**Système de Gestion de Quiz - Rapport Technique**

**1. Présentation du Projet**

Le Système de Gestion de Quiz est une application Java conçue pour faciliter la création, la gestion et la participation aux quiz dans un environnement éducatif. Le système offre des interfaces distinctes pour les enseignants et les étudiants, permettant une interaction complète avec les quiz.

**2. Architecture du Système**

**Structure des Classes**

Le système est composé de quatre classes principales :

1. **Main** : Point d'entrée et flux de contrôle principal de l'application
2. **Etudiant** : Représente les entités étudiantes et leurs activités liées aux quiz
3. **Quiz** : Gère la création, le stockage et l'évaluation des quiz
4. **Qcm (Question à Choix Multiples)** : Définit la structure et le comportement des questions individuelles de quiz

**Diagramme des Relations**

Main --> Quiz : gère

Main --> Etudiant : gère

Quiz "\*" --> "\*" Qcm : contient

Quiz --> Etudiant : suit les réponses

**3. Fonctionnalités Principales**

**Fonctionnalités Enseignant**

* Créer de nouveaux quiz
* Visualiser les quiz existants
* Modifier les questions de quiz
* Supprimer des quiz
* Visualiser la participation des étudiants
* Analyser les statistiques de quiz

**Fonctionnalités Étudiant**

* S'inscrire/Se connecter
* Consulter les quiz disponibles
* Passer des quiz
* Consulter ses scores personnels
* Consulter les corrections de quiz

**4. Détails Techniques de l'Implémentation**

**Structures de Données**

* ArrayList<Quiz> : Stocke tous les quiz créés
* ArrayList<Etudiant> : Stocke les étudiants inscrits
* HashMap<Etudiant, Integer[]> : Suit les réponses des étudiants
* HashMap<String, Integer> : Stocke les scores des étudiants

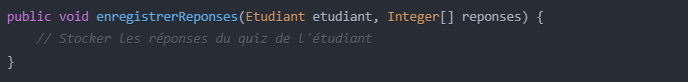
**Méthodes Clés**

**Création de Quiz**

A screenshot of a computer

Description automatically generated

**Suivi des Réponses des Étudiants**



**Génération de Statistiques de Quiz**

**A black background with white text

Description automatically generated**

**5. Flux d'Expérience Utilisateur**

**Workflow Enseignant**

1. **Connexion en mode enseignant**
2. **Choisir une action (créer/modifier/supprimer un quiz)**
3. **Effectuer l'action sélectionnée**
4. **Visualiser les statistiques du quiz**

**Workflow Étudiant**

1. **S'inscrire ou se connecter**
2. **Consulter les quiz disponibles**
3. **Sélectionner et compléter un quiz**
4. **Consulter le score et la correction**

**6. Défis Techniques et Solutions**

**Défi : Suivi des Réponses**

**Solution : Utilisation de HashMap<Etudiant, Integer[]> pour mapper efficacement les étudiants à leurs réponses de quiz**

**Défi : Modification Dynamique de Quiz**

**Solution : Implémentation de méthodes comme ajouterOption() et setQuestion() pour permettre l'édition flexible des quiz**

**7. Améliorations Potentielles**

* **Implémenter un stockage persistant (intégration de base de données)**
* **Ajouter une analyse statistique plus avancée**
* **Créer une interface graphique utilisateur (GUI)**
* **Implémenter une authentification utilisateur avec mots de passe**

**8. Conclusion**

**Le Système de Gestion de Quiz offre une solution robuste et orientée objet pour la gestion de quiz éducatifs, proposant flexibilité et fonctionnalités complètes pour les enseignants et les étudiants.**

### **Description détaillée de la classe Quiz:**

**La classe Quiz gère la structure et les fonctionnalités d'un quiz. Elle regroupe les QCM (questions à choix multiples), enregistre les réponses des étudiants, et fournit des outils pour analyser les résultats et les performances.**

### **Attributs**

1. **theme (String)**
   * **Représente le thème ou le module du quiz (exemple : "POO", "Base de données").**
   * **Utilisé pour identifier le quiz.**
2. **auteur (String)**
   * **Le nom de l’enseignant créateur du quiz.**
3. **nbqcm (int)**
   * **Nombre de QCM contenus dans le quiz.**
4. **qcmTable (ArrayList<Qcm>)**
   * **Liste des questions (QCM) constituant le quiz.**
   * **Chaque élément est une instance de la classe Qcm.**
5. **reponsesEtudiants (HashMap<Etudiant, Integer[]>)**
   * **Associe chaque étudiant à un tableau de réponses.**
   * **Chaque réponse est un entier correspondant à l’index de l'option choisie pour chaque QCM.**

### **Méthodes**

#### **1. creerquiz()**

* **Permet à l’enseignant de créer un nouveau quiz.**
* **Demande à l'utilisateur d'entrer :**
  + **Le thème du quiz.**
  + **Le nom de l'auteur.**
  + **Le nombre de questions.**
* **Itère pour créer chaque QCM via la méthode creerqcm() de la classe Qcm.**
* **Stocke chaque QCM dans la liste qcmTable.**

#### **2. visualiserquiz()**

* **Affiche les informations du quiz : thème, auteur, et toutes les questions (QCM).**
* **Utilise la méthode toString() de la classe Qcm pour formater l'affichage des questions.**

#### **3. ajouterQcm(Qcm qcm)**

* **Ajoute un nouvel objet Qcm à la liste qcmTable.**
* **Permet d’étendre un quiz existant.**

#### **4. enregistrerReponses(Etudiant etudiant, Integer[] reponses)**

* **Enregistre les réponses d’un étudiant pour ce quiz.**
* **Associe l’étudiant à un tableau de réponses dans la map reponsesEtudiants.**

#### **5. enregistrerScoreEtudiant(Etudiant etudiant, int score)**

* **Sauvegarde le score total obtenu par un étudiant pour ce quiz.**
* **Met à jour les scores dans l'objet Etudiant.**

#### **6. afficherStatistiques()**

* **Analyse les réponses pour chaque QCM et affiche :**
  + **Le pourcentage de réponses justes.**
  + **Le pourcentage de réponses fausses.**
* **Utilise les données de reponsesEtudiants et compare les réponses des étudiants avec l'index de la réponse correcte pour chaque QCM.**

#### **7. visualiserEtudiantsAvecScores()**

* **Affiche la liste des étudiants ayant participé au quiz, avec leur score.**
* **Pour chaque étudiant :**
  + **Compare leurs réponses avec les réponses correctes.**
  + **Calcule le score total.**

#### **8. afficherClassement()**

* **Trie les étudiants ayant participé au quiz en fonction de leur score.**
* **Affiche les étudiants dans un ordre décroissant, avec leur rang.**

### **Getters et Setters**

**Pour assurer l'encapsulation et faciliter l'accès aux attributs, des getters et setters sont définis pour chaque attribut.**

* **Getters :**
  + **getTheme(), getAuteur(), getNbqcm(), getQcmTable(), getReponsesEtudiants()**
  + **Permettent de récupérer les informations sur le quiz et les réponses.**
* **Setters :**
  + **setTheme(String theme), setAuteur(String auteur), setNbqcm(int nbqcm)**
  + **Permettent de modifier les propriétés du quiz.**

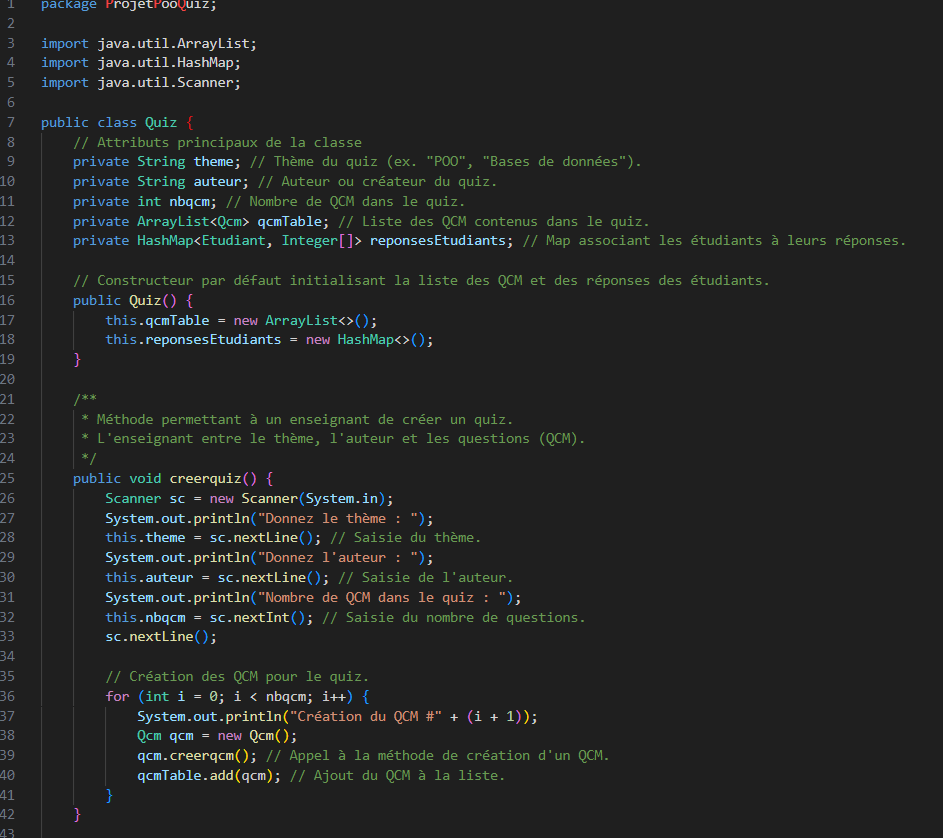
### **Scénarios d'utilisation**

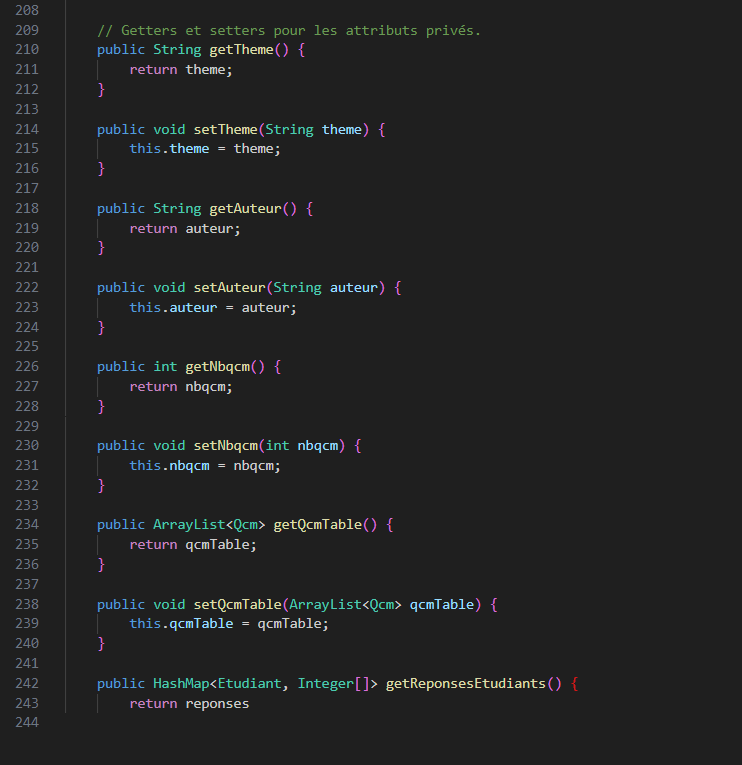
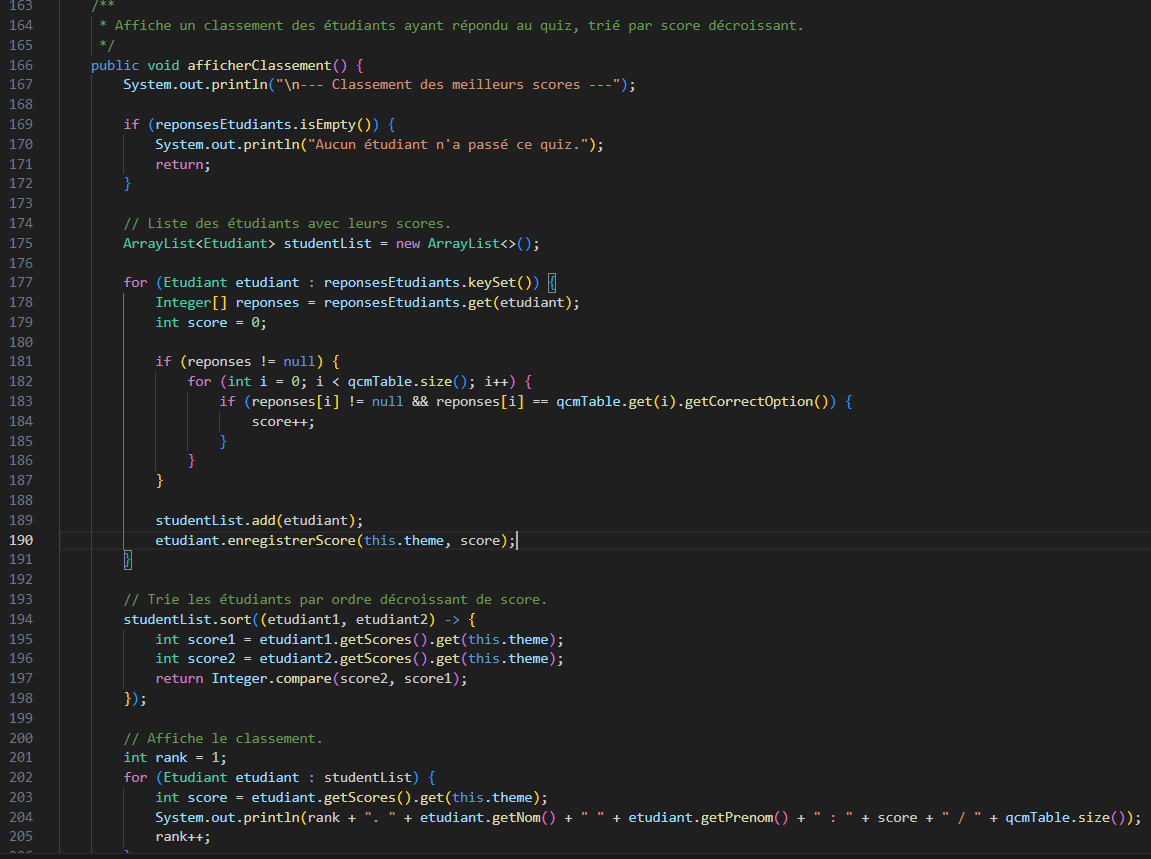
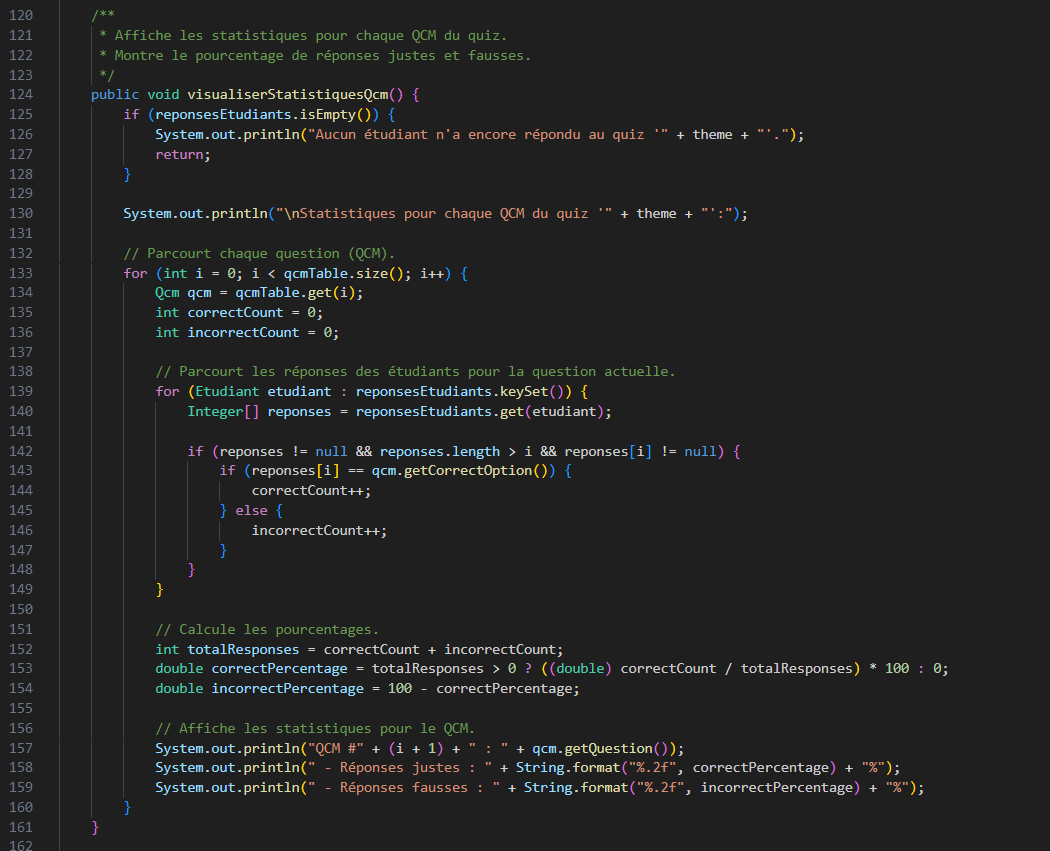
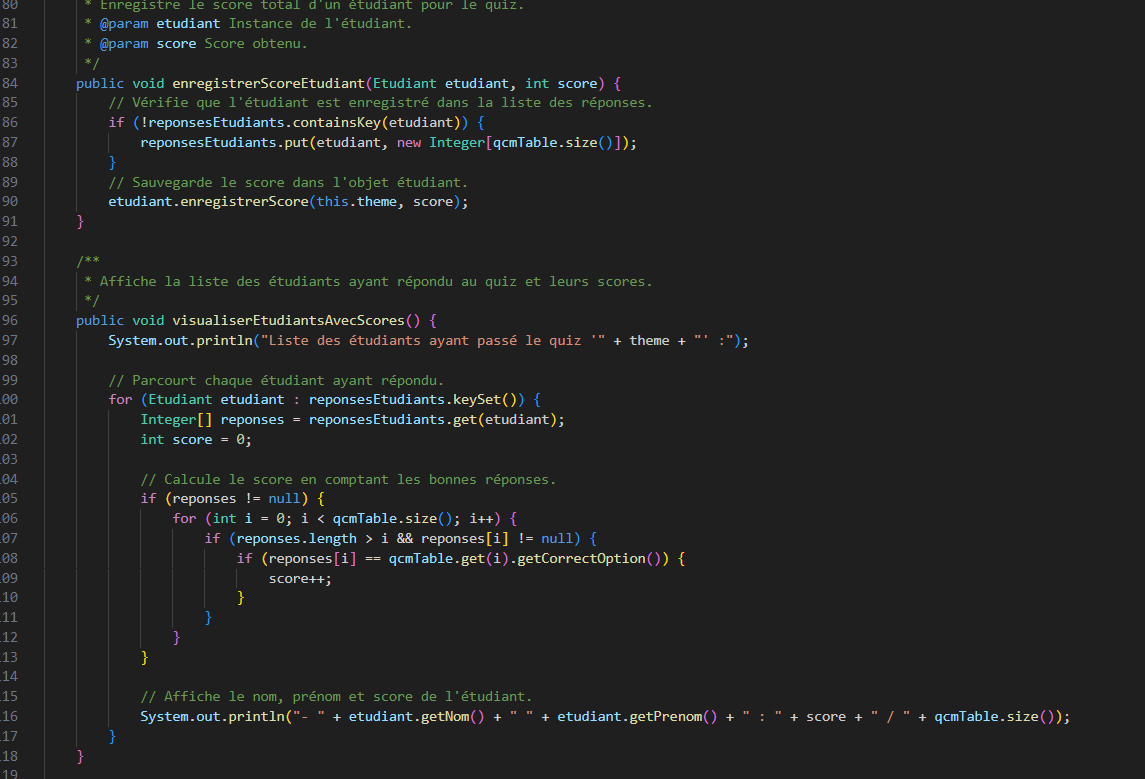
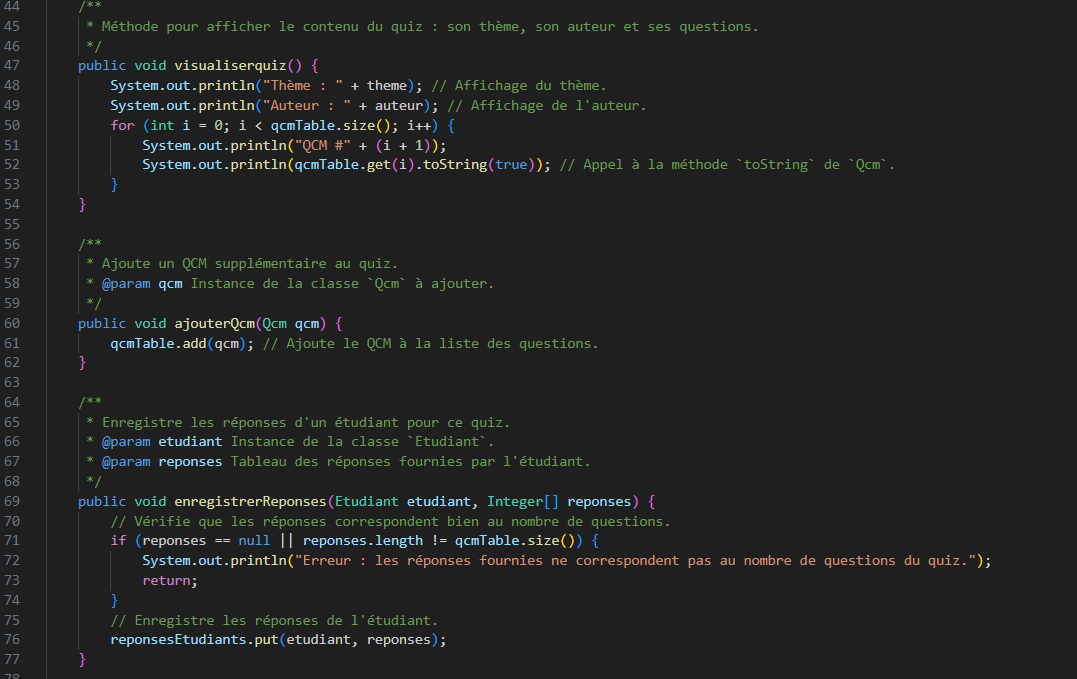
1. **Création d'un quiz :**
   * **L'enseignant utilise creerquiz() pour définir le thème et ajouter des questions (QCM).**
   * **Chaque question est configurée avec des options et une réponse correcte via la classe Qcm.**
2. **Participation d'un étudiant :**
   * **Les réponses de l'étudiant sont enregistrées via enregistrerReponses().**
3. **Analyse des résultats :**
   * **Les scores des étudiants sont sauvegardés avec enregistrerScoreEtudiant().**
   * **L'enseignant peut afficher les statistiques de performance pour chaque QCM avec afficherStatistiques().**
4. **Classement :**
   * **L'enseignant visualise le classement des étudiants avec afficherClassement().**

### **Avantages de la conception**

* **Flexibilité :  
  Les méthodes comme ajouterQcm() permettent de modifier ou enrichir un quiz existant.**
* **Encapsulation :  
  Les getters et setters protègent les données sensibles et permettent un contrôle sur leur accès.**
* **Modularité :  
  Les fonctionnalités (création, gestion, analyse) sont séparées, ce qui rend le code maintenable.**

**Cette classe est essentielle pour gérer les fonctionnalités éducatives, l'évaluation des connaissances et l'analyse des performances des étudiants.**

****

****

### **Explication détaillée des méthodes et attributs de la classe Etudiant :**

#### **Attributs**

1. **nom et prenom : Identité de l'étudiant.**
2. **cin : Numéro unique pour identifier chaque étudiant.**
3. **nbgrp : Numéro de groupe académique de l'étudiant.**
4. **modulesRepondus : Liste des modules auxquels l'étudiant a déjà participé.**
5. **scores : Dictionnaire associant chaque module au score obtenu.**

#### **Méthodes**

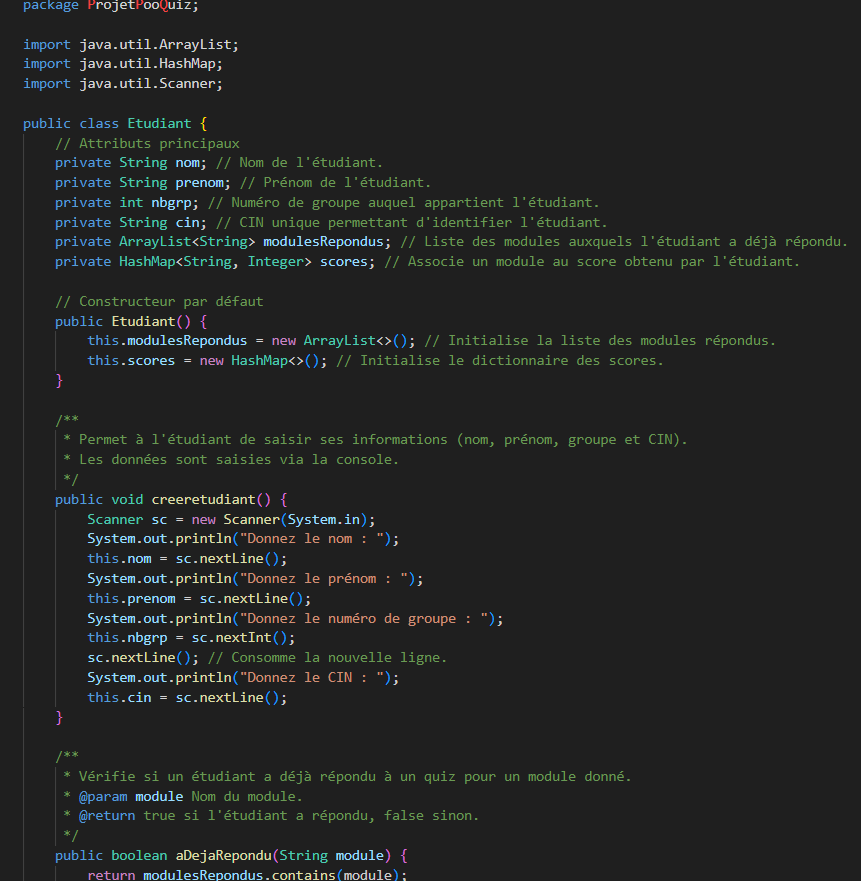
1. **creeretudiant()**
   * **Permet à l'utilisateur de saisir ses informations via la console.**
   * **Enregistre les données dans les attributs correspondants.**
2. **aDejaRepondu(String module)**
   * **Vérifie si un étudiant a déjà répondu au quiz d'un module.**
   * **Retourne true si le module est trouvé dans modulesRepondus, false sinon.**
3. **ajouterModuleRepondu(String module)**
   * **Ajoute un module à la liste des modules auxquels l'étudiant a répondu.**
   * **Évite les doublons grâce à une vérification préalable.**
4. **enregistrerScore(String module, int score)**
   * **Enregistre ou met à jour le score pour un module donné dans le dictionnaire scores.**
5. **afficherScores(ArrayList<Quiz> quizTable)**
   * **Affiche les scores obtenus par l'étudiant pour chaque module.**
   * **Utilise la liste des quiz pour trouver le nombre total de questions (points) pour chaque module.**

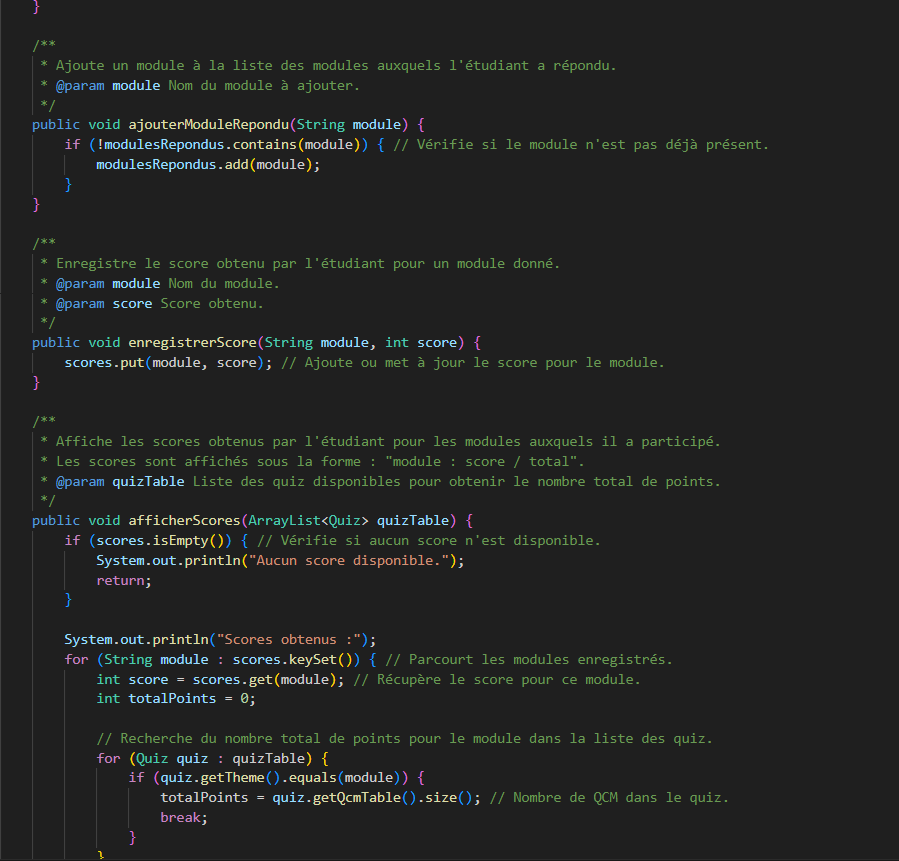
#### **Getters et Setters**

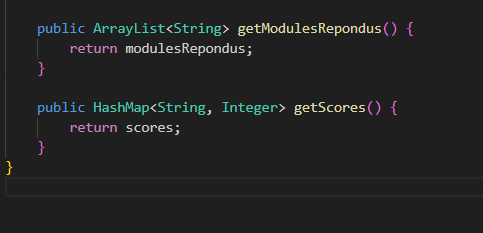
* **Permettent un accès contrôlé aux attributs privés.**
* **Utiles pour assurer l'encapsulation et la sécurité des données.**

### **Utilité de la classe Etudiant**

* **Elle centralise les informations et actions relatives aux étudiants.**
* **Permet de gérer les participations, scores et modules.**
* **Fournit un moyen de lier un étudiant à ses réponses dans un quiz spécifique (Quiz).**

****

****

****

### **Explication détaillée de la classe Qcm**

**La classe Qcm représente une question à choix multiples (QCM) dans un quiz. Elle centralise les informations essentielles pour gérer une question individuelle : son texte descriptif, les options proposées, et la réponse correcte.**

### **Attributs**

1. **question *(String)* :**
   * **Contient le texte de la question posée.**
   * **Exemple : *"Quelle est la capitale de la France ?"*.**
2. **options *(String[])* :**
   * **Tableau qui stocke toutes les options possibles pour répondre à la question.**
   * **Exemple : *{"Paris", "Londres", "Berlin", "Madrid"}*.**
3. **correctOption *(int)* :**
   * **Indique l'index (basé sur 0) de l'option correcte dans le tableau options.**
   * **Exemple : Si la bonne réponse est "Paris", alors correctOption = 0.**
4. **nbop *(int)* :**
   * **Stocke le nombre total d'options proposées pour cette question.**

### **Méthodes**

#### **1. creerqcm()**

* **But : Cette méthode permet de créer une question QCM en saisissant :**
  1. **Le texte de la question.**
  2. **Les options de réponse.**
  3. **L'index de l'option correcte.**
* **Étapes :**
  1. **Saisie de la question :**
     + **La méthode utilise un scanner (Scanner) pour capturer la question entrée par l'utilisateur.**
  2. **Validation du nombre d'options :**
     + **Un minimum de deux options est imposé pour que le QCM soit valide.**
     + **Si l'utilisateur entre un nombre d'options inférieur à 2, la saisie est répétée.**
  3. **Saisie des options :**
     + **Les options sont saisies une par une et stockées dans le tableau options.**
  4. **Saisie de l'option correcte :**
     + **L'utilisateur entre le numéro de l'option correcte (1 à nbop).**
     + **La saisie est répétée si l'index donné ne correspond pas à une option valide.**

#### **2. ajouterOption(String option)**

* **But : Ajoute une nouvelle option au tableau options.**
* **Fonctionnement :**
  + **Crée un nouveau tableau avec une taille augmentée.**
  + **Copie toutes les options existantes dans le nouveau tableau.**
  + **Ajoute la nouvelle option à la fin.**
  + **Remplace le tableau existant par le nouveau.**
* **Exemple :**
  + **Avant ajout : {"Paris", "Londres", "Berlin"}**
  + **Ajout de "Madrid" → Après ajout : {"Paris", "Londres", "Berlin", "Madrid"}**

#### **3. toString(boolean showCorrectOption)**

* **But : Retourne une représentation textuelle complète du QCM, incluant :**
  + **La question.**
  + **Les options disponibles.**
  + **La réponse correcte (si le paramètre showCorrectOption est true).**

#### **4. Getters et Setters**

* **But : Permettent un accès contrôlé aux attributs privés de la classe.**
  + **Les getters récupèrent la valeur d'un attribut.**
  + **Les setters modifient la valeur d'un attribut.**
* **Exemples :**
  + **getQuestion() : Renvoie le texte de la question.**
  + **setQuestion(String question) : Met à jour la question.**
  + **getCorrectOption() : Renvoie l'index de l'option correcte.**
  + **setCorrectOption(int correctOption) : Met à jour l'option correcte.**

### **Utilisation typique**

#### **Création d'un QCM**

1. **Appeler la méthode creerqcm().**
2. **Saisir les informations nécessaires : question, options, et bonne réponse.**

#### **Ajout d'une nouvelle option**

1. **Appeler ajouterOption(String option) avec le texte de la nouvelle option.**
2. **L'option est ajoutée à la fin du tableau.**

#### **Affichage du QCM**

1. **Appeler toString(boolean showCorrectOption) :**
   * **Si showCorrectOption = true, la réponse correcte sera affichée.**
   * **Sinon, seule la question et les options seront affichées.**

### **Résumé**

**La classe Qcm est une abstraction simple mais puissante pour gérer des questions à choix multiples dans un quiz. Elle permet de :**

* **Créer des QCM de manière interactive.**
* **Ajouter des options dynamiquement.**
* **Récupérer et afficher les informations sur les QCM de manière flexible.**

**Avec cette classe, il est facile d'organiser et de manipuler les questions dans un quiz, tout en respectant les principes d'encapsulation et de modularité en Java.**