

Programmation orientée objet

Concepts de base

Objet

- Un **objet** est une entité pouvant être créée, stockée, manipulée et détruite
- Un objet possède des attributs (propriétés)
- Un objet offre un certain nombre de méthodes (fonctions membres) pour le manipuler
- Tout objet appartient à une classe

Classe

- Une **classe** représente un type d'objet
- Dans une définition de classe on spécifie:
 - les attributs de tout objet appartenant à cette classe
 - les méthodes permettant de manipuler un objet de cette classe
 - la visibilité de chaque attribut et méthode

Classe (suite)

- En général (on pourrait pratiquement dire toujours!), les attributs sont **privés**, c'est-à-dire qu'aucune fonction externe à l'objet n'y a accès
- Certaines méthodes sont **publiques** et peuvent donc être utilisées par des fonctions externes pour manipuler un objet; on dit que ces méthodes constituent **l'interface** de la classe

Classe (suite)

- Les méthodes publiques peuvent parfois utiliser des fonctions internes **privées** pour accomplir leur tâche

Relation Objet - Classe

- La relation qui lie objet et sa classe est la même qu'entre une variable et son type :

```
int nbInvités;
```

```
string nom;
```

```
Point centre;
```

Relation Objet - Classe

- La relation qui lie objet et sa classe est la même qu'entre une variable et son type :

`int` nbInvités;

variable

type

`string` nom;

`Point` centre;

Relation Objet - Classe

- La relation qui lie objet et sa classe est la même qu'entre une variable et son type :

`int nbInvités;`

`string nom;`

objet

classe standard C++

`Point centre;`

Relation Objet - Classe

- La relation qui lie objet et sa classe est la même qu'entre une variable et son type :

`int nbInvités;`

`string nom;`

`Point centre;`

objet

classe

Création d'un objet

Variable
identifiant l'objet

```
Time debutCours = Time (13,45,0) ;
```

Classe de
l'objet

Initialisation
de l'objet

Création d'un objet (forme abrégée)

```
Time debutCours(13,45,0);
```

Création d'un objet (valeurs par défaut)

```
Time debutCours = Time();
```

```
Time debutCours;
```

Forme abrégée
de l'initialisation
par défaut

Construit un objet de
type Time en fournissant
des valeurs par défaut à
ses attributs

Création d'un objet (valeurs par défaut)

```
Time debutCours = Time();
```

```
Time debutCours;
```

Il ne faut **pas** mettre de
parenthèses pour
l'initialisation par défaut

Définition d'une classe en C++

```
class NomDeLaClasse
```

```
{
```

```
public:
```

déclarations de constructeurs

déclarations de méthodes publiques

```
private:
```

déclarations de méthodes privées

attributs

```
};
```

Exemple de définition de classe

```
class Point
{
public:
    Point();
    Point(double coordonneeX, double coordonneeY);

    double getX();
    double getY();
    void move(double dx, double dy);

private:
    double x_;
    double y_;
};
```

Remarque: par convention, nous utiliserons toujours un _ pour distinguer les variables qui correspondent aux attributs d'une classe.

Implémentation des méthodes

- En général, en C++, les méthodes sont implémentées séparément de la définition de classe
- Syntaxe:

```
typeRetour NomClasse::nomFonction(param1, ...)  
{  
    instructions  
}
```


Exemple d'implémentation d'une méthode

```
void Point::move(double dx, double dy)
{
    x_ += dx;
    y_ += dy;
}
```

Manipulation d'un objet

- On peut obtenir la valeur d'un attribut d'un objet
- On peut changer la valeur d'un attribut d'un objet
- On peut demander à un objet de nous créer un nouvel objet
- On peut demander à un objet de manipuler un autre objet

Manipulation d'un objet (suite)

- Pour manipuler un objet, on utilise les méthodes qui sont définies par la classe à laquelle il appartient
- Par exemple, la classe `Time` définit une méthode `getHours()` qui retourne le champ *heure* de l'objet:

```
short h = debutCours.getHours();
```

Paramètres d'une méthode

- Toute méthode a un **paramètre implicite**, qui correspond à l'objet sur lequel elle est appliquée
- Les autres paramètres qui apparaissent dans l'en-tête de la méthode sont les **paramètres explicites**.

Paramètres d'une méthode

- Quand on écrit:
 - `point1.move(10, 3);`
- le compilateur crée en fait une fonction à trois paramètres:
 - `move(point1, 10, 3);`
- Mais tout cela est transparent pour nous

Interface d'une classe

- L'interface d'une classe comprend toutes les fonctions publiques dont les items suivants:
 - Les constructeurs (toute classe devrait posséder au moins un constructeur par défaut, qui ne prend aucun paramètre)
 - Les fonctions de modification (mutators)
 - Les fonctions d'accès (accessors)
 - Les fonctions utiles à l'objet

Principe d'encapsulation

- On n'a pas accès directement aux attributs d'un objet
- On modifie ou on obtient la valeur d'un attribut toujours par l'intermédiaire d'une méthode
- En résumé, on ne peut manipuler l'état d'un objet que par le biais des méthodes qui sont définies par sa classe (interface)

Constructeur

- Le rôle principal d'un constructeur est d'initialiser les attributs lors de la création d'un objet
- En général, on a un constructeur par défaut, qui ne reçoit aucun paramètre, et qui donne aux attributs des valeurs par défaut
- On peut aussi avoir des constructeurs qui acceptent comme paramètres les valeurs initiales que l'on veut donner aux attributs

Exemple de constructeur par défaut

```
Point::Point()  
{  
    x_ = 0.0;  
    y_ = 0.0;  
}
```

Exemple de constructeur avec paramètres

```
Point::Point(double coordonneeX,  
             double coordonneeY)  
{  
    x_ = coordonneeX;  
    y_ = coordonneeY;  
}
```

Quand un constructeur par défaut est-il appelé?

- Lors de la simple déclaration d'une variable d'objet:

```
Point p;
```

- Lorsqu'on utilise un tableau d'objets:

```
Point tabDePoints[10];
```

Quand un constructeur par défaut est-il appelé? (suite)

- Lorsque l'objet est lui-même un attribut d'un autre objet:

```
class Cercle
{
public:
    Cercle();
    ...
private:
    Point centre_;
    double rayon_;
};
```

```
Cercle::Cercle()
{
    rayon_ = 0.0;
}
```

Pour initialiser cet attribut, le constructeur par défaut de Point sera appelé, avant même d'appeler le constructeur de Cercle

Destructeur

- Un destructeur est une méthode qui est appelée lorsqu'un objet est détruit
- On s'en sert pour faire du ménage
- Par exemple, comme on le verra plus tard, c'est dans le destructeur qu'on désallouera des pointeurs appartenant à l'objet en question

Destructeur (suite)

- Le destructeur est défini en lui donnant le même nom que la classe précédé du tilde ~ :

```
Point::~~Point()
```

```
{
```

```
    // Rien à faire dans ce cas-ci
```

```
}
```

Quand un destructeur est-il appelé?

- Si l'objet est une variable locale à une fonction, il sera détruit à la sortie de cette fonction
- Si l'objet est déclaré dans le `main()`, il sera détruit à la sortie du programme
- Si l'objet est dynamique, le destructeur est appelé lorsqu'on exécute `delete` sur cet objet (on reviendra sur ce sujet lorsque l'allocation dynamique sera discutée)