TP1 - Classes

Travail à rendre

Les 3 fichiers modifiés, **avec les réponses aux questions en commentaire à la fin du fichier main.cpp.**

La classe Permis

Créez un nouveau projet incluant les 3 fichiers : permis.h, permis.cpp et main.cpp

1. a. Que contient le fichier `permis.h` ?
   b. Que contient le fichier `permis.cpp` ? À quoi sert la présence de `Permis::` ? Retirez `Permis::` de la définition de la fonction membre `affiche`. Que se passe-t-il ?
   c. Que contient le fichier `main.cpp` ?
   d. Compilez et exécutez le projet. Quels affichages sont produits ? Expliquez.

2. a. Complétez le fichier `main.cpp` en déclarant deux autres instance `p1` et `p2` (= deux variables) de la classe (= de type) `Permis`, en choisissant des noms, des prénoms, des dates, des numéros de permis, et des nombres de points. Vérifiez en appelant la fonction membre `affiche`.
   b. Testez l'instruction `cout << p1 << endl` dans `main.cpp`. Que se passe-t-il ? Pourquoi ?
   c. Testez l'instruction `cout << p1._nom << endl` dans `main.cpp`. Que se passe-t-il ? Pourquoi ?
   d. Pourquoi le prototype de la fonction `affiche` se termine-t-il par le mot-clé `const` et pas celui de la fonction `retraitPoints` ?

3. a. Ajoutez une méthode `getNom` qui retourne le nom du détenteur de l'objet cible. Testez.
   b. Ajoutez une méthode `incrementePoints` qui augmente le nombre de points du permis de 1. Si le permis a déjà MAXP points, la méthode lance une exception de type `invalid_argument` précisant la raison de l'exception. Testez.
   c. Ajoutez une méthode `decrementePoints` qui diminue le nombre de points du permis de 1. Si le permis a déjà MINP points, la méthode lance une exception `invalid_argument` précisant la raison de l'exception. Testez.

4. a. En utilisant une des méthodes définies précédemment, ajoutez dans les fichiers `permis.h` et `permis.cpp` une fonction **externe** (à la classe `Permis`) `ajoutePoints` de prototype :
   ```
   void ajoutePoints(Permis&, int);
   ```
   avec deux arguments : un permis et un entier positif. Cette fonction ajoute les points passés en deuxième argument au permis passé en premier argument, en ne dépassant jamais MAXP (si trop de points sont ajoutés, on reste sur MAXP points). Testez.
   b. Que se passe-t-il si on modifie le prototype de la fonction externe `ajoutePoints` :
   ```
   void ajoutePoints(const Permis&, int);
   ```

5. a. Modifiez les fichiers `permis.h` et `permis.cpp` pour ajouter à la classe `Permis` le destructeur suivant :
   ```
   Permis::~Permis() {
     cout << "Destruction du Permis" << endl;
   }
   ```
   b. Ajoutez l'affichage des données membres dans la définition du destructeur. Compilez. Expliquez les affichages à l'exécution : quand le destructeur est-il appelé ? Dans quel ordre les objets sont-il détruits ?

6. Un constructeur doit s'assurer que l'objet est bien formé. Modifiez le constructeur non vide pour déclencher une exception type `invalid_argument` si le nombre de points donné est aberrant. Traitez l'exception dans `main.cpp`.
   Grâce à l'encapsulation, les permis auront dorénavant toujours un nombre de points valide : les constructeurs ne créent que des permis valides, les méthodes publiques préservent cette propriété, et l'utilisateur n'a pas accès à l'attribut `_nbpoints` et ne peut donc pas y mettre une valeur aberrante.

7. Testez l'instruction suivante dans le programme principal :
   ```
   if (p1 == p2) cout << "c'est pareil" << endl;
   ```
   Que se passe-t-il ? Pourquoi ?

8. Ajoutez une fonction membre (= une méthode) `compare` pour comparer deux permis : la fonction renvoie `true` si les deux permis ont l ? ne nombre de points. Testez.

9. Déclarez un vecteur de Permis de taille 10. Observez la construction et la destruction des instances créées. Affichez le contenu du vecteur.
10. (s'il vous reste du temps) Modifiez la déclaration du constructeur avec des paramètres pour ajouter des valeurs par défaut à tous ses paramètres.
    1. Déclarer un Permis avec un argument de type string. Que se passe-t-il ? pourquoi ?
    2. Déclarer un Permis avec aucun argument. Que se passe-t-il ? Pourquoi ? Quelle solution peut-on adopter pour ne plus avoir d'erreur ?
11. (s'il vous reste du temps) Ajouter les instructions suivante s:

```
Permis pa("Toto");
if (pa.compare(string("Toto"))) cout << "on ne peut pas comparer un permis à un string" << endl;
else cout << "on peut comparer un permis à un string" << endl;
```

```
if (pa.compare(string("Titi"))) cout << "on ne peut pas comparer un permis à un string" << endl;
else cout << "on peut comparer un permis à un string" << endl;
```

    Que se passe-t-il ?

12. (optionnel) Ajouter une fonction membre pour saisir un permis. Penser aux vérification.
13. (optionnel) Ajouter une fonction membre pour modifier le nom du détenteur d'un permis.
14. (optionnel) Ajouter une fonction qui prend un vecteur de Permis en paramètres et affiche les noms des détenteurs des permis.

Modifié le: vendredi 24 novembre 2023, 15:54