CONSTRUCTEURS ET DESTRUCTEURS

1 Objectifs du TD

1) Comprendre les notions de constructeurs et de destructeurs.
2) Bien différencier les constructeurs par défaut, avec paramètre et de copie.
3) Prendre conscience de l'importance de contrôler le nombre d'appels aux constructeurs.
4) Utilisation des listes d'initialisation.
5) Utilisation implicite/explicite des constructeurs.

EXERCICE 1 –

1) Déclarez dans un fichier `Point.h` une classe Point avec deux champs privés `x` et `y` de type `double` et deux méthodes publiques `get_x` et `get_y` renvoyant les valeurs de `x` et `y`.
2) Définissez les méthodes dans un fichier `Point.cpp`.
3) Ajoutez à la classe un constructeur ayant en paramètre l'abscisse et l'ordonnée du point à créer, et initialisant l'ordonnée à 0 par défaut. Définissez ce constructeur (dans le `.cpp`). Testez ensuite successivement si les déclarations suivantes sont valides :

   ```
   Point p(3.1,4);
   Point p(2);
   Point p;
   ```

   Pour celle(s) qui ne le serai(en)t pas, réessayez en passant cette fois le constructeur à paramètres en commentaire, puis en déclarant et définissant un constructeur par défaut dans la classe. Que faut-il en conclure ?
4) Déclarez et définissez un constructeur de copie et un destructeur par défaut. Dans la suite la classe `Point` contiendra toujours ces 3 types de constructeurs et un destructeur.

EXERCICE 2 – Nous allons maintenant ajouter des compteurs permettant d'étudier le nombre d'appels aux différents constructeurs et au destructeur.

1) Ajoutez à la classe `Point` des variables statiques privées `nb_default`, `nb_param`, `nb_copy` et `nd_destroy` de type `int`, et des méthodes (publiques) permettant d'afficher ces variables à l'écran.
2) Initialisez ces variables et définissez les méthodes dans `Point.cpp`.
3) Modifiez les constructeurs et le destructeur de manière à ce que les variables `nb_default`, `nb_param`, `nb_copy` et `nd_destroy` contiennent respectivement le nombre d'appels effectués au constructeur par défaut, au constructeur avec paramètre, au constructeur de copie et au destructeur.
4) Vérifiez que votre code marche bien en affichant successivement le nombre d'appels aux différents constructeurs/destructeur effectués lors des exemples de 1.c.
5) Testez votre constructeur par defaut en exécutant `Point p(2,3); Point q=p;`. Passez votre constructeur par copie en commentaire et réessayez. Que doit-on en conclure ?

EXERCICE 3 –

1) Testez l'exécution des 2 lignes suivantes en affichant les différents nombres d'appels aux constructeurs/destructeur effectués :

   ```
   Point p(2,1);
   p=Point(3,2);
   ```

   Comprenez-vous à quoi correspondent les différents appels obtenus ? Cette manière de réaffecter un nouvel objet de la même classe à une variable n'est pas efficace, elle ne sera jamais utilisée. Comment faut-il donc faire ?
2) Déclarez et définissez une nouvelle méthode `reaffect` répondant à la question précédente. Testez-la ensuite en affichant les nombres d'appels.

EXERCICE 4 –

1) Ecrivez une classe Rectangle ayant comme champs deux points `point_1` et `point_2` et comme méthodes `get_p1`, `get_p2` et `reaffect`.
2) Ajoutez un constructeur par d efaut initialisant les deux points à `(0,0)` et `(1,1)`, un constructeur avec un paramètre initialisant le deuxième point à `(0,0)`, un constructeur avec deux paramètres, un constructeur de copie et le destructeur.

   On utilisera un passage de paramètre constant par réference, c'est à dire des paramètres de type `const Point &`.
3) Afficher les nombres d'appels aux constructeurs/destructeur de `Point` lors de l'exécution de :

```
Point A(0,1);
Rectangle r(A);
```

   A quoi correspondent les appels observés ?

EXERCICE 5 –

1) Utiliser maintenant la syntaxe suivante, dite à liste d'initialisation pour définir le constructeur à un paramètre :

```
Rectangle::Rectangle(const Point &p):point_1(p),point_2(Point(0,0)){};
```

   Affichez le nombre d'appels aux constructeurs effectués lors de l'exécution du code de la question (c). Que se passe-t-il cette fois ci ?
2) Adaptez le constructeur par défaut de la mème manière.

EXERCICE 6 –

1) Testez l'exécution `Rectangle r ; r.reaffect(2,2) ;`. Ajoutez l'instruction `explicit` au début de la déclaration du constructeur à paramètres dans `Point.h` et réessayez. A quoi sert cette instruction ?
2) Ajoutez un constructeur `Rectangle` prenant quatre `double` en paramètres.