

# Disseny modular II

## Programació 2

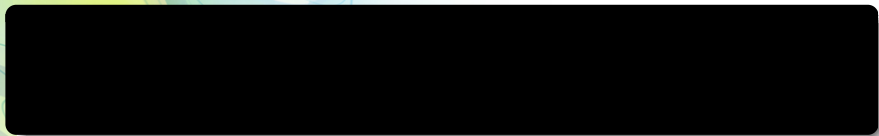
### Facultat d'Informàtica d'Informàtica, UPC

Conrado Martínez

Primavera 2019

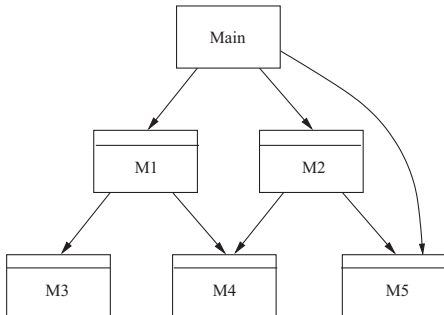
- Apunts basats en els d'en Ricard Gavalrà
- Aquestes transparències **no** substitueixen els apunts de l'assignatura, els complementen

# Part I



- 1 Jerarquies de Classes i Mòduls
- 2 Implementació de classes
- 3 Ampliacions de tipus de dades: mòduls funcionals i llibreries
- 4 Biblioteques i Genericitat

# 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**

# Diagrames modulars

- Mòdul del programa principal: sense arcs entrants
- Resta mòduls: mòduls de dades o funcionals
- Graf acíclic, no necessàriament un arbre
- “Jerarquia de classes” en programació OO

# 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 operacions

# 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 operacions

Les relacions d'ús poden ser:

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

## Exercici: Conjunt d'Estudiants

Volem definir una nova classe `Cjt_estudiants`, per gestionar conjunts d'estudiants

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

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

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

# Especificació de la classe Cjt\_estudiants

```
#include "Estudiant.hh"

// Un Cjt_estudiant Representa un conjunt d'estudiants,
// ordenat per DNI creixent amb un màxim nombre d'Estudiants
// Es poden consultar i modificar els seus elements
// (Estudiants) per DNI o per posició en l'ordre
// creixent de DNI

class Cjt_estudiants {
public:
    // Constructores

    /* Pre: cert */
    /* Post: crea un conjunt d'estudiants buit */
    Cjt_estudiants();

    // Destructora
    ~Cjt_estudiants();
```



# Especificació de la classe Cjt\_estudiants

```
// Modificadores

/* Pre: el conjunt no conté cap estudiant amb el DNI
       de l'Estudiant 'est'; la mida actual del conjunt
       és menor que la mida màxima permesa */
/* Post: s'ha afegit l'estudiant 'est' al conjunt */
void afegir_estudiant(const Estudiant &est);

/* Pre: el conjunt conté un estudiant amb el mateix DNI
       que l'Estudiant 'est' */
/* Post: l'Estudiant 'est' substitueix a l'estudiant del
       conjunt original que tenia el mateix DNI que 'est' */
void modificar_estudiant(const Estudiant &est);
```

# Especificació de la classe Cjt\_estudiants

```
/* Pre: 1 <= i <= nombre d'estudiants en el conjunt,  
       l'i-èssim Estudiant del conjunt en ordre creixent  
       per DNI té el mateix DNI que 'est' */  
  
/* Post: l'Estudiant 'est' substitueix a l'i-èssim estudiant  
        en ordre creixent de DNI del conjunt original */  
void modificar_iessim(int i, const Estudiant &est);
```

# Especificació de la classe Cjt\_estudiants

```
// Consultores

/* Pre: cert */
/* Post: el resultat és el nombre d'estudiants del conjunt */
int mida() const;

/* Pre: dni > 0 */
/* Post: torna cert si i només si el conjunt conté un Estudiant
        amb DNI igual al donat (dni) */
bool existeix_estudiant(int dni) const;

/* Pre: el conjunt conté un Estudiant amb DNI = dni */
/* Post: el resultat és l'Estudiant amb DNI = dni
        present en el conjunt */
Estudiant consultar_estudiant(int dni) const;
```

# Especificació de la classe Cjt\_estudiants

```
/* Pre: 1 <= i <= mida del conjunt */  
/* Post: torna l'Estudiant i-èssim del conjunt  
        en ordre creixent de DNI */  
Estudiant consultar_iessim(int i) const;  
  
// Mètode de classe  
/* Pre: cert */  
/* Post: el resultat es el nombre maxim d'estudiants que  
        pot arribar a tenir un Cjt_Estudiant */  
static int mida_maxima();
```

# Especificació de la classe Cjt\_estudiants

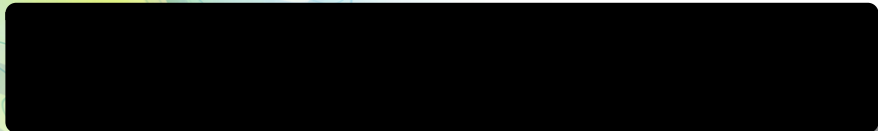
```
// Lectura i escriptura

/* Pre: cert */
/* Post: el paràmetre implícit conté el conjunt d'estudiants
        llegits del canal estàndard d'entrada */
void llegir();

/* Pre: cert */
/* Post: s'han escrit en canal estàndard de sortida els
        estudiants del conjunt en ordre ascendent per DNI */
void escriure() const;

private:
// elements privats de la classe: atributs,
// mètodes privats, ...
};
```

# Part I



- 1 Jerarquies de Classes i Mòduls
- 2 Implementació de classes
- 3 Ampliacions de tipus de dades: mòduls funcionals i llibreries
- 4 Biblioteques i Genericitat

# 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 mètodes i funcions auxiliars (privades, no visibles des de fora)

# Fitxers

Ideal: Separació completa públic i privat

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



# Fitxers

Ideal: Separació completa públic i privat

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

Com es fa:

- `.hh`  $\Rightarrow$  `part private`: Atributs i capçaleres de operacions privades
- `.hh`  $\Rightarrow$  `part public`: Capçaleres de les operacions públiques
- `.cc`  $\Rightarrow$  Implementació de les operacions públiques i privades

# Exemple: Implementació de la classe Estudiant

## Fitxer Estudiant.hh

```
class Estudiant {
public:

    // ...

    // Lectura i escriptura
    void llegir();
    void escriure() const;

private:
    int dni;
    double nota;
    bool amb_nota;
    static const int MAX_NOTA = 10;

    /* Invariant de la representacio:
       dni >= 0
       si amb_nota llavors 0 <= nota <= MAX_NOTA
    */
};
```

## Exemple: Implementació de la classe `Estudiant`

- En aquesta implementació de la classe `Estudiant` no es defineixen mètodes privats
- La constant estàtica `MAX_NOTA` és un atribut de classe

# Exemple: Implementació de la classe Estudiant

## Fitxer Estudiant.cc

```
#include <iostream>
#include "Estudiant.hh"
using namespace std;

Estudiant::Estudiant()
/* Pre: cert */
/* Post: crea un estudiant amb DNI=0 i sense nota */
{
    dni = 0; amb_nota = false;
}

Estudiant::Estudiant(int d)
/* Pre: dni>=0 */
/* Post: crea un estudiant amb DNI=d i sense nota */
{
    dni = d; amb_nota = false;
}

Estudiant::~Estudiant() {}
```

## Exemple: Implementació de la classe Estudiant

```
void Estudiant::afegir_nota(double n)
/* Pre: l'Estudiant no té nota i  $0 \leq n \leq \text{nota\_maxima}()$  */
/* Post: la nota de l'Estudiant passa a ser "nota" */
{
    nota = n;
    amb_nota = true;
}

void Estudiant::modificar_nota(double n)
/* Pre: l'Estudiant té nota i  $0 \leq n \leq \text{nota\_maxima}()$  */
/* Post: la nota de l'Estudiant passa a ser "nota" */
{
    nota = n;
}

int Estudiant::consultar_DNI() const
/* Pre: cert */
/* Post: torna el DNI de l'Estudiant */
{
    return dni;
}
```

19 / 65

## Exemple: Implementació de la classe Estudiant

```
double Estudiant::nota_maxima() // aquí no es posa "static"
/* Pre: cert */
/* Post: torna la nota màxima que pot tenir qualsevol Estudiant */
{
    return MAX_NOTA;
}

bool Estudiant::te_nota() const
/* Pre: cert */
/* Post: torna cert si i només si l'Estudiant té nota */
{
    return amb_nota;
}
```

## Exemple: Implementació de la classe Estudiant

```
void Estudiant::llegir()
/* Pre: el cin conté un enter no negatiu d i un double n */
/* Post: l'Estudiant passar a tenir el DNI d; si n està
        en el rang [0..nota_maxima()] llavors la nota de
        l'Estudiant passa a ser n; altrament l'Estudiant
        es queda sense nota */
{
    cin >> dni;
    double x;
    cin >> x;
    if (x >= 0 and x <= MAX_NOTA) {
        nota = x;
        amb_nota = true;
    } else {
        amb_nota = false;
    }
}
```

## Exemple: Implementació de la classe Estudiant

```
void Estudiant::escriure() const
/* Pre: cert */
/* Post: s'han escrit en el cout el DNI i la nota de l'Estudiant
        si l'Estudiant té nota, separats per un espai en blanc,
        o bé el DNI seguit de "NP" si l'Estudiant
        no té nota; a més de aquesta informació, a la sortida
        s'ha escrit un salt de línia */
{
    if (amb_nota) cout << dni << " " << nota << endl;
    else cout << dni << " NP" << endl;
}
```



# 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 → no cal revisar classes que l'usen

# Atributs

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

# Atributs

- Variables o constants de tipus previs
- Sempre els declarem a la part `private`
- `const` = és una constant (no modificable)

# Atributs

- Variables o constants de tipus previs
- Sempre els declarem a la part `private`
- `const` = és una constant (no modificable)
- `static` = és un atribut de classe (*compartit* per tots els objectes de la classe)
  - S'accedeix amb `classe::atribut` i no `objecte.atribut`

# Operacions auxiliars privades

- Útils per a implementar les públiques
- Capçalera a la part `private` del fitxer `.hh`: Només poden ser cridades per un mètode públic o privat de la classe
- Tenen accés als atributs dels objectes de la classe
- Poden ser mètodes (s'apliquen sobre l'objecte propietari) o mètodes de classe (`static`)

# Implementació de mètodes públics i privats

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

- `::` vol dir “No estem implementant una op. nova, sinó la que ja havíem 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
- en la implementació dels mètodes de classe no es posa `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:

- Quan accedim als atributs de l'objecte implícit en un mètode només s'escriu el nom de l'atribut
  - Ex: `dni` sols es refereix al camp `dni` del paràmetre implícit



## Accés a un camp/atribut

### Casos particulars:

- Quan accedim als atributs de l'objecte implícit en un mètode només s'escriu el nom de l'atribut
  - Ex: `dni` sols es refereix al camp `dni` del paràmetre implícit
- En alguns casos molt específics necessitem referir-nos explícitament a l'objecte implícit d'un mètode: `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.
- Pasar el paràmetre implícit com a paràmetre explícit d'una operació
- En realitat, `this` és un apuntador a l'objecte implícit; l'objecte implícit és `*this`

## Exemple: Implementació de `Cjt_estudiants`

Representació i invariant:

- Un atribut de classe constant `MAX_NEST`, que estableix el màxim nombre d'estudiants que pot haver en un conjunt

## Exemple: Implementació de `Cjt_estudiants`

Representació i invariant:

- Un atribut de classe constant `MAX_NEST`, que estableix el màxim nombre d'estudiants que pot haver en un conjunt
- Un atribut enter `nest`, el nombre d'estudiants en el conjunt,  $0 \leq \text{nest} \leq \text{MAX\_NEST}$
- Un atribut `vest` que és un vector de `Estudiants`, de mida `MAX_NEST` i que estarà ordenat per dni en tot moment
  - els mètodes `afegir_estudiant` i `llegir_cjt_estudiants` seràn més complexos (i costosos en temps) per a garantir que els continguts del vector `vest` estàn ordenats
  - afavoreix la cerca (perquè es pot fer dicotòmica)

## Exemple: Implementació de `Cjt_estudiants`

### Operacions privades

- Un mètode privat `ordenar_cjt_estudiants`
- Un mètode de classe (`static`) privat `cerca_dicot`, rep explícitament el vector d'`Estudiant` sobre el qual es fa la cerca

# 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
- És bona *praxis* escriure'l junt amb la representació: molt bona documentació!

# Invariant de la representació

## Classe Estudiant

- `dni >= 0`
- `si (amb_nota) llavors (0 <= nota <= MAX_NOTA)`

o be (implementacio sense booleà)

- `dni >= 0`
- `(nota == -1) o bé (0 <= nota <= MAX_NOTA)`

# Invariant de la representació

## Classe `Cjt_estudiants`

- $0 \leq \text{nest} \leq \text{vest.size()} = \text{MAX\_NEST}$ ,
- tots els estudiants de `vest[0..\text{nest}-1]` tenen dnis diferents,
- `vest[0..\text{nest}-1]` està ordenat creixentment pels DNI dels estudiants

# Implementació de classes

## Cjt\_estudiants.hh

```
class Cjt_estudiants {  
    ...  
private:  
    vector<Estudiant> vest;  
    int nest;  
    static const int MAX_NEST = 20;  
  
    /*  
    Invariant de la representacio:  
    * 0 <= nest <= vest.size() = MAX_NEST,  
    * tots els estudiants en vest[0..nest-1] tenen DNI  
    * diferents, i  
    * vest[0..nest-1] esta ordenat creixentment pels DNI  
    * dels estudiants  
    */  
};
```



# Implementació de classes

## Cjt\_estudiants.hh

```
class Cjt_estudiants {
    ...
private:
    ...
void ordenar_cjt_estudiants();
/* Pre: cert */
/* Post: els Estudiants del conjunt estan ordenats
        creixentment pels seus DNI */

static int cerca_dicot(const vector<Estudiant> &vest,
                      int left, int right, int x);
/* Pre: vest[left..right] està ordenat creixentment
        per DNI, 0 <= left, right < vest.size() */
/* Post: si a vest[left..right] hi ha un element
        amb DNI = x, el resultat és una posició que
        el conté; si no, el resultat es -1 */
};
```

# Implementació de classes

## Cjt\_estudiants.cc

```
#include "Cjt_estudiants.hh"
#include <iostream>
using namespace std;

Cjt_estudiants::Cjt_estudiants() {
    nest = 0;
    vest = vector<Estudiant>(MAX_NEST);
}

Cjt_estudiants::~Cjt_estudiants() {}
```

# Implementació de classes

## Cjt\_estudiants.cc

```
void Cjt_estudiants::afegir_estudiant(const Estudiant &est) {  
    int dni = est.consultar_DNI();  
    int i = nest - 1;  
    while (i >= 0 and dni < vest[i].consultar_DNI()) {  
        vest[i + 1] = vest[i];  
        --i;  
    }  
    vest[i + 1] = est;  
    ++nest;  
}
```

# Implementació de classes

## Cjt\_estudiants.cc

```
void Cjt_estudiants::modificar_estudiant(const Estudiant &est) {  
    // per la Pre, segur que trobem el DNI d'est com a DNI  
    // d'algun element de vest[0..nest-1]; apliquem-hi la cerca  
    // dicotomica  
    int i = cerca_dicot(vest, 0, nest-1, est.consultar_DNI());  
    // i es la posicio amb el DNI d'est  
    vest[i] = est;  
}  
  
void Cjt_estudiants::modificar_iessim(int i, const Estudiant &est) {  
    vest[i-1] = est;  
}
```

# Implementació de classes

## Cjt\_estudiants.cc

```
int Cjt_estudiants::mida() const {  
    return nest;  
}  
  
int Cjt_estudiants::mida_maxima() {  
    return MAX_NEST;  
}  
  
bool Cjt_estudiants::existeix_estudiant(int dni) const {  
    int i = cerca_dicot(vest, 0, nest-1, dni);  
    return (i != -1);  
}
```

# Implementació de classes

## Cjt\_estudiants.cc

```
Estudiant Cjt_estudiants::consultar_estudiant(int dni) const {  
    int i = cerca_dicot(vest, 0, nest-1, dni);  
    return vest[i];  
}
```

```
Estudiant Cjt_estudiants::consultar_iessim(int i) const {  
    return vest[i-1];  
}
```

# Implementació de classes

## Cjt\_estudiants.cc

```
void Cjt_estudiants::llegir() {
    cin >> nest;
    for (int i = 0; i < nest; ++i) vest[i].llegir();
    ordenar_cjt_estudiants();
    // noteu que l'apliquem sobre el objecte implícit
}

void Cjt_estudiants::escriure() const {
    for (int i = 0; i < nest; ++i) vest[i].escriure();
}
```

# Implementació de classes

## Cjt\_estudiants.cc

```
// observem que no hi ha referencia a nest
int Cjt_estudiants::cerca_dicot(const vector<Estudiant> &vest,
                                int left, int right, int x) {
    int i; bool found = false;
    while (left <= right and not found) {
        i = (left + right)/2;
        if (x < vest[i].consultar_DNI()) right = i - 1;
        else if (x > vest[i].consultar_DNI()) left = i + 1;
        else found = true;
    }
    // si l'element buscat existeix, i es la posicio que volem
    if (found) return i;
    else return -1;
}
```



# Implementació de classes

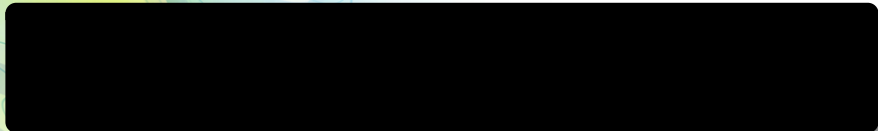
```
// ordena el vector d'estudiants per dni creixentment,  
// usant el metode de seleccio  
// Es un exemple. Milloraria usant algorismes d'ordenacio  
// mes rapids.  
void Cjt_estudiants::ordenar_cjt_estudiants() {  
    for (int i = 0; i < nest - 1; ++i) {  
        int min_dni = vest[i].consultar_DNI();  
        int pos_min = i;  
        for (int j = i+1; j < nest - 1; ++j)  
            if (min_dni > vest[j].consultar_DNI()) {  
                pos_min = j;  
                min_dni = vest[j].consultar_DNI();  
            }  
        Estudiant etemp = vest[i];  
        vest[i] = vest[pos_min];  
        vest[pos_min] = etemp;  
    }  
}
```

# Observacions

Algunes especificacions poden incloure *anotacions sobre eficiència*, que restringeixen el ventall d'implementacions vàlides

- Especificació #1 de `Cjt_estudiants`,
  - `afegir_estudiant`: “tarda temps proporcional a la mida del conjunt”
  - `existeix_estudiant`: “tarda temps logarítmic en la mida del conjunt”
- Especificació #2:
  - `afegir_estudiant`: “tarda temps constant”
  - `existeix_estudiant`: “tarda temps lineal en la mida del conjunt”

# Part I



- 1 Jerarquies de Classes i Mòduls
- 2 Implementació de classes
- 3 Ampliacions de tipus de dades: mòduls funcionals i llibreries
- 4 Biblioteques i Genericitat

# Ampliacions de tipus de dades

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

- 1 *Modificar la classe* existent per afegir 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

- 1 Afegir les capçaleres dels nous mètodes en el fitxer `.hh`
- 2 Implementar els nous mètodes en el fitxer `.cc`

**Pro** : Sovint més eficient

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

## Solució 1: Modificar la classe

- 1 Afegir les capçaleres dels nous mètodes en el fitxer `.hh`
- 2 Implementar els nous mètodes en el fitxer `.cc`

**Pro** : Sovint més eficient

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

Adicionalment, pot modificar-se la representació del tipus (per poder soportar eficientment els nous mètodes) 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, no són mètodes

## Solució 2: Definir operacions *fora de la classe*

### 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 el 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 i no sembla que es pugui reutilitzar en altres contextes
- Quan cal evitar que la classe original creixi desmesuradament; potser s'ha de plantejar un redisseny de les classes i introduir-ne noves classes

## 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

```
class Cjt_estudiants {  
public:  
    ...  
    void esborrar_estudiant(int dni);  
    /* Pre: el conjunt conté un estudiant amb DNI = dni */  
    /* Post: el conjunt conté els mateixos estudiants que  
        l'original menys l'estudiant amb DNI donat */  
    ...  
    Estudiant estudiant_nota_max() const;  
    /* Pre: el conjunt conté almenys un estudiant amb nota */  
    /* Post: el resultat és l'estudiant del conjunt 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 atribut és nou */  
  
    /* Invariant de la representacio:  
    ...  
    imax val -1 si cap estudiant té nota, i altrament  
    conté l'index més petit en vest d'un estudiant amb  
    nota màxima */
```

## Solució 1: Modificar la classe

- Creadores: `imax` s'inicialitza a -1
- Segueix sent -1 mentre cap estudiant del conjunt té nota
- `afegir_estudiant`: s'actualitza `imax`, si cal
- Idem amb `modificar_estudiant` i `modificar_iessim`
- `estudiant_nota_max()`: retorna `vest[imax]`
- `afegir_estudiant` i `estudiant_nota_max` **ténen temps constant**=independent del nombre d'estudiants en el conjunt
- `modificar_estudiant` i `modificar_iessim` **podem** necessitar un recorregut del vector sencer per a actualitzar `imax`

## Solució 1: Modificar la classe

```
/* Pre: el conjunt conté un estudiant amb DNI = dni */
/* Post: el conjunt conté els mateixos estudiants
    que l'original menys l'estudiant 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: Cest conté un estudiant 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*

```
#include "E_Cjt_estudiants.hh"

/* Pre: Cest conté un estudiant a Cest amb DNI = dni */
/* Post: Cest conté els mateixos estudiants que el seu
        valor original menys l'estudiant amb DNI = dni */
void esborrar_estudiant(Cjt_estudiants &Cest, int dni) {
    Cjt_estudiants Cestaux;
    int i = 1;
    while (dni != Cest.consultar_iessim(i).consultar_DNI()) {
        Cestaux.afegir_estudiant(Cest.consultar_iessim(i));
        ++i;
    }
    // per la pre, segur que trobarem a Cest un estudiant
    // amb DNI = dni; en aquest punt del programa, aquest
    // estudiant és Cest.consultar_iessim(i);
    // ara hem d'afegir els elements següents a Cestaux
    for (int j = i+1; j <= Cest.mida(); ++j)
        Cestaux.afegir_estudiant(Cest.consultar_iessim(j));
    Cest = Cestaux;
}
```



## Solució 2: Definir mètodes *fora de la classe*

```
/* 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 */
Estudiant estudiant_nota_max(const Cjt_estudiants &Cest) {
    int i = 1;
    while (not Cest.consultar_iessim(i).te_nota()) ++i;
    int imax = i; ++i;
    // per la pre, segur que trobarem a Cest un estudiant
    // amb nota; imax n'és el primer; comprovem la resta
    while (i <= Cest.mida()){
        if (Cest.consultar_iessim(i).te_nota())
            if (Cest.consultar_iessim(imax).consultar_nota() <
                Cest.consultar_iessim(i).consultar_nota())
                imax = i;
        ++i;
    }
    return Cest.consultar_iessim(imax);
}
```

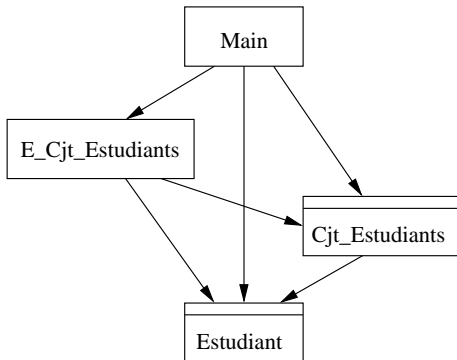
## Solució 2: Definir mètodes *fora de la classe*

### Observació:

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

# Diagrama modular

Un hipotètic programa principal que usa les dues operacions de `E_Cjt_estudiants`



# Part I



- 1 Jerarquies de Classes i Mòduls
- 2 Implementació de classes
- 3 Ampliacions de tipus de dades: mòduls funcionals i llibreries
- 4 Biblioteques i Genericitat

# Biblioteques

Col·leccions de mòduls que amplien el llenguatge

La *Standard C++ Library* ofereix una gran varietat de mòduls funcionals i de dades com ara `iostream`, `string`, `cmath`,...

# Standard Template Library (STL)

- La STL és un subconjunt de la biblioteca estàndar de C++. Inclou mòduls funcionals i de dades *genèrics*
- *Template* = plantilla
- Classes i funcions genèriques : classes i funcions amb tipus com a paràmetres Exemples:
  - Programació 1: `vector<T>`
  - Programació 2: `queue<T>`, `stack<T>`, `list<T>`

# Templates

## Una funció genèrica

```
template <typename T>
T minim(const vector<T>& v)
/* Pre: v és un vector no buit de T's; hi ha un ordre '<'
        total predefinit sobre els elements de tipus T */
/* Post: retorna l'element més petit de v */
{
    T mn = v[0];
    for (int i = 1; i < v.size(); ++i)
        if (v[i] < mn) mn = v[i];
    return mn;
}
```

# Templates

## Ús d'una funció genèrica

```
vector<int> v(100);  
int m = minim(v);  
//      ↑↑↑ crida a minim amb T=int  
vector<string> words;  
string smallest = minim(words);  
//      ↑↑↑ crida a minim amb T=string  
vector<Estudiant> vest;  
Estudiant mest = minim(vest);  
// falla perquè '<' no està definit  
// per a T=Estudiant!
```