JAVA : Syntaxe de base

1 Quelques principes

Java est un langage interprété pré-compilé. Les fichiers sources (xxx.java) sont transformés en un langage intermédiaire (xxx.class) par un compilateur (commande **javac**) qui peut être interprété soit par un interpréteur java (commande **java**) soit par un navigateur internet.

C'est un langage à objets très proche du point de vue de la syntaxe de C++.

Il possède de riches bibliothèques standard permettant de disposer de classes d'objets d'interface, d'accès à internet etc.

L'interpréteur java gère un noyau multitâche permettant de faire fonctionner en parallèle des processus (threads) pouvant partager des variables communes ou communiquer par fichier ou réseau. Il intègre les notions d'exclusion mutuelle permettant la résolution des problèmes de synchronisation et d'accès concurrents.

Java gère les exceptions (erreurs) et permet de leur associer des actions.

Java ne possède pas d'héritage multiple mais seulement un héritage de deux types : héritage à partir d'une classe normale ou à partir d'une classe virtuelle (appelée **interface**) ne contenant que des méthodes. Il est possible d'utiliser les deux types d'héritages à la fois.

Java distingue la déclaration d'un nom d'objet de celle de l'objet lui même. Ainsi il est possible d'avoir plusieurs noms pour un même objet ou des objets n'ayant pas de nom. Les pointeurs n'existent pas de façon explicite toutefois les noms d'objets en java se comportent comme des pointeurs (en C++ par exemple). De sorte que l'on ne maîtrise que leur allocation (opération **new**) et pas leur desallocation qui est gérée par java.

2 Syntaxe de base

La syntaxe de java est très proche de celle de C++.

2.1 Commentaires

Ils sont introduits par // et se terminent avec la ligne.

On peut aussi utiliser /* en début et */ en fin ce qui permet d'étendre les commentaires sur plusieurs lignes ou de faire des commentaires sur une partie de la ligne seulement.

2.2 Types primitifs

Les variables de type primitif ne sont pas des objets et ne se comportent donc pas comme des objets mais comme des types semblables aux autres langages.

Les types primitifs sont : **char int byte short long float** et **double** comme en C mais on trouve aussi le type **boolean** qui prend les valeurs true et false.

Java offre aussi des classes correspondant à ces **types (Character**, **Integer**, **Byte**, **Short**, **Long**, **Float**, **Double** et **Boolean**) . Elles permettent d'avoir un comportement standard de tous les objets utilisés.

2.3 Tableaux

Il est possible de définir des tableaux d'objets ou de types primitifs. Le dimensionnement du tableau se fait lors de sa création par **new** et non lors de la déclaration de son nom.

Définition d'un nom de tableau : nomDeClasse nomDeTableau**[]**; ou nomDeClasse**[]** nomDeTableau;

Création du tableau : nomDeTableau = **new** nomDeClasse**[**dimension**]**;

La taille d'un tableau est désignée par nomDeTableau.**length**

Remarque : Afin de dimensionner les tableaux il faut utiliser **new** même lorsqu'il s'agit de tableaux dont les éléments sont de type primitif . Par exemple : int t[]; t=new int[25]; // définit un tableau de 25 entiers

2.4 Classes prédéfinies

Java possède une grande quantité de classes regroupées par grands thèmes dans des bibliothèques standard. On y trouve la plupart des choses dont on peut avoir besoin. Par contre il souvent assez difficile de savoir quoi et où chercher.

2.4.1 Les bibliothèques de java

les classes prédéfinies en Java sont standardisées et placées dans des bibliothèques dont les noms sont eux mêmes standard :

- **java.lang** contient les classes de base (chaînes de caractères, mathématiques, tâches, exceptions ...)
- **java.util** contient des classes comme vecteurs, piles, files, tables ...
- **java.io** contient les classes liées aux entrées/sorties texte et fichier
- **java.awt** contient les classes pour les interfaces (fenêtres, boutons, menus, graphique, événements ...)
- **javax.swing** contient les classes pour les interfaces (évolution de awt)
- **java.net** contient les classes relatives à internet (sockets, URL ...)
- **java.applet** contient les classes permettant de créer des applications contenues dans des pages en HTML

Pour pouvoir utiliser les classes de ces bibliothèques il faut y faire référence en début de fichier en utilisant la directive **import** suivie du nom de la classe de la bibliothèque ou de * pour accéder à toutes les classes de cette bibliothèque: par exemple **import java.io.***
Certaines de ces bibliothèques contiennent des sous-bibliothèques qu'il faut explicitement nommer (par exemple **java.awt.events.*** pour les événements)

Remarque : les classes de la bibliothèque de base **java.lang** n'ont pas besoin de la commande **import**.

2.4.2 Les chaînes de caractères

Elles sont définies dans les classes **String** et **StringBuffer** de **java.lang**. La première est utilisée pour les chaînes fixes et la seconde pour les chaînes variables.
On y trouve, en particulier, les méthodes suivantes

- création d'une chaîne initialisée de classe String : new String(″valeur de la chaîne″);
- création d'une chaîne initialisée à partir d'un tableau de caractères : new String(tableau);
- création d'une chaîne initialisée de classe StringBuffer : new StringBuffer(String);
- création d'une chaîne non initialisée de classe StringBuffer : new StringBuffer(taille);
- création d'une chaîne à partir d'une variable primitive : String nom = String.**ValueOf**(variable);
  ou StringBuffer nom = StringBuffer.**ValueOf**(variable);
- comparaison de chaînes : boolean **equals**(String)
- recherche d'un caractère et obtention de son rang : int **indexOf**(char)
- recherche d'un caractère et obtention de son rang en faisant démarrer la recherche à partir d'un rang donné en second paramètre : int **indexOf**(char, int)
- recherche d'une sous chaîne et obtention de son rang : int **indexOf**(String)
- recherche d'une sous chaîne et obtention de son rang en faisant démarrer la recherche à partir d'un rang donné en second paramètre : int **indexOf**(String, int)
- extraction d'un caractère par son rang : char **charAt**(int)
- extraction d'une sous chaîne par rangs de début et de fin : String **subString**(int,int)
- longueur de la chaîne : int **length**()
- concaténation : on peut utiliser l'opérateur + ou la méthode String **concat**(String)

Pour les chaînes variables on a de plus les méthodes **append** et **insert**.
Cette liste est loin d'être exhaustive.

Transformations de chaînes en éléments simple et transformations réciproques :

- Pour transformer une chaîne en type numérique primitif il est possible d'utiliser les méthodes **Integer.parseInt**, **Long.parseLong**, **Byte.parseByte** et **Short.parseShort** qui acceptent en paramètre un objet de classe String et retournent la valeur correspondante.

Ces méthodes peuvent lever une exception de type **NumberFormatException** si la conversion n'est pas possible (pour le traitement des exceptions voir 7.).

- Pour transformer une chaîne de caractères en un tableau de caractères on utilisera la méthode **toCharArray()** de la classe **String** qui retourne un tableau de caractères. L'opération inverse peut être obtenue grâce au constructeur de **String** acceptant un tableau de caractères en paramètre.

2.4.3 Le graphique

Il fait appel à la classe **Graphics** de java.awt. On y trouve les méthodes habituelles de dessin. La couleur du tracé peut être définie par la méthode **setColor(Color)** et on peut connaître sa valeur par **Color getColor()**.

1°) Dessiner : Les dessins se font grâce aux méthodes suivantes :

Ligne : **drawLine**(x1, y1, x2, y2) // coordonnées des 2 extrémités

Rectangle vide : **drawRect**(int,int,int,int) // coordonnées (x et y) du coin supérieur gauche et dimensions (largeur et hauteur)

Rectangle plein : **fillRect**(int,int,int,int) // même fonctionnement

Rectangle à bords arrondis vide : **drawRoundRect**(int,int,int,int,int,int) // coordonnées du coin supérieur gauche (x et y), dimensions (largeur et hauteur), largeur et hauteur de l'arrondi

Rectangle à bords arrondis plein : **fillRoundRect**(int,int,int,int,int,int) // même fonctionnement

Ovale ou cercle vide : **drawOval**(int,int,int,int) // coordonnées du coin supérieur gauche (x et y) et dimensions du rectangle contenant cet ovale (largeur et hauteur)

Ovale ou cercle plein : **fillOval**(int,int,int,int) // même fonctionnement

Arc d'ovale ou de cercle : **drawArc**(int,int,int,int,int,int) // coordonnées du coin supérieur gauche (x et y), dimensions du rectangle contenant cet arc d'ovale (largeur et hauteur) et angles de début et de fin de tracé en degrés

Arc d'ovale ou de cercle plein : **fillArc**(int,int,int,int,int,int) // même fonctionnement

Polygone vide : **drawPolygon**(int[], int[], int) // liste des coordonnées en x, liste des coordonnées en y des points et nombre de points

Polygone plein : **fillPolygon**(int[], int[], int) // même fonctionnement

Effacement d'une zone rectangulaire : **clearRect**(int,int,int,int) // coordonnées du coin supérieur gauche (x et y) et dimensions du rectangle à effacer (largeur et hauteur)

Ecriture de caractères : **drawString**(String, int, int) dessine le texte donné dans le 1er paramètre. Les deux derniers paramètres désignent les coordonnées (x et y) à partir desquelles sera dessiné ce texte (y est utilisé pour placer le bas du texte). La couleur et la fonte utilisées sont celles associées à l'objet **Graphics**.

2°) Les images : La classe **Graphics** permet l'affichage d'images au format gif ou jpeg. pour cela on dispose de la méthode **drawImage**(Image, int, int, ImageObserver) dont les paramètres sont les suivants : image à dessiner, coordonnées du coin supérieur gauche où la placer et composant qui gère l'image (c'est en général celui dont on a utilisé la méthode **getGraphics()**). On peut ajouter deux paramètres après les coordonnées du coin supérieur gauche pour définir les dimensions désirées de l'image (en horizontal et vertical), elle sera alors être déformée pour adopter ces dimensions.

La classe **Image** permet de faire quelques traitements élémentaires sur les images en voici les principaux :

int getHeight(ImageObserver) retourne la hauteur de l'image en pixels.

int getWidth(ImageObserver) retourne la largeur de l'image en pixels.

Image getScaledInstance(int, int, int) retourne une copie redimensionnée de l'image. les deux premiers paramètres précisent la hauteur et la largeur désirées pour la copie. Si l'un d'entre eux vaut –1 il sera adapté pour conserver l'aspect initial de l'image. Le dernier paramètre indique l'algorithme à utiliser pour créer cette copie. Il peut prendre les valeurs :

Image.SCALE_DEFAULT algorithme standard
Image.SCALE_FAST algorithme rapide
Image.SCALE_SMOOTH algorithme qui préserve la qualité de l'image
Image.SCALE_REPLICATE algorithme simple qui se contente de dupliquer ou d'enlever des pixels
Image.SCALE_AVERAGING algorithme qui utilise une méthode basée sur la moyenne entre pixels voisins.

2.4.4 Les couleurs

En java les couleurs sont des objets de la classe **Color**. Cette classe est dotée d'un constructeur permettant de créer une couleur en donnant les proportions de ses composantes rouge, verte et bleue (ces proportions sont données par des nombres compris entre 0 et 255 que l'on peut considérer comme des pourcentages):

Color maCouleur = new **Color**(120 , 255 , 22);

Réciproquement on trouve des méthodes permettant de récupérer les composantes rouge verte et bleue d'une couleur ces méthodes sont **int getRed()** , **int getGreen()** et **int getBlue()**.

Les méthodes **Color darker()** et **Color brighter()** retournent une couleur plus foncée (resp. plus claire) que l'objet dont on a invoqué ces méthodes.

Il existe d'autres méthodes liées aux couleurs qui ne seront pas détaillées ici.

2.4.5 les fontes de caractères

En java les fontes de caractères sont des objets de la classe **Font**. Cette classe est dotée d'un constructeur permettant de créer une fonte en donnant son nom, son style et sa taille. Le style est une somme des constantes **PLAIN**, **BOLD** et **ITALIC** la taille est donnée en points :

Font maFonte = new **Font**(″ Serif″ , **Font.BOLD**+**Font.ITALIC**, 14);