Les classes : héritage et polymorphisme

Loïc Joly, Michel Simatic et Amina Guermouche

Télécom SudParis

2 avril 2025

Plan

1. Héritage
2. Copie des objets et héritage
3. Fonctions virtuelles
4. Fonction virtuelles pures

Héritage

1. → Héritage
2. Copie des objets et héritage
3. Fonctions virtuelles
4. Fonction virtuelles pures

Généralités

- Syntaxe : "La classe `Circle` hérite de la classe `Shape`" se traduit par `class Circle : public Shape`

```
class Shape{
  private:
     int pos_x, pos_y;
  protected:
     // Liste des attributs visibles par les classes filles
  public:
     // Liste des fonctions/méthodes
}
class Circle : public Shape {
  private:
     int rayon;
  public:
     // Liste des fonctions/méthodes
  }
```

- Différence entre `class` et `struct` :
  - Dans `class`, tout est `private` par défaut.
  - Dans `struct`, tout est `public` par défaut.

Copie des objets et héritage

1. Héritage
2. → Copie des objets et héritage
3. Fonctions virtuelles
4. Fonction virtuelles pures

Slicing

Lorsque l'on veut copier un objet de type "classe fille" (par exemple `Circle`) dans un objet de type "classe mère" (par exemple `Shape`), un phénomène appelé "*slicing*" se produit : seuls les champs définis dans la classe mère sont gardés dans l'objet créé.

```
class Shape{
  int x, y;
};

class Circle : public Shape{
  int rayon;
};
int main()
{
  Circle c{...}; // Appel au constructeur
  Shape s = c; // s ne contient que les elements de Shape (x et y), car slicing !
}
```

Les pointeurs évitent le slicing

Afin d'éviter le problème de *slicing*, on doit utiliser des pointeurs vers des objets.

```
class Shape{
  int x, y;
};

class Circle : public Shape{
  int rayon;
};

int main()
{
  vector<Shape*> v;
  Circle *c = new Circle(...);
  v.push_back(c); // Meme si v contient des Shape*, le pointeur pointe sur un objet de type Circle.
}
```

Fonctions virtuelles

1. Héritage
2. Copie des objets et héritage
3. → Fonctions virtuelles
4. Fonction virtuelles pures

Généralités sur les méthodes virtuelles

- Une méthode virtuelle est définie dans la classe mère et dans au moins une classe fille.
- La signature des fonctions doit être la même pour toutes les redéfinitions. Au niveau des classes filles, on parlera d'*override*.

```
class Shape{
  virtual void print(); // La fonction est definie dans le .cpp correspondant.
};

class Circle : public Shape{
  void print() override;
};
```

- Le mot clé *override* n'est pas obligatoire. Il est cependant recommandé car il permet au compilateur de nous signaler des erreurs lors de la redéfinition de la fonction (paramètres différents, type de retour différent...).

Mot-clé *virtual*

Grâce au mot clé `virtual`, le compilateur va regarder le type de l'objet appelant la méthode afin d'utiliser la bonne fonction.

```
class Shape{
  virtual void print(); // la fonction est definie dans le .cpp correspondant
};

class Circle : public Shape{
  void print() override;
};

int main()
{
  vector<Shape*> s;
  Circle *c = new Circle(...);
  s.push_back(c);
  s[0]->print(); // Appel a la fonction print() de la classe Circle car la premiere case de v contient un pointeur vers un objet de type Circle.
}
```

Fonction virtuelles pures

1. Héritage
2. Copie des objets et héritage
3. Fonctions virtuelles
4. → Fonction virtuelles pures

Généralités sur les méthodes virtuelles pures

- Parfois, la définition d'une méthode virtuelle dans la classe mère n'a pas de sens. Dans ce cas, on parle de méthode virtuelle pure.

```
class Shape{
  virtual void print() = 0; // Ici print est une fonction virtuelle pure.
};

class Circle : public Shape{
  void print() override;
};
```

- NB :

1. Toutes les classes héritant d'une classe abstraite doivent implémenter les fonctions virtuelles pures de la classe mère (sinon elles aussi seront abstraites).
2. Il est possible de définir la fonction virtuelle pure dans la classe mère. L'intérêt de cette fonction est de rendre la classe abstraite et donc non instantiable.
3. On peut appeler le constructeur d'une classe abstraite dans une classe fille. De ce fait, le constructeur d'une classe abstraite doit être protected étant donné que seules ses classes filles vont l'appeler.

Questions ?

1. Héritage
2. Copie des objets et héritage
3. Fonctions virtuelles
4. Fonction virtuelles pures