IPST-CNAM
Programmation JAVA
NFA 032
Mercredi 26 Juin 2013

Avec document
Durée : **2 h30**
Enseignant : LAFORGUE Jacques

1ère Session NFA 032

CORRECTION

L'examen se déroule en deux parties. Une première partie de 1h15mn, sans document, consacrée à des questions de cours, et une deuxième partie de 1h 15mn, avec document, consacrée en la réalisation de programmes Java.

Au bout de 1h15mn, les copies de la première partie seront ramassées avant de commencer la deuxième partie.

Pour la première partie, vous devez rendre le QCM rempli et les réponses aux questions libres écrites sur des copies vierges.

Pour la deuxième partie, vous écrivez vos programmes sur des copies vierges. Vous devez écrire les codes commentés en Java.

1ère PARTIE : COURS (sans document)

1. QCM (35 points)

Mode d'emploi :

Ce sujet est un QCM dont les questions sont de 3 natures :

- **les questions à 2 propositions**: dans ce cas une seule des 2 propositions est bonne.
  - +1 pour la réponse bonne
  - –1 pour la réponse fausse
- **les questions à 3 propositions** dont 1 seule proposition est bonne
  - + 1 pour la réponse bonne
  - –½ pour chaque réponse fausse
- **les questions à 3 propositions** dont 1 seule proposition est fausse
  - + ½ pour chaque réponse bonne
  - –1 pour la réponse fausse

Il s'agit de faire une croix dans les cases de droite en face des propositions.

On peut remarquer que cocher toutes les propositions d'une question revient à ne rien cocher du tout (égal à 0).

Si vous devez raturer une croix, faites-le correctement afin qu'il n'y ait aucune ambiguïté.

N'oubliez pas d'inscrire en en-tête du QCM, votre nom et prénom.

Vous avez droit à **4 points** négatifs sans pénalité.

NOM:	PRENOM:

En Java, la classe Hashtable est une classe de définition d'une Collection qui permet l'accès à ses éléments via une chaine de caractères		Q 1
1	OUI	**X**
2	NON	

En Java, la classe Collections permet de trier les éléments de n'importe quelle collection (classe qui implémente l'interface List) grâce à la méthode ***sort***. Cette méthode fonctionne si la classe d'appartenance des éléments de la collection implémente l'interface :		Q 2
1	Comparator	
2	Comparable	**X**
3	Comparer	

En java, la notion d'interface est un moyen de conception des interfaces homme machine		Q 3
1	OUI	
2	NON	**X**

Une classe abstraite est une classe dans laquelle toutes les méthodes sont abstraites (une méthode abstraite est une méthode sans code) et n'a pas de constructeur.		Q 4
1	OUI	
2	NON	**X**

Soit trois classes A, B et C qui héritent de la classe abstraite H et H implémente l'interface I. Pour créer une collection polymorphe, on peut faire :		Q 5
1	ArrayList<I> elements;	**X**
2	ArrayList<H> elements;	**X**
3	ArrayList<A,B,C> elements	

Soit les classes A et B qui héritent de C. Les classes A, B et C ne sont pas abstraites.
Soit la classe Stock contenant l'attribut : ArrayList<C> elements;
On peut écrire :
`elements.add( new A() )`
`elements.add( new B() )`
		Q 6

1	OUI	**X**
2	NON	

Une collection polymorphe est une collection qui peut être utilisée sous différentes formes : List, Set, Array, LinkedList, …		Q 7
1	OUI	
2	NON	**X**

On a le code suivant : Q 8

```
File fichier = new File("ListeDouble.bin");
FileOutputStream fos = new FileOutputStream(fichier);
DataOutputStream dos = new DataOutputStream(fos);
dos.writeInt(tab.length);
for(int i=0;i< tab.length;i++) dos.writeDouble( tab[i]);
dos.close();
```

1	Ce code crée un fichier de nom 'ListeDouble.bin' contenant la liste de doubles	**X**
2	Ce code crée un fichier dont les informations sont dans un format texte	
3	Ce code crée un fichier dont les informations sont dans un format binaire	**X**

La sérialisation est un principe natif du langage JAVA permettant de plier et déplier les attributs des objets afin de pouvoir transporter les objets via un fichier ou un socket Q 9

1	OUI	**X**
2	NON	

Soit le code JAVA suivant : Q 10

```
ArrayList<String> liste = new ArrayList<String>();
Collections.addAll("1","3","3","1","4","5","4");
liste = new ArrayList<String>( new HashSet<String>(liste) );
```

1	La dernière ligne de ce code n'a pas de sens	
2	La dernière ligne de ce code convertit un ensemble en une liste	
3	La dernière ligne de ce code enlève de "liste" tous les éléments redondants	**X**

En JAVA, une interface est un moyen de créer des traitements génériques Q 11

1	OUI	**X**
2	NON	

La déclaration de la méthode suivante : Q 12

```
public void traitement(String s) throws MyException
```

précise que la méthode doit capturer l'exception MyException dans le corps de sa méthode.

1	OUI	
2	NON	**X**

Soit le code JAVA suivant : Q 13

```
public void traitement(String nom) throws Exception
{
   Individu ind=null;
   try {
       ind = rechercher(nom);
       System.out.println(ind.toString());
   }
   catch(NonTrouveException ex) {
      throws new Exception("non trouve"); }
}
avec :
    la     méthode     'rechercher'     retourne     l'exception
'NonTrouveException' si le nom de l'individu n'est pas trouvé.
```

Alors :

1	si l'individu que l'on veut ajouter n'est pas trouvé alors la méthode retourne l'exception NonTrouveException	
2	si l'individu que l'on veut ajouter n'est pas trouvé alors la méthode retourne l'exception Exception	**X**
3	Si l'individu que l'on veut ajouter n'est pas trouvé alors la méthode ne retourne pas d'exception	

L'exception JAVA RuntimeException est une exception qui est retournée quand le moteur de la JVM (Runtime) plante		Q 14
1	OUI	
2	NON	X

Soit le code JAVA suivant :
`public void traitement()`
`{`
`  try{`
`  faire();`
`  }catch(Exception ex){ throws new RuntimeException("erreur"); }`
avec la méthode "faire" qui déclenche l'exception "MyException".
Alors :
		Q 15

1	ce code ne se compile pas correctement	
2	la méthode "traitement" retourne une exception	X
3	il manque dans l'en tête de la méthode la mention : "throws RuntimeException"	

En JAVA, l'instruction "accept" de ServerSocket tel que :
`ServerSocket ssoc = new ServerSocket(9999);`
`Socket soc = ssoc.accept();`
		Q 16

1	prévient le client qu'il peut envoyer ses informations au serveur	
2	se met en attente jusqu'à ce qu'un client crée un socket sur le port 9999	X

Le socket est un moyen d'échanger des informations entre deux programmes qui ne sont pas écrits dans un même langage		Q 17
1	OUI	X
2	NON	

Le code JAVA suivant calcule la somme des N premiers nombres entiers positifs récursivement :
`static int somme(int n)`
`  {`
`  return n + somme(n-1);`
`  }`
C e code est correct :
		Q 18

1	OUI	
2	NON	X

Le code JAVA suivant, liste les fichiers et les répertoires qui se trouvent sous le répertoire "/home/jl"
`File varfile;`
`varfile = new File("/home/jl");`
`for(String nom : varfile.list())`
`  System.out.println(nom);`
		Q 19

1	OUI	X
2	NON	

Les classes ArrayOutputSream et ArrayInputStream permettent d'écrire et lire dans les fichiers des tableaux contenant n'importe quel objet		Q 20
1	OUI	
2	NON	X

En JAVA, pour faire la sauvegarde d'un ensemble de données, on utilise, notamment, la classe DataOutputStream qui permet d'écrire les attributs des objets de type élémentaire (chaine, entier, double, …)	Q 21
1 OUI	X
2 NON	

La class File ne gère que les fichiers. Pour gérer des répertoires on utilise la classe FileDirectory	Q 22
1 OUI	
2 NON	X

En JAVA, un moyen pour sérialiser les objets de nos propres classes, est que chaque classe utilisée dans la composition des objets, implémente l'interface prédéfinie 'Serializable'	Q 23
1 OUI	X
2 NON	

Le code suivant permet de lire une chaine au clavier :
`BufferedReader in =`
`new BufferedReader(new InputStreamReader(System.in));`
`String str = in.readLine();`
	Q 24

1 OUI	X
2 NON	

En JAVA, créer un thread consiste à implémenter la méthode void run() dans une classe qui hérite de Thread	Q 25
1 OUI	X
2 NON	

Deux threads peuvent exécuter en même temps la même méthode d'un même objet et donc, par cela, modifier en même temps les mêmes attributs de l'objet	Q 26
1 OUI	X
2 NON	

Soit deux thread t1 et t2 qui utilisent la méthode définie ainsi `synchronized public void miseajour(){…` Quand t1 est en train d'exécuter cette méthode alors t2 (qui veut aussi l'exécuter) est en attente que t1 ait fini de l'exécuter	Q 27
1 OUI	X
2 NON	

Dans un appel récursif seul les paramètres d'appel de la méthode récursive sont empilés sur la pile d'exécution	Q 28
1 OUI	
2 NON	X

Le code JAVA suivant définit la structure d'une liste chainée :		Q 29

```
class Liste
{
    private int nb;         // Nombre d'élément de la liste
    private Cellule prem;   // Premier élément de la liste
    }
class Cellule
{
    Object  value;    // Valeur de l'élément
    Cellule suiv;     // Element suivant
}
```

1	OUI	**X**
2	NON	

La classe ArrayList est une structure de données récursive		Q 30
1	OUI	**X**
2	NON	

Soit le code suivant : Q 31

```
try{
   System.out.println("AAA");
   call();
   System.out.println("BBB");
}
catch(MyException ex) {
   System.out.println("DDD");
}
catch(Exception ex) {
   System.out.println("CCC");
}
```

avec la méthode call qui déclenche l'exception **MyException**.

Ce code affiche :

1	AAA CCC	
2	AAA DDD CCC	
3	AAA DDD	**X**

La méthode "start" est la méthode qui permet de démarrer un thread		Q 32
1	Cette méthode doit être implémentée dans la classe qui hérite de Thread	
2	Cette méthode appartient à la classe Thread. On y accède par héritage de la classe Thread	**X**
3	Cette méthode appelle la méthode run() que nous avons écrit dans notre thread	**X**

Soit une collection "liste" définie par la classe ArrayList<Individu>. Nous voulons trier les éléments de cette liste suivant 3 critères de tri différents. Pour cela :		Q 33
1	la classe Individu implémente l'interface Comparable et on code dans la méthode **compareTo** les 3 critères de tri. Le choix du tri se fait par la valeur d'un attribut statique	**X**
2	la classe Individu implémente 3 fois l'interface Comparable	
3	pour chacun des tris faire les appels : Collections.sort(liste, comparator1) Collections.sort(liste, comparator2) ou Collections.sort(liste, comparator3) où comparator1, comparator2, comparator3 sont des instances des classe Comparator1, Comparator2, Comparator3 qui implémentent la méthode **compare** de l'interface Comparator<Individu>	**X**

Soit une classe MyThread qui hérite de Thread et qui contient un attribut défini comme suit : `private static int tempo;` Tous les threads, instance de MyThread, ont tous la même valeur tempo (exemple: temps de temporisation des threads)		Q 34
1	OUI	**X**
2	NON	

Soit deux classes B et C qui héritent d'une classe abstraite A. Les classes B et C peuvent surcharger les méthodes publiques non abstraites de la classe A.		Q 35
1	OUI	**X**
2	NON	

2. Questions libres (15 points)

Chaque question est notée sur 5 points.
*Vous répondez à ces questions sur une **copie vierge** en mettant bien le numéro de la question, sans oublier votre nom et prénom.*

Q 1

Précisez le rôle des 3 instructions JAVA suivantes :

```
... throws MyException
try{...}catch(MyException ex){...} catch(Exception ex){...}
throw new MyException()
```

L'instruction "throws MyException" est utilisée dans l'en-tête d'une méthode pour préciser que la méthode peut retourner l'exception 'MyException'
L'instruction "try catch ..." est utilisée pour une exception qui se déclencherait. Si l'exception 'MyException' est déclenchée alors on exécute le code du 1er catch. Pour toute autre exception, on exécute le code du 2ème catch.
L'instruction 'throw new MyException' est utilisée pour déclencher l'exception 'MyException'.

Q 2

Donnez 2 exemples (principes différents) de l'utilisation d'une interface JAVA. Commentez.

Exemple 1:
ArrayList<ElementGeometrique> elements;
Dans la déclaration d'une collection Polymorphe pour laquelle tous les éléments de classes différentes sont vus comme une interface. Toutes ces classes implémentent l'interface

Exemple 2:
public Collections sort(List l,Comparator comp)
Pour trier génériquement les éléments d'une collection. On passe en paramètre de la méthode sort un objet qui implémente l'interface Comparator (contenant la méthode Compare).

Q 3

A quoi servent les classes ObjectInputStream et ObjectOutputStream ?
Expliquez.

Ces deux classes sont utilisées pour écrire et lire des objets (en une seule instruction) dans un fichier ou un socket.
Toute l'arborescence des attributs de l'objet est parcourue par Java et tous les attributs sont écrits puis reconstitués à la lecture. On appelle cela la sérialisation.

(Tourner la page)

2ème PARTIE : PROGRAMMATION (avec document)

Problème 1 [15 points]

```
import java.util.*;

public class Exercice1
{
   // Programme principal
   public static void main(String... args)
   {
      // Parametre du programme
      int maxAleat = 10; // Par défaut
      try{
         maxAleat = Integer.parseInt(args[0]);
         if (maxAleat>100)
            {
               maxAleat = 10;
               throw new Exception();
            }
      }catch(Exception ex){
         System.out.println("Erreur parametre : "+args[0]);}

      // Tableau et compteur
      int[] tab = new int[2];
      int compteur = 0;

      //Creation et demarrage des 2 threads
      AleatThread t1 = new AleatThread(0,tab,maxAleat);
      AleatThread t2 = new AleatThread(1,tab,maxAleat);
      t1.start();
      t2.start();

      while(true)
         {
            try{Thread.sleep(100);}catch(Exception ex){}
            if (tab[0]==tab[1])
               {
                  compteur++;
                  System.out.print(tab[0]+"-"+compteur+" ");
               }
         }
   }
}

// La classe de definition d'un thread
class AleatThread extends Thread
{
   // Indice du tableau dans lequel il stocke son tirage
   private int   indice;
   private int[] tab;      // Le tableau
   // Valeur max aleatoire
   private int   maxAleat;
```

```
    // Constructeur
    public AleatThread(int indice,int[] tab,int maxAleat)
    {
       this.indice   = indice;
       this.tab      = tab;
       this.maxAleat = maxAleat;
    }

    // Traitement du thread
    public void run()
    {
       while(true)
          {
             try{Thread.sleep(maxAleat);}catch(Exception ex){};
             // Tirage aleatoire
             Random rnd = new Random();
             // Affectation du tableau
             tab[indice]=rnd.nextInt(maxAleat)+1;
          }
    }
}
```

Problème 2 [15 points]

```
import java.util.*;

public class Exercice2
{
   // Programme principal
   public static void main(String... args)
   {
      // Creation de la collection
      ArrayList<Individu> liste = new ArrayList<Individu>();
      liste.add(new Individu("LAFONT","Pierre",date(31,10,1960)));
      liste.add(new Individu("ABBE","Paul",date(31,1,1980)));
      liste.add(new Individu("ZOE","Christelle",date(10,11,1972)));
      liste.add(new Individu("MARTIN","Yves",date(3,3,1983)));
      liste.add(new Individu("MARTIN","Andre",date(3,3,1993)));
      liste.add(new Individu("GONTRAN","Albert",date(5,12,2000)));

      // Tri par nom et prenom
      System.out.println("---------- Tri par nom");
      TrierNomPrenom tri1 = new TrierNomPrenom();
      Collections.sort(liste,tri1);
      for(Individu i:liste)System.out.println(i.toString());

      // Tri par date de naissance
      System.out.println("---------- Tri par date de naissance");
      TrierDateNaissance tri2 = new TrierDateNaissance();
      Collections.sort(liste,tri2);
      for(Individu i:liste)System.out.println(i.toString());
   }
```

```
        // Date JJ/MM/AA en chaine
        public static String dateToString(Calendar date)
        {
            int month = date.get(Calendar.MONTH)+1;
            return
date.get(Calendar.DAY_OF_MONTH)+"/"+month+"/"+date.get(Calendar.YEAR);
        }

         // Creer une nouvelle date
        private static Calendar date(int jour,int mois,int annee)
        {
            Calendar date = Calendar.getInstance();
            date.set(annee,mois-1,jour);
            return date;
        }
    }

    // Classe de definition d'un individu
    class Individu
    {
        public String nom;
        public String prenom;
        public Calendar dateNaissance;

        // Constructeur
        public Individu(String nom,String prenom,Calendar dateNaissance)
        {
            this.nom=nom;
            this.prenom=prenom;
            this.dateNaissance=dateNaissance;
        }

        // Individu en chaine
        public String toString()
        {
            return nom+" "+prenom+" "+Exercice2.dateToString(dateNaissance);
        }
    }

    // Classe de tri suivant le nom de l'individu
    class TrierNomPrenom implements Comparator<Individu>
    {
        public int compare(Individu a,Individu b)
        {
            if(a.nom.compareTo(b.nom)<0)
                return -1;
            else
                if (a.nom.compareTo(b.nom)>0)
                    return 1;
                else
                    if (a.prenom.compareTo(b.prenom)<0)
                        return -1;
                    else if (a.prenom.compareTo(b.prenom)>0)
                        return 1;
```

```
            else
                return 0;
    }
}

// Classe de tri suivant la date de naissance
class TrierDateNaissance implements Comparator<Individu>
{
    public int compare(Individu a,Individu b)
    {
        if(a.dateNaissance.compareTo(b.dateNaissance)<0)
            return -1;
        else
            if (a.dateNaissance.compareTo(b.dateNaissance)>0)
                return 1;
            else
                return 0;
    }
}
```

(Fin du sujet)