Projet de programmation orientée objet en `c++`
Partie 1

INSA CVL, MRI (4A)

5 octobre 2021

Le projet sera à rendre sous la forme d'une archive contenant le code source non compilé sur Célène. Elle devra comprendre un Makefile (les commandes `make` et `make clean` doivent être fonctionnelles). La propreté du code (indentation, choix judicieux des noms de variables, commentaires, documentation et respect des conventions) sera prise en compte pour l'évaluation.

Le but de ce projet est d'implémenter des classes qui permettront de gérer les différentes tâches d'une entreprise selon leur priorité. Chaque tâche sera définie par un nom, une description, et une priorité. On pourra avoir par exemple :

Tâche 1
- nom : *"Panne"*
- description : *"Réparer la machine No856"*
- priorité : 8

Tâche 2
- nom : *"Achat"*
- description : *"Acheter des pièces de rechange"*
- priorité : 4

Dans ce cas, la tâche 1 est prioritaire sur la tâche 2 et devra donc être traitée avant. On implémentera deux classes :

- La classe `Tache` qui définit une tâche selon son nom, sa description et sa priorité. Cette classe implémente un constructeur par défaut et un constructeur qui prend en paramètres un nom, une description et une priorité, et implémente un accesseur pour chacun de ses paramètres. Elle contient aussi une fonction `traiter()` qui permet de traiter la tâche, et qui renvoie 1 si le traitement s'est effectué correctement, et 0 en cas d'erreur. Il serait compliqué d'implémenter une fonction qui permette de traîter toutes les tâches possibles. Pour cette raison, nous allons simplifier le problème en supposant qu'une tâche est traitée correctement par la fonction `traiter()` avec une probabilité de 4/5.
- La classe `ListeDeTaches` qui définit une liste de tâches. Cette classe a pour paramètres un tableau d'objets de type `Tache` nommé `liste`, un entier `max` qui définit la taille de ce tableau, un entier `nb` qui définit le nombre de tâches contenues dans le tableau `liste`, et un entier `prochain` qui donne l'indice de la tâche qui a la plus grande priorité dans le tableau `liste` (et qui devra, par conséquent, être la prochaine tâche à traiter). Concrètement, la liste contiendra les tâches qui seront stockées entre l'indice 0 et l'indice $\texttt{nb} - 1$ du tableau `liste`. On aura donc toujours $\texttt{nb} \leq \texttt{max}$. Cette classe dispose d'un constructeur qui prend en paramètre un entier `max` et qui initialise une liste de tâches vide pouvant contenir au plus `max` tâches, des accesseurs, une méthode pour ajouter des tâches à la liste, et une méthode pour traiter la tâche la plus urgente de la liste.

Exercices

Exercice 1. Donnez une représentation UML des classes et de leurs relations.

Exercice 2. Implémentez la classe `Tache`. Implémentez une fonction statique `afficher_tache(Tache)` qui affiche une tâche sur la console. Testez que votre classe fonctionne bien dans le main.

Exercice 3. Implémentez la classe `ListeDeTaches`. Cette classe doit implémenter une fonction qui permet d'ajouter une tâche à la liste (`AjouterTache(Tache)`), et une fonction qui permet de traîter la tâche avec la plus haute priorité de la liste et qui renvoie un booléen indiquant si le le traîtement s'est effectué correctement ou non (`bool TraiterTache()`). Bien entendu, il faudra prendre soin que ces méthodes aient un comportement cohérent avec les attributs de la classe `ListeDeTaches`. Implémentez une fonction statique `afficher_listeDeTaches(ListeDeTaches)` qui affiche une liste de tâches sur la console. Testez que votre classe fonctionne bien dans le main.

Exercice 4. Dans le main, créez un objet de la classe `ListeDeTaches`, et ajoutez-y une dizaine de tâches. Attention, ces tâches ne doivent pas être ajoutées dans l'ordre de leur priorité. Donnez un code qui permet de traîter chacune des tâches de cet objet dans l'ordre de leur priorité. À chaque fois, on affichera le nom de la tâche à traiter, puis on affichera un message indiquant si la tâche a été traitée correctement ou pas.