

# Disseny modular I

## Programació 2

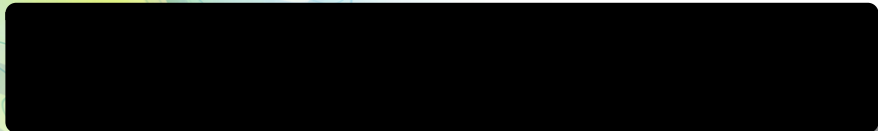
### Facultat d'Informàtica d'Informàtica, UPC

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 Abstracció i disseny modular
- 2 Descomposició funcional i per dades
- 3 Orientació a objectes
- 4 Especificació i ús de classes

# Motivació

- Com abordar projectes grans
- Quins ajuts ens pot donar el llenguatge de programació
- I quina disciplina hem de seguir els programadors

# 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 Descomposició funcional i per dades
- 3 Orientació a objectes
- 4 Especificació i ús de classes

## 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

# Abstracció per dades: tipus predefinit

`int`:

- Valors enters `MININT` .. `MAXINT`
- Operacions `+`, `*`, `%`, `/`, `<`, `>`, `==`, ...
- `a+b == b+a`; `a*b == b*a`, `a*(b+c)==a*b+a*c`, etc. (si no hi ha overflow)
- `a+0 == a`, `a*1 == a`, `a == a`, `a < a+1`, etc.
- ...

Que s'implementin en base 2 com a vectors de bits és irrellevant per a la majoria de problemes de Programació 1 i Programació 2

# Abstracció per dades: nous tipus

## Solució insatisfactòria - pro1

```
typedef struct {  
    double re, im;  
} Complex;  
  
void suma(Complex a, Complex b, Complex& c) {  
    c.re = a.re + b.re;  
    c.im = a.im + b.im;  
}
```

No hi ha res amagat. No hi ha contracte

Si decidim representar els complexos com a mòdul + angle (forma polar), cal canviar totes les accions/funcions que usen el tipus

# 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

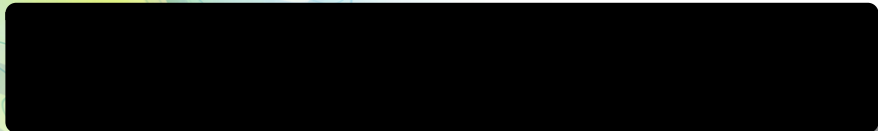
## 2 Fase d'implementació:

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

# Part I



- 1 Abstracció i disseny modular
- 2 Descomposició funcional i per dades
- 3 Orientació a objectes**
- 4 Especificació i ús de classes

# 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

## 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

## Exemple d'ús d'Estudiant: 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 */
```

## Exemple d'ús d'Estudiant: 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;  
}
```

## Exemple d'ús d'Estudiant: 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 */
```

## Exemple d'ús d'Estudiant: 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;  
}
```

# Paràmetre implícit

En C++ sense OO:

## Declaració d'una funció/acció

```
/* Pre: -- */  
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: -- */  
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();
```



## Exemple OO: crida a un mètode modificador

### Especificació

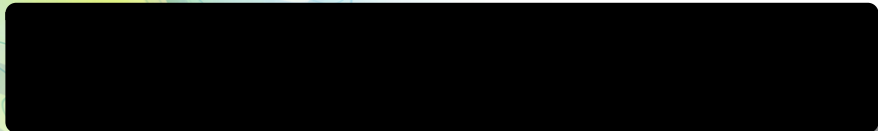
```
/* Pre: el paràmetre implícit té nota  
       i 'nota' és una nota vàlida (entre 0  
       i la nota màxima) */  
void modificar_nota(double nota);  
/* Post: la nota del paràmetre implícit  
       passa a ser 'nota' */
```

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);
```

# Part I



- 1 Abstracció i disseny modular
- 2 Descomposició funcional i per dades
- 3 Orientació a objectes
- 4 Especificació i ús de classes

## 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 sense informació

## Tipus d'operacions: *Constructores*

### Exemple 1: Constructora por defecte

```
Estudiant est;
```

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

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

Qué passa quan fem les següents declaracions?

```
vector<char> v;  
vector<char> w(10);  
vector<char> linea(10, '*');  
vector< vector<char> > M(10, vector<char>(5, '-'));
```

## Tipus d'operacions: *Destructora*

### Destructora

```
~nom_classe() { ... }
```

- En C++, una operació destructora d'objectes de la classe
- Fa operacions que puguin fer falta abans que l'objecte desaparegui
- Rarament la cridarem. No en parlarem més fins al Tema 7
- L'operació per defecte no fa res; s'aplica si no n'escrivim cap
- Podem redefinir-la
- Es crida automàticament al final de cada bloc amb les variables declarades en el bloc

## Tipus d'operacions: *Destructora*

```
while (...) {  
    Estudiant e1;  
    ...  
    if (...) {  
        Estudiant e2;  
        ...  
        // aquí es fa la crida e2.~Estudiant()  
    }  
    ...  
    // aquí es fa la crida e1.~Estudiant()  
}
```

Pregunta: qué passa si declarem `vector<Estudiant> v(10)?`

## 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

## 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 precondition 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: -- */  
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;
```

# 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 */  
  
    // Destructora: esborra automàticament els objectes  
    //              locals en sortir d'un àmbit de  
    //              visibilitat  
    ~Estudiant();
```

# 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 */
```