Disseny modular II

R. Ferrer i Cancho

Universitat Politècnica de Catalunya

PRO2 (curs 2011-2012)
Versió 0.3

Avís: aquesta presentació no pretén ser un substitut dels apunts oficials de l'assignatura.

On som?

- Tema 1: Disseny modular i disseny basat en objectes
- 2a sessió.

Avui

- Com definir nous mòduls funcionals?
- Com descompondre un programa en mòduls? Metodologia

Dependències entre mòduls

Relació d'ús d'un mòdul per un altre

- Tipus bàsic de dependència.
- Utilitat:
  - Definició de nous tipus de dades a partir d'existents
  - *Ampliació/enriquiment* d'un tipus amb noves operacions.

Tipus de relacions (d'ús):

- *Visibles* en la jerarquia de classes.
- *Ocultes*: resultat d'implementacions concretes.

Diagrama modulars (del curs) I

Exemple:

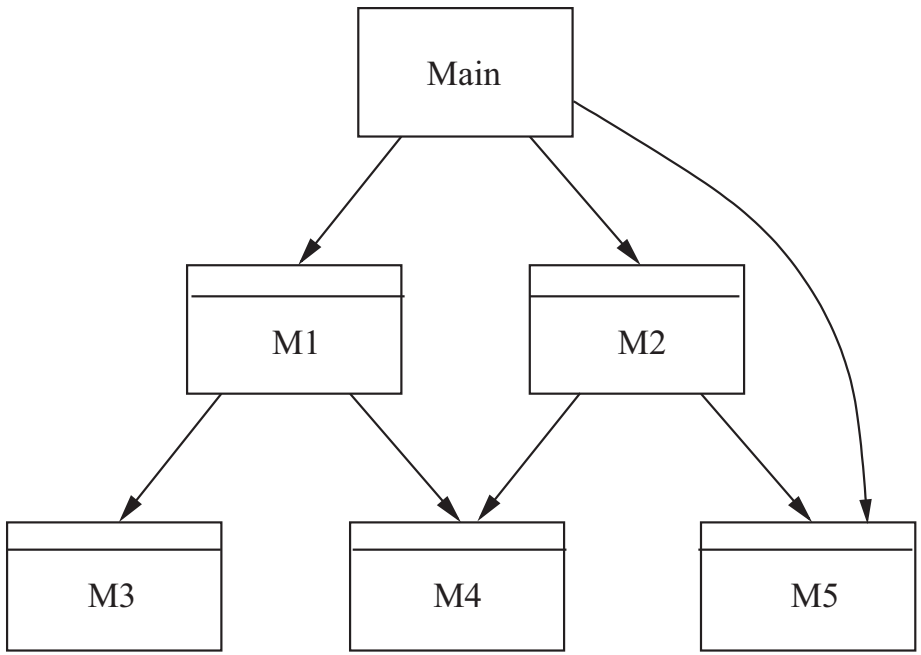


Diagrama modulars II

Definició: representacions gràfiques de les relacions d'ús visibles entre els diferents mòduls que formen un programa.

- Mòdul del programa principal:
  - Part superior del diagrama.
  - Conté el programa principal.
- Resta moduls: (normalment) mòduls de dades.
  - Significat dels 1ers veïns, 2ons veïns,... del modul del programa principal.

Propietats d'un bon diagrama modular

- Graf dirigit acílic (un mòdul pot ser usat per més d'un mòdul)
- Cicles $\rightarrow$ Mal disseny
- Compte: "Jerarquia de mòduls (o de classes)", suggereix erròniament un arbre.

Exercici: definir un nou tipus de dades, Cjt_estudiants, per manipular conjunts d'estudiants

- Sessió 1: especificació de la classe Estudiant.
- Cjt_estudiants *usa* la classe Estudiant.
- Relació d'ús en C++: directiva #include seguida del nom de l'arxiu de C++ que conté l'especificació de la classe usada (Estudiant.hh).

Nota: per especificar la classe Cjt_estudiants no cal disposar de la implementació de la classe Estudiant.

Especificació de la classe Cjt_estudiants

Veure exemples_disseny_modular_II.pdf

Ampliació d'un tipus: afegir-li noves funcionalitats (més mètodes)

Exemple: a partir de l'especificació de la classe `Cjt_estudiants`.

- Afegir operació d'esborrament d'estudiants.
- Consulta de l'estudiant amb nota màxima d'un conjunt.

Mecanismes d'ampliació:

- *Modificar la classe* existent per afegir els nous mètodes (solució 1)
- *Enriquiment* = definir les noves operacions *fora de la classe* (solució 2).
- Extensió de la classe usant l'*herència* del llenguatge (solució 3).

Solució 1: *Modificar la classe* existent per afegir els nous mètodes

Simple però hauríem de modificar

- Implementació (de la classe)
- Especificació (de la classe)

Problemes:

- Canvi del "contracte" d'ús de la classe (afecta als mòduls que l'usen)
- Pèrdua independència entre mòduls desitjada

Aplicar si

- Tenim accés a la implementació original
- Els beneficis superen els inconvenients.

Exemple I

Afegir a l'especificació de Cjt_Estudiants

```
class Cjt_estudiants {
...

// 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 */
```

Exemple II

```
// 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 2: definir les noves operacions *fora de la classe*

- No es modifica ni l'especificació ni la implementació de la classe original i, per tant, tampoc com és el seu ús.
- Les noves operacions s'especifiquen i s'implementen per separat
  - *A la classe on es faran servir* o
  - Donant lloc a un *nou mòdul funcional*.

Desavantatge:

- Mètodes no orientats a objecte (no propietat dels objectes de la classe original).
- Accions o funcions convencionals (objectes clase enriquida com a paràmetre explícit).

Especificació d'un mòdul funcional en C++

- Definir un arxiu `.hh` amb les especificacions Pre/Post de les noves operacions (no són mètodes)
- Operacions tindran paràmetre implícit associat?

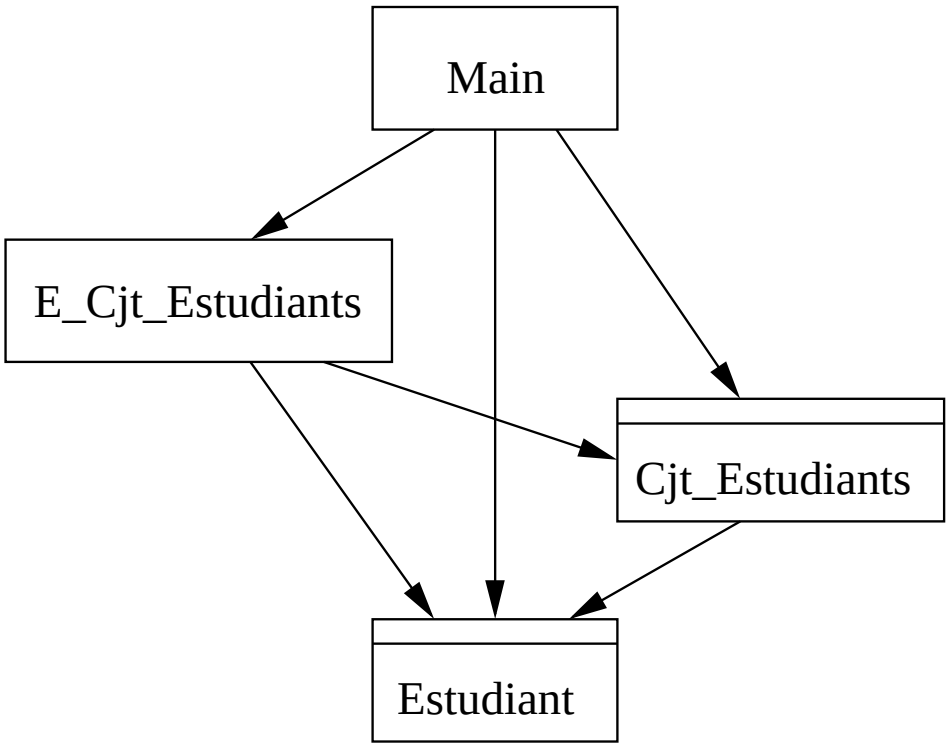
Exemple: E_Cjt_estudiants.hh

```
#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 */
```

Avançament: per separar la implementació de l'especificació del mòdul funcional, implementarem les noves operacions en un arxiu .cc que no serà visible des dels mòduls que usin l'ampliació.

Diagrama modular

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



Criteri per agrupar les operacions de l'enriquiment en un nou mòdul funcional

- Té un cert sentit sentit: si les noves operacions són prou generals (= eines per ser usadas des de diferent mòduls o en la resolució de diferents problemes).
- No sentit: si les noves operacions sobre tipus coneguts només es necessiten per resoldre un problema molt particular.
  - Especificar-les i implementar-les com a operacions auxiliars o privades de la classe on calguin.

Solució 3: Extensió de la classe usant l'*herència* del llenguatge

Herència: definir nous tipus de dades que heretin atributs i mètodes d'un tipus definit per una classe existent.

- Tipus de dependència entre classes/mòduls.
- Fonamental en llenguatges OO.
- Exemple: ampliació de la classe `Cjt_estudiants`, herència ens permetria definir una nova classe que heretés els atributs i els mètodes de l'original i que incorporés els mètodes nous.
- Important: implica definir un nou tipus de dades (els nous mètodes serien propietat d'objectes de la nova classe).
- S'estudiarà en cursos posteriors. Prohibida a PRO2.

+ relacions d'ús: biblioteques

Col·leccions de procediments o classes que extenen el llenguatge i són d'utilitat en el desenvolupament dels programes.
Tipus:

- Biblioteques estàndards d'un llenguatge. Exemple:
  - *"Standard C++ Library"*: funcionalitats bàsiques per realitzar diferent tasques
    Exemple: interacció amb el sistema operatiu, contenidors i manipuladors de dades i algorismes d'ús habitual. Mòduls que usarem en aquest curs:
    - `<iostream>`: canals estàndard d'entrada i sortida
    - `<string>`: classe *string* ($\sim$ **mòdul de dades**)
    - `<cmath>`: funcions matemàtiques ($\sim$ **mòdul funcional**).
- Qualsevol altra biblioteca que l'extengui.

Standard Template Library (STL)

Subconjunt Standard C++ Library que defineix `templates` (=plantilles): classes genèriques (tipus com a paràmetres).
Exemples:

- PRO1: `<vector>`
- PRO2: `<queue>` i `<stack>`.

Què significa

- `string s;` ?
- `vector<int> v(10);` ?

Implementació d'una classe

Fases:

- Implementar el tipus: proveir una representació determinada en termes dels tipus existents (definició i implementació dels atributs).
- Implementar les operacions: codificar les seves operacions en termes d'instruccions.

Novetat: introduir la notació del llenguatge C++ per implementar operacions d'una classe.

Recordatori

Recordatori:

- Independència entre l'ús dels mòduls i la seva implementació.
- Exemple: implementarem la classe `Estudiant` que ja hem usat sense haver-la implementat encara.
- Mecanismes independència: les paraules `private` i `public` de C++.
- `public`: usable des de fora de la classe
- `private`: només usable des de dins de la classe.

Atributs són `private`. Què implica?

Implementació dels atributs i llur accés

- Implementació dels atributs del nou tipus: mitjançant qualsevol combinació dels tipus coneguts de què disposem: vectors, booleans, enters,... o qualsevol altre que creem.
- Implementació dels mètodes: implicarà en molts casos haver d'accedir al contingut dels atributs d'un objecte.

Notació general accés a atribut: `nom_objecte.nom_atribut`
Exemple:

```
x.c
```

Correcte si:

- `x` és un objecte de tipus `T`
- `c` és un atribut de `T`

Fitxers

- (Especificació d'una classe en C++: 1 fitxer `.doc` (sessió 1))
- Implementació d'una classe en C++: 2 fitxers
  - `.hh`: almenys les capçaleres dels mètodes i certa informació sobre la representació del tipus
  - `.cc`: almenys el codi dels mètodes.

Fitxer .hh I

Capçaleres dels mètodes:

- Les mateixes que a la seva especificació.
- Romanen públiques.

Representació del tipus:

- Tres atributs o camps normals:
  - un enter pel DNI
  - un real per la nota
  - un booleà per definir si l'estudiant té nota o no.
- Constant `MAX_NOTA`, declarada `static`: el seu valor serà compartit per tots els possibles objectes de tipus `Estudiant`.

Fitxer .hh II

- Atributs i la constant es declaren privats.
- La constant MAX_NOTA és un *atribut constant*
- Noteu que no necessitem usar el constructor `struct` del C++ per agrupar els atributs dins d'una classe (*class* es una generalització de `struct`).

Fitxer .hh III

Invariant de la representació:

- Propietats per delimitar els possibles valors que els camps poden prendre per representar objectes vàlids de la classe.
- S'han de mantenir en tot moment i es considera condició implícites a les pre i les post de les operacions **públiques**.
- S'ha d'afegir al fitxer .cc com a part de la seva documentació.

Veure doc annex (`exemples_disseny_modular_II.pdf`): `Estudiant.hh`.

Fitxer .cc I

Fitxer .cc

- Conté codificació dels mètodes.
- Notació: nom_classe::nom_operacio(...)...

```
bool Estudiant::te_nota() const
/* Pre: cert */
{
    return amb_nota;
}
/* Post: el resultat indica  si el paràmetre implícit
   té nota o no */
```

- Operacions static només tenen accés als atributs static de la classe.

Fitxer .cc II

- Objectiu notació: lligar cada operació amb les seva capçalera al corresponent arxiu `.hh` donar el dret de veure els camps dels objectes de la classe.
- En un fitxer `.cc` es poden incloure operacions auxiliars sense paràmetre implícit: declarades `static`
  (una altra possibilitat: totalment alienes a la classe/no orientades a objecte (sense `nom_classe::`))

Accés a un camp/atribut

Forma general: `nom_objecte.nom_atribut`

Casos particulars:

- `this`: per referir-se al paràmetre implícit (objecte propietari).
  - Us obligatori quan coexisteixin en un mateix bloc de codi alguna variable amb el mateix nom que un atribut.
  - Exemple: mètodes `afegir_nota` i `modificar_nota`.
- Estalvi de `nom_objecte.`: (si no hi ha confusió de noms) `dni` sols es refereix al camp `dni` del paràmetre implícit.
  Exemple: mètode `te_nota()`

`this` es pot passar com a paràmetre d'una operació.

Estudiant.cc

Veure doc annex (exemples_disseny_modular_II.pdf): Estudiant.cc.

Exercici: implementació alternativa del tipus Estudiant

Objectiu: demostrar la independència de la implementació)

- Estalvi de l'atribut booleà
- Usar l'atribut `nota` per identificar quan l'estudiant no té nota (exemple: posant-li algun valor negatiu, per exemple el -1; si nota $\neq$ -1 aleshores té nota altrament no en té).

Canviar la implementació triada sense que afecti l'especificació $\rightarrow$ no cal revisar els algorismes que usin la classe.

+implementacions: fitxer Cjt_estudiants.hh

- Constant MAX_NEST
  - No era en l'especificació.
  - Objectiu: limitar la capacitat dels conjunts i poder implementar-los amb un vector
  - Declarada static. Què implica?
- Atribut nest: ne d'estudiants del conjunt / zona ocupada del vector
- Vector ordenat (penalitza a afegir_estudiant i llegir; afavoreix operacions de cerca: ús de cerca dicotòmica)
- Operació privada: ordenar.
- Operació privada **static**: cerca_dicot
  Operacions static: operacions auxiliars sense paràmetre implícit però dins de la classe.

Cjt_estudiants.hh

Veure doc annex (exemples_disseny_modular_III.pdf): Cjt_estudiants.hh.

+ implementacions: fitxer Cjt_estudiants.cc

Remarques:

- vest vector d'estudiants ordenat creixentment per DNI (ímplicit a la pre i post de totes les operacions).
- Noteu que la precondició d'afegir_estudiant s'ha de restringir a causa d'haver triat implementació sobre un vector.
- No sempre podem implementar les operacions exactament com han estat especificades.
- **Nova operació pública** `static int mida_maxima()` (poder comprovar les noves precondicions).
- Operació auxiliar (privada) per a les cerques dicotòmiques (no apareix a l'especificació)

Veure doc annex (exemples_disseny_modular_III.pdf): Cjt_estudiants.cc.

Formes d'ampliar la classe Cjt_estudiants

Amplicació:

- Nova operació per esborrar un estudiant donat el seu DNI.
- Nova operació per obtenir l'estudiant amb nota màxima.

Recordem:

- Solució 1: Modificar directament els arxius `Cjt_estudiants.hh` i `Cjt_estudiants.cc` amb el codi de les noves operacions.
- Solució 2: Definir un mòdul funcional amb les noves operacions (no mètodes orientats a objecte).
- Solució 3: Herència (no el veurem al curs).

Ampliació la classe Cjt_estudiants: solució 2

Veure doc annex (exemples_disseny_modular_III.pdf):
Solució 2: E_Cjt_estudiants.cc.

Solució 2: la proposta per esborrar_estudiant és molt dolenta (un nou conjunt amb els estudiants que han de romandre a Cest + assignació al final).

- Motiu: disposem d'operacions per modificar Cest directament, cap d'elles serveix per reduir-ne la mida.

Ampliació la classe Cjt_estudiants: solució 1

Solució 1: millor (+elegant i +curta) si treballem des de dins de la classe (podem accedir als camps).

- Fer els canvis que creguem oportuns en la implementació del tipus per aconseguir una implementació més eficient de les operacions.
  - Nou atribut al `Cjt_estudiants.hh` que ens permeti recordar la posició de l'estudiant amb la nota màxima
- Desavantatge: implica canviar l'especificació de la classe `Cjt_estudiants` i les seves regles d'ús.

Nou Cjt_estudiants.hh

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

Nou Cjt_estudiants.cc I

```
void Cjt_estudiants::esborrar_estudiant(int dni)
/* Pre: existeix un estudiant al paràmetre implícit amb DNI = dni */
{
  int i = 0;
  while (dni != vest[i].consultar_DNI())  ++i;

  // per la pre, segur que trobarem a Cest un estudiant amb DNI = dni;
  // en aquest punt del programa, aquest estudiant és vest[i];
  // ara desplacem els elements següents per ocupar el lloc de vest[i],
  // actualitzem la mida i mirem si s'ha d'actualitzar imax

  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;
}
/* Post: el paràmetre implícit conté els mateixos estudiants que
   l'original menys l'estudiant amb DNI = dni */
```

Nou Cjt_estudiants.cc II

```
Estudiant Cjt_estudiants::estudiant_nota_max( ) const
  /* Pre: el paràmetre implícit conté almenys un estudiant
     amb nota */
{
 return vest[imax];
}
  /* Post: el resultat és l'estudiant del paràmetre implícit amb nota màxima;
     si en té més d'un, és el de dni més petit */
...
```

Remarques sobre les millores

Implementació d'`esborrar_estudiant`:

- Estalvi de totes les crides a `afegir_estudiant` (cadascuna de les quals suposava un recorregut del vector,
- L'assignació amb el conjunt auxiliar.

Implementació de `estudiant_nota_max`

- Ja no cal cercar la nota màxima del vector, ja que la tenim a la posició `imax`.

Exercici:

- Implementació operació privada `recalcular_posicio_imax` (es crida a `esborrar_estudiant`).
- Altres canvis en la implementació de les operacions públiques a `Cjt_estudiants.cc` per inicialitzar i actualitzar l'atribut `imax`.

Metodologia de disseny modular

4 pinzellades de `Metodologia_disseny_modular.pdf`