Examen de TOB/LO
(1h45, avec documents)

NOM : **Prénom :** **Signature :**

Remarques préliminaires.

– Ne pas mettre de commentaires de documentation sauf s'ils sont nécessaires à la compréhension.

– Barème indicatif :

exercice	1	2	3	4	5	6	7	8	9	10
point(s)	4	1	2	2	1	1	2	3	2	2

1 Compréhension du cours

Exercice 1 Répondre de manière concise aux questions suivantes.

1.1 On distingue généralement deux types d'égalité.

1.1.1 Expliquer la différence entre égalité logique et égalité physique.

1.1.2 Indiquer comment s'exprime l'égalité logique en Java

1.1.3 Indiquer comment s'exprime l'égalité physique en Java

1.2 Considérons la déclaration du listing suivant. Nous supposons qu'elle apparaît dans la classe `Point` vue en cours, TD et TP.

```
1 public class Point {
2    public static final Point origine = new Point(0, 0);
3    ...
4 }
```

1.2.1 Expliquer ce que signifient les mots-clés **`public`**, **`static`** et **`final`**.

1.2.2 Indiquer, en justifiant la réponse, si on peut être sûr que l'attribut `origine` de la classe `Point` aura toujours pour coordonnées $(0, 0)$.

1.3 On considère la classe TestPermuter dont le code est donné au listing 1.

Listing 1 – La classe TestPermuter

```
1 class TestPermuter {
2
3    public static void permuter(double a, double b) {
4       double tmp = a;
5       a = b;
6       b = tmp;
7    }
8
9    public static void main(String[] args) {
```

```
10          double x1 = 1.0;
11          double x2 = 2.0;
12          permuter(x1, x2);
13          System.out.println("x1 = " + x1);
14          System.out.println("x2 = " + x2);
15      }
16  }
```

1.3.1 Expliquer pourquoi il est obligatoire ici de déclarer la méthode `permuter` en **`static`**.

1.3.2 Indiquer comment utiliser la méthode `permuter` depuis une autre classe appelée `A`.

1.3.3 Indiquer ce qui s'affiche à l'écran quand on exécute la classe TestPermuter et expliquer pourquoi.

1.4 On considère une interface I qui spécifie la méthode m() et une poignée p non nulle déclarée de type I, expliquer pourquoi on est sûr que p.m() a un sens, c'est-à-dire qu'il y a un code défini pour m(). Indiquer quel est le code qui sera exécuté.

2 Jeu du Devin

Le jeu du devin se joue à deux joueurs. Le premier joueur, le *maître*, choisit un nombre compris entre 1 et 999. Le second, le *devin*, doit le trouver en un minimum d'essais. À chaque proposition du devin, le maître indique si le nombre proposé est trop grand, trop petit ou le nombre à trouver. Quand le nombre est trouvé par le devin, la partie est finie et le nombre d'essais est affiché.

On veut écrire une version du devin qui permet de choisir l'intervalle dans lequel le nombre doit être cherché, choisir si le devin est un humain ou l'ordinateur, choisir si le maître est un humain ou l'ordinateur, proposer différentes manière de jouer pour le devin ou le maître, etc.

Pour représenter l'évaluation faite par le maître et utilisée par le devin, on utilisera le type Evaluation suivante :

```
1  public enum Evaluation { TROP_GRAND, TROP_PETIT, BON }
```

Il s'agit d'un type énuméré de Java. Les instructions

```
Evaluation evaluation = Evaluation.TROP_GRAND;
System.out.println("Le nombre est " + evaluation);
```

affichent :

Listing 2 – Raffinage du jeu du devin entre un maître et un esclave

```
1  Comment « Arbitrer un jeu entre un devin et un maître » ?
2     Indiquer au devin les limites du jeu
3     Demander au maître de choisir un nombre
4     Répéter
5        Demander au devin de faire une proposition
6        Demander au maitre d'évaluer la proposition
7        Donner au devin le résultat de l'évaluation
8     JusquÀ nombre trouvé
9     Féliciter le devin
```

```
Le nombre est TROP_GRAND.
```

Rappel : Il est possible de faire un **`switch`** sur une expression de type énuméré.
Rappel : La classe `java.util.Random` fournit une méthode `nextInt(int n)` qui retourne un nombre aléatoire compris entre 0 et n-1.

Exercice 2 : La classe Joueur
Le listing 3 présente le texte de la classe Joueur. Cette classe représente un joueur du jeu du devin, soit un devin, soit un maître. La classe Joueur est déclarée abstraite alors qu'elle ne contient pas de méthode abstraite.

2.1 Expliquer ce qu'implique de déclarer une classe abstraite et ce qui le justifie ici.

2.2 Expliquer ce que signifie **`final`** devant l'attribut `nom`.

Listing 3 – La classe Joueur

```
1  abstract public class Joueur {
2
3     final private String nom;
4
5     public Joueur(String nom) {
6        this.nom = nom;
7     }
8
9     public String getNom() {
10       return this.nom;
11    }
12
13 }
```

Exercice 3 : Le Maître
Le maître peut être un humain ou un ordinateur. Il s'agit d'un joueur chargé de :
- choisir un nombre compris dans les limites définies par l'arbitre (intervalle [min, max]),
- évaluer une proposition.

Le listing 4 donne le code de classe Maitre. Malheureusement, ce code est imcomplet.

3.1 Indiquer la relation entre Maitre et Joueur.

3.2 Expliquer pourquoi il est nécessaire de définir un constructeur sur la classe Maitre.

3.3 Expliquer pourquoi les méthodes choisirNombre et evaluationProposition sont abstraites.

3.4 Compléter (sur le sujet) le code de la classe Maitre.

Listing 4 – La classe Maitre

```
1  abstract public class Maitre
2  {
3
4
5
6
7     /** Choisir le nombre à faire trouver. */
8     abstract public void choisirUnNombre(int min, int max);
9
10    /** Donner l'évaluation d'une proposition. */
11    abstract public Evaluation evaluation(int proposition);
12
13 }
```

Exercice 4 : La classe MaitreOrdinateur
La classe MaitreOrdinateur modélise un ordinateur qui joue le rôle de maître. Ainsi, il choisit aléatoirement un nombre dans l'intervalle proposé et évalue la proposition faite en fonction du nombre qu'il a choisi préalablement. Écrire la classe MaitreOrdinateur.
Remarque : Sur le même principe, on pourrait définir une classe MaitreHumain mais le code de cette classe n'est pas demandé.

Exercice 5 : L'exception `TricheException`
Définir une exception `TricheException` non contrôlée par le compilateur.

Exercice 6 : Le Devin
Plusieurs devins sont envisageables : un devin qui correspond à un humain, un devin qui correspond à l'ordinateur, un devin qui utilise une dichotomie pour trouver le nombre, etc.

Étant donné le raffinage proposé au listing 2, écrire la classe Devin.

Exercice 7 : Le Devin avec dichotomie
Écrire une classe `DevinDichotomie` qui utilise une recherche par dichotomie pour trouver le nombre à trouver. Une exception `TricheException` doit être levée si le devin se rend compte qu'il y a triche (l'intervalle de recherche est vide).

Exercice 8 : Le Devin avec mémoire
Écrire une classe DevinElephant qui choisit aléatoirement le nombre à proposer mais vérifie qu'il ne l'a pas déjà proposé.

Exercice 9 : L'arbitre
Écrire le code de la méthode arbitrer correspondant au raffinage du listing 2 et affichant en plus :
- le message triche si le devin détecte que le maître triche
- le nombre d'essais utilisés pour trouver le nombre

Exercice 10 : Diagramme de classe
Dessiner le diagramme de classe UML qui fait apparaître les classes qui composent cette application. On utilisera une représentation requêtes/commandes sans faire apparaître les constructeurs.