

Module03 – POO en C++



Objectifs

- Classe
- Constructeurs
- Héritage
- Méthode virtuelle / virtuelle pure

Classe – Déclaration

- La déclaration doit être faite dans un fichier d'entête
- Dans le cours, vous ferez un fichier d'entête par classe qui aura le même nom que la classe

```
class MaClasse {  
public:  
    // ...  
  
protected:  
    // ...  
  
private:  
    // ...  
};
```

Classe – Méthodes spéciales

- Constructeurs : méthodes exécutées après l'allocation de la mémoire nécessaire à la création d'un objet. Il y en a 4 types normalisés
- Destructeur : méthode exécutée avant la désallocation de la mémoire allouée à un objet
- Opérateurs d'affectation : méthodes exécutées lors de l'affectation d'un objet dans un autre

Classe – Constructeurs 1 / 4

- Constructeur par défaut
 - Constructeur sans paramètre
 - Est créé automatiquement si vous ne déclarez pas d'autre constructeur

Televiseur.h

```
class Televiseur {  
public:  
    Televiseur();  
    // ...  
private:  
    int m_canalActuel;  
    int m_volume;  
    bool m_estAllume;  
};
```

Televiseur.cpp

```
Televiseur::Televiseur() :  
    m_canalActuel(1),  
    m_volume(20),  
    m_estAllume(false)  
{  
    // ...  
}
```

Classe – Constructeurs 2 / 4

- Constructeur d'initialisation
 - Constructeur avec paramètres

Televiseur.h

```
class Televiseur {  
public:  
    Televiseur(int p_canalActuel,  
                int p_volume,  
                bool p_estAllume = false);  
    // ...  
private:  
    int m_canalActuel;  
    int m_volume;  
    bool m_estAllume;  
};
```

Televiseur.cpp

```
// Ctor d'initialisation avec un paramètre qui a une valeur par défaut  
Televiseur::Televiseur(int p_canalActuel, int p_volume, bool p_estAllume) :  
    m_canalActuel(p_canalActuel),  
    m_volume(p_volume),  
    m_estAllume(p_estAllume)  
{  
    ;  
}
```

Classe – Constructeurs 3 / 4

- Constructeur par copie
 - Constructeur qui prend un **référence constante** de l'objet à copier
 - Est créé automatiquement si vous ne déclarez pas d'autre constructeur

Televiseur.h

```
class Televiseur {  
public:  
    Televiseur(const Televiseur&  
p_objetACopier);  
    // ...  
private:  
    int m_canalActuel;  
    int m_volume;  
    bool m_estAllume;  
};
```

Televiseur.cpp

```
Televiseur::Televiseur(const Televiseur&  
p_objetACopier) :  
    m_canalActuel(p_objetACopier.m_canalActuel),  
    m_volume(p_objetACopier.m_volume),  
    m_estAllume(p_objetACopier.m_estAllume)  
{  
    // ...  
}
```


Classe – Constructeurs 4 / 4

- Constructeur par déplacement
 - Constructeur qui prend un **référence constante** de l'objet à copier
 - Est créé automatiquement si vous ne déclarez pas d'autre constructeur

Televiseur.h

```
class Televiseur {  
public:  
    Televiseur(Televiseur&&  
p_rvalue);  
    // ...  
private:  
    int m_canalActuel;  
    int m_volume;  
    bool m_estAllume;  
};
```

Televiseur.cpp

```
Televiseur::Televiseur(Televiseur&& p_rvalue) :  
    m_canalActuel(p_rvalue.m_canalActuel),  
    m_volume(p_rvalue.m_volume),  
    m_estAllume(p_rvalue.m_estAllume)  
{  
    // ...  
}
```

Sera revu avec l'allocation dynamique de ressources

Classe – Destructeur

- Un destructeur est créé automatiquement par le compilateur si vous n'en créez pas un
- Sert à libérer les ressources dont l'objet a responsabilité

Televiseur.h

```
class Televiseur {  
public:  
    ~Televiseur();  
    // ...  
};
```

Televiseur.cpp

```
Televiseur::~~Televiseur() {  
    // ...  
}
```

Sera revu avec l'allocation dynamique de ressources

Classe – Utilisation d'un constructeur délégué

- À partir d'un constructeur vous pouvez appeler un autre constructeur de la même classe

Televiseur.cpp

```
// Utilisation d'un constructeur délégué
Televiseur::Televiseur()
: Televiseur(1, 20, false) {
    // ...
}
```

Préconditions

- Deux méthodes classiques en C++

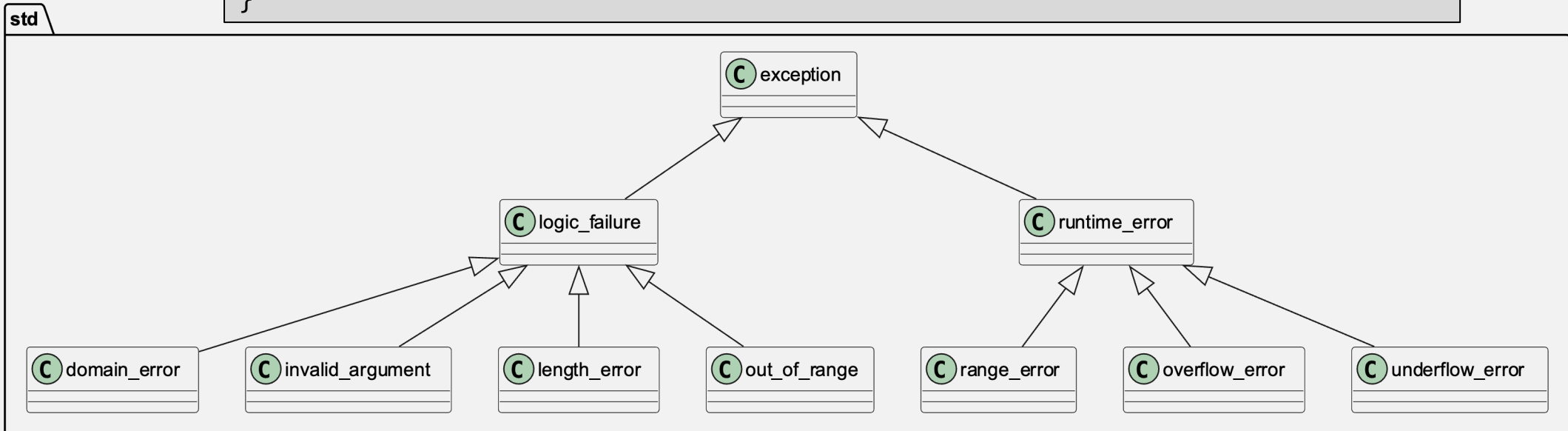
- Utiliser des assertions :

Ex. : `assert(p_volume >= 0 && p_volume <= 100);`

- Utiliser des levées d'exceptions

Ex. :

```
if (p_volume < 0 || p_volume > 100) {  
    throw std::invalid_argument("Le volume doit être compris entre 0 et 100");  
}
```



Héritage

- On peut étendre / spécialiser une classe en héritant de ses propriétés et en les complétant / modifiant
- Le C++ permet d'hériter de plus d'une classe : ce n'est pas abordé dans le présent cours. Une réponse rapide est qu'il faut ajouter le mot clef « virtual » devant chaque classe héritée. Il faut aussi appeler explicitement les constructeurs des classes parentes. Globalement, **l'héritage multiple** pose beaucoup de problème et **est à éviter** !

```
class Televiseur : public Actionnable {  
    // ...  
};
```

Méthodes

```
class Televiseur {  
public:  
    virtual bool estAllume() const;  
    virtual void allumer();  
    virtual void eteindre();  
};
```

- Les méthodes permettent d'implanter les comportements des objets
- Elles ont accès à l'objet à partir du mot clef « this » qui représente l'adresse de l'objet courant (La notion d'adresse est détaillée dans les modules suivants)
- Pour le moment, retenez la notation `this->m_ABC` pour accéder à la donnée membre `m_ABC`
- Pour appeler un membre de la classe mère, il faut utiliser le nom de la classe suivi de « `::` » et du nom du membre (Ex. `MaClasseParent::m_maDonnee`)
- Par défaut, une méthode n'est pas virtuelle (comme en C#). Si vous la redéfinissez, cette nouvelle version ne sera peut-être jamais appelée (Voir type dynamique : type réel de l'objet connu à l'exécution et le type statique : type déclaré dans le code connu à la compilation)

Méthodes spéciales – Généralités et affectation

- Comme dans beaucoup d'autre langage, C++ permet de déclarer et définir des opérateurs afin de faciliter l'écriture et la lecture du code
- Exemple d'opérateurs : +, /, -, *, new, delete, =, ==, !=, etc.
- Ici, nous parlerons seulement de l'opérateur d'affectation qui peut être redéfini avec deux variations :
 - `<classe>& operator=([const] <classe>& p_objetACopier)`
 - `<classe>& operator=(<classe>&& p_rvalue)`

Méthodes spéciales – Généralités et affectation

Televiseur.h

```
Televiseur& operator=(const Televiseur& p_objetAAffecter);  
Televiseur& operator=(Televiseur&& p_objetAAffecter);
```

Televiseur.cpp

```
Televiseur& Televiseur::operator=(const Televiseur& p_objetAAffecter) {  
    if (this != &p_objetAAffecter) {  
        this->m_canalActuel = p_objetAAffecter.m_canalActuel;  
        this->m_volume = p_objetAAffecter.m_volume;  
        this->m_estAllume = p_objetAAffecter.m_estAllume;  
    }  
  
    return *this;  
}  
  
Televiseur& Televiseur::operator=(Televiseur&& p_objetAAffecter) {  
    if (this != &p_objetAAffecter) {  
        this->m_canalActuel = p_objetAAffecter.m_canalActuel;  
        this->m_volume = p_objetAAffecter.m_volume;  
        this->m_estAllume = p_objetAAffecter.m_estAllume;  
    }  
  
    return *this;  
}
```


Abstraction

- La notion d'interface n'existe pas en C++
- Une classe peut être abstraite
- Une classe est abstraite si et seulement si la classe à au moins une méthode abstraite, appelée aussi méthode virtuelle pure
- Si vous n'avez pas de méthode à rendre abstraite, vous pouvez déclarer le destructeur comme étant un destructeur virtuel pur : il faudra quand même ici le définir

```
class Actionnable {  
public:  
    virtual bool estAllume() const = 0;  
    virtual void allumer() = 0;  
    virtual void eteindre() = 0;  
};
```

Visibilité – Membres

- **public :**
 - Tout le monde peut accéder aux membres déclarées avec cette visibilité
- **protected :**
 - Seules les classes filles peuvent accéder aux membres déclarées avec cette visibilité
- **private :**
 - Seule la classe qui les déclare peut accéder aux membre déclarées avec cette visibilité

Visibilité – Héritage

- L'utilisation des visibilités au moment de la déclaration de l'héritage permet de limiter la visibilité des membres des classes héritées
 - **public** : les visibilités des membres des classes hérités sont inchangées
 - **protected** : les visibilités des membres « public » des classes hérités deviennent « **protected** »
 - **private** : les visibilités des membres « **public** » / « **protected** » deviennent « **private** »

Références

- https://en.cppreference.com/w/cpp/language/value_category :
catégorie de variable
- https://en.cppreference.com/w/cpp/language/copy_constructor :
constructeurs par copie
- https://en.cppreference.com/w/cpp/language/copy_assignment :
opérateur d'affectation
- <https://en.cppreference.com/w/cpp/error/exception> : Exceptions