JAVA : PRINCIPES

Notions : Classe, héritage, propriétés, méthodes, portée, débogage

INTRODUCTION

En inventant un nouveau langage portable, objet, libre, l'éditeur Sun a voulu gagner la bataille du Web. Très vite repris dans sa version allégée par *Netscape* sous la forme du *JavaScript* incorporé dans le code *HTML*, le langage a gagné ses lettres de noblesses lorsqu'il est devenu un outil autonome capable de concurrencer les *C++*, *Basic* et autres *Pascal*.

Aujourd'hui, Java c'est :

- Un langage de définition d'***appliquettes*** (***applets***) incorporables aux pages Web, l'équivalent Microsoft s'appelle *ActiveX*.
- Un langage d'applications client-serveur centrées autour du Web avec les ***Servlets*** et ***applets***, équivalents du contrôle *Winsock* de Microsoft.
- Un langage de pages dynamiques avec ***JSP*** (*Java Server Page*) ⇒ équivalent à l'*ASP* Microsoft ou au *PHP*.
- Un langage de développement de programmes objets. Permet la distribution et la communication entre objets sur le réseau avec ***RMI*** (*Remote Method Invocation*)
- Et d'autres choses encore : ***J2EE*** concurrent de *.Net* de Microsoft, ***Java Beans*** contre *ActiveX*...

PRINCIPES DU LANGAGE

Java est un langage objet où tout est objet et dérive de la classe ***Object***.

Seuls quelques types primitifs ont été conservés : *byte, short, int, long, float, double, char*. Leurs équivalents existent sous forme de classes (possédant donc des méthodes associées), dont : *Integer, Double, Char*.

Comme le langage ***C*** dont il hérite, *Java* est sensible à la casse.

Une convention (valable d'ailleurs pour tous les langages objet) est de ne mettre une majuscule que pour les classes, les méthodes et propriétés de classe et le constructeur. Le reste doit commencer par une minuscule.

Les indications de début et de fin sont représentées par des accolades ***{ … }***, chaque ligne se termine par un point-virgule ***;***. Toute méthode en *java* est une fonction et doit retourner une valeur. Toutefois, pour constituer des procédures, on retournera l'information vide ***void***.

La méthode ***main ( )*** d'une classe est celle qui est lancée juste après l'***instanciation*** (***constructeur***). Elle ne retourne aucune information.

Java fonctionne sur une machine virtuelle (***JVM*** : Java Virtual Machine), il est donc capable de s'exécuter sans modification sur n'importe quel processeur ou système d'exploitation. On créer un fichier textuel d'extension *.java*, on le compile avec le compilateur fourni par Sun (*javac.exe* dans le

JDK) pour produire un fichier *.class* semi-compilé, et c'est ensuite la machine *java* (*java.exe* sous Windows) qui exécute le code en le transformant au format du SE.

En java, **tout est OBJET**. Tout dérive d'*Object*.

I LES CLASSES

1.1 Déclaration

En ***programmation objet***, une **classe** est la représentation d'une composante du monde réel, comportant des **propriétés** qui définissent sa structure et des **méthodes** qui décrivent ses comportements. On appelle **membres** d'une *classe* l'ensemble de ses *propriétés* et ***méthodes***.

```
public class MaClass {
//Propriétés
……
……
//méthodes
……
……
}
```

1.2 Héritage et héritage multiple

Une classe peut hériter d'une autre classe et, partant, pouvoir disposer de toutes les propriétés et méthodes de la classe dont elle dérive.

```
public class MaSousClass extends MaClass {
//Propriétés
……
……
//méthodes
……
……
}
```

Java ne gère pas explicitement l'***héritage multiple***. Cependant, il dispose d'un système d'***implémentation*** qui permet de dire d'une classe qu'elle reproduit la structure d'une autre.

```
public class MaSousClass extends MaClass implement AutreClass{
//Propriétés
……
……
//méthodes
……
……
}
```

1.3 Constructeur

Le constructeur d'une classe est la méthode qui décrit les actions à mener lors de l'instanciation d'une classe. On réservera notamment l'espace mémoire nécessaire, on attribuera un ***OID*** (*Object IDentifier*) à l'instance, etc. Le constructeur est une méthode publique, sans notion de retour, portant le même nom que la classe. On peut disposer de plusieurs constructeurs en fonction des paramètres que l'on passe (polymorphisme).

```
public class MaSousClass extends MaClass {
//------------------Propriétés
private int prop1;
private String prop2;
//-----------------Méthodes
public MaSousClass() {
    super();
}
public MaSousClass(String s) {
    prop2  s ;
}
……….
}
```

L'appel au constructeur se fera par l'utilisation de la méthode new.

```
public class UneAutreClass {
//------------------Propriétés
MaSousClass uneInstance ;
//-----------------Méthodes
public void uneMethode() {
    uneInstance = new MaSousClass();
}
……….
}
```

1.4 La procédure Main

La procédure **main** associée à une classe est l'action par défaut qui sera exécutée juste après l'instanciation. On ne peut y trouver que des références à des *propriétés* et *méthodes* ***publiques*** de la classe. Elle prend une liste de paramètres stockés dans un tableau de chaînes de caractères.

```
public class UneClass {
//------------------Propriétés
… ;
//-----------------Méthodes
public void main(String args[ ]) {
    UneClass uneInstance = new UneClass();
}
……….
```

```
}
```

II PORTÉE ET COMPORTEMENT DES OBJETS ET MEMBRES

La portée d'un objet est la limite de sa visibilité : uniquement à l'intérieur de l'objet, dans une classe et ses sous classes, par tout le monde.

Par défaut, un membre est visible dans tout le package dans lequel il est défini

private	visibilité par les seuls membres de la classe (pas visible dans les sous-classes)
protected	visibilité par les membres de la classe et des sous-classes
public	accessible à l'extérieur de la classe (pour les classes et les membres)

Le comportement d'un objet indique ses possibilités d'extension

Le comportement par défaut, sans précision, indique un membre standard

static	Le membre est défini pour la classe et commun à toutes ses instances
abstract	La classe ou la méthode ne peut être instanciée directement, on devra redéfinir le code de la méthode ou créer une classe héritière
final	La classe ou la méthode ne pourra être redéfinie à un niveau inférieur (héritage ou surcharge interdit)

La déclaration d'un membre (méthode ou propriété) peut donc cumuler les deux ensembles :

[Portée] [Comportement] Type membre ;	ou [Portée] [Comportement] Classe
public static int nbInstances ; protected final void bilan( ) {…}; private Boolean trouvé ;	public abstract Identité {}

III LES PRINCIPAUX TYPES

3.1 Les types de base

TYPES SIMPLES

ALGORITHMIQUE	VISUAL	BASIC	JAVA	C
ENTIER	BYTE, INTEGER, LONG	byte, short, int, long ou Integer, Long	BYTE, SHORT, INT, LONG	
REEL	SINGLE, DOUBLE	float, double ou Float, Double	FLOAT, DOUBLE	
BOOLEEN	BOOLEAN	boolean ou Boolean	INT	
CHAINE	STRING ou STRING(x)	char ou Char, String	CHAR, *CHAR	
DATE	DATE	Date		
INDEFINI	VARIANT			

TYPES COMPLEXES

ALGORITHMIQUE	VISUAL BASIC	JAVA	C
TABLEAU [...] DE typ	ARRAY(liste de valeurs) NomTableau(...) AS typ	typ NomTableau[ ]	typ *NomTableau(...)
ENREGISTREMENT TYPE	Il s'agit d'une classe	STRUCT	donnée comportant un seul caractère

Les types commençant par une majuscule sont des classes de référence, auxquelles sont associées diverses méthodes. Les types en minuscule sont des types primitifs.

D'innombrables classes existent dans Java. Celles définissant les types les plus couramment utilisés sont : *String, Double, Integer, Boolean, Date*.

3.2 Les tableaux d'objets

En java, la déclaration d'un tableau consiste à donner son type et son nom suivi de crochet.

L'instanciation d'un tableau sert à définir le nombre de case qu'il contient.

```
Public String tableauChaine[ ] ;
tableauChaine = new String[50] ;
```

Lorsque l'on déclare un tableau d'objets complexes, il est important de penser à instancier à la fois le tableau et les objets contenus dans ses cases.

```
Public Client tabClients[ ] ;
tabClients = new Client[50] ;     //initialise le tableau
tabClients[10] = new Client( );   // initialise la case 10 avec le
constructeur de l'objet Client
```

3.3 Java et le casting (transtypage)

La conversion des données d'un type vers l'autre (chaîne en nombre, entier en réel, ...) est chose compliquée. Java est en effet un langage fortement typé (par ses *classes*) et il distingue un *int* d'un *Integer*, un *double* d'un *Double*... Le passage d'un type à l'autre se nomme le ***casting*** (*to cast* : faire correspondre, en anglais) ou le ***transtypage***.

Pour que deux objets puissent être de types compatibles, il faut qu'il existe un lien commun entre les deux (lien d'héritage) ou une méthode dans une classe permettant la conversion dans l'autre classe. S'il existe un lien d'héritage ou entre types primitifs, on utilisera le casting direct :

```
Type1 objet1 ;
Type2 objet2 ;
…
objet1= (Type1) objet2
```

Exemples

```
int  i ;                                public class Animal {  } ;
```

```
double d ;

d = (double) i ;
i = (int) d ;
```

```
public class Humain {  } ;
…
Animal a = new Animal( ) ;
Humain leChainonManquant ;
leChainonManquant = (Humain) a ;
```

Dans tous les autres cas, il faudra que la méthode de conversion existe, ou bien il sera nécessaire de la créer. Voici quelques principes que l'on peut multiplier à l'infini.

Pour transformer	**vers**	**utiliser la méthode**
Integer ii, Double dd, Date da etc	String gg	gg = ii.toString( ) gg = dd.toString( ) gg = da.toString( )
double d	String gg	gg = Double.toString(d); ou gg = String.valueOf(d)

String gg	Integer ii	try {
		ii = new Integer(gg.trim()); \\}
		catch (NumberFormatException e) {
		…
		}
		On doit intercepter une erreur si la chaîne est mal conformée. On procèdera de la même façon avec Double, Date, etc

Integer ii	Double dd	dd = new Double(ii.intValue())
Integer, Long, Float ou Double nn:	int i, double d,float f, etc	i = nn.intValue() d = nn.doubleValue() f = nn.floatValue()
short, char, int n	Integer ii	ii = new Integer(n)
short, char, int, long, float ou double n	Integer ii	ii = new Integer((int) n)
short, char, int, long, float ou double n	Double dd	dd = new Double(n);

3.4 Les paquetages (Package)

Dérivé du C, Java en récupère un outil très puissant mais fort contraignant : la capacité de réutiliser des composants développés par d'autres, à l'image de l'ajout de composants en VB (pour les onglets par exemple). La contrainte associée à cette techniques réside dans le fait que Java ne considère aucun de ces composants comme élément par défaut, et que dès que l'on souhaite manipuler des chaînes de caractère (String), des interfaces graphiques (AWT) ou d'autres éléments standards, il est nécessaire d'en importer les définitions.

Les composants associés à Java sont enregistrés dans des Packages ou Paquetages regroupant un ensemble de fonctionnalités proches. 7 de ces bibliothèques sont essentielles :

- java.lang : c'est le cœur du système, avec la définition de tous les objets de bases (structures de bases et threads)
- java.util : des outils pour l'accès aux données plus complexes (vecteurs, classes, …) et un générateur de nombres aléatoires

- java.io : entrées sorties
- java.net : manipulations sur le réseau (TCP/IP en particulier, avec le composant servlet)
- java.awt : ensembles des composants d'interface graphique qui tend à disparaître
- java.swing : évolution de AWT simplifiant la gestion événementielle et les erreurs
- java.applet : pour définir des applets
- java.sql : capacité d'exécuter des interrogations de base de données

Ces bibliothèques comportent deux parties :

- les interfaces : ce sont des fonctions utilisables mais ce ne sont pas des classes
- les classes : qui peuvent être instanciées

Lorsque l'on souhaite utiliser des fonctions de ces Packages, il faut :

- importer le package en entier : import java.awt.* ;
- importer la classe qui nous intéresse : import java.lang.String

IV LES STRUCTURES DE CONTRÔLE

Alternatives et choix

ALGORITHMIQUE	**VISUAL BASIC**	**JAVA**	**C**			

SI condition
ALORS
ActionsVrai
SINON
ActionsFaux
FIN SI
	IF condition
	THEN
	ActionsVrai
	ELSE
	ActionsFaux
	END IF
		IF (condition)
		{ActionsVrai;}
		ELSE
		{ActionsFaux;}
		| IF
		(condition)
		{ActionsVrai;}
		ELSE
		{ActionsFaux;}
		|
			SELON
			expression
			FAIRE
			CAS val1 :
			actions1
			FIN CAS
			...
			CAS valn :
			actionsn
			FIN CAS
			AUTRES :
			actionautre
			FIN CAS
			FIN SELON
				SELECT CASE
				expression
				CASE val1
				actions1
				...
				CASE valn
				actionsn
				ELSE
				actionautre
				END SELECT
					SWITCH
					(expression) {
					CASE val1 :
					{actions1}
					BREAK
					...
					CASE valn :
					{actionsn}
					BREAK
					DEFAULT :
					{actionautre}
					}
						SWITCH
						(expression) {
						CASE val1 :
						{actions1}
						BREAK
						...
						CASE valn :
						{actionsn}
						BREAK
						DEFAULT :
						{actionautre}
						}

Répétitives

ALGORITHMIQUE	**VISUAL BASIC**	**JAVA**	**C**
POUR var ALLANT DE début A fin PAR PAS DE x FAIRE INSTRUCTIONS FIN POUR	FOR var = début TO fin STEP x INSTRUCTIONS NEXT var	FOR {var = début ; var<fin+1; var = var+x} { INSTRUCTIONS };	FOR {var = début ; var<fin+1; var = var+x} { INSTRUCTIONS };
TANT QUE condition FAIRE INSTRUCTIONS FIN TANT QUE	DO WHILE condition INSTRUCTIONS LOOP *il existe d'autres possibilités	WHILE (condition) { INSTRUCTIONS };	WHILE (condition) { INSTRUCTIONS };
REPETER INSTRUCTIONS JUSQU'A condition	DO INSTRUCTIONS UNTIL condition	DO INSTRUCTIONS WHILE non(condition);	

V LES OPERATEURS

5.1 Opérateurs logiques

Opérateur	Définition
!	NOT booléen (unaire) Change true en false ou false en true. En raison de sa priorité basse, vous devez inclure cette instruction entre parenthèses.
&	AND évaluation (binaire) Renvoie true seulement si les deux opérandes valent true. Évalue toujours deux opérandes.
^	XOR évaluation (binaire) Renvoie true si un des deux opérandes seulement vaut true. Évalue deux opérandes.
|	OR évaluation (binaire) Renvoie true si un ou les deux opérandes valent true. Évalue deux opérandes.
&&	AND conditionnel (binaire) Renvoie true seulement si les deux opérandes valent true. Il est dit “conditionnel” car il n'évalue le second opérande que si le premier vaut true.
||	OR conditionnel (binaire) Renvoie true si un ou les deux opérandes valent true ; renvoie false si les deux valent false. Le second opérande n'est pas évalué si le premier vaut true.

5.2 Opérateurs d'affectation

Opérateur	Définition
=	Affecte la valeur de droite à la variable de gauche.
+=	Ajoute la valeur de droite à la valeur de la variable de gauche ; affecte la nouvelle valeur à la variable initiale.
-=	Soustrait la valeur de droite de la valeur de la variable de gauche ; affecte la nouvelle valeur à la variable initiale.
*=	Multiplie la valeur de droite avec la valeur de la variable de gauche ; affecte la nouvelle valeur à la variable initiale.
/=	Divise la valeur de la variable de gauche par la valeur de droite ; affecte la nouvelle valeur à la variable initiale.

5.3 Opérateurs de comparaison

Opérateur	Définition
<	Inférieur à
>	Supérieur à
⇐	Inférieur ou égal à
>=	Supérieur ou égal à
==	Égal à
!=	Différent de

VI ERREURS ET DÉBOGAGE

Le langage Java est à la fois un outil puissant et un système complexe. Si les connaisseurs du langage C ne seront pas déstabilisés, les programmeurs Basic ou Pascal pourraient se sentir perdus.

C’est pourquoi il est nécessaire de penser OBJET lorsque l’on attaque Java, et d’oublier le procédural et l’événementiel dans la conception du cœur du programme.

Certaines erreurs classiques reviennent, donnant des messages abscons que vous apprendrez à repérer et interpréter. Une erreur Java est nommée Exception et doit être interceptée.

Cette fiche repère les erreurs générales en les proposant par grands domaines.

6.1 BOUCLES ET TEST

Pour cette partie, les erreurs classiques sont : boucle sans fin (problème algorithmique plus que Java), alternative qui ne s'exécutent jamais, tests mal formulés ou formulés à l'envers. Pour éviter ces erreurs, appliquez ces quelques conseils :

- Le début d'un paquet d'instructions est signalé par une accolade ***{*** et fermé par une autre accolade ***}***.
- Le ***;*** met fin à un *if* et annule le *else*.
- Hérité du C, toute condition de test (dans un *if, for, while*…) doit être mise entre parenthèses.
- La comparaison de deux valeurs basiques (*int, boolean, double*…) se fait par le signe **==**.
- La ***comparaison de chaînes de caractère*** se fait par ***chaine1.equals(chaine2)***.
- Le ou logique s'écrit ‰‰||‰‰ (ALT Gr + 6 deux fois)
- Le et logique s'écrit ***&&***.
- La négation est représentée par le ***!***.

Rappel des écritures et principales erreurs

SI	If (test) { . . . . ;} else { . .. . . ;} attention à l'oubli d'accolades, seule la première ligne serait prise dans le si ou le alors
POUR	for (valeur de départ ; test de continuation ; pas d'évolution) { } attention à l'oubli d'accolades qui n'exécuterait que la première instruction en boucle, attention au test (celui d'un tant que)
TANT QUE	while (test de continuation) { } attention aux accolades
REPETER	do { } while (test de continuation) attention aux accolades attention au test (celui d'un tant que)

6.2 L'erreur NullPointerException

Cette erreur intervient chaque fois que l'on tente d'accéder aux éléments publics d'un objet non initialisé (par un ***new***). En clair, si vous essayez de faire

```
objet1.méthode()
```

et que vous n'avez jamais fait

```
objet1 = new ClasseObjet1()
```

Vous obtiendrez ce message

ATTENTION Pour les tableaux d'objets, vous devez initialiser le tableau

```
ClasseObjet1[ ] tableau
```

```
tableau = new ClasseObjet1[xx]
```

Et aussi initialiser le contenu de chaque case pour y accéder

```
tableau[a] = new ClasseObjet1()
```

6.3 Héritage et portée

Les erreurs pour l'utilisation de membres (propriétés et méthodes) non disponibles sont dues à une mauvaise écriture ou, plus probablement, à une erreur de portée.

- Les membres ***private*** ne sont pas héritables
- Les membres ***protected*** et ***public*** sont hérités
- On ne peut jamais utiliser un membre ***protected*** en dehors de la classe ou sous-classe

6.4 Try et Catch, Throws

Toute action susceptible a priori de créer une erreur (notamment les entrées sorties fichiers, les interactions avec des bases de données, les manipulations de dates) doivent être encadrées par les instructions ***try*** et ***catch***.

```
try {
    actions dangereuses;
}
catch (typeErreur variableErreur) {
    System.out.println(variable.Erreur);
}
```

Certaines méthodes ou classes peuvent ne pas traiter directement une erreur mais renvoyer l'interception à un niveau supérieur. On utilisera alors la clause

```
throws TypeException
```

.

6.5 Débogage

LA METHODE MAIN

Pour tester une application java (c'est à dire l'interaction entre les instances des différentes classes), il est nécessaire de créer une classe qui constituera le programme principal.

Cette classe devra au moins posséder une méthode ***main( )*** et éventuellement les éléments nécessaires pour lancer une interface graphique.

ATTENTION La méthode main est un élément ***static***. Cela signifie qu'elle ne peut utiliser les *méthodes* et *propriétés* non définies comme ***static*** (et donc tous les paramètres définis dans la classe

qui l'utilise). C'est pourquoi il est habituel de déclarer dans la méthode main une instance de la classe programme. Exemple

```
public class ProgramTest() {
//propriétés
protected int uneProp ;
…
//méthodes
public int methodTest( ) {….}
…
public static void main(String[] args) {
    ProgramTest unTest = new ProgramTest() ;
    int x = unTest.methodTest( ) ;
    …
}
…
}
```

SYSTEM.OUT ET SYSTEM.IN

De manière à interagir avec une application sans entrer dans la phase de production d'une interface graphique, on pourra effectuer l'affichage de résultats avec

```
System.out.println(objet..)
```

Ou l'entrée de valeur (en utilisant si nécessaire le transtypage) avec

```
x = (Type) System.in.readln()
```