Exercices Java

Par. N. Chenfour

Exercice 1 :

1. Ecrire une classe Calcul qui offre les méthodes statiques suivantes
   - factoriel
   - puissance
   - racine carrée
   - partie entière
2. Tester la classe
3. Ecrire une classe Produit (ref, desig, prix, q) : la référence est un entier qui prend successivement les valeurs : 0, 1, 2, …
4. Etendre la classe Produit de manière à obtenir une classe non « instanciable ». La nouvelle classe offre une méthode statique permettant de créer et retourner des instances de manière à ne pas dépasser un maximum de produits bien déterminé (exp : 10).

Exercice 2 :

1. Réaliser une classe **Entier** permettant la gestion du type int.
2. On demande d'étendre la classe Entier pour offrir les Méthodes suivantes :
   - Une Méthode *toBinaire* qui retourne sous forme d'une chaîne de caractères le format binaire de l'entier.
   - Une méthode *format* qui permet de formater l'entier afin d'être affichable sur un nombre de caractères bien déterminés (précisé en paramètre de la méthode).
   - Une méthode *toString* qui permet de retourner le format affichable de l'entier.

Exercice 3 :

1. On demande la réalisation d'une Liste chaînées d'objets implémentée récursivement (en ne supposant pas l'existence des classes LinkedList, Vector, etc.). On réalisera alors une classe « List » caractérisée par les propriétés suivantes :
   - value : Object
   - next : un pointeur sur la même classe

   La classe « List » offre aussi les méthodes de services suivantes :
   + boolean isEmpty()
   + void add(Object element)
   + int size()
   + Object get(int index)
   + Object remove(int index)
   + void clear()
   + void set(int index, Object element)
   + String toString()

2. On veut réaliser dans cet exercice une autre liste d'objets basée sur les tableaux. Celle-ci que nous appelons « Array » offre les mêmes services que la classe précédente.

Exercice 4 :

1. On demande d'implémenter une classe **Ouvrage** (cote, titre, liste des auteurs, année et maison d'édition) permettant d'offrir les possibilités suivantes :
    - Création d'un ouvrage.
    - Choisir un format affichable de l'ouvrage
    - Des accesseurs (méthodes get et set) permettant de gérer les différentes informations de l'ouvrage.
    - Une fonction booléenne permettant de déterminer si l'ouvrage est écrit par un auteur donné en paramètre.
    - Une fonction booléenne permettant de déterminer si l'ouvrage traite un thème bien déterminé donné sous forme d'une chaîne à rechercher dans le titre de l'ouvrage (utiliser la méthode *indexOf* de la classe *String*)
2. Ecrire une classe **Biblio** qui offre la possibilité de gérer une liste d'ouvrages. Elle offre alors les possibilités suivantes :
    - Ajouter et supprimer un ouvrage.
    - Extraire l'ouvrage N° i.
    - Rechercher un ouvrage de cote donnée.
    - Afficher la liste des ouvrages écrits par un auteur bien déterminé
    - Afficher la liste des ouvrages traitant un thème bien déterminé

Exercice 5 :

1- On demande la réalisation d'une classe Array qui gère comme propriété un tableau d'objets tout en offrant la possibilité de redimensionnement du tableau après chaque appel à une méthode *add* permettant d'ajouter un nouvel objet au tableau. Donner deux versions de la méthode add : la première avec 1 seul paramètre (l'objet), l'autre version avec 2 paramètres (l'objet et la position d'insertion). Définir aussi une méthode *remove* pour supprimer un objet désigné par son indice dans le tableau ou par sa référence, et une méthode *get* qui retourne l'objet désigné par son indice.

2- Etendre la classe Array pour gérer un tableau de produits. On ajoute une méthode add qui accepte les différentes informations produit, une méthode toString permettant l'affichage du tableau de produits ainsi qu'une propriété sorted de type boolean qui indique si le tableau sera trié ou non. Cette propriété est communiquée à la classe par l'intermédiaire de l'un de ces constructeurs ou par l'intermédiaire d'une méthode *setSorted*. Le tri est réalisé relativement à la désignation.

3- Ecrire une classe ListeProduits qui assure les mêmes fonctionnalités que celles de la classe Array mais en gérant une liste chaînée de produits et non un tableau. On utilisera la classe **LinkedList** du poackage **java.util** et ses méthodes **add**, **remove** et **get**.

4- Redéfinir la classe ListeProduits en implémentant explicitement les fonctionnalités de liste chaînée (récursivement) sans utilisation de la classe LinkedList.

Exercice 6 :

Définir une classe Primitive qui permet de recevoir par l'intermédiaire de ces 8 constructeurs les différents types de base (types primitifs). La classe permet l'accès et la mise à jour de la valeur introduite par le constructeur. On définira les méthodes *get* et *set* avec la fonctionnalité adéquate.

Exercice 7 :

Réaliser une classe Numeric permettant les opérations suivantes :

1- La classe permet de créer des objets enveloppant le type double.

2- Elle offre la possibilité de formater un réel en précisant le nombre de chiffres après la virgule et/ou en précisant la taille de l'information. Cette opération doit être réalisée avec différentes possibilités :

- `public void setPrecision(int precision)`
- `public void setFullyPrecision(int precision)` , qui complète avec des zéro à droite si nécessaire.
- `public void setFormat(int size, int precision)`
- `public void setFullyFormat(int size, int precision)` , qui complète avec des zéro à gauche et à droite si nécessaire.
- `public static String setPrecision(double val, int precision)`
- `public static String setPrecision(String val, int precision)`
- `public static String setFullyPrecision(float val, int precision)`
- `public static String setFullyPrecision(String val, int precision)` qui complète avec des zéro si nécessaire.
- même chose pour setFormat et setFullyFormat s'appliquant à un paramètre double ou string

Les méthodes non numériques agissent sur la valeur de l'objet Numeric, tandis que les méthodes statiques agissent sur leur paramètre tout en retournant l'informations sous le format souhaité.

On utilisera la classe **StringBuffer** et ses méthodes : **indexOf**, **setLength** et **append**.

3- Une méthode statique getValueOf qui retourne la valeur double d'une paramètre String. La classe reconnaît les deux représentations du point décimal : « . » et « , ». En plus la méthode gère les erreurs en s'arrêtant devant le premier caractère non numérique.

On utilisera la méthode **replaceAll** et **charAt** de la classe **String**, ainsi que la méthode statique **parseDouble** de la classe **Double**.

4- Définir une méthode toString dans la classe Numeric permettant de retourner le format affichable de la valeur numérique en fonction du format et de la précision désirée.

Exercice 8 :

On voudrait réaliser un programme de gestion d'un stock de matériel informatique et de logiciel.

1- Créer une interface Produit avec comme méthodes :

```
String getDesignation()
float getPrixUnitaire()
float getQuantité()
```

`char getNature()` qui devra retourner la nature Matériel ('M') ou Logiciel ('L') du produit.

2- Réaliser une classe Matériel et une classe Logiciel qui implémentent l'interface Produit. La classe Matériel dispose d'un constructeur de 3 paramètres pour communiquer les attributs Désignation, PU et Q. La classe Logiciel dispose d'un constructeur à 5 paramètres : Désignation, PU, Q, Editeur et Année d'Edition. La classe logiciel dispose aussi de deux méthode de plus : getEditeur() et getAE().

3- Réaliser une classe GestionDeStock qui dispose des membres suivants :
- Une LinkedList permettant de stocker la liste de Matériels et Logiciels du stock.
- Une méthode Ajouter qui reçoit en paramètre un Produit.
- Une méthode Lister qui affiche la liste des informations adéquates de chaque produit du stock.

4- Imaginer et implémenter d'autres opérations sur le stock dans la classe GestionDeStock.

Exercice 9 :

On voudrait faire une modélisation objet de la structure d'une base de données. Pour cela, on considère que la base de données est une collection de tables. Chaque table et caractérisée par 2 parties : la partie structure de données et la partie données.
La structure de données est définie par l'intermédiaire d'une liste de champs, où un champ est caractérisé par : le nom et le type. On acceptera 3 types : numérique, chaîne de caractère et le type boolean. le type Chaînes de caractères. Tous les types sont définis par leur nom, mais les chaînes sont en plus définies par la longueur.
La partie données est définie comme une collection de tuples, où le tuple est définie comme un liste de valeurs.

1. Proposer et implémenter une structure de classes qui répond à la spécification aux spécifications précédemment définies.
2. Proposer et réaliser une interface graphique permettant la création d'une base de données.

Exercice 10 :

On voudrait faire une modélisation objet de la structure d'un script SQL. Pour cela, on considère que le script est une collection de requêtes SQL. Nous sommes intéressé uniquement par 4 requêtes : Select, Insert, Create Table et Drop Table.
La requête Select est définie par 3 parties : liste des champs (qui prendra la valeur null pour représenter *), nom de la table (on suppose une seule ; pas de jointure) et critère de recherche. Le critère de recherche est une liste de conditions. On va considérer qu'une condition et constituée de 3 parties : l'opérande N° 1, l'opérande N° 2 et l'opérateur. L'opérateur est une chaîne de caractères et les opérandes sont : des noms de champs, des numériques, des constantes chaînes (entre côtes) ou des constantes booléennes (true ou false).
La requête Insert est définie par le nom de la table et la liste des valeurs à insérer dans la table. 3 types de valeurs sont acceptés : les chaînes, les numériques les booléennes (true ou false).
Pour les requêtes Create et Drop, elles sont juste définies par le nom de la table à créer ou à supprimer.

3. Proposer et implémenter une structure de classes qui répond aux spécifications précédemment définies.
4. Proposer et réaliser une interface graphique permettant la création d'un script de requêtes SQL.

Exercice 11 :

On voudrait réaliser une application de gestion d'une bibliothèque de documents avec les différents services classiques que sont : l'ajout d'un nouveau document, la mise à jour, la recherche et l'emprunt. Il y a 3 types de documents dans la bibliothèque : des livres, des rapports et des revues. Les documents sont définis par : leur code, la discipline (informatique, mathématique, etc…), thème (java, compilation, etc…), le type (livre, rapport ou revue), date d'édition, langue, nombre d'exemplaires, nombre d'exemplaire disponibles. Les livres et les rapports sont en plus caractérisés par une liste d'auteurs (nom, prénom, année de naissance, pays). Les livres par la maison d'édition et un code ISBN. Les revues par le mode de parution (quotidien, hebdomadaire, mensuel, trimestriel, semestriel ou annuel). En plus un document peut se présenter en 1 ou plusieurs exemplaires. Où un exemplaire et caractérisé par : Un numéro d'inventaire (un auto incrément), nature (origine ou photocopie), le prix et son état de disponibilité.
En plus, la bibliothèque dispose d'un ensemble d'adhérents ayant le droit d'emprunter les documents disponibles. Dans ce cas, on devrait aussi gérer une liste d'emprunts. Où un emprunt est caractérisé par le code du document et celui de l'adhérent + la date d'emprunt. Un adhérent est caractérisé par son code, nom, prénom, date de naissance, lieu, téléphone, adresse, adresse email et privilège. Sachant qu'il existe 3 types de privilèges : Etudiant, Prof et administrateur. Le privilège est caractérisé par le nom, nombre de documents possibles, et nombre maximum de jours pour un emprunt.

5. Proposer et implémenter une structure de classes qui répond aux spécifications précédemment définies.
6. Proposer et réaliser une interface graphique permettant l'ajout d'un document à la bibliothèque.

Exercice 12 :

On voudrait automatiser la gestion du système modulaire dans un établissement d'enseignement supérieur.

La formation de base est constituée de 6 semestres (3 années d'études). Celle-ci est suivie par un Master de 4 semestres. Soit au total 10 semestres (S1, S2, … S10) d'enseignement universitaire (5 années). Dans chaque semestre, les cours sont organisés en modules (4 modules par semestre). Un module étant caractérisé par un code, un nom et une liste de pré-requis : une liste de modules dont dépend l'inscription dans ce module.

Exemple : soit un module **Mi** ayant comme pré requis **Mj** et **Mk** de l'année précédente. Si l'étudiant n'a pas validé Mj ou Mk, il ne pourra pas s'inscrire dans Mi.

En plus, le module est constitué d'un nombre quelconque de ce qu'on appelle des éléments de module. Un élément de module est caractérisé par son code, son nom et son coefficient.

Un étudiant est caractérisé par un certain nombre d'informations personnelles dont on retient : CNE, CIN, Nom, Prénom, Date et ville de naissance, Adresse, email.

L'étudiant est en plus connu par ces informations universitaire : nous utiliseront le niveau (1..5) et les différentes notes par semestre (nous n'allons pas gérer les redoublements).

Donner l'ensemble des classes Java possibles avec proposition des services utiles par classe.

Exercice 13 :

Il s'agit de réaliser une application de gestion d'Hôtel. On considère alors que l'hôtel est caractérisé par une liste de chambres. Une chambre est caractérisée par les propriétés suivantes :

- numéro
- type (simple ou double)
- état (libre, occupée)
- nombre de lits
- prix

L'application gère aussi les clients qui sont caractérisés chacun par les propriétés suivantes :

- id
- nom
- ville
- pays

Lorsqu'un client réserve une chambre, on devrait enregistrer sa réservation dans une liste de réservation (une classe), sachant que chaque réservation est définie par :

- Le Client
- La chambre réservée
- La période : date d'arrivée (définir une classe Date) et nombre de jours.

Lorsqu'un client occupe une chambre, on devrait enregistrer l'information dans une liste d'occupations (une classe). Chaque occupation est caractérisée par le client et la chambre.

En fin, la classe « Hotel » devrait fournir les opérations suivantes :

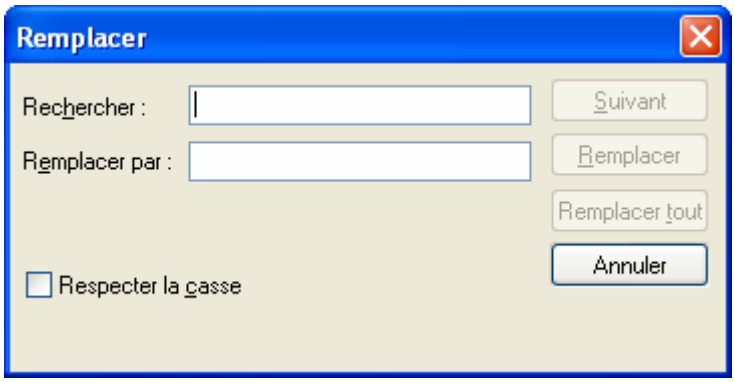
1. Ajouter une nouvelle Chambre
2. Ajouter un nouveau client
3. Chercher une chambre par son numéro
4. Réserver une chambre pour un client
5. Attribuer une chambre à un client
6. déterminer quel Client occupe une chambre donnée par son numéro

On demande la réalisation des classes Java permettant l'implémentation de la description précédente.

Exercice 14 :

L'interface suivante contient les objets Swing des classes suivantes :

```
public JTextField(int columns)
public JButton(String text)
public JCheckBox(String text)
```



1. Redessiner l'interface avec précision des différents panneaux et layout managers.
2. Réaliser l'ensemble des classes utiles pour obtenir l'interface précédente, sachant que :
   - Le Panneau conteneur reçoit comme paramètre de son unique constructeur une chaîne de caractère (de type StringBuffer).
   - La classe contenant le panneau conteneur est une boite de dialogue dont l'un des constructeur est le suivant :

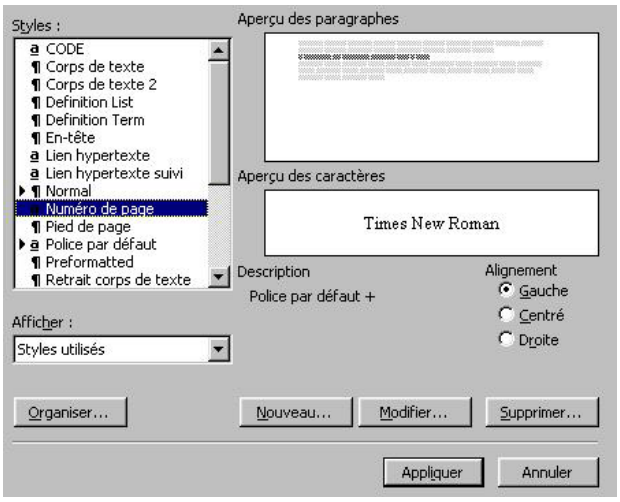
   ```
   public JDialog(Frame owner, String title, boolean modal)
   ```

3. Programmer le bouton « Remplacer » pour remplacer la première occurrence de la chaîne saisie dans le champ « Rechercher : » dans la StringBuffer par la chaîne saisie dans le champ « Remplacer par : ». Sachant que la classe StringBuffer offre les 2 méthodes suivantes :
   - **`int indexOf(String str)`** : retourne la position de la chaîne str dans la StringBuffer (la méthode retourne -1 si str est introuvable)
   - **`replace(int start, int end, String str)`** : remplace la sous chaîne situé entre "start" et "end-1" par la chaîne str.

Exercice 15 :

On demande de faire la conception et l'implémentation de l'interface suivante. Faites une description détaillée des Panneaux et Layouts Manager utilisés et définir des classes réutilisables pour réduire la complexité des des panneaux à manipuler. On créera par exemple les classes suivantes :

1- ButtonPanel
2- LabeledTextField
3- LabeledListBox
4- LabeledComboBox
5- RadioPanel
6- LabeledComponent



Exercice 16 :

On demande la réalisation d'une classe Calculatrice qui étend la classe JPanel.

Exercice 17 :

Réaliser une classe TextBox qui étend la classe JTextField tout en offrant les possibilités de copier/couper/coller.

Exercice 18 :

Réaliser une classe ButtonPanel à plusieurs boutons organisés horizontalement ou verticalement selon le choix de l'utilisateur.

Exercice 19 :

On demande la réalisation d'une classe **LabeledComponent** qui permet de créer un Composant Swing quelconque avec une étiquette à gauche. La classe permettra de fixer la largeur de l'étiquette, la fonte et/ou la taille.

Exercice 20 :

Réaliser une classe **Header** qui permet de créer un header (pour un panneau) constitué d'un ensemble de lignes dont le texte est centré par rapport au panneau conteneur. Le texte est communiqué à l'objet header par l'intermédiaire de son constructeur avec en paramètre un tableau de chaînes de caractères. La classe offre une méthode add permettant d'ajouter une nouvelle ligne à l'entête ainsi qu'une méthode remove pour supprimer une ligne de l'entête. La classe offre aussi la possibilité de choisir la couleur et la taille de l'écriture.

Exercice 21 :

On veut réaliser une classe **ChoicePanel**, permettant d'offrir des choix à l'utilisateur de l'interface. Les choix sont fournis à l'aide d'un tableau de chaînes de caractères au constructeur de la classe. La classe offre en plus d'autres constructeurs et méthodes permettant de préciser le type de choix (exclusif ou multiple), ainsi que le type de composants à utiliser : JRadioButton, JCheckBox ou JToggleButton.

Exercice 22 :

On demande La réalisation d'une classe SimpleMenu permettant de créer et gérer les menus déroulant d'une manière simple et adéquate. La classe SimpleMenu doit étendre la classe JMenuBar et rendre transparent la gestion des JMenu ainsi que les JMenuItem. Elle doit offrir un constructeur permettant de communiquer une matrice de chaînes qui définissent toutes les options (verticales ou horizontales) qui constituent le menu.

Exercice 23 :

De la même manière qu'à l'exercice précédent, réaliser une classe SimplePopup pour la gestion des menus flottants (JPopupMenu).

Exercice 24 :

Réaliser une classe StatusBar à plusieurs panneaux. La classe permettra d'ajouter un panneau à la demande, de retirer un panneau ou de créer une barre d'état avec un nombre de panneau bien déterminé. On pourra aussi choisir d'utiliser ou non une bordure pour les panneaux.

Exercice 25 :

Réaliser une classe FrameBorder qui permet l'ajout et la gestion d'un Menu, d'une Toolbar et d'une Statusbar.

Exercice 26 :

Réaliser un panneau Formulaire permettant de créer des formulaires simples à base des zones de texte. Le formulaire permettra de récupérer à la demande la valeur de chaque champ du formulaire ou un tableau des différentes valeurs saisies.

Exercice 27 :

On demande la réalisation d'une classe **CommandButton**. C'est une classe qui va permettre de créer des boutons dont le type de bordure bascule d'un « RaisedBevelBorder » à un « LoweredBevelBorder » en fonction de l'état « Pressed » ou « Released ». En plus le bouton change de couleur quand le curseur de la souris passe sur la surface de celui-ci.

Exercice 28 :

On demande la réalisation d'une classe **BorderButton**, dont la bordure n'apparaît que si le curseur de la souris passe sur la surface du bouton. La classe offrira aussi la possibilité de changement de type de bordure en utilisant une liste de constantes à définir dans la classe.

Exercice 29 :

On voudrais Réaliser une autre classe pour la création des boutons, la classe **ImageButton**. Un bouton sera alors créé à partir de 3 images de même taille qui représentent les 3 états obtenus à l'aide des 3 méthodes différentes :

```
void setIcon(Icon)
void setPressedIcon(Icon)
void setRolloverIcon(Icon)
```

La classe offre un constructeur qui accepte comme paramètres l'étiquette du bouton ainsi que les noms des 3 images. Pour réduire le nombre de paramètre on pourra supposer que pour une étiquette "OK" par exemple, les images auront les noms : ok_i.gif, ok_p.gif et ok_r.gif (en minuscule). L'extension .gif peut aussi être un paramètre pour pouvoir accepter d'autres formats d'image (.jpg).

Exercice 30 :

En utilisant la classe Précédente, on veut réaliser des menus à base d'images. On demande alors de réaliser les classes suivantes :

1- **HorizontalImageMenu**
2- **VerticalImageMenu**
3- **ToolBarImage** (réalisée à base de JToolBar)

Les 3 classes acceptent en paramètre de leur constructeurs un tableau d'étiquettes, et optionnellement le type des images (.gif ou .jpg).

Exercice 31 :

On voudrais simplifier l'usage des deux contrôles JComboBox et JList. Pour cela, on créera une nouvelle classe **SimpleList** qui pemettra le choix du type de la liste (JList ou JComboBox). La classe offrira les méthode de manipulation adéquates : clear, addItem, removeItem, getValue, setValue, getSelected, setSelected, …