Problemes LP – Curs 2018-19
Tema 2: Herència

Exercici 1 (nivell mig)

Utilitzant herència volem crear una jerarquia de classes que ens permeti guardar i gestionar dades dels alumnes i professors de la UAB.

1. Per començar, haurem de declarar una classe `Persona` que contingui la informació comuna d'alumnes i professors. D'aquesta classe base, derivarem després classes específiques per alumnes i professors.

   La classe `Persona` haurà de guardar el niu i el nom de la persona (ja sigui un estudiant o un professor) i la llista d'assignatures a les que està vinculat (en el cas d'estudiants seran les assignatures de les que està matriculat i en el cas de professors, les assignatures en les que imparteix docència).

   La interfície pública de la classe Persona ha de ser la següent:

```
public:
   Persona();
   Persona(const string &niu, const string &nom);
   ~Persona();

   string getNiu() const;
   string getNom() const;
   bool cercaAssignatura(const string &nomAssignatura);
   bool afegeixAssignatura(const Assignatura& assignatura);
```

   És a dir, a més a més dels constructors i del destructor i de mètodes per recuperar el nom i el niu, ha de tenir un mètode `cercaAssignatura` per comprovar si una determinada assignatura identificada pel nom està dins de la llista d'assignatures de la persona (retorna cert si hi és i fals en cas contrari) i un mètode `afegeixAssignatura` per afegir les dades d'una assignatura a la llista d'assignatures (retorna fals si l'assignatura ja estava a la llista i, en aquest cas, no la torna a afegir).

   Per guardar la llista d'assignatures podeu fer servir les classes `list` o `forward_list` de la llibreria estàndard STL. Per guardar la informació de les assignatures us donem ja feta una classe `Assignatura` que conté les dades i els mètodes que necessitem per cada assignatura:

```
class Assignatura
{
public:
   Assignatura() :
      m_nom(""), m_nCreditsEstudiant(0), m_nCreditsProfessor(0) {};
   Assignatura(const string& nom, const int nCreditsEstudiant,
               const int nCreditsProfessor) :
               m_nom(nom), m_nCreditsEstudiant(nCreditsEstudiant),
               m_nCreditsProfessor(nCreditsProfessor) {}
   ~Assignatura() {};
```

```
    string getNom() const { return m_nom; }
    int getCreditsEstudiant() const { return m_nCreditsEstudiant; }
    int getCreditsProfessor() const { return m_nCreditsProfessor; }
private:
    string m_nom;
    int m_nCreditsEstudiant;
    int m_nCreditsProfessor;
};
```

Com podem veure, de cada assignatura guardem el nom i el nº de crèdits que aquesta assignatura li representa a un alumne i a un professor (que no necessàriament han de tenir el mateix valor).

2. Derivar una classe `Estudiant` a partir de la classe base `Persona` de l'apartat anterior. A part de tota la informació derivada de `Persona`, haurà de tenir un nou atribut per guardar la titulació a la que està matriculat l'estudiant.

   La interfície pública de la classe `Estudiant` (apart dels mètodes heretats de Persona) haurà de ser la següent:

```
public:
    Estudiant();
    Estudiant(const string &niu, const string &nom, const string
&titulacio);
    ~Estudiant();

    string getTitulacio() const;
    int getCreditsMatriculats() const;
```

   El mètode `getCreditsMatriculats` haurà de retornar la suma de crèdits-estudiant de totes les assignatures de les que està matriculat.

3. Derivar una classe `Professor` a partir de la classe base `Persona` de l'apartat anterior. A part de tota la informació derivada de `Persona`, haurà de tenir atributs per guardar el departament en el que està adscrit i el nº màxim de crèdits de docència que hauria de fer.

   La interfície pública de la classe `Professor` (apart dels mètodes heretats de Persona) haurà de ser la següent:

```
public:
    Professor();
    Professor(const string &niu, const string &nom,
                const string &departament, const int maximCredits);
    ~Professor() {}

    int getMaximCredits() const;
    string getDepartament() const;
    int getCreditsImpartits() const;
```

   El mètode `getCreditsImpartits` haurà de retornar la suma de crèdits-professor de totes les assignatures en les que imparteix docència.

Exercici 2 (nivell avançat)

Una classe pot derivar més d'una classe base al mateix temps. És el que s'anomena herència múltiple. En aquest cas, la classe derivada hereta tots els atributs i mètodes de totes les classes base i a més a més pot afegir els seus propis atributs i mètodes com a l'herència simple. La sintaxi per indicar l'herència múltiple és la següent:

```
class classeDerivada : public classeBase1, public classeBase2, ...
```

En aquest exercici veurem un exemple d'herència múltiple. Volem modificar la jerarquia de classes de l'exercici anterior per incloure informació dels diferents grups de classe que té una assignatura. Per això, seguirem els passos següents:

1. Declarar una classe `Grup` que contingui com a atributs un nº de grup i un string que permeti guardar l'horari de les classes. La interfície pública de la classe ha de ser la següent:

```
public:
   Grup();
   Grup(int nGrup, const string &horari);
   ~Grup();

   int getGrup() const;
   string getHorari() const;
```

2. Declarar una classe `GrupClasse` que sigui derivada la classe `Grup` i de la classe `Assignatura` de l'exercici anterior. A part de tota la informació derivada d'aquestes dues classe, la nova classe haurà de tenir un atribut que permeti guardar el nº de crèdits que pel professor representa fer aquell grup de l'assignatura (que podrà tenir un valor diferent del nº de crèdits del professor guardats a la classe `Assignatura`). La interfície pública de la classe ha de ser la següent:

```
public:
   GrupClasse();
   GrupClasse(const string &nom, const int nCreditsEstudiant,
         const int nCreditsProfessor, int nGrup,
         const string &horari, const int nCreditsProfessorGrup);
   ~GrupClasse();

   int getCreditsProfessor();
```

   En el constructor `nCreditsProfessor` és el nº de crèdits totals de l'assignatura i `nCreditsProfessorGrup` és el nº de crèdits que suposa fer només aquest grup de classe. El mètode `getCrèditsProfessor` redefineix el mètode original de la classe `Assignatura` i ha de retornar el nº de crèdits-professor del grup.

3. Modificar la declaració de la classe `Persona` per convertir la llista de assignatures en una llista d'objectes de la nova classe `GrupClasse`. Fer totes les modificacions necessàries als mètodes de les classes `Persona`, `Estudiant` i `Professor` per adaptar-se a aquest canvi.

Exercici 3 (nivell mig)

Volem utilitzar la classe `LlistaProductes` que hem anat desenvolupant durant les sessions de classe dins d'un programa que gestioni totes les vendes d'una cadena comercial. Les vendes dels productes es poden fer presencialment en una botiga física de la cadena o bé de forma online, per internet. El tipus d'informació que necessitem guardar és diferent segons el tipus de venda.

Per les vendes que es fan presencialment a una botiga necessitem guardar el codi del producte, el nº d'unitats que s'han comprat, la data de la compra, l'import total i un identificador de la botiga on s'ha fet la compra.

Per les vendes que es fan per internet, apart del codi i nº d'unitats del producte, la data de la compra i l'import total (igual que per les compres presencials), necessitem guardar l'adreça d'enviament i les despeses d'enviament.

- Crear i implementar amb herència la jerarquia de classes necessària per guardar els diferents tipus de vendes amb la informació que s'ha descrit anteriorment i els mètodes que facin falta per poder implementar el que s'explica a continuació.
- Crear i implementar una classe `GestioVendes` que inclogui un llistat de tots els productes del catàleg (utilitzant la classe `LlistaProductes` que hem fet a les sessions de classe) i un llistat únic de totes les vendes (ja siguin presencials o online) que es realitzen. Aquesta classe haurà de tenir a la interfície pública, almenys els mètodes següents:
    - Un operador d'assignació
    - Un mètode `llegeixProductes` que permeti llegir d'un fitxer la informació de tots els productes del catàleg:

      ```
      void llegeixProductes(const string& nomFitxer);
      ```

      El fitxer tindrà el format següent:

      TIPUS_PROD_1 // (suposem Llibre)
      CODI_1
      PREU_ 1
      TITOL_1
      AUTOR_1
      N_PAGINES_1
      TIPUS_PROD_2 // (suposem Electrodomèstic)
      CODI__2
      PREU_ 2
      MARCA_2
      MODEL_2
      VOLUM_2
      ...

El tipus de producte serà 0 si el producte correspon a un llibre i 1 si correspon a un electrodomèstic. Per cada llibre, apart del codi i del preu tindrem el títol, l'autor i el nº de pàgines i pels electrodomèstics, apart del codi i del preu tindrem la marca, el model i el volum de l'embalatge.

- Un mètode `afegeixVendaPresencial` que permeti afegir les dades d'una nova venda presencial, rebent com a paràmetre el codi i nº d'unitats del producte, la data de la compra i l'identificador de la botiga. El mètode ha de calcular i retornar l'import total de la compra segons el preu del producte i el nº d'unitats. Si el codi de producte no existeix a la llista de productes ha de retornar -1.

```
float afegeixVendaPresencial (const string& codiProducte,
   int nUnitats, const string& data, const string& botiga)
```

- Un mètode `afegeixVendaOnLine` que permeti afegir les dades d'una nova venda per internet, rebent com a paràmetre el codi i nº d'unitats del producte, la data de la compra i l'adreça d'enviament. El mètode ha de calcular les despeses d'enviament i l'import total i retornar l'import total de la compra. Si el codi de producte no existeix a la llista de productes ha de retornar -1.

```
float afegeixVendaOnLine (const string& codiProducte,
   int nUnitats, const string& data, const string& adreca)
```

- Un mètode `escriuVendes` que permeti escriure en un fitxer les dades de totes les vendes que s'han fet.

```
void escriuVendes(const string& nomFitxer);
```

El fitxer de sortida haurà de tenir el format següent:

CODI_PRODUCTE_1
N_UNITATS_1
DATA_1
IMPORT_TOTAL_1
BOTIGA_1 (si venda presencial)
CODI_PRODUCTE_2
N_UNITATS_2
DATA_2
IMPORT_TOTAL_2
DESPESES_ENVIAMENT_2 (si venda online)
ADREÇA_ENVIAMENT_2 (si venda online)
...

Com podem veure, les dades que s'han d'escriure al fitxer són diferents en funció del tipus de venda.

Us donem el codi ja implementat de la classe `LlistaProductes` que hem fet a les sessions de classe. Només hem modificat el codi del mètode `calculaPreu` per reflectir que el preu s'ha de calcular amb o sense despeses d'enviaments segons si el tipus de venda és online o presencial. Si necessiteu afegir-hi algun mètode (per exemple per recuperar les dades d'un producte a partir del seu codi), ho podeu fer.

Alternativament, si voleu, podríeu utilitzar la classe `LlistaPolimorfisme` genèrica que us proposem implementar a l'exercici 5 per guardar i gestionar tant la llista de productes com la llista de vendes.

Exercici 4 (nivell mig)

Volem gestionar tots els préstecs que es fan de les publicacions que tenim guardades en una biblioteca. Les publicacions de la biblioteca poden ser de dos tipus, llibres o revistes periòdiques. Dels llibres hem de guardar el títol, l'autor, el nº de còpies que hi ha a la biblioteca del llibre i el nº de dies que es pot prestar (el nº de dies de préstec es fixa per cada llibre en funció de la demanda que té). De les revistes periòdiques hem de saber el títol i la periodicitat (mensual, trimestral o anual). Totes les publicacions (llibres o revistes) tenen un codi que les identifica. Les revistes a més a més tenen exemplars. De cada exemplar s'ha de guardar el nº d'exemplar que l'identifica. Els préstecs de les revistes només es poden fer d'exemplars particulars. Suposarem que dels exemplars de les revistes en tenim només una còpia.

Apart de les dades de totes les publicacions volem guardar també les dades de tots els préstecs que s'han fet. De cada préstec s'ha de guardar un identificador de l'usuari que fa el préstec, el codi de la publicació, la data de préstec i la data en què s'ha de fer el retorn de la publicació. La data de retorn depèn del tipus de publicació. Pels llibres, ha de ser la data de préstec més el nº de dies de préstec que es guarda per cada llibre. Pels exemplars de les revistes el nº de dies de préstec sempre és igual a 30.

Heu de crear i implementar una classe `Biblioteca` que permeti gestionar tota la informació de les publicacions i dels préstecs. Aquesta classe ha de tenir un llistat conjunt de publicacions independentment del seu tipus. Haurà de tenir, almenys els mètodes següents:

- Un mètode `llegeixPublicacions` que permeti llegir d'un fitxer la informació de totes les publicacions de la biblioteca:

  ```
  void llegeixPublicacions(const string& nomFitxer);
  ```

  El fitxer tindrà el format següent:

  TIPUS_PUBLICACIO_1 // (suposem Llibre)
  CODI_1
  TITOL_1
  AUTOR_1
  N_COPIES_1
  N_DIES_1
  TIPUS_PUBLICACIO_2 // (suposem revista)
  CODI_2
  TITOL_2
  PERIODICITAT_2
  N_TOTAL EXEMPLARS_REVISTA
  Nº_EXEMPLAR_1 Nº_EXEMPLAR_2 Nº_EXEMPLAR_3 ....
  ...

  El tipus de publicació serà 'L' si la publicació correspon a un llibre i 'R' si correspon a un exemplar d'una revista. Fixem-nos que per totes les publicacions hem de llegir el codi i el títol. Pels llibres, a més a més, haurem de llegir l'autor, el nº de còpies

que hi ha a la biblioteca d'aquest llibre i el nº de dies que es pot deixar en préstec. Per les revistes, llegirem la periodicitat i després tenim el nº total d'exemplar de la revista que hi ha a la biblioteca i una línia amb tots els números d'exemplar que es tenen, separats per coma.

- Un mètode `prestar` que permeti afegir les dades d'un nou préstec a la llista de préstecs, rebent com a paràmetre l'identificador d'usuari, el codi de la publicació i la data del préstec i només en les revistes el nº d'exemplar que s'ha de prestar. Fixeu-vos que el paràmetre `nExemplar` està declarat amb un valor per defecte de 0, de forma que quan estem fent el préstec d'un llibre no caldrà donar-li valor explícit (automàticament serà 0 per defecte). El mètode ha de calcular i retornar al paràmetre per referència `dataRetorn` quina és la data esperada de retorn de la publicació, calculada segons si és un llibre o un exemplar d'una revista segons s'ha explicat abans.
  Si el codi de la publicació no existeix dins del catàleg de la biblioteca s'ha de retornar `false`. Igualment si existeix però no es pot prestar també retornarà `false`. Pels llibres haurem de controlar que el nº de préstecs actius no sigui superior al nº de còpies del llibre. Pels exemplars de revista que no estigui ja prestat (només en tenim una còpia). En qualsevol altre cas retornarà `true`.

```
bool prestar(const string& idUsuari, const string& codi,
      const Data& dataPrestec, Data& dataRetorn,
      int nExemplar = 0);
```

- Un mètode `retornar` que permeti registrar que es retorna un préstec i l'elimini de la llista de préstecs. Aquest mètode rep com a paràmetres tots els valors que permeten identificar un préstec: identificador d'usuari, codi de publicació i nº d'exemplar (només en el cas d'exemplar de revistes, agafa el valor per defecte 0 en el cas dels llibres). Rep també com a paràmetre la data en què es fa el retorn de la publicació. Dins del mètode aquesta data s'ha de comparar amb la data prevista de retorn del préstec (calculada quan s'afegeix el préstec) i si hem superat la data límit s'ha de retornar `false` al paràmetre per referència `dataCorrecta`. Si no, s'hi ha de posar `true`.
  Si el préstec (identificador d'usuari, codi de publicació i nº exemplar si cal) no existeix a la llista de préstecs o si el préstec no es pot retornar perquè la publicació no estava prestada s'ha de retornar `false` com a valor de retorn de la funció. En cas contrari `true`.

```
bool retornar(const string& idUsuari, const string& codi,
           const Data& data, bool &dataCorrecta,
           int nExemplar = 0);
```

Us donem el codi ja implementat de la classe `Data` que us vam demanar a l'exercici 2 del tema de repàs de MP. Aquesta classe us ha de servir per guardar i calcular les dates de préstec i retorn.

Exercici 5 (nivell mig)

Volem implementar una classe que tingui una funcionalitat molt semblant a la classe `LlistaProductes` que hem desenvolupat durant les sessions de classe, però que ens pugui servir per gestionar una llista que guardi objectes d'una jerarquia de classes qualsevol i no només de la jerarquia de productes. Aquesta classe s'anomenarà `LlistaPolimorfisme` i s'haurà de declarar com un *template* que rebi com a tipus, el tipus de la classe base de la jerarquia amb la que la volem utilitzar (la classe `Producte` en el cas de l'exemple de les sessions de classe). Aquesta classe haurà de tenir els mètodes següents:

- Un constructor de còpia
- Un operador d'assignació
- Un mètode `afegeixElement` que rebi com a paràmetre per referència un objecte del tipus T del *template* i afegeixi aquest nou element al final de la llista. Abans d'afegir-lo haurem de crear un nou objecte dinàmic del tipus de la classe derivada que sigui una còpia de l'objecte que es passa com a paràmetre per referència.

  ```
  void afegeixElement(T& element);
  ```

- Un mètode `eliminaElement` que rebi com a paràmetre un `string` i elimini l'element de la llista que es correspongui amb aquest `string`. Aquest mètode ha de suposar que el tipus T del *template* té sobrecarregat l'operador == de forma que es pot comparar un objecte de tipus T amb un `string`. Per exemple, en el cas de la jerarquia dels productes aquest operador == compararia si el codi del producte és igual a l'string que es passa com a paràmetre (us donarem una versió de la jerarquia de productes modificada perquè pugueu veure com seria aquest operador). A l'hora d'eliminar ens hem d'assegurar d'alliberar la memòria dinàmica de l'objecte.

  ```
  bool eliminaElement(const string& clau);
  ```

- Un mètode `getElement` que rebi com a paràmetre un `string` i retorni per referència al paràmetre `element` l'element de la llista que es correspongui amb aquest `string`. Igual que el mètode anterior, aquest mètode ha de suposar que el tipus T del *template* té sobrecarregat l'operador == de forma que es pot comparar un objecte de tipus T amb un `string`. Si l'element no està dins de la llista el valor de retorn del mètode serà `false`, i `true` en cas contrari.

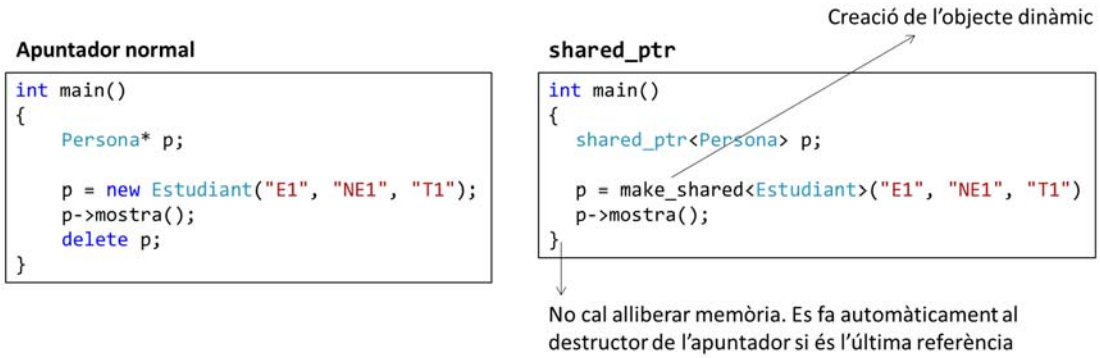
  ```
  bool getElement(const string& clau, T*& element) const;
  ```

Per provar el codi d'aquesta classe us donarem la classes de la jerarquia de productes modificades per incloure l'operador == per comparar un producte amb un string.

Un cop implementada i validada aquesta classe, podeu provar d'utilitzar-la per guardar les llistes dels exercicis 4 i 5. Tot us hauria de seguir funcionant correctament.

Exercici 6 (nivell avançat)

Un dels problemes que tenim quan treballem amb llistes polimòrfiques és la creació i destrucció dels objectes dinàmics associats a la llista, que s'ha de fer explícitament quan eliminem els elements de la llista (com per exemple al destructor i al constructor de còpia de la classe `LlistaProductes` que hem vist a classe).
A la llibreria estàndard existeix un tipus d'apuntador intel·ligent (similar al que hem plantejat a l'exercici 6 del tema de templates) que ens pot evitar haver de fer aquesta destrucció explícita dels objectes dinàmica. Aquest apuntador intel·ligent està implementat a la classe `shared_ptr` - https://en.cppreference.com/w/cpp/memory/shared_ptr -. Bàsicament el que ens assegura aquesta classe és que quan destruïm un apuntador intel·ligent de tipus `shared_ptr` l'objecte dinàmic associat a aquest apuntador també s'allibera, si ja no existeix cap altre apuntador intel·ligent que estigui apuntant a la mateixa adreça.
D'aquesta forma, quan eliminem els apuntadors de la llista ja no caldrà alliberar l'objecte dinàmic explícitament amb el `delete`, perquè s'alliberarà automàticament si no hi ha més apuntadors apuntant a l'objecte.
De la mateixa forma, quan fem una còpia de la llista ja no cal duplicar els objectes dinàmics perquè els apuntadors intel·ligents gestionen les múltiples referències a un mateix objecte i no l'alliberaran fins que no quedi cap referència a aquell objecte.
A l'exemple següent podeu veure la diferència entre la gestió dels apuntadors i la memòria amb apuntadors normals o amb apuntadors intel·ligents de la classe `shared_ptr`.



- Modifiqueu algunes de les llistes polimòrfiques que hem utilitzat als exercicis de classe o les que heu implementat als exercicis 4, 5, i 6 per gestionar els apuntadors amb la classe `shared_ptr` enlloc d'utilitzar apuntadors normals. Comproveu que si no feu la destrucció o duplicació explícita dels objectes dinàmics tot segueix funcionant correctament.