## 1. Vos classes comportent des fonctions d'accès

Ces fonctions d'accès seront utilisées dans une classe amie abstraite "interface"... Voici un exemple (interfa1.cpp):

```
#include <iostream.h>
class Entrepot{
  friend class interface;
  private:
     long nbBennes;
  public:
     Entrepot(long n) : nbBennes(n) {  }
     long getBennes(void) const { return nbBennes; }
};

class SalleElectrolyse{
  friend class interface;
  private:
     double tauxProduction;
     double nbMegots;
     double nbAnodes;
     Entrepot * e;
  public:
     SalleElectrolyse(double taux, Entrepot * ptrE)
        : tauxProduction(taux), nbMegots(0), nbAnodes(20), e(ptrE) {  }
     double getTauxProduction(void) const { return tauxProduction; }
     double getNbMegots(void) const { return nbMegots; }
     double getNbAnodes(void) const { return nbAnodes; }
};

class interface{
  protected: // sont protected pour l'accès dans « interfaceDos »
     Entrepot * e;
     SalleElectrolyse * s;
  public:
     interface(Entrepot * ptrE, SalleElectrolyse * ptrS) : e(ptrE), s(ptrS) {  }
     // toutes les fonctions d'interface
     virtual void afficherEntrepot(void) const = 0;
     virtual void afficherTauxProduction(void) const = 0;
     virtual void afficherMegotsElectrolyse(void) const = 0;
     virtual void afficherAnodesElectrolyse(void) const = 0;
};

class interfaceDos : public interface{
  public:
     interfaceDos(Entrepot * ptrE, SalleElectrolyse * ptrS)
        : interface(ptrE, ptrS) {  }
     void afficherEntrepot(void) const {
        cout << "nb de bennes = " << e->getBennes() << endl;
     }
     void afficherTauxProduction(void) const {
        cout << "taux de prod. = " << s->getTauxProduction() << endl;
     }
     void afficherMegotsElectrolyse(void) const {
        cout << "nb mégots = " << s->getNbMegots() << endl;
     }
     void afficherAnodesElectrolyse(void) const {
        cout << "nb anodes = " << s->getNbAnodes() << endl;
     }
};
```

```
int main(void){
   Entrepot E(100);
   SalleElectrolyse S(62.3, &E);

   interfaceDos projecteur(&E, &S);

   projecteur.afficherEntrepot();
   projecteur.afficherTauxProduction();
   projecteur.afficherMegotsElectrolyse();
   projecteur.afficherAnodesElectrolyse();

   return 0;
}
```

## 2. Vos classes ne comportent pas de fonctions d'accès

Ces fonctions d'accès seront codées dans une classe amie abstraite "interface"... puisque malheureusement ou heureusement (c'est selon), l'amitié ne s'hérite pas! Ainsi, la classe « interface » est amie de « Entrepot » mais pas la classe « interfaceDOS ». (interfa2.cpp)

```
#include <iostream.h>
class Entrepot{
  friend class interface;
  private:
     long nbBennes;
  public:
     Entrepot(long n) : nbBennes(n) {  }
};

class SalleElectrolyse{
  friend class interface;
  private:
     double tauxProduction;
     double nbMegots;
     double nbAnodes;
     Entrepot * e;
  public:
     SalleElectrolyse(double taux, Entrepot * ptrE)
        : tauxProduction(taux), nbMegots(0), nbAnodes(20), e(ptrE) {  }
};

class interface{
  private:
     Entrepot * e;
     SalleElectrolyse * s;
  public:
     interface(Entrepot * ptrE, SalleElectrolyse * ptrS) : e(ptrE), s(ptrS) {  }
     // toutes les fonctions d'interface
     virtual void afficherEntrepot(void) const = 0;
     virtual void afficherTauxProduction(void) const = 0;
     virtual void afficherMegotsElectrolyse(void) const = 0;
     virtual void afficherAnodesElectrolyse(void) const = 0;

     // toutes les fonctions d'accès regroupées ici…
     long getBennes(void) const { return e->nbBennes; }
     double getTauxProduction(void) const { return s->tauxProduction; }
     double getNbMegotsElectro(void) const { return s->nbMegots; }
     double getNbAnodesElectro(void) const { return s->nbAnodes; }
};
```

```
class interfaceDos : public interface{
  public:
     interfaceDos(Entrepot * ptrE, SalleElectrolyse * ptrS)
        : interface(ptrE, ptrS) {  }
     void afficherEntrepot(void) const {
        cout << "nb de bennes = " << getBennes() << endl;
     }
     void afficherTauxProduction(void) const {
        cout << "taux de prod. = " << getTauxProduction() << endl;
     }
     void afficherMegotsElectrolyse(void) const {
        cout << "nb mégots = " << getNbMegotsElectro() << endl;
     }
     void afficherAnodesElectrolyse(void) const {
        cout << "nb anodes = " << getNbAnodesElectro() << endl;
     }
};

int main(void){
   Entrepot E(100);
   SalleElectrolyse S(62.3, &E);

   interfaceDos projecteur(&E, &S);

   projecteur.afficherEntrepot();
   projecteur.afficherTauxProduction();
   projecteur.afficherMegotsElectrolyse();
   projecteur.afficherAnodesElectrolyse();

   return 0;
}
```

## 3. Souplesse du code

Quelque soit la solution choisie, on y gagne. Si plus tard, on désire construire un interface pour Windows, on pourra créer la classe « interfaceWin » qui héritera de la classe « interface » :

```
class interfaceWin : public interface{
  public:
     interfaceWin (Entrepot * ptrE, SalleElectrolyse * ptrS)
        : interface(ptrE, ptrS) {  }
     void afficherEntrepot(void) const {
        // code spécifique à Windows
     }
     void afficherTauxProduction(void) const {
        // code spécifique à Windows
     }
     void afficherMegotsElectrolyse(void) const {
        // code spécifique à Windows
     }
     void afficherAnodesElectrolyse(void) const {
        // code spécifique à Windows
     }
};
```