Dynamic Cast

AP2 - programmation objet en C++

Semestre 2, année 2009-2010

Département d'informatique
IUT Bordeaux 1

Février 2010

Le projet

Situation : un aquarium contient des éléments qui peuvent être des décors ou des poissons

- tous les éléments ont un nom
- les décors ont un prix
- les poissons ont une couleur

Exemple

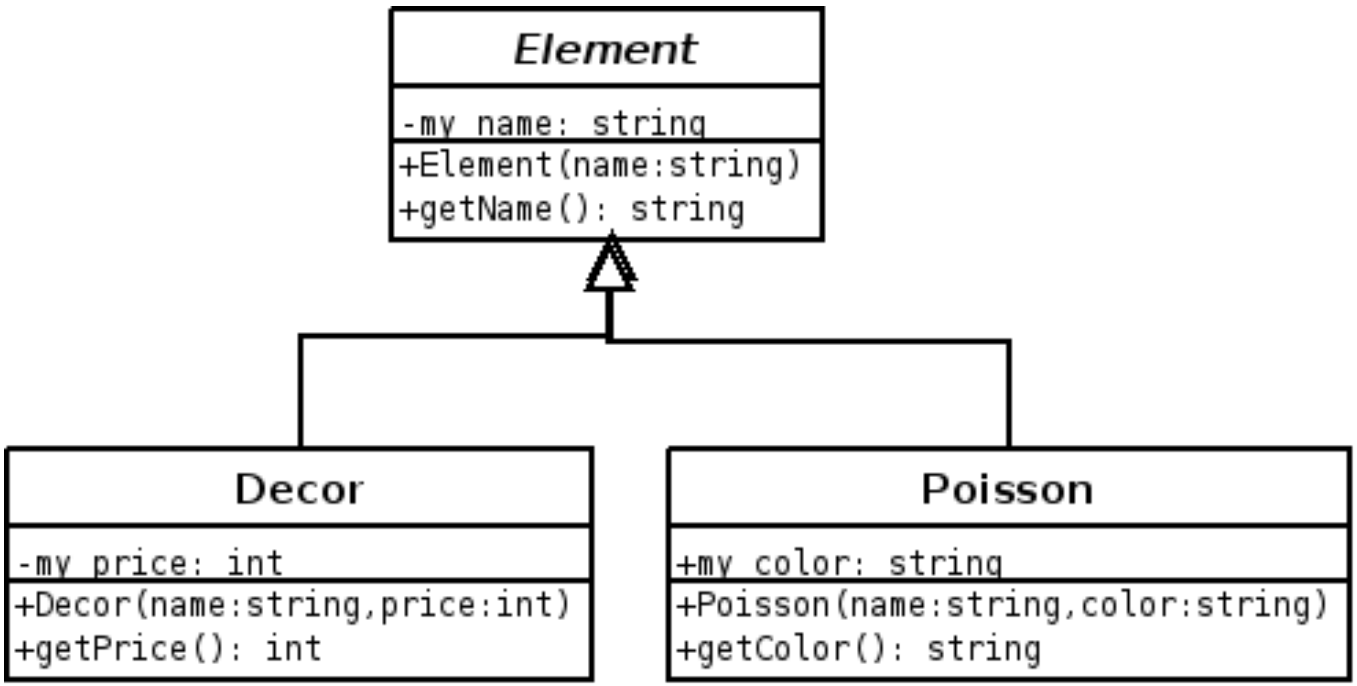
pseudo-code

- ajouter décor bocal 30 Euros
- ajouter poisson bubulle rouge
- ajouter décor plante 2 Euros
- ajouter poisson wanda jaune
- ...
- pour tout élément
  - afficher son nom
  - si c'est un décor, afficher son prix
  - si c'est un poisson, afficher sa couleur

Classes

- classe abstraite `Element`,
  avec attribut commun `name`
- 2 classes dérivées :
  - `Decor`,
    avec attribut spécifique `price`
  - `Poisson`,
    avec attribut spécifique `color`

Diagramme de classes



`Element` est une classe abstraite : tous les Elements appartiennent à une des classes dérivées.

remplissage ...

aquarium.cc

```
#include "Element.h"
#include "Decor.h"
#include "Poisson.h"
...
int main() {
 Element * aquarium[5];
 aquarium[0] = new Decor   ("bocal",    40);
 aquarium[1] = new Decor   ("cailloux", 2);
 aquarium[2] = new Decor   ("eau",      1);
 aquarium[3] = new Poisson ("bubulle", "rouge");
 aquarium[4] = new Poisson ("wanda",   "jaune");
...
}
```

parcours

aquarium.cc, extrait

```
int main() {
   Element * aquarium[5];
   ...
   for (int i = 0; i< 5; i++) {
      cout << i << " => " << aquarium[i] -> getName();

      // si c'est un poisson ...
      // si c'est un décor ...

      cout << endl;
   }
}
```

Résultat attendu

Exécution

```
0 => bocal, qui coute 40E
1 => cailloux, qui coute 2E
2 => eau, qui coute 1E
3 => bubulle, qui est un poisson rouge
4 => nestor, qui est un poisson jaune
```

transypage

L'opérateur `dynamic_cast` permet de convertir un pointeur vers une classe plus spécifique.

Exemple

```
Element * pElement;
Poisson * pPoisson;
...
pPoisson = dynamic_cast<Poisson *> pElement;
```

Si la conversion échoue, il renvoie `NULL`.

utilisation du transypage

aquarium.cc, dans la boucle

```
...
// si c'est un poisson ...
 Poisson * pPoisson = dynamic_cast<Poisson *> (aquarium[i]);
 if (pPoisson != NULL) {
   cout << ", qui est un poisson " << pPoisson->getColor();
 }  else {

 // si c'est un décor ...
   Decor * pDecor = dynamic_cast<Decor *> (aquarium[i]);
   if (pDecor != NULL) {
      cout << ", qui coute " << pDecor->getPrice() << "E";
   }
 ...
```

Remarque

Pour le transtypage

- la classe de base doit être abstraite
- c'est-à-dire contenir au moins une fonction virtuelle

par exemple, destructeur virtuel

Element.h

```
class Element {
  ...
  virtual ~Element();
}
```

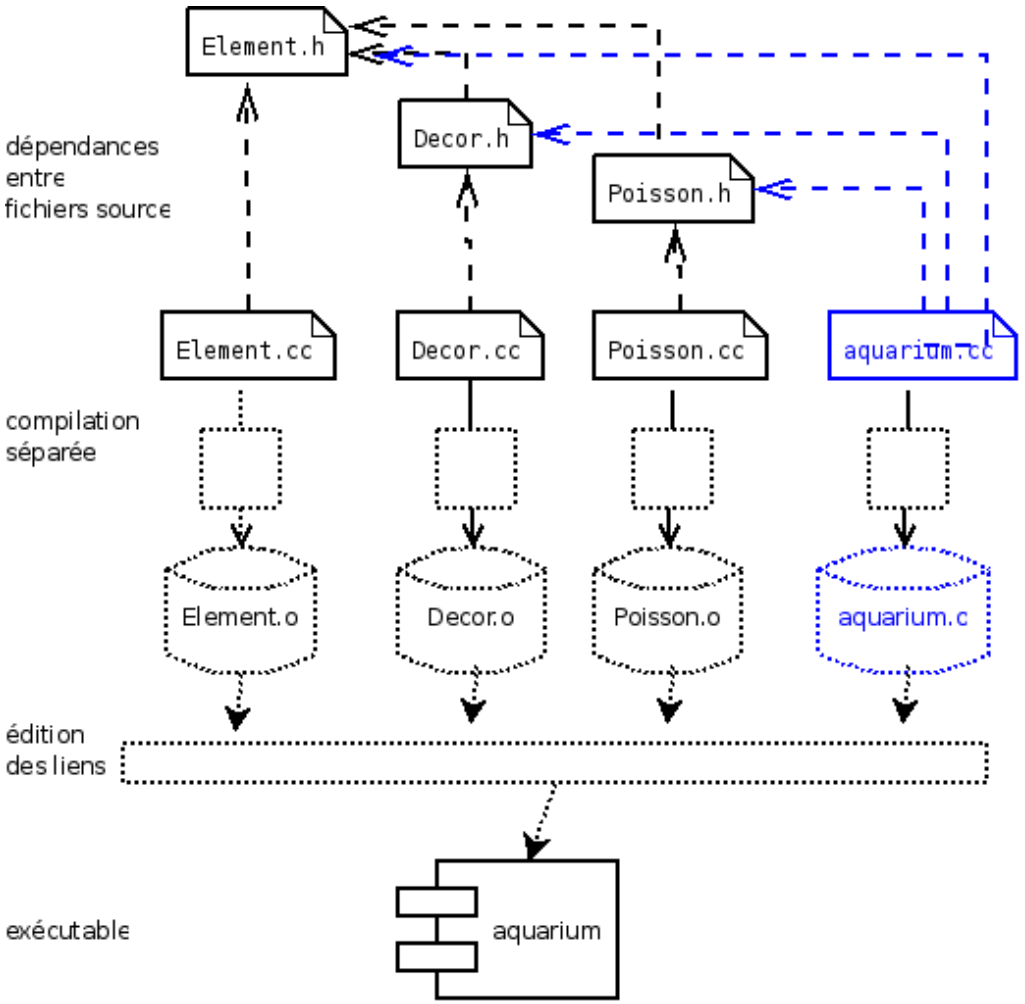
Element.cc

```
Element::~Element() {
  // rien
}
```

Annexes

- Vue générale du projet
- `Makefile`
- sources .cc et .h

Projet


Element.h
Decor.h
Poisson.h
dépendances
entre
fichiers source
Element.cc
Decor.cc
Poisson.cc
aquarium.cc
compilation
séparée
Element.o
Decor.o
Poisson.o
aquarium.o
édition
des liens
exécutable
aquarium

Makefile

```
CXXFLAGS=-Wall -pedantic

aquarium: aquarium.o Element.o Decor.o Poisson.o
        $(LINK.cc) -o $@ $^

clean:
        -rm *.o *~

# DO NOT DELETE

aquarium.o: Element.h Decor.h Poisson.h
Decor.o: Decor.h Element.h
Element.o: Element.h
Poisson.o: Element.h Poisson.h
```

Element.h

```
#ifndef ELEMENT_H
#define ELEMENT_H
#include <string>
using namespace std;

class Element
{
protected:
  string my_name;

public:
  Element(string name);
  string getName() const;

  virtual ~Element();
};
#endif
```

Element.cc

```
#include "Element.h"

Element::Element(string name) {
  my_name = name;
}

string Element::getName() const {
  return my_name;
}

Element::~Element() {
  // rien
}
```

Decor.h

```
#ifndef DECOR_H
#define DECOR_H

#include "Element.h"

#include <string>
using namespace std;

class Decor : public Element
{
 private:
  int my_price;
 public:
  Decor (string name, int price);
  int getPrice() const;
};
#endif
```

Decor.cc

```
#include "Decor.h"

Decor::Decor (string name, int price) :
    Element(name) {
    my_price = price;
 }

int Decor::getPrice() const {
  return my_price;
}
```

Poisson.h

```
#ifndef POISSON_H
#define POISSON_H

#include "Element.h"

#include <string>
using namespace std;

class Poisson : public Element
{
private:
 string my_color;

public:
 Poisson(string name, string color);
 string getColor() const;
};
#endif
```

Poisson.cc

```
#include "Element.h"
#include "Poisson.h"

#include <string>
using namespace std;

Poisson::Poisson(string name, string color) :
  Element(name)
{
  my_color = color;
}

string Poisson::getColor() const {
  return my_color;
}
```

aquarium.cc

```
// aquarium, avec des Elements :
//poissons, décors

#include <iostream>
#include <string>

#include "Element.h"
#include "Decor.h"
#include "Poisson.h"

using namespace std;

int main() {
 Element * aquarium[5];

 aquarium[0] = new Decor("bocal",40);
 aquarium[1] = new Decor("cailloux",2);
 aquarium[2] = new Decor("eau",1);
 aquarium[3] = new Poisson("bubulle",
                           "rouge");
 aquarium[4] = new Poisson("nestor",
                           "jaune");

 for (int i = 0; i< 5; i++) {
   cout << i << " => "
        << aquarium[i] -> getName();

   // si c'est un poisson ...
   Poisson * pPoisson
      = dynamic_cast<Poisson *> (aquarium[i]);
   if (pPoisson != NULL) {
     cout << ", qui est un poisson "
          << pPoisson->getColor();
   } else {
     Decor * pDecor
      = dynamic_cast<Decor *> (aquarium[i]);
     if (pDecor != NULL) {
       cout << ", qui coute "
            << pDecor->getPrice() << "E";
     }
   }
   cout << endl;
 }
}
```