(Suite Chapitre 3) Classes abstraites

(3.6.1) Une fonction-membre virtuelle d'une classe est dite *purement virtuelle* lorsque sa déclaration est suivie de `= 0`, comme ci-dessous :

```
class A
{
        .....
        virtual truc machin() = 0;
        .....
};
```

Une fonction purement virtuelle n'a pas de définition dans la classe. Elle ne peut qu'être surchargée dans les classes dérivées.

(3.6.2) Une classe comportant au moins une fonction-membre purement virtuelle est appelée *classe abstraite*. A retenir :

Aucune instance d'une classe abstraite ne peut être créée.

L'intérêt d'une classe abstraite est uniquement de servir de "canevas" à ses classes dérivées, en déclarant l'interface minimale commune à tous ses descendants.

(3.6.3) **Exemple**

Il n'est pas rare qu'au cours d'un programme, on ait besoin de stocker temporairement des informations en mémoire, pour un traitement ultérieur. Une structure de stockage doit permettre deux actions principales :

- mettre un nouvel élément,
- extraire un élément.

On parle de *pile* lorsque l'élément extrait est le dernier en date à avoir été mis (structure *LIFO* pour *Last In, First Out*) et de *queue* ou *file d'attente* lorsque l'élément extrait est le premier en date à avoir été mis (structure *FIFO* pour *First In, First Out*).

Voulant programmer une classe `Pile` et une classe `Queue`, nous commencerons par écrire une classe abstraite `Boite` décrivant la partie commune à ces deux classes :

```
//---------------------- déclaration de la classe Boite ----------------------

class Boite            // classe abstraite décrivant une structure de stockage
                       // les éléments stockés sont de type "pointeur sur Objet"
                       // la classe Objet est supposée déjà déclarée
{
public:
        Boite(int n = 10);            // construit une boîte contenant au maximum n pointeurs
        ~Boite();                     // destructeur
        virtual void Mets(Objet *po) = 0;      // met dans la boîte le pointeur po
        virtual Objet *Extrais() = 0;  // extrait un pointeur de la boîte
        int Vide();                   // indique si la boîte est vide
        int Pleine();                 // indique si la boîte est pleine
protected:
        int vide, pleine;             // indicateurs
        int taille;                   // capacité de la boîte
        Objet **T;                    // considéré comme tableau de pointeurs sur Objet
};

//---------------------- définition de la classe Boite ----------------------

Boite::Boite(int n)
{
        taille = n;
        T = new Objet* [taille];      // on peut prévoir ici un test de débordement mémoire
        vide = 1;
        pleine = 0;
};

Boite::~Boite()
{
        delete [] T;                  // libère l'espace pointé par T
};
```

```
int Boite::Vide()
{
        return vide;
};

int Boite::Pleine()
{
        return pleine;
};

//---------------------- déclaration de la classe Pile ----------------------

class Pile :  public Boite            // classe décrivant une pile
{
public:
        Pile(int n = 10);             // construit une pile contenant au maximum n pointeurs
        void Mets(Objet *po);         // met dans la pile le pointeur po
        Objet *Extrais();             // extrait un pointeur de la pile
protected:
        int nbelements;               // nombre effectif d'éléments contenus dans la pile
};

//---------------------- définition de la classe Pile ----------------------

Pile::Pile(int n) :  Boite(n)
{
        nbelements = 0;
};

void Pile::Mets(Objet *po)
{
        T[nbelements++] = po;
        vide = 0;
        pleine = nbelements == taille;
}

Objet *Pile::Extrais()
{
        Objet *temp;
        temp = T[--nbelements];
        pleine = 0;
        vide = nbelements == 0;
        return temp;
}

//---------------------- déclaration de la classe Queue ----------------------

class Queue :  public Boite           // classe décrivant une queue
{
public:
        Queue(int n = 10);            // construit une queue contenant au maximum n pointeurs
        void Mets(Objet *po);         // met dans la queue le pointeur po
        Objet *Extrais();             // extrait un pointeur de la queue
protected:
        int tete,                     // indice où se trouve l'élément le plus ancien
            queue;                    // indice où se mettra le prochain élément
                                      // (T est utilisé comme un tableau circulaire)
};

//---------------------- définition de la classe Queue ----------------------

Queue::Queue(int n) :  Boite(n)
{
        tete = queue = 0;
};

void Queue::Mets(Objet *po)
{
        T[queue++] = po;
        queue %= taille;              // retour au début du tableau si nécessaire
        vide = 0;
        pleine = tete == queue;
}

Objet *Queue::Extrais()
{
        Objet *temp;
```

```
        temp = T[tete++];
        tete %= taille;                // idem
        pleine = 0;
        vide = tete == queue;
        return temp;
}
```

Voici par exemple comment nous pourrions utiliser la classe `Queue` :

• création d'une file d'attente contenant au plus 1000 éléments :

```
Queue *q = new Queue (1000);
```

• mise d'un élément dans la file :

```
if (! q -> Pleine())
        q -> Mets(pObjet);             // pObjet pointe sur un Objet supposé créé par ailleurs
```

• extraction d'un élément de la file :

```
if (! q -> Vide())
        pObjet = q -> Extrais();
```

• destruction de la file :

```
delete q;
```