Disseny modular III
Exemples

19 de febrer de 2015

1 Implementació classe `Cjt_estudiants`

1.1 `Cjt_estudiants.hh`

```
#include "Estudiant.hh"
#include <vector>

class Cjt_estudiants {

  private:
    vector<Estudiant> vest; // ordenat pels DNI dels estudiants
    int nest;
    static const int MAX_NEST = 60;

  /* Invariant de la representació:
     - vest[0..nest-1] està ordenat creixentment pels DNI dels estudiants
     - 0 <= nest <= vest.size() = MAX_NEST
  */

    void ordenar(); // només per al mètode llegir
    /* Pre: cert */
    /* Post: els elements del paràmetre implícit 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 */

  public:

    Cjt_estudiants();

    ~Cjt_estudiants();

    void afegir_estudiant(const Estudiant &est);

    void modificar_estudiant(const Estudiant &est);

    void modificar_iessim(int i, const Estudiant &est);

    int mida() const;

    static int mida_maxima(); // nova operacio

    bool existeix_estudiant(int dni) const;

    Estudiant consultar_estudiant(int dni) const;

    Estudiant consultar_iessim(int i) const;

    void llegir();

    void escriure() const;
};
```

1.2 Cjt_estudiants.cc

```
#include "Cjt_estudiants.hh"

Cjt_estudiants::Cjt_estudiants()
/* Pre: cert */
{
 nest = 0;
 vest = vector<Estudiant>(MAX_NEST);
}
```

```
/* Post: el resultat és un conjunt d'estudiants buit */

Cjt_estudiants::~Cjt_estudiants(){}

void Cjt_estudiants::afegir_estudiant(const Estudiant &est)
/* Pre: el paràmetre implícit no conté cap estudiant amb el dni d'est; el
     nombre d'estudiants del p.i. és menor que la mida màxima permesa */
{
  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;
}
/* Post: s'ha afegit l'estudiant est al paràmetre implícit */

int Cjt_estudiants::cerca_dicot(const vector<Estudiant> &vest, int left, int right, in
/* Pre: vest[left..right] està ordenat creixentment per DNI,
        0<=left, right<vest.size() */
{
  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 posició que volem
  if (found) return i;
  else return -1;

/* 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 */
}

void Cjt_estudiants::modificar_estudiant(const Estudiant &est)
/* Pre: existeix un estudiant al paràmetre implícit amb el DNI d'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 dicotòmica
  int i = cerca_dicot(vest, 0, nest-1, est.consultar_DNI());
  // i es la posició amb el DNI d'est
  vest[i] = est;
}
/* Post: l'estudiant del paràmetre implícit original amb el DNI
   d'est ha quedat substituït per est */

void Cjt_estudiants::modificar_iessim(int i, const Estudiant &est)
/* Pre: 1 <= i <= nombre d'estudiants del paràmetre implícit,
   l'element i-èssim del p.i. en ordre creixent per DNI conté
   un estudiant amb el mateix DNI que est */
{
   vest[i-1]= est;
}
/* Post: l'estudiant i-èssim del p.i. ha quedat substituït
   per est */

int Cjt_estudiants::mida() const
/* Pre: cert */
{
  return nest;
}
/* Post: el resultat és el nombre d'estudiants del paràmetre
implícit */

int Cjt_estudiants::mida_maxima() // la paraula clau static del .hh no s'ha de repetir
/* Pre: cert */
{
  return MAX_NEST;
}
/* Post: el resultat és el nombre màxim d'estudiants que pot arribar
   a tenir el paràmetre implícit */

bool Cjt_estudiants::existeix_estudiant(int dni) const
/* Pre: dni >= 0 */
{
  // apliquem la cerca dicotòmica a l'interval [0..nest-1]
```

```
  int i = cerca_dicot(vest, 0, nest-1, dni);
  return (i!=-1);
}
/* Post: el resultat indica si existeix un estudiant al paràmetre
   implícit amb DNI = dni */

Estudiant Cjt_estudiants::consultar_estudiant(int dni) const
/* Pre: existeix un estudiant al paràmetre implícit amb DNI = dni */
{
  // per la Pre, segur que trobem dni com a DNI d'algun element
  // de vest[0..nest-1]; apliquem-hi la cerca dicotòmica
  int i = cerca_dicot(vest, 0, nest-1, dni);
  // i és la posició amb DNI = dni
  return vest[i];
}
/* Post: el resultat és l'estudiant amb DNI = dni que conté el
   paràmetre implícit */

Estudiant Cjt_estudiants::consultar_iessim(int i) const
/* Pre: 1 <= i <= nombre d'estudiants que conté el paràmetre implícit */
{
  return vest[i-1];
}
/* Post: el resultat és l'estudiant i-èssim del paràmetre implícit
   en ordre creixent per DNI */

void Cjt_estudiants::llegir()
/* Pre: estan preparats al canal estàndar d'entrada un enter entre 0 i
   la mida maxima permesa, que representa el nombre d'elements que llegirem,
   i les dades de tal nombre d'estudiants diferents */
{
  cin >> nest;
  for (int i = 0; i < nest; ++i) vest[i].llegir();
  ordenar(); // noteu que l'apliquem sobre el p. i.
}
/* Post: el paràmetre implícit conté el conjunt d'estudiants llegits
   del canal estàndard d'entrada */

void Cjt_estudiants::escriure() const
/* Pre: cert */
```

```
{
 for (int i = 0; i < nest; ++i)
    vest[i].escriure();
}
/* Post: s'han escrit pel canal estàndard de sortida els estudiants del
   paràmetre implícit en ordre ascendent per DNI */
```

2 Ampliació de la classe `Cjt_estudiants`)

2.1 Solució 1: modificació de la classe enriquida, `Cjt_estudiants`

Veure doc presentació.

2.2 Solució 2 (enriquiment): `E_Cjt_estudiants.cc`

```
#include "E_Cjt_estudiants.hh"

void esborrar_estudiant(Cjt_estudiants &Cest, int dni)
/* Pre: existeix un estudiant a Cest amb DNI = 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;
}
/* 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 */
{
  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);
}
/* Post: el resultat és l'estudiant de Cest amb nota màxima;
   si en té més d'un, és el de dni més petit */
```