

Examen de Programmation Java

mercredi 26 juin 2013

Examen papier. Durée 2H.

Aucun document sauf une feuille A4 recto-verso.

Pas d'ordinateur.

Sauf précision, la version de Java utilisée est Java SE 7.

On vous donne en annexe des extraits de la documentation Java SE 7.

Pour les exercices 1 à 4, justifier en une phrase chaque réponse.

Exercice 1

Q1. De combien de classe(s) peut hériter une classe ?

Q2. Combien d'interface(s) peut implémenter une classe ?

Q3. De combien d'interface(s) peut hériter une interface ?

Q4. Combien d'interface(s) peut implémenter une interface ?

Q5. L'utilisation de l'annotation `@Override` change-t-elle la compilation d'un programme par rapport à sa non utilisation ?

Q6. L'utilisation de l'annotation `@Override` change-t-elle l'exécution d'un programme par rapport à sa non utilisation ?

Q7. Pour chaque instruction ci-dessous, préciser pour quelle(s) version(s) de Java il y a une erreur de compilation. On se limitera aux versions de Java SE : 4, 6 et 7.

- a) `Set s1 = new HashSet();`
- b) `Set<Personne> s2 = new HashSet<>();`
- c) `Set<Personne> s3 = new HashSet<Personne>();`
- d) `// res de type boolean, fruit de type String`
`switch (fruit) {`
`case "banane": res = true; break;`
`case "fraise": res = true; break;`
`default: res = false;`
`}`
- e) `// res de type boolean`
`// fruit de type enum Fruit{banane,fraise,salade,poisson,pain}`
`switch (fruit) { String`
`case banane: res = true; break;`
`case fraise: res = true; break;`
`default: res = false;`
`}`

Exercice 2

Soit la classe `Personne` avec un nom (`String`) et un âge (`int`) pour laquelle il n'a pas été écrit de constructeur.

```
public class Personne {  
    private String nom;  
    private int age;  
}
```

Q1. Peut-on créer une instance de la classe `Personne` ? Si oui comment et que valent les attributs `nom` et `age` ?

Exercice 3

```
public class Voiture {
    public void accelerer() { /* non précisé */ }
}

public class VoitureSport extends Voiture {
    public void accelerer() { /* non précisé */ }
    public void accelerer(int q) { /* non précisé */ }
}
```

On considère les instructions suivantes utilisant ces 2 classes :

- a) `Voiture v1 = new Voiture();`
- b) `Voiture v2 = new VoitureSport();`
- c) `VoitureSport vs1 = new VoitureSport();`
- d) `VoitureSport vs2 = new Voiture();`
- e) `VoitureSport vs3 = (VoitureSport) v1;`
- f) `VoitureSport vs4 = (VoitureSport) v2;`
- g) `Voiture v3 = (Voiture) vs1;`
- h) `Voiture v4 = (Voiture) vs2;`
- i) `v1.accelerer();`
- j) `v2.accelerer();`
- k) `vs1.accelerer();`
- l) `vs2.accelerer();`
- m) `v1.accelerer(10);`
- n) `v2.accelerer(10);`
- o) `vs1.accelerer(10);`
- p) `vs2.accelerer(10);`

Q1. Quelles sont les lignes qui provoquent des erreurs de compilation ?

Q2. En enlevant les lignes ne passant pas à la compilation, quelles sont celles qui provoquent des erreurs d'exécution ?

Q3. Pour les lignes i) j) k) et l) préciser en cas d'exécution correcte quelle est la méthode qui a été effectuée entre celle de `Voiture` et de `VoitureSport`.

Q4. Pour les lignes e) f) g) et h) préciser les transtypages (cast) inutiles.

Q5. On ajoute l'annotation `@Override` aux 2 méthodes `accelerer` de la classe fille comme ci-dessous. Est-ce correct ?

```
public class VoitureSport extends Voiture {
    @Override
    public void accelerer() { /* non précisé */ }
    @Override
    public void accelerer(int q) { /* non précisé */ }
}
```

Exercice 4

Soit la classe `Garage` contenant une collection de voitures. On vous donne uniquement l'entête de cette classe et de 2 de ses méthodes :

```
public class Garage implements Iterable<Voiture> {
    // ajoute au garage les voitures de la collection en paramètre
    public void ajouterVoitures(Collection voitures){}
```

```

// ajoute dans la collection en paramètre les voitures du garage de la
// couleur en paramètre
public void extraireVoitures(Collection voitures, Couleur couleur){
}
}

```

Q1. Préciser le paramètre de genericité du type `Collection` pour le paramètre `voitures` pour les 2 méthodes `ajouterVoiture` et `extraireVoitures`. Veiller à garder la souplesse de l'héritage.

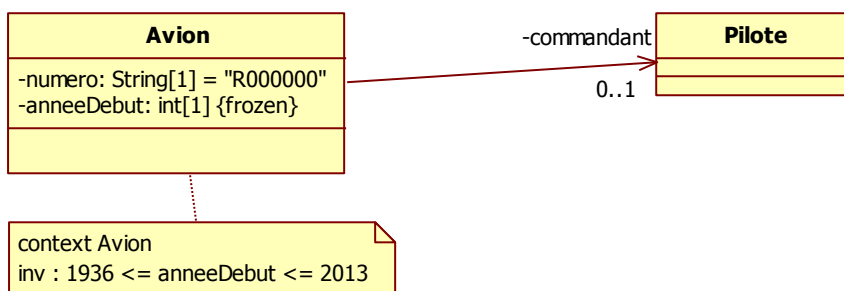
Q2. On considère une classe cliente utilisant la classe `garage`. Donner le code de la méthode d'affichage complet d'un garage en réutilisant la représentation textuelle de base de chaque voiture. L'entête de la méthode à coder est la suivante :

```
private static void afficherGarage(Garage garage) { ... }
```

Exercice 5

On vous donne le diagramme de classes ci-dessous. Un avion a un numéro de série (par défaut R000000), une année de début d'exploitation non modifiable et éventuellement un commandant. On considère que la classe `Pilote` est écrite en Java avec la redéfinition adéquate des méthodes d'`Object` `equals`, `hashCode` et `toString`.

Rappel UML : les crochets sur un attribut indique les cardinalités de celui-ci.



Q1. Ecrire le code Java de la classe `Avion` en tenant compte des contraintes UML et OCL ainsi que des consignes suivantes :

- un constructeur partiel avec uniquement l'année de début
- un constructeur complet avec numéro, année de début et commandant
- les accesseurs (getters et setters) nécessaires
- redéfinir les méthodes `equals`, `hashCode` et `toString` de la classe `Object` en tenant compte des 3 informations numéro, année de début et commandant.
- Exploiter au maximum les possibilités de Java 7 pour gérer le caractère obligatoire ou facultatif d'un attribut.
- On utilisera l'exception `NullPointerException` ou `IllegalArgumentException` si une contrainte UML ou OCL n'est pas vérifiée.

Exercice 6

Une table de hachage permet de stocker des paires "clef-valeur" dans le but d'uniformiser la complexité des accès en lecture et écriture. On définit une fonction de hachage qui, à toute valeur à stocker passée en paramètre, lui attribue un entier compris entre 0 et une valeur max. Chaque valeur de hachage contient un ensemble d'éléments dans lequel on stocke une structure de deux champs, la clef et la valeur.

Par exemple, pour une valeur de hachage comprise entre 0 et 5, si on a :

- h("bonjour") == 2
- h("salut") == 4
- h("coucou") == 2

Et si on souhaite associer les couples :

- "bonjour" => "jaune"
- "salut" => "vert"
- "coucou" => "rouge"

Alors la table de hachage sera :

- [0] => {}
- [1] => {}
- [2] => { ["bonjour", "jaune"], ["coucou", "rouge"] }
- [3] => {}
- [4] => { ["salut", "vert"] }
- [5] => {}

En Java, on va hériter de l'interface `eisti.Map<K,V>`, donnée en annexe, pour créer notre table de hachage et de l'interface `java.util.Map.Entry<K,V>` pour gérer les couples stockés.

On vous donne le code de la classe `Element` et le début du celui de la classe `TableHachage` :

//classe Element

```
public class Element<K,V> implements java.util.Map.Entry<K,V> {
    private final K key;
    private V value;
    public Element(K key, V value) { this.key=key; this.value=value; }

    @Override
    public K getKey() { return this.key; }

    @Override
    public V getValue() { return this.value; }

    @Override
    public V setValue(V value) { this.value=value; return this.value; }
}
```

//classe TableHachage

```
import java.util.List;
import java.util.ArrayList;
import eisti.Map;

public class TableHachage<K,V> implements Map<K,V> {
    public final static int MAX_HACHAGE=100;

    private final List<List<Element<K,V>>> tableHachage;

    public TableHachage() {
        // création du tableau de MAX_HACHAGE listes vides
        tableHachage = new ArrayList<>(MAX_HACHAGE);
        for (int i=0;i<MAX_HACHAGE;++i) {
            List<Element<K,V>> alveoles = new ArrayList<Element<K,V>>();
            tableHachage.add(alveoles);
        }
    }
}
```

```
// ramène le hashCode d'un objet dans l'intervalle [0,MAX_HACHAGE[
private int h(K key) {
    int res=key.hashCode();
    if (res<0) res*=(-1);
    return res%MAX_HACHAGE;
}

//TODO : à compléter
}
```

Q1. Donner le code de la méthode `keySet` de la classe `TableHachage`.

Q2. Donner le code de la méthode `containsValue` de la classe `TableHachage`.

Q3. Donner les instructions permettant de créer une table de hachage permettant d'associer un booléen à une voiture (en considérant la classe `Voiture` existante).

Annexes

Extrait de l'API de la classe `java.lang.Object`

boolean [`equals`](#)(Object obj)
Indicates whether some other object is "equal to" this one.

Class<?> [`getClass`](#)()
Returns the runtime class of this `Object`.

int [`hashCode`](#)()
Returns a hash code value for the object.

String [`toString`](#)()
Returns a string representation of the object equal to the value of :
`getClass().getName() + '@' + Integer.toHexString(hashCode())`

API de l'interface `eisti.Map<K,V>`

void [`clear`](#)()
Removes all of the mappings from this map (optional operation).

boolean [`containsKey`](#)(K key)
Returns `true` if this map contains a mapping for the specified key.

boolean [`containsValue`](#)(V value)
Returns `true` if this map maps one or more keys to the specified value.

Set<Map.Entry<K,V>> [`entrySet`](#)()
Returns a [`Set`](#) view of the mappings contained in this map.

V [`get`](#)(K key)
Returns the value to which the specified key is mapped, or `null` if this map contains no mapping for the key.

boolean [`isEmpty`](#)()
Returns `true` if this map contains no key-value mappings.

Set<K> [`keySet`](#)()
Returns a [`Set`](#) view of the keys contained in this map.

V [`put`](#)(K key, V value)
Associates the specified value with the specified key in this map.

void [`putAll`](#)(Map<? extends K,? extends V> m)
Copies all of the mappings from the specified map to this map.

V [`remove`](#)(K key)
Removes the mapping for a key from this map if it is present.

int [`size`](#)()
Returns the number of key-value mappings in this map.

Collection<V> [`values`](#)()
Returns a [`Collection`](#) view of the values contained in this map.

NB :

- les interfaces `Set`, `Map.Entry`, `Collection` sont importées de `java.util`.
- l'interface `eisti.Map<K,V>` est une version modifiée de `java.util.Map<K,V>` en remplaçant les paramètres de type `Object` par `K` ou `V`.

Extrait de l'API de l'interface `java.util.List<E>`

<code>boolean</code>	<code>add</code> (<code>E e</code>) Appends the specified element to the end of this list (optional operation).
<code>void</code>	<code>add</code> (<code>int index, E element</code>) Inserts the specified element at the specified position in this list (optional operation).
<code>boolean</code>	<code>addAll</code> (<code>Collection<? extends E> c</code>) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).
<code>boolean</code>	<code>addAll</code> (<code>int index, Collection<? extends E> c</code>) Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
<code>void</code>	<code>clear</code> () Removes all of the elements from this list (optional operation).
<code>boolean</code>	<code>contains</code> (<code>Object o</code>) Returns <code>true</code> if this list contains the specified element.
<code>E</code>	<code>get</code> (<code>int index</code>) Returns the element at the specified position in this list.
<code>int</code>	<code>indexOf</code> (<code>Object o</code>) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
<code>boolean</code>	<code>isEmpty</code> () Returns <code>true</code> if this list contains no elements.
<code>Iterator<E></code>	<code>iterator</code> () Returns an iterator over the elements in this list in proper sequence.
<code>int</code>	<code>lastIndexOf</code> (<code>Object o</code>) Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
<code>E</code>	<code>remove</code> (<code>int index</code>) Removes the element at the specified position in this list (optional operation).
<code>boolean</code>	<code>remove</code> (<code>Object o</code>) Removes the first occurrence of the specified element from this list, if it is present (optional operation).
<code>boolean</code>	<code>removeAll</code> (<code>Collection<?> c</code>) Removes from this list all of its elements that are contained in the specified collection (optional operation).
<code>boolean</code>	<code>retainAll</code> (<code>Collection<?> c</code>) Retains only the elements in this list that are contained in the specified collection (optional operation).
<code>E</code>	<code>set</code> (<code>int index, E element</code>) Replaces the element at the specified position in this list with the specified element (optional operation).
<code>int</code>	<code>size</code> () Returns the number of elements in this list.
<code>List<E></code>	<code>subList</code> (<code>int fromIndex, int toIndex</code>) Returns a view of the portion of this list between the specified <code>fromIndex</code> , inclusive, and <code>toIndex</code> ,

exclusive.

API de la classe `java.util.Objects`

static `compare`(T a, T b, Comparator<? super T> c)
<T> int Returns 0 if the arguments are identical and `c.compare(a, b)` otherwise.

static `deepEquals`(Object a, Object b)
boolean Returns true if the arguments are deeply equal to each other and false otherwise.

static `equals`(Object a, Object b)
boolean Returns true if the arguments are equal to each other and false otherwise.

static int `hash`(Object... values)
Generates a hash code for a sequence of input values.

static int `hashCode`(Object o)
Returns the hash code of a non-null argument and 0 for a null argument.

static `requireNonNull`(T obj)
<T> T Checks that the specified object reference is not null.

static `requireNonNull`(T obj, String message)
<T> T Checks that the specified object reference is not null and throws a customized `NullPointerException` if it is.

static `toString`(Object o)
String Returns the result of calling `toString` for a non-null argument and "null" for a null argument.

static `toString`(Object o, String nullDefault)
String Returns the result of calling `toString` on the first argument if the first argument is not null and returns the second argument otherwise.

API de l'interface `java.lang.Iterable<T>`

`java.util.Iterator<T>` `iterator`()
Returns an iterator over a set of elements of type T.