L'exercice 1 reprend l'exercice 3 du TP précédent mais il y a des changement, vous pouvez donc reprendre le code du TP1 (si vous l´avez fait) mais n'oubliez pas de l´adapter.
Les deux exercices de ce TP sont indépendants et de difficulté croissante, n'hésitez pas a changer d'exercice si vous êtes vraiment bloqués.

Exercice 1 Soit le début de classe suivant :

```
class Etudiant {
        String nom;     // le nom de l'étudiant
        String prenom; // le prenom
        int num;        // numéro d'étudiant
        double note;    // la note de l'étudiant (sur 20)
        /* A COMPLÉTER ... */
      }
```

Celle-ci permet de stocker quelques informations importantes d'un étudiant.

1. Ajoutez à cette classe un constructeur `Etudiant(String nom, String prenom, int num, double note)` prenant en paramètre un nom, un prénom, un numéro d'étudiant ainsi qu'une note et les affectant aux attributs concernés (on suppose que l'utilisateur donne toujours une notre entre 0 et 20).
2. Dans la classe `Etudiant` :
   (a) écrivez une fonction `void affiche()` pour afficher le contenu de l'objet courant (`this`) au format "Nom Prénom (Numéro d'étudiant) : Note" ;
   (b) créez un `main` dans une classe séparée pour tester votre classe `Etudiant` en construisant un exemple puis en l'affichant *(vous testerez systématiquement chaque nouvelle fonction)* ;
   (c) écrivez, dans `Etudiant`, une fonction `char[] initiales()` qui renvoie un tableau contenant exactement deux caractères (de type `char`) : le premier correspond à l'initiale du prénom et le deuxième à l'initiale du nom ;[1] vous pourrez utiliser la méthode `charAt` de la classe `String`, qui prend en argument la position de la lettre qu'on souhaite obtenir dans la chaîne ;
   (d) ajoutez une méthode `boolean passage()` qui renvoie le booléen disant si l'étudiant peut passer (c'est à dire si sa note est supérieure ou égale à 10) ;
3. Il s'agit maintenant de calculer la moyenne de tous les élèves entrés. Pour cela on commence par ajouter les attributs suivants :

```
static int nombreDEleves = 0;     // le nombre d'élèves déjà entrés
static double sommeDesNotes = 0; // la somme totale des notes entrées
```

   (a) Que signifie le mot-clé `static` pour un attribut ? pour une méthode ?
   (b) Modifiez tous le constructeur de sorte que `nombreDEleves` augmente de 1 à chaque création d'élève et que `sommeDesNotes` augmente d'autant que la note (`sommeDesNotes = sommeDesNotes + this.note`).
   (c) Écrivez la méthode `static int moyenne()` qui renvoie la moyenne de tous les élèves entrés jusqu'à présent.
4. Un collègue vous fait remarquer que l'on peut remplacer les déclarations des attributs `nom`, `prenom`, `num` et `note` par :

```
        final String nom;     // le nom de l'étudiant
        final String prenom; // le prenom
        final int num;        // numéro d'étudiant
        final double note;    // la note de l'étudiant (sur 20)
```

1. Un `return` ne peut renvoyer qu'un seul objet. Il existe des façons élégantes en Java pour contourner cette difficulté lorsqu'on veut qu'une méthode retourne plusieurs objets, mais nous nous limitons ici à l'utilisation d'un petit tableau.

(a) Que signifie le mot-clé `final` ?

(b) Trouvez deux intérêts à faire ce changement.

5. On ajoute maintenant dans la classe `Etudiant` les attributs suivants :

```
static final int COEF1 = 1;  // le coefficient du 1er devoir
static final int COEF2 = 1;  // le coefficient du 2nd devoir
static final int COEFEX = 2; // le coefficient de l'examen final
```

(a) Que signifie le mot-clé `static final` ?

(b) Doit on modifier les constructeurs déjà créés ?

(c) Ajoutez un constructeur
`Etudiant(String nom, String prenom, int num, double devoir1, double devoir2, double exam)`
qui calcule automatiquement la note final de l'élève en fonction de ses notes `devoir1`, `devoir2`, et `exam`.

Exercice 2 La classe suivante simule un compte en banque, le code étant fourni par la banque centrale :

```
public class CompteBanque {

    static int minimum=0; // représente le découvert autorisé
    int montant;          // argent sur le compte

    CompteBanque(int montant){
        this.montant = montant;
    }

}
```

1. Définissez une méthode `boolean retirer(int n)` prenant en argument un entier `n` permettant de retirer `n` du compte en banque, sauf si la valeur finale est en dessous du minimum. Elle renvoie `true` si le virement a été effectuée et `false` sinon.
2. Pourquoi la variable `minimum` est-elle mise en `static` ? Définir une méthode (elle aussi `static`) `setMinimum` permettant de changer celui-ci. Que se passe-t-il si la méthode `setMinimum` n'est pas en `static` ?
3. Écrire une classe `Operation` qui représentera la classe accessible aux utilisateurs, cette classe est moins sécurisée.
   (a) Écrivez dans la classe `Operation` un `main` qui crée un compte en banque `compte` et modifie directement `compte.montant` pour lui mettre une valeur négative.
   (b) Pourquoi ce comportement n'est pas souhaité ?
   (c) Pour éviter ce problème, on choisi d'utiliser le mot-clé `private` et de rendre privé l'attribut `montant`, c'est à dire que l'on remplace la ligne 4 par `private int montant`. Qu'est ce que cela signifie ?
   (d) On remarque alors que l'utilisateur (qui n'a accès que à `Operation`) ne peut plus voir le montant d'un compte ni mettre de l'argent dessus. C'est pour cela que l'on utilise alors un "geteur" `int getMontant()` qui renvoie la valeur de `montant` et une méthode `void déposer(int montant)` qui permet de faire un dépôt.
   (e) A partir de maintenant tout les attributs doivent être précisés ou bien `private` ou bien `public`, y compris dans le code déjà écrit. [2]
4. On se demande alors si l'application est bien sécurisée.
   (a) Réécrivez le `main` de `Operation` qui doit maintenant créer un compte en banque `compte` et déposer de l'argent dessus.

2. Ne pas préciser si un attribut est public ou privé ne veut pas dire qu'il est public, mais a un autre comportement qui est au delà du programme, c'est pour cela que l'on précise.

(b) On peut alors se demander d'où vient cet argent. Le plus souvant celuis-ci vient dún virment, on crée donc (dans `CompteBanque`) une méthode `boolean transfert(int n, CompteBanque c)` qui transfert de l'argent du compte `c` vers le compte courant (c'est à dire `this`).

(c) Les méthodes `retirer` et `déposer` ne sont plus ni utiles ni souhaitées pour `Operation`, elles peuvent donc devenir privées.

(d) A partir de maintenant toute les méthodes doivent être précisées ou bien `private` ou bien `public`.

5. L'application est-elle alors sécurisée ?

(a) Réécrivez le `main` de `Operation` qui doit maintenant créer un compte en banque `compte` avec une somme astronomique dessus.

(b) Comme au dessus, on ne sait pas d'où vient l'argent. On va donc ajouter un constructeur `CompteBanque(int n, CompteBanque c)` qui débite le compte `c` pour créer notre compte.

(c) Le constructeur `CompteBanque(int n)` n'est plus ni utile ni souhaité pour `Operation`, il peut donc devenir privé.

(d) A partir de maintenant tout les constructeurs doivent être précisés ou bien `private` ou bien `public`.

6. L'application est maintenant sécurisée ! Et pour cause, on ne peut rien faire avec...

(a) Réécrivez le `main` de `Operation` qui doit juste créer un compte. [3]

(b) Il nous faut un compte référent, qui correspondrait en gros a l'argent liquide apporté. Pour ça, on crée un nouvel attribut dans `CompteBanque`, c'est l'attribut `public static final CompteBanque CAISSE = new CompteBanque(100000000)`.

(c) Expliquez dans un commentaire ce que veut dire cette ligne.

7. Et maintenant, est-on sécurisé ? Presque !

(a) Réécrivez le `main` de `Operation` qui doit maintenant créer deux comptes puis faire un transfert. Rien ne dit alors que l'opérateur a effectivement des droits sur ces comptes...

(b) Ajoutez, dans `CompteBanque`, un attribut `private final int Code`.

(c) Modifiez `boolean transfert(int n)` pour qu'il prenne un nouvel argument `int code` et modifiez `CAISSE` pour qu'y soit inscrit un numéro (celui que vous voulez) qui correspondra aux code des banquiers accrédités à donner de l'espèce.

(d) Modifiez `boolean transfert(int n, CompteBanque c)` pour qu'il prenne deux nouveaux arguments, `int codeThis` et `int codeC`, affecte `codeThis` comme code pour le nouveau compte et ne met rien dessus à moins que `codeC` soit bien le code de `c`.

(e) Modifiez `CompteBanque(int n, CompteBanque c)` pour qu'il prenne un nouvel argument, `int codeC` et ne débite le compte `c` que si `int codeC` est le bon code.

8. Prouvez que cette application est bien sécurisée. [4]

3. Indice : ce n'est pas possible !
4. Je blague...