BRISSEAU Aurélien
GANDON Hippolyte
301A

Projet Ping Pong

année scolaire 2012-2013

Introduction :

Le projet Ping-Pong est réalisé dans le cadre du cours de programmation objet dans le langage de programmation java. Ce projet a pour but de réaliser un programme permettant la gestion d'une compétition de tennis de table ; ici les résultats des matchs sont simulés par un nombre aléatoire.

Celui ci permet une première approche de la mise en place d'un code comprenant plusieurs classes avec des liens entre celles-ci.

Classes :

Classe : **Match**

Attributs :

Variable de classe : Nombre aléatoire

Variables privées :

Tableau de 2 Joueurs

Chaîne de caractères : Heure

Entier : Numéro de table

Entier : Résultat du match

Méthodes :

Constructeur avec modification du nombre de points global des joueurs

Appel de la méthode *addPts* de la classe **Joueur**

affichJoueurs : Méthode d'affichage des fiches des joueurs

Appel de la méthode *affich* de la classe **Joueur**

affich : Méthode affichant les informations sur le match (Joueurs, table, heure)

Appel de la méthode *affichNomPrenom* de la classe **Joueur**

affichResultat : Méthode affichant les informations sur le match (Joueurs, table, heure ainsi que l'issue de celui-ci)

Appel de la méthode *affichNomPrenom* de la classe **Joueur**

getPerdant : retourne le joueur perdant du match

getVainqueur : retourne le joueur vainqueur du match

Classe : **Classement**

Attributs :

Variable de classe : Scanner

Variables privées : Tableau de 16 Joueurs

Méthodes :

Constructeur avec saisie

Appel du constructeur de la classe **Joueur**

Appel de la méthode privée *tri*

Constructeur avec données préenregistrées

Appel du constructeur de la classe **Joueur**

Appel de la méthode privée *tri*

tri : Méthode de tri du tableau en fonction des points

Appel de la méthode *getPtsAvant* de la classe **Joueur**

affich : Méthode d'affichage des fiches des joueurs (soit tous soit un par son nom)

Appel de la méthode *affich* de la classe **Joueur**

Appel de la méthode *getNom* de la classe **Joueur**

getJoueur : retourne le joueur i(0,1,…,15) du classement

Appel de la méthode *getNom* de la classe **Joueur**

Classe : **Poule**

Attributs :

Variables privées :

Entier : Numéro de Poule

Tableau de 4 Joueurs

Tableau de 6 Matchs

Méthodes :

Constructeur : Joueurs sélectionnés dans le classement c à partir du numéro de Poule ; table indique la première table utilisée par la poule ; puis tri par nombre de victoires

Appel de la méthode *getJoueur* de la classe **Classement**

Appel du constructeur de la classe **Match**

Appel de la méthode privée *tri*

affichJoueurs : Méthode d'affichage de toutes les fiches de joueurs de la poule

Appel de la méthode *affich* de la classe **Joueur**

getJoueur : retourne le joueur i(0,1,2,3) de la poule

affichMatchs : Méthode d'affichage de tous les matchs de la poule

Appel de la méthode *affich* de la classe **Match**

affichResultats : Méthode d'affichage de tous les matchs de la poule avec leur résultats !

Appel de la méthode *affichResultat* de la classe **Match**

affichClassement : Méthode d'affichage du classement de la poule

Appel de la méthode *affichNomPrenom* de la classe **Joueur**

Appel de la méthode *getVict* de la classe **Joueur**

tri : Méthode calculant le nombre de victoires pour chaque joueur puis triant le tableau par ordre décroissant du nombre de victoire des joueurs

Appel de la méthode *getVainqueur* de la classe **Match**

Appel de la méthode *victoire* de la classe **Joueur**

Appel de la méthode *getVict* de la classe **Joueur**

Appel de la méthode *getPtsAvant* de la classe **Joueur**

Classe : **Joueur**

Attributs :

Variables privées :

Chaîne de caractères : Nom

Chaîne de caractères : Prénom

Chaîne de caractères : Club

Entier : Nombre de points (classement national)
Entier : Nombre de points avant la compétition
Entier : Nombres de victoires

Méthodes :

Constructeur avec copie du nombre de points indiqués dans la variable ptsavant et victoire initialisé à 0

affich : Méthode d'affichage de la fiche du joueur i.e. toutes ses infos
addPts : Méthode de mise à jour du nombre de points
victoire : Méthode de mise à jour du nombre de victoires
getPtsAvant : retourne le nombre de points du joueur avant la compétition
getVict : retourne le nombre de victoires du joueur
getNom : retourne le nom du joueur
affichNomPrenom : Méthode affichant les nom et prénom du joueur

Aucune des méthodes de la classe **Joueur** ne fait appel à des méthodes d'autres classes si ce n'est "*equals*" de **String**.

Jeux de tests :

La génération de la compétition se déroule comme suit :

- Le classement des 16 joueurs est créer et trier puis, à partir de ce classement, les 4 poules sont crées. Par exemple pour la poule

  Poule p2= new Poule(2,c,3); crée la poule 2 à partir du classement c
  j[0]=c.getJoueur(num-1); récupère le second joueur du classement
  j[1]=c.getJoueur(num+3); récupère le sixième joueur du classement

  …

- Lors de leurs créations les matchs sont générés

  m[0]= new Match(j[0],j[1],"9:00",table); génère le premier match de la poule et son issue

  m[1]= new Match(j[2],j[3],"9:00",(table+1)); génère le second match de la poule et son issue

  …

- Les joueurs de la poule sont ensuite triés en fonction de leur nombre de victoires et leur points avant la compétition en cas d'égalité. Ainsi j[0] est le premier de la poule et j[1] le deuxième. Nous détaillons ce tri plus bas.
- Les matchs de quarts de finale sont créés en sachant les poules triées.

  Match q1=new Match(p1.getJoueur(0),p4.getJoueur(1),"14:00",1);
  Premier quart de finale généré entre le 1er de la poule 1 et le 2e de la poule 4

  …

- Les matchs de demi finales sont instanciés en utilisant les fonctions *getVainqueur* et *getPerdant* sur les matchs de quarts de finale.

  Match d1=new Match(q1.getVainqueur(),q2.getVainqueur(),"15:00",1);

  …

- De même pour les matchs de finale et petite finale

Le tri dans une poule s'exécute de la manière suivante :

```
93     private void tri(){
94         for (int i=0;i<6;i++)
95         {
96             m[i].getVainqueur().victoire();
97         }
98         boolean permut; //tri à bulles :D
99         do
100        {
101            permut= false;
102            for (int i=0; i<3;i++)
103            {
104                if (j[i].getVict()<j[i+1].getVict()||((j[i].getVict()==j[i+1].getVict())&&
    (j[i].getPtsAvant()<j[i+1].getPtsAvant())))
105                {
106                    Joueur tmp=j[i+1];
107                    j[i+1]=j[i];
108                    j[i]=tmp;
109                    permut=true;
110                }
111            }
112        }while (permut ==true);
113    }
```

Prenons l'exemple suivant : Poule 1 juste avant l'exécution de l'algorithme de tri et de calcul du nombre de victoire

Joueur	Nombre de points avant la compétition	Nombre de points après la compétition	Nombre de victoires
A indice 0	620	608	0
B indice 1	599	603	0
C indice 2	518	520	0
D indice 3	513	513	0

Match	j1	j2	result
0	A	B	0
1	C	D	1
2	A	C	0
3	B	D	1
4	A	D	0
5	B	C	1

Ligne 94 : i=0
Ligne 96 : *m[0].getVainqueur().victoire() ;* → m[0].result=0 donc *m[0].getVainqueur()* renvoie B

B indice 1	599	603	1

Ligne 94 : i=1
Ligne 96 : *m[1].getVainqueur().victoire() ;* → m[1].result=1 donc *m[1].getVainqueur()* renvoie C

C indice 2	518	520	1

…

Joueur	Nombre de points avant la compétition	Nombre de points après la compétition	Nombre de victoires
A indice 0	620	608	0
B indice 1	599	603	3
C indice 2	518	520	2
D indice 3	513	513	1

Ligne 101 : permut = false
Ligne 102 : i=0
Ligne 104 : *j[0].getVict() ;* renvoie le nombre de victoire de A soit 0
j[1].getVict() ; renvoie le nombre de victoire de B soit 3
0 < 3 on a donc :

Joueur	Nombre de points avant la compétition	Nombre de points après la compétition	Nombre de victoires
B indice 0	599	603	3
A indice 1	620	608	0
C indice 2	518	520	2
D indice 3	513	513	1

Et permut=true
Ligne 102 : i=1
Ligne 104 : 0 < 2

…

Le tri est un tri à bulle où la permutation à lieu si le nombre de victoire du joueur i est plus petit que celui du joueur i+1 ; ou en cas d'égalité en nombres de victoire si le joueur i avait moins de points avant la compétition.

Conclusion :

Notre programme permet une modélisation satisfaisante de la compétition avec saisis des informations sur les joueurs, calculs des résultats (aléatoire) et affichage de ceux-ci.

Nous n'avons pas considéré les homonymes, cela peut être une amélioration à apporter : lors de la saisie d'un nom qui correspond à deux joueurs le programme demanderait la saisie du prénom comme précision.