Numéro de copie :

ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

Examen ILO

16/01/2014

Durée : 1h45

Documents autorisés : Polycopiés de cours uniquement. Pas de calculettes.

Vous répondrez aux questions directement sur le sujet, puis insérerez le sujet dans une copie numérotée sans oublier de noter le numéro de la copie en haut de page.

1 – Questions de cours (9 points)

A défaut d'indication particulière (Java ou C++) les questions concernent les deux langages.

Entourez ou soulignez la ou les bonnes réponses à chaque question.

Si vous sélectionnez une bonne et une mauvaise réponse dans une même question, les deux s'annulent.

1) En C++ lorsque l'on instancie un tableau d'objets chaque case du tableau est instanciée avec le constructeur par défaut.
   a) vrai
   b) faux
2) Quel est le mode d'accès par défaut d'un membre d'une classe en Java ?
   a) public.
   b) protected.
   c) private.
   d) autre.
3) Quel est le mode d'accès par défaut d'un membre d'une classe en C++ ?
   a) public.
   b) protected.
   c) private.
   d) autre.
4) En C++ lors d'un héritage privé : class B : private A
   a) Les membres privés de A deviennent publics dans la classe B.
   b) Les membres protégés de A deviennent publics dans la classe B.
   c) Les membres publics de A deviennent privés dans la classe B.
5) Qu'est que le paradigme de l'iceberg ?
   a) Le fait d'avoir des méthodes privées dans une classe.
   b) Le fait de déclarer une classe dans une autre classe.
   c) Le fait de masquer les attributs et de rendre publiques les méthodes qui y accèdent.
6) Quelle est la durée de vie d'une variable en java ?
   a) C'est le garbage collector qui décide de la durée de vie d'une variable.
   b) Le bloc dans lequel elle est déclarée.
7) « this » représente une référence vers l'instance courante en java ou un pointeur vers l'instance courante en C++. Tester la condition (this == null) en java ou (this == NULL) en C++ a un sens :
   a) dans une méthode d'instance
   b) dans une méthode de classe
   c) nulle part

8) Polymorphisme : Les méthodes surchargées sont différentiées entre elles par :
   a) Le nombre d'arguments.
   b) Le type de retour.
   c) Le type des arguments.
9) En Java, la classe B hérite de la classe A et ces deux classes implémentent toutes les deux la méthode f0. Quelles sont les méthodes déclenchées par le code suivant ?

```
A aa = new A(...);
A ab = new B(...);
aa.f0();
ab.f0();
```

   a) Pour l'instruction `aa.f0()` : f0 de la classe A.
   b) Pour l'instruction `aa.f0()` : f0 de la classe B.
   c) Pour l'instruction `ab.f0()` : f0 de la classe A.
   d) Pour l'instruction `ab.f0()` : f0 de la classe B.
10) En Java, une interface peut :
   a) être héritée.
   b) être implémentée.
   c) être instanciée.
   d) être référencée (⇔ on peut avoir des variables de ce type).
11) Une classe abstraite peut contenir :
   a) des méthodes abstraites.
   b) des méthodes concrètes.
12) Que fait le "véritable" constructeur par défaut ?
   a) Il appelle le constructeur de copie.
   b) Il appelle le constructeur par défaut de la classe mère s'il y en a une.
   c) Il initialise chacun des attributs de la classe.
13) Constructeurs : Lorsqu'aucun constructeur n'est définit dans une classe que se passe t'il ?
   a) Il existe un constructeur de copie qui réalise la copie bit à bit de l'objet.
   b) Il existe un constructeur par défaut.
   c) On ne peut pas instancier une telle classe.
14) Pourquoi la définition d'un constructeur ne fait elle pas apparaître son type de retour ?
   a) Parce qu'il est implicite.
   b) Parce que c'est à l'utilisateur de le préciser.
   c) Parce que c'est inutile.
15) En C++ Il y a appel implicite au constructeur de copie d'un objet
   a) Lorsque cet objet est déclaré dans une méthode ou une fonction
   b) Lorsque cet objet est passé en argument d'une méthode ou d'une fonction
   c) Lorsque cet objet est renvoyé d'une méthode ou d'une fonction
   d) Lorsque l'adresse de cet objet est passée en argument d'une méthode ou d'une fonction
16) Héritage des constructeurs
   a) Les constructeurs ne sont pas hérités.
   b) Les constructeurs sont hérités.
   c) Seul le constructeur de copie est hérité.
   d) Seul le constructeur par défaut est hérité.

17) Qu'est ce que le polymorphisme d'héritage ?
   a) Le fait qu'une classe puisse hériter d'une autre classe.
   b) Le fait qu'une instance d'une classe fille puisse être considérée comme une instance de la classe mère.
   c) Le fait qu'une instance de la classe mère puisse être considérée comme une instance d'une classe fille.
   d) Le fait que l'on puisse surcharger les méthodes de la classe mère dans la classe fille.

18) En Java, La méthode clone() héritée de la classe Object permet de créer des copies de soi-même. Parmi les assertions suivantes, lesquelles sont vraies ?
   a) x.clone() == x
   b) x.clone().getClass().equals(x.getClass())
   c) x.equals(x.clone())

19) Si "a" et "b" sont deux instances distinctes de la même classe en Java et qu'elles sont égales au sens de la méthode "equals", qu'est ce que cela implique ?
   a) a == b est faux
   b) a == b est vrai
   c) a.hashCode() == b.hashCode() est faux
   d) a.hashCode() == b.hashCode() est vrai

20) L'utilisation d'un paramètre de type en Java Generics (par exemple MaCollection<E>) permet
   a) d'interroger une collection sur la nature de son contenu en utilisant l'introspection
   b) de garantir l'hétérogénéité des types dans une collection
   c) de garantir l'unicité des types dans une collection

21) Parmi les instructions génériques suivantes, lesquelles ne provoquent pas d'erreur de compilation en Java ?
   a) `Collection<?>[] tabCol = new List<?>[10];`
   b) `Collection<Object>[] tabCol = new List<?>[10];`
   c) `Collection<String>[] tabCol = (Collection<String>[]) new List<?>[10];`
   d) `Collection<String>[] tabCol = new List<?>[10];`
   e) `Collection<String>[] tabCol = new List<String>[10];`

22) La classe java standard LinkedList<E> est aussi :
   a) RandomAccess
   b) Une AbstractList<E>.
   c) Une ArrayList<E>
   d) Une List<E>
   e) Une Queue<E>

23) Parmi les algorithmes standard applicables aux collections Java figure l'algorithme Collections.**reverse**(List<?> list) qui permet d'inverser l'ordre des éléments dans une liste. Celui ci est donc aussi applicable à :
   a) Un Vector<?>
   b) Une Stack<?>
   c) Une LinkedList<?>
   d) Un LinkedHashSet<?>
   e) Un copyOnWriteArraySet<?>

24) Exceptions : si une méthode d'un objet peut soulever une exception de part les instructions qu'elle utilise mais n'encadre pas ces instructions par un bloc try {...} catch (Exception ...) {...} que risque t'il de se passer si une exception est levée ?
   a) l'exception provoquera une erreur d'exécution directement lors de l'appel de cette méthode.
   b) l'exception sera transmise à l'appelant.

25) En UML, une relation de composition entre deux classes implique :
   a) Que la classe qui contient une instance d'une autre classe reçoive cette instance sans être responsable de sa création et de sa destruction.
   b) Que la classe qui contient une instance d'une autre classe soit responsable de sa création et de sa destruction.
26) En UML, une relation d'agrégation entre deux classes implique :
   a) Que la classe qui contient une instance d'une autre classe reçoive cette instance sans être responsable de sa création et de sa destruction.
   b) Que la classe qui contient une instance d'une autre classe soit responsable de sa création et de sa destruction.
27) Pour chaque cas d'utilisation en UML, quel diagramme doit-on faire ?
   a) Un diagramme d'objets
   b) Un diagramme de classes
   c) Un diagramme de séquence
28) Parmi les diagrammes UML suivants, deux d'entre eux sont considérés comme étant équivalents car ils décrivent la même chose :
   a) Le diagramme d'objets
   b) Le diagramme de classes
   c) Le diagramme de collaboration
   d) Le diagramme de séquence
   e) Le diagramme des cas d'utilisation
29) L'interface Collection<E> en Java définit une méthode « Iterator<E> iterator() » qui lorsqu'elle est implémentée dans les classes qui implémentent l'interface Collection<E> permet d'obtenir un itérateur sur ces collections concrètes. Quel patron de conception est implémenté dans cette méthode iterator() ?
   a) Le pattern Abstract Factory
   b) Le pattern Factory Method
   c) Le pattern Iterator
   d) Le pattern Template Method
   e) Le pattern "Singleton"

Numéro de copie :

2 – Mini Torture Test (6 points)

Après avoir étudié le code Java suivant :

```
import static java.lang.System.out;

public class A
{
	public A()
	{
		out.println("A() ");
	}
	public void f0()
	{
		out.println("A.f0() ");
	}
	public void f1(A arg)
	{
		out.println("A.f1(A) ");
		f0();
		arg.f0();
	}
}
```

```
import static java.lang.System.out;

public class B extends A
{
	protected A a;
	public B()
	{
		out.println("B() ");
		a = new A();
	}
	public void f0()
	{
		out.println("B.f0() ");
	}
}
```

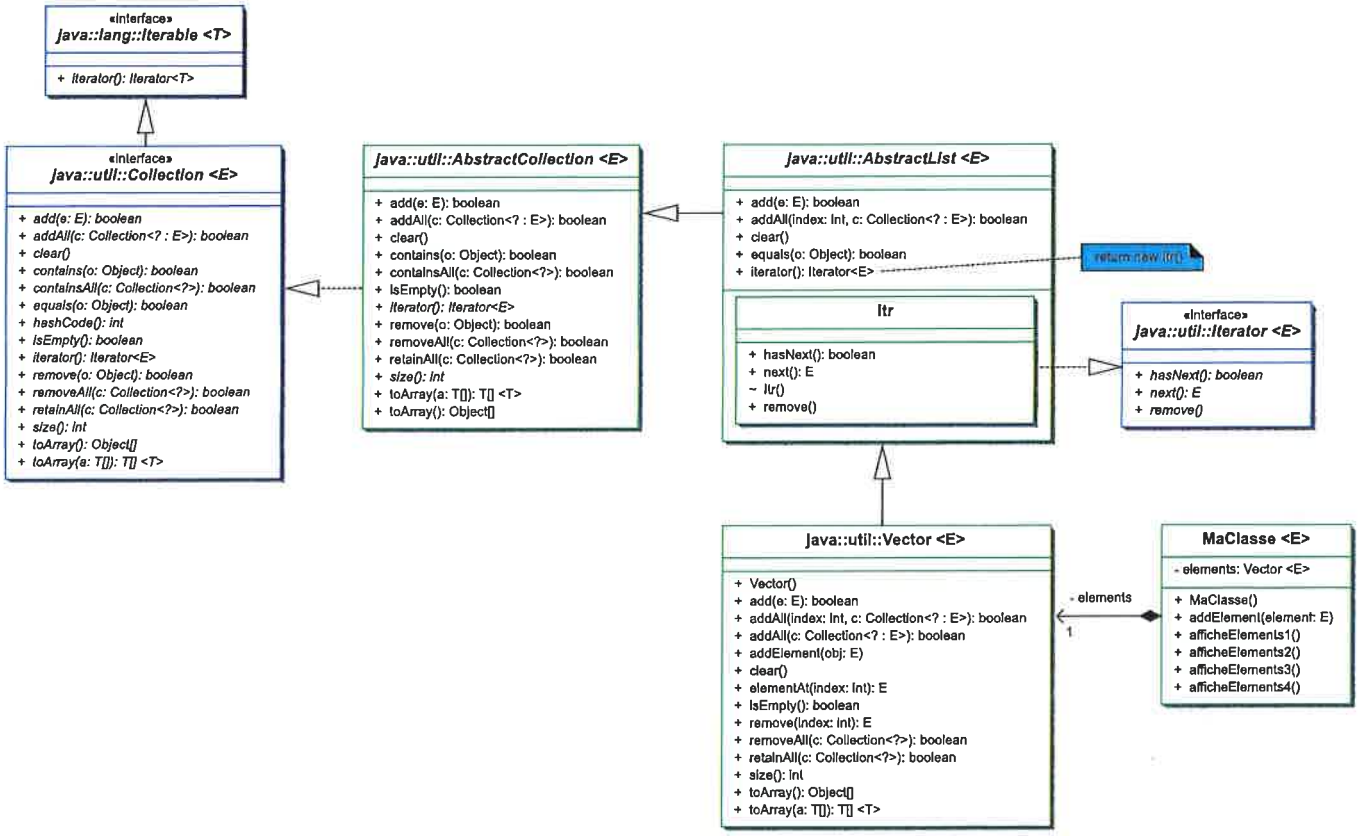
```
public class TestAB
{
	public static void main(String args[])
	{
		int nbCases = 2;
		A[] tabA = new A[nbCases];

		tabA[0] = new A(); // ..............................instruction 1
		tabA[1] = new B(); // ..............................instruction 2

		for (int i = 0; i < nbCases; i++)
		{
			tabA[i].f0(); // ..............................instruction 3 (a & b)
			tabA[i].f1(tabA[(i+1)%nbCases]); // ...........instruction 4 (a & b)
		}
	}
}
```

Indiquez pour les lignes du programme principal dénotées *instruction 1* à *instruction 4* les sorties console correspondantes. Remarque : l'*instruction 3a* correspond au premier tour de la boucle for et l'*instruction 3b* correspond au second tour de la boucle for (idem pour l'instruction 4).

Instructions		Sorties Console
1	`tabA[0] = new A();`	
2	`tabA[1] = new B();`	
3a (i=0)	`tabA[i].f0();`	
4a (i=0)	`tabA[i].f1(tabA[(i+1)%nbCases]);`	
3b (i=1)	`tabA[i].f0();`	
4b (i=1)	`tabA[i].f1(tabA[(i+1)%nbCases]);`	

3 – Parcours de collection (3 points)

Soit le diagramme de classe suivant :



La classe MaClasse<E> contient un Vector<E> d'éléments "elements" qui lui-même implémente l'interface Collection, qui elle-même hérite de l'interface Iterable. Ecrivez 3 méthodes différentes permettant de parcourir les éléments du Vector "elements" afin de les afficher dans la console, en utilisant uniquement les méthodes mentionnées dans le graphe de classe ci-dessus et sans utiliser les méthodes "toArray".

a) Première méthode

```
public void afficheElements1()
{
        for (_____)
        {
                System.out.println(_____);
        }
}
```

b) Deuxième méthode

```
public void afficheElements2()
{
        for (_____)
        {
                System.out.println(_____);
        }
}
```

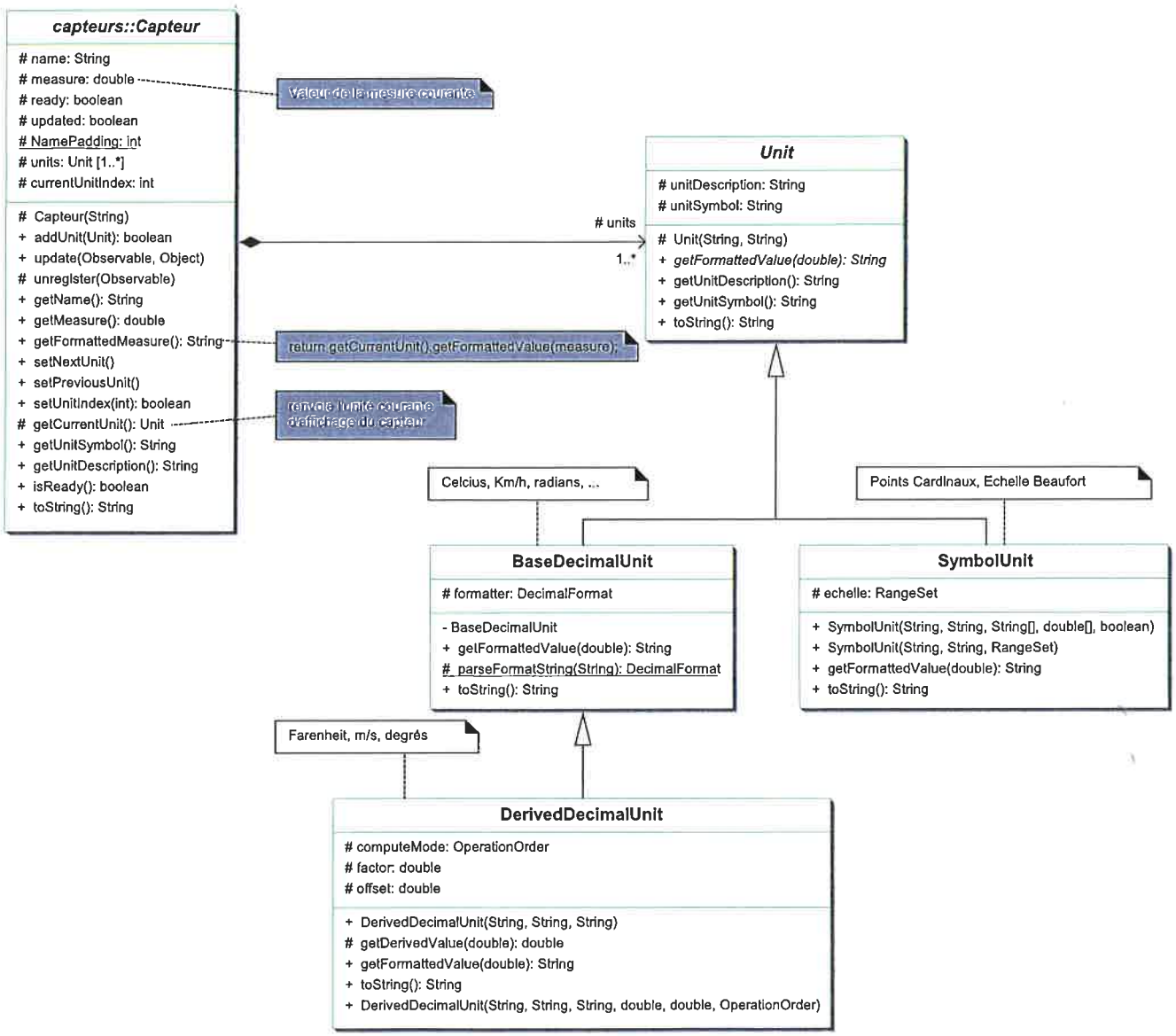
c) Troisième méthode

```
public void afficheElements3()
{
        for (_____)
        {
                System.out.println(_____);
        }
}
```

4 – Design Patterns (2 points)

Capteurs et unités

Un capteur est une entité permettant d'afficher sa mesure courante dans différentes unités. Il possède donc une collection d'unités dans lesquelles il peut exprimer sa mesure grâce à la méthode getFormattedMeasure() qui fait appel à la méthode getFormattedValue(double) de l'unité courante du capteur obtenue grâce à getCurrentUnit().



La collection « units » de la classe Capteur contient des instances des classes filles de Unit. La méthode getFormattedValue(double) est abstraite dans la classe Unit mais concrète dans ses classes filles. Quel est le Design pattern mis en œuvre entre le Capteur et les Unités au travers de la méthode getFormattedMeasure() du capteur ?