Contrôle continu 2

*Durée : **2h00**. Documents autorisés. Dans la correction, il sera tenu compte de la clarté du code et de l'utilisation judicieuse de techniques ou conventions telles que les passages par référence, utilisation de `const`, séparation entre les méthodes faisant des entrées-sorties et celles n'en faisant pas, bonne gestion de la mémoire et de façon plus générale, d'une modélisation conforme à la programmation orientée objet en C++14. Il sera aussi tenu compte de la présence d'un `main` permettant de tester vos classes. La présence d'erreurs de compilation/édition de liens sera sanctionnée. À la fin de la séance, faites une archive (tar.gz ou zip) ne contenant que les sources (pas de fichiers objets ni exécutable) dont le nom sera votre **nom de famille** et envoyez l'archive à david.genest@univ-angers.fr .*

1. Définir un type `jourdelasemaine` représentant un jour de la semaine sous la forme d'un entier (0 : lundi... 6 : dimanche). Définir une fonction `estavant` permettant de comparer deux `jourdelasemaine` a et b, retournant vrai si a est avant b dans la semaine, et une fonction `tostring` qui, à partir d'un `jourdelasemaine` retourne l'intitulé de ce jour sous la forme d'une chaîne.

2. Définir une classe `horaire` composée de deux attributs entiers appelés h et m représentant un horaire. Par exemple 11:20 sera représenté par une valeur de h valant 11 et une valeur de m valant 20. Munir cette classe d'un constructeur à deux arguments, accesseurs, méthode `tostring` retournant une chaîne de la forme « 11:20 ».

3. Écrire une classe `creneau` (correspondant à un créneau d'enseignement dans un emploi du temps). Un créneau est composé d'un jour de la semaine et d'un horaire. Munir cette classe d'un constructeur, d'accesseurs, pas de mutateurs et de :
   opérateur d'affectation ;
   opérateurs de comparaison == et != ;
   opérateur de comparaison < permettant de savoir si un `creneau` est situé avant un autre dans un emploi du temps d'une semaine ;
   opérateur de sortie sur flux.

4. On désire maintenant représenter des enseignements. Chaque enseignement a un nom (chaîne) et un créneau. Parmi les enseignements, on distingue les cours, chaque cours étant effectué par un enseignant dont on mémorisera le nom, et les TP (on ne mémorisera pas le nom de l'enseignant). Le nombre maximum d'étudiants pouvant suivre un TP est 20 alors qu'il est choisi par l'enseignant dans le cas d'un cours. Un cours dure systématiquement 80 minutes alors que la durée des TP est variable, mais ne peut pas être inférieure à 60 minutes.
   Écrire une (des) classe(s) pour représenter ces enseignements, vous n'écrirez que les accesseurs nécessaires, mais au moins ceux permettent d'accéder à la durée de l'enseignement et au nombre d'étudiants maximum.

5. Écrire une méthode `tostring` retournant une chaîne du type `Cours NomEnseignement jour, horaire,durée nbmax NomEnseignant` ou `TP NomEnseignement jour,horaire,durée nbmax`.

6. Écrire une méthode `conflit` permettant de tester s'il y a un conflit entre deux enseignements, c'est-à-dire deux enseignements qui ont lieu au même moment. Par exemple, appelée sur un `enseignement` du mercredi 10:00 de durée 60 et un `enseignement` du mercredi 10:30 de durée 80, la méthode retournera `true` car il y a effectivement un conflit entre ces deux enseignements.

7. Écrire une classe `EDT` permettant de regrouper tous les enseignements d'un **e**mploi **d**u **t**emps, à l'aide d'une liste. Faites particulièrement attention à écrire un code robuste dans cette question et les suivantes, notamment pour ce qui concerne la gestion de la mémoire.

8. Écrire une méthode `supprimerens` qui prend comme paramètre un nom d'enseignements et qui supprime tous les enseignements de ce nom d'un `EDT`.

9. Écrire une méthode `ajouterens` permettant d'ajouter un enseignement à un `EDT`. Cette méthode vérifiera que l'enseignement à ajouter n'est pas en conflit avec un enseignement déjà présent dans l'`EDT`. Si tel est le cas, elle lèvera une exception de type `exceptionenseignement`, sous-classe de `std::exception`, que vous définirez.

10. Écrire une méthode `coursde` prenant comme paramètre un nom d'enseignant et retournant les enseignements effectués par cet enseignant.

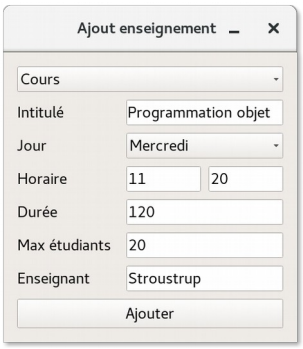
11. Écrire une méthode `chercher` qui prend comme paramètre un jour de la semaine et un horaire et qui retourne l'enseignement qui a lieu à ce moment-là. Attention, ce n'est pas forcément l'heure de début de l'enseignement, qui peut avoir commencé avant l'heure passée.

12. Permettre de faire des copies d'un `EDT`. La copie obtenue devra être indépendante de l'objet original. Si de nouveaux types d'enseignements (tels que les TD) sont créés, elle devra fonctionner, sans requérir de modifications.

13. Écrire une méthode `saisie` demandant à l'utilisateur s'il veut saisir un cours ou un TP, demandant ensuite les informations sur l'enseignement, et ajoutant à l'`EDT` l'enseignement saisi par l'utilisateur. Si une exception est levée lors de l'exécution de `ajouterens`, un message d'erreur sera affiché par la méthode `saisie` et l'utilisateur sera invité à saisir un autre horaire (et uniquement l'horaire).

14. Rajouter la méthode `correct` retournant `true` si un `EDT` si tout cours d'un nom donné est accompagné d'au moins un TP du même nom, `false` sinon.

15. Écrire une classe avec Qt permettant d'ajouter un enseignement à un EDT déjà existant. La fenêtre devra être comme sur la capture d'écran ci-contre : une liste déroulante permet de choisir Cours ou TP, un champ de texte permet de saisir l'intitulé, une liste déroulante permet de choisir le jour, trois champs de texte permettent de choisir l'horaire de début et la durée, puis le nombre d'étudiants et l'enseignant. Le bouton *Ajouter* appellera la méthode `ajouterens`[1], et en cas d'exception affichera une boîte de dialogue d'erreur.



16. Modifier l'interface graphique afin que, lorsque l'utilisateur choisit Cours dans la liste déroulante, la durée soit fixée à 80 dans l'interface graphique, et quand il choisit TP, le nombre d'étudiants est fixé à 20 dans l'interface, et le nom de l'enseignant est vidé

1 Certaines informations saisies par l'utilisateur seront ignorées : par exemple dans le cas d'un cours, la durée saisie sera ignorée puisque la durée d'un cours est systématiquement 80 minutes.