IPST-CNAM
Programmation JAVA
NFA 032
Mercredi 25 Juin 2014

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

1ère Session NFA 032

CORRECTION

1ère PARTIE : COURS (sans document)
Durée: 1h15

1. QCM (35 points)

Dans une classe abstraite.		Q 1.
1	les méthodes doivent être toutes abstraites (une méthode abstraite est une méthode sans code)	
2	le constructeur n'existe pas et ne peut pas être écrit car il n'est pas possible de créer une instance d'une classe abstraite	
3	les attributs peuvent être de n'importe quel genre (public, private, final, static, protected)	X

Tous les attributs d'une classe abstraite doivent être **protected**.		Q 2.
1	OUI	
2	NON	X

Une classe abstraite est une classe qui est déclarée "abstract" même si elle ne contient pas de méthodes abstraites		Q 3.
1	OUI	X
2	NON	

Soit les classes A et B qui héritent de C. Les classes A et B ne sont pas abstraites. La classe C est abstraite. Soit la classe Stock contenant l'attribut : ArrayList<**C**> elements; On peut écrire :		Q 4.
1	`elements.add( new A() )`	X
2	`elements.add( new B() )`	X
3	`elements.add( new C() )`	

Soit deux classes A et B qui héritent de la classe C qui n'est pas abstraite. Les classes A et B peuvent surcharger les méthodes publiques de la classe C:		Q 5.
1	OUI	X
2	NON	

Soit la classe C1 qui hérite de C2. C1 n'est pas une classe abstraite. C2 est une classe abstraite.
Soit le constructeur de C1 :

```
public C1(int x)
{
   super(x) ;

}

C1 objetC1 = new C1(100) ;
```

Ce code est correct — Q 6.

1	OUI	X
2	NON	

Soit le code suivant : — Q 7.

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

1	attr_B est un attribut privé de B	
2	attr_B est un attribut public de B	X
3	attr_C est un attribut protected de C	X

Soit le code suivant : — Q 8.

```
X objet = C.m() ;
```

1	C ne peut pas être une interface. C est nécessairement une classe	
2	C peut être une interface	X

En JAVA, une interface peut contenir des attributs statics — Q 9.

1	OUI	X
2	NON	

En JAVA, une interface permet de : — Q 10.

1	définir des collections polymorphes	X
2	créer des méthodes abstraites dans une classe non abstraite	
3	créer des traitements génériques	X

Soit le code JAVA suivant : — Q 11.

```
public void traitement()
{
   try{
      faire();
   }catch(Exception ex){ throw new RuntimeException("erreur"); }
```

avec la méthode "faire" qui déclenche l'exception "MyException" qui hérite de Exception.
Alors :

1	ce code ne se compile pas correctement : il manque dans l'en tête de la méthode la mention : "throws RuntimeException"	
2	la méthode "traitement" retourne une RuntimeException	X
3	ce code ne se compile pas correctement : ce n'est pas "catch(Exception ex)" qu'il faut écrire mais "catch(MyException ex)".	

Soit le code JAVA suivant : Q 12.

```
public void traitement(String nom) throws Exception
{
   Individu ind=null;
   try {
       ind = rechercher(nom);
       System.out.println(ind.toString());
   }
   catch(NonTrouveException ex) {
      throw new MyException("non trouve"); }
}
avec :
     la méthode 'rechercher' qui 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	
3	si l'individu que l'on veut ajouter n'est pas trouvé alors la méthode retourne l'exception MyException	**X**

La déclaration de la méthode suivante : Q 13.

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

précise que la méthode retourne l'exception MyException.

1	OUI	**X**
2	NON	

Soit le code suivant : Q 14.

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

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

Ce code affiche :

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

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 15.

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 **Collection.*sort*(List<T> list)** Cette méthode fonctionne si la classe d'appartenance, ici T, des éléments de la collection implémente l'interface :		Q 16.
1	Comparator	
2	Comparable	**X**
3	Comparer	

En Java, la méthode suivante permet de trier les éléments d'une collection : **Collection.*sort*(List<T> list, Comparator<? super T> c)**		Q 17.
1	Comparator est ici une classe prédéfinie du langage Java qui permet de comparer les éléments de la collection	
2	Comparator est ici une interface qui contient la méthode void compare(T o1, T o2) qui est utilisée pour comparer les éléments de la collection	**X**
3	Comparator est ici une classe abstraite du langage Java qu'il faut dériver en une classe qui surcharge la méthode void compare(T o1, T o2) qui est utilisée pour comparer les éléments de la collection	

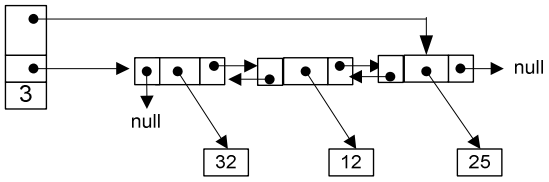
Soit le code JAVA suivant : Q 18.

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

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

Le code JAVA suivant définit la structure d'une liste doublement chainée : Q 19.

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



Le code correct pour ajouter un élément en fin de liste est :

1	public void **add**(Object value) {
	Cellule cel = new Cellule();
	cel.value = value;
	nb++;
	if (prem==null){
	prem = cel;
	dern = cel;
	}else{
	cel.prec=dern;
	cel.suiv=null;
	dern.suiv=cel;
	dern=cel; } }
		X

2	public void **add**(Object value) {
	Cellule cel = new Cellule();
	cel.value = value;
	nb++;
	cel.prec=dern;
	cel.suiv=null;
	dern.suiv=cel;
	dern=cel; }
	}	

3	public void **add**(Object value) {
	Cellule cel = new Cellule();
	cel.value = value;
	nb++;
	if (prem==null){
	prem = cel;
	dern = cel;
	}else{
	dern=cel;
	cel.prec=dern;
	dern.suiv=cel;
	cel.suiv=null;
	} }	

En Java, une collection polymorphe de contact est une collection qui est créée de la manière suivante :		Q 20.
1	ArrayList< ?> collection = new ArrayList< ?>()	
2	ArrayList<ContactAbstract> collection = new ArrayList< ContactAbstract >() où ContactAbstract est une classe abstraite dont héritent toutes les classes d'éléments que l'on veut ajouter	**X**

En Java, la méthode non statique suivante de la classe ArrayList : **public void sort() ,** permet de trier les éléments de la liste		Q 21.
1	OUI	
2	NON	X

En Java, on déclare un tableau de la manière suivante : **I tab[] = new B[100]** où **I** est une interface et B implémente l'interface I.		Q 22.
1	Cela n'est pas possible	
2	Cela est possible et on peut écrire : **t[i]=new I();**	
3	Cela est possible et on peut écrire : **t[i]=new B();**	X

En Java, pour rechercher un élément dans un tableau de Contact, ArrayList<Contact>, on peut utiliser la méthode prédéfinie public boolean **contains**. Dans ce cas il faut que :		Q 23.
1	la méthode public int compareTo(Object o) soit implémentée dans la classe Contact	
2	la méthode public boolean comparer(Object o) soit implémentée dans la classe Contact	
3	la méthode public boolean equals(Object o) soit implémentée dans la classe Contact	X

La class File ne gère que les fichiers. Pour gérer des répertoires on utilise la classe FileDirectory		Q 24.
1	OUI	
2	NON	X

Le code suivant permet de lire une chaine au clavier :
`BufferedReader in =`
`  new BufferedReader(new InputStreamReader(System.in));`
`String str = in.readLine();`
		Q 25.

1	OUI	X
2	NON	

En JAVA, pour sérialiser un objet, il est nécessaire que la classe d'appartenance de l'objet, implémente l'interface prédéfinie 'Serializable'		Q 26.
1	OUI	X
2	NON	

Lequel des 3 codes suivants permet de lire une chaine au clavier :			Q 27.
1		`InputStream in = System.in);` `String str = in.readLine();`	

2		`BufferedReader in = new BufferedReader(new`
		`  InputStreamReader(System.in));`
		`String str = in.readLine();`
			X

		`FileInputStream in = new FileInputStream(System.in);` `String str = in.readLine();`	

Les classes ObjectOutputSream et ObjectInputStream permettent d'écrire et lire dans les fichiers des tableaux contenant n'importe quel objet		Q 28.
1	OUI	X
2	NON	

Le code JAVA suivant, liste les fichiers et les répertoires qui se trouvent sous le répertoire "/home/jl"
`File varfile = new File("/home/jl");`
`while(varfile.hasMoreDirectory())`
`  System.out.println(varfile.nextDirectory());`
		Q 29.

1	OUI	
2	NON	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 — Q 30.

```
FileInputStream fis = new FileInputStream(new File("exemple.txt"));
DataInputSream dis = new DataInputStream(fis);
dis.close()
```

1	OUI	X
2	NON	

Soit le code suivant : — Q 31.

```
public class Individu implements Serializable {String nom; String prenom;}
```

Puis,

```
Individu ind = new Individu();
File fic = new File(".","fic.bin");
FileOutputStream fin = new FileOutputStream(fic);
ObjectOutputStream fich = new ObjectOutputStream(fin);
fich.writeObject((Object)ind);
fich.close() ;
```

Ce code est correct

1	OUI	X
2	NON	

On a le code suivant : — Q 32.

```
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 binaire	X
3	Ce code crée un fichier dont les informations sont dans un format texte	

Le code JAVA suivant calcule la somme des N premiers nombres entiers positifs récursivement : — Q 33.

```
static int somme(int n)
    {
        if (n==0) return 0;
        else return n + somme(n-1);
    }
```

C e code est correct :

1	OUI	X
2	NON	

En JAVA, les classes DataOutputStream et DataInputStream héritent respectivement de FileOutputStream et FileInputStream afin d'hériter des méthodes d'écriture et de lecture dans un fichier. — Q 34.

1	OUI	
2	NON	X

La classe ArrayList est une structure de données récursive — 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

Dans la programmation objet, expliquez ce qu'est une collection polymorphe.
Citez et expliquez les deux cas, en Java, de création d'une collection Polymorphe.

Dans la programmation objet, une collection polymorphe est une collection qui contient des objets de classes d'appartenances différentes. En Java, une collection de type Object peut contenir n'importe quel objet.
Il existe deux façons de créer une classe polymorphe :
- toutes les classes d'appartenance des objets de la liste polymorphe héritent d'une même classe abstraite, et le type de la collection est cette classe abstraite
- toutes les classes d'appartenance des objets de la liste polymorphe implémentent une même interface, et le type de la collection est cetteinterface.

Q 2

Quel est le principe de la sérialisation ?
A quoi sert la sérialisation ?

Le principe de la sérialisation est de plier tous les attributs d'un objet et tous les sous-attributs récursivement sous une forme linéaire qui peut être écrite dans un fichier ou sur un socket.
Et aussi de lire cette forme linéaire pour la déplier afin de reconstruire les objets et les sous-objets.
La sérialisation est utilisée pour créer des fichiers de sauvegardes de certains objets, et pour échanger des objets sur le réseau.

Q 3

En Java, une table de hachage est implémentée par la classe Hashtable.
Définir et expliquez le principe de fonctionnement d'une telle table.

En java, une Hashtable est une collection dans laquelle on ajoute un élément associé à une clef. La collection gère donc des couples (clef, élément). On accède à un élément en fonction de sa clef.

Afin d'optimiser la recherche d'une clef dans les couples, le principe de fonctionnement est qu'il existe un algorithme (algorithme de hachage) permettant de convertir la clef en un entier quasi-unique qui sert d'index de rangement.
Dans le cas où plusieurs clefs différentes correspondent à un même entier, les couples (clef, élément) correspondants sont rangés séquentiellement.

(Tourner la page)

2ème PARTIE : PROGRAMMATION (avec document)
Durée: 1h15

Problème [50 points]

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

public class releveBancaire
{
    private String numeroCompte;
    private String moisReleve;
    private double soldeDebut;
    private double soldeFin;

    private ArrayList<Transaction> transactions;

    // Constructeur
    public ReleveBancaire(String numeroCompte,
                          String moisReleve,
                          double soldeDebut,
                          double soldeFin)
    {
        this.numeroCompte = numeroCompte;
        this.moisReleve   = moisReleve;
        this.soldeDebut   = soldeDebut;
        this.soldeFin     = soldeFin;

        this.transactions = new ArrayList<Transaction>();

        // Pour tester la correction, on crée en dur des transactions
        transactions.add(new      Cheque(Calendar.getInstance(),"Coiffeur",45.60,123456,"Mr
Durand",false));
        transactions.add(new CarteBleue(Calendar.getInstance(),"Essence",65.78,true));
        transactions.add(new
Cheque(Calendar.getInstance(),"Courses",345.60,123457,"Carrefour",true));
        transactions.add(new CarteBleue(Calendar.getInstance(),"App photo",53.00,false));
        transactions.add(new          Cheque(Calendar.getInstance(),"Depot",200.00,4534,"Mr
Dupont",false));
        transactions.add(new CarteBleue(Calendar.getInstance(),"App photo",53.00,true));

    }

    // Ecriture du releve dans un fichier texte
    //
    public void creerFichierTexte() throws IOException
    {
        // Construction du nom du fichier
        String nomFichier = numeroCompte+"_"+moisReleve+".txt";

        // Le fichier est créé dans le répertoire courant
        File f = new File(nomFichier);
        FileOutputStream fos = new FileOutputStream(nomFichier);
        PrintStream ps = new PrintStream(fos);

        // Ecriture des informations generales du compte
        ps.print(String.format("%20s : %30s\n",
                               "Compte",
                               numeroCompte));
        ps.print(String.format("%20s : %30s\n",
                               "Mois",
                               moisReleve));
```

```
        // Ecriture du solde de debut
        ps.print(String.format("\n%20s : %30s\n",
                               "Debut de solde",
                               soldeDebut));

        // Ecriture de toutes les transaction du relevé
        ps.println("\nLes transactions:\n");
        for(Transaction t : transactions)
            {
                t.ecrireTexte(ps); // Une transaction par ligne
                ps.println();
            }

        // Ecriture du solde de fin
        ps.print(String.format("\n%20s : %30s\n",
                               "Fin de solde",
                               soldeFin));

        ps.close();
    }

    // Trie des transactions du releve
    //    par type de transaction croissant et par date
    //
    public void trier()
    {
        Collections.sort(transactions);
    }

    // Programme principale pour tester la correction
    //
    public static void main(String[] args) throws IOException
    {
         releveBancaire releve = new releveBancaire("8910Y",
                                                    "Janvier",
                                                    2345.56,
                                                    123.4);

        releve.trier();
        releve.creerFichierTexte();
    }
}

//Classe abstraite dont toutes les transactions héritent
//  Elle implémente l'interface Comparable pour faire le tri
//
abstract class Transaction implements Comparable<Transaction>
{

    protected int type;  /* Type de la transaction
                            -1 : non défini
                            1 : chèque honoré
                            2 : chèque non honoré
                            3 : carte bleue utilisée en France
                            4 : carte bleue utilisée à l'étranger */

    private Calendar date;   // Date de la transaction
    private String libelle;  // Description de la transaction
    private double montant;  // Montant de la transaction

    // Constructeur
    //
    public Transaction(Calendar date,
                       String libelle,
                       double montant)
    {
        this.type    = -1;  // par defaut
        this.date    = date;
        this.libelle = libelle;
```

```
        this.montant = montant;
    }

    // Ecrit, sans revenir à la ligne, les attributs communs à toute
    //   les transactions
    public void ecrireTexte(PrintStream ps)
    {
        ps.print(String.format("%02d/%02d/%4d %12s %6.2f",
                               date.get(Calendar.DAY_OF_MONTH),
                               date.get(Calendar.MONTH),
                               date.get(Calendar.YEAR),
                               libelle,
                               montant));

    }

    // Implémentation de l'interface Comparable<Transaction>
    //  Utilisé pour trier les transactions du relevé.
    //
    public int compareTo(Transaction t2)
    {
        Transaction t1 = this;

        // Si les types sont différents alors on trie dans
        //   l'ordre croissant du type
        if (t1.type != t2.type)
            {
                if (t1.type<t2.type) return (-1);
                else return (1);
            }
        // Sinon les types sont identiques, on ordonne alors
        //  en fonction de la comparaison de date
        else
            {
                return t1.date.compareTo(t2.date);
            }
    }
}

// Classe de definition d'un chèque
//  Hérite de Transaction
//
class Cheque extends Transaction
{
    private int numero;
    private String benificiaire;
    private boolean honore;

    // Constructeur
    //
    public Cheque(Calendar date,
                  String libelle,
                  double montant,
                  int numero,
                  String benificiaire,
                  boolean honore)
    {
        super(date,libelle,montant);  // Appel su constructeur hérité
        this.numero = numero;
        this.benificiaire = benificiaire;
        this.honore = honore;
        // Initialisation du type de la transaction
        if (honore) type=1;
        else type = 2;
    }

    // Ecrire les attributs spécifiques au chèque
    //
    public void ecrireTexte(PrintStream ps)
```

```
    {
        super.ecrireTexte(ps);
        String s;
        if(honore) s="";
        else s="pas honore";
        ps.print(String.format("%3s %07d  %10s %10s",
                               "",
                               numero,
                               benificiaire,
                               s));
    }
}

// Classe de définition d'une carte bleue
//   Hérite de Transaction
//
class CarteBleue extends Transaction
{
    private boolean etranger;

    // Constructeur
    //
    public CarteBleue(Calendar date,
                      String libelle,
                      double montant,
                      boolean etranger)
    {
        super(date,libelle,montant);
        this.etranger = etranger;
        if (etranger) type=4;
        else type=3;
    }

    // Ecrire les attributs spécifiques a la carte bleue
    //
    public void ecrireTexte(PrintStream ps)
    {
        super.ecrireTexte(ps);
        String s;
        if(etranger) s="etranger";
        else s="";
        ps.print(String.format("%5s %10s", "", s));
    }

}
```

(Fin du sujet)