

# Disseny modular I

## Programació 2

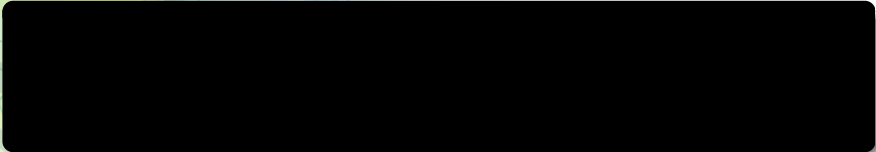
### Facultat d'Informàtica d'Informàtica, UPC

Professorat de PRO2

Tardor 2022

- Col·laboracions (en ordre alfabètic): Juan Luis Esteban, Ricard Gavaldà, Conrado Martínez, Fernando Orejas
- Aquestes transparències **no** substitueixen els apunts de l'assignatura, els complementen

# Part I

- 
- 1 Abstracció i disseny modular
  - 2 Disseny basat en objectes
  - 3 Especificació i ús de classes
  - 4 Exemples d'ús de la classe Estudiant
  - 5 Classe `Cjt_estudiants`

# Com abordar programes grans

Descomposició en *mòduls*. Clàssica en enginyeria

Facilita

- raonar sobre correctesa, eficiència, etc. per parts
- fer programes llegibles, reusables, mantenibles, etc.
- treballar en equip

## Què és una bona descomposició modular?

- *Independència*: canvis en un mòdul no han d'obligar a modificar altres mòduls.
- *Coherència interna*: els mòduls tenen significat per si mateixos. Interactuen amb altres mòduls de manera simple i ben definida

# Abstracció

Eina de raonament en programes grans:

Oblidar, temporalment, alguns detalls del problema per tal de transformar-lo en un o bé més simple o bé més general

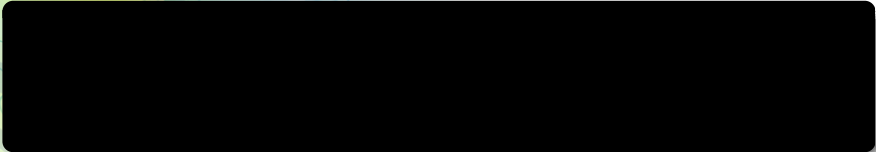
# Especificació Pre/Post

```
/* Pre:  $a > 0$  i  $b \geq 0$  */  
  int pot(int a, int b);  
/* Post: el resultat és  $a$  multiplicat per ell mateix  
         $b$  vegades */
```

## Especificació vs. implementació

- **Regla:** Un canvi en la implementació d'una funció que respecti la Pre/Post no pot mai fer que un programa que la usa deixi de funcionar
- Especificació = Contracte entre usuari i implementador
- Especificació = Abstracció de l'implementació

# Part I

- 
- 1 Abstracció i disseny modular
  - 2 Disseny basat en objectes**
  - 3 Especificació i ús de classes
  - 4 Exemples d'ús de la classe Estudiant
  - 5 Classe `Cjt_estudiants`



## Tipus de mòduls

- **Mòdul funcional:** conté un conjunt d'operacions noves necessàries per resoldre algun problema o subproblema
- **Mòdul de dades:** conté la definició d'un nou tipus i les seves operacions; és habitual a Programació 2

Com els fem “abstractes”?

- Mòdul funcional: només deixem veure les especificacions de les operacions
- Mòdul de dades: només deixem veure les capçaleres de les operacions del tipus i una explicació de com es comporten

# Tipus Abstracte de Dades (TADs)

Definim un tipus no per com està implementat, sinó per quines operacions podem fer amb les variables del tipus

Un tipus es defineix donant:

- El nom del tipus
- Operacions per construir, modificar i consultar elements del tipus
- Descripció de *qué* fan les operacions (no *com*)
- Un tipus de dades pot tenir diverses implementacions. El tipus **és** la seva especificació, no les seves implementacions

# TADs i independència entre mòduls

- 1 **Fase d'especificació:**  
Decidir operacions del TAD i contractes d'ús
- 2 **Fase d'implementació:**  
Decidir una representació i codificar les operacions

Conseqüència:

Un canvi en la implementació d'un TAD que no afecti l'especificació de les seves operacions no pot mai fer que un programa que usa el TAD deixi de funcionar

# Orientació a objectes

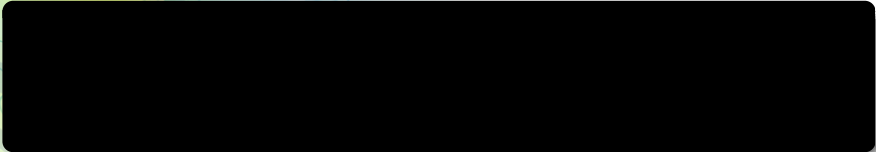
Una manera de separar especificació d'implementació,  
d'implementar Tipus Abstractes de Dades  
A Programació 2 només veurem *una part* de la utilitat d'aquesta  
manera de pensar

Més en altres assignatures: *herència* i *polimorfisme*

# Classes i objectes

- Les variables i constants d'un tipus són **objectes**
- Una **classe** és el patró comú al *objectes* d'un tipus
- A l'inrevés: Donada una classe, podem definir-ne objectes o instàncies
- Cada classe defineix els **atributs** (= camps) i els **mètodes** (= operacions) del tipus.
- Cada objecte és **propietari** dels seus atributs i mètodes
- Els mètodes tenen un **paràmetre implícit**: el seu propietari o objecte sobre el qual s'aplica el mètode
- Podem fer més accions/funcions que operen amb el tipus, però si no són dins de la classe no són mètodes

# Part I

- 
- 1 Abstracció i disseny modular
  - 2 Disseny basat en objectes
  - 3 Especificació i ús de classes**
  - 4 Exemples d'ús de la classe Estudiant
  - 5 Classe `Cjt_estudiants`

## Exemple: La classe `Estudiant`

Un `Estudiant` es caracteritza per:

- Un DNI, que és un enter no negatiu, obligatori
- Una nota, optativa. Si en té, és un real (`double`) entre 0 i un cert valor màxim (p.ex., 10). Si no la té, es considera NP

# Especificació de classes en C++

```
class Estudiant {
public:
    // Constructores
    Estudiant();
    /* Pre: cert */
    /* Post: el resultat és un estudiant amb DNI = 0
           i sense nota */

    Estudiant(int dni);
    /* Pre: dni >= 0 */
    /* Post: el resultat és un estudiant amb DNI = dni
           i sense nota */
};
```



# Especificació de classes en C++

```
...  
// Modificadores  
void afegir_nota(double nota);  
/* Pre: l'estudiant implícit no té nota i  
      'nota' és una nota vàlida */  
/* Post: la nota de l'estudiant implícit  
        passa a ser 'nota' */  
  
void modificar_nota(double nota);  
/* Pre: l'estudiant implícit té nota i  
      'nota' és una nota vàlida */  
/* Post: la nota de l'estudiant implícit  
        passa a ser 'nota' */
```

# Especificació de classes en C++

```
...  
// Consultores  
int consultar_DNI() const;  
/* Pre: cert */  
/* Post: retorna el DNI de l'estudiant */  
  
bool te_nota() const;  
/* Pre: cert */  
/* Post: retorna cert si i només si  
        l'estudiant té nota */  
  
double consultar_nota() const;  
/* Pre: l'estudiant té nota */  
/* Post: retorna la nota de l'estudiant */  
  
// Mètodes de classe  
static double nota_maxima();  
/* Pre: cert */  
/* Post: retorna la nota màxima que pot  
        tenir qualsevol estudiant */  
  
**18 / 41  
};
```

# Paràmetre implícit

En C++ sense OO:

## Declaració d'una funció/acció

```
/* Pre: cert */  
bool te_nota(const Estudiant &e);  
/* Post: El resultat és cert si i només si e té nota */
```

Amb OO:

## Declaració d'un mètode

```
/* Pre: cert */  
bool te_nota() const;  
/* Post: El resultat és cert si i només si el  
paràmetre implícit té nota */
```

Noteu el `const` referit a l'objecte implícit

# Exemple OO: crida a un mètode

Forma general:

```
<nom_de_l'objecte>.<nom_del_mètode>(<altres paràmetres>)
```

## Crida a una funció

```
bool b = te_nota(est);
```

## Aplicació/invocació d'un mètode

```
b = est.te_nota();
```

## Tipus d'operacions: *Creadores* o *Constructores*

**Creadores** = funcions que serveixen per crear objectes nous  
En C++, hi ha constructors:

- Tenen el mateix nom de la classe i creen un objecte nou d'aquest tipus
- No ténen paràmetre implícit! Creen un object abans no existent!
- La llista de paràmetres permet distingir entre diverses constructors
- **Constructora per defecte**: sense paràmetres, crea un objecte nou donant valors per defecte als atributs

## Tipus d'operacions: *Constructores*

### Exemple 1: Constructora por defecte

```
Estudiant est;
```

### Exemple 2: Constructora amb un paràmetre de tipus `int`

```
Estudiant est(46567987);
```

## Tipus d'operacions: *Modificadores*

- Transformen l'objecte propietari (paràmetre implícit), potser amb informació aportada per altres paràmetres
- Normalment en C++ retornen `void`; són accions
- Seguretat: Tots els canvis es fan via mètodes ben definits

## Exemple OO: crida a un mètode modificador

### Especificació

```
/* Pre: el paràmetre implícit té nota
      i 'nota' és una nota vàlida (entre 0
      i la nota màxima) */
void modificar_nota(double nota);
/* Post: la nota del paràmetre implícit
      passa a ser 'nota' */
```

Fixeu-vos: sense `const`, el paràmetre implícit pot ser modificat pel mètode  $\Rightarrow$  el mètode rep el seu paràmetre implícit per *referència*

### Crida

```
est.modificar_nota(x);
```



## Tipus d'operacions: *Consultores*

- Proporcionen informació sobre l'objecte propietari, potser amb ajut d'informació aportada per altres paràmetres
- Normalment porten `const` perquè no modifiquen el paràmetre implícit
- Normalment funcions, tret que hagin de retornar més d'un resultat; en aquest cas poden ser accions amb més d'un paràmetre de sortida (passat per referència)

## Tipus d'operacions: *Consultores*

### Exemple 1: Ús d'un mètode consultor

```
double x = est.consultar_nota();
```

### Exemple 2: Ús d'un mètode consultor

```
if (est.te_nota()) ... else ...
```

Aquest mètode consultor és necessari perquè hi ha operacions que tenen com a precondició que l'estudiant tingui o no tingui nota

## Mètodes de classe

- Són propis de la classe, no de cada objecte
- No tenen paràmetre implícit


### Mètode de classe

```
/* Pre: cert */  
static double nota_maxima();  
/* Post: el resultat és la nota màxima que  
poden tenir qualsevol estudiant */
```

### Crida d'un mètode de classe

```
cin >> nota;  
if (nota >= 0 and nota <= Estudiant::nota_maxima())  
    e.modificar_nota(nota);  
else  
    cout << "La nota introduïda no és vàlida" << endl;
```

# Part I

- 
- 1 Abstracció i disseny modular
  - 2 Disseny basat en objectes
  - 3 Especificació i ús de classes
  - 4 Exemples d'ús de la classe Estudiant
  - 5 Classe `Cjt_estudiants`

# Canviar un NP per 0

## Ús de la classe Estudiant

```
/* Pre: tots els elements de v són Estudiants  
amb DNIs diferents */  
bool canvia_np_per_zero(vector<Estudiant>& v, int dni);  
/* Post: si v conté un Estudiant amb DNI = dni, i aquest  
no té nota, llavors aquest estudiant passa  
a tenir nota 0; la resta de v no canvia;  
el resultat diu si l'estudiant s'ha trobat */
```

# Canviar un NP per 0

## Ús de la classe Estudiant

```
bool canvia_np_per_zero(vector<Estudiant>& v, int dni) {
    int i = 0;
    while (i < v.size()) {
        if (v[i].consultar_DNI() == dni) {
            if (not v[i].te_nota())
                v[i].afegir_nota(0);
            return true;
        }
        ++i;
    }
    return false;
}
```

# Calcular nota mitjana

## Un altre exemple d'ús

```
/* Pre: tots els Estudiants de v tenen DNIs diferents */  
double nota_mitjana(const vector<Estudiant>& v);  
/* Post: el resultat és la nota mitjana dels Estudiants  
que tenen nota; si cap Estudiant de v té nota,  
retorna -1 */
```


# Calcular nota mitjana

## Un altre exemple d'ús

```
double nota_mitjana(const vector<Estudiant>& v) {
    int n = 0;
    double suma = 0;
    for (int i = 0; i < v.size(); ++i) {
        if (v[i].te_nota()) {
            ++n;
            suma += v[i].consultar_nota();
        }
    }
    if (n > 0)
        return suma/n;
    else
        return -1;
}
```



# Part I

- 
- 1 Abstracció i disseny modular
  - 2 Disseny basat en objectes
  - 3 Especificació i ús de classes
  - 4 Exemples d'ús de la classe Estudiant
  - 5 **Classe** `Cjt_estudiants`

# Conjunt d'Estudiants

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

Per indicar que `Cjt_estudiants` fa servir `Estudiant`.

```
#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();
```

## 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 maxm d'estudiants que
        pot arribar a tenir un Cjt_Estudiant */
static int mida_maxima();
```

# Arrodonir notes

## Exemple d'ús de Cjt\_estudiants.hh

```
/* Pre: cert */  
void arrodonir_notes(Cjt_estudiants & c);  
/* Post: s'han arrodonit les notes dels estudiants de c  
que tenien nota*/
```

## Operació auxiliar

```
/* Pre: est te nota */  
void arrodonir(Estudiant& est){  
    est.modificar_nota(((int) (10. * (est.consultar_nota()  
                                + 0.05))) / 10.0);}  
/* Post: est passa a tenir la seva nota original arrodonida */
```



# Arrodonir notes

## Exemple d'ús de Cjt\_estudiants.hh

```
void arrodonir_notes(Cjt_estudiants & c){
    for (int i = 1; i <= c.mida(); ++i) {
        Estudiant e;
        e = c.consultar_iessim(i);
        if (e.te_nota()) {
            arrodonir(e);
            c.modificar_iessim(i,e);
        }
    }
}
```

Cal un objecte Estudiant auxiliar per poder fer servir modificar\_iessim.