Examens JAVA

Programmation Orientee Objets (Java) (Université Moulay Ismaïl)

Année Universitaire : 2018/2019
Département d'Informatique
MIP/I (S5) : module I511
Le 12/06/2019

Contrôle de fin de semestre POO : Langage Java (durée 1h30mn)

Exercice 1 (7 points)

Voici la première ligne de la définition d'une classe ImageViewer

Public class ImageViewer implements ActionListner

Dans la documentation on trouve que **ActionListner** a une seule méthode appelée **actionPerformed().**

1. Qu'est-ce que **ActionListner** ?
2. Que doit contenir la classe **ImageViewer** pour qu'il n'y ait pas d'erreur à la compilation ?
3. On trouve souvent que les types dont le nom se termine par **Listner** ont un équivalent dont le nom se termine par **Adapter**. Exemple MouseListner/MouseAdapter, KeyListner/KeyAdapter …ce n'est pas le cas pour ActionListner, pourquoi ?
4. Le but de cette question est de créer une fenêtre contenant un bouton de label "Test" qui, lorsqu'il reçoit un clic, affiche "Test clic". Ecrire les instructions qui permet de

   a. Créer la fenêtre et le bouton.

   b. Créer une classe implémentant l'interface **ActionListener** et effectuant l'affichage dans la méthode dédiée.

   c. Relier le listener au bouton grâce à la méthode du bouton dédiée.

Exercice 2 (13points)

On cherche à écrire au cours de ce problème les classes nécessaires à la gestion de la facturation d'une entreprise qui vend des produits.

1. Ecrire une classe **Produit** permettant de modéliser les produits vendus par cette entreprise. Un produit est caractérisé par sa description, sa référence et son prix. Encapsuler ces caractéristiques ;

   a. Ecrire un constructeur pour cette classe.

   b. Redéfinir la méthode toString() de manière judicieuse.

   c. Redéfinir la méthode equals: 2 produits seront égaux si leurs références sont identiques.

2. Un **client** peut être soit une personne physique, soit une personne morale (entreprise, association, collectivité, locale)

   a. Ecrire une classe abstraite **Client** qui est caractérisée par un code client, une adresse, un code postal et une ville.

   b. Ecrire une classe **ClientPersonnePhysique** qui hérite de la classe Client, avec comme paramètre supplémentaire un nom et un prénom et un constructeur et la methode toString().

3. Au cours d'un achat, un client peut acquérir plusieurs produits, et chaque produit dans des quantités différentes. Une ligne de facture détermine la quantité d'un même produit acquis au cours d'un achat.

a. Ecrire une classe **LigneFacture** qui désigne le produit acquis par un client, la quantité achetée et le montant de cet achat (il faut enregistrer ce montant dans la ligne de facture, car le prix du produit peut varier dans le temps).

b. Redéfinir la méthode toString() de manière judicieuse.

4. Une facture est composée des lignes de factures correspondant à l'achat d'un client, à une date donnée.

a. Ecrire une classe Facture en choisissant judicieusement la structure de données et un constructeur adéquat.

b. Ecrire une méthode totalFacture() permettant de calculer le montant à payer par le client.

AU. : 2019/2020/Session d'Automne
Filière : LST Génie Logiciel, S5
Module : I511
Date : 30/01/2020

جامعة مولاي إسماعيل
UNIVERSITÉ MOULAY ISMAÏL

كلية العلوم والتقنيات
FACULTÉ DES SCIENCES ET TECHNIQUES

EXAMEN DE PROGRAMMATION ORIENTEE OBJET

(Durée : 2h)

Question du cours (1 pt)

Définir les concepts suivants : Polymorphisme, Classe abstraite.

Exercice 1 (3 pts)

1. Choisissez la mauvaise affirmation :
   (A) Une interface peut être le type d'une référence.
   (B) Une interface déclare des méthodes sans les implémenter.
   (C) Une interface est instanciable.
2. Combien d'instances, de la classe A, crée le code suivant ?

```
A x,y,z;
x=new A();
y=x;
z=new A();
```

   (A) Aucune
   (B) Deux
   (C) Trois

3. Pour les classes A et B définies comme suit :

```
class A {
 public int f() {return(5);}
 public static int g() {return (6);}}

class B extends A {
 public int f() {return(2);}
 public static int g() {return (4);}}
```

Qu'affichera le code suivant ?

```
B b=new B(); A a =b;
System.out.println(a.f()*a.g());
```

   (A) 30
   (B) 20
   (C) 12

Exercice 2 (4 pts)

Soit la classe `Point` définie comme suit :

```
public class Point{
  private int x, y;
  public Point(int x, int y) {this.x = x ; this.y = y ;}
  public void affCoord(){System.out.println ("Coordonnées : " + x + " " + y) ;}
}
```

1. Ajoutez deux méthodes pour déplacer un point d'une longueur sur l'axe des x et des y. `deplace(int dx, int dy)` dans le cas où `dx` et `dy` sont différents et `deplace(int dx)` dans le cas contraire. Quel est le concept utilisé ici ?
2. Donnez le code de la classe `PointNom`, dérivée de la classe `Point` permettant de manipuler des points définis par deux coordonnées (`int`) et un nom (`char`). Ajoutez les méthodes suivantes :
   a. Le constructeur de la classe `PointNom`,
   b. `affCoord` : redéfinissez la méthode `affCoord` de la classe mère pour afficher les coordonnées et le nom d'un objet de type `PointNom`.

Exercice 3 (12 pts)

Nous souhaitons réaliser un programme pour la gestion des filières de notre établissement. Les classes principales de ce programme sont : `Professeur` et `Etudiant` dérivées de la classe `Personne` et une autre classe `Filière`. Pour cette raison :

1. Créez une classe abstraite nommée `Personne` qui possède les attributs privés `nom` et `prénom`, un seul constructeur qui initialise le nom et le prénom d'une personne et une méthode abstraite `afficher()` pour afficher les informations d'une personne.
2. Ajoutez la sous-classe `Professeur` avec les attributs privés suivants : `dept` (département du professeur) et `listFilieres` qui regroupe toutes les filières dont ce professeur est responsable, ensuite ajoutez les méthodes suivantes :

a) Le constructeur de la classe `Professeur`.

b) `ajouterFiliere()` pour ajouter une filière à la liste `listFilieres`, et `supprimerFiliere()` pour la supprimer de cette liste. Un message sera affiché dans le cas où la liste est vide.

c) `afficher()` : affiche les informations du professeur selon le format suivant : `Le professeur np1 pp1, Informatique` (*voir le tableau ci-dessous*).

3. Ajoutez la sous-classe `Etudiant`, qui contient un attribut privé `filière` (Filière de l'étudiant), un constructeur qui initialise, seulement, le nom et le prénom de l'étudiant et l'implémentation de la méthode `afficher()` qui affiche les informations d'un étudiant comme suit : `L'étudiant n1 p1` (*voir le tableau ci-dessous*).

4. Ajoutez la classe `Filière` qui contient les attributs privés suivants `code` (de type `long`), `libelle`, `responsable` (de type `Professeur`), l'attribut public `listEtudiants` pour regrouper les étudiants de la filière et un attribut public et statique `listFilieres` pour regrouper *toutes les filières de l'établissement*. Ensuite, ajoutez les méthodes suivantes :

   a) Le constructeur de la classe `Filière`.

   b) `getResp()` : affiche les informations du responsable de la filière.

   c) `setResp()` : modifie le responsable de la filière et met à jour la liste des filières de l'ancien et de nouveau responsable.

   d) `ajouterEtudiant()` pour ajouter un étudiant à la liste des étudiants de la filière et `supprimerEtudiant()` pour le supprimer de cette liste.

   e) `afficherF()` : affiche les informations d'une filière sous cette forme :

```
Filière:534, GL
    Responsable:Le professeur np1 pp1 , Informatique
      Liste des etudiants:
                1- L'étudiant  n1 p1
                2- L'étudiant  n2 p2
```

5. Ajoutez une méthode `s_inscrire(Filière f)`, de la classe `Etudiant`, permettant à un étudiant de s'inscrire à une filière donnée. Dans cette méthode, on vérifie l'existence de l'étudiant dans *les listes des étudiants* (`listEtudiants`) *de toutes les filières*. Si l'étudiant est trouvé dans une de ces listes, la méthode lancera une exception `AjoutEtudExcept` que vous devez créer en y ajoutant une méthode d'affichage `messageErreur()`.

6. Dans une classe de test `GestionFilieres`:

   a) Créez les objets suivants :

Filière (`code, libellé`)	Responsable (`nom, prénom, dept`)	Etudiants (`nom, prénom`)
534,"GL"	"np1", "pp1", "Informatique"	- "n1", "p1" - "n2", "p2"
567,"MIP"		- "n3", "p3" - "n4", "p4"
987,"MA"	"np2", "pp2", "Math"	- "n5", "p5" - "n6", "p6"

   b) Appelez la méthode `s_inscrire()` pour ajouter chaque étudiant à sa filière.

   c) Affichez les informations de toutes les filières de l'établissement.

Bonne chance

Année Universitaire: 2015/2016
Département d'Informatique
MIP/I (S5): module I511
Le 10/02/2016

Contrôle de fin de semestre POO : Langage Java (durée 2h)

Exercice 1 (6 points)

Question 1 : (1 pt) Pour le compilateur Java, l'instruction: `A a = (B) b ;` est correcte si:

a) la classe B est une sous-classe de A

b) la classe B est une superclasse de A

c) le type déclaré de b est une sous-classe de B

d) le type déclaré de b est une superclasse de B

Question 2 : (1 pt) Un attribut statique est aussi appelé :

a) variable d'instance

b) variable de classe

c) variable d'interface

d) variable locale

Question 3 : (2 pt) Soient les deux classes d'exception suivantes :

```
class Exc1 extends RuntimeException { .... }
class Exc2 extends IOException { .... }
```

Soit la classe suivante:

```
public class TestException{
        public void f1(){ throw new Exc1(); }
        public void f2(){throw new Exc2(); }}
```

Expliquer pourquoi f1() se compile sans problème, tandis qu'une erreur de compilation se produit lors de l'analyse de f2() ? Proposer une solution pour résoudre ce problème ?

Question 4 : (1 pt) Indiquer si l'affirmation est vraie ou faux :

Faux	VRAI	Affirmation
	X	Une classe abstraite (déclarée abstract) ne peut pas être instanciée
	X	Le receveur d'un appel de méthode peut être de type primitif
	X	Un champ static indique que le champ est commun à toutes les instances de la classe.
X		Une méthode définie private est accessible dans toutes les classes appartenant au même package
X		La classe Object est une sous-classe de toutes les classes que l'on peut définir.

Question 5 : (1 pt) Pour les classes A et B définies comme suit:

```
class A {
public int x;
public A() {x=5; }}
```

```
class B extends A {
    public B() {x++;}
    public B(int i){this(); x=x+i; }
    public B(String s){super();x--;}}
```

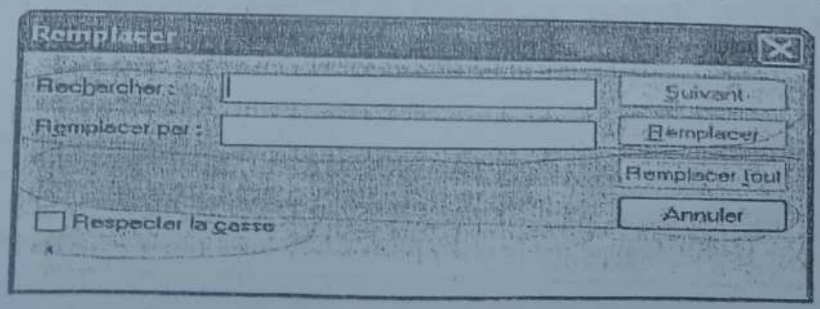
Qu'affichera le code suivant ? Justifier votre réponse

```
B b1=new B();  B b2 =new B(2003); B b3= new B("Bonjour");
System.out.println(b1.x +" et"+ b2.x+"et encore"+ b3.x );
```

Exercice 2 (14points)

Nous disposons d'un parking permettant de garer des voitures. Les voitures seront stationnées sur des places numérotées en partant de 0. Le nombre de place est fixé une fois pour toutes à la construction du parking. Pour se fait, en créant d'abord une classe voiture complète. La classe voiture comporte les attributs suivants : nom du propriétaire, marque, type et nombre de chevaux et les méthodes suivantes : constructeur initialisant tous les attributs, constructeur sans paramètres, les modificateurs, les accesseurs et la méthode String **toString()**.

1. Ecrire la classe **Voiture**.
2. Ecrire les trois classes d'exceptions suivantes: **PlaceNonLibreException**, **PlaceLibreException, HorsParkingException**.
3. Ecrire la classe **Parking** avec un constructeur adéquat sachant que la classe Parking admet comme attribut : un tableau de voiture et un nombre de place limité.
4. Ecrire une méthode **garer** qui prend en paramétré une voiture, un numéro de place et qui gare la voiture à la place donnée si elle est libre, et lève des exceptions de type **PlaceNonLibreException** et **HorsParkingException** (déclenchée lorsqu'un numéro de place n'est pas valide.)
5. Ecrire une méthode **sortir** qui prend en paramétré un numéro de place et qui retire du garage la voiture à cette place. La méthode retourne l'objet voiture récupéré. Elle lève une exception de type **PlaceLibreException**.
6. Ecrire une méthode **toString** affichant l'état du garage: Pour chaque place, le numéro, ainsi que les caractéristiques de la voiture qui l'occupe si elle existe.
7. Ecrire la classe **TestPrking** qui permet de tester la classe **Parking**. (appeler les 3 méthodes)
8. L'interface ci-dessous contient les objets Swing des classes : public JTextField(int colmns), public JButton(String text), public JCheckBox(String text) public JLabel(String);



Réaliser l'ensemble des classes nécessaires pour obtenir cette interface avec précision des différents panneaux et layout managers. Sachant que:

- le panneau conteneur reçoit en paramètres de son unique constructeur une chaine de caractère type StringBuffer (classe héritant de JPanel).
- la classe contenant le panneau conteneur est une boite de dialogue dont l'un des constructeur est: **public JDialog(Frame owner, String title, boolean modal);**

جامعة مولاي إسماعيل
UNIVERSITÉ MOULAY ISMAÏL

كلية العلوم والتقنيات
FACULTÉ DES SCIENCES ET TECHNIQUES

Année Universitaire : 2016/2017
Département d'Informatique
MIP/I (S5) : module I511
Le 04/02/2017

Contrôle de fin de semestre POO : Langage Java (durée 2h00mn)

Exercice 1 (8 points)

Voici la première ligne de la définition d'une classe ImageViewer

Public class ImageViewer implements ActionListner

Dans la documentation on trouve que **ActionListner** a une seule méthode appelée **actionPerformed()**.

1. Qu'est-ce que **ActionListner** ?
2. Que doit contenir la classe **ImageViewer** pour qu'il n'y ait pas d'erreur à la compilation ?
3. On trouve souvent que les types dont le nom se termine par **Listner** ont un équivalent dont le nom se termine par **Adapter**: Exemple MouseListner/MouseAdapter, KeyListner/KeyAdapter …ce n'est pas le cas pour ActionListner, pourquoi ?
4. Le but de cette question est de créer une fenêtre contenant un bouton de label "Test" qui, lorsqu'il reçoit un clic, affiche "Test clic". Ecrire les instructions qui permet de

   a. Créer la fenêtre et le bouton.
   b. Créer une classe implémentant l'interface **ActionListener** et effectuant l'affichage dans la méthode dédiée.
   c. Relier le listener au bouton grâce à la méthode du bouton dédiée.

Exercice 2 (12 points)

L'objectif de cette application est la gestion informatisé d'un cabinet médical. Elle doit procéder à la gestion informatisée des dossiers médicaux des patients et elle s'occupe de la gestion des rendez-vous et des visites.

1. Définir une classe **Patient** dont les caractéristiques sont : le code du patient (affecté de façon incrémentale par rapport au nombre de patients), nom, prénom, adresse, date de naissance. Et les méthodes suivantes : Un constructeur avec tous les paramètres, les accesseurs des champs et la méthode **toString** () qui renverra les informations du patient.

2. Définir une classe **Visites** dont les caractéristiques sont Date visite, Heure visite, code patient, montant payé. Et les méthodes suivantes : un constructeur à deux paramètres « Date visite et Heure visite », un constructeur sans paramètres qui appelle le premier pour la date de visite et l'heure de visite, les accesseurs des champs et la méthode **toString** () qui renverra les informations de cette visite.

3. Définir une classe **RendezVous** dont les caractéristiques sont Date Rendez-vous, Heure rendez-vous, code patient et Observation. Et les méthodes suivantes : un constructeur à trois paramètres « Date Rendez-vous, Heure rendez-vous, code patient », les accesseurs des champs et la méthode **toString**() qui renverra les informations de ce rendez-vous.

4. classe **CabinetMedical**

a. Définir une classe **CabinetMedical** dont les caractéristiques sont : tableau des patients, tableau des visites, tableau des rendez-vous.

b. Ajouter une méthode **ajouterPatient** qui ajoute un patient à l'ensemble des patients du cabinet.

c. Ajouter une méthode **patientExistant** ayant comme paramètres le nom et le prénom et qui retourne le code du patient s'il existe et le chiffre « -1» si nom.

d. Définir une classe d'exception : **exceptionMedcinOccupé.**

e. Ajouter une méthode **ajouterRDV** qui ajoute un rendez-vous et qui lève l'exception définie en d).

f. Ajouter une méthode **afficherRDVDuJour** qui va pour une date donnée en paramétré afficher la liste des rendez-vous de ce jour.

g. Ajouter une méthode **patientAyantVistes** qui affiche le liste des patients (code, nom, prénom) ayant visité le cabinet pendant la dernière semaine.

h. Ajouter une méthode **supprimerPatient** qui permet de supprimer un patient ainsi que ces visites et rendez-vous.