IPST-CNAM
Programmation JAVA
NFA 032
Mercredi 25 Juin 2014

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

1ère Session NFA 032

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)
Durée: 1h15

---

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:

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)	

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

Une classe abstraite est une classe qui est déclarée "abstract" même si elle ne contient pas de méthodes abstraites		Q 3.
1	OUI	
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() )`	
2	`elements.add( new B() )`	
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	
2	NON	

Soit la classe C1 qui hérite de C2. C1 n'est pas une classe abstraite. C2 est une classe abstraite. Q 6.

Soit le constructeur de C1 :

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

}
```

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

Ce code est correct

1	OUI	
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	
3	attr_C est un attribut protected de C	

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	

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

1	OUI	
2	NON	

En JAVA, une interface permet de : Q 10.

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

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	
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	

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	
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	
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	
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	
3	Comparer	

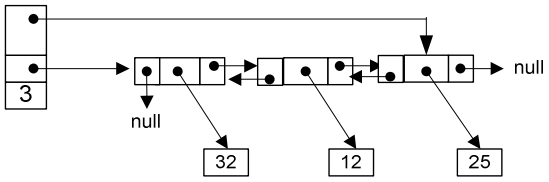
En Java, la méthode suivante permet de trier les éléments d'une collection : **Collection.*sort*(List<T> list, Comparator<? super T> c)**		Q 17.
1	Comparator est ici une classe prédéfinie du langage Java qui permet de comparer les éléments de la collection	
2	Comparator est ici une interface qui contient la méthode void compare(T o1, T o2) qui est utilisée pour comparer les éléments de la collection	
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 :
`ArrayList<String> liste = new ArrayList<String>();`
`Collections.addAll(liste,"1","3","3","1","4","5","4");`
`liste = new ArrayList<String>( new HashSet<String>(liste) );`
		Q 18.

1	La dernière ligne de ce code enlève de "liste" tous les éléments redondants	
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; } }	

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	

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	

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();**	

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	

La class File ne gère que les fichiers. Pour gérer des répertoires on utilise la classe FileDirectory		Q 24.
1	OUI	
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 25.

1	OUI	
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	
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();`	

		`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	
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	

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
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
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
2	Ce code crée un fichier dont les informations sont dans un format binaire
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
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

La classe ArrayList est une structure de données récursive — Q 35.

1	OUI
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.

Q 2

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

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.

(Tourner la page)

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

Problème [50 points]

On se propose de créer la classe ** releveBancaire** qui contient le relevé bancaire d'un particulier d'un mois donné.
Un relevé est constitué des éléments suivants :
- le numéro du compte
- le mois du relevé
- le solde (positif ou négatif) de début du relevé
- le solde (positif ou négatif) de fin du relevé
- la liste des transactions bancaires réalisées.

Il existe deux types de transaction différents, les chèques et les cartes bleues (on ne gère pas ici les autres types : virement, prélèvement, ...).
La classe abstraite **Transaction** contient les attributs et méthodes communs aux **Cheque** et **CarteBleue** qui héritent de Transaction.
Toutes les transactions sont caractérisées par un type de transaction, une date, un libellé et le montant (positif ou négatif).
Les chèques sont caractérisés par le numéro de chèque, le nom du bénéficiaire et un état qui indique si un chèque que vous avez déposé sur votre compte n'a pu être honoré.
Les cartes bleues sont caractérisées par un indicateur qui indique si la transaction a été faite sur le territoire français ou à à l'extérieur du territoire français.

Le "type de transaction" dans la classe Transaction est :
- 1 s'il s'agit d'un chèque honoré
- 2 s'il s'agit d'un chèque non honoré
- 3 s'il s'agit d'une carte bleue national
- 4 s'il s'agit d'une carte bleue à l'étranger.

1/ Ecrire les attributs et les constructeurs des classes **releveBancaire**, **Transaction**, **Cheque** et **CarteBleu**.

2/ Ecrire la méthode générique de la classe **releveBancaire**:

public void creerFichierTexte()

Cette méthode écrit dans un fichier texte le relevé bancaire dans sa totalité afin qu'il soit lu en éditant tout simplement le fichier. Le nom du fichier est la concaténation du numero de compte et du mois du relevé avec l'extension .txt.

Il faut que cette méthode soit suffisamment générique pour que nous n'ayons pas à la modifier dans le cas où on voudrait créer un nouveau type de transaction (une nouvelle classe qui hériterait de Transaction).

Vous écrivez bien sur, si besoin, les méthodes des classes **Transaction**, **Cheque** et **CarteBleue** dont vous avez besoin pour écrire la méthode **creerFichierTexte.**

NB: vous n'avez pas besoin d'écrire les getteurs et setteurs des attributs. Vous vous les donnez.

Vous devez obtenir un relevé le plus proche de cet exemple :

```
              Compte :                          8910Y
                Mois :                        Janvier

      Debut de solde :                        2345.56

Les transactions:

25/05/2014      Courses 345,60    0123457   Carrefour
25/05/2014     Coiffeur  45,60    0123456   Mr Durand pas honore
25/05/2014        Depot 200,00    0004534   Mr Dupont pas honore
25/05/2014    App photo  53,00
25/05/2014      Essence  65,78        etranger
25/05/2014    App photo  53,00        etranger

        Fin de solde :                          123.4
```

3/ Ecrire la méthode qui permet de trier le relevé bancaire par "type de transaction" puis dans chacun des types par date croissante.

(Fin du sujet)