IPST-CNAM
Programmation JAVA
NFA 002
Mercredi 27 Juin 2012

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

CORRECTION

2ème Session NFA 002

1. QCM (35 points)

En Java, la classe Hashtable<K,V> est une classe de définition d'une collection d'élément, dont les éléments sont de type V et l'indice d'accès est une entier K qui va de 0 à size()-1		Q 1
1	OUI	
2	NON	X

En Java, la classe Collections permet de trier les éléments de n'importe quelle collection (classe qui implémente l'interface **List<T>**) grâce à la méthode **Collections.*sort(List<T>)*** et si la classe d'appartenance des éléments de la collection implémente l'interface **Comparable**		Q 2
1	OUI	X
2	NON	

JAVA est un langage orientée objet dont les fichiers sources sont directement interprétables		Q 3
1	OUI	
2	NON	X

Une classe abstraite est une classe comme les autres et qui peut contenir des méthodes abstraites (sans code). Dans ce cas il est à la charge des classes dérivées d'implémenter ces méthodes abstraites		Q 4
1	OUI	X
2	NON	

Soit le code correct suivant : Q 5

```
public class A extends B
{
   private int attr_A;
   public A() {
      attr_A = 10;
      attr_B = "TOTO";
      C.attr_C = 100; }
}
```

1	attr_B peut être un attribut privé de B	
2	attr_C peut être un attribut statique et publique de C	X
3	attr_C peut être un attribut statique et privé de C	

Soit deux classes B et C qui héritent d'une classe abstraite A. Les classes B et C peuvent utiliser par héritage les méthodes publiques non abstraites de la classe A.		Q 6
1	OUI	X
2	NON	

Une classe qui contient au moins une méthode abstraite doit être déclarée abstraite.		Q 7
1	OUI	X
2	NON	

En Java, on déclare un tableau de la manière suivante T[] tab = new T[100].		Q 8
1	T peut être une classe abstraite	X
2	T peut être une interface	X
3	T ne peut pas être de type primitif	

En JAVA, si une classe C implémente plusieurs interfaces I1, I2, I3		Q 9
1	La classe C ou les classes dérivées de C doivent implémenter toutes les méthodes de toutes les interfaces I1, I2, I3	X
2	La classe C ou les classes dérivées de C doivent implémenter toutes les méthodes de I1 mais pas de I2 et I3	

En JAVA, une interface peut contenir des attributs qui sont statiques et publiques		Q 10
1	OUI	X
2	NON	

En JAVA, une interface permet de :		Q 11
1	créer une méthode générique dont le traitement utilise des méthodes de l'interface	X
2	créer des classes abstraites	

La déclaration d'une méthode suivante : `public void traitement(String s) throws MyException` précise que la méthode propage l'exception MyException.		Q 12
1	OUI	X
2	NON	

Soit le code suivant qui ajoute un Individu dans un tableau : Q 13

```
public void ajouter(String nom) throws Exception
{
   Individu ind=null;
   try {
      ind = rechercher(nom);
   }
   catch(NonTrouveException ex) {
      tab[n++] = ind; }
}

La méthode rechercher retourne l'exception NonTrouveException si
le nom de l'individu n'est pas trouvé.
```

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 et le tableau n'est pas plein alors l'individu est ajouté au tableau	X
3	Si l'individu que l'on veut ajouter n'est pas trouvé et le tableau est plein alors la méthode retourne une exception prédéfinie Java (IndexOutOfBoundsException)	X

Soit le code suivant : Q 14

```
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 ***UneAutreException.***

Ce code affiche :

1	AAA CCC	X
2	AAA	
3	AAA DDD	

En JAVA, un package est une classe qui hérite de Package et permet de créer une collection à travers les méthode pack() et unpack()	Q 15
1 OUI	
2 NON	X

En JAVA le "package" est un design pattern qui modélise les principes de compression et dedécompression des objets dans un socket (pliage/dépliage ou pack/unpack)	Q 16
1 OUI	
2 NON	X

Le design pattern Factory est un modèle de conception de la mise en œuvre d'une communication client serveur	Q 17
1 OUI	
2 NON	X

Le Singleton est un design pattern qui est une classe contenant la méthode unique **public T getInstance().** A chaque appel cette méthode retourne un objet unique de type T	Q 18
1 OUI	X
2 NON	

Le code suivant crée un descripteur de fichier dont le nom est "exemple.txt" permettant par exemple de tester si le fichier existe Q 19

```
File fichier;
fichier = new File("exemple.txt");
```

1 OUI	X
2 NON	

La classe File permet de parcourir les fichiers d'un répertoire	Q 20
1 OUI	X
2 NON	

On a le code suivant :		Q 21

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

La sérialisation est un service du langage Java qui permet d'écrire et de lire n'importe quel objet dans un fichier binaire (Serializable) ou un fichier texte (XML)		Q 23
1	OUI	X
2	NON	

Le code suivant est correct :		Q 24

	`File f = new File("/etc/passwd");`
	`System.out.println(f.exists());`
	`System.out.println(f.canRead());`
	`System.out.println(f.canWrite());`
	`System.out.println(f.getLength());`
	
	`File d = new File("/etc/");`
	`System.out.println(d.isDirectory());`
	
	`String[] files = d.list();`
	`for(int i=0; i < files.length; i++)`
	`  System.out.println(files[i]);`	

1	OUI	X
2	NON	

Le package est une unité de programmation permettant de regrouper et architecturer les classes du langage Java (prédéfinies ou développées) dans des répertoires et accessibles aux autres unités de programmation (programme Java, Applet, …).		Q 25
1	OUI	X
2	NON	

Un package est un répertoire qui ne peut pas contenir d'autres répertoires.		Q 26
1	OUI	
2	NON	X

L'option **–d** de la commande **javac** permet de générer les fichiers .class dans un autre répertoire que celui contenant les fichiers .java.		Q 27
1	OUI	X
2	NON	

En JAVA, un thread est une JVM qui s'exécute en parallèle de la JVM dans laquelle le thread a été créé.		Q 28
1	OUI	
2	NON	X

Soit le thread définit de la manière suivante :
`public class MonThread extends Thread {.... . }`
`et MonThread t = new MonThread();`
 	
On démarre le thread t de la manière suivante :
		Q 29

1	t.run()	
2	t.start()	X
3	new Thread(t).start()	

La classe A hérite de B qui hérite de C. C est une classe abstraite qui implémente une interface I. A et B ne sont pas des classes abstraites		Q 30
1	C peut implémenter une partie des méthodes de l'interface I	X
2	A et B doivent à elles deux implémenter toutes les méthodes de I qui n'ont pas été implémentées par C	X
3	A ne peut pas implémenter des méthodes de l'interface I	

Soit le code suivant :
`class MonRunnable extends Toto implements Runnable {`
`public void run() { // traitement } }`
 	
`Pour créer et démarrer le thread :`
			Q 31

1		`new Thread(new MonRunnable ()).start();`	X
2		`MonRunnable p = (MonRunnable)(new Thread());` `p.start()`	

3		`MonRunnable p = new MonRunnable ();`
		`Thread q = new Thread(p);`
		`q.start();`
			X

Un tableau Java peut contenir des type primitifs.		Q 32
1	OUI	X
2	NON	

Une collection polymorphe est une instance d'une classe P<T> contenant des objets dont les classes d'appartenance peuvent être différentes et toutes ces classes héritent de la classe abstraite T:		Q 33
1	OUI	X
2	NON	

En JAVA, la classe RuntimeException qui hérite de la classe Exception permet de déclencher une exception pour laquelle il n'est pas utile de déclarer comme devant être propagé		Q 34
1	OUI	X
2	NON	

La sérialisation est une interface prédéfinie de JAVA qui permet d'écrire et lire les objets dans un fichier		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

Expliquez à quoi sert l'interface.

En JAVA, une interface est utilisée pour définir des traitements génériques ou des collections polymrphes. Une interface contient des méthodes abstraites qui sont implémentées par les classes qui implémente l'interface. Les instances de ces classes sont alors utilisées en paramètres de traitement, de constructeur ou comme type d'élément de collection.

Q 2

Quels sont les deux moyens permettant de créer un thread ?
Pourquoi utiliser l'un ou l'autre ?

Un thread est soit une classe qui hérite de la classe prédéfinie Thread, soit une classe qui implémente l'interface Runnable.
On est obligé d'utiliser le deuxième cas quand la classe hérite déjà d'une autre classe car JAVA ne permet pas de faire de l'héritage multiple.

Q 3

A quoi est utilisé le design paterne factory ?
Quel est son principe fondamental ?

Le design pattern Factory est utilisé pour créer des objets à la demande. Ces objets sont créés en interne du Factory par instanciation de différentes classes. Son principe fondamental étant que le Factory retourne une interface unique de manipulation des objets créés afin de rendre transparent l'usage des classes de création. De plus le factory permet de gérer les objets créés dans une collection et définit des traitements sur cette collection (recherche, filtre, …)

(Tourner la page)

2ème PARTIE : PROGRAMMATION (avec document)

Probleme [30 points]

```
// Classe de definition du progemme principal qui teste l'agenda
//
public class Probleme
{
    public static void main(String args[])
    {
        // Création d'un agenda
        //
        Agenda agenda = new Agenda();

        // Creation de plusieurs rendez-vous, de classe différente et
tous ces rdv sont
        // ajoutés dans l'agenda
        //

        RdvJournalier rdv1 = new RdvJournalier("2012-09-01",
                                               "9h30",
                                               "10h00",
                                               "Passer prendre Jean à
l'aéroport");

        RdvPeriodique rdv2 = new RdvPeriodique("2012-08-01",
                                               "2012-12-01",
                                               "HEBDO",
                                               "10h30",
                                               "12h30",
                                               "Point sur le
développement");

        RdvPeriodique rdv3 = new RdvPeriodique("2012-07-01",
                                               "2012-12-01",
                                               "MENSUEL",
                                               "10h30",
                                               "12h30",
                                               "Point sur le
développement");

        // Parceque tous ces rdv héritent de la classe abstraite
RendezVous, on peut
        // utiliser la même méthode pour les ajouter dans l'agenda
        agenda.ajouter(rdv1);
        agenda.ajouter(rdv2);
        agenda.ajouter(rdv3);

        // Affichage de l'agenda
        agenda.afficher();
    }
}

import java.util.*;

// Un agenda est uen collection de rendez-vous.
// La classe RendezVous est une classe abstraite ce qui permet d'ajouter
//  des rdv de classes différentes du moment que ces classes héritent de
la
//  classe abstraite
//
public class Agenda
{
    // Collection polymorphe de rendez-vous
    public ArrayList<RendezVous> rdvs;
```

```
    // Constructeur qui crée la collection à vide
    public Agenda()
    {
        rdvs = new ArrayList<RendezVous>();
    }

    // Ajout d'un rendez-vous. Le rdv en parametre est une isntance de
n'importe quelle classe
    //  qui hériterait de RendezVous
    public void ajouter(RendezVous rdv)
    {
        rdvs.add(rdv);
    }

    // Affiche tous les rdv de la collection en appelant la méthode
    //    d'affichage de chacun des objets. En fonction de sa classe
d'apaprtenance
    //   chaque objet appellera la méthode afficher de sa classe.
    //
    public void afficher()
    {
        trier();
        for(RendezVous rdv : rdvs)
            {
                System.out.println("----------------------");
                rdv.afficher();
            }
    }

    // Méthode de tri des rdv de l'agenda
    //
    public void trier()
    {
        Collections.sort(rdvs,new ComparerRdv());
    }
}



// Classe de comparaison des rendez-vous utilisée pour trier
//  les rdv de l'agenda
//
class ComparerRdv implements Comparator<RendezVous>
{
    public int compare(RendezVous o1,RendezVous o2)
    {
        return( o1.getDateRdv().compareTo(o2.getDateRdv()) );
    }

}

// Classe commune de définition d'un rendezvous
// La classe est abstraite car la méthode afficher est abstraite
//
public abstract class RendezVous
{
    // Les attribust communs à tous les rendez-vous.
    // Tout rdv a une heure de début, de fin et un texte
    protected String heureDebut;
    protected String heureFin;
    protected String texte;

    // Cosntructeur
    public RendezVous(String heureDebut,
                      String heureFin,
                      String texte)
    {
        this.heureDebut = heureDebut;
        this.heureFin = heureFin;
```

```
        this.texte = texte;
    }

    // Ce sera aux classes réelles qui héritent de RendezVous, de faire
    //  l'affichage du rendez-vous.
    // C'est pour cela que les attributs de cette classe sont protected
(afin
    //  que les classes réelles accèdent aux attributs)
    abstract public void afficher();

    // Retourne la date du rdv
    //
    abstract public String getDateRdv();

}

// Classe de définition d'un rdv journalier
// La classe hérite de la classe abstraite RendezVous
//
public class RdvJournalier extends RendezVous
{
    // Date du rdv
    public String date;

    // Constructeur
    public RdvJournalier(String date,
                         String heureDebut,
                         String heureFin,
                         String texte)
    {
        // Initialisation des attribut de la classe héritée
        super(heureDebut,heureFin,texte);
        this.date = date;
    }

    // Affichage du rdv journalier
    public void afficher()
    {
        System.out.println("Date du rdv    : " + date);
        System.out.println("Heure de debut : " + heureDebut);
        System.out.println("Heure de fin   : " + heureFin);
        System.out.println("Texte          : " + texte);
    }

    // Retourne la date du rdv
    //
    public String getDateRdv()
    {
        return date;
    }


}

// Classe de définition d'un rdv periodique
// La classe hérite de la classe abstraite RendezVous
//
public class RdvPeriodique extends RendezVous
{
    // Date du premier rendez-vous
    public String dateDebut;
    // Date de fin de la periode
    public String dateFin;
    // Type de periode : MENSUEL ou HEBDO
    public String periode;

    // Constructeur
    public RdvPeriodique(String dateDebut,
                         String dateFin,
                         String periode,
```

```
                        String heureDebut,
                        String heureFin,
                        String texte)
    {
        super(heureDebut,heureFin,texte);

        this.dateDebut = dateDebut;
        this.dateFin = dateFin;
        this.periode = periode;
    }

    // Affichage du rendez-vous periodique
    public void afficher()
    {
        System.out.println("Date du premier rdv    : " + dateDebut);
        System.out.println("Heure de debut         : " + heureDebut);
        System.out.println("Heure de fin           : " + heureFin);
        System.out.println("Texte                  : " + texte);
        System.out.println("Periode                : " + periode);
    }

    // Retourne la date du rdv
    //
    public String getDateRdv()
    {
        return dateDebut;
    }
}
```

(Fin du sujet)