```
////////////////////////////////////////////////////////////////////////////
// fichier Element.h
////////////////////////////////////////////////////////////////////////////
#ifndef ELEMENT_H
#define ELEMENT_H

class Element {
  protected:
    int my_x, my_y, my_z;  // coordonnées
  public:
    Element( int x, int y, int z );
    virtual ~Element();
    virtual bool mobile() const = 0;
};

#endif // #ifndef ELEMENT_H

////////////////////////////////////////////////////////////////////////////
// fichier Element.cc
////////////////////////////////////////////////////////////////////////////
#include <iostream>
#include "Element.h"

using namespace std;

Element::Element( int x, int y, int z )
{
  cout << "appel constructeur Element..." << endl;
  my_x = x;
  my_y = y;
  my_z = z;
}

Element::~Element()
{
  cout << "appel destructeur Element..." << endl;
}
```



```
////////////////////////////////////////////////////////////////////////////
// fichier Animal.h
////////////////////////////////////////////////////////////////////////////
#ifndef ANIMAL_H
#define ANIMAL_H

#include <string>
#include "Element.h" // declarons la classe de base

using namespace std;

class Animal : public Element {   // heritage ...
 private:
   string my_n;   // on peut rajouter le nom de l'animal
 public:
   Animal( string n, int x, int y, int z );
   ~Animal();
   bool mobile() const;
   void deplacer( int x, int y, int z );
};

#endif // #ifndef ANIMAL_H

////////////////////////////////////////////////////////////////////////////
// fichier Animal.cc
////////////////////////////////////////////////////////////////////////////
#include <iostream>
#include <string>
#include "Element.h"
#include "Animal.h"

using namespace std;

Animal::Animal( string n, int x, int y, int z )
: Element (x, y, z)
{
  cout << "appel constructeur Animal..." << endl;
  my_n = n;
}

Animal::~Animal()
{
  cout << "appel destructeur Animal..." << endl;
}

bool
Animal::mobile() const
{
  return true;
}

void
Animal::deplacer( int x, int y, int z )
{
  cout << "deplacement de ("<< my_x << "," << my_y << "," << my_z << ")";
  my_x += x;
  my_y += y;
  my_z += z;
  cout << " vers ("  << my_x << "," << my_y << "," << my_z << ")" <<endl;
}
```

```
////////////////////////////////////////////////////////////////////////////
// fichier Autre.h
////////////////////////////////////////////////////////////////////////////
#ifndef AUTRE_H
#define AUTRE_H

#include <string>
#include "Element.h" // declarons la classe de base

using namespace std;

class Autre : public Element {   // heritage ...
 private:
   string my_n;  // on peut rajouter le nom
 public:
   Autre( string n, int x, int y, int z );
   ~Autre();
   bool mobile() const;
};

#endif // #ifndef AUTRE_H

////////////////////////////////////////////////////////////////////////////
// fichier Autre.cc
////////////////////////////////////////////////////////////////////////////
#include <iostream>
#include <string>
#include "Element.h"
#include "Autre.h"

using namespace std;

Autre::Autre( string n, int x, int y, int z )
: Element (x, y, z)
{
  cout << "appel constructeur Autre..." << endl;
  my_n = n;
}

Autre::~Autre()
{
  cout << "appel destructeur Autre..." << endl;
}

bool
Autre::mobile() const
{
  return false;
}
```



```
////////////////////////////////////////////////////////////////////////////
// fichier "main.cc"
////////////////////////////////////////////////////////////////////////////
#include <iostream>
#include "Element.h"
#include "Animal.h"
#include "Autre.h"

using namespace std;

int
main()
{
  Animal poisson("poisson", 2, 5, 1);

  Autre algue("algue", 5, 9, 10);


  Element *pelement;

  //utilisation du cast pour transtypage
  pelement = &algue;
  if (pelement->mobile())
    dynamic_cast<Animal *>(pelement)->deplacer(1,0,0);

  pelement = &poisson;
  if (pelement->mobile())
    dynamic_cast<Animal *>(pelement)->deplacer(1,0,0);

  // parcours d'un tableau de pointeurs
  cout << endl << "Parcours d'un tableau de pointeurs" << endl;
  Element* tab[2];
  tab[0] = &algue;
  tab[1] = &poisson;
  for (int i=0 ; i<2 ; i++)
    if (tab[i]->mobile())
      dynamic_cast<Animal *>(tab[i])->deplacer(1,0,0);

  return 0;
}

*****************************************************************************
* TRACE
*****************************************************************************
appel constructeur Element...
appel constructeur Animal...
appel constructeur Element...
appel constructeur Autre...
deplacement de (2,5,1) vers (3,5,1)

Parcours d'un tableau de pointeurs
deplacement de (3,5,1) vers (4,5,1)
appel destructeur Autre...
appel destructeur Element...
appel destructeur Animal...
appel destructeur Element...
```