IPST-CNAM
Programmation JAVA
NFA 032
Mercredi 4 Juillet 2018

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:

En programmation objet, le lien d'héritage est un lien de relation entre deux classes B et A (B hérite de A) qui permet, entre autre, que les méthodes de B, privées ou publiques, peuvent utiliser directement les méthodes publiques de A		Q 1.
1	OUI	X
2	NON	

En programmation objet, le lien d'héritage est un lien de relation entre deux classes B et A (B hérite de A) qui permet, entre autre, que les méthodes de B, privées ou publiques, peuvent utiliser directement les attributs privés et publiques de A		Q 2.
1	OUI	
2	NON	X

Un des usages important de l'héritage est de factoriser les attributs communs à plusieurs classes dans une autre classe dont elles héritent.		Q 3.
1	OUI	X
2	NON	

Soit B qui hérite de A, alors tout constructeur de B hérite toujours d'un constructeur de A (B appelle, via super, un constructeur de A).		Q 4.
1	OUI	X
2	NON	

Si le constructeur d'une classe B utilise l'instruction "super();" alors cela signifie que B hérite d'une classe A dans laquelle il n'existe pas de constructeur ou il existe un constructeur sans paramètres.		Q 5.
1	OUI	X
2	NON	

En JAVA, le mot clef "super" n'est utilisé dans une classe que pour appeler les constructeurs de la classe dont elle hérite.		Q 6.
1	OUI	
2	NON	X

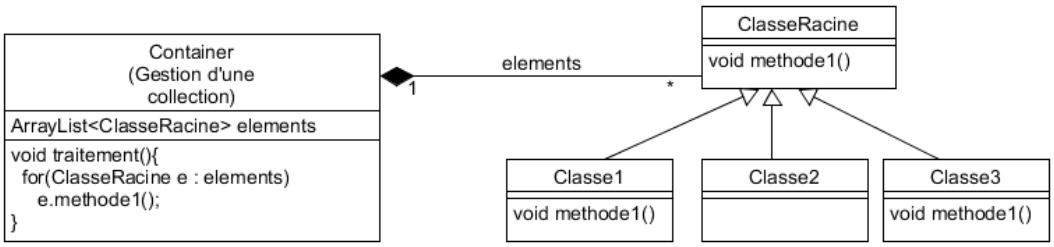
En JAVA, une classe B qui hérite de A, ne peut surcharger que les méthodes privées de la classe A.		Q 7.
1	OUI	
2	NON	X

En JAVA, toutes les classes, prédéfinies ou non prédéfinies, héritent de la classe prédéfinie Object.		Q 8.
1	OUI	X
2	NON	

En programmation objet JAVA, le polymorphisme d'une collection d'élément est possible quand toutes les classes d'appartenance des éléments héritent d'une même classe abstraite.		Q 9.
1	OUI	X
2	NON	

En programmation objet JAVA, une méthode générique est une méthode qui appartient à une classe qui implémente au moins une interface.		Q 10.
1	OUI	
2	NON	X

Soit le schéma suivant : Q 11.



1	La classe "ClasseRacine" peut être une classe quelconque	X
2	La méthode "méthode1" de la classe "ClasseRacine" doit être une méthode abstraite	
3	la méthode "traitement" est un traitement générique	X

Dans une classe abstraite toutes les méthodes doivent être des méthodes abstraites.		Q 12.
1	OUI	
2	NON	X

En programmation objet, soit une méthode dont un paramètre est une interface alors on doit passer en paramètre une instance d'une classe qui doit implémenter cette interface.		Q 13.
1	OUI	X
2	NON	

Soit une classe B qui hérite d'une classe A. A possède un seul constructeur codé avec des paramètres. B n'a pas de constructeur codé. Cela provoque une erreur de compilation.		Q 14.
1	OUI	X
2	NON	

En JAVA, il est possible de créer une instance d'une classe abstraite.		Q 15.
1	OUI	
2	NON	X

Soit les classes quelconques A et B qui héritent de C. C n'est pas une classe abstraite.
Soit la classe Stock contenant l'attribut : ArrayList<**C**> elements;
On peut écrire le code suivant :
`elements.add( new A() )`
`elements.add( new B() )`
`elements.add( new C() )`
		Q 16.

1	OUI	X
2	NON	

En JAVA, une exception est un objet		Q 17.
1	OUI	X
2	NON	

En JAVA, une exception ne peut être utilisée que dans les méthodes privées.		Q 18.
1	OUI	
2	NON	X

Le code suivant est correct : Q 19.

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

1	OUI	
2	NON	**X**

Soit le code JAVA suivant : Q 20.

```
public void traitement(String nom)
{
   Individu ind=null;
   try {
       ind = rechercher(nom);
       System.out.println(ind.toString());
   }
   catch(NonTrouveException ex) {
      throw new RuntimeException("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 RuntimeException	**X**
3	Si l'individu que l'on veut ajouter n'est pas trouvé alors la méthode ne retourne pas d'exception	

Soit le code suivant : Q 21.

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

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

Ce code affiche :

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

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 **Collections.*sort*(List<T> list)** Cette méthode fonctionne si la classe d'appartenance, ici T, des éléments de la collection implémente l'interface :		Q 22.
1	Comparator	
2	Comparable	**X**
3	Compare	

En Java, la méthode statique **static public void sort(List<E> liste)** de la classe ArrayList permet de trier les éléments de la collection *liste* passée en paramètre		Q 23.
1	OUI	
2	NON	**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 24.
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**

En JAVA, pour rechercher un élément, la collection Hashset est plus performante que la classe ArrayList		Q 25.
1	OUI	**X**
2	NON	

En JAVA, la classe Hashtable<K,V> est une classe de collection permettant de stocker des éléments de type V qui sont ordonnés par ordre croissant sur l'index K		Q 26.
1	OUI	
2	NON	**X**

En JAVA, la classe File permet de gérer les fichiers et les répertoires.		Q 27.
1	OUI	**X**
2	NON	

Soit le code suivant :

```
File f;
f = new File("/home/jl","fic.txt");
```

Ce code crée un fichier de nom "fic.txt" dans le répertoire "/home/jl". Le fichier créé est vide (taille = 0).

Q 28.

1	OUI	
2	NON	**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 29.

1	OUI	**X**
2	NON	

Pour écrire et lire dans un fichier des chaînes de caractère, des entiers, des doubles, … on peut utiliser les classes prédéfinies DataOutputStream et DataInputStream.		Q 30.
1	OUI	X
2	NON	

Pour écrire et lire dans un socket des chaînes de caractère, des entiers, des doubles, … on peut utiliser les classes prédéfinies DataOutputStream et DataInputStream.		Q 31.
1	OUI	X
2	NON	

Le code suivant permet de lire une chaine de caractère dans la console d'exécution d'un programme Java :

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

		Q 32.
1	OUI	X
2	NON	

Le socket est un canal de communication permettant de faire communiquer deux JVM qui s'exécutent sur une même machine.		Q 33.
1	OUI	X
2	NON	

En JAVA, la classe ServerSocket est une classe qui hérite de Collection. Elle permet de gérer et stocker des sockets.		Q 34.
1	OUI	
2	NON	X

En JAVA, la méthode **accept** de la classe ServerSocket est en attente de la création d'un socket par une application distante. Elle retourne une instance de la classe Socket.		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.*

QUESTION 1 :

Expliquez comment il est possible de créer une collection polymorphe en Java.

Pour créer une collection polymorphe dont les éléments sont par exemple de type A, B ou C, il faut :

- déclarer la collection dont le type d'élément est d'une classe abstraite par exemple H ou d'une interface I
- les classes A, B et C doivent hériter de la classe abstraite H ou doivent implémenter l'interface I
- tous les traitements réalisés (traitements génériques) sur la collection polymorphe doivent utiliser les méthodes (abstraites ou non abstraites) définies dans la classe H ou les méthodes de l'interface I (et non les méthodes des classes filles de H).

QUESTION 2 :

Le principe de communication par socket entre deux applications Java repose sur les classes **Socket**, **ServerSocket**, **DataOuputStream**, **DataInputStream.**
Expliquez, dans ce contexte, le rôle de chacune de ces classes.

ServerSocket : classe qui permet de créer un objet qui va pouvoir se mettre en attente d'écriture d'un client dans un socket (méthode 'accept') sur un port TCP/IP.
Socket : classe permettant de créer un canal de communication sur le port du serveur de socket sur lequel il sera possible de lire et d'écrire des informations.
DataOutputStream : classe permettant d'écrire des informations de types informatiques (UTF, Int, Double, …) dans un socket
DataInputStream : classe permettant de lire des informations de types informatiques (UTF, Int, Double, …) depuis un socket

QUESTION 3 :

Expliquez le rôle de **l'ajout** d'une nouvelle méthode dans le graphe d'héritage de classes.
Expliquez le rôle de la **surcharge** d'une méthode dans le graphe d'héritage de classes.

Le rôle de l'ajout d'une nouvelle méthode dans le graphe d'héritage est d'étendre, sans les modifier, les propriétés des classes qui en héritent. Ces nouvelles propriétés sont utilisables à travers les instances de la classe qui définit cette nouvelle méthode et des classes qui en héritent. On parle de "généralisation".

Le rôle de la surcharge d'une méthode dans le graphe d'héritage est de remplacer, sans la modifier, une méthode dont on hérite. On parle de "spécialisation". La méthode de surcharge est accessible à travers les instances de la classe qui définit cette nouvelle méthode.

FIN DE LA 1ère PARTIE

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

PROBLEME [50 points]

1/

```
public class AbstractJoueur
{
  private String  ident;
  private String  mdp;

  public AbstractJoueur(String ident, String mdp)
  {
    this.ident=ident;
    this.mdp = mdp;
  }

  public String toString()
  {
    return String.format("%-30s %-20s ", ident,mdp);
  }}

public class Abonne extends AbstractJoueur
{
  private int dureeAbonnement;
  private String  dateAbonnement;

  public Abonne(String ident,
                String mdp,
                int dureeAbonnement,
                String dateAbonnement)
  {
    super(ident,mdp);
    this.dureeAbonnement = dureeAbonnement;
    this.dateAbonnement = dateAbonnement;
  }

  public String toString()
  {
    String str= super.toString();
    return str+String.format("%2d %8s ",
                             dureeAbonnement,
                             dateAbonnement);
  }}

public class Visiteur extends AbstractJoueur
{
  private String[] jeux;

  public Visiteur(String ident,
                  String mdp,
                  String[] jeux)
  {
    super(ident,mdp);
    this.jeux = jeux;
  }

  public String toString()
  {
    String str = super.toString();
    for(String s:jeux)
      str = str +" "+s;
    return str;
```

```
}}
```

2/

```
public void importerJoueur(String ficJoueurs)
{
  if (! ficJoueurs.equals("") )
    {
      String[] lignes = Terminal.lireFichierTexte("data/"+ficJoueurs);
      if (lignes == null) {return; //erreur }
      else
        {
          // Parcours de chaque ligne d'un joueur
          //
          for(String ligne:lignes)
            {
              String[] champs = ligne.split(";");
              if (champs[0].equals("ABONNE"))
                {
                  String ident = champs[1];
                  String mdp   = champs[2];
                  int duree = Integer.parseInt(champs[3]);
                  String date = champs[4];

                  joueurs.add(new Abonne(ident,mdp,duree,date));
                }
              if (champs[0].equals("VISITEUR"))
                {
                  String ident = champs[1];
                  String mdp   = champs[2];
                  int nbJeux = champs.length-3;
                  String[] jeux = new String[nbJeux];
                  for(int i=0;i<nbJeux;i++) jeux[i]=champs[3+i];
                  joueurs.add(new Visiteur(ident,mdp,jeux));
                }}}}}
```

3/

```
public String listerJoueurs()
{
  String res="";
  for(AbstractJoueur j : joueurs)
    res=res+"\n"+j.toString();
  return res;
}
```

4/

```
public ArrayList<AbstractJoueur> abonnesExpires()
{
  ArrayList<AbstractJoueur> lj = new ArrayList<AbstractJoueur>();

  for(AbstractJoueur j:joueurs)
    if (j instanceof fr.cnam.projet.Abonne)
      {
        Abonne ab = (Abonne)j;
        if ( (differenceJours(ab.getDateAbonnement(),aujourdhui())/30)
               > ab.getDureeAbonnement())
          lj.add(ab);
      }
  return lj;
}
```

(Fin du sujet)