Généralités sur le Langage Java et éléments syntaxiques.

Ce document n'est pas un polycopié, c'est seulement un résumé des principales notions abordées en cours, il vient en complément du cours d'amphi et ne contient pas toutes les notions et les illustrations qui ont été présentées.

Introduction

Un programme doit satisfaire plusieurs critères de qualités comme :
Lisibilité, Modularité, Efficacité, Robustesse, Extensibilité, Sécurité. Cependant certains de ces critères sont antagonistes comme par exemple, extensibilité et sécurité. De fait, la programmation s'apparente plus à un « art » qu'à une véritable science. Il n'existe pas un meilleur programme dans l'absolu, simplement il existe des programmes qui maximisent certains objectifs.

Généralité sur le langage Java.

L'exécution d'un programme Java se fait en deux phases :

Une phase de compilation qui à partir d'un code source (« fichier.java ») produit un fichier un « fichier.class ».

Une phase d'interprétation qui consiste à charger le code compilé représenté par (« fichier.class ») dans la machine virtuelle Java (JVM pour Java Virtuelle Machine) puis à l'interpréter dans la machine virtuelle.

Caractéristiques du langage Java.

- Portabilité
  - Langage indépendant de la machine cible.
- Sûreté
  - interprétation dans la machine virtuelle
  - perte d'efficacité du fait que le langage est interprété
- Véritable langage objet.
  - Un mariage entre smalltalk et C++, « plus pur » en quelque sorte que C++, intégration

de concept présent dans UML comme la notion d'interface.
    - Héritage multiple de code est abandonné, ni un plus ni un moins par contre l'héritage privé n'est pas implémenté ce qui est dommage.
    - Multi threaded
- Présence d'un « Ramasse Miettes » (garbage collector).
- Environnement de programmation JDK
    - Complet de l'image au son en passant par le réseau.
        - AWT
        - Son
        - Mécanisme de réflexion (les objets, les types, les services sont considérés comme des objets de première classe).

Syntaxe de base du Langage

Le langage Java inclut 99% de la syntaxe du langage C (les pointeurs n'apparaissent pas clairement mais à l'exception des types primitifs tout n'est que pointeur.)

La première application Java, l'application « Hello.java »

Quelques remarques.
Les conventions de codage du langage Java peuvent être consultées à l'adresse suivante :
ftp://ftp-developpez.com/cyberzoide/java/JavaStyle.pdf

A retenir les classes doivent avoir leur première lettre en majuscule, à l'exception des constantes qui elles seront toutes écrites en majuscules, toutes les autres entités (variables, variables de classes, variables d'instances, services de classes, services d'instances) commenceront par une minuscule.

Il ne peut y avoir en Java, qu'une seule classe public par « fichier.java », si il y a une classe public dans un « fichier.java » le nom de la classe doit être le nom du fichier. Par exemple,

```
public class Test
{
   static int somme(int i, int j)
         {
           return i+j;
         }
}
```

Il y a dans ce fichier une classe public qui s'appelle Test, elle doit donc être dans le fichier « Test.java ».

Les mot clef « private » et « public ».

Tant que les différents concepts objets ne sont pas présentés, il est difficile de présenter la notion de contexte de classe. Dans un premier nous allons considérer que le contexte de classe correspond au contexte d'un module écrit en C. Nous détaillerons la signification de cette notion après avoir brièvement présentés les deux autres mots clefs qui sont « public » et « private ».

Les mots clefs « private » et « public » sont des mots clefs qui permettent le contrôle de l'accès à

l'information. L'accès à l'information en Java est contrôlé par l'utilisation de 4 mots clefs:

- **Public** toute entité qui contient la classe peut accéder (lecture/écriture) à l'information publique.
- **Protected** seules les classes définies dans le même package ou qui héritent de la classe peuvent avoir accès à cette information.
- **Rien** : dans ce cas seules les classes définies dans le même package peuvent accèder à cette information.
- **Private** seules les entités définis à l'intérieur de la paire d'accolades qui définit la classe peut avoir accès à une information privée.

Pour l'instant nous ne pouvons aborder que les deux mots clefs private et public. Pour cela on peut les rapprocher des notions abordées en C.

Entité de classe et Entité d'instance en Java.

Les mots clefs « public » et « private » servent donc à contrôler l'accès aux entités du programmes que ce soit des variables ou bien des éléments équivalent à des fonctions.

Il existe deux types différents d'entités:

- **Entité de classes**: Elles correspondent aux entités d'un module C. En Java, ces entités sont précédées du mot clef « static » mais il faut distinguer celles qui sont précédés de :

  - « public static », elles ressemblent d'une certaines façon aux entités déclarées présentent dans le « fichier.h ». Elles sont à rapprocher des fonctions globales ou des variables globales déclarées en C.
  - « private static », elles ressemblent d'une certaine façon aux entités déclarées présentent dans le « fichier. C ». Elles sont à rapprocher des fonctions ou variables locales d'un modules

  Les variables ou services précédés pas « private static » peuvent être rapprochées des variables ou des fonctions locales à un module C. Dans ce cas, elles sont précédées du mot clef « static » et elles sont déclarés dans le « fichier.c » du module.

- **Entités d'instances :** Ces entités ne sont pas précédées du mot clefs static, elles correspondent aux champs d'une structure C. On distinguera les champs qui ne sont pas des pointeurs de fonctions et qui représentent les champs structuraux, des champs qui sont des pointeurs de fonctions qui représentent des champs comportementaux. Ces entités sont assimilables aux structures qui sont présentes dans le point C. Nous développerons ces points dans les prochains cours.

Interface et implémentation.

Nous retrouvons en Java les mêmes notions que pour un module C:

- Interface : L'ensemble des informations qui peuvent être à l'extérieur d'un module. En principe, les informations contenues dans l'interface ce résume à un ensemble de services, on parle dans ce cas, d'interface fonctionnelle public. La propriété importante liée à cette notion est celle de compatibilité ascendante de l'interface. Cette propriété demande à ce que les ensembles qui étaient valables à un instant T0 continue à un instant T1 postérieur à T0.

Cette propriété concerne aussi bien les services que les variables.

- Implémentation : L'ensemble des informations (y compris le code) qui sont nécessaires à l'implémentation des services présentés dans l'interface fonctionnelle public. L'implémentation n'est qu'un choix informatique temporaire qui doit évoluer et qui est en partie lié au degrés de maturité du module.

Comme l'implémentation d'une classe est amenée à évoluer et que l'interface fonctionnelle public ne peut croître qu'en respectant la compatibilité ascendante, il est nécessaire de séparer clairement l'interface fonctionnelle de l'implémentation d'un module. Car si une partie de l'implémentation est présente à l'intérieur de l'interface fonctionnelle, cette partie de l'implémentation est définitivement fixée car la propriété de compatibilité ascendante bloquera l'évolution du module.

Exemple « Hello World ».

Le premier exemple de classe Java, le fameux Hello

```
public class Hello
{
        public static void main(String []argv)
         {
            String s = "Hello";
            System.out.printl(s);
         }
}
```

Les différents éléments syntaxiques du programme sont:

1. La classe Hello est une classe publique elle doit donc être définie dans le fichier « Hello.java ».
2. Si on veut exécuter cette application Java il faut exécuter la suite de commande
   - javac Hello.java
   - java Hello

En java, la notion de programme principal est différente de celle du langage C. Il peut y avoir plusieurs occurrences

Types primitifs en Java.

On appellera type primitif tous les types qui ne sont pas liés à une classe et qui repésentent en quelque sorte les briques de bases nécessaires à la construction des classes. Afin de pouvoir participer à la portabilité du langage java il est nécessaire que la taille des types soit fixés dans la norme de Java.

Les types primitifs sont les suivant:
byte(8 bits), short(16 bits), int (32 bits), long (64 bits), float (32 bits), double (64 bits), char (16 bits), boolean (1 bit).

Pour les types primitifs les bits constituants la variable représente un codage de la valeur de ces types primitifs.

Par exemple, la déclaration short i = 3; écrit dans la variable i le codage 0x03

Type non primitifs en Java.

Tous les types qui ne sont des types primitifs seront associés à une classe ou à une interface. Nous dirons alors que les types non primitifs servent à référencer ou à adresser des « objets »

Contrairement aux types primitifs les variables associées à ces types contiendront seulement une adresse qui référencera le véritable objet physique. Si en Java, il n'existe pas de pointeurs visibles, toutes les variables déclarées avec un type non primitifs sont en réalités associées à des pointeurs.

Par exemple, la déclaration :
String s = « Une chaine » s'interprète de la manière suivante,

1. création d'un objet de type String en utilisant la constante littérale « Une chaîne », A une adresse mémoire par exemple 0XAAFF.
2. La variable s est une variable qui référence un objet de type String (plus exactement de type compatible). La valeur de la variable s est celle de l'adresse de l'objet référencé c'est à dire 0XAAFF.

A l'exception des types primitifs, les variables ne représentent que des références à des entités (objets), nous allons voir que dans la section suivante, l'affectation s'exprime comme un changement de référence et non un changement de contenu de l'objet. Il s'agit s'une simple réaffectation de pointeur.

Interprétation des opérateurs =, ==

L'opérateur d'affectation x =y consiste a mettre dans la variable x les bits de la variable y. Il s'agit d'une affectation bit à bit. Suivant le type des variable x et y, il existe deux interprétations :

1. Si x et y sont des types primitifs, la variable x prend alors la même valeur que la valeur y.
2. Si x et y ne sont pas des types primitifs, alors la variable x référence la même entité (objet) que la variable y. Les variables x et y sont des variables qui adresse le même objet. Ceux sont deux noms qui permettent de manipuler une même chose.

L'opérateur de comparaison x==y consiste à comparer les bits de la variable et de la variable y. Il s'agit d'une comparaison bit à bit. . Suivant le type des variable x et y, il existe deux interprétations :

3. Si x et y sont des types primitifs, la comparaison est vrai si la valeur de la variable est égale à la valeur de la variable y.
4. Si x et y ne sont pas des types primitifs, alors la réponse est vrai si la variable x référence la même entité (objet) que la variable y.

Les tableaux en Java.

La déclaration d'un tableau en Java est la suivante:

Type [] nom; déclare une variable nom qui est une référence sur un tableau de type T. En Java, T [] représente véritablement un type. Un exemple de déclaration peut-être.

Int [] tableau: La variable tableau est une référence à un objet de type tableau de int (int []). Dans ce cas, la déclaration en langage C la plus voisine est int *tableau;

String [] argv: La variable argv est une référence à un objet de type tableau d'objets de type String (String []). Dans ce cas, la déclaration en langage C la plus voisine est String *argv;

Il n'y pas la possibilité en Java de déclarer la taille d'un tableau en même temps que la déclaration. L'instruction int t[10] n'est pas correcte en Java.

Pour les tableaux, la syntaxe pour créer un un objet de type tableau est:

- new int[10] crée un tableau qui contient 10 entiers
- new String [10] crée un tableau de 10 référence à des objets de type String, par contre aucun objet de type string n'est crée avec cette instruction.

L'instruction new T[N], équivalente à malloc(sizeof(T)*N), à deux interprétations en fonction de la nature de T :

- Si T est un type primitif alors la place mémoire allouée par cet appel est N fois la taille d'un T. Les N cellules crées sont consécutives.
- Si T n'est pas un type primitif alors la place mémoire alloué par cet appel est N fois la taille d'une variable référence à un objet de type T. Il n'y a pas de création d'objet.

Pour une variable qui référence n'importe quel tableau t de n'importe quel type, on peut connaître la taille d'un tableau en utilisant l'instruction :

```
t.length
```

Un premier exemple qui permet de remplir un tableau d'entier avec les 10 premiers nombre pair

```
public class ExempleVecteurPair
{
	public static void main (String []argv)
	{
		int [] tab;
		tab = new int[10];
		for(int i=0; i< tab.length; i++)
			tab[i] = 2*i;
	}
}
```

Les tableaux bidimensionnels T [][] sont équivalents T ** c'est à dire à des tableaux de tableaux. Comme pour les tableaux mono-dimensionnels là aussi il y a deux interprétations en fonction de la nature réelle du type T.

1. Dans le cas où T est un type primitif, les éléments T[ï] pointent sur un tableau d'entiers
2. Dans le cas où T est un type non primitif, les éléments T[i] pointent sur un tableau de référence à des objets de type T. Dans ce cas, on peut considérer que l'on a une troisième

indirection pour accèder à l'information.

Un exemple de code, pour illustrer une utilisation d'une matrice triangulaire inférieure.

Exemple de Code :

```
public class Vecteur1 {
	public static void main (String []argv)
	{
		int [][] tab;
		tab = new int[3][];

		for(int i=0; i< tab.length; i++)
		{
			tab[i] = new int[i+1];
		}

		for(int i=0; i< tab.length; i++)
			for(int j= 0; j < tab[i].length;j++)
		{
			tab[i] [j]= i+j+1;
		}

		for(int i=0; i< tab.length; i++)
		{
			System.out.println(" ");
			for(int j= 0; j < tab[i].length;j++)
			{
				System.out.print(" " + tab[i] [j]);

			}
		}
	}
}
```

Détail syntaxique sur les tableaux,

On peut directement affecter un tableau en utilisant l'initialisation au moment de la déclaration. Syntaxiquement c'est la même que pour le langage C. L'exemple suivant, illustre diverse utilisation de cette possibilité.

```
public class ExempleTableauInitialisation
{
		public static void main (String []argv)
		{
			// initialisation d'un tableau d'entiers
			int [] tab = {1,2,3,4};
			// initialisation d'un tableau bi dimensionel
			int [][] tab1 = {{1},{2,3},{4,5,6}};
			// initialisation d'une chaine de caractère
			String [] tabString = {"CE MATIN", "il fait beau"};
```

```
                }
}
```