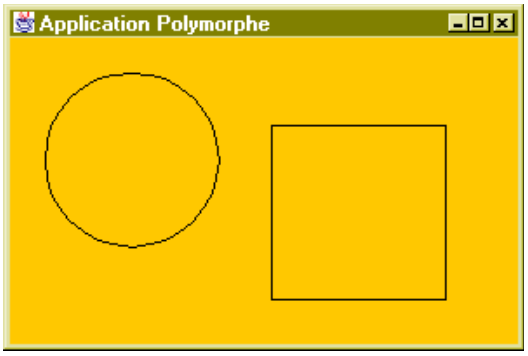
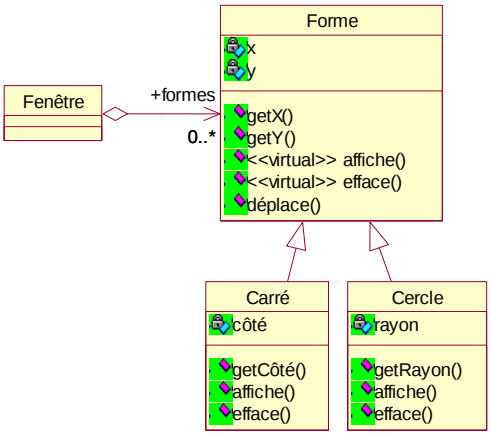
Imaginons que nous ayons besoin de mettre en œuvre un logiciel de dessins constitué d'un ensemble de formes, comme des cercles, des carrés, etc. Ces formes doivent pouvoir s'afficher sur la surface de travail que constitue la fenêtre. Par ailleurs, lorsque nous demandons de rafraîchir le contenu de la fenêtre, parce que, par exemple, cette dernière se situe dans la barre des tâches, il faut aussi que toutes les formes, quelles qu'elles soient, s'affichent automatiquement.



Dans ce contexte, nous imaginons bien une hiérarchie composée de toutes les classes représentant chacune des formes du logiciel de dessin, comme par exemple, la classe « *Cercle* », ainsi que la classe « *Carré* ». Par ailleurs, puisque nous en connaissons le principe, nous pouvons profiter du polymorphisme, en créant une classe de base appelée « *Forme* » qui est l'ancêtre de toutes les autres classes de dessin. Ainsi, grâce à ce polymorphisme, il sera aisé de demander le bon affichage sur chacune des classes représentant la hiérarchie.

Pour que ce mécanisme puisse se mettre en œuvre, Il faut donc proposer une méthode virtuelle « *affiche* » sur la classe de base. Ainsi, au moment du rafraîchissement de l'ensemble du dessin, c'est la méthode « *affiche* » correspondant à la bonne classe qui sera sollicitée.



Toutefois, en regardant de plus près, nous n'aurons jamais d'objets relatifs à la classe « *Forme* ». En effet, seuls les objets, représentant une forme bien précise, seront créés pour l'application, comme par exemple, les objet de type « *Cercle* ». Déclarer un objet de la classe « *Forme* » n'a pas de sens. Il s'agit d'une abstraction. Cette classe n'existe que pour les classes dérivées, qui elles, correspondent à quelque chose de concret. Il est d'ailleurs de notre devoir d'empêcher que la classe « *Forme* » puisse fournir des objets. Une telle classe est alors appelée classe abstraite. A contrario, les classes comme « *Cercle* » ou « *Carré* » sont appelées des classes concrètes. Nous ne pouvons déclarer des objets que sur des classes concrètes.

Je viens de dire que déclarer un objet de la classe « *Forme* » n'a pas de sens. Effectivement, nous serions totalement incapable de proposer un affichage précis sur cette classe. Nous sommes obligés de connaître le type de forme pour savoir comment l'afficher. Par exemple, je sais comment faire un cercle, alors que pour une forme, je ne sais pas. Une forme est simplement générique, c'est juste un concept, ou si vous voulez une abstraction.

D'ailleurs, que pourrions-nous mettre dans la méthode « *affiche* » de la classe « *Forme* ». Comme le montre l'exemple ci-contre, le contenu de cette méthode est totalement vide.

```
2  class Forme
3  {
4    int x, y;
5  public:
6    Forme(int x, int y)           { this->x = x; this->y = y; }
7    int getX() const              { return x; }
8    int getY() const              { return y; }
9    virtual void affiche()        { }
10   virtual void efface()         { }
11   void deplace(int dx, int dy)  { x+=dx; y+=dy; }
12 };
```

Ces méthodes ne font rien

Mise en œuvre d'une classe abstraite

Contrairement au langage Java, il n'est pas possible d'indiquer directement qu'une classe est abstraite. Ceci dit, généralement une classe est abstraite parce que certaines méthodes sont elles-mêmes abstraites. Une méthode est abstraite lorsque aucune action n'est proposée. C'est effectivement ce que nous avons sur les méthodes « *affiche* » et « *efface* ». En réalité, pour que ces méthodes soient réellement désignées comme abstraites et bien indiquer qu'elles ne font vraiment rien, elles doivent être explicitement initialisées à *0*. Donc à la suite de la signature de la méthode, vous devez placer « ***= 0 ;*** ». Ces méthodes sont appelées, dans le langage C++, des *méthodes virtuelles pures*.

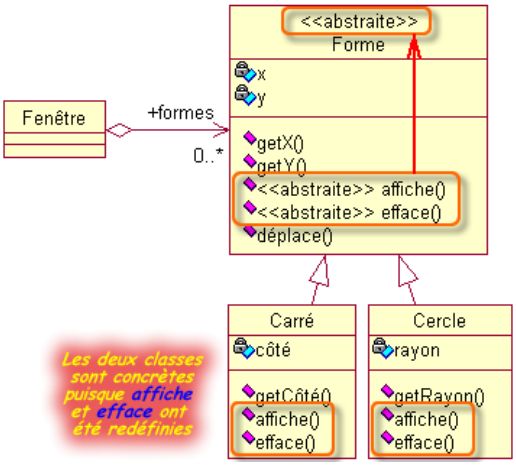
```
2  class Forme
3  {
4    int x, y;
5  public:
6    Forme(int x, int y)           { this->x = x; this->y = y; }
7    int getX() const              { return x; }
8    int getY() const              { return y; }
9    virtual void affiche() = 0;
10   virtual void efface() = 0;
11   void deplace(int dx, int dy)  { x+=dx; y+=dy; }
12 };
```

Méthodes virtuelles pures (méthodes abstraites)

Lorsque vous héritez d'une telle classe, vous devez impérativement, si vous désirez que votre classe devienne concrète, redéfinir toutes les méthodes virtuelles pures. Si cette condition n'est pas réalisée, la classe dérivée est elle-même une classe abstraite. Cela signifie que vous ne pourrez pas non plus créer un objet depuis cette classe dérivée. C'est logique puisque certaines méthodes ne seraient pas décrites, et du coup, l'objet ne saurait pas comment se comporter précisément.

Conclusion

Finalement, dans le langage C++, une classe est abstraite lorsqu'elle dispose d'au moins une méthode abstraite, c'est-à-dire une méthode virtuelle pure. Une fois, qu'une classe est définie abstraite, il n'est plus possible de créer un objet relatif à cette classe. Une erreur de compilation se produit si vous tentez l'expérience.



Utilité des classes abstraites

Les classes abstraites font parties de la théorie du polymorphisme. Une classe abstraite est une classe utilisée pour factoriser des propriétés communes à plusieurs classes, mais qui est trop générique pour instancier des objets. Une classe abstraite n'a donc en principe aucun objet, car elle n'a pas assez de propriétés pour que l'on puisse la caractériser avec suffisamment de précision.

Une classe abstraite n'existe que pour créer d'autres classes qui héritent d'un comportement minimum et qui assurent la pérennité du fonctionnement attendu. Ces nouvelles classes dérivées se spécialisent en proposant de nouvelles propriétés, mais doivent avant tout respecter le contrat établi par la classe abstraite, c'est-à-dire qu'elle doivent impérativement, proposer chacune leur propre définition de toutes les méthodes abstraites. Du coup, par ce système, nous assurons un fonctionnement sûr et stable, puisque chaque objet pour exister doit être correctement constitué.