TP n°1 : retrouvons le goût du langage C, notions de classe, de méthode, d'objet, d'héritage

1. Une classe « fenêtre »

Ouvrez le fichier « Fenetre.java » avec un éditeur de texte pour le visualiser. Nous allons détailler chaque élément de ce fichier.

```
public class Nom {
...
}
```

permet de définir une nouvelle classe d'objets. Une classe permet de définir autant d'objets que l'on souhaite ayant les caractéristiques et le fonctionnement imposés par la définition de la classe; c'est comme un calque servant à faire plusieurs dessins tous indépendants (les objets de cette classe).

Une classe contient éventuellement des déclarations de variables, ces variables seront créées pour chaque objet de la classe mais elles pourront avoir des valeurs différentes dans chaque objet pris isolément. Nos objets « Fenetre » seront caractérisés ici par une largeur et une hauteur (actuelle et maximum pour chacun) de type entier, un tableau de caractères à deux dimensions servant à décrire le contenu de notre fenêtre.

Une classe contient éventuellement des **méthodes**, ce sont des fonctions liées à l'objet. Ces méthodes seront utilisables pour chaque objet de la classe. L'appel à une méthode particulière de l'objet se fait par envoi d'un **message** à l'objet, message qui provoque l'exécution de la méthode (un peu comme une interruption provoque l'appel de la procédure de traitement d'interruption). Nos objets « Fenetre » seront caractérisés ici par trois méthodes :

- **Fenetre** qui est une méthode de même nom que la classe, donc servant à construire un objet de cette classe. C'est ce que l'on appelle un **constructeur** ;
- **afficher** qui permet de présenter la fenêtre à l'écran ;
- **retailler** qui permet de redimensionner la fenêtre.

Toutes ces variables et méthodes caractérisent l'objet en tant que donnée informatique : on trouve dans la classe les variables de type souhaité pour l'objet, et toutes les fonctions de traitement de l'objet (consultation et modification des variables internes, comportement de l'objet en fonction de leurs valeurs, …). C'est une approche « orientée objet » de la programmation : les objets communiqueront entre eux par message pour faire fonctionner l'ensemble du logiciel, un message est simplement un appel d'une méthode de l'objet.

Expliquons maintenant le détail de chaque méthode :

```
// constructeur de l'objet Fenetre
public Fenetre (char car_remplissage) {
   int i,j;
   hauteurmaxi = 3;
   largeurmaxi = 20;
   texte = new char[hauteurmaxi][largeurmaxi];
   for (i=0; i<hauteurmaxi; i++)
       for (j=0; j<largeurmaxi; j++)
          texte[i][j] = car_remplissage;
   largeur = largeurmaxi;
   hauteur = hauteurmaxi;
}
```

Ce constructeur ***Fenetre*** a pour paramètre le caractère de remplissage de la fenêtre, qui servira de trame de fond à celle-ci. L'opérateur **new** permet de créer un nouvel objet, ici un objet « tableau de caractères à deux dimensions » dont on fixe la taille. Au passage, on appelle le constructeur d'un tel tableau avec **char[...][....]** Ensuite, on remplit le tableau créé avec le caractère de remplissage. Puis, on met à jour la taille actuelle de la fenêtre aux valeurs maximum. Il n'y a pas de type pour la valeur retournée par cette méthode, car c'est un constructeur qui ne renvoie pas de valeur mais crée un objet « Fenetre ».

Méthode ***afficher*** : cette méthode ne renvoie pas de valeur (type void => équivalent à une procédure). L'affichage de texte ou de valeur de variable à l'écran se fait au moyen des méthodes prédéfinies **println** (affichage avec retour à la ligne) ou **print** (affichage simple). Ces deux méthodes appartiennent à la classe System.out qui est la classe pour les « sorties du système d'exploitation ». Pour faire appel à ces méthodes, on indique donc la classe suivie de la méthode voulue : System.out.println , par exemple.

Méthode ***retailler*** : modifie la taille actuelle de la fenêtre. En langage Java on retrouve les opérateurs booléens du langage C :

- C1 && C2 => ET logique : il faut que les deux conditions C1 et C2 soient satisfaites ;
- C1 | | C2 => OU logique : il suffit que l'une des deux conditions C1, C2 soit satisfaite ;
- ! condition => NEGATION : il faut que la condition soit fausse.

Passons à l'exécution....

Pour créer et utiliser un objet de la classe fenêtre, nous allons définir une classe ***Principal*** dans le fichier Principal.java. Nous mettrons dans cette classe notre programme principal.

Créez la classe Principal en mettant une seule méthode: **public static void main (String args[ ])** qui contiendra la déclaration d'un objet f de type Fenetre, l'instanciation de cet objet en effectuant **f = new Fenetre ( );** et divers appels à afficher et retailler qui seront de la forme **f.afficher (...);** et **f.retailler (...);**

La méthode main est la première appelée lors de l'exécution, il n'y en a en principe qu'une seule pour tout l'ensemble du logiciel quelque soit le nombre de classes qu'il comporte. On peut donner à l'exécution des arguments qui seront tous associés au tableau de chaînes de caractères **args**. Cette association ressemble tout à fait à l'association des arguments d'un programme en langage C avec les éléments argc et argv pouvant figurer dans la fonction main. Nous reviendrons sur l'utilisation d'arguments dans un programme java.

- Compilez la classe Fenetre avec la commande **javac Fenetre.java**
- Le compilateur produit le fichier Fenetre.class
- Compilez la classe Principal avec la commande **javac Principal.java**
- Le compilateur produit le fichier Principal.class
- Exécutez votre exemple d'utilisation de fenêtre avec la commande **java Principal**

2. Plusieurs objets « fenêtre » gérés par un objet « gestionnaire »

Ouvrez maintenant le fichier Gestionnaire.java, et examinons la nouvelle classe ***Gestionnaire*** que nous allons utiliser.

```
public class Gestionnaire {
      //variables nécessaires à cet objet
      private Fenetre tab_fen[];
      private int nbre_fenetres;
      private char remplissage;
      ...
```

Nous définissons dans notre classe Gestionnaire un tableau d'objets Fenetre, un entier permettant de compter le nombre de fenêtres crées, et un caractère utilisé comme le caractère de fond de la fenêtre. Ces variables sont ensuite initialisées dans le constructeur de la classe Gestionnaire.

On définit ensuite deux méthodes pour récupérer des informations de l'utilisateur. La méthode **read** attachée à la classe System.in (classe des entrées système) permet de saisir la valeur d'une touche du clavier. Cette méthode est sujette à problèmes : la touche CTRL-C, par exemple, va interrompre la lecture et le programme; le clavier est peut être mal raccordé à la station, etc. On a donc parfois une *exception*, sorte d'interruption logicielle, que l'on peut :

- faire remonter à des classes ou des méthodes ayant demandé l'opération à problèmes;
- ou bien traiter en local dans la méthode, si l'on sait pallier au problème.

read peut provoquer des « java.io.exceptions » (exception de la classe système d'entrées-sorties). Comme ce genre de problème nous dépasse, on passe l'exception à quelqu'un qui saura sans doute mieux la gérer que nous, on utilise **throws** pour faire suivre le problème. L'exception va donc remonter vers ceux qui ont voulu utiliser notre méthode **car_cmde** (qu'ils assument les problèmes que je rencontre et que je ne sais pas traiter...). **read** renvoie une valeur entière (le code de la touche

tapée). Pour retrouver le caractère, on fait une conversion : **(char)** permet cela. Les paramètres valmin et valmax permettent de préciser dans quel intervalle de valeurs on souhaite récupérer le caractère tapé au clavier.

```
//Méthode demandant à l'utilisateur un caractère compris entre 2
valeurs
public char car_cmde(char valmin,char valmax) throws
java.io.IOException {
    char valeur;
    do {
        valeur = (char)System.in.read();
         } while ((valeur < valmin) || (valeur > valmax));
    return (valeur);
}
```

La méthode num_cmde est similaire à la précédente, elle sert à récupérer la valeur d'une touche « chiffre » tapée au clavier. C'est la soustraction avec la valeur du code de '0' qui permet d'avoir cette valeur. Exemple : touche tapée '4', le read envoie la valeur du code de '4', soustrait de la valeur du code de '0' donne la valeur entière 4. Les codes des touches '0' à '9' étant consécutifs.

```
//Methode demandant a l'utilisateur un chiffre >=0 et < a valmaxi
public int num_cmde(int valmaxi) throws java.io.IOException {
     int valeur;
     do {
           valeur = System.in.read() -(int)'0';
     } while ( ( valeur < 0 ) || (valeur > valmaxi));
     return (valeur);
}
```

Méthode affichage : s'il y a des fenêtres créées, on peut faire confiance à celui qui a écrit la méthode d'affiche dans la classe fenêtre. On appelle cette méthode pour chaque objet fenêtre répertorié dans notre tableau de fenêtres.

Méthode de création d'un objet fenêtre supplémentaire : on connaît déjà l'action de **new**. Comme en C, les variables peuvent être incrémentées par l'opérateur ++.

Méthode modif : on utilise les méthodes de lecture au clavier écrites plus haut. On ne sait toujours pas traiter les « java.io.exceptions », on les passe au voisin... (throws), enfin à celui qui a bien voulu utiliser la méthode modif. Le nombre de fenêtres actuellement créées est passé en paramètre (nbremaxi) pour limiter le choix de l'utilisateur sur le numéro de fenêtre à modifier. On fait appel à la méthode retailler définie dans fenêtre pour modifier la taille de la fenêtre choisie.

Méthode activer : on y crée le tableau tab_fen pour recevoir jusqu'à dix fenêtres. On y trouve le menu principal du gestionnaire. Remarque : les structures de contrôle sont les mêmes en langage C et en Java (for, do ... while, switch -case, …)

Exécution : modifiez votre classe « Principal » pour non plus créer un objet fenêtre et appeler ses méthodes, mais créer un objet « Gestionnaire » et appeler seulement la méthode «activer» du gestionnaire. Testez la création de fenêtres et leur redimensionnement.

3. Chouette, on hérite !!! C'est SUPER...

Reprenez la classe Fenetre et la classe Principal de l'exercice 1.

Dans un nouveau fichier Jolie.java, définir la nouvelle classe ***Jolie*** héritée de la classe Fenêtre avec une méthode afficher différente, faisant un encadrement de fenêtre plus joli que précédemment (avec des * autour par exemple). Pour faire cela, définir la classe comme suit :

```
public class Jolie extends Fenetre {
      public Jolie (char car) {
            super(car);
      }
      //Méthode d'affichage d'une fenêtre
      public void afficher () {
      .....
```

Le mot ***extends*** signifie que la classe Jolie hérite de toutes les variables et méthodes de la classe qui suit extends, ici la classe Fenetre. Dans la nouvelle classe Jolie, on peut faire référence à la classe parente Fenetre par le mot ***super.*** Lors d'un héritage, la classe fille peut toujours désigner la classe parente dont elle hérite avec le mot clé super.

On crée un constructeur d'objet de la classe Jolie, comme on l'avait fait pour Fenetre, on donnait comme paramètre le caractère de remplissage de la jolie fenêtre que l'on veut créer. Ce constructeur fait lui-même appel au constructeur Fenetre (puisqu'il en hérite) : super (car) correspond à l'appel du constructeur Fenetre avec pour argument le caractère de remplissage (car super, ici, est la classe Fenetre !).

Il ne vous reste plus qu'à compléter la méthode afficher, similaire à celle de fenêtre, mais où vous choisirez d'afficher des bordures étoilées par exemple. N'oubliez pas de fermer la définition de la classe jolie par une } à la fin.

Compilez Jolie; testez la méthode retailler sur un objet de type Jolie (retailler est valable pour la classe « Jolie » sans avoir à la redéfinir, car cette méthode est héritée de fenêtre : retailler n'apparaît donc pas dans la définition de la classe Jolie, mais on peut l'appeler quand même). C'est tout l'avantage de l'héritage! Compilez et testez le mélange de fenêtres classiques et jolies ainsi que leur redimensionnement.

4. Paramétrer l'héritage de classes

Recopiez l'ensemble des fichiers avec extension java dans un nouveau répertoire. Éditez le fichier Fenetre.java, et ajoutez en tête de la déclaration des variables hauteur et largeur, le mot clé **static**. Mettez la méthode afficher en commentaire si vous l'avez redéfini dans la classe Jolie.

static, devant une variable d'une classe, signifie que tous les objets de cette classe et tous les objets de classes filles (la classe Jolie dans notre cas) vont utiliser la même variable. Avec l'exemple de largeur et hauteur, nous avions jusqu'à présent ces deux variables attachées à chaque objet créé. Ici, elles sont déclarées comme attachées à la classe (statiques à la définition de la fenêtre). Il n'en existe qu'une unique représentation, commune à tous les objets de la classe fenêtre ou de ses classes filles.

Testez à nouveau le programme en créant des fenêtres et des jolies fenêtres, modifiez la taille d'une fenêtre quelconque et interprétez ce qu'il se passe.

Rajoutez le mot clé **static** devant la méthode *afficher* de la classe Fenetre, recompilez l'ensemble, et si vous ne pouvez pas, analysez pourquoi d'après les messages d'erreur du compilateur.