Contrôle 3: Informatique 11 avril 2018

Cours de mathématiques spéciales Semestre de printemps

(écrire lisiblement et avec au maximum trois couleurs différentes s.v.p.)

Nom : ……...

Prénom : ……..….

Question	Barème	Points
1	105	
2	55	
3	40	
Total	200	

Note :

Indications générales

- Durée de l'examen : **105 minutes**.
- Posez votre **carte d'étudiant** sur la table.
- La réponse à chaque question doit être rédigée **à l'encre** sur la place réservée à cet effet à la suite de la question.

  Si la place prévue ne suffit pas, vous pouvez demander des feuilles supplémentaires aux surveillants ; chaque feuille supplémentaire doit porter **nom, prénom, nº du contrôle, branche et date**. Elle ne peut être utilisée que pour **une seule question**.
- Les feuilles de brouillon ne sont pas à rendre ; elles ne seront **pas corrigées** ; des feuilles de brouillon supplémentaires peuvent être demandées en cas de besoin auprès des surveillants.
- Les feuilles d'examen doivent être rendues **agrafées**.

Indications spécifiques

- Toutes les questions qui suivent se réfèrent au langage de programmation **Java** (à partir du **JDK 8.0**).
- A part les polycopiés du cours, **aucune** autre source bibliographique n'est autorisée et ne doit donc être consultée durant le contrôle.

Remarques initiales :

- Il est conseillé de lire chaque question jusqu'à la fin, avant de commencer la rédaction de la solution.
- N'oubliez pas d'importer les éléments dont vous avez besoin (et qui ne font pas partie du package ***java.lang***).
- Les indices des éléments des tableaux commencent à zéro.

Question 1

On considère un projet Java muni d'un package nommé ***cms_ctr3*** et qui contient une classe nommée ***Etudiant*** qui est fournie et qui ne doit pas être modifiée, une interface nommée ***FI_Comparateur*** et une classe nommée ***EtudiantPlus*** que vous devez écrire selon les indications données plus loin, ainsi qu'une classe "principale" nommée ***CP_Ctr3Exo1*** dont le canevas est donné et que vous devez écrire selon les consignes précisées sous forme de commentaires.

1.1 Il faut d'abord lire et comprendre le code (fourni ci-dessous) de la classe publique ***Etudiant*** qui permet la création d'objets correspondant à des étudiants caractérisés par leurs noms et leurs âges. Cette classe contient notamment :

- la méthode statique ***confronter*** qui permet de comparer deux étudiants en fonction de leurs âges (et selon la convention habituelle concernant la valeur entière retournée par une méthode qui compare deux objets) ;
- la méthode statique ***trier*** qui a comme argument un tableau de type ***Etudiant*** et qui ne retourne aucun résultat ; cependant, cette méthode trie les étudiants du tableau argument en ordre croissant de leurs âges (grâce à l'utilisation de la méthode ***confronter***) ; suite à l'appel de la méthode ***trier***, le tableau argument contiendra les mêmes éléments qu'avant l'appel mais ordonnés (à partir du premier élément qui correspondra à l'étudiant le plus jeune et jusqu'au dernier élément qui correspondra à l'étudiant le plus âgé).

Vous trouvez ci-dessous le code complet de la classe ***Etudiant*** (et il n'y a pas de réponse à fournir pour ce point).

```
package cms_ctr3;

public class Etudiant{
     private String nom;
     private int age;

     public String getNom(){ return nom; }
```

```
	public int getAge(){ return age; }

	public Etudiant(String nom, int age) {
		this.nom = nom;
		this.age = age;
	}

	static int confronter(Etudiant et1, Etudiant et2) {
		if(et1.age < et2.age) return -1;
		if(et1.age > et2.age) return 1;
		return 0;
	}

	//méthode de tri (en ordre croissant) par insertion
	static void trier(Etudiant etudiants[]) {
		Etudiant temp;
		int i, j;
		for(j = 1; j <etudiants.length; j++)
		{
			temp = etudiants[j];
			i = j-1;
			while(i >= 0 && confronter(etudiants[i], temp) > 0)
			{
				etudiants[i+1] = etudiants[i];
				i = i-1;
			}
			etudiants[i+1] = temp;
		}
	}//fin de la méthode trier

	static void afficher(Etudiant tab[]) {
		for(Etudiant el : tab){
			System.out.println(el);
		}
		System.out.println("************************************");
	}
}//fin de la classe Etudiant
```

1.2 La méthode ***trier*** de la classe ***Etudiant*** présente le désavantage que le tri du tableau argument se fait (toujours) selon l'âge des étudiants. Le travail qui vous est demandé a comme but de rendre l'opération de tri plus versatile, en permettant de préciser le critère de tri au moment de l'appel d'une nouvelle version de la méthode ***trier***. Cette version surchargée de la méthode ***trier*** sera définie dans une nouvelle classe nommée ***EtudiantPlus*** qui dérive de la classe ***Etudiant*** et que vous allez écrire au point **1.3**.

Mais d'abord, vous devez définir (pour ce point **1.2**) une interface fonctionnelle publique nommée ***FI_Comparateur***, qui fait partie du package ***cms_ctr3*** et qui déclare (seulement) la méthode fonctionnelle nommée ***comparer***. Plus précisément, cette méthode doit avoir deux arguments de type ***EtudiantPlus*** et retourner un résultat de type numérique entier.
(L'intérêt de la méthode ***comparer*** sera de faire un travail similaire à celui de la méthode ***confronter*** de la classe ***Etudiant*** mais selon des critères qui pourront être différents d'une implémentation à l'autre).

Ecrire ci-dessous le code complet de l'interface ***FI_Comparateur***.

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

1.3 Comme annoncé plus haut, vous devez maintenant écrire le code complet d'une classe publique nommée ***EtudiantPlus***, qui fait partie du package ***cms_ctr3*** et qui dérive (c'est-à-dire qui est la fille) de la classe de base (ou de la classe mère) ***Etudiant***.

Plus précisément, la classe ***EtudiantPlus*** :

- déclare un champ (propre) d'instance, privé, nommé ***moyenne*** (car il correspond à la note moyenne d'un étudiant), de type numérique réel et sans valeur initiale explicite ;
- définit une méthode **getter** nommée en respectant la convention Java pour le nommage des getters et qui retourne (sans aucune vérification particulière) la valeur du champ privé ***moyenne*** ;
- définit un constructeur public avec trois arguments nommés ***nom***, ***age*** et ***moyenne***, et ayant les mêmes types que les champs d'instance homonymes ; ce constructeur doit respecter les consignes suivantes;
    - par un appel au constructeur de la classe mère, les valeurs des deux premiers arguments sont stockées dans les champs homonymes (hérités) de l'objet créé ;
    - la valeur du troisième argument est stockée (sans aucune vérification particulière) dans le champ homonyme (propre) de l'objet créé ;
- définit une version surchargée statique de la méthode héritée ***trier*** ; en fait, afin de devenir plus versatile (comme expliqué au début du point **1.2**), la nouvelle méthode de tri (qui est une méthode d'ordre supérieur) doit :
    - avoir deux arguments : le premier nommé ***etudiants*** de type tableau de type ***EtudiantPlus*** et le deuxième nommé ***comparateur*** de type interface fonctionnelle définit au point **1.2** ;
    - ne pas retourner de résultat ;
    - garder presque le même corps que la méthode de la classe mère ; entre l'accolade ouvrante et l'accolade fermante qui délimitent le corps de la méthode ***trier*** de la classe fille, on vous demande d'écrire seulement les deux lignes qui sont différentes par rapport au corps de la méthode homonyme de la classe mère ; n'oubliez surtout pas de remplacer convenablement (à l'aide du deuxième argument de la méthode surchargée) l'appel de la méthode ***confronter*** ;
- redéfinit la méthode ***toString*** (héritée de la classe ascendante ***Object***) de sorte qu'elle retourne la chaîne de caractères :

    XXX de YYY ans a la moyenne ZZZ.

où **XXX**, **YYY** et **ZZZ** sont, respectivement, les valeurs des champs ***nom***, ***age*** et ***moyenne*** de l'objet appelant.

Ecrire ci-dessous le code complet de la classe ***EtudiantPlus***.

1.4

Ecrire le code complet de la classe publique ***CP_Ctr3Exo1*** qui appartient au package ***cms_ctr3***. Cette classe "principale" contient notamment la méthode ***main*** qui utilise la classe ***EtudiantPlus***.

Ci-dessous se trouve l'affichage obtenu suite à l'exécution de la classe ***CP_Ctr3Exo1***.

```
Etudiants non triés :
Toto de 20 ans a la moyenne 5.2.
Titi de 24 ans a la moyenne 4.75.
Tata de 18 ans a la moyenne 5.85.
**********************************
Etudiants triés selon la moyenne :
Titi de 24 ans a la moyenne 4.75.
Toto de 20 ans a la moyenne 5.2.
Tata de 18 ans a la moyenne 5.85.
**********************************
Etudiants triés selon l'âge :
Tata de 18 ans a la moyenne 5.85.
Toto de 20 ans a la moyenne 5.2.
Titi de 24 ans a la moyenne 4.75.
**********************************
Etudiants triés selon le nom :
Tata de 18 ans a la moyenne 5.85.
Titi de 24 ans a la moyenne 4.75.
Toto de 20 ans a la moyenne 5.2.
**********************************
```

On présente ci-dessous le squelette de la classe publique ***CP_Ctr3Exo1*** et vous devez compléter ce canevas en fonction des indications données en commentaires.

```
//déclaration du package
.....
.....
.....
//en-tête de la classe "principale"
.....
.....
.....
{
      //en-tête de la méthode main
.....
.....
.....
      {
```

```
        //déclarer une variable locale nommée tab de type tableau
        //de type EtudiantPlus et l'initialiser avec les
        //références de 3 objets correspondant aux étudiants :
        //Toto de 20 ans avec la moyenne 5.20
        //Titi de 24 ans avec la moyenne 4.75
        //Tata de 18 ans avec la moyenne de 5.85
.....
.....
.....
.....
.....
.....
.....
        //afficher le message Etudiants non triés :
.....
.....
.....
        //appeler la méthode afficher pour le tableau tab
.....
.....
.....
        //déclarer une variable locale nommée comparateur et de
        //type FI_Comparateur et l'initialiser avec l'adresse d'un
        //objet (unique) de type classe anonyme qui (re)définit
        //la méthode fonctionnelle pour que la comparaison se
        //fasse selon la moyenne
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
```

```
            //dans la variable locale comparateur définie ci-dessus
            //stocker l'adresse d'une expression lambda qui
            //(re)définit la méthode fonctionnelle pour que
            //la comparaison se fasse selon la moyenne
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
            //appeler la méthode trier pour le tableau tab et la
            //variable comparateur
.....
.....
.....
            //afficher le message Etudiants triés selon la moyenne :
.....
.....
.....
            //appeler la méthode afficher pour le tableau tab
.....
.....
.....
            //appeler la méthode trier en précisant comme premier
            //argument (effectif) le tableau tab et comme deuxième
            //argument (effectif) une expression lambda qui
            //(re)définit la méthode fonctionnelle pour que la
            //comparaison se fasse selon l'âge
.....
.....
.....
.....
.....
.....
.....
```

.....

.....

.....

```
            //afficher le message Etudiants triés selon l'âge :
```

.....

.....

.....

```
            //appeler la méthode afficher pour le tableau tab
```

.....

.....

.....

```
            //appeler la méthode trier en précisant comme premier
            //argument (effectif) le tableau tab et comme deuxième
            //argument (effectif) une expression lambda qui
            //(re)définit la méthode fonctionnelle pour que la
            //comparaison se fasse selon le nom
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
            //Afficher le message Etudiants triés selon le nom :
```

.....

.....

.....

```
            //Appeler la méthode afficher pour le tableau tab
```

.....

.....

.....

```
      }//fin de la méthode main
}//fin de la classe principale
```

Question 2

Soit le fichier ***CP_Ctr3Exo2.java*** contenant le code source suivant :

```
package cms_ctr3;

public class CP_Ctr3Exo2
{
	static String interpreter(double imc) throws SurpoidsExp
	{
		try {
			if(imc < 18.5) throw new MaigreurExp(imc,
					"Vous êtes trop maigre !");
			if(imc > 25) throw new SurpoidsExp(imc);
		}catch(MaigreurExp me) {
			System.out.println(me.getMessage());
			if(imc > 16.5) return "Il faut manger plus !";
			throw me;
		}finally {
			System.out.println("Mangez au moins 5 fruits " +
					"ou légumes par jour !");
		}
		System.out.println("Faire du sport est une bonne idée !");
		return "Votre état est parfait !";
	}//fin de la méthode interpreter

	public static void main(String[] args)
	{
		double[] indices = {16, 22, 35, 17, 29};
		for(int i=indices.length-1; i>=0; i--) {
			try {
				System.out.println(interpreter(indices[i]));
			}catch(MaigreurExp me) {
				System.out.println("La valeur " + me.imc +
					" est trop basse !");
				System.out.println("Vous devez aller voir " +
					"un nutritionniste !");
			}catch(SurpoidsExp se) {
				System.out.println(se.getMessage());
```

```
                    if(se.imc < 30) {
                        System.out.println("Mais pas de panique "
                                + "car ça va encore !");
                    }else {
                        System.out.println("Il faut vraiment " +
                                "faire attention !");
                    }
                }
                System.out.println("*****************************");
            }
        }//fin de la méthode main
}//fin de la classe principale

class SurpoidsExp extends Exception
{
    double imc;
    SurpoidsExp(double imc)
    {
        super("L'IMC " + imc + " est trop élevé !");
        System.out.println("Une simple exception !");
        this.imc = imc;
    }
}//fin de la classe SurpoidsExp

class MaigreurExp extends RuntimeException
{
    double imc ;
    MaigreurExp(double imc, String message)
    {
        super(message);
        this.imc = imc;
        System.out.println("Une exception spéciale !");
    }
}//fin de la classe MaigreurExp
```

Précisez à la page suivante les messages qui seront affichés à l'écran suite à l'exécution du projet correspondant au code source indiqué ci-dessus.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Question 3

Toto est un programmeur Java qui aimerait éviter de se faire voler les coordonnées de ses comptes bancaires. Pour cette raison, à l'aide d'un programme qui appelle plusieurs fois la méthode ***writeLong*** de la classe ***DataOutputStream***, il a enregistré dans un fichier (purement) binaire les numéros de ses comptes à la suite, mais dans un certain ordre (et sans aucun autre détail les concernant). De plus, Toto a enregistré dans un autre fichier, cette fois texte, les informations concernant les mêmes comptes, en remplaçant systématiquement chaque numéro de compte par la chaîne de caractères "spéciale" `/*****/` et en respectant le même ordre que celui utilisé pour écrire le fichier binaire.

Votre tâche est d'écrire une méthode nommée ***resoudre*** qui lit deux fichiers comme ceux mentionnés ci-dessus et écrit à l'écran (dans la fenêtre console) les lignes du fichier texte mais en remplaçant chaque occurrence de la chaîne de caractères "spéciale" `/*****/` par le bon numéro de compte.

Par exemple, supposons que dans le fichier binaire on a écrit, à la suite, les numéros de compte `10000, 11000, 12000, 13000 et 14000`, et que dans le fichier texte on a écrit les 6 lignes suivantes :

```
Ci-dessous sont tous mes comptes.
/*****/ est mon compte principal en Suisse,
tandis que mon compte de secours /*****/
est au Lichtenstein.
Les 3 autres comptes, à savoir /*****/, /*****/ et /*****/, sont
dans les Iles Caïman, dans les Iles Vierges et à Monaco.
```

Suite à l'appel de la méthode que vous allez écrire, on pourra voir dans la fenêtre console les 6 lignes suivantes :

```
Ci-dessous sont tous mes comptes.
10000 est mon compte principal en Suisse,
tandis que mon compte de secours 11000
est au Lichtenstein.
Les 3 autres comptes, à savoir 12000, 13000 et 14000, sont
dans les Iles Caïman, dans les Iles Vierges et à Monaco.
```

Plus précisément, vous devez écrire le code complet d'une classe publique nommée ***Service***, qui fait partie d'un package nommée ***cms_ctr3*** et qui contient une seule méthode publique, statique et nommée ***resoudre***. Cette méthode ne retourne aucun résultat et doit avoir deux arguments de type chaîne de caractères nommés ***sourceBin*** et ***sourceTexte*** et qui correspondent au fichier binaire et, respectivement, au ficher texte mentionnés auparavant. Vous ne connaissez pas le contenu du fichier texte (en particulier, vous ne savez pas combien de lignes il contient),

mais vous savez que le fichier binaire contient un nombre suffisant de valeurs dans le bon ordre (et il ne faut donc pas traiter à part le cas où on arrive à la fin du fichier binaire avant la fin du fichier texte).

Dans le corps de la méthode ***resoudre*** (qui ne traite localement aucune exception), vous devez notamment prévoir :

- une variable adéquate pour lire le fichier binaire ;
- une variable adéquate pour lire le fichier texte ;
- d'autres variables utiles par la suite ;
- une boucle appropriée pour lire (complétement) le fichier texte ligne par ligne ;
- d'autres dispositions pour bien finir le travail avec les fichiers.

Pour chaque ligne lue, vous devez :

- découper convenablement la ligne en tokens (par une tokenisation adéquate) ;
- prévoir une boucle appropriée pour parcourir (tous) les tokens.

Pour chaque token, en fonction de sa valeur, vous pouvez procéder ainsi :

- si le token provient de la chaîne de caractères "spéciale" mentionnée auparavant, on affiche dans la fenêtre console le numéro de compte bancaire correspondant qui sera lu dans le fichier binaire ;
- autrement (c'est-à-dire si le token correspond à du texte "normal"), on affiche dans la fenêtre console le token tel quel.

Il conviendra de faire attention à l'utilisation judicieuse des méthodes ***print*** et ***println*** afin d'obtenir à l'écran l'affichage demandé (conformément à l'exemple de la page précédente).

Ecrivez ci-dessous le code complet de la classe ***Services***.

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

A part ce texte, cette page doit rester normalement blanche (mais vous pouvez l'utiliser pour la rédaction de vos réponses si la place prévue à cet effet s'est avérée insuffisante).