Programmation orientée objet

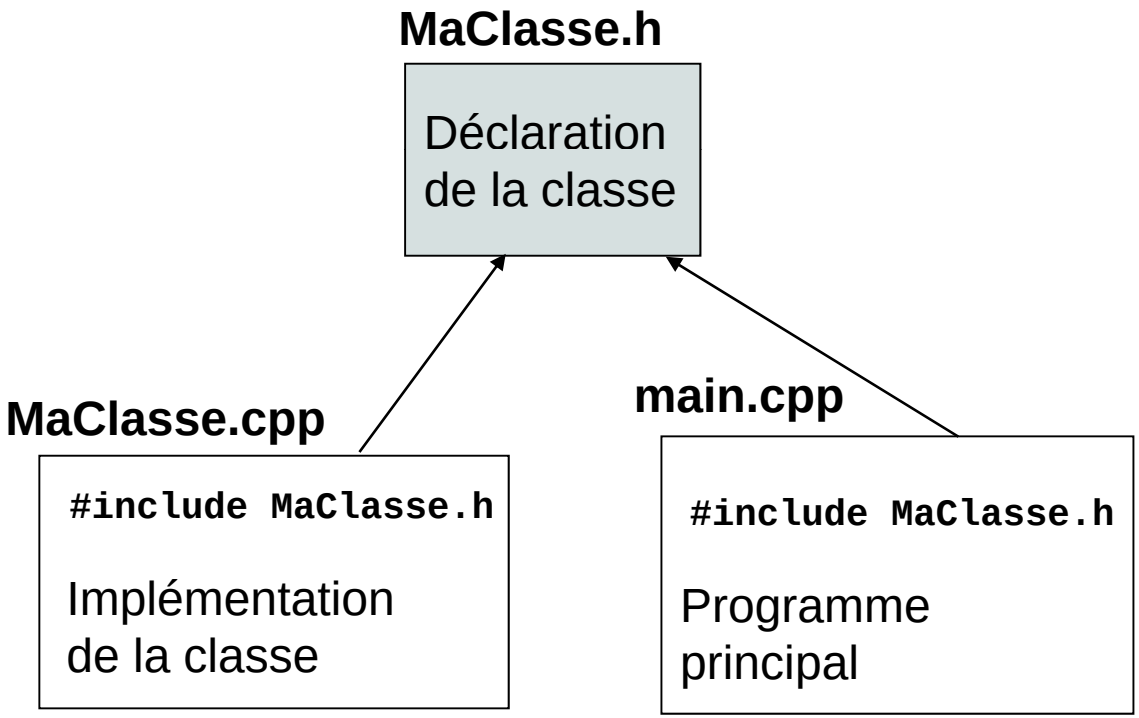
Compilation séparée

Michel Gagnon

Compilation séparée

- Pour compiler un programme principal, il suffit de connaître les déclarations de classes
- Les implémentations des classes seront compilées séparément
- Une fois compilés le programme principal et les implémentations de classe, les fichiers ainsi obtenus sont combinés pour produire l'application exécutable (étape de *liaison*)

Compilation séparée (suite)



Compilation séparée et classes génériques

- Supposons, par exemple, une classe Liste générique ayant une méthode **`trouver()`** qui elle-même exécute une méthode de l'objet de type générique qui lui est passé en paramètre
- Supposons qu'il s'agit de la méthode **`uneMethode()`**
- Il faut donc que cette méthode soit définie pour le type en question
- Ainsi, dans le programme principal, la déclaration suivante devrait etre refusée par le compilateur, si la classe Point ne possède pas la méthode uneMethode() **`:`**

```
Liste< Point > listeDePoints;
```

Compilation séparée et classes génériques (suite)

- Si la classe est générique, il n'est donc pas suffisant d'inclure seulement la déclaration de la classe
- Le compilateur **doit** savoir si toutes les méthodes implémentées sont valides pour le type spécifié

Compilation séparée et classes génériques - exemple

```
template <typename T>
class Liste
{
 public:
    bool trouver(T objet);
};

template <typename T>
bool Liste< T >::trouver(T objet)
{
  ...
  for (int = 0; i < taille; ++i) {
     ...
     if (objet.uneMethode()) {
        ...
     }
  }
  ...
}
```

Cette méthode doit exister pour la classe T.

Supposons que l'implémentation est dans le .cpp

Lorsqu'on compile main.cpp, on sait que la méthode trouver() existe, mais on ne sait pas qu'elle exige que l'objet de la classe T contienne la méthode uneMethode().

liste.h

```
template <typename T>
class Liste
{
 public:
    bool trouver(T objet);
    ...
};
```

point.h

```
class Point
{
 public:
    Point() ;
    double getX();
    double getY();
    ...
};
```

liste.cpp

```
template <typename T>
bool Liste< T >::trouver(T objet)
{
  ...
  for (int = 0; i < taille; ++i)
   {
      ...
      if (objet.uneMethode()) {
          ...
      }
  }
  ...
}
```

La classe Point ne contient pas la méthode uneMethode().

main.cpp

```
#include “liste.h”
#include “point.h”
int main()
{
   Liste< Point > listeDePoints;
   ...
}
```

Supposons que l'implémentation est dans le .h

liste.h

```
template <typename T>
class Liste
{
public:
   bool trouver(T objet);
   ...
};

template <typename T>
bool Liste< T >::trouver(T objet)
{
  ...
  for (int = 0; i < taille; ++i)
   {
      ...
      if (objet.uneMethode()) {
          ...
      }
  }
  ...
}
```

Lorsqu'on compile main.cpp, on sait maintenant que la méthode trouver() exige que l'objet de la classe T contienne la méthode uneMethode().

point.h

```
class Point
{
 public:
    Point() ;
    double getX();
    double getY();
    ...
};
```

main.cpp

```
#include “liste.h”
#include “point.h”
int main()
{
   Liste< Point > listeDePoints;
   ...
}
```

Compilation séparée et classes génériques (suite)