Programmation avancée en C++
GIF-1003

Révision Final

T. Eude

UNIVERSITÉ LAVAL
Département d'informatique
et de génie logiciel

Révision

- Considérations de génie logiciel
- Implantation de hiérarchies de classes
- Gestion de la mémoire
- Gestion des erreurs et des exceptions
- Frameworks
- Librairie Standard du C++ (STL)

3. Considérations de génie logiciel

3.1 Modélisation : Diagramme de classe

3.2 La théorie du contrat

Département d'informatique - Université Laval

UNIVERSITÉ LAVAL

Un contrat protège les deux parties

- Protège le client (dans ce cas ci vous-mêmes)
  - Spécifie quelles sont ses obligations.
  - En retour, est assuré d'un certain résultat.

- Protège le fournisseur (sous-contractant)
  - Spécifie le minimum requis (de la part du client):
    - le fournisseur ne peut être tenu responsable d'avoir manqué à des engagements ne figurant pas dans le contrat.

La spécification d'un contrat

Spécification d'une interface

- peut être vue comme un contrat entre un **client** (l'utilisateur du composant/classe) et un **fournisseur** (le composant/classe).

Le contrat dit ce qui doit être rencontré par le client

- pour pouvoir appeler une opération de l'interface :

  précondition.

Le contrat dit aussi ce qui doit être réalisé par le fournisseur

- ce qui est garanti au client
- pour rencontrer ce qui est prévu par l'opération

  postcondition.

Les assertions : le contrat du logiciel

- Pour produire du logiciel correct et robuste :
  - rendre les obligations et les garanties explicites pour chaque appel de méthode.
- Mécanismes pour exprimer ces conditions :

❑ **assertions**

❑ **préconditions**

❑ **postconditions :**

- assertions s'appliquant à des méthodes en particulier,

❑ **invariants** de classe :

- assertions s'appliquant à toutes les méthodes d'une classe en tout temps.

Pour une spécification complète

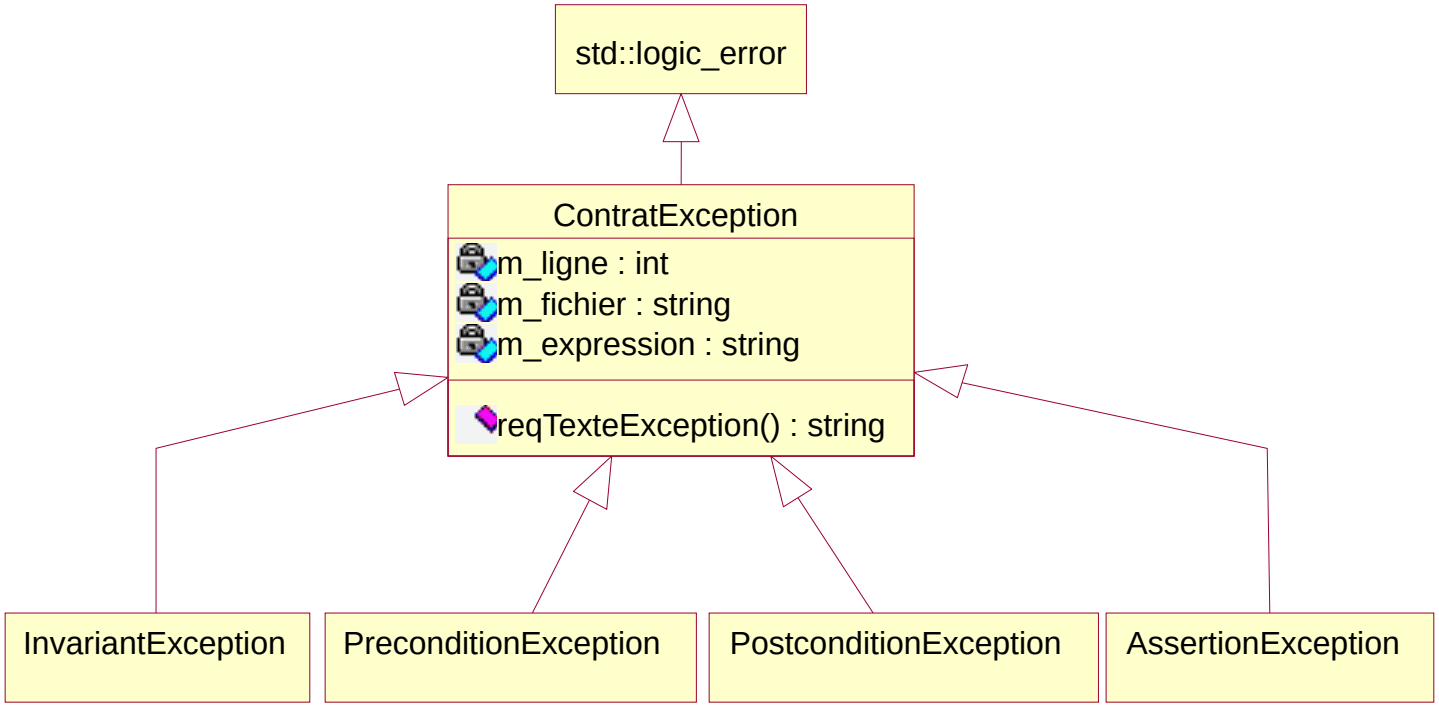
- Une spécification plus complète d'une opération offerte par un composant deviendrait:
  - le nom de l'opération
  - le nombre, le type et l'ordre des paramètres
  - les préconditions
  - les postconditions
  - le type de retour.

Invariant de classe

- Autre type d'assertion
  - représente l'ensemble des *conditions logiques* devant être respectées pour assurer un comportement correct d'une classe.

- Immédiatement après la création d'un objet de cette classe,
- l'invariant doit être *respecté* et *préservé* tout au long de la vie de l'objet.

Hiérarchie d'exceptions de contrat



Les MACROs pour le contrat

- "fonction" définie par un #define.
- Remplacement textuel, fait par le préprocesseur.
- Dans le fichier `ContratException.h`,
  1. Définition de ces classes
  2. + définition des MACROs suivantes:
     - `PRECONDITION(test);`
     - `POSTCONDITION(test);`
     - `ASSERTION(test);`
     - `INVARIANT(test);`
     - `INVARIANTS();`
- Regardons `ContratException.h`

3. Considérations de génie logiciel

3.1 Qualité d'une interface de classe

3.2 La théorie du contrat

3.3 Le test unitaire

Département d'informatique - Université Laval

UNIVERSITÉ LAVAL

Mise en œuvre

- Implantation d'un test unitaire sur une classe : on ne teste pas
  - n'importe quoi,
  - n'importe comment
  - sans structure.

- Doit être structuré.
- Doit être **répétable**
  - i.e. on doit pouvoir le faire exécuter le plus souvent possible.
- Doit fournir un rapport d'exécution extrêmement simple (ok, pas ok)

Google Test

- Test:
- Ensemble de fonctions (Macros) contenant des *assertions*,
  - déclarations vérifiant si une condition est vraie.
  - résultat d'une assertion
    - *success* (test réussi),
    - *nonfatal failure* (test échoué)
    - *fatal failure* (test échoué et arrête la fonction).
- Cas de tests :
  - un ou plusieurs tests,
  - regroupés selon la structure du code testé.

Programmation avancée en C++
GIF-1003

Thierry EUDE

Module 4 : Implantation de hiérarchies de classes

UNIVERSITÉ LAVAL
Département d'informatique
et de génie logiciel

Classe de base et classe dérivée

Dans le contexte d'héritage,

- on dit souvent qu'un objet **est une sorte** d'un autre objet.

Exemple :

un Rectangle **est une sorte** de Quadrilatère.

- Le Rectangle **hérite** d'un Quadrilatère;

✓classe Quadrilatère : **classe de base**,

✓classe Rectangle : **classe dérivée**.

Construction et destruction : Ordre d'appel

Construction des objets de la hiérarchie

- de la classe de base tout en haut jusqu'à la classe dérivée.

Destruction des objets de la hiérarchie

- de la classe dérivée jusqu'à la classe de base.

Héritage : contrôle des accès

Visibilités :

- Privée : Accès interne de la classe.
- Publique : Accès à tous les utilisateurs.
- Protégée : Accès aux héritiers.

Les méthodes d'une classe ont **toujours** accès aux attributs de la classe, peu importe la visibilité.

- les données privées ne sont manipulables que par les méthodes de la classe à laquelle les données appartiennent.
- L'héritage ne donne pas d'accès privilégié pour la visibilité privée.

Ajout de méthodes

- **Triangle** qui est une classe dérivée de **Polygone**
  - a ***immédiatement accès*** à cette nouvelle fonctionnalité,
- **Triangle2** qui utilise **Polygone**
  - doit être ajoutée manuellement dans la classe **Triangle2**.

- Résultat intéressant d'un point de vue de gestion du code.

- tout héritier de la classe Polygone a accès à ses méthodes qu'elles soient écrites ***maintenant*** ou dans le ***futur***.

Remplacer, hériter, augmenter ...

- Une classe dérivée D peut décider de prendre action sur une méthode m() de la classe de base B:
  - La classe dérivée remplace la méthode B::m() par D::m() en implantant un algorithme différent.
  - La classe dérivée hérite de B::m() sans changement.
  - La classe dérivée augmente la méthode B::m() par une méthode D::m() qui appelle d'abord B::m() avant de faire d'autre tâches.

Héritage multiple

Avantages:

- possibilités étendues de réutilisation.
- plus de flexibilité pour spécifier les classes.
- permet une modélisation plus adaptée à la façon dont nous pensons naturellement...

Désavantages:

- perte de simplicité conceptuelle ainsi que perte de clarté dans l'implantation.

Désastre potentiel:

- hériter des mêmes attributs en plusieurs copies... (aussi appelé «deadly diamond»)
- peut survenir sournoisement en cours de la maintenance et de l'évolution.

Différentes formes

- En C++, trois formes :
  - Polymorphisme ad hoc :
    - Surcharge de méthode(déjà vu)
  - Polymorphisme paramétrisable :
    - Les templates
  - Polymorphisme pur

Le polymorphisme pur

- Méthodes virtuelles, alliées à l'héritage:
  - permet de rendre les programmes extensibles par l'héritage de comportements existants
  - ✓ajout de comportements spécifiques au besoin.

- Traitement d'une famille de classes - une hiérarchie - de façon **générique**.
  - On n'a pas à identifier le type de la classe avec un **switch** comme dans les langages non OO.

- Traitement générique à partir de la classe de base : **méthodes virtuelles**.
  - à l'échelle d'une hiérarchie de classes.

Exemple des polygones …

- Tous les héritiers devront supporter le protocole imposé par la classe **Polygone**.
- L'implantation de ces méthodes pourra être un peu différente d'une classe à l'autre.
- Manipulation possible d'un ensemble d'objets de différents types:
  - à partir de pointeurs sur des objets **Polygone**
  - Le programme détermine à l'exécution le type de l'objet, et donc, la méthode à appeler.

- Le mécanisme des méthodes virtuelles permet ce comportement.

Signature des méthodes virtuelles

- Une fonction virtuelle doit avoir la même signature et même type de retour dans toutes la hiérarchie.

- Une fois qu'une fonction a été déclarée virtuelle, elle le demeure dans toute la hiérarchie à partir de ce point.

- Bonne pratique :
  - déclarer **virtual** les fonctions qui le sont et à tous les niveaux (par soucis de transparence).

Utilisation de la hiérarchie :

Création de polygones

```
// --- Création des Polygones...
vector<Polygone*> vPoly;

Triangle  t1(Point(10,10), Point(30,20), Point(10, 30));
Triangle  t2(Point(20,20), Point(40,20), Point(20,40));
Carre     c1(Point(50,50), 10);
Carre     c2(Point(0,0), 20);
Rectangle r1(Point(30,30), Point(60,60));
Rectangle r2(Point(5,60), Point(100,90));

vPoly.push_back(&t1);
vPoly.push_back(&r2);
vPoly.push_back(&r1);
vPoly.push_back(&c2);
vPoly.push_back(&t2);
vPoly.push_back(&c1);
```

- Création de polygones de tous types : Triangle, Carre et Rectangle.
- Vecteur servant à faire du traitement générique...
- Insérés dans un vecteur de pointeurs à des objets Polygone.

Traitement générique sur les polygones

```
// --- Utilisation polymorphique du vecteur
// --- sans connaître le type de Polygone
 for (int i=0; i<vPoly.size();i++)
    {
       vPoly[i]->afficher(cout);
       cout << endl;
    }
```

dynamiquement … à l'exécution
- détermination du type de l'objet
- appel de la bonne méthode.

Sortie :

```
Triangle  (10,10) (30,20) (10,30)
Rectangle (5,60) (100,60) (100,90) (5,90)
Rectangle (30,30) (60,30) (60,60) (30,60)
Carre     (0,0) (20,0) (20,20) (0,20)
Triangle  (20,20) (40,20) (20,40)
Carre     (50,50) (60,50) (60,60) (50,60)
```

Méthodes virtuelles

- Seule une fonction membre peut être virtuelle
- Un constructeur ne peut être virtuel

- Un destructeur peut être virtuel

Classes abstraites ou concrètes

- En général :
  - Création d'une classe => créer des instances.
  - ✓Ce n'est pas toujours le cas...

- Si création d'une classe **abstraite**,
  - ➢Pas de création d'instance de cette classe.

  Exemple :

  on peut se servir d'une telle classe pour regrouper des concepts sous une classe théorique.
- Bien plus que la théorie …
  - ➢imposer une interface générale que toutes les classes héritières devront réimplanter.

Programmation avancée en C++
GIF-1003

Thierry EUDE

Module 5 : La gestion de la mémoire

UNIVERSITÉ LAVAL
Département d'informatique
et de génie logiciel

Types de mémoire

- trois grandes zones de mémoire:
  - Zone de la pile d'appel
  - Zone d'adressage statique
  - Zone d'allocation dynamique sur le monceau

• Zone d'allocation dynamique

- Les deux premières zones ont leur utilité
  - demeurent insuffisantes pour la plupart des programmes sérieux.
- dans un programme, on ne peut
  - estimer la quantité de mémoire nécessaire
  - prévoir à quel moment celle-ci sera nécessaire.
  - réserver une très grande partie de la mémoire simplement parce qu'on prévoit en avoir besoin.

- utiliser **l'allocation dynamique** pour obtenir et libérer de la mémoire lorsqu'on en a vraiment besoin.

En C++

- pour allouer de la mémoire dynamiquement,

❑ opérateur **new** :

```
X* p = new X(arguments du constructeur);
```

- On doit absolument récupérer l'espace alloué par le **new**
  - sinon on laisse ce qu'on appelle des déchets (memory leaks).

❑ opérateur **delete** :

- On récupère l'espace occupé par un objet sur le **monceau**
- Appel de l'opérateur **delete** sur le pointeur obtenu à partir d'un **new** :

```
delete p;
```

▪ Copie d'objet

- comprendre ce qui se passe lorsqu'on copie un objet sur un autre.

- deux sortes de copie:
  - 1. La copie de **surface** (ou shallow copy)
  - 2. La copie en **profondeur** (ou deep copy)

Si on ne fait rien de particulier,

➢ le langage C++ fait toujours une copie de **surface** lors de la copie d'un objet.

Si on désire une copie en **profondeur**

➢ il faut l'implanter.

▪ Gestion de mémoire supplémentaire

- Définir le **constructeur** et le **destructeur** d'une classe ne permet pas de régler complètement la gestion de la mémoire.
  - Il faut aussi se prémunir contre l'**assignation**.

le langage C++ permet l'assignation de tout objet en générant un opérateur d'assignation qui fait une copie de **surface** (shallow copy)

✓dans le cas qui nous intéresse ce n'est pas du tout ce que l'on veut.

Le constructeur copie

- opération appelée de façon implicite par le langage:
  - Exemple:
    - lorsque l'on passe un objet par **valeur**
    - ou que l'on retourne un objet par **valeur**.
- Le constructeur copie est automatiquement généré par le langage C++
  - il fait une copie de surface,
  - ce qui encore ne fait pas notre bonheur...
- Ce constructeur doit absolument être défini lorsqu'on désire gérer des ressources avec la classe.
  - C'est le cas de la string.

En Bref : Forme canonique de Coplien

- Pour toute classe qui contient des allocations dynamique de la mémoire,

➢ implanter les 3 méthodes :

- destructeur
- constructeur de copie
- surcharge de operator =

▪ Simplement empêcher la copie ou l'assignation

- ❑ Pb. Vous avez une classe qui gère des ressources,
  - ○ allocation et libération,
- ❑ On ne veut pas qu'il y ait de copie d'un objet de ce type.

- ▪ Il est possible d'empêcher la copie d'un objet simplement en rendant **inaccessibles** les méthodes qui font la copie,
  - ○ le **constructeur copie** (passage et retour par valeur)
  - ○ l'**assignation** (assignation d'un objet à un autre objet du même type).

- ▪ déclarer les méthodes **constructeur copie** et **opérateur d'assignation** dans la section **privée** de la classe.
- ▪ Vous n'avez pas à implanter ces méthodes.

Programmation avancée en C++
GIF-1003

Thierry EUDE

Module 6.Gestion des erreurs et des exceptions : Fonctionnement et utilisation

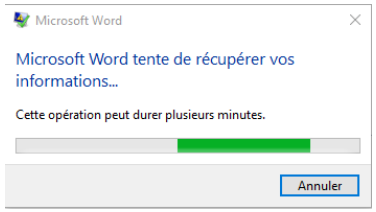
UNIVERSITÉ LAVAL
Département d'informatique
et de génie logiciel

Erreurs et exceptions

- On rencontre fréquemment des erreurs ou des exceptions dans les programmes.
- Erreurs :
  - erreurs d'entrée de données par l'usager
  - erreurs du matériel ou de périphérique
  - erreurs de limitations ressources du système
  - erreurs de programmation de composants logiciels.

Réagir à ces conditions d'erreur au niveau du code

- retourner un code d'erreur
- assigner une valeur à une variable globale d'erreur
- ignorer l'erreur (!)
- imprimer un message d'erreur
- arrêter le programme
- **lancer une exception.**



• Lancer une exception

- exceptions
  - ajoutées dans beaucoup de langages OO dont le C++.

- Le mécanisme des exceptions :
  - lancer des exceptions sur le site de l'erreur.
  - transférer le contrôle à une autre partie du code
    - gérer la situation avec compétence.

  Exemple :
  - «Incapable d'écrire ces octets» pourrait être devenir «Disque plein» à un niveau supérieur.

• Une fonction ne retourne pas normalement

une fonction lance une exception:

- recherche d'un site d'interception pour cette exception.
- La fonction ne retourne pas normalement
  - se termine immédiatement
  - Exécution transférée au site d'interception le plus proche (pour cette exception).

```
double r = divise (3, 0); //lance DivParZeroException

cout << "Résultat: " << r ; //Ne sera pas exécuté
```

```
double divise (int d, int n)
{
     if (n==0) throw DivParZeroException();
   return (double)d/n;
}


try
{
   // --- lance DivParZeroException
   double r = divise (3, 0);
   cout << "Résultat: " << r << endl;
}
catch (DivParZeroException& e)
{
   // --- Traitement de l'exception
}
```

• Hiérarchie d'exceptions

- La gestion des exceptions :
  - basée sur le type d'exception (la classe),
    - ➢ construire de petites classes appartenant à une hiérarchie.
- Librairie standard du C++ :
  - Hiérarchie d'exceptions
  - Du côté `logic_error`, i.e. erreur de programmation.
  - Du côté `runtime_error` par contre, il s'agit d'exception pouvant toujours se produire.
  - La théorie du contrat : exceptions attachées à cette hiérarchie

• Hiérarchie d'exceptions de contrat

- Si on veut gérer toutes les exceptions de contrat,
  - mentionner
    `catch (ContratException& e)`
  - toutes les exceptions de contrat : precondition, postcondition, invariant et assertion seront attrapées.
- Si on veut gérer seulement les préconditions,
  - mentionner :
    `catch (PreconditionException &e).`
- On pourrait gérer toutes les erreurs de programmation en mentionnant :
  `catch (logic_error& e)`

• Attraper toutes les exceptions et relancer

pour faire une action protectrice

pour faire un nettoyage quelconque.

```
try
{   // code
}
catch (...)
{   // Gestion de n'importe quelle exception
    throw; // Relance l'exception
}
```

• La pile d'appel

Gestion de la pile d'appel par le mécanisme des exceptions:

- Le flot linéaire d'exécution des instructions ne s'applique plus lorsque qu'une exception est lancée.
- Au moment où une exception se produit, le contrôle passe au site d'interception le plus proche pour cette exception.
- Tous les objets entre l'énoncé `throw` et l'énoncé `catch` seront détruits et leur destructeur est appelé.

(Normalement les objets sur la pile sont détruits lorsque l'exécution du programme atteint la fin d'un bloc ou d'une fonction.)

• Gestion des ressources

La possibilité qu'une exception puisse être lancée et qu'on ne puisse plus compter sur le flot linéaire d'exécution ajoute une préoccupation :

Sommes-nous «exception-safe» ?

Exemples:

```
 FILE* fichierP = fopen("input.dat");
foo(fichierP);     // peut lancer une exception
fclose(fichierP); // peut ne jamais être exécuté

Employe* eP = new Employe();
foo(eP);     // peut lancer une exception
delete eP;  // peut ne jamais être libérée
```

Programmation avancée en C++
GIF-1003
Hiver 2011

Thierry EUDE

8. Frameworks

UNIVERSITÉ LAVAL
Département d'informatique
et de génie logiciel

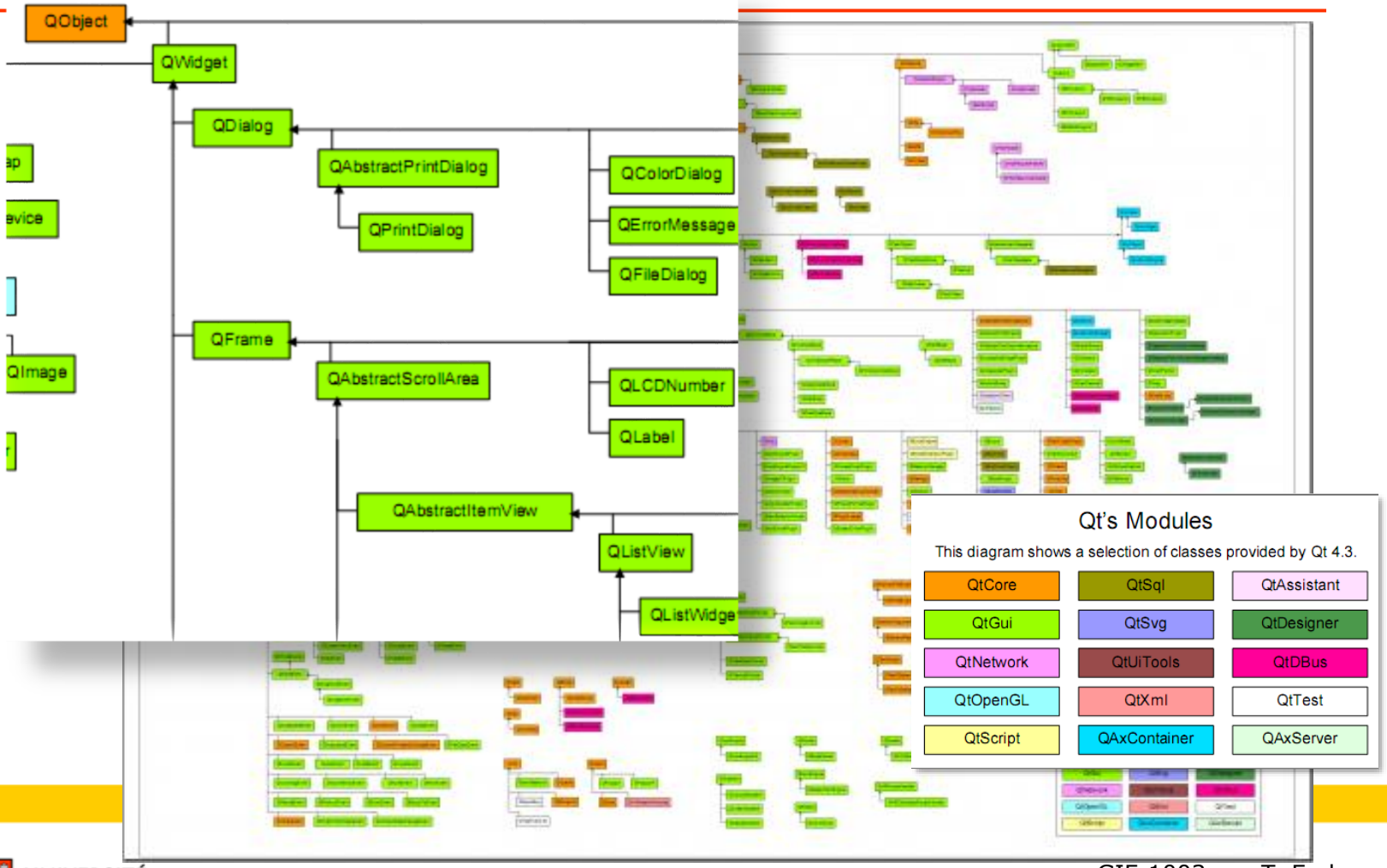
• Framework : définitions

- ❑ pour faciliter le développement d'applications dans un contexte déterminé.
- ❑ ensemble de classes qui coopèrent et permettent des conceptions réutilisables dans des catégories spécifiques de logiciels.
- ❑ utilisation ➔ sous-classage (héritage et polymorphisme) de classes abstraites.

• Framework et librairie de classes

- framework : très proche d'une librairie de classes
  - ➢ tous les deux des composants réutilisables codés.
- bibliothèque de classes = ensemble de composants,
  - ○ classes pouvant être réutilisées
    - ➢ mécanisme d'agrégation et d'héritage.
- frameworks = librairies de classes +
  - ○ relations,
  - ○ interactions entre des instances de classes.

Framework : Qt



Programmation avancée en C++
GIF-1003

Thierry EUDE

7. Classes et fonctions paramétrables, Librairie Standard du C++ (STL)

UNIVERSITÉ LAVAL
Département d'informatique
et de génie logiciel

7.1 Les classes et les fonctions paramétrables : Les modèles (Templates)

Université Laval
Département
d'informatique

UNIVERSITÉ LAVAL

• Mécanisme d'instanciation

- Processus pour obtenir une classe utilisable à partir d'une template : l'**instanciation** de la template.

```
          instanciation          instanciation
                    type spécifique       création d'objets
template ------------> classe -------------> objet

Pile<T> ---------> Pile<string> -------> m_laPileStr

Pile<T> ---------> Pile<double> ------> m_laPileReel
```

L'interface

```
template <typename T>
class TPile
{
public:
           TPile              (int capac = 0);
   bool    estVide            () const;
   bool    estPleine          () const;
   void    init               ();
   void    push               (const T&);
   T       pop                ();
   int     reqCapacite        () const;
   T       top                () const;

private:
   void    verifieInvariant   () const;

   std::vector<T> m_laPile;
   int            m_capacite;
};
```

Ajout de cet énoncé

Le type **double** devient **T**

Les méthodes : Notes supplémentaires

- Les méthodes ne sont plus dans un fichier *.cpp* mais dans le *.h* après la déclaration de la classe ou dans un *.hpp* inclut par le *.h*.

Exemple

```
template<typename T>
T max(const T& a, const T& b)
{
   return (a > b) ? a : b;
}

double x = 2.5;
double y = 4.0;
double z = max (x, y);
```

• Conclusion

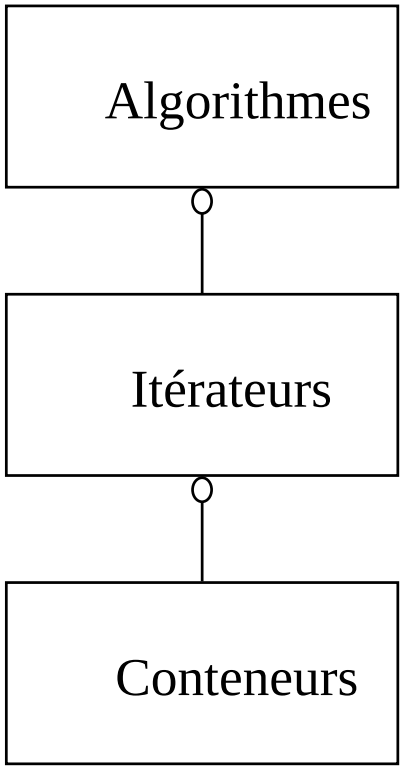
- L'utilisation des templates en C++ :
  - est incontournable
  - offre des possibilités très intéressantes
  - peut imposer des contraintes sur l'instanciation non documentées
  - les tests unitaires de ces classes ou fonctions sont difficiles dû au nombre illimité de possibilités de type utilisé.

7.2 Standard Template Library (STL)

Université Laval
Département
d'informatique

UNIVERSITÉ LAVAL

Structure de la librairie STL



- Composantes principales de STL:

❑ **Algorithmes**: définir des procédures de traitement.

❑ **Conteneurs**: gérer un ensemble d'espaces mémoire selon les structures de données.

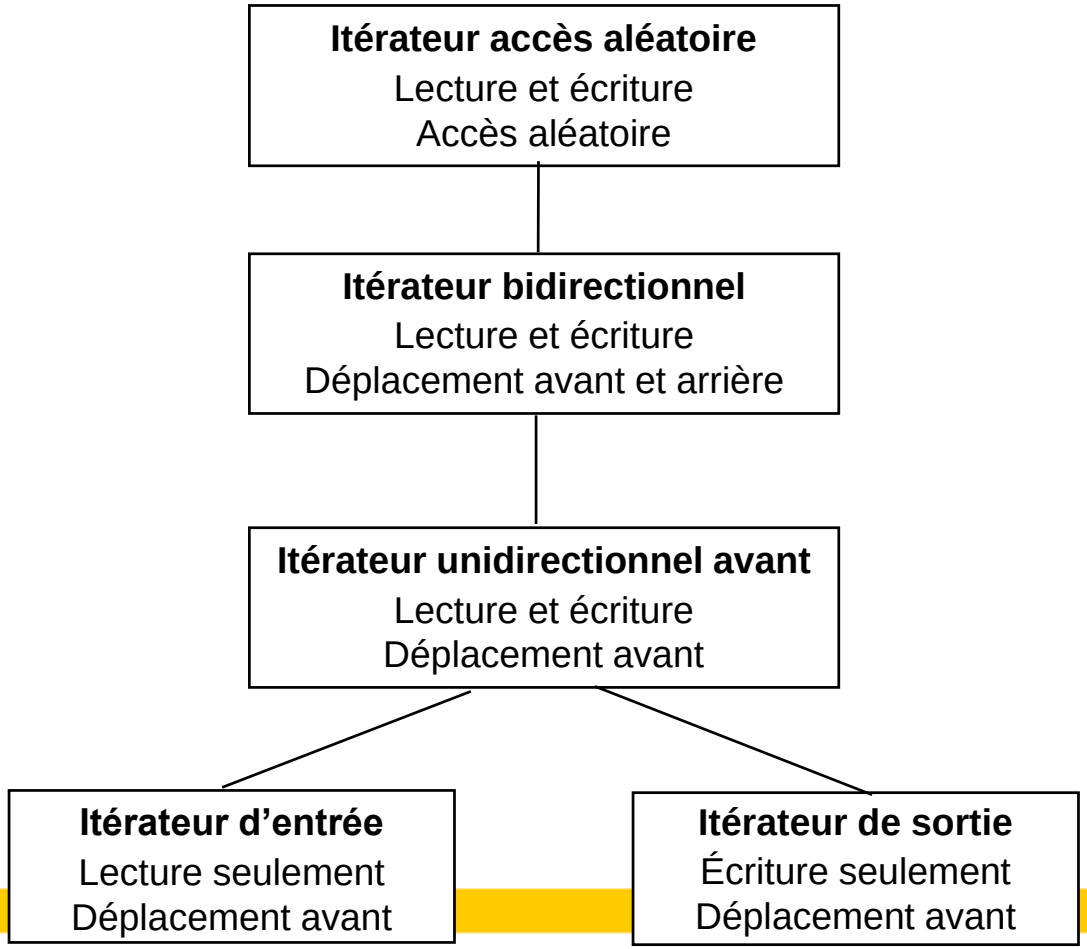
❑ **Itérateurs**: procurer à un algorithme une façon de traverser un conteneur.

Les itérateurs

- Comme avec les pointeurs sur un vecteur C++, l'itérateur peut être incrémenté ou décrémenté pour parcourir le conteneur (`iter++ ou iter--`).

- Lorsqu'on spécifie un intervalle avec deux itérateurs, `iter1` et `iter2`, il est nécessaire que `iter2` soit accessible à partir de `iter1`, i.e. un nombre fini de `iter1++` pour atteindre `iter2`.

Hiérarchie des itérateurs



- Les itérateurs

Les contenants dans STL

séquences	**associations**	**adaptateurs**
vector	set	stack
deque	multiset	queue
list	map	priority queue
	multimap	

- Les itérateurs
- Les contenants dans STL
  - **Les algorithmes génériques**

❑ Les algorithmes fournis avec STL sont découplés du conteneur sur lequel ils agissent

- utilisation des itérateurs.

❑ Tous les conteneurs dans la même catégorie peuvent utiliser les mêmes algorithmes.

MERCI DE VOTRE ATTENTION

TOUS MES VOEUX DE RÉUSSITE