**Durée : 1h AUCUN DOCUMENT - AUCUNE CALCULATRICE**

**Lisez attentivement chaque partie du sujet avant de vous lancer dans vos réponses.**

**Partie 1 : 8 QCM ≈ 15 minutes (8 POINTS). Une seule réponse par question :**

**1 point par réponse juste, -0,5 point par réponse fausse, 0 point sans réponse**

**Q1)** On veut écrire une classe qui ferait partie d'un package appelé "outils" et qui utilisera plusieurs classes du package java.util. Choisissez le bout de code le plus approprié pour démarrer le fichier source :

**(A) import java.util.\*;**

**(B) import java.util.\*; package outils;**

**(C) package outils; import java.util.\*;**

**(D) import outils.\*; import java.util.\*;**

**Q2)** Quel est le résultat de l’expression ("calcul" + 3 + (4 \* 5)) ?

**(A) "calcul320"**

**(B) "calcul35"**

**(C) "calcul23"**

**(D) Compilation error**

**Q3)** Choisissez le mot *qui n’est pas* un mot-clé Java.

**(A) void (B) int (C) String (D) class**

**Q4)** Modifiez le programme suivant pour qu’il affiche "Success!".

public class G {

public G (int i) { System.out.println("Success!"); }

}

public class H extends G {

public static void main (String argv[]){

H h = new H();

}

public H() { /\*???\*/ }

}

**(A) Remplacez /\*???\*/ avec this(10);**

**(B) Remplacez /\*???\*/ avec super(10);**

**(C) Remplacez /\*???\*/ avec super();**

**(D) Remplacez /\*???\*/ avec G(10);**

**Q5)** Sélectionnez l’expression qui donne l’indice du dernier élément d’un tableau nommé arr

**(A) arr.length**

**(B) arr.length-1**

**(C) arr.length()**

**(D) arr.length()-1**

**Q6)** Un attribut d’une classe avec la *visibilité par défaut* est visible par …

**(A) ... toutes les classes du programme**

**(B) ... toutes les classes de son package**

**(C)... seulement par la classe elle-même**

**(D)... seulement par la classe elle-même et ses sous-classes**

**Q7)** Combien de références de type String sont créées par l’instruction ci-dessous ?

String[] strings = new String[10];

**(A) 0 (B) 9 (C) 10 (D) 11**

**Q8)** Qu’affiche le programme?

public class A {

public void prn () { System.out.println (this.getVal()); }

public int getVal () { return 10; }

}

public class B extends A {

@Override

public int getVal () { return 20; }

}

public class C extends B {

public static void main (String args[]) {

B b = new B();

b.prn();

C c = new C();

c.prn();

}

}

**(A) “10”, “10” (B) “20”, “10” (C) “20”, “20” (D) Une erreur**

**Partie 2 : code Java ≈ 45 minutes (12 POINTS)**

Des extraits de la documentation des API Java sont fournis en **annexe** dans la dernière page.

Pour chacune des classes suivantes, n’oubliez pas d’importer les éventuelles librairies nécessaires. Nous supposons que ces classes font partie du même package.

* **(4 points)** Définir une classe *Date* avec trois attributs **protected** : le *jour*, le *mois* et l’*année* de type **int**. On y ajoute les éléments suivants :
* Trois constructeurs :
* le premier par défaut avec les valeurs 13 pour le jour, 2 pour le mois et 2016 pour l’année ;
* le second avec tous les paramètres nécessaires pour les attributs ;
* le troisième par copie de l’objet en paramètre.
* Une méthode *echanger()* qui échange la *Date* en paramètre avec la date de l’objet référencé (courant).
* La méthode surchargée (par héritage) *equals()*, avec en paramètre un objet de la classe *Date*, dont l’objectif est de retourner un booléen indiquant si le jour, le mois et l’année de l’objet en paramètre sont tous les 3 égaux à ceux de l’objet référencé.
* Une méthode *afficher()* sans paramètre qui affiche le jour, le mois et l’année de la date référencée.
* La méthode redéfinie *afficher()* qui affiche la chaîne en paramètre.
* Un getter qui retourne le mois
* Un setter qui modifie le mois.

public class Date {

// attributs **sur 0,5 point**

protected int jour, mois, annee ;

// 3 constructeur **sur 1 point**

public Date() {jour=13 ; mois=2 ; annee=1016 ;}

public Date(int jour, int mois, int annee) {this.jour=jour ; this.mois=mois ; this.annee=annee ;}

public Date(Date d) {jour=d.jour ; mois=d.mois ; annee=d.annee ;}

// 4 méthodes : **0,5 point chacune soient sur 2 points**

public void echanger(Date d) {Date temp ; temp=d ; d=this ; this=temp;}

@Override public boolean equals(Date d) {

boolean egal;

if (jour==d.jour && mois==d.mois && annee=d.annee)

egal=true ;

else

egal=false ;

return egal ;

}

public void afficher() {System.out.println(”La date est le ” + jour + ”/” + mois + ”/”+ annee ;}

public void afficher(String s) {System.out.println(s);}

// getter/setter **sur 0,5 point**

public int getMois() {return mois;}

public void setMois(int mois) {this.mois=mois;}

}

* **(4 points)** Définir une classe *TestDate* avec le **main** qui effectue les traitements suivants :
* Saisir au clavier un jour et une année. Ne pas blinder la saisie.
* Initialiser aléatoirement un mois entre 1 et 12.
* Instancier un premier objet de la classe *Date*  avec ce jour, ce mois et cette année.
* Instancier un second objet de la classe *Date*  par copie du premier objet.
* Instancier un troisième objet de la classe *Date* avec le constructeur par défaut.
* Tester si les 2 premiers objets ont des dates égales en appelant la méthode *equals()*. Si c’est le cas, afficher que les 2 dates sont identiques. Sinon (dates différentes) effectuer les actions suivantes :
* appeler la méthode *echanger()* pour échanger le premier et le troisième objet ;
* afficher la date du premier et du troisième objet ;
* modifier le mois du second objet en le remplaçant par celui du premier objet ;
* afficher la date du second objet.

// Importations **sur 0,25 point**

import java.util.Scanner;

import java.util.Random ;

public class TestDate {

public static void main (String args[]) {

// saisie au clavier du jour et du mois **sur 1 point**

Scanner clavier = new Scanner(System.in) ;

System.out.println(”entrez le jour et l’année : ”) ;

int jour=clavier.nextInt() ;

int annee=clavier.nextInt() ;

// Initialiser le mois aléatoirement **sur 0,5 point**

Random r = new Random() ;

int mois=r.nextInt(12)+1;

// Instancier les 3 objets **sur 0,75 point (0,25 chaque instanciation)**

Date d1 = new Date(jour, mois, annee);

Date d2 = new Date(d1) ;

Date d3 = new Date();

// test d’égalité et le reste **sur 1,5 point**

if (d1.equals(d2))

System.out.println(”Les 2 premières dates sont identiques”) ;

else {

d1.echanger(d3) ;

d1.afficher() ;

d3.afficher() ;

d2.setMois(d1.getMois()) ;

d2.afficher() ;

}

}

}

* **(2 points)** Définir une classe *DateFormat* qui hérite de la classe *Date*. Cette classe définit les éléments suivants :
* Un constructeur avec 3 paramètres le jour, le mois et l’année : il a pour objectif d’instancier un objet de la classe *Date* avec les 3 paramètres et appeler par héritage le constructeur par copie de cet objet.
* La méthode surchargée (par héritage) *toString()* dont l'objectif est de retourner une expression sous la forme de la chaîne « la date américaine est le *année*/*jour*/*mois* ».

public class DateFormat extends Date { // **sur 0,5 point**

// constructeur par héritage **sur 1 point**

public DateFormat(int jour, int mois, int annee) {

Date d = new Date(jour, mois, annee) ;

super(d) ;

}

// méthode surchargée **sur 0,5 point**

@Override

public String toString() {

return ”la date américaine est le ” + annee + ”/” + jour + ”/” + mois ;

}

}

* **(2 points)** Dans la suite du **main**, écrire les traitements suivants :
* Instancier un objet de la classe *DateFormat* avec le jour, le mois et l’année définis dans le **main** précédemment.

// **sur 1 point**

DateFormat d = new DateFormat(jour, mois, annee) ;

* Afficher cet objet de la classe *DateFormat* en appelant les bonnes méthodes.

// **sur 1 point**

d.afficher(d.toString()) ;

**ANNEXE : extraits de la documentation des API Java**

## java.util

## Class Scanner

|  |  |
| --- | --- |
| **Method Summary** | |
| [String](http://docs.oracle.com/javase/6/docs/api/java/lang/String.html) | [**next**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#next())()  Finds and returns the next complete token from this scanner. |
| **float** | [**nextFloat**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#nextFloat())()  Scans the next token of the input as a float. |
| **int** | [**nextInt**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#nextInt())()  Scans the next token of the input as an int. |
| [String](http://docs.oracle.com/javase/6/docs/api/java/lang/String.html) | [**nextLine**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#nextLine())()  Advances this scanner past the current line and returns the input that was skipped. |

**Class Random**

|  |  |
| --- | --- |
| **Method Summary** | |
| **float** | [**nextFloat**](http://docs.oracle.com/javase/6/docs/api/java/util/Random.html#nextFloat())()  Returns the next pseudorandom, uniformly distributed float value between 0.0 and 1.0 from this random number generator's sequence. |
| **int** | [**nextInt**](http://docs.oracle.com/javase/6/docs/api/java/util/Random.html#nextInt(int))(int n)            Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence. |

**Java.lang**

**Class Math**

|  |  |
| --- | --- |
| **Method Summary** | |
| **static double** | [**random**](http://docs.oracle.com/javase/6/docs/api/java/lang/Math.html#random())()            Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0. |