# CENTRE SCOLAIRE SAINTE-JULIENNE

## TA 8 – POO Composition

## Exercices Java – Série 4 – Enoncés

### I- Mise en situation

Tu es analyste-programmeur dans une société et tu dois passer un test en langage Java. A travers une série d'exercices, tu dois comprendre et maitriser le langage Java pour obtenir la prime salariale.

### II- Objets d'apprentissage

| Appliquer | Transférer |
|---|---|
| • Modéliser une logique de programmation orientée objet<br>• Déclarer une classe<br>• Instancier une classe (objet)<br>• Utiliser les méthodes de l'objet instancié<br>• Traduire un algorithme dans un langage de programmation<br>• Commenter des lignes de codes.<br>• Tester le programme conçu | • Développer une classe sur la base d'un cahier des charges en respectant le paradigme de la programmation orientée objet (POO)<br>• Programmer en recourant aux classes nécessaires au développement d'une application orientée objet<br>• Corriger un programme défaillant<br>• Améliorer un programme pour répondre à un besoin défini |
| **Connaître** | |
| • Différencier la programmation impérative de la programmation orientée objet<br>• Caractériser une classe<br>• Décrire la création d'un objet (instanciation)<br>• Identifier l'instance d'une classe<br>• Caractériser les attributs dans une classe (encapsulation)<br>• Caractériser les méthodes dans une classe (encapsulation)<br>• Décrire la création d'un constructeur<br>• Différencier les types de visibilités | |

### III- Travail à accomplir

1. Analyser l'énoncé du point IV correspondant au numéro de l'exercice demandé.
2. Modéliser en diagramme de classes l'exercice.
3. Réaliser l'exercice.
4. Commenter le travail.
5. Visualiser le travail.
6. Sauvegarder le document suivant les instructions données.
7. Imprimer le(s) document(s)

## IV- Enoncés

1. Ex01

   Créer les classes **Adresse**, **Institut**, **Section**, **Etudiant** et **TestInstitut** qui seront utilisées comme expliqué ci-dessous.

   La classe Adresse contient une variable d'instance: une chaine de caractères contenant la localité (elle pourrait contenir d'autres variables d'instance).

   La classe Institut contient les variables d'instance suivantes: une chaîne de caractères contenant le libellé de l'institut, une Adresse ainsi qu'une Section. La classe Section contient les variables d'instance suivantes: deux chaines de caractères contenant le nom de la section et celui du coordinateur ainsi que 5 Etudiants (tableau).

   La classe Etudiant contient les variables d'instance suivantes: une chaine de caractères contenant le nom de l'étudiant et ainsi que son Adresse.

   On placera dans chaque classe les ***méthodes habituelles*** (accesseurs et toString).

   Dans la classe TestInstitut, créer toutes les informations de l'Institut (les données seront placées en dur dans l'appel au constructeur), puis les afficher.

   Tester votre application étape par étape.

2. Ex02

   Modifier l'exercice 1 comme expliqué ci-après. Les classes **Adresse** et **Etudiant** sont inchangées.

   La classe **Section** contient une table de 5 entrées (nombre maximal d'étudiants acceptés dans une section), le nom de la section, le nom du coordinateur et le nombre d'étudiants se trouvant effectivement dans la section. Prévoir maximum 3 sections dans la classe **Institut**. A la création de la section, toutes les informations y sont placées hormis les étudiants. Ceux-ci seront placés dans les sections via une méthode de la classe Section qui place un étudiant dans une section.

   Dans la classe **TestInstitut**, définir des tables de données; construire l'institut puis placer les étudiants; enfin, afficher le contenu de l'institut.

3. EX03

   Définir une classe **Carte** contenant deux variables d'instance de type entier: valeur et couleur. En plus des méthodes habituelles, définir une méthode ***calculerRang*** qui définit le rang de la carte: 2 de pique = rang 1, 2 de trèfle = rang 2, …, as de cœur = 52. La méthode toSring retourne le libellé de la carte (ex.: 3 de pique, roi de cœur, as de carreau).

   La classe **JeuDeCartes** contient les deux variables d'instance suivantes: un tableau de référents vers les 52 cartes du jeu et un entier qui renseignera à tout moment le nombre de cartes encore présentes dans le jeu. Elle contient la méthode **remplir** qui initialise le tableau de cartes et la méthode **mélanger** qui mélange les cartes (série(s) de permutations de manière aléatoire ou récursive par exemple). Ces deux méthodes sont appelées ***dès la construction du jeu de cartes***. Prévoir aussi deux autres méthodes: **tirer** et **jouer**. La méthode tirer retourne le référent de la dernière carte du jeu. La méthode jouer simule le jeu de la bataille entre l'ordinateur et l'utilisateur. On lui passe **en paramètre** le nombre de fois que chaque joueur tirera une carte. A chaque fois, les libellés de cartes seront affichés, les rangs des cartes seront comparés afin d'afficher qui gagne. En fin de partie, on affichera le

nombre de fois que chaque partie a remporté un tirage. La classe **TestJeuDeCartes** construit un jeu de cartes et appelle la méthode jouer.

4. Ex04

Idem que l'exercice 3 sauf qu'il faut **laisser la classe de test renseigner le nombre de cartes** dans le jeu lors de sa construction (52 ou 32).
Il est demandé de **gérer le cas de l'égalité** pour le tirage et le résultat final. L'égalité du tirage remet en jeu le point au tour suivant (il faut modifier le rang des cartes). L'égalité du résultat final proclamera le joueur vainqueur.
Utiliser des énumérations pour la couleur (CŒUR, PIQUE, TREFLE et CARREAU) et la valeur: AS, DEUX, TROIS, QUATRE, CINQ, SIX, SEPT, HUIT, NEUF, DIX, VALET, DAME, ROI.

Pour manipuler les énumérations vous pouvez avoir besoin des méthodes suivantes:

Ex: `enum Animal { KANGOUROU, TIGRE, CHIEN, SERPENT, CHAT };`

Obtenir l'index de la valeur selon l'ordre de déclaration (premier = 0).
```
Animal.CHIEN.ordinal() /* -> 2 */
```

Obtenir le nom de la valeur.
```
Animal.CHAT.name() /* -> "CHAT" */
```

(méthode statique) Obtenir la valeur dont le nom est spécifié en paramètre.
```
Animal.valueOf("TIGRE") /* -> Animal.TIGRE */
```

(méthode statique) Obtenir un tableau contenant toutes les valeurs déclarées.
```
Animal.values()[3] /* -> Animal.SERPENT */
```