IPST-CNAM
Programmation JAVA
NFA 032
Mercredi 24 Juin 2015

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: |
|---|---|

Soit le code suivant ::

```
public class A extends C
{
  public void faire() {
    B b;
    b.x = 100;
    y = 200;
  }
}
```

Les classes A, B et C sont toutes des classes public.

| Q 1. | | |
|---|---|---|
| 1 | si x est un attribut public de B et y est un attribut protected de C alors ce code est correct | **X** |
| 2 | si x est un attribut public de B et y est un attribut private de C alors ce code est correct | |
| 3 | si x est un attribut public de B et y est un attribut public de C alors ce code est correct | **X** |

| Quand la classe A hérite de B, alors : | | Q 2. |
|---|---|---|
| 1 | A a accès à tous les constructeurs public de B | **X** |
| 2 | A a accès à tous les attributs private de B | |
| 3 | A a accès à toutes les méthodes private de B | |

On a une classe A qui contient qu'un seul constructeur :

```
public A(int x){ }
```

On a la classe B qui hérite de A, qui contient le constructeur suivant :

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

| Q 3. | | |
|---|---|---|
| Ce code est correct | | |
| 1 | OUI | |
| 2 | NON | **X** |

| Soit une classe B qui hérite d'une classe A, et B n'a pas de constructeur alors : | | Q 4. |
|---|---|---|
| 1 | lors de la création d'un objet de type B, une erreur d'exécution se produit si la classe A n'a pas défini de constructeur | |
| 2 | lors de la création d'un objet de type B, les attributs privés de A sont alloués en mémoire et initialisés par le constructeur hérité de A | **X** |
| 3 | Il n'est pas possible de créer un objet de B car la classe B n'a pas de constructeur | |

| Soit la classe C2 qui hérite de la classe C1.<br>Si C1 définit une méthode alors C2 peut définir la même méthode | | Q 5. |
|---|---|---|
| 1 | OUI | **X** |
| 2 | NON | |

| En programmation objet, une collection polymorphe est une classe qui hérite de plusieurs classes | | Q 6. |
|---|---|---|
| 1 | OUI | |
| 2 | NON | **X** |

| En JAVA, une classe peut hériter de plusieurs autres classes | | Q 7. |
|---|---|---|
| 1 | OUI | |
| 2 | NON | **X** |

| Si une classe B hérite d'une classe A alors : | | Q 8. |
|---|---|---|
| 1 | B hérite des méthodes privées de A | |
| 2 | lors de la création d'un objet de type B, les attributs de A sont alloués en mémoire et initialisés par le constructeur hérité de A | X |

| La caractéristique "protected" d'un attribut de la classe C est utilisée pour : : | | Q 9. |
|---|---|---|
| 1 | interdire l'écriture mais pas la lecture de l'attribut par les classes extérieures à C | |
| 2 | les classes qui héritent de C peuvent accéder, en lecture et écriture, à cet attribut mais pas les classes extérieures au package de C | X |

| Dans les langages orientés objet, le polymorphisme est la capacité pour un objet de se transformer en n'importe quel autre type d'objet | | Q 10. |
|---|---|---|
| 1 | OUI | |
| 2 | NON | X |

| Soit la classe C2 qui hérite de la classe C1 | | Q 11. |
|---|---|---|
| 1 | La classe C2 peut utiliser directement les attributs privés de C1 | |
| 2 | La classe C2 peut utiliser un des constructeurs de C1 | X |

| En Java, il est possible de faire l'héritage entre deux classes abstraites | | Q 12. |
|---|---|---|
| 1 | OUI | X |
| 2 | NON | |

| L'interface permet de : | | Q 13. |
|---|---|---|
| 1 | gérer des collections polymorphes (les éléments sont de type d'une interface) | X |
| 2 | gérer les fichiers (interface avec le système d'exploitation) | |
| 3 | passer en paramètre d'une méthode "un traitement" (traitements génériques) | X |

| En JAVA, une exception est un objet | | Q 14. |
|---|---|---|
| 1 | OUI | X |
| 2 | NON | |

Le code suivant est correct : Q 15.

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

| 1 | OUI | X |
|---|---|---|
| 2 | NON | |

Le code suivant est correct : Q 16.

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

| 1 | OUI | X |
|---|---|---|
| 2 | NON | |

| En JAVA, l'instruction suivante permet de déclencher une exception<br>`throw new Exception("Impossible de faire l'action");`<br>" | Q 17. |
|---|---|
| 1 OUI | X |
| 2 NON | |

| En JAVA, l'instruction suivante permet de déclencher une exception<br>`throw new RuntimeException("Impossible de faire l'action");` | Q 18. |
|---|---|
| 1 OUI | X |
| 2 NON | |

| Le code suivant est correct :<br>`public void action(int parameter)`<br>`{`<br>`  if (parametre==0)`<br>`  throw new MonException("action","parametre == 0");`<br>`  else`<br>`  faireLeTraitement();`<br>`}`<br>`avec`<br>`public class MonException extends Exception`<br>`{`<br>`  public MonException(String nomMethode,String erreur)`<br>`  {`<br>`  super("Erreur dans "+nomMethode+":"+erreur);`<br>`  }`<br>`}` | Q 19. |
|---|---|
| 1 OUI | |
| 2 NON | X |

| En Java, la classe Hashtable<K,V> permet de créer un tableau dont les éléments sont de type de la classe V qui hérite de K | Q 20. |
|---|---|
| 1 OUI | |
| 2 NON | X |

| En Java, la méthode suivante permet de trier les éléments d'une collection :<br>**Collection.*sort*(List<T> list, Comparator<? super T> c)** | Q 21. |
|---|---|
| 1 Comparator est ici une classe prédéfinie du langage Java qui permet de comparer les éléments de la collection | |
| 2 Comparator est ici une classe abstraite du langage Java qu'il faut dériver en une classe qui surcharge la méthode int compare(T o1, T o2) qui est utilisée pour comparer les éléments de la collection | |
| 3 Comparator est ici une interface qui contient la méthode int compare(T o1, T o2) qui est utilisée pour comparer les éléments de la collection | X |

| En Java, on déclare un tableau qui contient des éléments dont la classe d'appartenance implémente une interface I :<br>`I[] tab;` | Q 22. |
|---|---|
| 1 Cela est possible et on ajoute dans le tableau des objets de type I : t[i]=new I(); | |
| 2 Cela est possible et on ajoute dans le tableau des objets de type B : t[i]=new B(); avec la classe B qui implémente l'interface I | 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 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 | X |

| La sérialisation est un principe natif du langage JAVA permettant de plier et déplier les attributs des objets afin de pouvoir transporter les objets via un fichier ou un socket | | Q 24. |
|---|---|---|
| 1 | OUI | X |
| 2 | NON | |

| Soit le code suivant :<br>`public class Individu implements Serializable {String nom; String prenom;}`<br>Puis,<br>`Individu ind = new Individu();`<br>`File fic = new File(".","fic.bin");`<br>`FileOutputStream fin = new FileOutputStream(fic);`<br>`fin.writeObject((Object)ind);`<br>`fin.close() ;`<br>Ce code est correct | | Q 25. |
|---|---|---|
| 1 | OUI | |
| 2 | NON | X |

| En JAVA, créer un thread consiste à implémenter la méthode void run() dans une classe qui hérite de Thread | | Q 26. |
|---|---|---|
| 1 | OUI | X |
| 2 | NON | |

| Deux threads peuvent exécuter en même temps la même méthode d'un même objet et donc, par cela, modifier en même temps les mêmes attributs de l'objet | | Q 27. |
|---|---|---|
| 1 | OUI | X |
| 2 | NON | |

| Soit deux thread t1 et t2 qui utilisent la méthode objet définie ainsi<br>`synchronized public void miseajour(){. . .}`<br>t1 réalise l'appel suivant : o1.misaeajour();<br>t2 réalise l'appel suivant : o2.miseajour();<br>o1 et o2 sont deux objets différents.<br>Quand t1 est en train d'exécuter cette méthode de l'objet o1 alors t2 (qui veut aussi l'exécuter sur o1) est en attente que t1 ait fini de l'exécuter | | Q 28. |
|---|---|---|
| 1 | t2 est en attente que t1 ait fini d'exécuter la méthode | |
| 2 | t1 et t2 exécutent en parallèle la méthode | X |

| La méthode "start" est la méthode qui permet de démarrer un thread | | Q 29. |
|---|---|---|
| 1 | Cette méthode doit être implémentée dans la classe qui hérite de Thread | |
| 2 | Cette méthode appartient à la classe Thread. On y accède par héritage de la classe Thread | X |
| 3 | Cette méthode appelle la méthode run() que nous avons écrit dans notre thread | X |

| Le socket est un canal de communication permettant de faire communiquer deux logiciels | | Q 30. |
|---|---|---|
| 1 | Ces deux logiciels peuvent être situés sur deux machines différentes connectés en réseau | X |
| 2 | Ces deux logiciels peuvent être situés sur la même machine | X |
| 3 | Ces deux logiciels doivent être situés sur des machines différentes | |

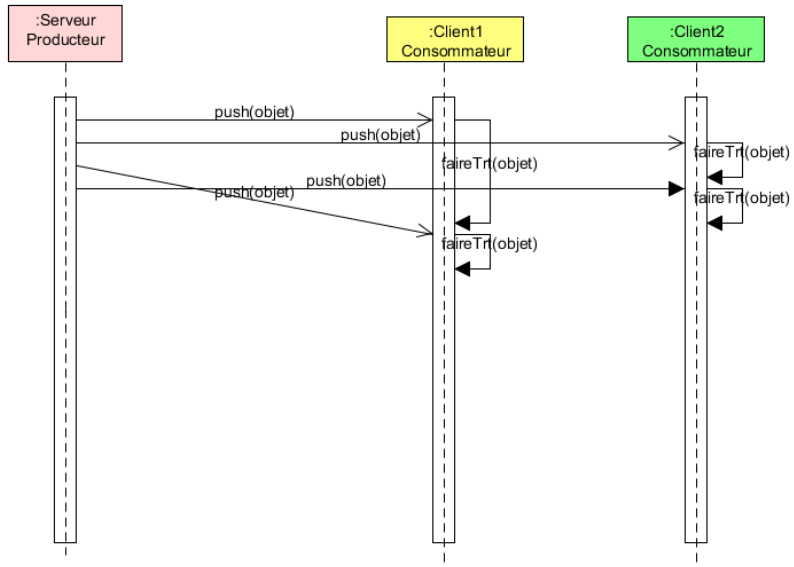
| En JAVA, la classe ServerSocket permet de créer un serveur de socket.<br>Le code peut être : | | Q 31. |
|---|---|---|
| 1 | `SeverSocket ssoc = new ServerSocket("localhost",9100)` | |
| 2 | `SeverSocket ssoc = new ServerSocket("localhost")` | |
| 3 | `SeverSocket ssoc = new ServerSocket(9100)` | X |

| En JAVA, les données échangées, via un socket, entre un client et un serveur, sont des données qui sont écrites et lues dans un flot séquentiel d'information | | Q 32. |
|---|---|---|
| 1 | OUI | X |
| 2 | NON | |

| Une communication synchrone entre un producteur et un consommateur est que | | Q 33. |
|---|---|---|
| 1 | la production des évènements est indépendante de leurs consommations | |
| 2 | la production des évènements est synchrone à leurs consommations | X |

| Le "mode push synchrone" est :<br>le producteur envoie une donnée à chacun des consommateurs sans se mettre en attente de la consommation de la donnée par les consommateurs.<br>Les consommateurs reçoivent alors la donnée, réalise leur traitement et n'envoie pas un accusé au producteur. | | Q 34. |
|---|---|---|
| 1 | OUI | |
| 2 | NON | X |

Q 35.




Ce diagramme de séquence correspond à un mode de communication :

| | | |
|---|---|---|
| 1 | push-synchrone | |
| 2 | push-asynchrone | 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.*

**Q 1**

Expliquez ce qu'est une classe abstraite et dans quel cas on l'utilise.

**Une classe abstraite est une classe qui contient au moins 1 méthode abstraite.**
**Les méthodes abstraites doivent être implémentées par les classes qui héritent de cette classe abstraite.**
**On ne peut pas créer un objet du type de la classe abstraite.**
**La classe abstraite permet de :**
**- créer des collections polymorphes.**
**- créer des méthodes génériques dont certaines méthodes utilisées sont des méthodes abstraites**
**- dans un langages orientées objet qui fait de l'héritage multiple et n'a pas d'interface comme le langage Java (ex C++), de créer des "interfaces" dans ce cas la classe abstraite n'a pas d'attribut et toutes ses méthodes sont abstraites.**

**Q 2**

Citez au moins 4 grands principes qu'il est nécessaire de respecter pour réaliser une communication client/serveur en utilisant un socket en JAVA.

**1/ Le serveur de socket doit s'exécuter sur un port qui ne soit pas déjà pris et sur lequel aucune interdiction d'utilisation n'est positionnée.**
**2/ Le client doit utiliser le même numéro de port que celui utilisé par le serveur.**
**3/ Le serveur doit lire sur le socket les informations telles qu'elles on été écrites par le client.**
**4/ Le client doit lire sur le socket la réponse telle qu'elle a été écrite par le serveur.**
**5/ Si le serveur attend un socket différent pour chaque requête alors le client doit créer un socket à chaque construction d'une requête.**

**Q 3**

Expliquez pourquoi il est avantageux d'utiliser un intermédiaire pour réaliser la communication avec des producteurs et des consommateurs d'évènements.

**Les avantages d'utiliser un intermédiaire sont :**
**- les producteurs ne connaissent pas l'existence des consommateurs. Seul l'intermédiaire connaît les consommateurs**
**- de même les consommateurs ne connaissent pas les producteurs.**
**- il est possible de créer plusieurs canaux de communication**
**- ainsi on peut réaliser une programmation évènementielle dont le couplage entre les producteurs et les consommateurs est paramétrable et adaptable.**

**(Tourner la page)**

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

## Problème [50 points]

```
// Classe principale d'execution du serveur
//
public class Serveur
{
    public static void main(String a_args[])
    {
        // Creation de l'annuaire
        Annuaire annuaire = new Annuaire();

        // Creation du serveur de socket
        new ServeurSocketAnnuaire(annuaire);
    }
}
```

```
// Classe de definition de l'annuaire
// Elle gere un tableau de Contact
//
public class Annuaire
{
    private ArrayList<Contact> elements;  // La liste des contacts

    // ------------------------------------------------
    // Constructeur
    //
    public Annuaire()
    {
        // Creation de la liste des rendez-vous
        elements = new ArrayList<Contact>();
    }

    // ------------------------------------------------
    // Methode pour ajouter un contact
    // Si le contact existe alors pas d'ajout et retourne false
    // Sinon ajoute le contact et retourne true
    //
    public boolean ajouterContact(String nom,
                                  String prenom,
                                  String telephone)
    {
        Contact contact = new Contact(nom,prenom,telephone);

        if (elements.contains(contact)) return false;
        elements.add(contact);
        return true;
    }

    // ------------------------------------------------
    // Methode qui recherche une chaine dans le nom et le prenom
    //  des contacts de l'annuaire
    // Retourne la liste des contacts trouves
    //
    public ArrayList<Contact> rechercherContatcs(String chaine)
    {
        String chainerech = chaine.toLowerCase();
        ArrayList<Contact> res = new  ArrayList<Contact>();
        for(Contact c : elements)
            {
                String nom = c.getNom().toLowerCase();
                String prenom = c.getPrenom().toLowerCase();
                if ( (nom.indexOf(chainerech)!=-1) ||
```

```
                    (prenom.indexOf(chainerech)!=-1) ||
                    (chaine.equals("")) )
                   res.add(c);
            }
        return res;
    }
}
```

```
// Classe de definition d'un Contact
//  implemente l'interface Comparable pour trier
//
public class Contact implements Comparable<Contact>
{
    private String nom;        // Le nom du contact
    private String prenom;     // Le prenom du contact
    private String telephone;  // Le telephone du contact

    //-------------------------------------------------------
    // Constructeur
    //
    public Contact(String nom,String prenom,String telephone)
    {
        this.nom=nom;
        this.prenom=prenom;
        this.telephone=telephone;
    }

    //-------------------------------------------------------
    // Methode equals utilise pour l'appel a la methode contains
    //  dans Annuaire
    // Deux contacts sont egaux si ils ont meme nom et meme
    //  prenom
    //
    public boolean equals(Object contact)
    {
        Contact c = (Contact)contact;
        return ( (c.nom.equals(nom)) && (c.prenom.equals(prenom)) );
    }
    //-------------------------------------------------------
    // Pour trier sur le nom puis si les noms sont egaux sur le prenom
    //
    public int compareTo(Contact c)
    {
        int ordre = this.nom.compareTo(c.getNom());
        if (ordre==0) // meme nom
            return this.prenom.compareTo(c.getPrenom());
        else
            return ordre;
    }

    //-------------------------------------------------------
    // Le contact sous la forme d'une chaine
    //
    public String toString()
    {
        return String.format("%20s %20s %15s",nom,prenom,telephone);
    }

    // Les getteurs
    //
    public String getNom(){return nom;}
    public String getPrenom(){return prenom;}
    public String getTelephone(){return telephone;}
}
```

```
// Classe qui cree un serveur de socket qui se met en
//  attente de l'ouverture d'un socket par un client et
//  traite la requete
// Le serveur de socket utilise le port en dur dans le code
```

```
//    = 9100
//
public class ServeurSocketAnnuaire
{
    //------------------------------------------------------
    // Constructeur
    //
    public ServeurSocketAnnuaire(Annuaire annuaire)
    {
        try{
            // Creation du serveur de socket
            ServerSocket  ssoc = new ServerSocket(9100);

            // Boucle sur l'acceptation d'un socket
            //
            while(true)
                {
                    // Attente d'acceptation d'un socket
                    //
                    System.out.println("En attente...");
                    Socket soc = ssoc.accept();
                    // Un socket a etabli une communication
                    System.out.println("Socket accepte");

                    // Recuperation des flots de lecture et d'ecriture du
socket
                    InputStream is = soc.getInputStream();
                    OutputStream os = soc.getOutputStream();
                    DataInputStream dis = new DataInputStream(is);
                    DataOutputStream dos = new DataOutputStream(os);

                    System.out.println("Lecture du socket");
                    String requete = dis.readUTF();
                    System.out.println("REQUETE: "+requete);

                    // ------
                    // REQUETE : RECHERCHER_CONTACT
                    //
                    if (requete.equals("RECHERCHER_CONTACT"))
                        {
                            // Lecture de "chaine"
                            String chaine = dis.readUTF();

                            // Recherche des contacts dans l'annuaire
                            ArrayList<Contact> res;
                            res = annuaire.rechercherContatcs(chaine);

                            // Ecriture des contacts trouves
                            dos.writeInt(res.size()); // Le nombre
                            for(Contact c:res)
                                {
                                    // Ecriture du contact
                                    dos.writeUTF(c.getNom());
                                    dos.writeUTF(c.getPrenom());
                                    dos.writeUTF(c.getTelephone());
                                }
                        }

                    // ------
                    // REQUETE : AJOUTER_CONTACT
                    //
                    if (requete.equals("AJOUTER_CONTACT"))
                        {
                            // Lecture du contact
                            String nom = dis.readUTF();
                            String prenom = dis.readUTF();
                            String telephone = dis.readUTF();

                            // Ajout du contact
                            boolean ok = annuaire.ajouterContact(
```

```
                                                nom,prenom,telephone);

                                // Ecriture de la reponse
                                if (ok)
                                    dos.writeUTF("OK");
                                else
                                    dos.writeUTF("NOK");
                            }
                        }
                }catch(Exception ex){};
        }
}
```

2/

```
// Classe de definition du client qui utilise le serveur
// Le host est en dur dans le code : localhost
// Le port est en dur dans le code : 9100
//
public class Client
{
    // ----------------------------------------------------
    // Programme principal
    //
    public static void main(String[] args) throws Exception
    {
        // Appel de la requete de recherche de contact qui retourne
        //  tous les contacts du serveur d'annuaire
        requeteRechercherContact("");
    }

    // ----------------------------------------------------
    //  Requete qui permet de recherche un contact sur le serveur
d'annuaire
    //
    private static void requeteRechercherContact(String chaine) throws
Exception
    {
        // Creation d'un socket
        Socket soc = new Socket("localhost",9100);
        InputStream is = soc.getInputStream();
        OutputStream os = soc.getOutputStream();
        DataInputStream dis = new DataInputStream(is);
        DataOutputStream dos = new DataOutputStream(os);

        // Ecriture de la requete
        //
        dos.writeUTF("RECHERCHER_CONTACT");
        dos.writeUTF(chaine);

        // Lecture de la reponse a la requete
        //
        // On lit le nombre de contact et on lit les contacts retournes
        // que l'on stocke dans un tableau temporaire afin de le trier
        // avant de l'afficher a l'ecran
        //
        ArrayList<Contact> contacts=new ArrayList<Contact>();
        int nb = dis.readInt();
        for(int i=0;i<nb;i++)
            {
                String nom = dis.readUTF();
                String prenom = dis.readUTF();
                String telephone = dis.readUTF();
                contacts.add(new Contact(nom,prenom,telephone));
            }
        // Tri du tableau temporaire
        Collections.sort(contacts);

        // Affichage du tableau temporaire
```

```
        System.out.println("Recherche de : "+chaine);
        for(Contact c:contacts)
            System.out.println(c.toString());
        System.out.println("----------------------------------------");

        soc.close();
    }
}
```

***(Fin du sujet)***