Implementación de clases

Para implementar una clase (*) usamos dos ficheros:

- `clase.hh` atributos de la clase y las cabeceras de las operaciones -> destinado a ser incluido en los programas que usan la clase
- `clase.cc` código de las operaciones -> destinado a ser compilado y linkar su `.o` con los de las demás clases y el `main`

Ejemplos:

- `Estudiant (.hh y .cc)`
- `Cjt_estudiants (.hh y .cc)`

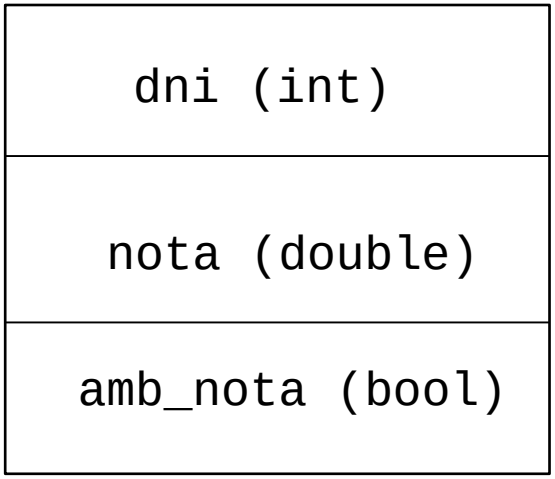
(*) Veremos excepciones al final del curso

Implementación de clases: atributos

- Los atributos de una clase se representan mediante una lista de campos
- Los campos pueden ser tipos simples, `structs` u otras clases
- Se escriben en la parte `private` del fichero `.hh`, para que solo sean accesibles por la implementación

Campos de Estudiant

Nuestra primera versión de `Estudiant` tiene tres campos: `dni`, `nota` y `amb_nota` (más uno `static: MAX_NOTA`)



Campos de Estudiant

```
class Estudiant { // fichero Estudiant.hh

private:
  int dni;
  double nota;
  bool amb_nota;
  static const int MAX_NOTA = 10;
```

- **`static`:** todos los estudiantes tienen el mismo `MAX_NOTA`
- **`const`:** no se puede cambiar su valor
- Distintos programas pueden manejar distintas notas máximas editando el `.hh`

Invariante de la representación

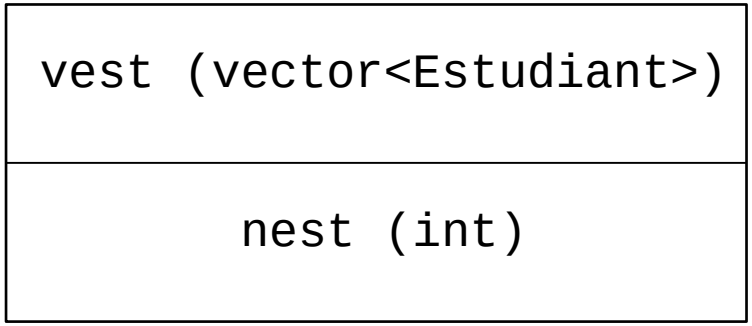
- Propiedades que han de cumplir los valores de los campos para dar lugar a objetos válidos de la clase
- Sólo intervienen en la implementación de las operaciones
- Se pueden suponer como parte de la **pre** de cada operación pública de la clase
- Se han de suponer como parte de la **post** de cada operación pública de la clase

Para la clase Estudiant:

- `0 <= dni`
- si `amb_nota` entonces `0 <= nota <= MAX_NOTA`

Campos de Cjt_estudiants

Nuestra primera versión de `Cjt_estudiants` tiene tres campos: un `vector` de `Estudiant`, un entero (el tamaño del conjunto en cada momento) y el tamaño máximo permitido, que será `static`



Campos de Cjt_estudiants

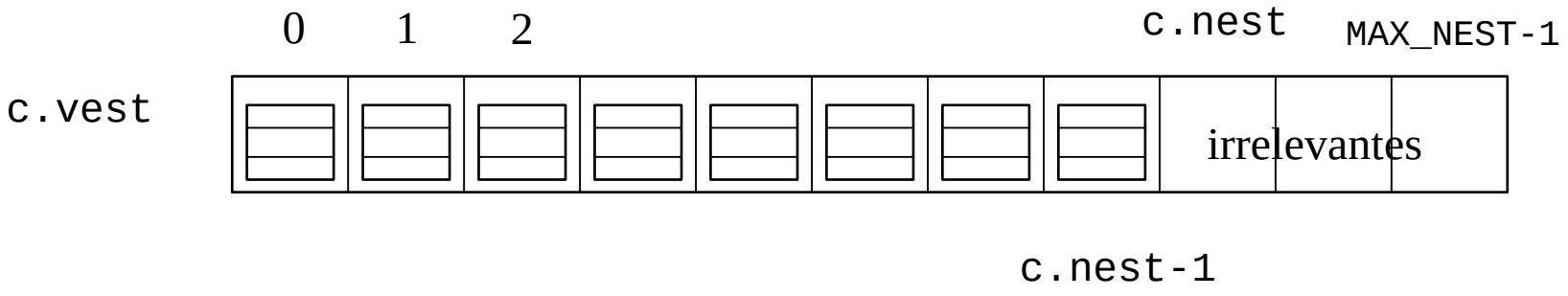
```
class Cjt_estudiants { // fichero .hh

private:
   vector<Estudiant> vest;
   int nest;
   static const int MAX_NEST = 20;
```

- Distintos programas pueden manejar distintas tamaños máximos editando el `.hh`

Campos de Cjt_estudiants

Si representamos un conjunto `c` con más detalle veremos



Por ejemplo, para obtener el DNI del tercer estudiante de `c`, si estamos dentro la clase podremos hacer

```
c.vest[2].consultar_DNI()
```

pero si estamos fuera, tendremos que hacer

```
c.consultar_iessim(3).consultar_DNI()
```

Invariante de la representación

Para la clase `Cjt_estudiants`:

- `0 <= nest <=  vest.size() = MAX_NEST`

(significa que el conjunto no está lleno)

- `vest[0..nest-1]` está ordenado crecientemente por el DNI de los estudiantes

Implementación de clases: operaciones

- **Públicas**: se usarán fuera de la clase
  - sus cabeceras aparecen en la parte `public` del `.hh`
  - normales (necesitan un parámetro implícito: la mayoría)
  - `static` (no necesitan un parámetro implícito: pocas)

- **Privadas**: se usarán solamente dentro de la clase (auxiliares de las públicas)
  - sus cabeceras aparecen en la parte `private` del `.hh`
  - normales: pocas
  - `static`: la mayoría

Implementación de clases: operaciones

- El código de las operaciones publicas y privadas se escribe en el fichero `.cc`. que ha de comenzar por un `#include` de su correspondiente `.hh`

```
#include “Estudiant.hh”
```

- El nombre de cada operación ha de ir precedido del nombre de la clase, separado por ::

```
double Estudiant::consultar_nota() const
```

Operaciones de Estudiant

Si dentro de una operación nos referimos a un campo de una clase **sin acompañarlo de un objeto**, se trata del correspondiente campo del parámetro implícito

```
double Estudiant::consultar_nota() const
{
  return nota; // es la nota del param. impl.
}
```

Operaciones de Estudiant

Salvo el parámetro implícito, los campos de los objetos de una clase se manejan como los de un `struct`

Ejemplo: si añadimos un constructora copiadora a `Estudiant:`

```
Estudiant::Estudiant(const Estudiant& e)
{
  dni = e.dni;
  nota = e.nota;
  amb_nota = e.amb_nota;
}
```

Operaciones de Estudiant

Si necesitamos referirnos al parámetro implícito de forma explícita, usamos la palabra `this`

```
void Estudiant::afegir_nota(double nota)
{
  this->nota = nota;
  /* una notació equivalent alternativa és
     (*this).nota = nota; */
  amb_nota = true;
}
```

Operaciones de Cjt_estudiants

Hemos definido dos privadas, una normal y otra `static`

```
void ordenar_cjt_estudiants ()
/* 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 per DNI
   creixentment, 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 és -1 */
```

Operaciones de Cjt_estudiants

`ordenar_cjt_estudiants` solo se usa para poder leer un conjunto y que quede ordenado sin que el usuario lo tenga que ordenar previamente

```
void Cjt_estudiants::llegir(){
  cout << "Escriu la mida del conjunt i els elements"
  << endl;
  cin >> nest;
  for (int i=0; i<nest; ++i) vest[i].llegir();
  ordenar_cjt_estudiants();
}
```

Operaciones de Cjt_estudiants

`cerca_dicot` se usa para saber si un estudiante con un cierto DNI está o no el conjunto y determinar su posición

```
bool Cjt_estudiants::existeix_estudiant(int dni) const {
  int i = cerca_dicot(vest,0,nest-1,dni);
  return (i!=-1);
}

Estudiant Cjt_estudiants::consultar_estudiant(int dni) const
{
  int i = cerca_dicot(vest,0,nest-1,dni);
  // el DNI de vest[i] és “dni”
  return vest[i];
}
```

Ejercicios de implementación

- Añadir una operación a `Estudiant` que haga lo mismo que la operación `redondear_e_a` del fichero `red1.cc`

```
void redondear_e_a(Estudiant & est)
```

pasa a

```
void redondear_e_a()
```

- Añadir una operación a `Cjt_estudiants` que haga lo mismo que la operación `presentats` del fichero `presentats_conj.cc`

```
double presentats (const Cjt_estudiants  &c)
```

pasa a

```
double presentats () const
```