Disseny modular I

R. Ferrer i Cancho

Universitat Politècnica de Catalunya

PRO2 (curs 2011-2012)
Versió 0.2

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

Web oficial assignatura: `www.cs.upc.edu/~pro2/`
Contacte:

- rferrericancho@cs.upc.edu
- `http://www.cs.upc.edu/~rferrericancho/PRO2.html`

On som?

- Tema 1: Disseny modular i disseny basat en objectes:
- Durada: 2.5 sessions.

Motivació del tema (projectes grans,...).

Mòduls i classes

- Propietats desitjables dels programes
- Abstracció funcional i de dades
- Metodologia en el disseny de mòduls: distinció entre especificació i implementació
- Encapsulament de dades en llenguatges orientats a objectes (OO): classes i objectes; atributs i mètodes

Especificació i ús de la classe `Estudiant`

- Especificació
- Ús d'`Estudiant` amb vectors
  - Percentatge de presentats
  - Arrodoniments
  - Cerca per dni d'un estudiant

La filosofia de l'assignatura

Dues formes d'abordar la programació i el disseny modular

- Basat en llenguatges:
  - Presentació i comparació de llenguatges concrets.
  - Elecció d'un llenguatge concret i estudi de característiques i aplicacions.
- Basat en conceptes (abstracció):
  - Presentació de les característiques principals presents a la majoria de llenguatges d'interès per al curs.
  - Identificació de les propietats desitjables dels programes.
  - Estudi de tècniques associades a aquestes característiques per a la resolució de problemes i la construcció de programes que implementin les solucions amb les propietats desitjades.
  - Elecció d'un llenguatge concret com a exemple, cas d'estudi i eina de desenvolupament de treballs pràctics associats als conceptes introduïts al curs.

Quina és la millor orientació?

Desavantatges de l'aproximació basada en llenguatges:

- Massa llenguatges (i massa llenguatges d'un mateix tipus).

Avantatges de l'aproximació basada en conceptes:

- Comprensió més profunda.
- Adaptació més fàcil a llenguatges nous.
- Capacitat superior per entendre l'adequació d'un llenguatge a un objectiu concret.

PRO2

- Aproximació basada en conceptes restringint-nos a llenguatges imperatius orientats a objectes.
- Selecció concreta: C++ (exemples i pràctiques lab)
- Conceptes de *mòdul* (=part) i *disseny modular*: mecanismes d'orientació a objectes de C++.
- Divisió (estructuració de programes) en mòduls (=parts).

Què és un bon programa?

- Objectiu: produir programes fiables i fàcils d'entendre, modificar, mantenir i reusar.
- Motivació: minimitzar costos associats al disseny, ...manteniment,..., del programa.
- Problemes amb programes grans sense divisió en parts.

Estratègia: dividir el programa en *mòduls* (=parts).

Què es una bona descomposició modular?

- *Independència*: canvis en un mòdul no han d'obligar a modificar altres mòduls.
  Objectiu: modificabilitat, manteniment, treball en equip,...
- *Coherència interna*: els mòduls han de tenir una entitat pròpia i interactuar amb altres mòduls d'una forma simple i ben definida.
  Objectiu: programes fàcils d'entendre, reusabilitat, treball en grup,....

Cal: eines de raonament i metodologies adequats + llenguatge de programació adient.

Eina de raonament per a bones descomposicions modulars

abstracció = oblidar-se de certs detalls del problema per tal de transformar el nostre problema en un de més simple o més general.

Exemples:

- Ús de paràmetres en funcions i accions (abstraure's de valors concrets).
- *Especificació Pre/Post* en funcions i accions.

Què es una especificació Pre/Post?

- La capçalera de l'operació: resultat, nom, paràmetres, etc;
- La precondició: propietats que han de complir els paràmetres perquè l'operació faci el que està previst
- La postcondició: propietats que han de complir els resultats després de cridar l'operació, incloent-hi els paràmetres per referència si volem modificar-los.

Exemple

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

Interpretació de la pre/post

- Si les dades de l'operació satisfan la precondició $\rightarrow$ els resultats compliran la postcondició.
- Altrament: l'operació podria donar errors d'execució o guardar valors absurds als resultats, però en qualsevol cas el responsable d'aquest mal funcionament és l'usuari de l'operació.

Implicació: per a tota operació s'ha de disposar dels mitjans per poder comprovar la precondició abans de cada crida.

Regla d'ús: per usar (una funció) cal saber *què* fa però no cal saber *com* ho fa.

Exemple: arrodonir sobre un conjut de reals

```
void arrodonir(vector<double> &Creals);
/* Pre: cert */
/* Post: Creals conté els valors originals arrodonits
         a la dècima més propera */
```

Abstracció de la implementació

- Qualsevol canvi a la seva implementació que no afecti a la seva Pre/Post tampoc no afectarà el seu ús
- Especificació = contracte d'ús

Exemple: abstracció de la implementació, un pas més

- Conjunt de reals `Creals` implementat amb un vector.
- Abstracció de la implementació concreta del conjunt de reals
- Solució: definir i usar un nou tipus `ConjReals`

```
void arrodonir(ConjReals &Creals);
/* Pre: cert */
/* Post: Creals conté els valors originals arrodonits
         a la dècima més propera */
```

Tipus de descomposició o abstracció

- ***Abstracció funcional***

  Definició: "encarregar" la solució d'una part del problema a alguna operació independent, descrita simplement amb els seus paràmetres i la seva especificació i deixant la seva implementació per a futurs refinaments.
  - $\sim$ Ampliar el nostre llenguatge de programació.
  - Ja vista a PRO1.
- ***Abstracció de dades***
  - Definició: crear i afegir nous tipus de dades al nostre llenguatge.
  - $\sim$ Ampliar el nostre llenguatge de programació.
  - Especialment escaient en problemes grans.

Tipus de dades coneguts

- Nom pel tipus, que l'identifica.
- Operacions que ens permeten construir, modificar o consultar elements del tipus.

Objectius: ús dels nous tipus sigui idèntic al dels tipus existents
Mitjà: ús *independent* de les possibles implementacions particulars de les dades i les operacions.

Tipus de mòduls

- ***Mòdul de dades***: conté la definició d'un nou tipus i les seves operacions (normalment a PRO2).
- ***Mòdul funcional***: conté un conjunt d'operacions noves necessàries per resoldre algun problema o subproblema.

Detalls importants:

- En tots dos casos: ocultació de la implementació concreta de les operacions.
- Mòduls de dades: a més, ocultació de la definició del nou tipus de dades.

Com assolir la independència? (entre mòduls)

1. **Fase d'especificació:** Suposarem l'existència d'una representació i d'unes operacions per manipular-la.
   Clau: ens abstraurem de representacions concretes però assumirem un cert comportament de les operacions (*una especificació* $\sim$ *contracte d'ús del tipus de dades*).
2. **Fase d'implementació:** decidir la representació més adequada i la codificació de les operacions sobre aquesta representació.
   Clau: aquesta representació s'ha de poder canviar quan es consideri oportú sense que això afecti a l'especificació.

Mòduls de dades: classes

Defineix

- Un *tipus* com un determinat domini de valors.
- Un conjunt d'operacions que treballen sobre diversos paràmetres.

Possibilitats:

- En C estàndard o C++ (sense OO): un d'aquests paràmetres hauria de ser del tipus que s'està definint i representaria la dada sobre la qual actua l'operació.
- En la majoria de llenguatges OO: *encapsular* dades en unitats o mòduls anomenats *classes* que defineixen una estructura d'*atributs* (representació del tipus) i *mètodes* (operacions).

Objectes i classes I

- Donada una classe podem definir *objectes* del tipus de la classe.
- En programació no OO: una *variable* concreta i tipus.
- classes = patró de com han de ser els objectes d'un cert tipus.
- classe = descripció del patró
- objecte = espècimen concret d'una classe
- Exemples: classe Estudiant.

Objectes i classes II

- Un objecte no és un contenidor de dades sinó una *instància* de la classe.
- *Cada objecte és propietari dels seus atributs i mètodes*.

Implicacions:

- Com que els mètodes són considerats com a uns components més dels objectes (al igual que els atributs), aquests no hi figuren com paràmetres (*paràmetre implícit*).
- L'objecte propietari d'un mètode (que pot ser creat, consultat o modificat per aquest) no apareix explícitament a la seva capçalera.

Exemple OO: parámetre implicit

Operació `te_nota` de la classe `Estudiant`.

- En C++ sense OO:

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

- Amb 00:

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

Paràmetre implícit (per ref): objecte propietari.
Significat de `const`: evita que el mètode modifiqui objecte propietari (detecció en *temps de compilació*).

Exemple 00: crida a un mètode

Forma general:

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

Exemple:
est: objecte ja creat de la classe Estudiant
b: variable booleana

- No:

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

- Sí:

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

Exemple 00: crida a un mètode que modifiqui l'objecte propietari

Mètode `modificar_nota` dins de la classe `Estudiant`

- Especificació:

```
void modificar_nota(double nota)
/* Pre: el paràmetre implícit té nota
        i 0 <="nota"<= nota_maxima() */
/* Post: la nota del paràmetre implícit
         passa a ser "nota" */
```

  Detall: no es `const` $\rightarrow$ objecte propietari modificable.

- Crida (canvia la nota per `x`):

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

Cas concret: gestionar les notes d'una assignatura

Nou tipus de dades per emmagatzemar la informació dels estudiants (classe `Estudiant`).
Avui: Especificació i alguns casos d'ús de la classe Estudiant.
Interès:

- Introduir la notació necessària de C++ que usarem al llarg del curs.
- Punt de partida per properes sessions.
- ...

Especificació de la classe `Estudiant`

Important: classe en C++ = mètodes + atributs però

- Atributs no apareixen a l'especificació.
- Objectiu: independència entre especificació i implementació.
- Part `private`:
  - Només la descripció general del tipus.
  - Més endavant: declarar els atributs de la representació de la classe.
- Part `public`:
  - Especificació mètode: capçaleres + especificació pre/post.
  - public = info oferta a altres programes o classes per poder ser usada.

Especificació de la classe Estudiant

Veure especificacio_classe_Estudiant.pdf.

Claus especificació

- Independencia espeficació-implementació: per usar la classe `Estudiant` no cal saber
  - Com s'ha implementat el tipus `Estudiant` ni les seves operacions.
  - La gestió interna de la qüestió de tenir nota o no (valor booleà, valors negatius a la nota, ...).
- Estalvi a l'especificació pre/post:
  - A la pre: s'assumeix que tots els paràmetres de tipus Estudiant contenen elements definits, és a dir, obtinguts amb operacions sobre el resultat de la crida a constructures.
  - Operacions de lectura i escriptura: suposar que les dades llegides o escrites són correctes, amb el compromís de concretar els detalls a la implementació.

Tipus d'operacions: *Creadores d'objectes*

Definició: funcions que serveixen per crear objectes nous amb una informació mínima inicial o resultat de càlculs més complexos.
Constructores:

- Tipus de creadores que tenen el mateix nom de la classe (veure capçalera) i retornen un objecte nou d'aquest tipus.
- Es poden definir diferent versions de constructores d'una classe diferenciades per la seva llista de paràmetres.
- Constructora per defecte: sense paràmetres, crea un objecte nou sense informació.
- Exemple, la declaració

```
Estudiant est;
```

crida la constructora `Estudiant()` i produeix un estudiant sense dades tret de DNI = 0.

Tipus d'operacions: *Modificadores*

- Transformen l'objecte propietari (paràmetre implícit) amb informació aportada per altres paràmetres, si cal.
- No haurien de poder modificar altres objectes (C++ ho permet via el pas de paràmetres per referència).
- Accions (normalment)

Exemples de modificadores I

Operacions `afegir_nota` i `modificar_nota`.

- Diferencia conceptual.
- Només amb `modificar_nota` podria ser suficient (si li relaxem la precondició),
- Avantatges: fiabilitat en l'ús del tipus, llegibilitat,... (i per tant la modificabilitat i la mantenibilitat).

Tipus d'operacions: *Consultores*

- Proporcionen informació sobre l'objecte propietari (amb ajut d'informació aportada per altres paràmetres).
- Normalment funcions (tret que hagin de retornar més d'un resultat, en aquest cas poden ser accions amb més d'un paràmetre per referència.
- Exemple 1: `consultar_nota`.

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

- Exemple 2: `te_nota` (necessària perquè que hi ha operacions que tenen com a requisit (pre) que l'estudiant tingui o no tingui nota).

Exemple d'ús de la classes estudiant: percentatge de presentats.

- Volem fer un programa que donat un vector d'estudiants ens digui el percentatge de presentats, és a dir, d'estudiants amb nota.
- Clau: per usar un mòdul ha de ser suficient amb disposar de la seva especificació.
- *Especificacio*

```
double presentats(const vector<Estudiant> &vest)
/* Pre: vest conté almenys un element */
/* Post: el resultat és el percentatge de
         presentats de vest */
```

Implementació I

- Estratègia:
  - Nombre de presentats = nombre d'estudiants amb nota.
  - Esquema de recorregut.
  - Compatge progressiu: Considerem que hem comptat els estudiants amb nota fins a un punt "i" (sense incloure'l) i avançem considerant el següent (l'i-èssim).

Implementació II

```
double presentats(const vector<Estudiant> &vest)
/* Pre: vest conté almenys un element */
{
 int numEst = vest.size();
 int n = 0;
 for (int i = 0; i < numEst; ++i) if (vest[i].te_nota()) ++n;
 /* n és el nombre d'estudiants amb nota de vest */
 double pres = n*100/double(numEst);
 return pres;
}
/* Post: el resultat és el percentatge
         de presentats de vest */
```

Exemple d'ús de la classes estudiant: arrodoniments

- Enunciat:
  Programa que, donat un vector d'estudiants, el modifica arrodonint-ne les notes a la dècima més propera (es pot fer com a acció o com a funció).
- Especificació:

```
void arrodonir_notes(vector<Estudiant> &vest);
/* Pre: cert */
/* Post: vest té les notes dels estudiants arrodonides
         respecte al seu valor inicial */
```

Estratègia de disseny

- Abstracció funcional: suposem que tenim una funció que s'encarrega d'arrodonir un real.
- Disseny descendent: aquesta funció la dissenyarem al final.
- Objectiu: separar el problema tècnic de com fer l'arrodoniment d'un real, del nostre problema general que és tractar les notes d'un vector d'estudiants, arrodonint-les.

Implementació I

```
void arrodonir_notes(vector<Estudiant> &vest)
/* Pre: cert */
{
  int numEst = vest.size();
  for (int i = 0; i < numEst; ++i) {
      /* mirem si vest[i] té nota */
      if (vest[i].te_nota()) {
         /* obtenim la nota de vest[i] arrodonida */
         double aux = arrodonir(vest[i].consultar_nota());
         /* modifiquem la nota de vest[i] amb
            la nota arrodonida */
         vest[i].modificar_nota(aux);
      }
  }
}
/* Post: vest té les notes dels estudiants arrodonides
         respecte al seu valor inicial */
```

Implementació II

Comentaris:

- Ús la variable aux per augmentar la llegibilitat.
- Cal usar `modificar_nota` perquè l'estudiant ja té nota.

Implementació de la funció `arrodonir`,

```
double arrodonir(double r)
/* Pre: cert */
{
  return int(10*(r + 0.05)) / double(10);
}
/* Post: el resultat és el valor original de r arrodonit a
         la dècima més propera (i més gran si hi ha empat) */
```

Comentari: ús funció `int()` de C++.

Exemple d'ús de la classes estudiant: cerca d'estudiant per dni

- Enunciat
  Cerca per dni d'un estudiant en un vector d'estudiants.
- Especificació:

```
bool cerca_lineal(const vector<Estudiant> &vest, int dni)
/* Pre: cert */
/* Post: el resultat indica si l'estudiant
         amb DNI = dni hi és a vest */
```

Implementació

```
bool cerca_lineal(const vector<Estudiant> &vest, int dni)
/* Pre: cert */
{
  bool b = false;
  int numEst = vest.size();
  int i = 0;
  while (i < numEst and not b) {
    if (vest[i].consultar_DNI() == dni) b = true;
    else ++i;
  }
  return  b;
}
/* El resultat indica si l'estudiant
   amb DNI = dni hi és a vest */
```