**Durée : 1h30. Aucun document autorisé. Aucune calculatrice.**

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

1. **10 QCM ≈ 20 minutes (5 POINTS). Une seule réponse par question (0,5 point par réponse juste)**

Q1) Soit la déclaration de 3 variables *locales* ci-dessous. Sélectionnez l’expression qui compile et exécute sans erreur.

String a = "aaa", b = "a", c;

**a) c += a; b) c = a + b; c) c = a - b; d) c = 'a';**

Q2) Choisissez le mot *qui n’est pas* un mot-clé Java.

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

Q3) Identifiez la ligne numérotée du programme ci-dessous qui génère une d’erreur.

public class DD {

private int a = 10;

protected int b;

}

public class EE extends DD {

int c;

public void display() {

System.*out*.println (a); // Line (1)

System.*out*.println (b); // Line (2)

System.*out*.println (c); // Line (3)

}

public static void main(String[] args) {

new EE().display();

}

}

**a) Line (1) b) Line (2) c) Line (3) d) Aucune de ces lignes ne génère une erreur**

Q4) 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**

Q5) Sélectionnez la seule phrase qui est fausse.

**a) Une classe abstraite peut être instanciée**

**b) Une classe abstraite peut définir des méthodes concrètes**

**c) Une classe abstraite peut définir des méthodes abstraites**

**d) Une classe abstraite peut définir des attributs**

Q6) Sélectionnez le bout de code que l’on utilisera pour une classe qui fait partie du package tools et qui utilise des classes des packages java.lang et java.util.

**a) import java.util.\*; import java.lang.\*; package tools;**

**b) import java.util.\*; import java.lang.\*;**

**c) package tools; import java.util.\*; import java.lang.\*;**

Q7) Qu’affiche le programme à l’exécution?

public class N

{

private int i = 1;

public static void go (N n, int i){

i = 3;

n.i = 4;

}

public static void main (String argv[]) {

N n = new N();

int i=2;

N.go(n,i);

System.out.print(n.i + " " + i);

}

}

**a) "1 2" b) "1 3" c) "4 2" d) "4 3"**

Q8) Parmi les déclarations suivantes, sélectionnez l’instruction légale.

interface T {

void t();

}

class M implements T {

void t();

void m();

}

class N extends M {

void n();

}

**a) T t1 = new T();**

**b) T t2 = new N(); t2.n();**

**c) N t3 = new M();**

**d) M t4 = new N();**

Q9) 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**

Q10) Qu’affiche le programme?

class A {

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

int getVal () { return 10; }

}

class B extends A {

int getVal () { return 20; }

}

public class C extends B {

public static void main (String args[]) {

new B().prn();

new C().prn();

}

}

**a) “10”, “10” b) “20”, “10” c) “20”, “20” d) Une erreur**

### 2. Code à commenter (15 points) – durée ≈ 70 minutes

On cherche à gérer une liste d'élèves avec leurs notes et les profs qui les évaluent.

* Chaque prof et élève ont comme propriété commune le nom et le prénom ;
* La élèves sont regroupés dans un tableau ;
* Chaque élève de ce tableau a les informations suivantes : un numéro d’identifiant unique de l’élève, son nom, son prénom et un tableau de 10 notes ;
* Pour chaque note, sont indiqués la valeur de la note, l’élève corrigé et le prof correcteur ;
* Chaque élève ne peut consulter que l’ensemble de ses propres notes ainsi que sa moyenne ;
* Chaque prof peut effectuer les opérations suivantes : rechercher un élève dans le tableau des élèves à partir de son numéro, modifier des notes de l’élève recherché, ainsi que trier les élèves de ce tableau par moyenne des notes.

***Votre code devra être commenté pour plus de clarté.***

* 1. [2 pts] Ecrire une classe *Personne* avec les caractéristiques suivantes :
* Les attributs en visibilité « **protected** » : le nom et le prénom
* Constructeur qui initialise tous les attributs de la personne référencée avec les paramètres

public class Personne {

// Attributs de niveau de visibilité « protected »

protected String nom;

protected String prenom;

public Personne (String nom, String prenom) { this.nom = nom; this.prenom= prenom; }

*}*

* 1. [4 pts] Ecrire la classe *Eleve* qui hérite de *Personne* avec les caractéristiques suivantes :
* Les attributs : le nombre de notes en constante de valeur 10, un tableau d’objets de la classe *Note* (voir question **2.4**) **public** à instancier par le nombre de notes, et un identifiant privé
* Constructeur qui initialise les attributs de l’élève référencé avec ses paramètres, dont le nom et le prénom par héritage
* Méthode *moyenne()* qui calcule et retourne la moyenne des notes de l’élève référencé ou 0 si toutes les notes sont vides.
* L’accesseur en lecture pour l’identifiant.

public class Eleve extends Personne {

// Constante

final int NB\_NOTES = 10 ; // nombre maximum de notes par élève

// attributs

public Note notes[] = new Note[NB\_NOTES] ; // tableau public instancié des notes de l’élève

private int identifiant ; // numéro d’identifiant privé de l’élève

public Eleve (String nom, String prenom, int identifiant) {

super(nom, prenom) ; this.identifiant = identifiant ;

}

public float moyenne () {

int i = 0 ; // nombre de notes

float somme = 0 ; // somme des notes

// calcul de la somme des notes non vides

do {

if (note[i] != null) {

somme += note[i].valeur ;

i++ ;

}

} while (i<10) ;

if (i==0)

return 0 ; // toutes les notes sont vides

else

return somme/i; // retourner la moyenne des notes non vides

}

public int getId() {return identifiant;}

*}*

* 1. [3 pts] Ecrire la classe *CollectionEleves* avec les caractéristiques suivantes :
* Les attributs : le nombre d’élèves en constante et le tableau d’élèves **public** et **static** à instancier par le nombre d’élèves
* Une méthode *rechercher()* **static**, avec en paramètre un identifiant, qui recherche l’identifiant d’un élève dans le tableau des élèves. Si l’élève recherché existe, cette méthode retourne l’élève trouvé. Si l’élève n’existe pas, elle retourne un objet vide

public class CollectionEleves {

// Attributs

final int NB\_ELEVES = 500 ;

public static Eleve liste[] = new Eleve[NB\_ELEVES] ; // tableau publique statique des élèves

public static Eleve rechercher (int identifiant) {

Eleve e = null ; // élève à rechercher

boolean trouve = false ; // booléen indiquant si l’élève existe

int i = 0 ; // indice de la liste

// boucle de recherche dans la liste

while (i<CollectionEleves.liste.length && !trouve) {

e = CollectionEleves.liste[i];

if (e.getIdentifiant() == identifiant)

trouve = true;

i++;

}

if (!trouve)

return null ; // élève non trouvé

else

return e; // retourner l’élève trouvé

}

*}*

* 1. [2 pts] Ecrire la classe *Note*avec les caractéristiques suivantes :
* Les attributs **public** : l’élève corrigé, le prof (voir question **2.5**) correcteur et la valeur de la note
* Le constructeur qui initialise les attributs de la note référencée avec ses paramètres

public class Note {

// attributs publics

public Prof correcteur ; // prof correcteur qui a mis la note

public Eleve corrige ; // élève corrigé

public float valeur ; // valeur de la note

public Note (Prof correcteur, Eleve corrige, float valeur) {

this.correcteur = correcteur ; this.corrige = corrige ; this.valeur = valeur ;

}

*}*

* 1. [4 pts] Ecrire la classe *Prof* qui hérite de *Personne* avec les caractéristiques suivantes :
* Une méthode *setNote* avec 3 paramètres : un identifiant (d’élève), la valeur de la note (voir question **2.4**) et un indice **i** (numéro de case pour le tableau de notes : voir question **2.2).** Cette méthode recherche cet élève dans la *CollectionEleves* : voir méthode *rechercher()* de la question **2.3**. Si l’élève recherché n’existe pas (l’élève est vide), afficher un message d’erreur. Si l’élève existe, modifier la note d’indice **i** dans le tableau des notes : si cette note est vide, instancier cette note. Modifier la valeur de la note avec la valeur en paramètre. Le prof correcteur à modifier est le prof référencé
* La méthode qui trie et affiche les élèves du tableau d’élèves par ordre décroissant de moyenne (voir méthode *moyenne()* de la question **2.2**). Pour chaque élève dont la moyenne n’est pas nulle, cette méthode affichera son nom, son prénom et sa moyenne.

public class Prof extends Personne {

public void setNote (int identifiant, float valeur, int i) {

Eleve corrige ; // déclarer l’élève corrigé à rechercher

// rechercher l’élève corrigé dans la collection des élèves

corrige = CollectionEleves.rechercher(identifiant) ;

if (corrige==null) {

System.out.println(″Cet élève est inexistant″) ;

}

else if (corrige.note[i] == null) // si la note d’indice i de l’élève est vide

corrige.note[i] = new Note(this, corrige, valeur);

else {

corrige.note[i].valeur = valeur;

corrige.note[i].correcteur = this;

}

}

// Méthode de tri par moyenne décroissante pour les élèves ayant au moins une note

public void trier () {

Eleve listetriee = new Eleve[NB\_ELEVES] // liste à trier instanciée

int i, j; // indices de parcours de la liste d’élèves

Eleve eleve1, eleve2 ;

float moy1, moy2 ;

// ajout des élèves ayant une moyenne dans la liste à trier

for (i=0 ; i < CollectionEleves.liste.length ; i++) {

eleve1 = CollectionEleves.liste.[i];

moy1 = eleve1.moyenne(); // moyenne de l’élève

listetriee[i]=eleve1 ; // ajouter l’élève dans la liste à trier

}

// trier la liste à trier par moyenne décroissante

for (i=0 ; i < listetriee.length-1 ; i++)

for (j=i+1 ; j < listetriee.length ; j++) {

// Récupérer chaque paire d’élèves pour comparer leur moyenne

eleve1 = listetriee.[i];

moy1 = eleve1.moyenne();

eleve2 = listetriee.[j];

moy2 = eleve2.moyenne();

if (moy2 > moy1) {

// Permuter les 2 élèves dans la liste à trier

listetriee[i] = eleve2 ;

listetriee[j] = eleve1 ;

}

}

// Afficher les élèves de la liste triée

for (i=0 ; i < listetriee.length ; i++) {

eleve1 = listetriee.[i];

if (eleve1.moyenne() != 0)

System.out.println(eleve1.nom+” ; ”+ eleve1.prenom+” ; ”+ eleve1.moyenne());

}

}

}