Exercice complet

D'un point de vue technique, cet exercice permet de valider l'acquisition de différents concepts et techniques de base de la programmation orientée objet : la création de classes et différents principes attachés, le déploiement d'un logiciel de plusieurs classes, l'utilisation de classes que l'on n'a pas créées soi même etc.

En pratique, il s'agit de comparer des stratégies d'investissement sur trois cours en lançant plusieurs simulations. La démarche est séparée en étapes. La difficulté augmente avec chaque étape mais le travail reste très guidé.

(0) Notez la manière de manipuler une liste de valeurs réelles. On utilise un objet de type vector qui exige d'avoir préalablement fait le lien vers la déclaration de la classe : #include **<vector>** dans les instructions au début. Par exemple, pour définir un vecteur de double :

vector<double> listeValeursReelles;

Pour introduire des données dans le vecteur :

```
double x=0.8;
double y=0.9;
listeValeursReelles.push_back(x);
listeValeursReelles.push_back(y);
```

Pour récupérer les valeurs et les manipuler :

```
double g=listeValeursReelles.at(0); /*g prend la valeur de x*/
double u=listeValeursReelles.at(1); /*g prend la valeur de y*/
```

Note : le type de l'objet est **vector<double>** et non pas **vector**. Cf pour une fonction qui prend un objet de type **vector<double>** : **void manipulationListe(vector<double> liste)**.

Noter que le premier élément est numéroté à 0. La taille de la liste **listeDoubles** de type **vector<double>** est donnée par **listeDoubles.size()**. Donc les indices varient entre **0** et **listeDoubles.size()-1**

(1) Implémentez une classe Action avec pour attributs :

- un attribut **nomAction** de type chaîne de caractères qui correspond au nom du titre,
- un attribut **valeursHistoriques** : une liste de valeurs réelles qui correspondent aux différentes valeurs prises par le titre au cours du temps.

Placez un **#include <iostream>** en première ligne du fichier source.

En dessous des instructions #include : placer **using namespace std;**

(2) Implémentez les méthodes suivantes pour la classe Action :

- Une méthode **variationGens(int per1,int per2)** qui renvoie la variation de l'action entre per1 et per2.
- Un constructeur de Action qui prend pour paramètre une chaîne de caractères et renseigne **nomAction** avec cette chaîne de caractères.
- Implémentez une fonction **moyenne(int per1,int per2)** qui calcule la valeur moyenne entre deux périodes données.
- Implémentez une fonction **ecartType(int per1,int per2)** qui calcule la variance du cours entre deux périodes données. Sachant que vous disposez de la fonction

  pow(double number,double pow)

  double x=pow(number,pow); affecte à x number à la puissance pow.

  Il faut bien sûr au préalable avoir faire un **#include <cmath>** au début du fichier.

(3) (Facultatif) Proposez une solution pour que lors de chaque appel de **moyenne** et **ecartType**, les calculs ne soient relancés que dans le cas où l'attribut **valeursHistoriques** a été modifié.

(4) On dispose désormais de cours d'actions contenus dans des fichiers textes indépendants : les fichiers c1.txt, c2.txt, c3.txt :

http://rdorat.free.fr/Enseignement/POO/Java/ExRef/c1.txt

http://rdorat.free.fr/Enseignement/POO/Java/ExRef/c2.txt

http://rdorat.free.fr/Enseignement/POO/Java/ExRef/c3.txt

On veut pouvoir créer des occurrences de la classe Action qui prennent les valeurs enregistrées dans ces fichiers pour historiques de cours. On vous fournit un fichier cpp :

http://rdorat.free.fr/Enseignement/POO/CPP/ExRef/RecupCours.cpp

à placer à côté du fichier de code qui contient votre classe Action. Au début de la du fichier source qui contient votre classe Action :

#include "RecupCours.cpp"

Vous pouvez ainsi utiliser l'ensemble des fonctionnalités de ce fichier source. Vous pouvez notamment utiliser :

vector<double> recuperationValeurs(string chemin); : chemin doit être un chemin vers l'un des fichiers c1.txt, c2.txt. Enregistrez les fichiers c1.txt, c2.txt sur votre disque dur. Les fichier c1.txt,..., c3.txt sont de la forme :

0 17

1 18.5

2 19

3 19.7

4 16
5 16.7
6 16.8

Si on met le fichier c1.txt dans le répertoire c:\Cours, pour récupérer le tableau de double qui correspond à un fichier, on fait :

vector<double> v=recuperationValeurs("c:\\Cours\\c1.txt");

Modifiez le constructeur de la classe Action, de manière à ce qu'il reçoive un chemin comme paramètre en plus du nom de l'action et qu'il utilise la fonction **recuperationValeurs** pour récupérer le cours d'une action dans son tableau **valeursHistoriques**.

(5) (Facultatif) Que se passerait il si on ne mettait pas la classe RecupCours dans le répertoire bin ?

(6) Dans le main, créez trois objets de type Action qui prennent respectivement les cours contenus dans les trois fichiers c1.txt, c2.txt et c3.txt. Testez la compilation et l'exécution (à moins de spécifier des instructions d'affichage, rien ne doit apparaître à l'exécution). Faites quelques vérifications en vérifiant que les tableaux **valeursHistoriques** des objets Action contiennent bien les valeurs des fichiers associés (c1.txt, c2.txt, x3.txt) par quelques affichages.

(7) On commence par initialiser le générateur aléatoire : placez **srand(time(0));** en début de main pour initialiser le générateur aléatoire. A partir de là, **rand()** renvoie un entier pris au hasard dans [0;RAND_MAX]. RAND_MAX est une constante.

En utilisant un générateur aléatoire, on lance une simulation dans le main. Un portefeuille est ici une répartition entre trois actions. Le portefeuille initial est 0.33, 0.33 et 0.34. A chaque période :

a- on liquide le portefeuille courant

b- on choisit un nouveau portefeuille, soit trois valeurs dans [0:1] de somme 1.

c- on achète le nouveau portefeuille

Au cours du temps, on peut constater l'évolution de la valeur accumulée (perte cumulée ou gain cumulé). Implémentez cette simulation avec une stratégie aléatoire qui choisit à chaque étape b) son portefeuille au hasard. Pour implémentez cette stratégie aléatoire, vous pouvez utiliser la fonction suivante après l'avoir placée en dessous de la fonction main, elle renvoie un tableau de trois valeurs dans [0:1] et dont la somme est 1.

```
/*Tire trois valeurs réelles telles que la somme est à 1*/
vector<double> repartition3()
```

```
{
vector<double> repartition;
double x=rand()/RAND_MAX;
double y=rand()/RAND_MAX;
        while(x+y>=1)
        {
        x=rand()/RAND_MAX;
        y=rand()/RAND_MAX;
        }
double z=1-x-y;
int tirage=rand()%3;
        if(tirage== 0)
        repartition.push_back(x);
        else if(tirage==1)
        repartition.push_back(y);
        else
        repartition.push_back(z);
int tirage2=rand()%3;
        while(tirage2==tirage)
        tirage2=rand()%3;

        if(tirage2== 0)
        repartition.push_back(x);
        else if(tirage2==1)
        repartition.push_back(y);
        else
        repartition.push_back(z);

int tirage3=3-tirage-tirage2;
        if(tirage3== 0)
        repartition.push_back(x);
        else if(tirage3==1)
        repartition.push_back(y);
        else
        repartition.push_back(z);
return repartition;
}
```

Il suffit ensuite de faire appel à la fonction **repartition3()** pour obtenir un vecteur de 3 valeurs dans [0:1] dont la somme est 1. Par exemple **vecteur<double> v=repartition3();** affecte à v un vecteur de 3 valeurs v.at(0), v.at(1) et v.at(2) dont la somme est 1, soit un portefeuille tiré aléatoirement.

Après avoir implémenté la stratégie aléatoire, vous pouvez tester la valeur qu'elle permet d'accumuler ?

(8) Inventez une autre stratégie que la stratégie aléatoire et testez la.

(9) Inventez une autre stratégie contenant une composante aléatoire et testez la.

(10) Proposer une évolution de votre code qui permettrait de faire plusieurs simulations pour chaque stratégie aléatoire afin que les comparaisons entre stratégies ne soient pas biaisées.