IPST-CNAM
Programmation JAVA
NFA 032
Mercredi 22 Juin 2016

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)
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:

Soit les classes A et B qui héritent de la classe C. A et B contiennent la déclaration d'un même attribut. Cela est une erreur de programmation (erreur de compilation) car cet attribut doit se trouver dans la classe C.		Q 1.
1	OUI	
2	NON	**X**

Soit une classe B qui hérite d'une classe A et, A et B n'ont pas de constructeur codé. Lors de la création d'un objet B :		Q 2.
1	une erreur d'exécution se produit car A et B n'ont pas de constructeur	
2	l'objet B est créé, les attributs de A et B sont alloués en mémoire et initialisés par défaut	**X**

Soit une classe B qui hérite d'une classe A. A a un seul constructeur codé avec des paramètres. B n'a pas de constructeur codé. Cela provoque une erreur de compilation.		Q 3.
1	OUI	**X**
2	NON	

En programmation objet, une collection est polymorphe si la classe d'appartenance des éléments de la collection hérite de plusieurs classes		Q 4.
1	OUI	
2	NON	**X**

Une classe abstraite est une classe dans laquelle au moins une méthode est abstraite		Q 5.
1	OUI	**X**
2	NON	**X**

Dans le cours cela est dit et écrit donc la réponse devrait être OUI, mais hélas cela est faux dans le cours car en Java (au moins maintenant), une classe est abstraite quand elle est déclarée abstraite. Cela suffit, même s'il n'y a pas de méthode abstraite.
Quelque soit la réponse, la note retenu est de 1 pour tous.

Une classe abstraite ne peut pas avoir de constructeur codé.		Q 6.
1	OUI	
2	NON	**X**

Tous les attributs d'une classe abstraite doivent être publics sinon les classes qui héritent de cette classe abstraite ne peuvent pas accéder directement à ses attributs.		Q 7.
1	OUI	
2	NON	**X**

La caractéristique "protected" d'un attribut de la classe C est utilisée pour que les classes qui héritent de C puissent accéder directement à cet attribut		Q 8.
1	OUI	**X**
2	NON	

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 9.
1	ArrayList<I> elements;	**X**
2	ArrayList<H> elements;	**X**
3	ArrayList<A,B,C> elements	

Soit le code suivant : Q 10.

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

1	attr_B est un attribut protected de B	X
2	attr_B est un attribut privé de B	
3	attr_C est un attribut static public de l'interface C	X

Soit le code suivant : Q 11.

```
abstract public class H {
   public H(){ }
}

public class A extends H
{
    static public getInstance(){
      return new A();
    }
}
```

Ce code est correct

1	OUI	X
2	NON	

En JAVA, une exception est un objet dont la classe d'appartenance peut être : Q 12.

1	la classe prédéfinie Exception	X
2	une classe qui hérite de la classe RuntimeException	X
3	une classe qui hérite de la classe Object car Object hérite de la classe Exception	

Le code suivant est correct : Q 13.

```
public void action(int parametre) throws Exception
{
   if (parametre==0)
      throw new Exception("Erreur");
   else
       faireLeTraitement();
}
```

1	OUI	X
2	NON	

En JAVA, l'instruction suivante permet de déclencher une exception Q 14.

```
throw new MyException("Impossible de faire l'action");
```

avec
MyException une classe qui hérite de Exception

1	OUI	X
2	NON	

La déclaration de la méthode suivante : Q 15.

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

précise que la méthode peut déclencher l'exception MyException dans le corps de sa méthode.

1	OUI	X
2	NON	

Soit le code suivant : Q 16.

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

Erreur dans le sujet ce code aurait du être après :

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

Ce code affiche :

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

Erreur dans le sujet : implique la note retenue pour cette question est pour tous 1 quelque soit la réponse.

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 statique : Q 17.
Collections.*sort*(List<T> list)
Cette méthode fonctionne si la classe T implémente l'interface Comparable

1	OUI	**X**
2	NON	

En Java, la classe Collections est une classe abstraite dont hérite la classe ArrayList Q 18.

1	OUI	
2	NON	**X**

En Java, la différence entre l'interface List et l'interface Collection est que l'interface List définit des méthodes supplémentaires basées sur la notion d' « **index** ». Q 19.

1	OUI	**X**
2	NON	

Soit la déclaration suivante : Q 20.
public ArrayList<AbstractH> liste = new ArrayList<AbstractH>();
avec AbstractH qui est une classe abstraite et les classes A et B qui héritent de AbstractH.
Nous pouvons écrire le code suivant :

1	liste.add(new A());	**X**
2	liste.add(new B());	**X**
3	liste.add(new AbstractH());	

Soit la déclaration suivante :
public ArrayList<H> liste = new ArrayList<H>();
avec H qui n'est pas une classe abstraite et les classes A et B qui héritent de H.
Nous pouvons écrire le code suivant :
`liste.add( new A() );`
`liste.add( new B() );`
		Q 21.

1	OUI	**X**
2	NON	

Soit une interface Java **InterfaceEx** et soit une méthode de la classe **A** dont la signature est la suivante : `public String unTraitement(InterfaceEx i)`		Q 22.
1	L'interface **InterfaceEx** doit implémenter la méthode **unTraitement**	
2	La classe A doit implémenter les méthodes de l'interface **InterfaceEx**	
3	Il est possible de passer en paramètre de la méthode **unTraitement** un objet dont la classe d'appartenance **B** implémente l'interface **InterfaceEx**.	**X**

Le code JAVA suivant, liste les noms des fichiers et les noms des répertoires qui se trouvent dans le répertoire "/home/jl"
`File varfile;`
`varfile = new File("/home/jl");`
`for(String nom : varfile.list())`
`  System.out.println(nom);`
		Q 23.

1	OUI	**X**
2	NON	

Le code JAVA suivant, crée physiquement un fichier de nom "toto.txt" dans le répertoire "data" : `File varfile = new File("./data/toto.txt");`		Q 24.
1	OUI	
2	NON	**X**

Le code suivant crée un fichier de nom "exemple.bin" dans le répertoire courant d'exécution. Le fichier est créé contenant des informations binaires (une chaine, un double):
`FileOutputStream fos = new FileOutputStream(new File("exemple.bin"));`
`DataOutputStream dos = new DataOutputStream(fos);`
`dos.writeUTF("EXEMPLE");`
`dos.writeDouble(123.456);`
`dos.close()`
		Q 25.

1	OUI	**X**
2	NON	

Le code suivant crée un fichier texte (qui peut être lu avec un éditeur de texte) de nom "exemple.txt" dans le répertoire courant d'exécution :
`FileOutputStream fos = new FileOutputStream(new File("exemple.txt"));`
`PrintStream ps = new PrintStream(fos);`
`ps.println("EXEMPLE");`
`ps.close()`
		Q 26.

1	OUI	**X**
2	NON	

En JAVA, tout objet dont la classe d'appartenance hérite de la classe ObjectOutputStream, peut être sérialisé.		Q 27.
1	OUI	
2	NON	**X**

En JAVA, pour créer un thread T, on peut :		Q 28.
1	implémenter la méthode void run() dans la classe T qui implémente l'interface Runnable	**X**
2	implémenter la méthode void run() dans la classe Thread qui hérite de T	
3	implémenter la méthode void run() dans la classe T qui hérite de Thread	**X**

La méthode "start" est la méthode qui permet de démarrer un thread. Cette méthode appartient à la classe Thread. On y accède directement ou par héritage de la classe Thread.		Q 29.
1	OUI	**X**
2	NON	

Soit une classe MyThread qui hérite de Thread et qui contient un attribut défini comme suit : `private static int tempo;` Alors tous les threads (instances de MyThread), partage la même valeur tempo (exemple: temps de temporisation des threads)		Q 30.
1	OUI	**X**
2	NON	

En JAVA, la classe ServerSocket est une classe prédéfinie qui hérite de Thread. Ainsi, elle permet de traiter les requêtes de plusieurs clients en parallèle		Q 31.
1	OUI	
2	NON	**X**

Le socket est un canal de communication permettant de faire communiquer deux logiciels distants se trouvant sur deux machines à travers un réseau TCP/IP		Q 32.
1	OUI	**X**
2	NON	

Soit le code du serveur suivant :
`ServerSocket ssoc = new ServerSocket(9100);`
`Socket soc = ssoc.accept();`
`DataInputStream dis= new DataInputStream(soc.getInputStream());`
`String requete = dos.readUTF();`
		Q 33.

1	dès son exécution la méthode *accept* retourne un socket. Le serveur se met alors en attente de lecture avec l'instruction dos.readUTF(), il attend qu'un client écrit sur le socket	
2	dès son exécution la méthode *accept* se met en attente qu'un client crée un socket sur le port 9100.	**X**

En JAVA, sur un socket, il est possible d'écrire n'importe quel type d'information (type primitifs, types référence)		Q 34.
1	OUI	**X**
2	NON	

Dans l'architecture client-serveur classique, le serveur est le "maître". C'est-à-dire que c'est à son initiative qu'il traite les requêtes des clients en demandant ces requêtes aux différents clients.		Q 35.
1	OUI	
2	NON	**X**

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.*

QUESTION 1 :
Expliquez le rôle d'une classe abstraite dans la conception d'une collection polymoprhe.

**Dans une collection polymorphe, la classe abstraite est le type de l'élément de la collection. Elle contient des méthodes abstraites implémentées par les classes héritantes ce qui permet à une classe qui utilise une telle collection de créer des traitements génériques sur tous les types qui héritent de cette classe abstraite.
Elle contient bien sur tous les attributs communs aux classes héritantes dont notamment les attributs utilisés pour par exemple trier une telle collection. Ainsi c'est elle qui contient le traitement de comparaison utilisé par l'algorithme de tri.**

QUESTION 2 :
Expliquez avec précision les différences qui existent entre les classes **Exception** et **RuntimeException**
(principes, syntaxe de code, rôle, …)

**La classe RuntimeException hérite de la classe Exception.
La classe Exception est utilisée pour créer des exceptions qui doivent être signalées en entête des méthodes qui sont susceptibles de les retourner. On utilise le mot clef "throws" suivi de "Exception" pour cela.
Alors que la classe RuntimeException est utilisée pour créer des exceptions qui n'ont pas besoin d'être signalées en entête des méthodes qui sont susceptibles de les retourner.**

QUESTION 3 :
Expliquer ce qu'est la notion d'héritage dans le cadre des langages de programmation orientée objet.

**La notion d'héritage est une relation entre deux classes de la programmation objet. On dit que A hérite de B.
Cela signifie qu'une instance de A se comporte comme si elle était une instance de B :
- A contient ses attributs mais aussi tous les attributs de B
- A peut utiliser les méthodes publics ou protected de B
- A peut utiliser directement les attributs publics ou protected de B
L'héritage permet de factoriser les attributs communs à plusieurs classes (principe de généralisation).
L'héritage permet aussi de surcharger dans A des méthodes de la classe B (principe de spécialisation).
Un objet de type A peut être passé en paramètre d'une méthode qui accepte un objet de type B.
L'héritage permet aussi de créer des collections polymorphe.**

FIN DE LA 1ère PARTIE

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

PROBLEME [50 points]

Question 1 :

```
import java.util.*;

public class Banque
{
   private ArrayList<AbstractCompte> elements;

   public Banque()
   {
      elements = new ArrayList<AbstractCompte>();
   }

   public String toString()
   {
      String res="";
      for(AbstractCompte c:elements)
         res=res+c.toString();
      return res;
   }
}
```

```
public abstract class AbstractCompte
{
   protected String nom;
   protected String numero;
   protected double solde;

   public AbstractCompte(String nom,String numero,double solde)
   {
      this.nom=nom;
      this.numero = numero;
      this.solde = solde;
   }

   public String toString()
   {
      String res =
         String.format("%30s : %30s\n%30s : %30s\n%30s:%7.2f\n",
                  "nom",nom,
                  "numero",numero,
                  "solde",solde);
      return res;
   }
}
```

```
public class CompteCourant extends AbstractCompte
{
   protected double autorisationDecouvert;

   public CompteCourant(String nom,String numero,double solde,
                        double autorisationDecouvert)
   {
      super(nom,numero,solde);
      this.autorisationDecouvert=autorisationDecouvert;
   }

   public String toString()
   {
      String res = super.toString();
      res = res +
         String.format("%30s : %7.2f\n",
                       "decouvert",autorisationDecouvert);
      return res;
   }
}
```

```
public class CompteEpargne extends AbstractCompte
{
   protected double taux;

   public CompteEpargne(String nom,String numero,double solde,
                        double taux)
   {
      super(nom,numero,solde);
      this.taux=taux;
   }

   public String toString()
   {
      String res = super.toString();
      res = res +
         String.format("%30s : %2.2f\n",
                       "taux",taux);
      return res;
   }
}
```

Question 2 :

```
public void importer(String nomFichier)
{
   String[] lignes = Terminal.lireFichierTexte(nomFichier);

   for(int i=0;i<lignes.length;i++)
      {
         String ligne = lignes[i];
         String[] champs = ligne.split("[;]",5);

         if (champs[0].equals("COURANT"))
            {
               CompteCourant cc = new CompteCourant(
                                          champs[1],
                                          champs[2],
                                          Double.parseDouble(champs[3]),
                                          Double.parseDouble(champs[4]));
               elements.add(cc);
            }
         if (champs[0].equals("EPARGNE"))
            {
               CompteEpargne ce = new CompteEpargne(
                                          champs[1],
                                          champs[2],
                                          Double.parseDouble(champs[3]),
                                          Double.parseDouble(champs[4]));
               elements.add(ce);
            }
      }
}
```

Question 3 :

Il faut modifier la déclaration de la classe AbstractCompte :

```
public abstract class AbstractCompte implements Comparable<AbstractCompte>
```

et écrire la méthode suivante dans cette classe :

```
public int compareTo(AbstractCompte c)
{
   if (! nom.equals(c.nom))
      {
         return nom.compareTo(c.nom);
      }
   else
      {
         if (solde<c.solde) return 1;
         else return -1;
      }
}
```

puis dans la classe Banque écrire la méthode :

```
public void trier()
{
   Collections.sort(elements);
}
```

Question 4 :

```
public String comptesADecouvert()
{
   String res="";
   for(AbstractCompte c : elements)
      {
         if (c instanceof CompteCourant)
            {
               CompteCourant cc = (CompteCourant)c;
               if ( (cc.solde<0) && (- cc.solde > cc.autorisationDecouvert) )
                  res = res+cc.numero+"\n";
            }
      }
   return res;
}
```

(Fin du sujet)