> **Ce cours est une mise à jour du cours C++ de OpenClassRoom pour le mettre à jour pour le C++11/14. Le cours d'origine est consultable sur la page suivante : Programmez avec le langage C++, par Mathieu Nebra et Matthieu Schaller. Ce cours est sous licence CC-BY-NC-SA.**

Les classes templates

Voyons maintenant comment réaliser des classes template, c'est-à-dire des classes dont le type des arguments peut varier. Cela peut vous sembler effrayant, mais vous en avez déjà utilisé beaucoup. Pensez à vector ou deque par exemple. Il est temps de savoir réaliser des modèles de classes utilisables avec différents types.

Je vous propose de travailler sur un exemple que l'on pourrait trouver dans une bibliothèque comme Qt. Lorsque l'on veut dessiner des choses à l'écran, on utilise quelques formes de base qui servent à décomposer les objets plus complexes. L'une de ces formes est le rectangle qui, comme vous l'aurez certainement remarqué, est la forme des fenêtres ou des boutons, entre autres.

Quelles sont les propriétés d'un rectangle ?

Un rectangle a quatre côtés, une surface et un périmètre. Les deux derniers éléments peuvent être calculés si l'on connaît sa longueur et sa largeur. Voilà pour les attributs.

Quelles sont les actions qu'on peut associer à un rectangle ?

Ici, il y a beaucoup de choix. Nous opterons donc pour les actions suivantes : vérifier si un point est contenu dans le rectangle et déplacer le rectangle.

Nous pourrions donc modéliser notre classe comme illustré à la figure suivante.

Modélisation de la classe Rectangle

> On considère ici un rectangle parallèle aux bords de l'écran, ce qui permet de simplifier les positions en utilisant un seul et unique nombre par côté.

Le type des attributs

Maintenant que nous avons modélisé la classe, il est temps de réfléchir aux types des attributs, en l'occurrence la position des côtés.

Si l'on veut avoir une bonne précision, alors il faut utiliser des double ou des float. Si par contre on considère que, de toute façon, l'écran est composé de pixels, on peut se dire que l'utilisation d'int est largement suffisante.

Les deux options sont possibles et on peut très bien avoir besoin des deux approches dans un seul et même programme. Et c'est là que vous devriez tous me dire : « Mais alors, utilisons donc des templates ! ». Vous avez bien raison. Nous allons écrire une seule classe qui pourra être instanciée par le compilateur avec différents types.

Création de la classe

Je suis sûr que vous connaissez la syntaxe même si je ne vous l'ai pas encore donnée. Comme d'habitude, on déclare un type générique T. Puis on déclare notre classe.

```
template <typename T>
class Rectangle{
    //…
};
```

Notre type générique est reconnu par le compilateur à l'intérieur de la classe. Utilisons-le donc pour déclarer nos quatre attributs.

```
template <typename T>
class Rectangle{

   //…

private:

   //Les côtés du Rectangle
```

```
   T m_gauche;
   T m_droite;
   T m_haut;
   T m_bas;

};
```

Voilà. Jusque là, ce n'était pas bien difficile. Il ne nous reste plus qu'à écrire les méthodes.

Les méthodes

Les fonctions les plus simples à écrire sont certainement les accesseurs qui permettent de connaître la valeur des attributs. La hauteur d'un rectangle est évidemment la différence entre la position du haut et la position du bas. Comme vous vous en doutez, cette fonction est template puisque le type de retour de la fonction sera un T.

Une première méthode

Nous pouvons donc écrire la méthode suivante :

```
template <typename T>
class Rectangle{
public:
    //…

   T hauteur() const
   {
      return m_haut-m_bas;
   }

private:

   //Les cotes du Rectangle
   T m_gauche;
   T m_droite;
   T m_haut;
   T m_bas;
};
```

Vous remarquerez qu'il n'y a pas besoin de redéclarer le type template T juste avant la fonction membre puisque celui que nous avons déclaré

avant la classe reste valable pour tout ce qui se trouve à l'intérieur.

> Et si je veux mettre le corps de ma fonction à l'extérieur de ma classe ?

Bonne question. On prend souvent l'habitude de séparer le prototype de la définition. Et cela peut se faire aussi ici. Pour cela, on mettra le prototype dans la classe et la définition à l'extérieur mais il faut indiquer à nouveau qu'on utilise un type variable T :

```
template <typename T>
class Rectangle{
public:
    //…

   T hauteur() const;

    //…
};

template<typename T>
T Rectangle<T>::hauteur() const
{
   return m_haut-m_bas;
}
```

Vous remarquerez aussi l'utilisation du type template dans le nom de la classe puisque cette fonction sera instanciée de manière différente pour chaque T.

> Souvenez-vous que tout doit se trouver dans le fichier .h !

Une fonction un peu plus complexe

Une des fonctions que nous voulions écrire est celle permettant de vérifier si un point est contenu dans le rectangle ou pas. Pour cela, on doit passer un point (x;y) en argument à la fonction. Le type de ces arguments doit évidemment être T, de sorte que l'on puisse comparer les coordonnées sans avoir de conversions.

```
template <typename T>
```

```
class Rectangle{
public:

   //…

   bool estContenu(T x, T y) const
   {
      return (x >= m_gauche) && (x <= m_droite) && (y >=
m_bas) && (y <= m_haut);
   }

private:
    //…
};
```

Vous remarquerez à nouveau l'absence de redéfinition du type T. Quoi, je me répète ? C'est sûrement que cela devient clair pour vous.

Constructeur

Il ne nous reste plus qu'à traiter le cas du constructeur. À nouveau, rien de bien compliqué, on utilise simplement le type T défini avant la classe.

```
template <typename T>
class Rectangle{
public:

   Rectangle(T gauche, T droite, T haut, T bas)
        :m_gauche(gauche),
         m_droite(droite),
         m_haut(haut),
         m_bas(bas)
   {}

   //…
};
```

Et comme pour toutes les autres méthodes, on peut définir le constructeur à l'extérieur de la classe. Vous êtes bientôt des pros, je vous laisse donc essayer seuls.

> On pourrait ajouter une fonction appelée dans le constructeur qui vérifie que le haut se trouve bien au-dessus du bas et de même pour droite et gauche.

Finalement, voyons comment utiliser cette classe.

Instanciation d'une classe template

Il fallait bien y arriver un jour ! Comment crée-t-on un objet d'une classe template et en particulier de notre classe Rectangle ?

En fait, je suis sûr que vous le savez déjà. Cela fait longtemps que vous créez des objets à partir de la classe template vector ou map. Si l'on veut un Rectangle composé de double, on devra écrire :

```
int main()
{
   Rectangle<double> monRectangle(1.0, 4.5, 3.1, 5.2);

   return 0;
}
```

L'utilisation des fonctions se fait ensuite comme d'habitude :

```
int main()
{
   Rectangle<double> monRectangle(1.0, 4.5, 3.1, 5.2);

   cout << monRectangle.hauteur() << endl;

   return 0;
}
```

Pour terminer ce chapitre, je vous propose d'ajouter quelques méthodes à cette classe. Je vous parlais d'une méthode deplacer() qui change la position du rectangle. Essayez aussi d'écrire les méthodes surface() et perimetre().

Enfin, pour bien tester tous ces concepts, vous pouvez refaire la classe ZFraction de sorte que l'on puisse spécifier le type à utiliser pour stocker le numérateur et le dénominateur. Bonne chance !

En résumé