Tema 2. Herència
Sessió 5

LP 2019-20

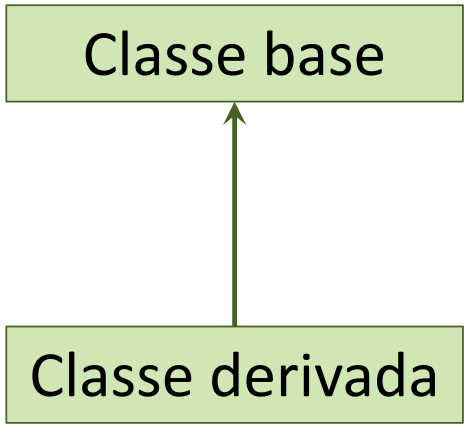
Exercici

Volem crear un conjunt de classes per poder guardar tots els productes del catàleg d'una empresa de venda de productes online. Al catàleg hi podria haver una varietat molt gran de productes. De moment, només considerarem llibres i electrodomèstics. Tots els productes tenen un codi que els identifica i un preu de venda. Dels llibres, a més a més, volem guardar el títol, l'autor i el nº de pàgines. Dels electrodomèstics, la marca, el model i el volum que ocupa el seu embalatge.

- Declareu, **utilitzant herència**, el conjunt de classes necessari per guardar aquesta informació:
  - Afegiu els atributs necessaris a cada classe.
  - Com a mètodes, declareu només mètodes `getXXX` i `setXXX` per recuperar i modificar el valor dels atributs.

Herència

- L'**herència** és un mecanisme de la POO que permet reutilitzar eficientment codi ja implementat
- L'herència es justifica quan volem fer l'especialització d'una classe



- La **classe derivada** (o **subclasse**) es crea com una especialització de la **classe base** (o **superclasse**)
  - La classe derivada hereda les característiques (atributs i mètodes) de la classe base
  - A més a més, conté les seves característiques (atributs i mètodes) particulars
  - NO és una simple instància de la classe existent

Exemple

```
class Persona
{
public:
   Persona() {}
   ~Persona() {}
   string getNiu() const { return m_niu; }
   string getNom() const { return m_nom; }
   void setNiu(const string& niu) { m_niu = niu; }
   void setNom(const string& nom) { m_nom = nom; }
private:
   string m_niu;
   string m_nom;
};
```

Classe base:

- Conté atributs i mètodes comuns

Classe derivada:

- Conté atributs i mètodes específics
- Hereda (i per tant també conté) atributs i mètodes comuns de la classe base

```
#include "Persona.h"
class Estudiant : public Persona
{
public:
   Estudiant() {}
   ~Estudiant() {}
   void setTitulacio(const string& titulacio) {   m_titulacio = titulacio; }
   string getTitulacio() const { return m_titulacio; }
private:
   string m_titulacio;
};
```

Indica que deriva de la classe Persona

Exemple

```
class Persona
{
public:
   Persona() {}
   ~Persona() {}
   string getNiu() const { return m_niu; }
   string getNom() const { return m_nom; }
   void setNiu(const string& niu) { m_niu = niu; }
   void setNom(const string& nom) { m_nom = nom; }
private:
   string m_niu;
   string m_nom;
};
```

Classe base:

- Conté atributs i mètodes comuns

Classe derivada:

- Conté atributs i mètodes específics
- Hereda (i per tant també conté) atributs i mètodes comuns de la classe base

```
#include "Persona.h"
class Professor : public Persona
{
public:
   Professor() {}
   ~Professor() {}
   void setDepartament(const string& departament) { m_departament = departament; }
   string getDepartament() const { return m_departament; }
private:
   string m_departament;
};
```

Indica que deriva de la classe Persona

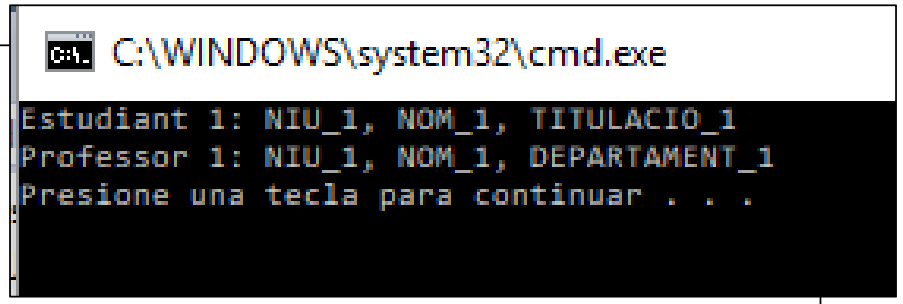
Exemple

```
int main()
{
  Estudiant e1;
  e1.setNiu("NIU_1");
  e1.setNom("NOM_1");
  e1.setTitulacio("TITULACIO_1");
  cout << "Estudiant 1: " << e1.getNiu() << ", " << e1.getNom() << ",    " <<
       e1.getTitulacio() << endl;

  Professor p1;
  p1.setNiu("NIU_1");
  p1.setNom("NOM_1");
  p1.setDepartament("DEPARTAMENT_1");
  cout << "Professor 1: " << p1.getNiu() << ", " << p1.getNom() << ", " <<
       p1.getDepartament() << endl;
}
```

Mètodes comuns de la classe base Persona

Mètodes específics de la classe derivada Estudiant



Herència: constructors i destructors

```
int main()
{
   Estudiant e1;
   e1.setNiu("NIU_1");
   e1.setNom("NOM_1");
   e1.setTitulacio("TITULACIO_1");
   cout << "Estudiant 1: " << e1
      e1.getNom() << ", " << e1.

   Estudiant e2(e1);
   cout << "Estudiant 2: " << e2
      e2.getNom() << ", " << e2.

   Estudiant e3("TITULACIO_2");
   cout << "Estudiant 2: " << e3
      e3.getNom() << ", " << e3.
}
```

```
C:\WINDOWS\system32\cmd.exe
Inicialitzacio constructor per defecte
=====
Constructor Defecte Persona
Constructor Defecte Estudiant
Estudiant 1: NIU_1, NOM_1, TITULACIO_1

Inicialitzacio constructor copia
=====
Constructor Defecte Persona
Constructor Copia Estudiant
Estudiant 2: , , TITULACIO_1

Inicialitzacio constructor parametres
=====
Constructor Defecte Persona
Constructor Parametres Estudiant
Estudiant 2: , , TITULACIO_2

Destruccio objectes
=====
Destructor Estudiant
Destructor Persona
Destructor Estudiant
Destructor Persona
Destructor Estudiant
Destructor Persona
Presione una tecla para continuar . . .
```

```
lass Persona

ublic:
   Persona() : m_niu(""), m_nom("") {}
   Persona(const string &niu, const string &nom):
      m_niu(niu), m_nom(nom) {}
   Persona(const Persona& p)
      { m_niu = p.m_niu; m_nom = p.m_nom; }
   ~Persona() {}
```

```
ass Estudiant : public Persona

blic:
   Estudiant() : m_titulacio("") {}
   Estudiant(const string &titulacio) :
      m_titulacio(titulacio) {}
   Estudiant(const Estudiant& e)
      { m_titulacio = e.m_titulacio;}
   ~Estudiant() {}
```

Constructors:

1. Crida al constructor per defecte de la classe base
2. Crida al constructor específic de la classe derivada

Destructors:

1. Crida al destructor de la classe derivada
2. Crida al destructor de la classe base

Herència: constructors i destructors

Com ho hem de fer si no volem cridar al constructor per defecte de la classe base i volem cridar al constructor amb paràmetres o al constructor de còpia?

- Hem de posar la crida explícita que calgui dins del constructor de la classe derivada

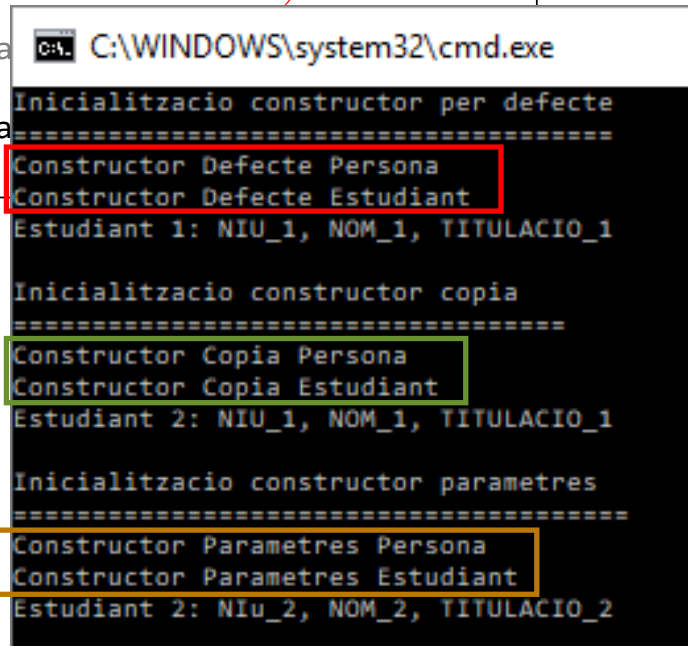
```
class Estudiant : public Persona
{
public:
   Estudiant() : m_titulacio("") {}
   Estudiant(const string &niu, const string &nom, const string &titula
                  m_titulacio(titulacio) {}
   Estudiant(const Estudiant& e): Persona(e) { m_titulacio = e.m_titula
   ~Estudiant() {}
```

Afegim paràmetres per incialitzar classe base

Inicialitzem classe base

Cridem constructor de còpia classe base

```
class Persona
{
public:
   Persona() : m_niu(""), m_nom("") {}
   Persona(const string &niu, const string &nom):
      m_niu(niu), m_nom(nom) {}
   Persona(const Persona& p)
      { m_niu = p.m_niu; m_nom = p.m_nom; }
   ~Persona() {}
```



Exercici

- Implementeu constructors per defecte, amb paràmetres i de còpia per les classes `Producte`, `Llibre` i `Electrodomestic`

Exercici

- Volem afegir un mètode **`mostra`** a les classes `Producte`, `Llibre` i `Electrodomestic` per mostrar les dades per pantalla de cadascuna de les classes
  - On l'hem de declarar?

Herència: Redefinició de mètodes

```
class Llibre: public P
{
public:
   ...
   void mostra();
   ...
```

- Mètode de la classe derivada
- Mostra informació específica de la classe derivada
- Substitueix la definició de `mostra` de la classe base

```
class Producte
{
public:
   ...
   void mostra();
   ...
```

- Mètode de la classe base
- Mostra informació comuna de la classe base

```
void Llibre::mostra()
{
   Producte::mostra();
   cout << "Titol: " << m
   cout << "Autor: " << m
   cout << "N. pagines: " << m_nPagines << endl;
}
```

- Crida explícita a `mostra` de la classe base
- S'ha de posar si volem mostrar informació de la classe base

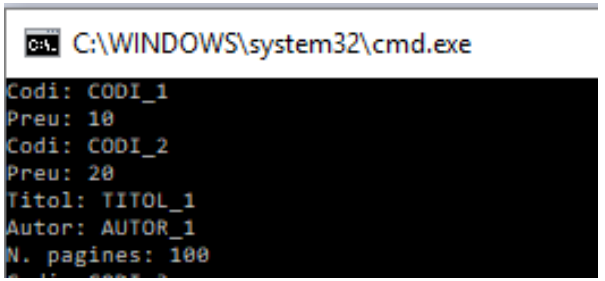
```
void Producte::mostra()
{
   cout << "Codi: " << m_codi << endl;
   cout << "Preu: " << m_preu << endl;
}
```

```
Producte p;
...
p.mostra();

Llibre l;
...
l.mostra();
```

Crida a `mostra` de la classe base

Crida a `mostra` de la classe derivada



Exercici

- Volem afegir un mètode `calculaDespeseEnviament` per determinar les despeses d'enviament d'un producte. Les despeses d'enviament es calculen de la forma següent:
  - Per tots els productes, si el preu del producte (sigui del tipus que sigui) és inferior a 100€ s'aplica una tarifa fixa de 1€. Si el preu del producte és superior a 100€ s'aplica un percentatge de l'1% sobre el preu del producte amb un màxim de 5€
  - A més a més, pels llibres, si el nº de pàgines és superior a 500€ s'aplica un sobrecost d'1€ sobre les despeses calculades segons el punt anterior.
  - Pels electrodomèstics s'hi afegeix un sobrecost d'1€ per cada 20 litres (o fracció) del seu volum.

Exercici

- Volem afegir un mètode `calculaPreu` per determinar el preu final d'una comanda d'n unitats d'un producte, on n és un paràmetre del mètode.
  - El preu es calcula com el preu base del producte més les despeses d'enviament, però amb la possibilitat d'aplicar un descompte que depèn del tipus de producte:
    - Pels llibres, si la comanda és superior a 10 unitats, s'aplica un descompte del 5%, i si és superior a 100 unitats un descompte del 10%.
    - En els cas dels electrodomèstics, s'aplica un descompte del 10% si es compra més d'una unitat.

Herència: accés a les dades de les classes base i derivada

```
float Llibre::calculaPreu(int nUnitats)
{
   float preu = (m_preu * nUnitats) + calculaDespesesEnviament();
   if (nUnitats > 10)
      preu -= 0.05 * preu;
   else
      if (nUnitats > 100)
         preu -= 0.1 * preu;
}
```

Error: Accés a la part privada de la classe base

- Una classe derivada no té accés a les dades privades de la classe base
- Si ens interessa accedir a la part privada de la classe base:
  - Declarar els atributs/mètodes a la classe base com a **`protected`**:
    - Les classes derivades hi poden accedir com si fossin `public`
    - La resta de classes no derivades ho segueixen veient com a `private` i per tant, no hi poden accedir

Herència: accés a les dades de les classes base i derivada

El procés d'herència es pot fer de dues maneres:

- `public`

```
class Llibre: public Producte
```

- `private`

```
class Llibre: private Producte
```

		Herència **`public`**		Herència **`private`**	
Qualificació a la **classe base**	**Accés** des de **classes no derivades**	**Accés** des de **classe derivada**	**Qualificació** a la **classe derivada**	**Accés** des de **classe derivada**	**Qualificació** a la **classe derivada**
`private`	no accessible	no accessible	`private`	no accessible	`private`
`protected`	no accessible	accessible	`protected`	accessible	`private`
`public`	accessible	accessible	`public`	accessible	`private`

Herència: accés a les dades de les classes base i derivada

Exemple

```
class Producte
{
public:
   string getCodi() { return m_codi; }
   float getPreu() { return m_preu; }
   ...
```

```
...
private:
   string m_codi;
protected:
   float m_preu;
};
```

class Llibre: public Producte

```
void Llibre::mostra() const
{
   cout << "Codi": " << m_codi << endl;
   cout << "Preu": " << m_preu << endl;
   ...
}
```

```
int main()
{
   Llibre l;
   cout << "Codi": " << l.getCodi() << endl;
   cout << "Preu": " << l.getPreu() << endl
}
```

class Llibre: private Producte

```
void Llibre::mostra() const
{
   cout << "Codi": " << m_codi << endl;
   cout << "Preu": " << m_preu << endl;
   ...
}
```

```
int main()
{
   Llibre l;
   cout << "Codi": " << l.getCodi() << endl;
   cout << "Preu": " << l.getPreu() << endl;
}
```