IPST-CNAM
Programmation JAVA
NFA 002
Mercredi 27 Juin 2012

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

CORRECTION

1ère Session NFA 002

# 1ère PARTIE : COURS (sans document)

## 1. QCM (35 points)

| En Java, la classe Hashtable<K,V> permet de gérer une collection d'éléments dont l'accès se fait par une donnée, appelée Key, et non par un rang comme cela est le cas dans la classe ArrayList | | 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***.<br>Cette méthode fonctionne si la classe d'appartenance des éléments de la collection implémente l'interface : | | Q 2 |
|---|---|---|
| 1 | Comparator | |
| 2 | Comparable | X |
| 3 | Compiler | |

| L'héritage multiple de classe est possible en Java | | Q 3 |
|---|---|---|
| 1 | OUI | |
| 2 | NON | X |

| Une classe abstraite est une classe dont les types des attributs ne sont pas connus. C'est lors de l'instanciation de l'objet que les types des attributs sont résolus | | Q 4 |
|---|---|---|
| 1 | OUI | |
| 2 | NON | X |

Soit le code suivant : Q 5

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

attr_B est un attribut protected de B
attr_C est un attribut static public de l'interface C
```

Ce code est correct

| | | |
|---|---|---|
| 1 | OUI | X |
| 2 | NON | |

| Soit deux classes B et C qui héritent d'une classe abstraite A.<br>Les classes B et C peuvent utiliser par héritage les attributs protected 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 qui contient des éléments dont la classe d'appartenance est une classe abstraite C. | | Q 8 |
|---|---|---|
| 1 | Cela n'est pas possible | |
| 2 | Cela est possible et on ajoute dans le tableau des objets de type C ( t[i]=new C(); ) | X |
| 3 | Cela est possible et on ajoute dans le tableau des objets de type B ( t[i]=new B(); ) et la classe B hérite de C | X |

| En JAVA, une classe peut implémenter plusieurs interfaces | | Q 9 |
|---|---|---|
| 1 | OUI | X |
| 2 | NON | |

| En JAVA, une interface peut contenir des attributs : | | Q 10 |
|---|---|---|
| 1 | statics | X |
| 2 | non statics | |
| 3 | privés | |

| En JAVA, une interface permet de : | | Q 11 |
|---|---|---|
| 1 | passer en paramètre d'une méthode un objet dont la classe d'appartenance implémente l'interface | X |
| 2 | définir des traitements génériques sur des collections polymorphes | X |
| 3 | créer des classes abstraites | |

| La déclaration d'une méthode suivante :<br>`public void traitement(String s) throws MyException`<br>précise que la méthode doit traiter l'exception MyException dans le corps de sa méthode. | | Q 12 |
|---|---|---|
| 1 | OUI | |
| 2 | NON | X |

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 ***MyException.***

Ce code affiche :

| | | |
|---|---|---|
| 1 | AAA<br>BBB | |
| 2 | AAA<br>DDD<br>CCC | |
| 3 | AAA<br>DDD | X |

En JAVA, un package est un regroupement de classe Java (.java et .class).
Ce regroupement (création du package) se fait avec la commande de compilation (option -d) :
**javac -d <repertoire de regroupement> *.java** — Q 15

| | | |
|---|---|---|
| 1 | OUI | |
| 2 | NON | X |

Un design pattern est un modèle de description qui représente souvent une architecture de classes et dont l'objectif est de rendre réutilisable des choix de conception — Q 16

| | | |
|---|---|---|
| 1 | OUI | X |
| 2 | NON | |

Le design pattern Factory est un modèle de conception de la mise en œuvre — Q 17

| | | |
|---|---|---|
| 1 | d'une communication client serveur | |
| 2 | d'une fabrique d'objets décrits par une interface commune | X |
| 3 | de la mise en facteur des méthodes de différentes classes dans une même interface | |

| Le Singleton est un design pattern qui | | Q 18 |
|---|---|---|
| 1 | est une classe contenant une méthode static qui crée un objet s'il n'est pas déjà créé et le retourne, sinon le retourne | X |
| 2 | de créer un nouvel objet à chaque fois que l'on appelle la méthode static getInstance mais chacun des objets implémente une interface unique | |
| 3 | permet d'obtenir un objet qui est toujours le même. Il est unique dans la JVM d'exécution. | X |

| Le code suivant crée un fichier de nom "exemple.txt" dans le répertoire courant d'exécution. Le fichier est créé, vide de toute information<br>`File fichier;`<br>`fichier = new File("exemple.txt");` | | Q 19 |
|---|---|---|
| 1 | OUI | |
| 2 | NON | X |

| En JAVA, la gestion des entrées sorties se fait notamment par les classes qui se trouvent dans le package java.io | | Q 20 |
|---|---|---|
| 1 | OUI | X |
| 2 | NON | |

| On a le code suivant :<br>`File fichier = new File("ListeDouble.bin);`<br>`FileOutputStream fos = new FileOutputStream(fichier);`<br>`DataOutputStream dos = new DataOutputStream(fos);`<br>`dos.writeInt(tab.length);`<br>`for(int i=0;i< tab.length;i++) dos.writeDouble( tab[i]);`<br>`dos.close();` | | Q 21 |
|---|---|---|
| 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 :<br>`import java.io.*;`<br><br>`public class Terminal{`<br>`  static BufferedReader in =`<br>`  new BufferedReader(new InputStreamReader(System.in));`<br><br>`  public static String lireString() throws IOException`<br>`  {`<br>`  return in.readLine();`<br>`  }`<br>`}` | | Q 24 |
|---|---|---|
| 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 répertoire de package est une chemin d'accès vers un répertoire qui contient les classes du package et ce répertoire ne peut pas contenir d'autres répertoires. | | Q 26 |
|---|---|---|
| 1 | OUI | |
| 2 | NON | X |

| Soit le fichier suivant : C:\CodeJava\exercices\cnam\util\Terminal.java.<br>Le fichier Terminal.java contient en 1ère ligne : package exercices.cnam.util<br>Le répertoire C:\bin est vide.<br>On réalise la commande suivante : javac -d C:\bin C:\CodeJava\exercices\cnam\util\Terminal.java<br>On obtient le résultat suivant :<br>C:\bin\ exercices\cnam\util\Terminal.class | | 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 |

| Pour créer et démarrer un thread, il faut : | | Q 29 |
|---|---|---|
| 1 | créer un objet dont la classe d'appartenance hérite de la classe Runnable et implémente la méthode run | |
| 2 | créer un objet dont la classe d'appartenance hérite de la classe Thread, implémente la méthode run et appeler la méthode start | X |

| La classe A hérite de B qui hérite de C. C est une classe abstraite qui implémente une interface I.<br>A et B ne sont pas des classes abstraites | | Q 30 |
|---|---|---|
| 1 | C peut implémenter qu'une partie des méthodes de l'interface I | X |
| 2 | B doit implémenter toues 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 | |

| L'interface permet de : | | Q 31 |
|---|---|---|
| 1 | gérer des collections polymorphes (les éléments sont de type d'une interface) | X |
| 2 | gérer les fichiers (interface avec le système d'exploitation) | |
| 3 | de passer en paramètre d'une méthode "un traitement" (traitements génériques) | X |

| Un tableau Java peut contenir des type primitifs, peut contenir des objets et la classe d'appartenance de ces objets peut être une classe abstraite | | Q 32 |
|---|---|---|
| 1 | OUI | X |
| 2 | NON | |

| Soit une collection "liste" définie par la classe ArrayList<Individu>. Nous proposons de vouloir trier les éléments de cette liste suivant 3 critères de tri différents. Pour réaliser ces 3 méthodes de tri différentes, il faut notamment : | | Q 33 |
|---|---|---|
| 1 | que la classe Individu implémente 3 interfaces différentes (1 interface par tri) | |
| 2 | créer 3 classes différentes Comparator1, Comparator2 et Comparator3 qui implémentent chacune l'interface Comparator<Individu> | X |
| 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 chacune l'interface Comparator<Individu> | X |

| En JAVA, la classe RuntimeException qui hérite de la classe Exception permet de déclencher une exception qui interrompt proprement et définitivement le runtime de la JVM. | | Q 34 |
|---|---|---|
| 1 | OUI | |
| 2 | NON | X |

| La sérialisation a un prix: la compatibilité binaire des informations écrites qui dépendent des versions JAVA et surtout de la stabilité des classes dont les objets sont écrits sur ce principe | | 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 servent les exceptions et expliquer son fonctionnement.

*Les exceptions servent à traiter les cas d'erreur proprement et efficacement.*
*Quand un développeur écrit une méthode, il doit prévoir quels sont les cas fonctionnels pour lesquels il lui est impossible de continuer le traitement. Il décide alors de remonter le cas d'erreur à l'appelant.*

*Les exceptions permettent d'interrompre l'exécution d'une méthode et de **remonter** à l'appelant le cas d'erreur sous la forme d'un objet dont la classe d'appartenance hérite de la classe prédéfinie Exception.*
*L'appelant peut alors **récupérer** cette exception et/ou la remonter à son tour à son appelant.*
*Quand une exception remonte de la méthode main alors le programme Java s'arrête.*

**Q 2**

Quels sont les composants (ou couches) d'une application Java basée sur les principes du modèle MVC.
Quel est le rôle de chacun de ces composants ?

*Les trois composants sont :*
*- le modèle de données*
*- les vues*
*- le contrôleur*

*Le modèle de donnée assure le stockage et le changement des données métiers de l'application. Il notifie ces changements aux vues de l'application.*
*Les vues assurent la représentation graphique des données métiers de l'application. Une vue s'abonne aux évènements de mise à jour des données et envoie des commandes au contrôleur.*
*Le contrôleur initialise les vues avec les données métiers, assure l'affichage des vues, l'enchaînement des vues entre elles et traite les commandes des vues qu'elle envoie au modèle.*

**Q 3**

En programmation objet, donner une définition du polymorphisme.
Quels sont les deux moyens, en JAVA, pour mettre en œuvre le polymorphisme ? Commentez.

*Le polymorphisme est une propriété des LOO qui permet de créer des collections contenant des objets de classes différentes.*
*Ces classes doivent alors toutes héritées d'une même classe abstraite ou implémenter la même interface.*
*Les éléments de la collection sont typés soit de la classe abstraite ou de l'interface.*
*Les méthodes qu'il est possible de faire sur les éléments de la collection sont soit les méthodes de la classe abstraite, soit les méthodes de l'interface.*

# 2ème PARTIE : PROGRAMMATION (avec document)

## Probleme [30 points]

```
import java.util.*;
import java.io.*;

// La classe de définition d'un agenda
public class Agenda
{
    // Le nom de fichier de sauvegarde
    private static final String nomFichierSauvegarde="AGENDA.txt";

    // La collection contenant les rendez-vous
    private ArrayList<RendezVous> rdvs;

    // Le constructeur :
    //   la collection est créée et  vide
    public Agenda()
    {
        rdvs = new ArrayList<RendezVous>();
    }

    // Ajoute un rendez-vous dans l'agenda puis trie les rendez-vous
    public void ajouter(RendezVous rdv)
    {
        rdvs.add(rdv);
        Collections.sort(rdvs);
    }

    // Affichage de l'agenda
    public void afficher()
    {
        for(RendezVous rdv : rdvs)
            {
                System.out.println(rdv.toString());
                System.out.println("------------------");
            }
    }

    // Méthode de sauvegarde de l'agenda
    // Le fichier contient dans la 1ère ligne le nombre de rendez-vous
    //  puis les rendez-vous un par un
    public void sauver()
    {
        try{
            FileOutputStream    fos    =    new    FileOutputStream(new
File(nomFichierSauvegarde));
            PrintStream flotOut = new PrintStream(fos);
            flotOut.println(rdvs.size()+"");
            for(RendezVous rdv : rdvs) rdv.ecrire(flotOut);
        }catch(Exception ex)
```

```
            {
                System.out.println("Probleme de sauvegarde : " + ex);
            }
    }

    // Méthode de chargement de l'agenda
    public void charger()
    {
        try{
            File file = new File(nomFichierSauvegarde);
            FileInputStream fis = new FileInputStream(file);
            BufferedReader    flotIn    =    new    BufferedReader(new
InputStreamReader(fis));
            String ligne1 = flotIn.readLine();
            int nbRdv = Integer.parseInt(ligne1);
            rdvs.clear();
            for(int i = 0; i <nbRdv; i++)
                {
                    RendezVous rdv = new RendezVous();
                    rdv.lire(flotIn);
                    rdvs.add(rdv);
                }
        }catch(Exception ex)
            {
                System.out.println("Probleme de chargement : " + ex);
            }
    }
}

import java.util.*;
import java.io.*;

// Classe de définition d'un rendez-vous
public class RendezVous implements Comparable<RendezVous>
{
    private String date;
    private String heureDebut;
    private String heureFin;
    private String texte;

    // Constructeur vide pour la lecture
    public RendezVous() {}

    // Constructeur de création d'un rendez-vous
    public RendezVous(String date, String heureDebut, String heureFin,
                      String texte)
    {
        this.date = date;
        this.heureDebut = heureDebut;
        this.heureFin = heureFin;
        this.texte = texte;
    }

    // Méthode de l'interface Comparable<RendezVous>
    //  afin de trier l'agenda
    //  1er critère de tri la date du rendez-vous
    //  2eme critère de tri l'heure de début du rendez-vous
    //
    public int compareTo(RendezVous rdv)
    {
        // 1er critere : la date
        int n = this.date.compareTo(rdv.date);
        if ( n==0) // Si même date
            {
                // 2eme critère : l'heure de début
                return this.heureDebut.compareTo(rdv.heureDebut);
            }
        else
            return n;
    }
```

```
    // Pour afficher à l'écran, le rendez-vous en chaine
    public String toString()
    {
        return (date + "/"+ heureDebut + " " + heureFin + "\n" +
                texte);
    }

    // Méthode qui lit un rendez-vous dans un stream de texte
    public void lire(BufferedReader flotIn) throws IOException
    {
        date = flotIn.readLine();
        heureDebut = flotIn.readLine();
        heureFin = flotIn.readLine();
        texte = flotIn.readLine();
    }

    // Méthode qui écrit un rednez-vous dans un stream de texte
    public void ecrire(PrintStream flotOut) throws IOException
    {
        flotOut.println(date);
        flotOut.println(heureDebut);
        flotOut.println(heureFin);
        flotOut.println(texte);
    }
}
```

***(Fin du sujet)***