

Unité IN3S02

Exemple de sujet de contrôle final (partie Java)

Enoncé

Conditions d'examen : 2h30, sur feuille blanche, sans document, notation sur 20 points

Consignes et informations générales :

- *Pour les étudiants internationaux, vous pouvez, si vous préférez, rédiger en anglais.*
- *Les solutions sont attendues en Java. Elles seront évaluées sur le plan sémantique, syntaxique et de la qualité de programmation. Tout élément de programme (classes, méthodes, variables, ...) mis en oeuvre doit être défini par ou en Java. L'utilisation de bibliothèques standard, sauf indication contraire, est admise. Le non respect des « contraintes de programmation » éventuellement spécifiées sera sanctionné comme du hors sujet.*
- *Le nombre de points est donné à titre indicatif.*

EXERCICE 1 (6 points)

Les questions suivantes sont indépendantes. Il est attendu des réponses explicatives courtes et précises (et qui ne soient pas des paraphrases !).

1.1- On considère la classe `TreeSet<E>` dont un extrait de la documentation est fourni en annexe 1 :

- Que signifie le suffixe `<E>` attaché au nom de la classe ?
- De quelle(s) classe(s) hérite la classe `TreeSet<E>` ?
- A quel paquetage appartient la classe `TreeSet<E>` ? Ecrire la directive d'importation à insérer dans le fichier utilisant la classe `TreeSet<E>`.
- La classe `TreeSet<E>` offre un constructeur sans argument (non décrit dans l'extrait de documentation fourni). Ecrire l'instruction de création d'un objet de ce type.
- Que signifie `extends AbstractSet<E>` ?
- Que signifie `implements Set<E>` ? Qu'en déduire sur les caractéristiques de `Set<E>` ?

1.2- On considère l'interface `Comparable<T>` dont un extrait de la documentation est fourni en annexe 2 :

- Pour quelle raison `Comparable<T>` est déclaré `public interface Comparable<T>` et non pas :
 - `public Class Comparable<T>` ?
 - `public abstract Class Comparable<T>` ?

1.3- On considère la classe `Calendar` dont un extrait de la documentation est fourni en annexe 3 :

- La classe `Calendar` est déclarée `abstract` : qu'est-ce que cela signifie ? Qu'est-ce que cela implique quant à son utilisation ?

- `DAY_OF_WEEK` est déclaré `static int` : qu'est-ce que cela signifie ?
- Le constructeur est déclaré `protected` : qu'est-ce que cela signifie ?
- `add` est déclaré `abstract void` : qu'est-ce que cela signifie ? Qu'est-ce que cela implique quant à son utilisation ?

1.4- On considère la classe `Integer` dont un extrait de la documentation est fourni en annexe 3 :

- `valueOf` est déclarée `static Integer` : qu'est-ce que cela signifie ? Qu'est-ce que cela implique quant à son utilisation ?

EXERCICE 2 (3 points)

Ecrire en Java une méthode `circularRightShift` répondant aux spécifications suivantes :

- procédure
- paramètre : un tableau `t` d'éléments de type `String`
- comportement : réalise, sans perte d'élément, un « décalage droite circulaire » de tous les éléments du tableau
- exemple :
avec le tableau initial `t = ["one", "two", "three", "four", "five"]`
→ le tableau final sera `t = ["five", "one", "two", "three", "four"]`
- contrainte de programmation : aucun autre tableau que `t`

EXERCICE 3 (6 points)

(Source : [@author](#))

3.1- Modéliser un compteur kilométrique

Un compteur kilométrique se compose d'un *enregistrement totalisateur* et d'un *enregistrement partiel*. Ces deux enregistrements conservent les distances parcourues sous la forme d'un nombre réel.

Quand un compteur est créé, la valeur de ces deux enregistrements est toujours nulle. L'enregistrement partiel peut être remis à zéro (mais pas à une autre valeur). Outre cette opération, la seule façon de modifier la valeur des deux enregistrements est d'ajouter des kilomètres en roulant.

Définir en Java une classe `Compteur` comportant :

- deux champs d'instance : `totalisateur` et `partiel`
- un constructeur sans paramètre
- les fonctions publiques d'accès aux valeurs des champs : `getTotalisateur()` et `getPartiel()`, retournant la valeur du champ
- une procédure publique `resetPartiel()` de remise à zéro du champ `partiel`
- une procédure publique `add`, qui incrémente les deux champs du nombre de kilomètres (un nombre réel) spécifié en paramètre
- la méthode générique `toString()`, qui retourne une représentation textuelle d'un compteur et de son état conforme à l'exemple suivant :

```
compteur = [ totalisateur = 500 | partiel = 108 ]
```

3.2- Modéliser un véhicule

Un véhicule se compose d'un *numéro d'immatriculation*, d'un *compteur*, d'un *réservoir*, et d'une *consommation kilométrique*. Les véhicules seront représentés par des instances de la classe `Vehicule`.

Le *numéro d'immatriculation* d'un nouveau véhicule est déterminé par la valeur d'un registre commun à tous les véhicules ; la valeur de ce registre est automatiquement incrémentée de 1 à chaque création d'un nouveau véhicule. Ce registre ne doit pas être accessible en dehors de la classe `Vehicule`. Le numéro d'immatriculation d'un véhicule ne doit pas être non plus accessible en dehors de la classe, mais sa valeur peut être obtenue par une méthode de lecture à partir de n'importe quelle classe.

Le *compteur kilométrique* est celui modélisé à la question 4.1 précédente. Il est créé à la création d'un véhicule. On obtient une référence à ce compteur par une méthode `getCompteur()` de la classe `Vehicule`.

La *capacité* du réservoir est de 50.0 litres pour tous les véhicules et ne doit pas être accessible de l'extérieur de la classe. La *jauge* du réservoir indique la quantité de carburant qui reste dans le réservoir. Elle propre à chaque véhicule. Elle ne doit pas être accessible de l'extérieur de la classe, mais sa valeur peut être obtenue par une méthode de lecture `getJauge()` à partir de n'importe quelle classe. A chaque fois qu'on fait le plein, cette jauge est remise à la valeur de la capacité du réservoir ; il n'est pas possible de ne remplir qu'une fraction du réservoir.

La *consommation* d'un véhicule (nombre de litres de carburant par kilomètre parcouru) est propre à chaque véhicule et est fixée à la création du véhicule. Cette consommation ne doit pas être accessible de l'extérieur de la classe.

Les deux opérations que l'on peut faire sur un véhicule sont : *faire le plein* et *rouler* sur une certaine distance. En roulant on consomme du carburant, donc on ne peut rouler que sur la distance permise par le réservoir et la consommation du véhicule ; le compteur n'est incrémenté que de la distance effectivement parcourue.

Définir en Java une classe `Vehicule` implémentant l'interface `Comparable` (cf annexe 2) et comportant :

- les attributs de classe
- les attributs d'instance
- un constructeur ayant pour paramètre la consommation
- des méthodes d'accès en lecture aux champs d'instance, selon le degré d'accessibilité prescrit ci-avant
- une procédure `faireLePlein()`
- une fonction `rouler`, qui prend en paramètre la distance que l'on souhaite parcourir et qui retourne la distance effectivement parcourue (les distances sont spécifiées par un nombre réel)
- la méthode générique `toString()`, qui retourne une représentation textuelle d'un véhicule et de son état, conforme à l'exemple suivant :
Véhicule 45 : compteur = [totalisateur = 500 | partiel = 108] ; jauge = 9.00533
- la méthode générique `compareTo`, qui compare ce véhicule avec le véhicule spécifié au regard des numéros d'immatriculation

EXERCICE 4 (3 points)

Soit à écrire un programme qui détermine le nom du jour correspondant à une date donnée. Par exemple, si vous saisissez comme date le 9 décembre 1986, le programme devra produire « mardi ». Le quantième (9 dans cet exemple) sera appelé « jour du mois ».

Pour cela, on utilisera les classes `Calendar`, `GregorianCalendar` et `Integer` dont un extrait de la documentation est fourni en annexe.

Ecrire en Java une classe `Date` possédant :

- un attribut `calendar` de type `Calendar` ou `GregorianCalendar`
- un constructeur qui prend en arguments trois entiers : le jour du mois, le mois et l'année, et initialise l'attribut `calendar` avec cette date
- une fonction entière `getDayOfWeek` qui renvoie le jour de la semaine correspondant à la date `calendar`. Exemple : avec l'exemple initial, la valeur retournée serait 3 (car mardi est compté comme le 3^{ème} jour de la semaine).
- une fonction `getNameDayOfWeek` qui utilise `getDayOfWeek` et renvoie le nom du jour de la semaine correspondant à la date `calendar`. Exemple : avec l'exemple initial, la valeur retournée serait « mardi ».
- une méthode principale `main` qui crée une date de type `Date` initialisée à la date passée en argument, détermine le nom du jour correspondant par appel à `getNameDayOfWeek`, et l'affiche dans un terminal.

EXERCICE 5 (2 points)

On considère la classe suivante :

```
public final class Singleton {  
  
    private static Singleton single = null ;  
    private static int value ;  
  
    private Singleton() { } // c'est bien private et non public !  
  
    public static Singleton instance() {  
        if (single == null) {  
            single = new Singleton() ;  
            value = new Random().nextInt() ;  
        }  
        return single ;  
    }  
  
    public int getValue() { return value ; }  
}
```

Cette classe est complète et ne comporte pas d'erreur de syntaxe.

Que signifie `final` dans l'en-tête de la classe ?

Ecrire en Java les quelques instructions permettant de :

- créer un objet `o1` de type `Singleton`
- créer une variable `v1` initialisée à la valeur du champ `value` de `o1`

Idem pour un second objet `o2` et une variable `v2`.

Est-ce que `v1==v2` est vrai ? Est-ce que `o1==o2` est vrai ? Justifier.

ANNEXE 1

Extrait de la documentation de la classe `TreeSet<E>`

`java.util`

Class `TreeSet<E>`

[java.lang.Object](#)

- └ [java.util.AbstractCollection<E>](#)
- └ [java.util.AbstractSet<E>](#)
- └ [java.util.TreeSet<E>](#)

All Implemented Interfaces

[Serializable](#), [Cloneable](#), [Iterable<E>](#), [Collection<E>](#), [Set<E>](#), [SortedSet<E>](#)

```
public class TreeSet<E>
  extends AbstractSet<E>
  implements SortedSet<E>, Cloneable, Serializable
```

This class guarantees that the sorted set will be in ascending element order, sorted according to the *natural order* of the elements (see [Comparable](#)), or by the comparator provided at set creation time, depending on which constructor is used. The implementation provides guaranteed $\log(n)$ time cost for the basic operations (`add`, `remove` and `contains`).

ANNEXE 2

Extrait de la documentation de l'interface `Comparable<T>`

`java.lang`

Interface `Comparable<T>`

```
public interface Comparable<T>
```

Method Summary

int	compareTo (T o)
	Compares this object with the specified object for order.

Method Detail

`compareTo`

```
int compareTo (T o)
```

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

ANNEXE 3

a) Extrait de la documentation de la classe Calendar

java.util

Class Calendar

[java.lang.Object](#)

└ [java.util.Calendar](#)

```
public abstract class Calendar
extends Object
implements Serializable, Cloneable, Comparable<Calendar>
```

Field Summary

static int	DAY_OF_MONTH Field number for <code>get</code> and <code>set</code> indicating the day of the month.
static int	DAY_OF_WEEK Field number for <code>get</code> and <code>set</code> indicating the day of the week.
static int	FRIDAY Value of the <code>DAY_OF_WEEK</code> field indicating Friday.
static int	MONDAY Value of the <code>DAY_OF_WEEK</code> field indicating Monday.
static int	MONTH Field number for <code>get</code> and <code>set</code> indicating the month.
static int	SATURDAY Value of the <code>DAY_OF_WEEK</code> field indicating Saturday.
static int	SUNDAY Value of the <code>DAY_OF_WEEK</code> field indicating Sunday.
static int	THURSDAY Value of the <code>DAY_OF_WEEK</code> field indicating Thursday.
static int	TUESDAY Value of the <code>DAY_OF_WEEK</code> field indicating Tuesday.
static int	WEDNESDAY Value of the <code>DAY_OF_WEEK</code> field indicating Wednesday.
static int	YEAR Field number for <code>get</code> and <code>set</code> indicating the year.

Constructor Summary

protected	Calendar () Constructs a <code>Calendar</code> with the default time zone and locale.
-----------	--

Method Summary

abstract void	add (int field, int amount) Adds or subtracts the specified amount of time to the given calendar field, based on the calendar's rules.
---------------	---

int	get (int field) Returns the value of the given calendar field.
void	set (int field, int value) Sets the given calendar field to the given value.
void	set (int year, int month, int day_of_month) Sets the values for the calendar fields YEAR, MONTH, and DAY_OF_MONTH.

Field Detail

DAY_OF_MONTH

```
public static final int DAY_OF_MONTH
```

Field number for `get` and `set` indicating the day of the month. The first day of the month has value 1.

DAY_OF_WEEK

```
public static final int DAY_OF_WEEK
```

Field number for `get` and `set` indicating the day of the week. This field takes values SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY.

Method Detail

get

```
public int get(int field)
```

Returns the value of the given calendar field.

set

```
public void set(int field, int value)
```

Sets the given calendar field to the given value.

set

```
public final void set(int year, int month, int day_of_month)
```

Sets the values for the calendar fields YEAR, MONTH, and DAY_OF_MONTH.

Month value is 0-based. e.g., 0 for January

b) Extrait de la documentation de la classe `GregorianCalendar`

`java.util`

Class `GregorianCalendar`

[java.lang.Object](#)

└ [java.util.Calendar](#)

└ **java.util.GregorianCalendar**

```
public class GregorianCalendar
extends Calendar
```

Constructor Summary

[GregorianCalendar](#) ()

Constructs a default `GregorianCalendar` using the current time in the default time zone with the default locale.

[GregorianCalendar](#) (int year, int month, int dayOfMonth)

Constructs a `GregorianCalendar` with the given date set in the default time zone with the default locale. Month value is 0-based. e.g., 0 for January.

c) Extrait de la documentation de la classe `Integer`

`java.lang`

Class `Integer`

[java.lang.Object](#)

└ [java.lang.Number](#)

└ **java.lang.Integer**

```
public final class Integer
extends Number
implements Comparable<Integer>
```

Constructor Summary

[Integer](#) (int value)

Constructs a newly allocated `Integer` object that represents the specified `int` value.

[Integer](#) ([String](#) s)

Constructs a newly allocated `Integer` object that represents the `int` value indicated by the `String` parameter.

Method Summary

static int	parseInt (String s)
	Parses the string argument as a signed decimal integer.
static Integer	valueOf (String s)
	Returns an <code>Integer</code> object holding the value of the specified <code>String</code> .