Université Paris **Langages à objets avancé**
M1 **Année 2020-2021**

Classes

Ce TP est trop long pour une seule séance, concentrez vous sur la première partie.

1 Horloges et calendriers

Horloges

1. Déclarez simplement une classe `Horloge` qui encapsule des valeurs pour des heures, des minutes et des secondes.
2. Pour vous simplifier l'affichage, on vous rappelle ici comment obtenir des commandes agréables à utiliser.

   Vous devriez être familier avec la redéfinition de `toString()` en java où on généralisait la syntaxe d'affichage : `System.out.print("qq␣chose")` à `System.out.print(unObjet)` en réécrivant dans la classe de `unObjet` la méthode `toString()`. C'est elle qui est appelée implicitement pour convertir l'objet vers une chaîne de caractères.

   En c++ un affichage est produit par une syntaxe légèrement différente : `cout << "qq␣chose"`. Il ne s'agit pas d' un appel de méthode avec un argument. On utilise `<<` avec un terme à sa gauche et un terme à sa droite : `<<` est un **opérateur**. C'est lui qu'on redéfinit pour se permettre d'écrire `cout << unObjet`. On procède ainsi :
   — dans le `.cpp` on implémente :

   ```
   // Surcharge de l'opérateur <<
   ostream& operator<<(ostream& out, A &x) {
    out << "voilà␣comment␣afficher␣un␣A" << x.attribut << endl;
    return out;
   }
   ```

   — alors que dans le `.hpp`, en dehors du bloc de la classe, on déclare la signature correspondante à la surcharge :
   `ostream& operator<<(ostream& out, A &x);`

   Redéfinissez à présent l'opérateur d'affichage pour votre classe `Horloge`
3. Ecrivez une méthode `tick()` qui fait passer une seconde. Pensez qu'après 23H 59M 59S, on passe à 0H 0M 0S. Comme on souhaite être alerté d'un changement de jour, faite en sorte que `tick()` retourne une valeur appropriée. D'ailleurs, avez vous bien fait en sorte qu'à la construction les intervalles de définition soient corrects ?
4. Testez cette classe en écrivant un programme qui lit les heures, les minutes et les secondes depuis l'entrée standard (utiliser `cin`), l'affiche, l'incrémente et affiche le résultat en affichant si un changement de jour a eu lieu.

Dates

Écrivez une classe `Date` (on pourra supposer que les mois ont tous 30 jours), son constructeur, une méthode d'affichage et une méthode pour passer au jour suivant.

Testez cette classe le la même façon que la précédente.

Calendriers

En utilisant les deux premières classes :

1. Déclarez une classe `Calendrier` qui représente la date et l'heure.
2. Ecrivez des accesseurs qui "safe", c'est à dire qu'ils ne doivent pas révéler un composant privé qui pourrait être modifié de façon non contrôlée.

   Remarquez que les objets sont passés en arguments d'une façon qui peut vous sembler déroutante et vous posera des problèmes. Voici quelques éléments de réponse. Pensez d'abord qu'en `c++` il y a 3 façons de passer des arguments : par référence, par adresse ou par copie, et jusqu'ici en `java` vous aviez été habitué à 2 façons (par adresse ou par copie). Ce sont ces nuances qui seront causes d'erreurs. Pour résumer :
   - un objet passé en argument, ou en résultat d'une méthode est en fait copié (par l'invocation d'un constructeur spécial)
   - un objet référencé (repéré par le signe &) doit avoir dans tous les contexte où il est utilisé au moins une variable qui le nomme. Vous ne pouvez donc pas retourner une telle variable si elle est créée localement. (ou vous obtiendrez des erreurs du type `error : cannot bind non-const lvalue reference`)
   - vous pouvez contourner ces difficultés en utilisant systématiquement des pointeurs, mais alors quelque chose de ce cours vous échappera...
3. Redéfinissez l'opérateur d'affichage pour les calendrier.
4. Ecrivez les méthodes adaptées pour faire avancer le temps, soit d'une durée donnée par une horloge, une date ou un calendrier.
5. Finalement, relisez une fois l'ensemble de votre travail en ajoutant le mot clé **const** partout où une méthode laisse soit l'objet concerné invariant soit les arguments passés invariants.

2 Modélisation : Un peu de musique

Présentation

Cette partie a pour but de modéliser de manière très simplifiée des informations sur des morceaux de musique classique.

Nous considérerons qu'un morceau de musique est fait pour un instrument principal : par exemple, si on parle d'une sonate pour violon, on considèrera que l'instrument principal est le violon.

De même, un morceau a une tonalité principale, par exemple « Mi mineur » pour une « fugue en Mi mineur ». (cf. plus bas)

Bien sûr, un morceau a un nom et un compositeur, un seul pour simplifier.

Explication des classes d'énumération

Les classes d'énumérations, sont des classes dont les objets ne peuvent avoir que les valeurs indiquées lors de la déclaration de la classe.

Exemple d'utilisation :

```
// Déclaration de la classe
enum class Dir {est, ouest, nord, sud};

// Surcharge de l'opérateur <<
inline ostream& operator<<(ostream& out, Dir d) {
  switch (d) {
    case Dir::est: out << "est";break;
    case Dir::ouest: out << "ouest"; break;
    case Dir::nord: out << "nord";break;
    case Dir::sud: out << "sud";break;
  }
  return out;
}

// Déclaration et initialisation d'un objet de type Dir
Dir d {Dir::ouest};

// Exemple d'utilisation avec la surcharge
cout << d << endl; // Affiche "ouest"
```

Vous utiliserez ce genre de classe partout où cela a un sens.

Tonalité

Une tonalité est la donnée d'un nom de note (Do, Ré, Mi, Fa, Sol, La ou Si), d'une altération éventuelle (dièse ou bémol) et d'un qualificatif (mineur ou Majeur).

Classes à implémenter

Après avoir dessiné leur diagramme UML, programmez les classes suivantes en testant régulièrement (il est possible d'ajouter des classes supplémentaires si nécessaire) :

Note : qui représente un nom de note (Do, Ré, Mi, Fa, Sol, La ou Si).

Altération : qui représente une altération ou une absence d'altération.

Qualificatif : mineur ou Majeur.

Tonalité : cf. plus haut.

Instrument : un instrument a un nom et une famille. Le violon fait partie des cordes, la trompette des vents, les cymbales des percussions...

Morceau : cf. plus haut.

Écrivez les méthodes d'affichage nécessaires.