Analyse et Programmation Orientées Objets / C++

Persistance des objets

Fichiers et langages objets

➡ Structure d'un fichier de persistance

Un fichier de persistance est organisé sous forme d'une séquence ordonnée d'enregistrements logiques

- Présence d'un marqueur de la classe d'origine
- Opération de **sérialisation**
- Les enregistrements sont de longueur variable
- Présence d'une marque de fin d'enregistrement (*eol*)

Langage C++ et persistance (1)

➡ Description statique (1)

Usage obligatoire des flux et de classes prédéfinies dédiées à la sérialisation

- Mise en place d'un flux attaché au fichier physique
- Surcharge des opérateurs << et >>
- Choix des Microsoft Foundation Class (MFC)
- Usage de la classe **Object**
- Fonction virtuelle **serialize**

Langage C++ et persistance (2)

➡ Description statique (2)

Mise en place de la sérialisation

- Dériver la classe **Object**
- Définir un constructeur par défaut
- Définir un constructeur dédié
- Surcharger la méthode **serialize**

Langage C++ et persistance (3)

➡ Description dynamique (1)

Description analogue aux M.F.C.

- Création d'une instance de **File**
- Assure l'interface avec le système d'exploitation
- Equivalence avec le pointeur de type FILE*
- Création d'une instance de **Archive** (flux) basée sur l'instance de **File**
- Archive dédiée à un type d'accès (load ou store)
- **Ouverture implicite**

Langage C++ et persistance (4)

- **Description dynamique (2)**

  Description analogue aux M.F.C.

  - Lecture des objets par opérateur >>
  - Ecriture des objets par opérateur <<
  - **Fermeture explicite** de l'archive (fin d'usage)
  - **Fermeture explicite** du fichier de données

La classe dérivée (1)

➡ **Attributs de la classe *Livre***

- Titre du livre (String)
- Identité de l'auteur (String)
- Nom de l'éditeur (String)

La classe dérivée (2)

➡ **Méthodes de la classe Livre**

- Constructeur par défaut
- Constructeur normal
- Surcharge de la méthode serialize
- Surcharge de l'opérateur <<

La classe dérivée (3)

```
# include  <Object.h>

class Livre : public Object  {
private :
String m_titre;
String m_auteur;
String m_editeur;
public :
--- Cf description transparent suivant

};
```

La classe dérivée (4)

--- Cf description transparent précédent

```
public :
    Livre();
    Livre(String, String, String);
    Livre(char*);
    ---
    void serialize (Archive&);
};
```

La classe dérivée (5)

```
# include "Livre.h"

void Livre::serialize (Archive& ar) {
    if (ar.isStoring()) {                          // Store

        ar << m_titre << m_auteur<<m_editeur ;
    }
    else {                                         // Load

        ar >> m_titre >> m_auteur>>m_editeur ;
    }
}
```

Enregistrement d'un objet

```
# include "Livre.h"
void main () {
File cF1 ("Livre.dat", File::modeCreate);
Archive arS (&cF1, Archive::store);
Livre* pL1= new Livre ("Ttt", "Aaa", "Eee" );

    pL1->serialize(arS);
    arS.Close();
    cF1.Close();
}
```

Chargement d'un objet

```
# include "livre.h"
void main () {
File cF1 ("Livre.dat", File::modeRead);
Archive arL (&cF1, Archive::load);
Livre* pL= new Livre();


    pL->serialize(arL);
    arL.close();
    cF1.close();
    cout << *pL << endl;
}
```