Analyse et Programmation Orientées Objets / C++

Le paradigme objet

Origine et justification

➡ **Position dans le cycle de vie**

- Assurer la continuité depuis la phase d'analyse (ACSI)

- Prolongement des concepts des S.G.B.D.
- Simplifier la phase de conception
- Encadrer la phase de programmation (Encapsulation)
- Faciliter l'intégration et la mise au point
- Améliorer la productivité (Réutilisabilité, extensibilité)
- Faciliter la maintenance (Lisibilité, modularité)

Concepts sous-jacents

➡ **Niveau conception/programmation**

- instanciation d'objets à partir de classes
- l'encapsulation
- l'héritage
- la composition
- le polymorphisme
- la surcharge
- la généricité

Conception orientée objets

➡ **Structuration logique**

- Tout application est une partition de classes
- Chaque classe est la description complète d'une entité
- Les entités sont identifiées en phase d'analyse (ACSI)
- Les attributs sont les propriétés de l'entité décrite
- Les méthodes sont les comportements de l'entité décrite
- Les fonctions résiduelles sont limitées à des traitements de servitude

Le langage C++ (1)

➡ **Le langage C est le langage support**

Tous les concepts de base du langage C sont conservés

- langage à structure de blocs et portée des variables
- structuration imbriquée des blocs
- prototype de fonctions et compilation séparée
- accès aux adresses et mécanismes d'allocation
- usage de bibliothèques externes
- édition de liens pour un fichier binaire exécutable
- point d'entrée (**main**) unique

Le langage C++ (2)

➡ Extension du langage C (1)

- Nouveau type de bloc (classes)
- Distinction entre fonctions et méthodes
- Restriction sur la portée des fonctions (sections)
- Introduction du polymorphisme des fonctions
- Introduction de la surcharge des opérateurs
- Introduction de la généricité (templates)
- Gestion nouvelle de toutes les E/S (flux)

Le langage C++ (3)

➡ **Extension du langage C (2)**

- Relation de composition
- Relation d'héritage
- Edition de liens dynamique (virtual)
- Persistance des objets
- Nouvelles structures de données (map notamment)
- Classes abstraites et méthodes virtuelles pures
- Classes internes

Programmation orientée objets (1)

➡ **Structuration physique**

- Chaque classe Xxxxx est décrite par deux fichiers :
  - *Xxxxx.h*
  - *Xxxxx.cpp*
- Chaque classe contient la déclaration de tous les **attributs** de l'entité décrite (*Xxxxx.h)*
- Chaque classe contient la déclaration des **méthodes** de l'entité décrite (*Xxxxx.cpp)*
- Le point de lancement de l'application est dans fichier indépendant

Programmation orientée objets (2)

➡ **Fichier *Xxxxx.cpp***

- Définition des **méthodes** de l'entité décrite
- Définition des fonctions privées nécessaires aux méthodes
- Chaque méthode contient la déclaration de toutes les variables qui lui sont nécessaires
- Chaque fonction privée contient la déclaration de toutes les variables qui lui sont nécessaires

Programmation orientée objets (3)

➡ **Fichier *Aaaaa.cpp***

- Définition du point de lancement de l'application (main)
- Même signature formelle implicite qu'en langage C

  int main (int argc, char*[] argv)

- La signature polymorphe standard reste valide

  void main ()