# Fiche d’exercices – Modularité

Pour chaque exercice ci-dessous, fournir le code **Java**, ainsi que la **modélisation UML**. Vous pouvez utiliser pour cela *Visual Paradigm for NetBeans***1**.

1. Ecrire une application à l’aide de la classe Etudiant (réalisée dans la fiche d’exercice précédente) permettant à un enseignant de renseigner les notes pour un groupe d’étudiants au collège et permettant la restitution des informations suivantes :
	* le résultat par élève ;
	* la moyenne du groupe ;
	* la note maximale du groupe ;
	* la note minimale du groupe.

A partir de la classe proposée, répondre aux questions suivantes :

1. Comment avez-vous séparé la notion d’élève et celle de groupe ?
2. Comment avez-vous séparé la saisie des notes et le calcul des moyennes ?
3. Avez-vous pu réutiliser la classe précédemment définie sans la modifier ?
4. Proposer une classe de calculs statistiques permettant le calcul de la moyenne d’un tableau de valeurs ( *double[]* ). Etendre cette classe afin de permettre le calcul de l’écart-type.

$$écart=\sqrt{\frac{\sum\_{}^{}\left(x-moyenne\right)^{2}}{nombre d'éléments}} $$

A partir de la classe proposée, répondre aux questions suivantes :

1. Avez-vous pu réutiliser le premier code proposé ? Avez-vous recopié le code ? Si oui, pourquoi ?
2. Analyser et critiquer le code ci-dessous. Justifier vos positions et corriger le code si nécessaire.

|  |  |
| --- | --- |
| 1234567891011121314151617181920212223242526272829303132333435363738 | public class **CoordonneesGPS** { double[] coord; //latitude et longitude String nom; //nom de la ville/localité public double[] **getCoord**() { return coord; } public void **setCoord**(double[] coord) { this.coord = coord; } public String **getNom**() { return nom; } public void **setNom**(String nom) { this.nom = nom; }  **public void show() {**  //on affiche les coordonnées en format dd ° mm ' ss " System.out.println("Lat : " + transforme(coord[0])  + "Lon : " + transforme(coord[1])); } **public void saisie()** { Scanner sc = new Scanner(System.in); coord = new double[2]; System.out.println("Latitude :"); coord[0] = sc.nextDouble(); System.out.println("Longitude :"); coord[1] = sc.nextDouble(); } **protected String transforme(double value) {** StringBuilder coord = new StringBuilder(); double totalSec; long l; //on transforme une coordonnée de degrés en dd ° mm ' ss " totalSec = value \* 3600; l = (long) totalSec / 3600; coord.append(l); coord.append("° ");  l = (long) (totalSec % 3600) / 60; coord.append(l); coord.append("' ");  l = (long) (totalSec % 3600) % 60; coord.append(l); coord.append("'' ");  return coord.toString(); }} |

1. Construire une classe TableauEntier, représentant un tableau d’entiers et respectant les indications suivantes :
	* Le nombre de cases doit être fixe, fourni par l’utilisateur (taille) ;
	* L’accès aux cases est direct, à partir de leur indice ;
	* Les indices des cases varient entre 0 et taille-1 ;
	* L’utilisateur lui-même décide de comment il remplit le tableau, dans quel ordre, quelle case il remplit, etc., à l’aide des indices ;
	* Le tableau doit pouvoir comporter n’importe quel numéro entier : -2, -1, 0, 1, 2…
	(tableau[i] ∈ℵ)
	* Le tableau doit contenir uniquement des *int* (pas *Integer* ou autre classe d’objet)
	* **Il est *interdit* d’utiliser les collections définies dans java.util.\* (ArrayList, Vector…)**

A partir de la classe proposée, répondre aux questions suivantes :

1. Comment avez-vous fait la gestion des cases vides ?
2. Comment avez-vous géré la suppression d’une case ?
3. Construire une application simple permettant à un utilisateur de remplir un tableau d’entier dans l’ordre qu’il le souhaite. L’application doit permettre à l’utilisateur, une fois le tableau rempli, de supprimer certains de ses éléments. Le tableau final doit alors être affiché à l’utilisateur.
4. Construire une classe TableauObjets, représentant un tableau d’objets et respectant les indications suivantes :
	* Le nombre de cases doit être fixe, fourni par l’utilisateur (taille) ;
	* L’accès aux cases est direct, à partir de leur indice ;
	* Les indices des cases varient entre 0 et taille-1 ;
	* L’utilisateur décide lui-même de comment remplir les cases et dans quelle ordre grâce aux indices ;
	* Le tableau doit pouvoir comporter des objets java de n’importe quelle classe, mais tous les objets doivent appartenir à une seule et unique classe ;
	* **Il est *interdit* d’utiliser les collections définies dans java.util.\* (ArrayList, Vector…)**

A partir de la classe proposée, répondre aux questions suivantes :

1. Comment avez-vous fait la gestion des cases vides ?
2. Comment avez-vous géré la suppression d’une case ?
3. Comment avez-vous pour assurer que tous les objets appartiennent à la même classe ?
4. Proposer une application qui teste les fonctionnalités de la classe TableauObjets et notamment la suppression d’un élément au centre du tableau et l’insertion d’objets de classes distinctes dans un même tableau (par exemple, un objet Integer et un objet String).

A partir de l’application proposée, répondre aux questions suivantes :

1. La suppression d’un élément au centre du tableau est-elle différente de la suppression d’un élément au début (le premier) ou à la fin (le dernier) du tableau ? Pourquoi ?

**1 Observations**:

* Le modèle UML fourni doit correspondre au code Java présenté.
* Lorsqu’il vous sera demander de rendre l’exercice, vous devez compacter la totalité du projet NetBeans (y compris le projet Visual Paradigm) dans un fichier zip portant votre nom et le numéro de l’exercice. Exemple : Nom-FicheIntro-Exercice1.zip.
* Lorsque des questions sont proposées avec les exercices, répondez à chaque question tout au début du code Java proposé à l’aide des commentaires (/\* … \*/).