TD5 : Héritage (suite)

Suite TD4

Finir le TD4 puis le compléter avec les questions suivantes.

On suppose maintenant que certains évènements durent plusieurs jours (conférences, festival, fête).

Classes Abstraites

a) Faire une classe abstraite « `Evt` » qui comportera la fonction virtuelle pure « `Afficher` ».
b) Modifier les schémas de dérivation des classes précédentes pour prendre en compte cette nouvelle classe. Remonter les attributs « `date` » et « `objet` » pour qu'ils fassent maintenant partie de la classe « `Evt` ».
c) Modifier les classes « `Ev_1j` » et « `Agenda` » et la fonction `operator<<()` afin de tenir compte de ces changements.
d) Vérifier que la classe `Evt` n'est pas instanciable.
e) Définir une classe « `Ev_pj` » (évènement de plusieurs jours) qui est un « `Evt` » avec une date de début et qui dure un certain nombre de jours (on peut aussi considérer que c'est un évènement qui se passe sur un intervalle de jours).

Design Pattern Template Method

Appliquer le design pattern template method en procédant de la manière suivante :

- Déclarer la méthode virtuelle pure "`string Evt::to_string() const`" qui renvoie une chaine de caractères décrivant un objet évènement.
- Implémenter cette méthode pour chacune des classes concrètes de la hiérarchie de classe en utilisant la classe standard `stringstream` (voir l'exemple de l'exercice corrigé N°24).
- Rendre la méthode `Afficher` concrète dans la classe `Evt` et éliminer les anciennes implémentations de cette méthode dans les classes concrètes.

Transtypage dynamique

Définir un opérateur « `operator<()` » dans toutes les classes où cela est nécessaire afin de comparer deux évènements dans le temps. Etudier les conversions (up-casting et down-casting) entre objets qui ont un lien de parenté.

Reconnaissance de type à l'exécution

Définir une méthode « `Agenda::Statistiques`» qui permet de connaître le nom des différents types d'évènement présents dans un agenda ainsi que le nombre d'occurrence d'évènements pour chaque type d'évènement. On supposera que ces types ne sont pas connus à l'avance (mais on pourra supposer qu'il n'existe pas plus de 10 types différents).

Générateur de nombres pseudo-aléatoires (Design pattern Adapter)

On dipose de la classe `AleatoireIntGenerateur` fournie dans les fichiers generateur.h et generateur.cpp.

La méthode statique `random()` utilise 3 attributs statiques `rootA`, `rootB`, `rootC`. Chaque fois que `random()` est appelée, ces trois attributs sont modifiées avec un calcul spécifique. Un réel de type

`double` de valeur comprise entre 0 et 1 est alors généré en extrayant la partie décimale d'un nombre obtenu à partir de `rootA`, `rootB`, `rootC`. La méthode `GetInt(max)`, qui utilise la méthode `random()`, permet alors de générer un entier dans l'ensemble `{0,...,max}` où `max` est une valeur de type `int` donnée.

- Remarque 1 : Les fonctions `floor` et `ceil` permettent respectivement d'obtenir le plus grand entier inférieur et le plus petit entier supérieur à un nombre réel donné.
- Remarque 2 : Il n'est absolument pas nécessaire de comprendre ce calcul pour faire l'exercice.

Soit la classe abstraite suivante :

```
class RandomGenerator {
protected:
      unsigned int max;
public:
      unsigned int GetMax() const { return max; }
      RandomGenerator(int m):max(m){}
      // renvoie un nombre aléatoire de {0,...,GetMax()}
      virtual unsigned int GetNumber() const=0;
};
```

Vous développez le nouveau logiciel de simulation **SIMULA3000** dans lequel vous utilisez un objet générateur de nombres pseudo-aléatoires **représenté par une variable de type pointeur sur la classe abstraite `RandomGenerator`**. Le constructeur prend en argument une valeur m qui indique que le générateur permettra de générer des nombres de l'ensemble `{0,...,m}`.

En utilisant le design pattern "**Adapter**", développer une nouvelle classe appelée `RandomIntGenerator` qui adapte la classe `AleatoireIntGenerateur` de telle sorte qu'un objet instanciant cette classe puisse être utilisé comme générateur de nombres pseudo-aléatoires dans **SIMULA3000**.

Faire cette question deux fois. Une fois en utilisant un **adaptateur de classe**, une fois en utilisant un **adaptateur d'objet**.