Disseny modular I

Ricard Gavaldà

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

Aquesta presentació no substitueix els apunts

Motivació

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

Contingut

Abstracció i disseny modular

Descomposició funcional i per dades. TADs

Orientació a objectes

Especificació i ús de classes en C++

Abstracció i disseny modular

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 es 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

```
int pot(int a, int b);
/* Pre: a > 0 i b >= 0 */
/* 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

Descomposició funcional i per dades. TADs

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 predefinits

`int`:

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

Tipus Abstracte de Dades, TAD

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 *què* fan les operacions (no *com*)

Un tipus de dades pot tenir diverses implementacions
El tipus "és" l'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 eixi de funcionar

Orientació a objectes

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
- Podem fer més accions/funcions que operen amb el tipus, però si no són dins de la classe no són mètodes

Exemple: Classe `Estudiant`

Un Estudiant es caracteritza per:

- Un DNI, que és un enter no negatiu, obligatori
- Una nota, optativa. Si en té, és un 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

```
bool canvia_np_per_zero(vector<Estudiant>& v, int dni);
/* Pre: tots els elements de v tenen dnis diferents */
/* Post: si v conté un Estudiant amb dni v, 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 */
```

Implementació

```
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

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

Implementació

```
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;
}
```

Exemple OO: parámetre implicit

Classe `Estudiant` amb una operació `te_nota`:
En C++ sense OO:

```
bool te_nota(const Estudiant &e);
/* Pre: -- */
/* Post: El resultat indica si e té nota */
```

Amb OO:

```
bool te_nota() const;
/* Pre: -- */
/* Post: el resultat indica si el paràmetre implícit
         té nota */
```

Noteu el `const` referit a l'objecte propietari

Exemple OO: crida a un mètode

Forma general:

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

- No: `b = te_nota(est);`
- Sí: `b = est.te_nota();`

Exemple OO: crida a un mètode que modifica l'objecte propietari

Mètode `modificar_nota` dins de la classe `Estudiant`

- Especificació:

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

- Nota: sense `const`
- Crida:

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

Especificació i ús de classes en C++

Especificació de classes en C++

Especificació en fitxer `.doc`

Distinció part `private` i part `public`

Conveni: Organitzar operacions en constructores/creadores, modificadores, consultores i entrada/sortida

Exemple: Fitxer `Estudiant.doc`

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

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

- Tipus de creadores que tenen el mateix nom de la classe (veure capçalera) i retornen un objecte nou d'aquest tipus
- Excepció: sense paràmetre implícit
- La llista de paràmetres permet distingir entre diverses constructores
- Constructora per defecte: sense paràmetres, crea un objecte nou sense informació

Tipus d'operacions: *Constructores*

Exemple 1: `Estudiant est;`

- crida la constructora `Estudiant()`

Exemple 2: `Estudiant est(46567987);`

- crida la constructora `Estudiant(int dni)`

Què passa quan fem les següents declaracions?

```
vector<char> v;
vector<char> v(10);
vector<char> v(10,'a');
vector<vector<char> > v(10,vector<char>(5,'a'));
```

Tipus d'operacions: *Destructora*

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

Crida: objecte.~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;
       ...
       // aqui s'insereix la crida e2.~Estudiant()
    }
    ...
    // aqui s'insereix la crida e1.~Estudiant()
}
```

Pregunta: què 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 per referència

Tipus d'operacions: *Consultores*

Exemple 1: `consultar_nota()`

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

Exemple 2: `te_nota()`

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

És necessària perquè hi ha operacions que tenen com a precondició que l'estudiant tingui o no tingui nota

Operacions `static`

- Són pròpies de la classe, no de cada objecte
- No tenen paràmetre implícit
- Exemple:

```
static double nota_maxima();
/* Pre: -- */
/* Post: el resultat es la nota maxima que
         poden tenir els elements de la classe */
```

- Crida: `Estudiant::nota_maxima()`