Définition et utilisation d'une classe `Etudiant` : classes, instances, méthodes simples, instructions conditionnelles

1 Définition de la structure de la classe

Nous vous proposons d'utiliser Eclipse qui va vous aider dans l'écriture du code. Vous allez définir en parallèle :

- Une classe `Etudiant` (sans main) qui décrira ce qu'est un objet étudiant et quelles opérations peuvent être appliquées à un étudiant.
- Une classe `ProgrammeEtudiant` (avec un main) qui permettra de créer des objets étudiants et d'appeler les différentes opérations.

1.1 Données de la classe `Etudiant`

Dans cette première partie, nous définissons la partie structurelle de la classe, c'est-à-dire ses attributs. La classe sera définie dans un premier fichier `Etudiant.java` dans un paquetage `etudiant`.
Vous choisirez les types des attributs. Les attributs devront être privés (leur type sera précédé du mot clef `private`). Un étudiant devra être décrit par :

- un nom
- trois notes (codées par un nombre réel) : une note de programmation, une note de système et une note de stage
- un âge
- un code d'inscription (codé par un entier) qui vaut 1 si c'est la première inscription de l'étudiant, et 2 s'il s'agit d'une réinscription
- un code de pays (codé par un entier) qui vaut 1 pour un étudiant français, 2 pour un étranger non francophone et 3 pour un étranger francophone

1.2 Première utilisation dans une classe-programme `ProgrammeEtudiant`

Définissez, dans le même paquetage, mais dans un deuxième fichier, nommé `ProgrammeEtudiant.java`, une classe possédant un main. Créez un étudiant, affichez un message, puis exécutez le programme. Cela vous permettra de vérifier que la classe-programme connaît l'autre classe.

2 Définition des constructeurs et accesseurs

Eclipse va maintenant vous être d'une grande aide pour créer les constructeurs et les accesseurs.

2.1 Constructeurs et accesseurs de la classe `Etudiant`

Pour générer des constructeurs à partir des attributs existant dans la classe : sur le code de la classe `Etudiant`, faire un clic droit → source → generate constructors using fields. Sélectionner dans le wizard qui apparaît les champs que l'on veut prendre en paramètres du constructeur. Pour l'instant, cocher la case *omit call to default constructor super()*. Prévoyez un constructeur qui initialise tous les attributs, et un constructeur qui initialise tout sauf les notes de l'étudiant.
Pour générer les accesseurs aux attributs existant dans la classe : sur le code, faire un clic droit → source → generate getters and setters. Les attributs sont proposés dans un wizard, où on choisit les attributs pour lesquels on désire générer les accesseurs et le point où ils vont être introduits.

2.2 Création d'étudiants dans une classe-programme `ProgrammeEtudiant`

Vous pouvez maintenant compléter la classe-programme en créant des étudiants avec des valeurs particulières pour leurs attributs. Puis à l'aide des accesseurs, vous pouvez modifier ces attributs.

- Créez Jean, étudiant québecois francophone, âgé de 24 ans, qui s'inscrit pour la première fois, sans notes connues,
- Créez Abdoulkhader, étudiant tunisien francophone, âgé de 23 ans, qui se réinscrit, sans notes connues,
- Créez Astrid, étudiante finlandaise non francophone, âgée de 26 ans, qui s'inscrit pour la première fois, sans notes connues,
- Créez Paolo, étudiant brésilien non francophone, âgé de 27 ans, qui s'inscrit pour la première fois, sans notes connues,
- Créez Zoé, étudiante française, âgée de 26 ans, qui s'inscrit pour la première fois, avec les notes 12 (programmation), 14 (système), 17 (stage).
- Donnez à Astrid les notes 16 (programmation), 15 (système), 14 (stage).
- Modifiez la note de programmation de Zoé qui devient 15.

3 Définition des méthodes

3.1 Amélioration des accesseurs et des constructeurs

Dans `Etudiant`. Modifiez les accesseurs

- aux attributs qui représentent des notes afin de n'accepter que des réels entre 0 et 20
- au code d'inscription afin de n'accepter qu'un entier égal à 1 ou 2
- au code pays afin de n'accepter qu'un entier égal à 1, 2 ou 3

Utilisez ces accesseurs dans vos constructeurs afin de contrôler la bonne initialisation des objets. Un message d'erreur sera affiché si la valeur de l'attribut n'appartient pas au domaine de valeurs.

Dans `ProgrammeEtudiant`. Testez la possibilité de changer la note de système de Zoé avec 22.

3.2 Définition d'une méthode toString

Soit `zo` la variable qui référence l'objet représentant Zoé. Testez dans le main l'instruction suivante : `System.out.println(zo);`
Ajoutez à la classe `Etudiant` une méthode `toString` qui retourne une chaîne contenant le nom et l'âge d'un étudiant. Re-testez `System.out.println(zo);`. Conclusion ?
Vous pourrez faire évoluer cette méthode au fur et à mesure de l'avancement de votre programme afin de tester l'état de vos objets.

3.3 Méthode moyenne

Ajoutez à la classe `Etudiant` une méthode `moyenne` qui retourne la moyenne d'un étudiant. Dans `ProgrammeEtudiant`, donnez des notes à tous vos étudiants et appelez `moyenne` pour vérifier son bon fonctionnement.

3.4 Méthode mention

Ajoutez à la classe `Etudiant` une méthode `mention` qui retourne une chaîne de caractères correspondant à la mention d'un étudiant. Dans `ProgrammeEtudiant`, appelez `mention` sur vos étudiants pour vérifier son bon fonctionnement.

3.5 Méthode `ligneResultats`

Ajoutez à la classe `Etudiant` une méthode `ligneResultats` qui retourne une chaîne de caractères d'une ligne précisant le nom, la moyenne et la mention, et, seulement s'il est ajourné, les modules obtenus (c'est-à-dire ceux où il a obtenu la moyenne). Dans `ProgrammeEtudiant`, appelez `ligneResultats` sur vos étudiants pour vérifier son bon fonctionnement.

3.6 Méthode `saisie`

Ajoutez à la classe `Etudiant` une méthode `saisie` qui prend un scanner en paramètre et récupère sur ce scanner toutes les informations concernant un étudiant. Dans `ProgrammeEtudiant`, appelez `saisie` sur vos étudiants pour vérifier son bon fonctionnement.

4 Introduction aux énumérations

Nous voudrions à présent définir un attribut décrivant la situation financière de l'étudiant. Celui-ci peut être principalement boursier, salarié ou avoir un autre financement. On peut utiliser un codage par un entier pour ces trois situations comme nous l'avons fait pour les codes pays et codes inscription. Une autre solution consiste à définir un type énumération qui regroupe un ensemble de constantes.
Pour le créer en Eclipse, ajoutez au package une `enum` par : File → new → Enum).

```
public enum SituationFinanciere {
 boursier, salarie, autre;
}
```

L'accès à la constante `boursier` sera réalisé grâce à l'expression `SituationFinanciere.boursier`.
Vous pouvez maintenant ajouter dans votre classe un attribut décrivant la situation financière d'un étudiant. Quelles modifications doit comporter votre code précédent pour le gérer ? Mettez-les en œuvre et testez en créant un étudiant boursier. Cherchez notamment à saisir au clavier et à afficher la situation financière.
Vous êtes en mesure d'améliorer la définition des attributs `codeIns` et `codePays` en créant des énumérations.