IPST-CNAM
Programmation JAVA
NFA 032
Mercredi 21 Juin 2017

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.

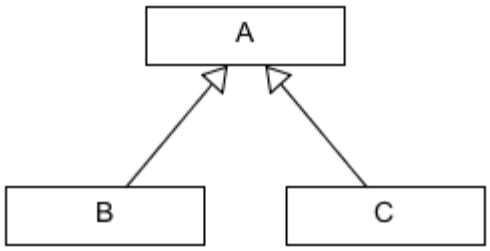
N'oubliez pas d'inscrire en en-tête du QCM, votre nom et prénom.

Vous avez droit à **4 points** négatifs sans pénalité.

NOM:	PRENOM:

En programmation objet, le principe d'héritage permet de :		Q 1.
1	factoriser les attributs de plusieurs classes en une classe dont elles héritent	X
2	spécialiser une classe en créant une classe dérivée qui surcharge certaines des méthodes de la classe d'héritage	X
3	rendre plus performant en terme d'exécution le programme objet	

Par définition de l'héritage, des classes peuvent hériter d'une classe abstraite.		Q 2.
1	OUI	X
2	NON	

Soit le symbolisme UML suivant :		Q 3.
 Cela signifie que B et C héritent de A.		
1	OUI	X
2	NON	

Soit les classes A et B qui héritent de la classe C. A et B contiennent la déclaration d'un même attribut. (même type et même nom). Cela est une erreur de programmation et déclenche une erreur de compilation car cet attribut doit se trouver dans la classe C.		Q 4.
1	OUI	
2	NON	X

Une classe A peut accéder à un attribut d'une autre classe B (sans utiliser de getteur) quand :		Q 5.
1	A hérite de B et que cet attribut est déclaré protected	X
2	cet attribut est public	X
3	A hérite de B et que cet attribut est déclaré private	

Soit la classe A qui hérite de B, alors A a accès à tous les constructeurs publics de B		Q 6.
1	OUI	X
2	NON	

On a une classe A qui contient deux constructeurs :		Q 7.
	`public A(int x){ }` `public A(){ }`	
On a la classe B qui hérite de A, qui contient le constructeur suivant :		

	`public B(){`
	`  attr = 100;`
	`}`	

1	Le code de la classe B est correct	X
2	Le code de la classe B n'est pas correct	

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 de B une erreur d'exécution se produit car A et B n'ont pas de constructeur		Q 8.
1	OUI	
2	NON	**X**

En programmation objet, une collection polymorphe est une collection dont les éléments sont d'un type donné. En JAVA, ce type peut être :		Q 9.
1	une classe quelconque	**X**
2	une classe abstraite	**X**
3	une liste de classes qui héritent toutes d'une même classe abstraite	

Il est impossible d'instancier une classe abstraite.		Q 10.
1	OUI	**X**
2	NON	

Une classe abstraite peut avoir un constructeur		Q 11.
1	OUI	**X**
2	NON	

Tous les attributs d'une classe abstraite doivent être **protected**.		Q 12.
1	OUI	
2	NON	**X**

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 13.
1	ArrayList<I> elements;	**X**
2	ArrayList<H> elements;	**X**
3	ArrayList<A,B,C> elements	

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 :
`elements.add( new A() )`
`elements.add( new B() )`
`elements.add( new C() )`
		Q 14.

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 15.
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**

En JAVA, on déclare une exception en créant une classe qui hérite de la classe **Exception**		Q 16.
1	OUI	**X**
2	NON	

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

Soit le code suivant :

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

Ce code est correct.		Q 18.
1	OUI	**X**
2	NON	

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

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

avec
`MyException` une classe qui hérite de `Exception`

		Q 19.
1	OUI	**X**
2	NON	

La déclaration de la méthode suivante :

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

précise que la méthode doit capturer l'exception MyException dans le corps de sa méthode.

		Q 20.
1	OUI	
2	NON	**X**

Soit le code suivant :

```
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 :		Q 21.
1	AAA CCC	**X**
2	AAA CCC DDD	
3	AAA DDD	

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	Comparer	

En JAVA, une collection est une classe qui implémente, directement ou indirectement, l'interface Collection<E>		Q 23.
1	OUI	**X**
2	NON	

En JAVA, toutes les collections ont la propriété d'itération. C'est-à-dire qu'il existe un moyen de parcourir les éléments de la collection.		Q 24.
1	OUI	**X**
2	NON	

La classe ArrayList hérite de la classe abstraite AbstractList. La classe HashSet hérite de la classe abstraite AbstractSet. La classe AbstractList gère des collections dont les éléments sont ordonnés suivant un index d'insertion. La classe AbstractSet gére des collections dont les éléments ne sont pas ordonnés et n'ont pas d'index d'insertion.		Q 25.
1	OUI	**X**
2	NON	

Une collection polymorphe est une collection qui hérite des 3 interfaces List, Set et Map		Q 26.
1	OUI	
2	NON	**X**

En Java, une collection polymorphe de contact est une collection qui est créée de la manière suivante :		Q 27.
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, une liste chainée (ex: classe LinkedList) est une liste dont les éléments ne sont pas continus dans la mémoire de la JVM..		Q 28.
1	OUI	**X**
2	NON	

En Java, pour écrire dans un fichier ou pour écrire dans un Socket, on peut utiliser les mêmes méthodes d'écriture et de lecture.		Q 29.
1	OUI	**X**
2	NON	

En Java, la classe File permet, entre autre, de parcourir les fichiers contenus dans un répertoire.		Q 30.
1	OUI	**X**
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
`FileInputStream fis = new FileInputStream(new File("exemple.txt"));`
`DataInputSream dis = new DataInputStream(fis);`
`dis.close()`
		Q 31.

1	OUI	**X**
2	NON	

Soit un serveur qui réalise le traitement suivant :
`ServerSocket ssoc = new ServerSocket(9100);`
`Socket soc = ssoc.accept();`
`DataInputStream dis= new DataInputStream(soc.getInputStream());`
`System.out.println(dis.readUTF());`
 	
Pour afficher sur le serveur la valeur "HELLO", le client doit écrire le code suivant :
		Q 32.

1	Socket soc = new Socket("localhost",9100);
	OutputStream os=soc.getOutputStream();
	PrintStream ps=new PrintStream(os);
	ps.print("HELLO");
	soc.close();	

2	Socket soc = new Socket("localhost",9100);
	OutputStream os=soc.getOutputStream();
	DataOutputStream dos=new DataOutputStream(os);
	dos.writeUTF("HELLO");
	soc.close();
		X

En JAVA, le socket est une technologie qui permet, sur une même machine, d'échanger des informations entre deux programmes (JVM) Java.		Q 33.
1	OUI	**X**
2	NON	

Soit un programme Java P1 (JVM) qui crée un serveur de socket sur le port 9100. Soit un programme Java P2 (JVM) qui crée un serveur de socket sur le port 9100.		Q 34.
1	Les deux programmes peuvent s'exécuter sur la même machine	
2	Le programme P1 peut s'exécuter sur la machine A, et le programme P2 peut s'exécuter sur une autre machine B	**X**

En JAVA, sur un socket, il est possible d'écrire n'importe quel type d'information (type primitifs, types référence)		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 les principes importants qui permettent de sauvegarder une collection polymorphe dans un fichier.

Une collection polymorphe contient des éléments de classes différentes mais qui héritent toutes de la classe de déclaration des éléments de la collection polymorphe.

Les principes importants qui permettent de sauvegarder une collection polymorphe sont :

- une collection polymorphe contient des éléments de classes différentes mais qui héritent toutes de la classe de déclaration des éléments de la collection polymorphe
- chaque classe écrit ses propres attributs dans le fichier (méthode write par exemple) et on utilise le lien d'héritage (super) pour réaliser la sauvegarde de tous les attributs hérités de l'objet
- avant d'écrire tous les attributs de l'objet, il faut écrire le type de l'objet afin que lors de la lecture on puisse connaître quel type d'objet on doit créer pour lire ses attributs

QUESTION 2 :

Soit la méthode suivante de la classe prédéfinie java Collection :
public static void sort(List<T> list, Comparator<? super T> c)
Expliquez le fonctionnement de cette méthode.

Cette méthode réalise le tri de la collection *list* passé en paramètre.
Le deuxième paramètre est un objet dont la classe d'appartenance implémente l'interface Comparator. Cette interface contient une seule méthode : public int comparer(T e1, T e2) qui est appelé par l'algorithme de tri et qui réalise la comparaison deux à deux des éléments de la collection. A la charge du développeur d'écrire son algorithme de comparaison de deux éléments dans cette méthode.

QUESTION 3 :

Quel est le rôle de la classe ServerSocket de Java ? Expliquez son fonctionnement.

Le rôle de la classe ServerSocket de Java est de créer un serveur de socket, c'est-à-dire un serveur qui permet d'accepter des communications par socket en provenance d'un client (un autre programme java situé sur la même machine ou sur une autre machine accessible à travers le réseau.

Le serveur de socket ouvre un port de communication sur sa machine. Le client ouvre un socket sur ce port en désignant le serveur par son host.

Cette classe contient une méthode *accept* qui attend qu'un client ouvre un socket sur le serveur de socket. Le serveur peut alors lire les informations qui ont été écrites par le client, et peut lui répondre en écrivant à son tour ses informations que le client devra lire.

FIN DE LA 1ère PARTIE

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

PROBLEME 2 [50 points]

1/

Le code de la classe **IHMStock** :

```
public class IHMStock implements FormulaireInt
{
  private Site site;

  public IHMStock(Site site)
  {
    this.site      = site;
    Formulaire form = new Formulaire("Gestion du stock",this,900,400);
    form.addText("REFERENCE","Reference",true,"");
    form.addText("NOM","nom",true,"");
    form.addText("PRIX","prix",true,"0.0");
    form.addText("QUANTITE","quantite",true,"1");
    form.addButton("VALIDER","Valider");

    form.setPosition(300,0);
    form.addZoneText("RESULTATS","Resultats",
                     true,
                     "",
                     600,300);
    form.afficher();
  }

  public void submit(Formulaire form,String nomSubmit)
  {
    if (nomSubmit.equals("VALIDER"))
      {
        String reference = form.getValeurChamp("REFERENCE");
        String nom = form.getValeurChamp("NOM");
        String prix = form.getValeurChamp("PRIX");
        String quantite  = form.getValeurChamp("QUANTITE");

        if (prix.equals("")) prix="0.0";

        int q = 0;
        double p = 0.0;
        try{
          q = Integer.parseInt(quantite);
          p = Double.parseDouble(prix);
          site.miseAJourStock(reference,nom,p,q);
          form.setValeurChamp("RESULTATS",site.listerTousProduits());
        }catch(Exception ex)
          {
            form.setValeurChamp("RESULTATS","Erreur de saisie");
          }
      }
}
```

Impact sur la classe Site :

```
   public void miseAJourStock(String reference,String nom,double prix,int
quantite)
  {
    Produit p = chercherProduit(reference);
    if (p!=null)
      {
        if (!nom.equals("")) p.setNom(nom);
        if (prix!=0.0) p.setPrix(prix);
        p.setQuantite(p.getQuantite()+quantite);
      }
    else
      stock.add(new Produit(reference,nom,prix,quantite));
  }
```

2/

Impact dans la classe Produit :

Il faut que la classe Produit implémente l'interface Comparable :

```
public class Produit implements Comparable<Produit>
```

Il faut implémenter la méthode compareTo :

```
  public int compareTo(Produit p)
  {
    if (this.quantite < p.getQuantite() )
      return -1;
    else
      return +1;
  }
```

Impact dans la classe Site :

Le traitement demandé :

```
public String trierParQuantite()
  {
    Collections.sort(stock);
    return listerTousProduits();
  }
```

3/

Impact dans la classe Site :

Le traitement demandé :

```
public String creerCommandeFournisseur(int seuil)
  {
    try{
      File file = new File("data/ComForunisseurs.txt");
      FileOutputStream fich = new FileOutputStream(
                    file.getAbsolutePath());
      PrintStream flot = new PrintStream(fich);

      for(Produit p : stock)
        {
          if (p.getQuantite()<=seuil)
            flot.println(p.getReference()+" "+p.getQuantite());
        }
      return “Le fichier data/ComForunisseurs.txt a été cree ») ;
    }catch(Exception ex){return “Erreur dans le traitement”};
  }
```