Course: C++ language

Ali ZAINOUL

for AELION
December 8, 2023

Table of contents

/* Principes fondamentaux de la Programmation Orientée Objet */ (POO)

Principes fondamentaux de la POO

Définition: La Programmation Orientée Objet (POO) est un des principes fondamentaux en programmation informatique, ses origines remontent aux années 1970 avec les langages Simula et Smalltalk, mais le principe a rapidement pris son envol grâce à la création du langage C++ qui est l'extension du langage C, avec en effet, cette quête de rendre les logiciels plus robustes.

Principes fondamentaux de la POO (suite)

Une mnémotechnique utile regroupant les **six principes** fondamentaux de la Programmation Orientée Objet est **"ACOPIE"**.

- **A***bstraction* (Abstraction)
- **C***lass* (Classe)
- **O***bject* (Objet)
- **P***olymorphism* (Polymorphisme)
- **I***nheritance* (Héritage)
- **E***ncapsulation* (Encapsulation)

Object

- En informatique, un objet est un conteneur (container) symbolique et autonome contenant des informations et des fonctions/méthodes concernant un sujet, manipulés dans un programme. Le sujet est souvent quelque chose de tangible appartenant au monde réel.

Class

- La classe est une structure informatique particulière dans le langage objet.
- Elle décrit la structure interne des données et elle définit les méthodes qui s'appliqueront aux objets de même famille (même classe) ou type.
- Elle propose des méthodes de création des objets dont la représentation sera donc celle donnée par la classe génératrice. Les objets sont dits alors instances de la classe. C'est pourquoi les attributs d'un objet sont aussi appelés variables d'instance et les messages opérations d'instance ou encore méthodes d'instance.

Encapsulation

- L'encapsulation est le fait de regrouper les données et les méthodes qui les manipulent en une seule entité appelée *capsule*.
- Elle permet d'avoir un code organisé, restreindre l'accès à certaines portions depuis l'extérieur de la capsule et avoir un code robuste.

Abstraction

- L'abstraction est l'un des concepts clés dans les langages de programmation orientée objet. Son objectif principal est de gérer la complexité (comprendre difficulté) en masquant les détails inutiles à l'utilisateur.
- L'abstraction est le processus consistant à représenter un objet dans la vie réelle en tant que modèle informatique.
- Cela consiste essentiellement à extraire des variables pertinentes, attachées aux objets que l'on souhaite manipuler, et à les placer dans un modèle informatique convenable.

Inheritance

- C’est un mécanisme pour transmettre toutes les méthodes d’une classe dite ”mère” vers une autre dite ”fille” et ainsi de suite.

Polymorphism

- Le nom de polymorphisme vient du *grec* et signifie qui peut prendre plusieurs formes. Cette caractéristique est un des concepts essentiels de la programmation orientée objet. Alors que l'héritage concerne les classes (et leur hiérarchie), le polymorphisme est relatif aux méthodes des objets.

Définition et utilisation des classes en C++

Définition : Une **classe** est une structure de données qui définit les attributs et les méthodes qui définissent un objet.

- Exemple 1:

```
// Class MyClass
class MyClass {
        private:
                std::string name; // Member
        public:
                MyClass(std::string _name) : name(_name) {} // Constructor
                std::string getName() const { return name; }
};
```

Définition et utilisation des classes en C++ (suite)

- Exemple 2:

```
class Person {
        private:
                std::string name;
                int age;
        public:
                Person(std::string _name, int _age) : name(_name), age(_age) {}
                std::string getName() const { return name; }
                int getAge() const { return age; }
};
```

Définition et utilisation des classes en C++ (suite)

- Exemple 3:

```
class Car {
        private:
                std::string brand, model;
                int year;
        public:
                Car(std::string _brand, std::string _model, int _year) : brand(_brand), model(_model), year(_year) {}
                std::string getBrand() const { return brand; }
                std::string getModel() const { return model; }
                int getYear() const { return year; }
};
```

Définition et instanciation des objets en C++

Définition : Un **objet** est une **instance** d'une classe.

- Exemples:

```
MyClass myClass("myOwnClass");
```

```
Person person("Ali", 30);
```

```
Car car("Alfa Romeo", "Giulietta", 2020);
```

Les attributs

Définition : un **attribut**, également appelé **membre** de classe, est une variable déclarée à l'intérieur d'une classe qui a pour but de stocker des données spécifiques à cette classe. Les attributs sont ainsi des caractéristiques ou des propriétés d'un objet qui sont définies dans la classe et qui peuvent être utilisées pour décrire l'état ou les caractéristiques de l'objet.

Les méthodes en C++

Définition : une **méthode** est une fonction ou un bloc de code associé à une classe ou à un objet. Une méthode permet ainsi de définir le comportement de l'objet ou de la classe.

- Si elle est appelée par une instance de la classe (objet), c'est donc une **méthode régulière** ou simplement **méthode**.
- Si elle est appelée par la classe elle-même (classe), c'est donc une **méthode statique** ou **méthode de classe**.

Les getters et les setters

Définition : Les **getters** et les **setters** sont des méthodes publiques utilisées respectivement pour accéder et modifier les valeurs des variables d'une classe.

- Les getters retournent la valeur de la variable, et ont la signature:
  returnType getVar() const { return var; }
- Les setters modifient la valeur de la variable, et ont la signature:
  void setVar(typeName _var) { var = _var;}

Utilisation des méthodes sur les objets en C++

- Les méthodes permettent d'interagir avec les membres (attributs) d'un objet (instance d'une classe).
- Exemples:

```
std::string name = myClass.getName();
```

```
std::string name = person.getName();
int age = person.getAge();
```

```
std::string brand = car.getBrand();
std::string model = car.getModel();
int year = car.getYear();
```

L’encapsulation en C++

Définition : L’encapsulation en C++ est un mécanisme qui permet de cacher les détails d’implémentation d’une classe aux autres classes. Il s’agit d’un concept clé de la programmation orientée objet qui permet de protéger les données d’une classe et de garantir leur intégrité.

L’encapsulation en C++

Objectifs de l’encapsulation

- Réduction des erreurs et des conflits de noms.
- Amélioration de la sécurité.
- Facilitation de la maintenance du code.
- Modification facile du code sans affecter les autres parties du programme.

Exemple d'encapsulation en C++

Supposons que nous avons une classe **BankAccount** qui contient des données sensibles telles que le solde du compte et le numéro de compte bancaire. Pour protéger ces données, nous pouvons les déclarer comme privées et utiliser des méthodes publiques pour y accéder et les modifier, par exemple **getBalance()** et **deposit()**. De cette manière, les autres classes ne peuvent pas accéder directement aux données sensibles de la classe **BankAccount**, mais doivent passer par les méthodes publiques qui garantissent que les données sont manipulées de manière sûre et sécurisée.

Exemple d'encapsulation en C++

- La classe **BankAccount** peut avoir des membres privés tels que le solde et le numéro de compte.
- Les méthodes publiques **getBalance()** et **deposit()** sont utilisées pour accéder et modifier le solde du compte.
- Les autres classes ne peuvent pas accéder directement au solde ou au numéro de compte, mais doivent passer par les méthodes publiques.

Exemple d'encapsulation en C++

- En encapsulant les données de cette manière, nous assurons que les informations sensibles ne sont pas accessibles depuis l'extérieur de la classe et que leur manipulation est effectuée de manière contrôlée. Cela permet de garantir l'intégrité des données et d'améliorer la sécurité de notre programme.

Access specifiers des attributs et des méthodes

Définition : Les **spécificateurs d'accès** sont des mots-clés qui définissent la portée et l'accessibilité des variables, des méthodes et des classes en C++. Il existe quatre spécificateurs d'accès en C++ : `public`, `private`, `protected` et `default`.

Les spécificateurs d'accès en C++

- Le spécificateur `public` permet un accès illimité, c'est-à-dire que la variable, la méthode ou la classe peut être accédée de n'importe où dans le programme.
- Le spécificateur `protected` permet un accès aux classes qui sont dans le même package et aux classes filles.
- Le spécificateur `private` permet uniquement l'accès à la classe elle-même.

Tableau des spécificateurs d'accès en C++ - Partie 1

Spécificateur	Même classe	Classe enfant	Tout le monde
`public`	Oui `class A { public: int x; };`	Oui `class B : public A {};`	Oui `A obj; obj.x = 42;`
`protected`	Oui `class A { protected: int x; };`	Oui `class B : protected A {};`	Non `A obj; obj.x = 42; // Erreur`

Tableau des spécificateurs d'accès en C++ - Partie 2

Spécificateur	Même classe	Classe enfant	Tout le monde
`private`	Oui `class A { private: int x; };`	Non `class B : private A {};`	Non `A obj; obj.x = 42; // Erreur`

L’héritage en C++

Définition: L’héritage permet à une classe d’hériter les attributs et les méthodes d’une autre classe.

- La classe qui hérite est appelée **sous-classe** ou **classe dérivée**.
- La classe dont on hérite est appelée **classe de base** ou **classe parente**.

Exemple d'héritage en C++

Supposons que nous avons une classe `Person` avec les attributs `name` et `age`.

- Nous avons une classe `Employee` qui hérite de `Person` et ajoute l'attribut `salary`.
- La classe `Employee` a accès aux attributs et méthodes de `Person`.

Princice du polymorphisme

Définition: Le polymorphisme est une caractéristique clé de la programmation orientée objet (POO) qui permet à un objet de se comporter de différentes manières en fonction du contexte dans lequel il est utilisé. En C++, il existe quatre types de polymorphisme : le polymorphisme de sous-typage, le polymorphisme paramétrique, le polymorphisme ad-hoc et le polymorphisme de liaison tardive.

Polymorphisme de sous-typage (ou polymorphisme d'héritage)
Le polymorphisme de sous-typage est basé sur l'héritage de classes en C++. Il permet à une classe de se comporter comme l'une de ses sous-classes. Lorsque des objets sont déclarés comme une superclasse, ils peuvent être assignés à des objets de n'importe quelle sous-classe de cette superclasse.
Par exemple, une classe "Chat" héritant de la classe "Animal" peut être traitée comme un "Animal". Cela permet aux méthodes de l'objet "Animal" d'être appelées sur un objet "Chat".

Polymorphisme paramétrique (ou générique) Le polymorphisme paramétrique, également connu sous le nom de polymorphisme générique, permet de définir des classes ou des méthodes génériques pouvant prendre en paramètre différents types de données. En utilisant les types génériques, une seule classe ou méthode peut être utilisée pour manipuler des objets de différents types.
Par exemple, la classe "Liste" peut être utilisée pour stocker des objets de différents types, tels que des "String", des "Integer" ou des "Double".

Polymorphisme ad-hoc (ou surcharge de méthode)
Le polymorphisme ad-hoc, également appelé surcharge de méthode, permet à une méthode d'avoir des comportements différents en fonction du type ou du nombre de ses paramètres. En C++, il est possible de définir plusieurs méthodes portant le même nom dans une même classe, à condition que les paramètres soient différents. Par exemple, une méthode "afficher" peut accepter soit un "String" soit un "Integer" en tant que paramètre. Le compilateur C++ choisira la méthode appropriée en fonction du type du paramètre.

Polymorphisme de liaison tardive (ou d'exécution)

Le polymorphisme de liaison tardive (ou polymorphisme d'exécution) permet à une méthode héritée d'être redéfinie dans une sous-classe pour fournir une implémentation spécifique à la sous-classe. La résolution de la méthode à appeler se fait à l'exécution en fonction du type de l'objet sur lequel la méthode est appelée.

Par exemple, une classe "Animal" peut avoir une méthode "parler" qui affiche le bruit générique d'un animal, mais cette méthode peut être redéfinie dans une sous-classe "Chat" pour produire un miaulement spécifique. Lorsque la méthode "parler" est appelée sur un objet "Chat", la méthode redéfinie dans la sous-classe sera appelée à la place de la méthode de la classe "Animal".

Types de polymorphisme en C++

- Le polymorphisme fait référence à la capacité d'objets de différentes classes de répondre au même appel de fonction.
- En C++, il existe deux principaux types de polymorphisme : le polymorphisme à la compilation et le polymorphisme à l'exécution.
- Le polymorphisme à la compilation est réalisé grâce à la surcharge de fonctions et à la surcharge d'opérateurs.
- Le polymorphisme à l'exécution est réalisé grâce aux fonctions virtuelles et s'applique lorsque le type réel de l'objet est déterminé pendant l'exécution.

Exemple complet

Un exemple complet se trouve dans le projet _Shape.

L’abstraction en C++

Définition: L’abstraction est un concept clé qui permet de créer des modèles généraux à partir de classes spécifiques. Elle permet de simplifier la complexité du code en cachant les détails d’implémentation et en se concentrant sur l’essentiel.

- Les classes abstraites sont des classes qui ne peuvent pas être instanciées, mais qui peuvent être héritées.
- Les méthodes abstraites sont des méthodes qui ne sont pas implémentées dans la classe abstraite elle-même, mais qui doivent être implémentées dans les classes qui en héritent.

L’abstraction en C++

- En utilisant l’abstraction en C++, vous pouvez créer des modèles génériques pour résoudre des problèmes spécifiques, sans vous soucier des détails d’implémentation.
- Cela rend votre code:
  - Plus modulaire;
  - Plus facile à comprendre;
  - Plus facile à maintenir;
  - Plus facile à réutiliser.

Exemple d'abstraction en C++

Supposons que nous avons une application de formes géométriques qui nécessite un calcul de l'aire et du périmètre de chaque forme. Nous pouvons créer une classe abstraite **Shape** qui contient des méthodes abstraites pour le calcul de l'aire et du périmètre. Ensuite, nous pouvons créer des classes pour chaque forme géométrique spécifique (Circle, Square, Triangle, etc.) qui héritent de la classe "Shape" et fournissent une implémentation des méthodes abstraites pour leur forme spécifique. Un exemple complet est donné ici: **ici**.

Les classes abstraites en C++

Les classes abstraites

- Une classe abstraite est une classe qui ne peut pas être instanciée.
- Les classes abstraites sont utilisées pour définir une classe de base qui peut être étendue par des sous-classes.

Les classes abstraites en C++ (suite)

Les classes abstraites

- Les classes abstraites peuvent contenir des méthodes abstraites, qui sont déclarées mais pas implémentées dans la classe abstraite.
- Les sous-classes d'une classe abstraite doivent implémenter toutes les méthodes abstraites de la superclasse ou être elles-mêmes déclarées abstraites.

Fonctions virtuelles en C++

- Les fonctions virtuelles sont utilisées en C++ pour atteindre le polymorphisme à l'exécution, également appelé polymorphisme dynamique.
- Lorsqu'une classe dérivée substitue une fonction virtuelle de sa classe de base, l'implémentation de la classe dérivée est appelée à la place de celle de la classe de base.
- Cela permet à différents objets de la même classe de base d'exhiber différents comportements en fonction de leurs types dérivés réels.

Exemple de fonctions virtuelles

- Considérons une classe de base appelée ‘Forme‘ avec une fonction virtuelle appelée ‘aire()‘.
- Nous créons deux classes dérivées, ‘Rectangle‘ et ‘Cercle‘, qui substituent la fonction ‘aire()‘ avec leurs propres implémentations.
- Nous pouvons ensuite créer des objets des deux classes et appeler la fonction ‘aire()‘, qui invoquera l’implémentation appropriée en fonction du type réel de l’objet.

Fonctions amies en C++

- Les fonctions amies en C++ sont des fonctions non membres qui ont accès aux membres privés et protégés d'une classe.
- Elles peuvent être utiles pour fournir à des fonctions externes un accès privilégié aux données privées d'une classe.
- Les fonctions amies ne font pas partie de la classe et n'ont pas de pointeur 'this', car elles ne sont associées à aucun objet particulier.

Exemple de fonctions amies

- Supposons que nous ayons une classe appelée ‘Voiture‘ avec des variables membres privées telles que ‘marque‘, ‘modèle‘ et ‘année‘.
- Nous pouvons déclarer une fonction amie appelée ‘afficherInfosVoiture()‘ qui peut accéder à ces membres privés et afficher les informations de la voiture.
- Cela nous permet de garder les variables membres privées tout en fournissant un moyen d’accéder et d’afficher les informations de manière externe.