TP4 JAVA

Concepts abordés

- Classe, Encapsulation, Héritage, Généricité, Polymorphisme, Collections, Interface, **Exceptions**

Consignes générales de travail

- Commencez chaque TP dans un nouveau *Projet Java (ici TP4)*
- Créez toujours un (ou plusieurs) paquetages pour contenir ses classes (ne jamais utiliser le paquetage par défaut).
- Écrivez chaque nouvelle classe dans son propre fichier (portant le nom de la classe).
- Pour ce TP, déclarez systématiquement vos attributs en private et vos méthodes en *package friendly* ou public.
- Définissez systématiquement les méthodes equals():boolean, hashcode:int et toString():String *pour chaque classe* que vous écrivez.
- Lisez chaque exercice en entier avant de commencer vos réponses.

Pour mener à bien ces exercices, il est impératif d'ouvrir

1) **la Javadoc http://docs.oracle.com/javase/6/docs/api/**
2) le tutoriel de Sun sur le collections
   http://docs.oracle.com/javase/tutorial/collections/index.html

Partie A: les exceptions

Exercice 1: Moyenne

La classe suivante permet de calculer la moyenne des entiers passés en paramètres au programme.

Il est demandé de gérer les exceptions **NumberFormatException** et **ArithmeticException dans la méthode main.**

Rappel: x / 0 pas bien du tout...

SOURCE DE LA CLASSE SANS LES EXCEPTIONS

```
package moyenneException;

public class MoyenneException{

	public static void main(String[] argv) {

		if (argv.length <= 0) {
			System.out.println("Il n'y a pas d'arguments");
			System.exit(1);
		}

		int moyenne = moyenne(argv);
		System.out.println("La moyenne est " + moyenne);
	}

	public static int moyenne(String[] argv) {
			int somme = 0;
			int cpt = 0;
			for (int i = 0; i < argv.length; i++) {

				somme += Integer.parseInt(argv[i]);
				cpt++;
			}
			return somme / cpt;
		}
}
```

Exercice 2: Racine

Il s'agit d'écrire une classe servant à calculer la racine carrée d'un nombre réel de type **double**.
Ce nombre sera passé comme argument au programme.
La méthode **racine** lance une exception dans le cas où le **double** qui lui est passé en paramètre est négatif.

- On définira une classe étendant la classe Exception, correspondant à la classe de l'exception qui sera levée dans le cas d'un opérande négatif.

- Dans le cas d'un opérande négatif, on lancera une exception de la classe définie depuis l'intérieur de la **méthode racine** de la classe Racine. Cette erreur ne sera pas attrapée par la **méthode racine** mais le sera dans la méthode main qui utilise la méthode calcule.

SOURCES DES CLASSES SANS LES EXCEPTIONS

```
public class Racine
{
        private double precision;

        Racine(double precision)
          {
           this.precision=precision;
          }

        double racine(double operande)
          {
           double solution, carre, a, b;

           a=0;
           b=operande;
           solution=(a+b)/2;
           while(b-a>precision)
                {
                    carre=solution*solution;
                    if (carre>operande) b=solution;
                    else a=solution;
                    solution=(a+b)/2;
                }
           return solution;
```

```
        }

        double getPrecision()      { return precision; }

        void setPrecision(double precision) {  this.precision=precision; }
}

public class UtiliseRacine
{
        public static void main(String[] argv)
        {
            Racine extracteur = new Racine(0.01);
            double operande= (Double.valueOf(argv[0])).doubleValue();
            double valeurRacine;

            valeurRacine=extracteur.racine(operande);
            System.out.println(valeurRacine);
            System.out.println("precision "+extracteur.getPrecision() +  " , erreur =" +
                                                    (valeurRacine-Math.sqrt(operande)));
            extracteur.setPrecision(0.00001);
            valeurRacine=extracteur.racine(operande);
            System.out.println(valeurRacine);
            System.out.println("precision "+extracteur.getPrecision() +  " , erreur =" +
                                                    (valeurRacine-Math.sqrt(operande)));

        }
    }
```

On obtient :

```
6.55865478515625
precision 0.01 , erreur =0.00121626085424964
6.557437360286713
precision 1.0E-5 , erreur =-1.1640152877134824E-6
```

Partie B: retour sur le TP3 et l'interface Condition

1) Créer une nouvelle **classe ExceptionArgumentIncorrect** sans attribut qui ne contient que le constructeur transmettant son paramètre message de type **String** au constructeur de sa super-classe.
2) Le constructeur de la classe **InclusIntervalle** doit lancer une telle exception lorsque la borne inférieure n'est pas strictement inférieure à la borne supérieure. Le message doit être de la forme 40 >= 35 si les paramètres sont 30 et 45.
   Que faut-il ajouter à la signature du constructeur pour ne pas avoir d'erreur de compilation?
3) Dans la classe **TestConditions**, afficher l'**exception + "Pas de comptage possible."** lorsque l'exception se produit, et dans ce cas, n'effectuer aucun comptage.

Partie C: une drôle de partie de football

Nous allons nous intéresser à différents acteurs d'une partie de football.

Il est vivement conseillé de lire toutes les questions de chaque partie avant de commencer à rédiger vos réponses. **Un diagramme de classes figure en Annexe**.

Première partie: classe Acteur

1) Dans un package **football.** définir une classe **Acteur**, non instanciable, possédant des attributs **nom** et **prenom** de type **String et age** de type **int.** Ces attributs ont une visibilité restreinte à la classe. Les attributs **nom** et **prenom,** une fois initialisés, ne pourront plus changer de valeur.
2) Déclarer une **méthode abstraite getSalaire** sans paramètre retournant une valeur numérique de type **double.**
3) Ecrire un **constructeur** à **3** paramètres permettant d'initialiser les attributs de la classe **Acteur.**
4) Ecrire les accesseurs en lecture et en écriture nécessaires pour ces 3 attributs.
5) Ecrire la méthode **toString** permettant de retourner toute l'information portée par les 3 attributs et le salaire de l'acteur.

Seconde partie: classe Joueur

6) Sachant qu'un joueur est un acteur particulier, définir, dans le package **football,** une classe **Joueur** possédant un attribut **club** de type **String** et un attribut **euroMillons** de type **double** représentant le montant potentiel de son transfert. Ces attributs ont une visibilité réduite au package **football**.
7) Dans la classe **Acteur**, de quelle manière pourrait être modifiée l'accessibilité des attributs et du constructeur. Quelle serait la conséquence de cette modification?

Dans la suite de l'exercice, vous pourrez utiliser cette modification si vous le souhaitez.

8) Ecrire un **constructeur** à **5** paramètres permettant d'initialiser une instance de la classe **Joueur**.
9) Ecrire un **constructeur** à **4** paramètres permettant d'initialiser une instance de la classe **Joueur** dont le club est le *''Knysna Syndrome Club''**. Ce constructeur devra utiliser celui à **5** paramètres.

10) Ecire l'accesseur en lecture pour l'attribut club.
11) Le salaire d'un joueur est un pourcentage aléatoire inférieur à 100% de la valeur de l'attribut euroMillions. Définir la méthode adéquate.
    Indication: utiliser Math.random() retournant un double compris entre 0.0 (inclus) et 1.0 (exclus).
12) Ecrire la méthode **toString** permettant de retourner toute l'information portée par tous les attributs et le salaire d'un joueur.

** en hommage à un épisode récent de la glorieuse histoire planétaire du football*

Troisième partie: classes TestJoueurActeur et Arbitre

1) Sachant qu'un arbitre est un acteur particulier, définir dans le package **football** une classe **arbitre,** possédant des attributs nom, prénom, age. Un arbitre est également défini par son **degré d'expérience**: **1 débutant, 2 confirmé, 3 expert.** Vous définirez toutes les méthodes nécessaires à l'utilisation de la classe Arbitre, notamment les méthodes **getSalaire** et **toString.** L'attribut **degreExperience** a une visibilité restreinte au package football.

Pour votre information, le salaire d'un arbitre est un pourcentage aléatoire inférieur à 10% de la valeur de l'attribut **degreExperience**.

2) Ecrire une classe de test **TestJoueurActeur** avec la méthode principale conventionnelle définissant un **container générique de type Set** dont vous choisirez la **classe d'implémentation.**
3) Ranger dans ce container 3 instances de la classe Joueur appartenant tous à la légendaire équipe *'Knysna Syndrome Club"*. Si vous ne connaissez rien au football, choisissez vos meilleurs amis et faites en des footballeurs...
4) Dans la classe de test, instanciez un arbitre de **nom** *Kivoitou*, de **prénom** *'Kantantou,* **âgé de** *45 ans* et de **degré d'expérience** *expert*.
5) Ranger l'arbitre dans le même container que les 3 joueurs précédents.
6) Ecrire un code permettant l'affichage des éléments du container.

Quatrième partie: classes Equipe et exceptions

Dans le cadre de notre partie de football, écrire une classe **Equipe** dans le package **footbal**l.

Ajouter un attribut de type **Equipe** dans la classe **Joueur** ainsi que son **accesseur en lecture** (revenir dans le code de la classe **Joueur**). Cet attribut est de visibilité restreinte au package.

Cette classe **Equipe** devra respecter les points suivants:

1) avoir une constante de classe de nom **NOMBREJOUEURS** de type **int** et de valeur **11.** Cette constante est de visibilité publique.
2) avoir un attribut **club** de type **String** représentant le nom du club de l'équipe. Cet attribut est de visibilité restreinte à la classe.
3) avoir un **container générique** de votre choix comme attribut afin de ranger uniquement les joueurs de l'équipe. Cet attribut est de visibilité restreinte au package.
4) avoir un constructeur à 1 paramètre de type String effectuant les traitements nécessaires sur les 2 attributs.
5) avoir la méthode **toString** permettant l'affichage du club et de tous les joueurs de l'équipe
6) avoir une méthode d'ajout d'un joueur dans le container.
   Cette méthode devra renseigné l'attribut **equipe** du joueur.
   Cette méthode devra lever les exceptions suivantes:
   - une exception de type **EquipePleineException** si la composition de l'équipe dépasse les 11 joueurs autorisés sur un terrain de football.
     Le message affiché lors du déclenchement de l'exception sera "*Les 12 apôtres ne jouaient pas au football!*".
   - une exception de type **JoueurHorsClubEquipeException** si le joueur n'est pas dans le même club que celui de l'équipe.
     Le message affiché lors du déclenchement de l'exception sera "*Judas que viens tu faire dans notre équipe? Tu ne fais pas partie de notre club! Ton club est <club1> et le nôtre est <club2>, va à Knysna si tu veux mais pas chez nous, sale traître!*", où <club1> sera substitué par le club du joueur incriminé et <club2> par celui de l'équipe.

Ecrire une classe de test **TestEquipe** permettant de tester les méthodes écrites précédemment.

indication: *pour la méthode d'ajout, si vous ne connaissez pas par coeur son nom, en regard de votre choix de container, vous pouvez l'appeler tout simplement* ***add*** .
Des méthodes de nom ***size, contains*** *pourront aussi être utiles*

Cinquième partie: classe Partie

Dans le cadre de notre partie de football, écrire une classe **Partie** dans le package **football.**
On suppose avoir ajouté un attribut **partie** de type **Partie** dans la classe **Arbitre** et **son accesseur en écriture**.

Cette classe, dont tous les attributs seront de visibilté restreinte au package **football**, devra respecter les points suivants:

1) avoir en attribut un tableau pour les 2 équipes en compétition**.**
2) avoir un attribut **arbitre** de type **Arbitre**.
3) avoir un attribut **score** de type **tableau de 2 int** (un entier pour chaque équipe)
4) avoir un constructeur à **3** paramètres, 2 pour les équipes, 1 pour l'arbitre afin d'initaliser les **3** attributs. Le score sera initialisé à 0-0 et l'arbitre sera en mesure de connaître la partie en cours.
5) Avoir une **méthode marquerUnBut** avec un paramètre de type **Joueur** et modifiant le **score** de son équipe en conséquence.
6) avoir une méthode **toString** permettant de retourner toute l'information portée par les 3 attributs.
7) Dans la classe **Arbitre** ajouter une **méthode validerUnBut** pour un joueur passé en paramètre.
8) Ecrire une classe de test **TestPartie** permettant de tester les méthodes écrites précédemment.

Sixième partie: classes Arbitre et JoueurHorsEquipeException

Dans la classe **Arbitre**, écrire une méthode d'expulsion d'un joueur mettant à jour le container de l'équipe du joueur expulsé.

Cette **méthode d'expulsion** devra lever une exception de type **JoueurHorsEquipeException** si le joueur ne fait pas partie de l'équipe: soit qu'il n'en ait jamais fait partie soit qu'il ait déjà été expulsé**.**

Le message affiché lors du déclenchement de l'exception sera *''<nom> ne fait pas partie de l'équipe!'',* où nom sera substitué par le nom du joueur.

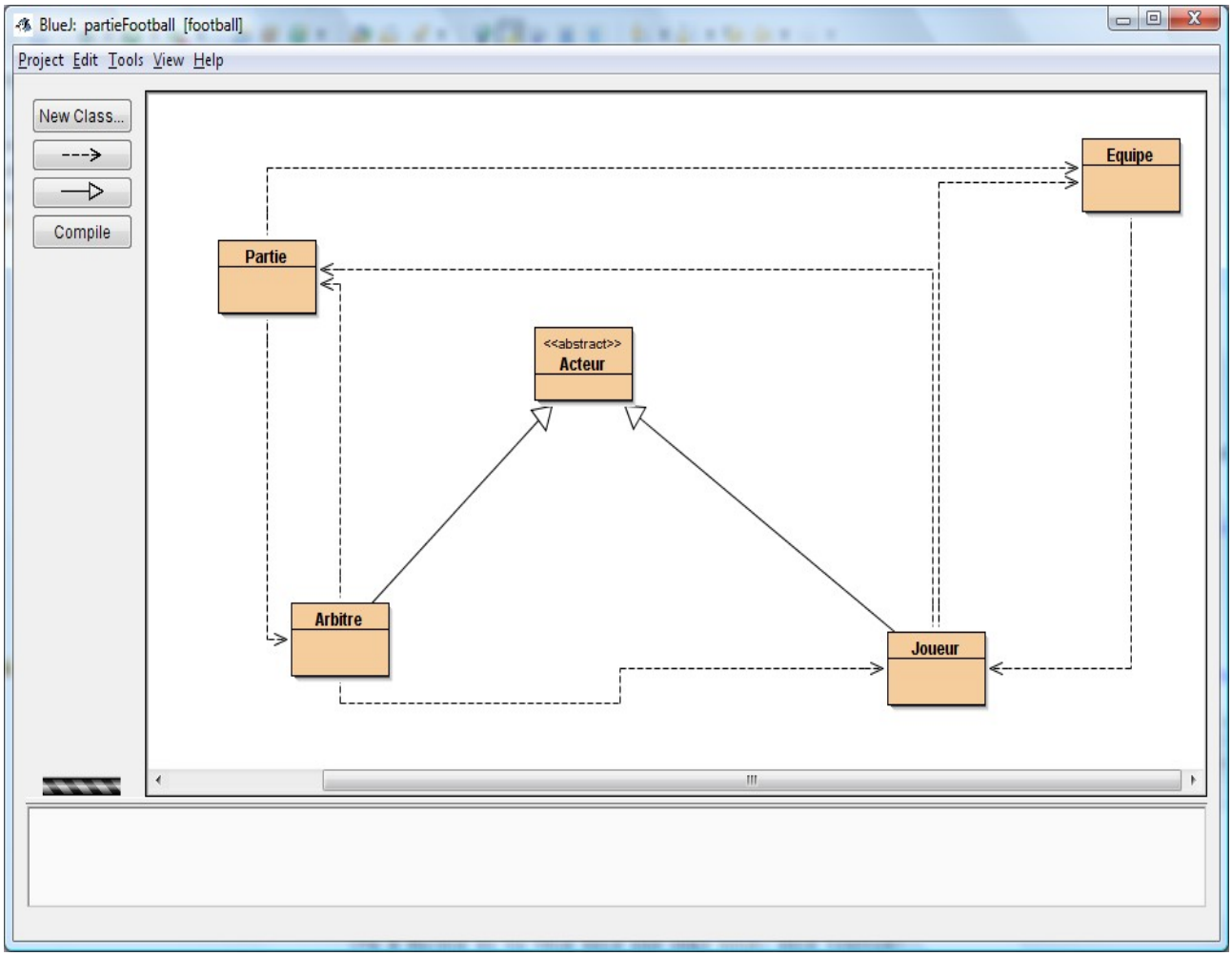
Compléter la classe de test **TestPartie** afin de tester la **méthode d'expulsion** d'un joueur.

indication: *pour la méthode de suppression, si vous ne connaissez pas par coeur son nom, en regard de votre choix de container, vous pouvez l'appeler tout simplement* ***remove****.*
Des méthodes de nom ***size, contains*** *pourront aussi être utiles.*

Septième partie:

On souhaiterait pouvoir classer tous les acteurs d'une partie de football selon leur salaire.
Implémenter ce qui vous semble nécessaire et écrire une classe de test **TestComparaisonSalaires.**

ANNEXE: DIAGRAMME DE CLASSES



Explications pour la fléche ----------> :

- La classe Partie est en relation avec les classes Equipe et Arbitre: des attributs de ces types y sont présents.
- La classe **Equipe** est en relation avec la classe **Joueur**: un *attribut* relatif à des joueurs y est présent et une *méthode à un paramètre* de type **Joueur**.
- La classe **Arbitre** est en relation avec les classes **Partie** et **Joueur** : un *attribut* de type **Partie** y est présent et 2 *méthodes ont un paramètre* de type **Joueur**.
- La classe **Joueur** est en relation avec les classes **Equipe** et **Partie** : un *attribut* de type **Equipe** y est présent et une *méthode à un paramètre* de type **Partie.**

Il faut réfléchir très attentivement à la création des instances de ces classes et à l'installation de références sur les objets.