DS GM4

Informatique : C++

Kotowicz Jean-Philippe
(jean-philippe.kotowicz@insa-rouen.fr)

Documents autorisés : tout ce qui se trouve dans le cours moodle C++ Durée : 3H

Cahier de travaux d'un architecte

Un architecte souhaite disposer d'un outil qui lui permettra de gérer tous les projets sur lesquels il travaille. Selon le type de bâtiment et le site où le bâtiment est construit, les contraintes vont être différentes.

Consignes :

Avant de commencer l'examen :

- Récupérer dans moodle l'archive : *evalC++.zip*
- Après l'avoir décompressée, nommez votre dossier *evalC++-NomPrenom*.

A la fin de l'examen :

- Cinq minutes avant la fin de l'examen :
  - Vérifier que votre répertoire contient tous vos dossiers et fichiers ;
  - Assurez-vous que votre projet compile ;
  - Créez une archive de votre répertoire nommée *evalC++-NomPrenom.zip* ;
  - Déposez sur moodle votre archive.
  - Confirmez votre dépôt.

Travail à faire :

1. CONSEILS :
   a. Tout au long du projet, vérifiez que le programme compile.
   b. Compléter le *main* (cf. 3ème partie) pour tester, au fur et à mesure, vos méthodes en ajoutant toutes les instances que vous jugerez utiles.
   c. Un fichier *makefile* (présent à la racine du dossier) permet de tout compiler séparément et de réaliser l'édition des liens pour créer l'exécutable quel que soit le nom de vos fichiers pourvu qu'ils soient dans les bons répertoires. L'exécutable créé s'appelle alors ***testEval*** et se trouve dans le répertoire *bin*.
2. Vous devez impérativement :
   a. Séparer le code dans différents fichiers hpp (à mettre dans le dossier include) et cpp (à mettre dans le dossier src) en évitant les inclusions multiples et les inclusions croisées.
   b. Protéger les paramètres en entrée des méthodes de toute modification.
   c. Empêcher la modification des attributs par les méthodes qui n'ont pas lieu de la faire.
   d. Faire en sorte que les classes filles puissent accéder aux attributs de la classe mère sans passer par les accesseurs.
   e. Implanter, si nécessaire, destructeurs et constructeurs par recopie.

Première partie : les bâtiments

I.1. Un Batiment est décrit par 2 attributs : une chaîne de caractères nom, un entier surface. Le nom est initialisé à la construction de l'objet par un argument transmis au constructeur, la surface est initialisée par défaut à 0. En cas d'erreur ou de changement, on peut modifier ces attributs par les méthodes modifNom(...) et modifSurface(...). *Donnez la définition de la classe* ***Batiment*** *et la définition de ses méthodes.*

I.2. On voudrait avoir une méthode nbBatiment() retournant le nombre de bâtiments qui ont été créés. *Ajoutez ce qu'il faut à votre code pour qu'une telle méthode existe.*

I.3. On désire maintenant référencer chaque Batiment par un numéro unique, fixé à la création de l'instance et que l'on peut récupérer par un appel à une méthode intitulée getNumero(). *Ajoutez les attributs et méthodes nécessaire pour réaliser une telle opération.*

I.4. Il s'agit maintenant de distinguer les maisons et les immeubles. Pour cela, nous créons deux classes, Maison et Immeuble, héritant de Batiment. *Effectuez les modifications de la classe* ***Batiment*** *si nécessaire et donnez la définition des classes* ***Maison*** *et* ***Immeuble****.*

I.5. Une maison aura au plus 1 étage, alors qu'un Immeuble en aura au moins 2. *Ajoutez un attribut* nbEtage *de type* ***vecteur d'entiers*** *(chaque entier étant la surface de l'étage en question) ainsi que les méthodes* nbEtages() *et* ajouteEtage() *(pensez à contrôler le bon nombre d'étages en fonction du type de bâtiment et à mettre à jour la surface).*

I.6. On souhaite que l'appel à une méthode getTypeBatiment(), commune à ces trois classes, retourne la string "BATIMENT", "MAISON" et "IMMEUBLE" pour respectivement une instance de type bâtiment, maison et immeuble. *Définir cette méthode pour réaliser cette opération.*

I.7. On souhaite que les instructions suivantes affichent : « IMMEUBLE » :

```
Batiment *ptr;
Immeuble tour1("Perret");
ptr=&tour1;
cout<<ptr->getTypeBatiment()<<endl;
```

Vérifiez que votre code n'affiche pas « BATIMENT ». Si c'est le cas, faites les modifications à apporter à votre code.

I.8. On souhaite que l'instruction : cout<<unBatiment; affiche le texte : « BATIMENT n°**N** : **NOM**, **SURFACE** m2, **nbEtages** étage(s) », où **N** est le numéro de unBatiment, **NOM** son nom, **SURFACE** la surface totale du bâtiment. La chaine BATIMENT sera remplacée par MAISON s'il s'agit d'une Maison et par IMMEUBLE s'il s'agit d'un Immeuble. **nbEtages** ne sera affiché que s'il est non nul (tenir compte du pluriel). *Faites la surcharge de l'opérateur << et modifiez la classe* ***Bâtiment*** *en conséquence si nécessaire.*

I.9. On décide finalement de rendre la classe Batiment abstraite. *Modifiez la classe* ***Batiment*** *en conséquence.*

I.10. Notre architecte souhaite également se lancer dans la construction d'hôtels particuliers qui sont à la fois une Maison et un Immeuble. *Créer la classe HotelParticulier qui va hériter de* ***Maison*** *et de* ***Immeuble****. Implanter les méthodes nécessaires.*

I.11. Un *HotelParticulier* disposera un attribut supplémentaire standing qui sera fixé à la création. Le standing d'un hôtel particulier vaudra 3 par défaut à la construction, mais pourra être diminué ou augmenté par des méthodes adéquates. *Ajouter cet attribut, les constructeurs nécessaires et les accesseurs adéquats.*

Deuxième partie : les sites de construction

On cherche maintenant à gérer les contraintes selon les sites.

II.1. Un site est décrit par un objet de la classe Site qui contiendra pour le moment un unique attribut booléen précisant si le site est **classé** ou non. Cet attribut est initialisé lors de la construction par un argument transmis au constructeur et ne peut être modifié ensuite. *Ecrire la déclaration de la classe **Site** et la définition de ses méthodes.*

II.2. Un site peut contenir un certain nombre de bâtiments. Pour cela, on ajoute à la classe Site une list d'objets de la classe Batiment qui contient tous les bâtiments (indifféremment maisons, immeubles ou hôtels particuliers) se trouvant sur le site. L'initialisation et la modification de cette liste se feront par les méthodes ajouterBatiment(…) et retirerBatiment(…). La méthode ajouterBatiment(…) prend en argument l'adresse d'un Batiment. La méthode retirerBatiment(…) prend en argument un entier référençant le numéro unique d'un Batiment et retourne un booléen valant true si la suppression a été effectuée avec succès, false sinon. *Faire tous les changements à apporter à la classe **Site** pour réaliser ces opérations. Attention, rappelez-vous que **Batiment** est une classe abstraite !!!*

II.3. *Surcharger l'opérateur << affichant tous les bâtiments d'un **Site**.*

II.4. On souhaite récupérer les bâtiments de chaque site comme suit : si s est un site, alors s[i], où i est un entier, retourne un pointeur sur le $i^{ème}$ bâtiment du site, NULL si i n'est pas un entier valide. *Donnez le code permettant de réaliser cette opération.*

II.5. On désire ajouter à la classe Batiment un attribut pointant sur son Site. *Quelles modifications faut-il apporter à la définition de la classe **Batiment** pour éviter la référence circulaire qui s'en suit ?*

II.6. Un bâtiment possède une méthode (abstraite) prevoirGrue() qui précise, selon certaines conditions, le nombre de grues nécessaires pour la construction. Pour cela, on utilisera l'attribut nbEtages. Si le bâtiment fait moins de NBMax étages (fixé à 2), la méthode prevoirGrue() va retourner 0, sinon si le Site est classé, il faut prévoir une seule grue, s'il n'est pas classé, il faudra 1 grue par 100m^2. *Modifiez les classes en conséquence.*

II.7. Disposant dans la classe Batiment d'une fonction nommée contraintesOK() permettant de vérifier qu'un bâtiment respecte bien les contraintes du site sur lequel il se trouve. On souhaite disposer dans Site d'une méthode siteAuxNormes() qui permet de tester si des bâtiments ne respectent pas les contraintes du site. La fonction retournera un booléen true si tous les bâtiments respectent les contraintes du site, false sinon. *Écrivez le code permettant de réaliser cette opération. La méthode contraintesOk() de la classe bâtiment pourra retourner aléatoirement vrai ou faux.*

Troisième partie : Programme principal

Pour tester vos classes, implanter au fur et à mesure :

III.1. *Créer 2 sites.*

III.2. *Pour chacun des 2 sites, un menu demande à l'utilisateur les bâtiments à ajouter.*

III.3. *Afficher l'ensemble des bâtiments de chaque site.*

III.4. *Déplacer le plus petit bâtiment du 1^{er} au 2^{nd} site.*

III.5. *Afficher le nombre de grues nécessaires pour chaque bâtiment du 1er site et vérifier que le 2ème site est aux normes.*