IPST-CNAM
Programmation JAVA
NFA 001
Mercredi 10 Février 2016

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

1ère Session NFA 031

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, avec document, consacrée à 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 le code en Java.

1ère PARTIE : COURS (sans document)

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:

En JAVA, il est possible d'écrire des méthodes en dehors de toute classe. On écrit les méthodes dans un fichier .java. Elles sont toutes des méthodes statiques.		Q 1.
1	OUI	
2	NON	**X**

En JAVA, une classe contient la déclaration des attributs et la déclaration des méthodes.		Q 2.
1	OUI	**X**
2	NON	

Dans un langage orienté objet, un objet appartient toujours à une classe.		Q 3.
1	OUI	**X**
2	NON	

Dans un langage orienté objet, un principe fort est que les données sont encapsulées dans une structure d'accès appelée une classe.		Q 4.
1	OUI	**X**
2	NON	

En programmation objet, la relation d'agrégation entre une classe A et B est utilisée pour décrire qu'un objet de type A se décompose en un sous-objet de type B. La vie du sous-objet B dépend de la vie de l'objet A auquel il appartient.		Q 5.
1	OUI	
2	NON	**X**

Soit le fichier suivant C:\CodeJava\exercices\fr\cnam\util\Terminal.java. Le fichier Terminal.java contient en 1ère ligne : **package exercices.fr.cnam.util;** Le répertoire C:\bin est vide. Dans C: on réalise la commande suivante : **javac -d C:\bin C:\CodeJava\exercices\fr\cnam\util\Terminal.java** On obtient dans le répertoire bin la création suivante :		Q 6.
1	fr/cnam/util/Terminal.class	
2	exercices/fr/cnam/util/Terminal.class	**X**

La commande javac prend en entrée un ou plusieurs fichiers .java. Elle les compile et crée un fichier exécutable binaire qui ensuite pourra être lancé sur le système d'exploitation.		Q 7.
1	OUI	
2	NON	**X**

Pour exécuter un programme JAVA, il faut créer une classe de nom **Main** dans laquelle on écrit le code du programme principal.		Q 8.
1	OUI	
2	NON	**X**

Pour exécuter un programme JAVA, il faut créer une méthode **main** dans n'importe quelle classe et dont la signature peut être :		Q 9.
1	public void main()	
2	public static void main()	
3	public static void main(String[] args)	**X**

Le fichier C1.java contient 2 classes : une classe publique C1 et une classe privée C2. La commande **javac** C1.java		Q 10.
1	crée qu'un fichier : C1.class	
2	compile la classe C1 et exécute la méthode main de la classe C1	
3	crée deux fichiers : C1.class et C2.class	**X**

Soit le code suivant : Q 11.

```
public class Exemple {
  public static void main(String args[]) {
    String str = args[0];
    System.out.println(str);
}}
```

Commande d'exécution : java Exemple toto

1	Ce programme ne se compile pas car il y a une erreur de syntaxe	
2	L'exécution échoue car il y a une erreur d'exécution	
3	L'exécution de ce programme affiche à l'écran la chaine de caractère passée en argument: toto	**X**

Une classe C appartient à un package **fr.cnam.exemple** si le fichier **C.java** se trouve dans le répertoire **fr/cnam/exemple** et si en en-tête du fichier **C.java** il y a la ligne : **package fr.cnam.exemple**;		Q 12.
1	OUI	**X**
2	NON	

Soit une classe contenant les méthodes **mstat1** et **m2**. mstat1 est une méthode statique et m2 n'est pas une méthode statique :		Q 13.
1	la méthode mstat1(statique) peut utiliser les attributs non statiques de la classe	
2	la méthode m2 (non statique) peut utiliser les attributs statiques de la classe	**X**
3	la méthode mstat1 (statique) peut utiliser les attributs statiques de la classe	**X**

Soit le code JAVA suivant : Q 14.

```
public class C {
  private int attribut;
  public C(){attribut=10;}
  public static void main(String[] args)
  {
    C  objet = new C();
    objet.attribut = Integer.parseInt(args[0]);
  }
}
```

La partie de code en gras est correct

1	OUI	
2	NON	**X**

En JAVA, les méthodes déclarées en dehors d'une classe sont appelées des méthodes statiques.		Q 15.
1	OUI	
2	NON	**X**

Le paramètre **-classpath** ou la variable d'environnement **CLASSPATH** est utilisée pour désigner une liste de plusieurs chemin (ou path) d'accès à des répertoires. Chacun de ces répertoires contient les fichiers .class ou les packages utilisés dans la compilation ou dans l'exécution d'un programme JAVA.		Q 16.
1	OUI	**X**
2	NON	

Un attribut privé (private) d'une classe A peut être directement utilisé par une méthode d'une autre classe B qui appartient au même package que A.		Q 17.
1	OUI	
2	NON	**X**

Un attribut protégé (protected) d'une classe A peut être directement utilisé par une méthode d'une autre classe B qui appartient au même package que A.		Q 18.
1	OUI	**X**
2	NON	

Soit la classe C1 qui contient un attribut privé. Une méthode de la classe C2 prend en paramètre une instance de la classe C1 et veut changer la valeur de cet attribut. Cela est possible que s'il existe un setteur sur cet attribut dans la classe C1.		Q 19.
1	OUI	**X**
2	NON	

Dans la programmation objet, en JAVA, le rôle du constructeur d'une classe est de :		Q 20.
1	initialiser les attributs de la classe	**X**
2	déclarer de nouveaux attributs	
3	construire la classe (ou .class) qui permet à un autre programme de créer les objets de la classe	

Soit la classe suivante : Q 21.

```
public class C1{
  private int[] tab;
  public C1(int[] tt){ tab = tt; }
}
```

L'instruction suivante :

```
C1 c1 = new C1();
```

est valide, et la valeur de tab est la valeur par défaut de Java : null

1	OUI	
2	NON	**X**

Dans le cadre du développement d'un programme Java, on définit une classe contenant aucune méthode et contenant que des attributs statics et publics.		Q 22.
1	Cela n'est pas possible.	
2	Cela est possible mais inutile	
3	Cela permet de déclarer des variables globales à tout le développement du programme Java	**X**

Soit le code JAVA suivant : Q 23.

```
public class Exemple
{
  ????? int var_x;
}

public class C
{
   public void traitement(Exemple ex)
   {
      ex.var_x = 100;
   }
```

Pour que ce code soit correct, il faut remplacer **?????** par :

1	private	
2	public	**X**

Dans le constructeur d'une classe, il est possible d'appeler une méthode (statique ou non statique) de la même classe.		Q 24.
1	OUI	**X**
2	NON	

Soit le code JAVA suivant : Q 25.

```
public class C {
  private int attribut;
  public C(int param){attribut=param;}
  public int getAttribut(){return attribut;}
  public static void main(String[] args)
  {
    C  objet = new C();
    System.out.println(objet.getAttribut());
  }
}
```

Ce code est correct

1	OUI	
2	NON	**X**

En JAVA, une classe peut contenir plusieurs constructeurs		Q 26.
1	OUI	**X**
2	NON	

Soit le code JAVA suivant : Q 27.

```
public class Livre
{
    // Le titre du livre
    private String titre;

    // Le constructeur
    public Livre(String titre){
      setTitre(titre);}

    // Setteur
    public void setTitre(String titre){
      this.titre=titre; }
}
```

L'utilisation (en gras) de setTitre dans le constructeur :

1	déclenche une erreur de compilation	
2	déclenche une erreur d'exécution	
3	est correct	**X**

En JAVA, le type prédéfini primitif **int** est un type référence (pointeur). Q 28.

1	OUI	
2	NON	**X**

Soit le code suivant : Q 29.

```
Livre l = new Livre();
l.nom = "Les cavernes d'acier";
ArrayList<Livre> livres = new ArrayList<Livre>();
livres.add(l);
l.nom="Face aux feux du soleil";
System.out.println(livres.get(0).nom);
```

Ce code affiche :

1	Face aux feux du soleil	**X**
2	Les cavernes d'acier	

En Java, il est possible de modifier la référence d'un objet passé en paramètre d'une méthode. Q 30.

1	OUI	
2	NON	**X**

Soit le code JAVA suivant: Q 31.

```
public class C {
  public static void initTab(int nb, int[] tab){
    tab = new int[nb]; }
}
```

Et par ailleurs :

```
int[] tableau;
C.initTab(10,tableau);
System.out.println(tableau.size());
```

Ce code :

1	déclenche une erreur d'exécution (nullPointerException)	**X**
2	affiche la valeur : 0	
3	affiche la valeur: 10	

Le code suivant permet d'augmenter la taille du tableau t1 : Q 32.

```
int[] t1 = new int[10];
t1 = {1,2,3,4,5,6,7,8,9,10};

t1 = augmenterTailleTab(t1,100);
```

Avec :

```
public static int[] augmenterTaille(int[] t,int
newTaille) {
  int[] tmp = new int[newTaille];
  for(int i=0;i<t.length;i++) tmp[i]=t[i];
  return tmp;
}
```

1	OUI	**X**
2	NON	

L'avantage d'utiliser un ArrayList à la place d'un tableau [] est : Q 33.

1	Le ArrayList permet de stocker des objets alors que le tableau [] ne peut stocker que des types primitifs (int, double, ….)	
2	Le ArrayList permet de stocker un nombre indéterminé d'objet	**X**

Soit le code JAVA suivant : Q 34.

```
String[] tab = new String[4];
tab[0]="UN"; tab[1]="DEUX";
for(String s : tab){System.out.println(s);}
```

1	Ce code affiche : UN DEUX null null	**X**
2	Ce code affiche : UN DEUX	

Dans la classe String la méthode static ***void replace(String old, String new, String target)*** permet de remplacer dans la chaine ***target*** toutes les occurrences de la chaîne ***old*** par la chaîne ***new***. Q 35.

Exemple:

```
String cible = "un deux un quatre un";
String.replace("un","trois",cible);
System.out.println(cible);
// affiche: trois deux trois quatre trois"
```

1	OUI	
2	NON	**X**

2. Questions libres (15 points)

Q 1

Expliquez pourquoi en JAVA, il est possible de modifier le contenu d'un objet qui est passé en paramètre d'une méthode. Justifier votre réponse.

En JAVA, il est possible de modifier le contenu d'un objet qui est passé en paramètre d'une méthode car, en Java, un objet est un pointeur, c'est-à-dire que la valeur passé en paramètre est l'adresse mémoire du bloc mémoire contenant les attributs de l'objet, et non l'objet lui-même. La référence (ou adresse) de l'objet étant passé en paramètre (passage par valeur), il est alors possible, dans le code de la méthode, d'accéder aux attributs de l'objet et/ou à ses méthodes.

Q 2

En JAVA, il existe deux classes différentes, **String** et **StringBuffer**, qui permettent de gérer les chaînes de caractères.
Quel est la différence importante entre ces deux classes ?
Citez un exemple original (pas celui de la classe Terminal vu en cours) pour lequel vous utiliseriez la classe StringBuffer.

La différence importante entre ces deux classes est qu'on ne peut pas modifier le contenu d'une String alors que l'on peut modifier le contenu d'une StringBuffer.

Par exemple, on peut utiliser la classe StringBuffer pour construire les valeurs résultats des méthodes de la classe Compte au lieu de faire de la concaténation de chaine.
Exemple :

```
public String listerToutesOperations()
    {
        StringBuffer res=new StringBuffer();
        for(Operation ope : elements){
            res.append(ope.toString();
            res.append("\n--------------------\n"); }
        return res.toString();
    }
```

Q 3

Expliquez ce qu'est un "**package**" en JAVA.

Un package Java est un répertoire contenant des fichiers .java et dont chacun de ces fichiers est marqué comme appartenant à un nom de package. On indique en début du fichier ce nom par le mot prédéfini "package". Exemple : package fr.cnam.projet;
La compilation de ces fichiers crée les chemins d'accès et crée les fichiers .class dans le répertoire de destination de compilation.
On peut ainsi importer (commande import) les packages contenant les classes que l'on veut utiliser dans du code Java.

2ème PARTIE : PROGRAMMATION (avec document)

Problème 1 [15 points]

```
// rechercher une operation par son libelle
public String rechercher(String chaine)
{
    String res = "";
    for(Operation ope:elements)
        {
            String libelle = ope.getLibelle().toLowerCase();
            boolean ok=false;
            StringTokenizer strtok = new StringTokenizer(chaine," ");
            while (strtok.hasMoreTokens())
                {
                    String mot = strtok.nextToken().toLowerCase();
                    if (libelle.indexOf(mot)!=-1) ok=true;
```

```
                }
            if (ok) res=res+ope.toStringLigne()+"\n";
        }
    return res;
}
```

Problème 2 [35 points]

1/

```
public class Banque
{
    private ArrayList<CompteClient> elements;  // La liste des operations

    public Banque(String nomFichier)
    {
        elements = new ArrayList<CompteClient>();

        // On recupere toutes les lignes du fichier
        String[] lignes = Terminal.lireFichierTexte(nomFichier);

        //Les lignes : les comptes client
        //  Pour chaque ligne on va decoder la ligne puis ajouter
        //   le compte client
        //
        for(int i=0;i<lignes.length;i++)
            {
                // Decodage de la ligne
                String ligne = lignes[i];
                String[] champs = ligne.split("[;]",5);

                //creation et ajout de l'operation dans la collection
                //
                ajouterCompteClient(champs[0],
                                    champs[1],
                                    champs[2],
                                    champs[3],
                                    champs[4]);
            }
    }

    // Methode qui cree et ajoute un compte client dans la collection
    //
    public void ajouterCompteClient(String dateCreation,
                                    String numeroClient,
                                    String nomClient,
                                    String dateAnniv,
                                    String nomCompte)

    {
        CompteClient compte = new CompteClient(dateCreation,
                                               numeroClient,
                                               nomClient,
                                               dateAnniv,
                                               nomCompte);
        // Ajout du compte client
        elements.add(compte);
    }

    // Recherche d'un compte d'un client
    public Compte lireCompte(String nomCompte)
```

```
{
    for(CompteClient cc:elements)
        if (cc.getNomCompte().equals(nomCompte))
            {
                Compte c = new Compte(
                                "data/Compte"+nomCompte+".txt");
                return c;
            }
    return null;
}

// Retourne les comptes d'un client
public ArrayList<String> comptesClient(String numeroClient)
{
    ArrayList<String> tmp = new  ArrayList<String>();
    for(CompteClient cc:elements)
        if (cc.getNumeroClient().equals(numeroClient))
            tmp.add(cc.getNomCompte());
    return tmp;
}
```