Java M2 TTT

Valérie Gautard

Contenu

2. Java, les bases

2.1. Introduction

2.1.1. Un bref historique

- origine : 1991 - ingénieur de SUN qui ont cherché à concevoir un langage applicable à de petits appareils électriques (code embraqué). L'idée est de traduire un programme source, non pas en langage machine mais en pseudo langage universel disposant des fonctionnalités communes à toutes les machines. Ensuite ce code intermédiaire, appelé byte code est compact et portable. Il suffit alors que la machine dispose d'un programme approprie appelé machine virtuelle permettant d'interpréter le code intermédiaire sur la machine concernée. Le projet de code embarqué n'a pas abouti en tant que tel.
- Ces concepts ont été repris en 1995 pour la réalisation du logiciel HotJava, navigateur web écrit par SUN en java capable d'exécuter des applets (application) écrites en byte code
- D'autres navigateurs web ont suivi et contribué à l'essor du langage : J2SE, JDK (engouement constant)
- Modèle objet simple mais puissant avec une API (bibliothèque standard)
- Langage généraliste pédagogique et utilisé dans les entreprises

2.1.2. Java et la programmation orientée objet (POO)

Objectif de la programmation orientée objet est de contribuer à :

- La fiabilité
- La réutilisation

Elle introduit de nouveaux concepts tels que : l'objet, l'encapsulation, les classes et l'héritage.

Les concepts

- POO :
  Le code est construit non pas autour de ses fonctionnalités mais autour de l'objet
- Concept d'encapsulation : (il n'est pas possible d'agir directement sur les données d'un objet, il est nécessaire de passer par une interface), abstraction des données
- Concept de classe : généralisation de la notion de type
- Concept d'héritage
- Concept de polymorphisme : on traite de la même manière des objets de types différents

2.2. Une première application

2.2.1. Un exemple simple

```
public class Exemple
{
   public static void main(String [] args)
   {
      int n;
      double x;

      n=5;
      x=2*n+1.5;

      System.out.println("n = "+n);
      System.out.println("x = "+x);

      double    y;
      y = n * x+12;
      System.out.println("y = "+y);

   }

}
```

2.2.2. Structure générale d'un programme

- Bloc : ensemble d'instructions délimitées par {}

- 3 formes de commentaires :

  /** commentaire */ forme de commentaire pouvant être exploitée par le générateur javadoc
  /* commentaire */
  // Commentaire jusqu'à la fin de ligne

- Un programme Java est une collection de classe qui décrit des objets qui seront manipulés à l'exécution
  - Structure d'un programme autour des objets
  - Une classe est constituée
    - De variable : désignant des données qui sont des valeurs de tout type
    - De méthode : action, cad fonction ou procédure

API (application programming interface) : classes définies par ailleurs et mises à la disposition du programmeur

Les classes unies par une même sémantique sont regroupées dans des paquetages

- La méthode main :
  - L'exécution d'un programme java commence par l'exécution de la méthode main
  - En-tête toujours identique
  - public : droit d'accès, obligatoire pour que le programme puisse s'exécuter
  - args :
    - Tableau de chaine de caractère
    - obligatoire permet de passer des arguments
  - static : précise que la méthode n'est pas liée à une instance (un objet) de la classe
- La casse doit être respectée

Java est un langage typé

2.2.3. Exécution d'un programme

- Saisir et sauvegarder le programme. Le nom du fichier doit impérativement être celui de la classe publique avec l'extension .java.

  Ex : Exemple.java
- Compilation : produit non pas du code source mais du code intermédiaire (bytecode). Si la compilation s'est bien déroulée alors on obtient un fichier avec l'extension .class

  Ex : Exemple.class
- Le code est ensuite lance par l'intermédiaire de la machine virtuelle Java

2.3. Un aperçu général de Java

2.3.1. Les types primitifs de Java

- Type entier :
  - byte 1 octet -128 ;+127
  - short 2 -32768 ;+32767
  - int 4
  - long 8
- Type flottant
  - float 4
  - double 8
- Type caractère
  - char : variable de type caractère

- ‘a’ ‘E’ : constantes de type caractère
- Par convention ;
  - \b retour arrière
  - \t tabulation
  - \n saut de ligne
  - \f saut de page
  - \r cariage return
  - \ » «
  - \ ‘ ‘
  - \\ \
- Type booleen boolean

2.3.2. Les variables

Def 1 :
Une variable est une case mémoire permettant de stocker des données

Def 2 :
On appelle identificateur le nom désignant une variable

- Le langage Java fait la différence entre majuscule et minuscule
- Un identificateur est formé de lettres ou de chiffres, _ et $ le premier caractère doit être une lettre ou _ et $

  Rque : il est conseille de ne pas utiliser $ qui intervient dans les mécanismes internes
- Les variables doivent être déclarées avant d’être utilisées
- Les variables peuvent être initialisées lors de leur déclaration

  Ex :

```
import java.util.*;

public class Var1
{
        public static void main (String args[])
        {
                int n=15;

                int k=10 ;
                int p = 2*n ;

                int j ;
```

```
        Scanner lire = new Scanner(System.in) ;
        j= lire.nextInt() ;
        int jj=2*j ;

    }
}
```

- Cas des variables non initialisées :
  En java, une variable n'étant pas initialisée ne peut pas être utilisée :

Ex

```
public class Var2
{
    public static void main (String args[])
    {
        int     n ;
        System.out.println (" n= " +n) ; //erreur de
        compilation

        int p=n+3 ; //erreur de compilation

    }
}
```

2.3.3. Les constantes

- Le mot clef final indique que la valeur d'une variable ne doit pas être modifiée pendant l'exécution du programme

Ex

```
import java.util.*;

public class Cons
{
    public static void main (String args[])
    {
        final int n=20 ;
        n=n+5 ; //erreur n est constante
        Scanner lire = new Scanner(System.in) ;
        n=.lire.nextInt() ; //erreur

        int p ;
        p= lire.nextInt();  //Ok bien que la valeur ne soit connue qu'a
        l'exécution

    }
}
```

- Rque : L'initialisation d'une variable final peut être différée.

2.3.4. Les operateurs

Les opérateurs ne sont définis que pour des opérandes de même type sauf par le jeu des conversions.

- Les operateurs arithmétiques :
  - operateurs binaires (portant sur deux opérandes)
    + - * /
  - operateur unaire :
    - (-n) +

    Les règles de priorités des operateurs sont celles de l'algèbre traditionnel (Comportement en cas d'exception)

- Les opérateurs relationnels

<	Inferieur à
<=	inferieur ou égal
>	supérieur
>=	supérieur ou égal
==	égal
!=	différent de

Ca particulier des valeurs infinity et nan

- Les operateurs de manipulation de bits

&	et
|	ou inclusif
^	ou exclusif
<<	décalage à gauche
>>	décalage à droite (avec propagation du bit de signe)
>>>	décalage logique à droite
~	complément à 1

Table de vérité :

opérande 1	0	0	1	1
opérande 2	0	1	0	1
et &	0	0	0	1
ou inclusif |	0	1	1	1
ou exclusif^	0	1	1	0

Exemple

n	01101110
p	10110011
n&p	00100010

n|p	11111111
n^p	11011101
~n	10010001
n << 2	10111000
n >> 3	00001101
p >> 3	11110110
p >>> 3	00010111

- Les opérateurs logiques

!	négation
&	et
^	ou exclusif
|	ou inclusif

rque : && (et), || (ou inclusif) sont des operateurs de court circuit. La seconde opérande n'est évaluée que si on a besoin de connaitre sa valeur pour décider si l'expression est vraie ou fausse.

oper

- L'operateur d'affectation :
  - =
  - variable = variable operateur expression ⇔ variable operateur = expression

    +=, -= *= /= ^= &= <<= >>= >>>=
- Les operateurs de décrémentation et incrémentation
  - pré-incrémentation ++y
  - post-incrémentation y++
  - pré-décrémentation --y
  - post-décrémentation y--

Exemple

```
public class Incrementation
{
   public static void main(String [] args)
   {
        float    x,y=2;


        y++;    /* y=3*/

        x=y++;          /* x=3 et y=4 */

        x=++y;          /* y=5 et x=5 */
        }
}
```

- Les conversions
  - Les conversions implicites vont du plus petit vers le plus grand :
    int → long → float → double
    Ces conversion sont appelées conversion d'ajustement.
  - Les operateurs numériques ne sont pas définis pour les types byte, char et short. Java prévoit que toute valeur d'un de ces types est convertie en int. On parle alors de promotion numérique ou encore de conversion systématique
  - Les conversions par affectation
    byte → short → int → long → float → double
    char → int → long → float → double
  - operateur de cast :
    (type) variable

2.4. Communiquer

2.4.1. Les entrées/sorties

Pour communiquer une information, l'ordinateur affiche un message à l'écran. On dit qu'il réalise une sortie : out.
A l'inverse lorsque l'utilisateur communique des données par l'intermédiaire du clavier, il effectue une entrée : in.

Dans les exemples précédents, l'affichage de texte et de valeur a été réalisé par une fonction prédéfinie du langage Java :
System.out.println() ;
Le langage Java fournit un ensemble de librairie qui facilite la programmation. Ainsi, la bibliothèque de fonctions mathématiques s'appelle Math, celle relative à la gestion des éléments bas niveau de l'ordinateur (écran, clavier) s'appelle System. La librairie system composée de deux sous-ensembles : in, out

Affichage de données

On utlise System.out.println() ;

Exemple :
Affichage d'une valeur :

```
int valeur = 12;
System.out.println(valeur);              // 12
```

Affichage d'un commentaire :

```
System.out.println(« Valeur vaut : » + valeur);        // Valeur vaut 12
```

Affichage d'une expression

int a = 10, b = 5 ;
System.out.println(« le produit de » + a + »et » + b + « vaut : » + a*b) ;

Rque pour une addition, on écrirait :

System.out.println(« la somme de » + a + »et » + b + « vaut : » + (a+b)) ;

Pour changer de ligne :

System.out.print(« Qui seme le vent «) ;
System.out.print(« recolte la tempete «) ;
→ Qui seme le vent recolte la tempete

System.out.println(« Qui seme le vent «) ;
System.out.print(« recolte la tempete «) ;
→ Qui seme le vent
recolte la tempete

La saisie de données

Le package java.util fournit la classe Scanner beaucoup plus facile d'utilisation que Sytem.in.read() ;

Pour l'utiliser, il faut :

- importer lporta classe Scanner :
  import java.util.* ;
- créer un objet de type Scanner :
  Scanner lectureClavier = new Scanner(System.in) ;
- utiliser une méthode de la classe Scanner :
  - pour saisir un entier : nextInt() ;
  - pour saisir un long : nextLong() ;
  - pour saisir un flottant : nextFloat() ;
  - pour saisir un double : nextDouble() ;
  - pour saisir un String : next () ;

Exemple

```
import java.util.*
public class Lire
{
        public static void main (String args[])
        {
```

```
                Scanner lecture = new Scanner(System.in) ;

                System.out.println(" Donner un entier ") ;
                int k=lecture.nextInt() ;
                System.out.println(" L'entier donne vaut : "  + k) ;

        }
    }


}
}
```

2.5. Les tableaux

2.5.1. Déclaration et création de tableaux

Les tableaux de données permettent de stocker des variables multidimmensionnées

Ex : vecteur, matrice….

Déclaration

Syntaxe :
TypeT tab[] ; ⇔ TypeT [] tab ;

Création

L'operateur new permet d'allouer l'emplacement d'un tableau

Syntaxe :

tab[] = new TypeT[size] où size est la taille du tableau

Remarque : La valeur de l'expression fournie par new est calculée au moment de l'exécution et non de la compilation. Elle peut donc être variable
Exemple :

```
import java.util.*
public class Tableau
{
    public static void main (String args[])
    {
        Scanner lecture = new Scanner(System.in) ;
        int n = lecture.nextInt() ;
```

```
                int t[] = new int[n] ;

        }
}
```

On peut aussi créer un tableau en l'initialisant :

```
int       n,p ;
…..
int [] = {1,n,n+p,2*p, 12}
```

On a alors cree un tableau de 5 entiers

2.5.2. Utilisation de tableaux

Accès aux éléments d'un tableau.

Syntaxe :

t[i] : permet d'accéder au ieme élément du tableau
Le premier élément du tableau correspond à l'indice 0

Exemple :

```
int t[]=new int[5] ;
….
t[0] = 15 ;
…
t[2] ++ ;
…
System.out.println(t[4]) ;
…..
```

```
import java.util.*;

public class Tableau
{
        public static void main (String args[])
        {
                int n ;
                double t1[] ;

                t1 = new double[5] ;
                t1[0] = 15 ;

                Scanner lire = new Scanner(System.in) ;
                 System.out.println(« donner le nombre de case voulu ») ;
                 n = lire.nextInt() ;
                 int t2[] = new int t2[n] ;
```

```
        int t3[]={1,n,2*n,12,6) ;
        System.out.println(" t3[4] = "  + t3[4] );
       System.out.println(" taille de t3 : "+ t3.length) ;

    }
}
```

Taille d'un tableau

Syntaxe :

t.length permet de connaitre le nombre d'éléments d'un tableau

2.6. Structure de programmation

2.6.1. Structure conditionnelle

2.6.1.1. Instruction simple, bloc d'instruction

Def :

Une instruction simple est une expression suivie d'un point virgule ;

Attention : ; fait parti de l'instruction, ce n'est pas un séparateur

Def :

Un bloc d'instruction est une suite d'instructions délimitées par des {}

Syntaxe

```
{
    /* suite d'instructions en tout genre */
}
```

Rque : un bloc d'instruction peut remplacer une instruction simple

2.6.1.2. Instruction if

Syntaxe :

```
if (expression)
{
        /*bloc d'instruction 1*/
}
else
{
        /*bloc d'instruction 2*/
}
```

Si expression est vraie alors le bloc d'instruction 1 est exécuté sinon le bloc d'instruction 2 est exécuté

Exemple :

```
import java.util.*;

public class Max
{
        public static void main (String args[])
        {
                float x,y;

                Scanner lire = new Scanner(System.in) ;
                System.out.println("Donner deux flottants");
                x =.lire.nextFloat();
                y = lire.nextFloat();

                if (x < y)
                        System.out.println("max = " + y);
                else
                        System.out.println("max = " + x);

  }
}
```

Imbrication des instructions if

Un exemple :

```
if (a<=b)  if (b<=c) System.out.println("ordonne");
else System.out.println("non ordonne");
```

Un else se rapporte toujours au dernier if rencontré auquel un else n'a pas encore été attribué.

Instruction switch

Le *switch* est une structure multibranchement dans laquelle une valeur est comparée à plusieurs constantes.

Syntaxe

```
switch (expression)
{
        case exp1 :
                BlocInstruction1;
                break;

        case exp2 :
                BlocInstruction2;
                break;

                .
                .
                .
                .
        case expN :
                BlocInstructionN;
                break;

        default
                BlocInstruction;
}
```

Rque :

- L'instruction break est nécessaire pour un choix exclusif
- L'utilisation de default est optionnelle
- Tous les cas doivent être différents
- Expi doivent être des entiers (ie de type int ou char)

Exemple :

```
import java.util.*;

public class Default
{
      public static void main (String[] args)
      {
            int n ;


            Scanner lire = new Scanner(System.in) ;
```

```
            System.out.print ("donnez un nombre entier : ") ;
            n = lirenextInt() ;
            switch (n)
            {
                  case 0 : System.out.println ("nul") ;
                        break ;
                  case 1 : System.out.println ("un") ;
                        break ;
                  default : System.out.println ("grand") ;
            }
            System.out.println ("Au revoir");
  }
}
```

2.6.2. Structure répétitive

Ces structures permettent de répéter un certain nombre de fois un ensemble de commandes.

while

Syntaxe :

```
while (expression)
{
      /*bloc d'instruction*/
}
```

Le bloc d'instruction est exécuté tant que expression est vraie

Exemple :

```
import java.util.*;

public class While1
{
      public static void main (String args[])
      {
            int n, som ;
            som = 0 ;
            Sacnner lire = Scanner(System.in);

            while (som < 100)
            {
                  System.out.print ("donnez un nombre : ") ;
                  n = lire.nextInt() ;
                  som += n ;
            }
            System.out.println ("Somme obtenue : " + som) ;
      }
```

```
}
```

do while

Syntaxe

```
do
{
        /*bloc d'instruction*/
} while (exp1) ;
```

Rque : Le bloc d'instruction est exécuté au moins une fois.

Exemple :

```
import java.util.*;

public class Do1
{
  public static void main (String args[])
  {
    int n ;
    Scanner lire = new Scanner(System.in) ;
    do
    {
          System.out.print ("donnez un nombre >0 : ") ;
          n = lire.nextInt() ;
          System.out.println ("vous avez fourni " + n) ;
     }
    while (n <= 0) ;
    System.out.println ("reponse correcte") ;
  }
}
```

for

Syntaxe :

```
for(exp1,exp2,exp3)
{
        /* bloc d'instruction */
}
```

Où exp1 : initialisation d'une variable de contrôle appelée aussi index
exp2 : expression conditionnelle définissant la condition d'arrêt de la boucle
exp3 : une ou des expressions modifiant l'index

Rque :

- Cette syntaxe peut être équivalente dans certains cas à while

- Il est possible d'omettre exp1, exp2 ou exp3 mais les ; restent obligatoires.

Exemple :

```
public class For1
{
   public static void main (String args[])
  {
    int i ;
    for (i=1 ; i<=5 ; i++)
     {
            System.out.println (i ) ;
            System.out.print ("passage dans la boucle ") ;
     }
  }
}
```