JAVA 2015 DS 4ETI 1ère session : avec correction

Partie I :

Question 1 : (1pt)

Combien d’instances de la classe A sont créées dans le code suivant (cocher la bonne réponse)?

A x, u, v;

x=new A( );

A y=x;

A z=new A( );

(a) Aucune

(b) Cinq

(c) Trois

(d) Deux

*Réponse : (d) deux instances de A sont créées par les deux instructions new, la première est référencée par x et y, la deuxième par z.*

Question 2 : barrer les affirmations qui sont fausses (1pt)

(a) Une classe peut implémenter plusieurs interfaces mais ne peut étendre qu’une seule classe.

(b) Une classe peut implémenter plusieurs classes mais ne peut étendre qu’une seule interface. *Faux*

(c) Une classe peut implémenter plusieurs classes et peut étendre plusieurs interfaces. *Faux*

(d) Une classe doit implémenter une seule interface et étendre une seule classe. *Faux*

Question 3 : (1pt)

La classe B est définie comme suit:

class B

{

public B( )

{

System.out.print(”Ciao”);

};

public B(int i)

{

this( );   
System.out.println(”Bonjour ”+ i );

};

}

Qu’affichera l’instruction suivante (cocher la bonne réponse)?

B monB=new B(2015);

(a) une erreur de compilation

(b) une erreur d’exécution

(c) CiaoBonjour 2015

(d) Bonjour 2015

*Réponse : (c) L’instruction invoque le constructeur avec un argument entier (2015). Ce dernier appelle explicitement le constructeur sans arguments (this() ) qui imprime ”Ciao”, et ensuite le message ”Bonjour 2015” est imprimé*

Question 4 : (1pt)

Les classes A et B sont 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 (cocher la bonne 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 );

(a) 6 et 2009 et encore 4

(b) 1 et 2004 et encore 4

(c) 1 et 2004 et encore 2003

(d) Une erreur d’exécution

*Réponse : (a) Le constructeur B() n’appelle explicitement ni this(), ni super(). Donc, par convention, c’est le constructeur de la super-classe A qui est appelé (implicitement) : b1.x=6.*

*Le constructeur B(2003) appelle le constructeur précédent avec le this() : b2.x=6. Ensuite on y ajoute 2003, ce qui donne finalement b2.x=2009.*

*Le constructeur B(”Bonjour”) appelle le constructeur de la super-classe A avec le*

*super() : b3.x=5. Ensuite on le décrémente et on a finalement b3.x=4.*

Question 5 (2 points) : répondre en 10 lignes maximum.

Rappeler la/les relations qui existent entre une interface et une classe qui l’implémente. A quelle condition une classe qui implémente une interface peut-elle être instanciée?

Qu’est-ce qu’une classe abstraite et à quoi peut-elle servir?

*Réponse : La relation entre une classe et une interface est une relation de concrétisation, ou d’implémentation : La classe a en charge de fournir une implémentation partielle ou totale des services (ou comportements) définis dans l’interface fonctionnelle publique.*

*Pour qu’une classe qui implémente une interface soit instanciable, elle doit fournir une implémentation pour l’ensemble des comportements déclarés dans l’interface fonctionnelle publique de l’interface.*

*Sinon c’est une classe abstraite : une classe abstraite fournit une partie du code pour l’implémentation des comportements définis dans l’interface qu’elle implémente.*

*L’intérêt des classes abstraites est de fournir une factorisation de code pour ses sous-classes, sans pour autant être instanciable directement.*

Question 6 : (2 points)

On considère les entités: Etudiant, Personne, Enseignant, EtudiantEnAlternance et EtudiantSportif, et les propriétés : nom, age, salaire, emploiDuTemps, anneDEtude, nomDepartement, nomDuService et sportPratique.

1. Dessinez une arborescence de classes cohérente, précisez le type de relation entre les différentes classes et placez les champs indiqués.
2. Justifiez vos réponses.

*Réponse : la classe Personne est la classe de base dont héritent les classes Etudiant et Enseignant. Les classes EtudiantSportif et EtudiantEnAlternance dérivent de la classe Etudiant.  
nom et age sont des champs de la classe Personne,  
salaire est un champ de la classe Enseignant et EtudiantEnAlternance.  
emploiDuTemps est un champ de la classe Etudiant, mais aussi de la classe Enseignant.*

Partie II (inspiré de deptinfo.cnam.fr )

Dans une banque, on veut représenter les comptes bancaires et les personnes titulaires de ces comptes. Un même compte peut avoir plusieurs titulaires (c’est le cas par exemple des comptes joints pour un couple). Dans ce cas, le même objet sera partagé entre les titulaires. Un titulaire peut avoir plusieurs comptes.

On veut tester une situation où trois titulaires, Paul, Kevin et Fatima ont des comptes dans une banque CNF. Paul et Fatima ont un compte joint mais Fatima a en plus un compte personnel, Kevin a seulement un compte personnel. Dans cette situation, on veut pouvoir :

1. enregistrer les 3 Titulaires,
2. créer les 3 comptes,
3. enregistrer des dépôts, des retraits et des virements sur ces comptes,
4. afficher les comptes des titulaires et le montant total de leurs avoirs.
5. afficher le bilan des sommes déposées dans cette banque.

Dans la conception, on choisit d'implémenter :

* une table associative (Map) de tous les comptes détenus dans la banque,
* pour chaque client un ensemble des comptes dont il est titulaire.
* pour chaque compte :
  + un ensemble de ses titulaires,
  + et une Map des mouvements (retrait et dépôt) faits sur le compte.

Vous compléterez le code donné ci-dessous en suivant les indications données.

On considèrera que :

* toutes les classes des packages Java sont correctement importées.
* les méthodes hashcode() et equals() nécessaires ont bien été écrites :
  + pour les comptes, elles utilisent les numéros de compte,
  + pour les titulaires, le nom des titulaires, en supposant que 2 titulaires ne peuvent pas avoir le même nom.

public class Compte {

private int numero;

private double solde;

private HashSet<Titulaire> mesTitu; // Ensemble de ses titulaires

private HashMap<Date, Double> mouvements;

/\*\* Constructeur  
 \* @param les titulaires

\* @param init solde initial

\*/

Compte(int num, ArrayList<Titulaire> tit, double init){

numero = num;

solde = init;

mesTitu = new HashSet<Titulaire>(tit);

mouvements = new HashMap<Date, Double>();

}

/\*\* Solde du compte

\* @return valeur courante du solde

\*/

public double getSolde() {

return solde;

}

/\*\* Numero du compte

\* @return numero du compte

\*/

public int getNum() {

return numero;

}

/\* Operations sur le compte \*/

/\*\* Depot du montant

\* @param n

\*/

public void deposer(double n){

solde = solde+n;

}

/\*\* Retrait du montant

\* @param m

\*/

public void retirer(double m) {

solde = solde-m;

}

}

//Insérer ici la réponse à la question 1

public String toString(){

String s= "Compte numero: "+ numero +" Solde: "+ solde;

return s;

}

}

class Titulaire{

private String nom; // Nom du titulaire

// Ensemble de ses comptes

private HashSet<Compte> mesComptes = new HashSet<Compte>( );

/\*\* Constructeur @param n nom du titulaire

\*/

public Titulaire(String n) {

nom =n;

}

/\*\* @return nom du titulaire \*/

public String getNom() {

return nom;

}

/\*\* ajouter un compte à ce Titulaire : fonction à écrire

\* @param le compte à ajouter

\*/

public void ajouterDans(Compte c){

mesComptes.add(c);

}

public void afficheTous() {

if (mesComptes.isEmpty( )) {

System.out.println("Aucun compte");

return;

}

System.out.println("Comptes de "+ this.nom + " :");

for (Compte c: mesComptes) {

System.out.println( c );

}

}

/\*\*QUESTION 8 : faire le bilan des comptes d'un Titulaire \*/

public double soldeTotal( ){

double s= 0;

for (Compte c: mesComptes) {

s = s + c.getSolde( );

}

return s;

}

}

public class Banque {

private String nom;

private int numComptes = 1; // Prochain numero a attribuer a la creation compte

private HashMap<Integer,Compte> tousComptes = new HashMap<Integer,Compte>();

/\*\* Constructeur

\* @param n nom de la banque

\*/

public Banque(String n) {

nom =n;

}

/\*\* Creer un nouveau compte avec   
 \* @param tit liste de titulaires  
 \* @param init solde initial

\*/

//Insérer ici la réponse à la question 4:   
 public int creerCompte(ArrayList<Titulaire> tit, double init){

int numc = numComptes;

Compte c = new Compte(numc,tit, init);

tousComptes.put(numc, c);

for (Titulaire t : tit) {

tousClients.put(t.getNom(), t);

t.ajouterDans(c);

}

numComptes++;

return numc;

}

/\*\* Teste si Question 5 : fonction à écrire

\* @param n correspond a un compte dans la banque

\* @return true si c'est le cas.

\*/

private boolean isNumCompte(int n) {

return tousComptes.containsKey(n);

}

/\*\* Obtenir le compte de numero Question 5 : fonction à écrire

\* @param n

\* @return ce compte s'il existe, null sinon

\*/

private Compte getCompteDeNum(int n){

return tousComptes.get(new Integer(n));

}

/\*\* Teste si Question 5 : fonction à écrire

\* @param n correspond au nom d'un client (titulaire) de la banque

\* @return true si c'est le cas.

\*/

private boolean isNomTit(String n) {

return tousClients.containsKey(n);

}

/\*\* Obtenir le titulaire de nom Question 5 : fonction à écrire

\* @param n nom du titulaire

\* @return ce titulaire s'il existe, null sinon

\*/

private Titulaire getTituDeNom(String n){

return tousClients.get(n);

}

/\*\* Obtenir le compte de numero @param n Question 5 : fonction à écrire

\* @return ce compte s'il existe, null sinon

\*/

private Compte getCompteDeNum(int n){

return tousComptes.get(new Integer(n));

}

/\* Depot

\* @param m le montant

\* @param num le numero du compte

\* @return true si depot effectue

\* false si compte inexistant

\*/

//Insérer ici la réponse à la question 6:

public boolean deposer(Date aujourdhui, int num, double m) {

if (!isNumCompte(num))

return false;

Compte c = getCompteDeNum(num);

c.deposer(aujourdhui, m);

return true;

}

/\* retrait

\* @param m le montant

\* @param num le numero du compte

\* @return true si retrait effectue

\* false si compte inexistant

\*/

//Insérer ici la suite de la réponse à la question 6:

public boolean retirer(Date aujourdhui, int num, double m) {

if (!isNumCompte(num))

return false;

Compte c = getCompteDeNum(num);

c.retirer(aujourdhui, m);

return true;

}

/\*\* Effectuer un virement

\* @param m le montant du virement

\* @param numS le numero du compte source

\* @param tit le titulaire

\* @param numD, le numero du compte

\* @return true si virement est parti de la banque

\*/

//Insérer ici la suite de la réponse à la question 6:

public boolean virerVers(double m, int numS, Titulaire tit, int numD){

if(!isNumCompte(numS) || !isNomTit(tit.getNom()) || !tit.hasNumCompte(numS))

return false;

Date aujourdhui = new Date();

DateFormat formatAffichDate = DateFormat.getDateTimeInstance(DateFormat.MEDIUM, DateFormat.MEDIUM);

System.out.println(formatAffichDate.format(aujourdhui));

retirer(aujourdhui,numS, m);

return deposer(aujourdhui, numD, m);

}

Question 1: (1 point)

Écrire la méthode nécessaire à l’affichage du solde d’un compte dont on affichera aussi le numéro.

Question 2 : (1 point)

Quelle est la visibilité du constructeur de Compte ? Pourquoi a-t-on choisi cette visibilité ?

Question 3: (1 point)

Quelles est la visibilité actuelle des variables d’instance des classes?

Comment faut-il modifier le code pour rendre privées toutes les variables d’instance ?

Question 4: (2 points)

Seule la banque peut créer des nouveaux comptes. Quand elle crée un nouveau compte, elle lui attribue un numéro exclusif, puis :

* elle l’ajoute à sa Map de comptes,
* elle l'ajoute à l'ensemble des comptes de chaque titulaire de ce compte,
* elle ajoute tous les titulaires à sa Map de titulaires.

Écrire les méthodes nécessaires à la création d’un compte par la banque et à l’ajout d’un compte chez un titulaire.

Question 5: (1 point)

Pour accéder aux attributs de la classe, on a besoin d'assesseurs (fonctions getXXX()). Lesquelles? Compléter les classes Compte et Titulaire.

Question 6: (3 points)

Un virement est une opération qui débite le compte d'un titulaire pour créditer un autre compte dont on ne connait que le numéro (le titulaire n'est pas forcément le même mais ce compte se trouve dans la même banque).

Après avoir

* vérifié si le compte source existe et appartient au titulaire qui demande le virement,
* vérifié l’existence du compte destinataire,

écrire les méthodes qui permettent, de faire le virement si ces conditions sont remplies:

* débiter le compte du titulaire du montant indiqué,
* puis le créditer sur le compte du destinataire.

Sinon on ne réalisera pas l’opération.

Question 7: (2 points)

Écrire une méthode dans la classe Banque permettant d'afficher le montant total des dépôts faits par un titulaire sur l'ensemble de ses comptes.

Question 8 : (1 point)

Dans ce programme, on veut utiliser au mieux les collections issues des bibliothèques java.utils.

En quoi le choix d’un HashMap pour stocker les comptes d’une banque, et celle d’un HashSet pour stocker les comptes d’un Titulaire est-il pertinent ?

???

/\*\* QUESTION 9 : classe Banque

\* @return solde cumule des dépôts de tous les comptes \*/

public double bilan( ){

double s= 0;

Set<Map.Entry<Integer, Compte>> sc = tousComptes.entrySet( );

for (Map.Entry<Integer, Compte> entry: sc) {

s = s + entry.getValue( ).getSolde( );

}

return s;

}

**DOCUMENTATION DE REFERENCE :**

**Interface Map<K,V>**

Type Parameters: K - the type of keys maintained by this map

V - the type of mapped values

**Method Summary**

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| void | [**clear**](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html#clear%28%29)()  Removes all of the mappings from this map (optional operation). |
| boolean | [**containsKey**](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html#containsKey%28java.lang.Object%29)([Object](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html) key)  Returns true if this map contains a mapping for the specified key. More formally, returns true if and only if this map contains a mapping for a key k such that (key==null ? k==null : key.equals(k)). |
| boolean | [**containsValue**](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html#containsValue%28java.lang.Object%29)([Object](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html) value)  Returns true if this map maps one or more keys to the specified value. More formally, returns true if and only if this map contains at least one mapping to a value v such that (value==null ? v==null : value.equals(v)). |
| [Set](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html" \o "interface in java.util)<[Map.Entry](http://docs.oracle.com/javase/7/docs/api/java/util/Map.Entry.html" \o "interface in java.util)<[K](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html" \o "type parameter in Map),[V](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html" \o "type parameter in Map)>> | [**entrySet**](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html#entrySet%28%29)()  Returns a [Set](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html" \o "interface in java.util) view of the mappings contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator's own remove operation, or through the setValue operation on a map entry returned by the iterator) the results of the iteration are undefined. |
| boolean | [**equals**](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html#equals%28java.lang.Object%29)([Object](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html) o)  Compares the specified object with this map for equality. |
| [V](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html) | [**get**](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html#get%28java.lang.Object%29)([Object](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html) key)  Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. |
| int | [**hashCode**](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html#hashCode%28%29)()  Returns the hash code value for this map. |
| boolean | [**isEmpty**](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html#isEmpty%28%29)()  Returns true if this map contains no key-value mappings. |
| [Set](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html)<[K](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html)> | [**keySet**](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html#keySet%28%29)()  Returns a [Set](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html) view of the keys contained in this map. |
| [V](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html) | [**put**](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html#put%28K,%20V%29)([K](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html) key, [V](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html) value)  Associates the specified value with the specified key in this map (optional operation). |
| int | [**size**](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html#size%28%29)()  Returns the number of key-value mappings in this map. |
| [Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)<[V](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html)> | [**values**](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html#values%28%29)()  Returns a [Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html) view of the values contained in this map. The collection is backed by the map, so changes to the map are reflected in the collection, and vice-versa. |

**Interface Set<E>**

Type Parameters: E - the type of elements maintained by this set

**Method Summary**

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| boolean | [**add**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html#add%28E%29)([E](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html) e)  Adds the specified element to this set if it is not already present (optional operation). |
| boolean | [**addAll**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html#addAll%28java.util.Collection%29)([Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)<? extends [E](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html)> c)  Adds all of the elements in the specified collection to this set if they're not already present (optional operation). If the specified collection is also a set, the addAll operation effectively modifies this set so that its value is the *union* of the two sets. The behavior of this operation is undefined if the specified collection is modified while the operation is in progress. |
| void | [**clear**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html#clear%28%29)()  Removes all of the elements from this set (optional operation). |
| boolean | [**contains**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html#contains%28java.lang.Object%29)([Object](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html) o)  Returns true if this set contains the specified element. |
| boolean | [**containsAll**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html#containsAll%28java.util.Collection%29)([Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)<?> c)  Returns true if this set contains all of the elements of the specified collection. If the specified collection is also a set, this method returns true if it is a *subset* of this set. |
| boolean | [**equals**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html#equals%28java.lang.Object%29)([Object](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html) o)  Compares the specified object with this set for equality. |
| int | [**hashCode**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html#hashCode%28%29)()  Returns the hash code value for this set. |
| boolean | [**isEmpty**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html#isEmpty%28%29)()  Returns true if this set contains no elements. |
| [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html)<[E](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html)> | [**iterator**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html#iterator%28%29)()  Returns an iterator over the elements in this set. |
| boolean | [**remove**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html#remove%28java.lang.Object%29)([Object](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html) o)  Removes the specified element from this set if it is present. More formally, removes an element e such that (o==null ? e==null : o.equals(e)), if this set contains such an element. Returns true if this set contained the element (or equivalently, if this set changed as a result of the call). (This set will not contain the element once the call returns.) |
| boolean | [**removeAll**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html#removeAll%28java.util.Collection%29)([Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)<?> c)  Removes from this set all of its elements that are contained in the specified collection (optional operation). If the specified collection is also a set, this operation effectively modifies this set so that its value is the *asymmetric set difference* of the two sets. |
| boolean | [**retainAll**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html#retainAll%28java.util.Collection%29)([Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)<?> c)  Retains only the elements in this set that are contained in the specified collection (optional operation). |
| int | [**size**](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html#size%28%29)()  Returns the number of elements in this set (its cardinality). |