TD/TP4 : semaines 11 à 13 : voir Planning du cours 2018 MAJ le 06/05
« Gestion d'un portefeuille» - Episode en 2 parties

Partie 1 : Gestion d'un portefeuille en mode console - semaines 11 et 12

Dans cette première partie, on souhaite réaliser les fonctionnalités nécessaires à la gestion d'un portefeuille en mode console, pour venir en aide aux « traders en herbe » que nous sommes :
- Rechercher un fonds d'investissement et un instrument
- Ajouter ou supprimer un fonds et un instrument
- Trier les fonds d'un instrument par montants
- Faire des statistiques sur les instruments et les fonds

Avant-propos :
■ Dans votre diagramme de classes du TD4, ajouter les nouvelles classes, relations interclasses, attributs et méthodes mentionnées dans les questions de la **partie 1**. **Ce diagramme par binôme de TP (indiquez les 2 noms) sera ramassé à la fin de la première séance de la semaine 11** : pour plus de détails, cliquer sur le lien Planning du cours 2018 MAJ le 06/05 également en haut de la page campus « POO Java ». Lisez attentivement les consignes de cette partie 1 sans vous occuper des détails techniques du code.
■ Dans votre code, n'oubliez pas de tester chaque méthode avec le « Contrôleur » (**main** dans une classe à part), selon la méthode *Modèle-Vue-Contrôleur*. Puis d'affichez les résultats dans la « Vue » en mode console dans une autre classe que celle du «Contrôleur ». Ajoutez les constructeurs et les getters/setters que vous jugez nécessaires dans le « Modèle ».

1.1. Définir les 3 classes suivantes du « Modèle » :

1) La classe *Fonds* contient un seul attribut : le montant du fonds *amount* de type **double**.

2) La classe *Instrument* contient un seul attribut : une collection d'objets **ArrayList**<*Fonds*> comme valeurs des fonds de l'instrument.

3) La classe *Portefeuille* possède 2 **HashMap** en attributs :
- Une **HashMap** pour les fonds, avec une clé de recherche du fonds de type *String* et un objet de la classe *Fonds* comme valeur associée à la clé du fonds ;
- Une **HashMap** pour les instruments, avec une clé de recherche de l'instrument de type *String* et un objet de la classe *Instrument* comme valeur associée à la clé du fonds.

N'oubliez pas d'implémenter tous les constructeurs par défaut (sans paramètre) et avec paramètres, pour pouvoir instancier des fonds et des instruments. Le constructeur par défaut de *Portefeuille* instancie les deux **HashMap**.

1.2. Dans la classe *Instrument* implémenter la méthode qui, dans sa collection d'objets, ajoute un objet de la classe *Fonds* en paramètre.

1.3. Dans la classe *Portefeuille*, implémenter la méthode de recherche d'un fonds, avec en paramètre la clé du fonds recherché. Si la clé en paramètre n'existe pas dans la **HashMap** des fonds, cette méthode génère et propage l'exception *FondsInexistant* ; sinon elle retourne le montant *amount* associé à cette clé.

1.4. Dans la classe *Portefeuille*, implémenter la méthode de recherche d'un instrument, avec en paramètre la clé de l'instrument recherché. Si la clé en paramètre n'existe pas dans la **HashMap** des instruments, cette méthode génère et propage l'exception *InstrumentInexistant* ; sinon elle retourne la collection d'objets **ArrayList**<*Fonds*> associé à cette clé.

1.5. Dans la classe *Portefeuille*, implémenter une méthode qui instancie et ajoute un nouveau fonds dans la **HashMap** des fonds. Cette méthode a deux paramètres : la clé et le montant du fonds à ajouter. Si la clé existe déjà, cette méthode génère et propage l'exception *FondsExistant*. Tester cette méthode de manière interactive dans le **main** :

- Lire au clavier la valeur de clé d'un fonds et son montant ;
- Appeler la méthode de recherche d'un fonds de la question **1.3** avec la clé saisie en paramètre ;
- Si l'exception *FondsInexistant* est générée, instancier et ajouter le fonds à la **HashMap** des fonds ; sinon générer l'exception *FondsExistant*.

1.6. Dans la classe *Portefeuille*, implémenter une méthode qui ajoute un nouveau fonds à un instrument. Cette méthode a deux paramètres : la clé de l'instrument et le nouveau fonds instancié à ajouter. Tester cette méthode de manière interactive dans le **main** :

- Lire au clavier la valeur de clé d'un instrument ;
- Appeler la méthode de recherche d'un instrument de la question **1.4** avec la clé saisie en paramètre ;
- Si l'exception *InstrumentInexistant* est générée, instancier l'instrument et l'ajouter à la **HashMap** des instruments.
- Appeler la méthode de la question **1.2** pour ajouter le nouveau fonds dans la collection de l'instrument.

1.7. Dans la classe *Portefeuille*, implémenter une méthode qui supprime un fonds (de sa **HashMap**), avec en paramètre sa clé. Cette méthode appellera d'abord la méthode de recherche du fonds (question **1.3**). Si l'exception *FondsInexistant* est générée, afficher un message d'erreur ; sinon supprimer le fonds recherché.

Avec les mêmes règles, implémenter aussi la méthode de suppression d'un instrument en vidant d'abord sa collection de fonds.

Comme dans les 2 questions précédentes, testez ces deux méthodes de manière interactive dans le **main**.

1.8. Définir une nouvelle classe qui implémente l'interface **Comparable**<*Fonds*> : voir le cours Interfaces comparable et comparator sur la page campus « POO Java ». Implémenter les deux méthodes suivantes, permettant de comparer le montant *amount* entre 2 fonds :

- La méthode *equals*, redéfinie par héritage de la classe **Object**, a pour paramètre un objet de la classe *Fonds*. Elle retourne **true** si le montant de l'objet en paramètre est égal au montant de l'objet du fonds référencé, sinon elle retourne **false**.

- La méthode *compareTo* a pour paramètre un objet de la classe *Fonds*. Elle retourne l'une des 3 valeurs suivantes : 1 si le montant du fonds référencé est supérieur au montant de l'objet en paramètre, 0 si les deux montants sont égaux (appel de la méthode *equals* précédente), sinon -1.

1.9. Dans la classe *Instrument*, implémenter une méthode qui trie sa collection de fonds par montant. Pour cela, servez-vous de la méthode **static sort** de la classe **java.util.Collections** (voir Documentation des API Java (javadoc) et cours Interfaces comparable et comparator) et de la méthode de comparaison dela question **1.8**. Testez votre tri en affichant dans la « Vue » les fonds d'un instrument, triés par montant.

1.10. Dans une nouvelle classe de la « Vue », implémenter les méthodes statistiques suivantes :

- Une méthode qui affiche, pour chaque instrument, sa clé, son nombre total de fonds et la somme totale des montants de ses fonds.
- Une méthode avec en paramètre la clé d'un fonds, à rechercher avec la méthode de la question **1.3**. Si l'exception *FondsInexistant* est générée, afficher un message d'erreur ; sinon afficher le pourcentage de chaque instrument pour ce fonds.

Tester ces méthodes dans le **main**.

La partie 1 est à déposer sur Déposez le code du TP4 partie 1 en mode console au plus tard le dimanche 27 mai 2018 à 23h55 sur la page campus « POO Java » : consultez bien les consignes spécifiées.

Partie 2 : « sérialisation/dé-sérialisation » du portefeuille et « Vue graphique évènementielle» de la gestion du portefeuille – semaines 12 et 13

Dans cette seconde et dernière partie du TP, on veut améliorer la gestion du portefeuille de la **partie 1** sur deux aspects :

- « Sérialiser » et « dé-sérialiser » les objets de ce portefeuille à partir d'un fichier
- Interagir en mode graphique pour intégrer toutes les fonctionnalités de la première partie de ce TP

2.1. Définir une nouvelle classe qui implémente l'interface **java.io.Serializable** (voir Documentation des API Java (javadoc), Sérialisation en Java et Exemples de sérialisation sur campus), et implémente les méthodes suivantes avec en paramètre un nom de fichier :

- Une méthode qui « sérialise » dans le fichier les 2 **HashMap** de la classe *Portefeuille* (voir question **1.1**). Propagez la (ou les) exception(s) prédéfinie(s) nécessaire(s).

- Une méthode qui charge du fichier les 2 **HashMap** « sérialisés » et les « dé-sérialise » dans les attributs des classes *Portefeuille*, *Instrument* et *Fonds*. Propagez la (ou les) exception(s) prédéfinie(s) nécessaire(s).

Tester ces méthodes dans le **main** :

- Lire au clavier le nom du fichier à « sérialiser » ;
- Lire au clavier la valeur de clé d'un fonds ;
- Appeler les deux méthodes ci-dessus avec le nom du fichier en paramètre ;
- En cas d'exception(s) générée(s), afficher un message d'erreur clair ; sinon afficher les statistiques des méthodes de la question **1.10**.

2.2. Implémenter votre « Vue graphique évènementielle » pour la gestion de votre portefeuille, dépendante de la **partie 1**, en respect des consignes suivantes :

- Pour chacune de vos classes « graphiques », commentez vos composants graphiques **Swing** et la mise en forme (*layout*) de vos containers (**JPanel** etc.), en expliquant les spécificités ces « layouts » (taille, position des composants etc.).

- La fenêtre d'accueil doit présenter un menu avec toutes les fonctionnalités nécessaires (un composant par fonctionnalité) à la gestion de votre portefeuille :
  - Sérialiser le portefeuille dans un fichier,
  - Dé-sérialiser les objets du portefeuille d'un fichier,
  - Recherche d'un fonds,
  - Recherche d'un instrument,
  - Ajout et suppression d'un fonds,
  - Ajout et suppression d'un instrument,
  - Statistiques,
  - Quitter la fenêtre d'accueil

- Le clic sur chaque composant (fonctionnalité) de la fenêtre d'accueil ouvrira une nouvelle fenêtre (sauf Quitter) avec une « Vue graphique évènementielle » spécifique, affichant des résultats ou permettant des saisies, sous la forme que vous souhaitez. Mais la visualisation et l'interaction avec l'utilisateur doivent être claires, fluides et ergonomiques.

La partie 2 est à déposer sur Déposez le code du TP4 partie 2 en mode sérialisé et graphique au plus tard le dimanche 10 juin 2018 à 23h55 sur la page campus « POO Java » : consultez bien les consignes spécifiées.