudiant_nota_max()` temps lineal (no constant)
- `esborrar_estudiant` lineal com abans, però més lenta

Diagrama modular

Un hipotètic programa principal que només provi les dues operacions de `E_Cjt_estudiants`



DEURES

Mirar i entendre les implementacions de les solucions 1 i 2

Entendre per què la solució 1 fa les noves ops molt més eficients

Biblioteques

Col.leccions de mòduls que amplien el llenguatge

"Standard C++ Library"

N'*usarem* els mòduls `<iostream>`, `<string>`, `<cmath>`

Standard Template Library (STL)

Template, plantilla, classe genèrica: Classe amb tipus com a paràmetres

Exemples:

- Programació 1: `<vector>`
- Programació 2: `<queue>`, `<stack>`, `<list>`

STL: Subconjunt de la Standard C++ Library que defineix `templates`

Preguntes

Què signifiquen?

- `vector<string> v;`
- `vector<string> v(10);`
- `vector<string> v(10,"m");`
- `string s;`
- `string s(10,'m');`
- `vector<vector<string> > v(10,vector<string>(20,"m"));`

- `vector<Estudiant> v(10);`
- `vector<Estudiant> v(10,Estudiant());`
- `vector<Estudiant> v(1,Estudiant(46382019));`

Metodologia de disseny modular

Metodologia de disseny modular

Disseny modular: construir una sèrie de mòduls de dades o funcionals i combinar-los per resoldre un problema determinat

5 fases:

1. Detectar les classes de dades implicades en el nostre problema, a partir de l'enunciat
2. Obtenir un esquema preliminar del programa principal (main)
3. Especificar les classes detectades que no puguem reusar d'algun projecte anterior
4. Escriure el programa principal, fent servir objectes de les classes especificades
5. Implementar les classes noves

Més en acostar-se la pràctica: Secció 2.3 dels apunts