Chapitre 2

Programmation fondamentale en Java

Le langage Java a fortement été inspiré par le langage C++ avec quelques variantes essentielles. D'une part il est plus simple : il n'y a ni conversion implicite de type, ni structure, ni pointeur, ni héritage multiple, ni polymorphisme, ni surcharge d'opérateur, ni préprocesseur. D'autre part il apporte des fonctionnalités nouvelles, telles que la programmation uniformisée de l'interface graphique et la programmation multi-tâche.

Nous allons voir dans ce chapitre, et les deux suivants, la programmation pure, c'est-à-dire le langage Java comme un langage C++ simplifié.

2.1 Structure d'un programme Java

Reprenons les programmes source de l'application et de l'applet que nous avons vues à titre de premiers exemples et commentons-les.

2.1.1 Structure d'une application

Première application.- Rappelons le programme source `Essai1.java` de notre première application :

```
class Essai1
   {
   public static void main(String args[])
      {
      System.out.println("Ceci est une application");
      }
   }
```

Commentaires.- 1^o) Le langage Java est un langage orienté objet (presque) pur. Un programme est une suite de définition de classes. Notre programme ne comprend la définition que d'une seule classe, à laquelle on a attribué le nom "Essai1".

- 2^o) Une seule classe du programme peut contenir la méthode `main()`, qui est le point d'entrée dans cette classe. La classe point d'entrée est indiquée par le nom du programme : c'est pourquoi on n'a pas le choix de celui-ci, c'est nécessairement « Essai1.java ».

- 3^o) La méthode `main()` est **statique**, c'est-à-dire qu'elle est appelée avec la classe et non liée à l'une de ses *instantiation* (un objet) par un opérateur point. Cette méthode est également **publique**, pour qu'elle puisse être visible de l'extérieur.

- 4^o) Les délimiteurs de bloc sont, comme en C et en C++, les accolades ouvrante '{' et fermante '}'.

- 5^o) La seule instruction est ici un appel de méthode. Elle se termine, comme en C et en C++, par un point-virgule ';'.

- 6^o) L'affichage sur le moniteur se fait grâce à la méthode, dont le seul argument est une chaîne de caractères, appelée `System.out.println()`. Il s'agit donc de la méthode `println()` (pour *PRINT LiNe*) de la classe `System.out`. La classe `System` est une classe prédéfinie de java, prenant en charge les liens avec le système. La classe `System.out` est une classe membre de celle-ci, concernant l'affichage sur la console.

- 7^o) Comme en C++, les constantes chaînes de caractères se placent entre guillemets verticaux $''$ (le même symbole pour le guillemet ouvrant et le guillemet fermant).

- 8^o) Le langage Java, comme le langage C++, fait la différence entre les majuscules et les minuscules.

2.1.2 Structure d'une applet

Première applet.- Rappelons le programme source `Essai2.java` de notre première applet :

```
import java.awt.*;
public class Essai2 extends java.applet.Applet
   {
   public void init()
      {
      resize(100, 150);
      }
   public void paint(Graphics g)
      {
      g.drawString("Ceci est une applet", 10, 10);
      }
   }
```

Commentaires.- 1^o) Les classes définies en Java par le programmeur sont en général dérivées de **classes Java standard**, c'est-à-dire livrées avec le kit de développement (elles se trouvent dans *Classes.Zip*, maintenant *Classes.jar*). Ces classes sont rassemblées dans des collections de classes appelées **paquetages** (*packages* en anglais). L'ensemble des paquetages Java standard est connu sous le nom de **bibliothèque de classes Java** (en anglais *Java class library*) ou *Java applications programming interface* (**Java API**).

- 2^o) Le programme source commence par indiquer le chemin des classes ou des paquetages dont il a besoin, grâce au mot clé `import`. Il s'agit de l'analogue du `path`, mais déclaré localement.

Les applets commencent souvent par :

```
import java.applet.Applet;
```

puisqu'une applet est une classe dérivée de la classe *Applet*. Dans ce cas le début du programme ci-dessus peut être réécrit de la façon suivante :

```
import java.awt.*;
import java.applet.*;
public class Essai2 extends Applet
```

le compilateur cherchant la classe `Applet` d'abord dans le package `java.lang` (pour *LANGuage*, non indiqué explicitement car il s'agit du paquetage par défaut ; il ne la trouve pas), puis `java.awt` (il ne la trouve pas non plus), puis enfin dans `java.applet` (il la trouve, sinon une erreur serait indiquée).

- 3^o) Nous avons aussi besoin d'utiliser du graphisme. On utilise pour cela le paquetage **awt** (pour *Abstract Windowing Toolkit*, c'est-à-dire boîte à outils de fenêtrage abstrait), qui fait partie des paquetages Java standard. On importe ici l'ensemble des classes du paquetage :

```
import java.awt.*;
```

mais nous aurions pu nous contenter de la seule classe à laquelle nous allons faire référence, à savoir à la classe *Graphics* :

```
import java.awt.Graphics;
```

- 4^o) La classe `Essai2` que nous définissons est une classe dérivée de la classe `Applet`. Ceci s'indique par le mot clé `extends`.

- 5^o) Cette classe est déclarée publique pour qu'elle puisse être exécutée lors de l'appel de l'applet.

- 6^o) La classe comprend deux méthodes, elles aussi à exécuter.

- 7^o) On remarquera la différence entre une application et une applet : une application a une méthode `main()` alors que ce n'est pas le cas d'une applet. Une applet commence, par contre, par trois méthodes `init()` et `paint()`, bien que de façon optionnelle.

- 8^o) La méthode `resize()` spécifie la dimension de la **fenêtre active** de l'applet, en pixels, le nombre de colonnes d'abord, le nombre de lignes ensuite. Si on dessine au-delà de cette fenêtre active, la partie du dessin situé au-delà de cette fenêtre ne sera pas visible par l'utilisateur.

- 9^o) Dans la méthode `paint()`, on commence par **instantier une classe** abstraite `Graphics`, c'est-à-dire à déclarer un objet `g` de type `Graphics`.

Une **classe abstraite** est une classe définie en un langage autre que le langage Java. On ne peut l'instantier qu'en déclarant l'objet comme argument d'une méthode.

- 10^o) On applique ensuite une méthode à cet objet, comme en C++, à l'aide de l'opérateur point '.'. En l'occurrence la méthode `drawString()` indique de dessiner (et non afficher car nous sommes sur une page graphique) une chaîne de caractères. Elle possède trois arguments : la chaîne de caractères et deux entiers indiquant où elle doit être placée à partir du bord en haut à gauche de la fenêtre, d'abord la colonne puis la ligne (en pixels).

2.2 Les commentaires

Le langage Java possède trois sortes de commentaires, les deux premiers étant empruntés aux langage C++ et C, le troisième étant une particularité de ce langage.

Commentaire abrégé.- Il s'agit du commentaire sur une ligne commençant par une double barre de fraction « // ».

Commentaire sur plusieurs lignes.- Il débute par « /* » et il se termine par « */ ».

Commentaire de documentation automatique.- Sa syntaxe est la suivante :

/** commentaire */

Il est utilisé par l'utitaire `javadoc`, livré avec le kit de développement, pour réaliser une documentation automatique.

2.3 Les calculs en Java

Nous allons indiquer comment se servir du langage Java en tant que calculette.

2.3.1 Les types élémentaires en Java

Le langage Java est un langage orienté objet (presque) pur. Le fait qu'il ne soit pas totalement pur est qu'il dispose cependant de huit **types élémentaires**, qui ne sont pas des classes, dont l'implémentation est bien définie (contrairement à ce qui se passe en C ou en C++) :

- le type **boolean** comprend les deux constantes **true** et **false** ;
- le type **byte** correspond à un entier relatif codé sur huit bits (un octet), dont les valeurs sont comprises entre -128 à 127 ;
- le type **short** correspond à un entier relatif codé sur seize bits (deux octets), de -32 768 à 32 767 ;

- le type **char** correspond à un caractère **unicode** (et non plus ascii, par exemple, codé sur huit bits), codé sur seize bits ;
- le type **int** est un entier relatif codé sur 32 bits, de -2 147 483 648 à 2 147 483 647 ;
- le type **long** est un entier relatif codé sur 64 bits, de - 223 372 036 854 775 808 à 223 372 036 854 775 807 ;
- le type **float** est un réel en virgule flottante suivant le standard **IEEE 754** codé sur 32 bits ;
- le type **double** est un réel en virgule flottante suivant le standard **IEEE 754** codé sur 64 bits.

2.3.2 Affichage du résultat d'un calcul

Nous avons vu comment afficher un texte en Java. Voyons comment afficher un nombre, et en particulier le résultat d'un calcul.

2.3.2.1 Cas d'une application

Affichage de nombres.- On peut utiliser la fonction `System.out.println()` déjà vue :

```
class Addition
   {
   public static void main(String args[])
      {
      System.out.println(2 + 3.05);
      }
   }
```

L'exécution donne bien 5.05.

On remarquera un différence entre Java, d'une part, et C et C++, d'autre part : en Java il n'y a pas de zéros supplémentaires.

Affichage de nombres et de texte.- Si on veut faire afficher, par exemple :

```
2 + 3.05 = 5.05
```

on mélangera texte et nombres de la façon suivante :

```
class Add_2
   {
   public static void main(String args[])
      {
      System.out.println("2 + 3.05 = " + (2 + 3.05));
      }
   }
```

L'exécution donne bien : 2 + 3.05 = 5.05.

Remarques.- 1^o) La première partie à afficher, la partie texte, est placée entre guillemets verticaux.

- 2^o) Le signe '+' qui suit n'est pas un signe arithmétique mais le signe de la concaténation en Java.

- 3^o) Le résultat numérique est placé entre parenthèses sinon le signe '+' serait considéré comme un signe de concaténation et nous verrions comme résultat :

```
2 + 3.05 = 23.05
```

2.3.2.2 Cas d'une applet

Conversion de nombres en texte.- Pour afficher un nombre, on peut utiliser la fonction déjà vue, à savoir `drawString()`, à ceci près qu'elle ne permet d'afficher que des chaînes de caractères. ll faut donc faire une conversion des nombres en chaînes de caractères. Une astuce consiste à imposer une conversion automatique en faisant précéder (ou suivre) un nombre d'un texte vide :

```
import java.awt.*;
public class Calcul extends java.applet.Applet
   {
   public void init()
      {
      resize(100, 150);
      }
   public void paint(Graphics g)
      {
      g.drawString(2 + 3 + "", 10, 10);
      }
   }
```

Remarques.- 1^o) Si on oublie la chaîne de caractères vide pour imposer la conversion, c'est-à-dire si on écrit :

```
g.drawString(2 + 3, 10, 10) ;
```

on obtient une erreur de compilation :

```
bash-2.03# javac Calcul.java
Calcul.java:10: Incompatible type for method. Can't convert
 int to java.text.AttributedCharacterIterator.
              g.drawString(2+3,50,50);
                           ^
1 error
```

2^o) Si on écrit la chaîne de caractères à gauche, et non plus à droite, de l'expression :

```
g.drawString("" + 2 + 3, 10, 10) ;
```

on n'obtient plus le résultat voulu, mais 23.

Ceci est dû à ce qui s'appelle la *règle d'associativité à gauche*. Le compilateur lit la chaîne de caractères vide, puis le signe d'addition ; il considère donc celui-ci comme le signe de concaténation. Lorsqu'il lit l'entier 2, il le convertit en la chaîne de caractères `"2"` (les chaînes de caractères ont préséances sur les entiers) ; la concaténation donne donc cette même chaîne de caractères. Lorsqu'il lit le signe d'addition, il le considère donc comme un signe de concaténation puis il transforme le nombre 3 en la chaîne de caractères `"3"`, d'où le résultat.

Autrement dit, il a lu :

```
("" + 2) + 3
```

là où nous aurions voulu qu'il lise :

```
"" + (2 + 3)
```

Bien entendu, il suffit de parenthéser nous-même pour faire la rectification.

Affichage de texte et de nombres.- On peut également mélanger texte et nombres :

```
import java.awt.*;
public class Calcul_2 extends java.applet.Applet
   {
   public void init()
      {
      resize(100, 150);
      }
   public void paint(Graphics g)
      {
      g.drawString("2 + 3.05 = " + (2 + 3.05), 10, 10);
      }
   }
```

en prenant bien soin du parenthésage.

2.3.3 Les opérations arithmétiques

L'addition, la soustraction, la multiplication et la division ont pour signes respectifs '+', '-', '*' et '/'.

Pour les entiers, l'opération '/' correspond au quotient dans la division euclidienne. Le reste est obtenu, comme en C++, grâce à l'opération '%'.

2.3.4 Les méthodes de la classe `Math`

Les fonctions prédéfinies sur les réels (et quelquefois aussi sur les entiers) se trouvent dans la classe `Math` du paquetage `java.lang`. Il est inutile d'importer explicitement cette classe, le compilateur le fait automatiquement.

Liste de ces fonctions.- Dans le tableau ci-dessous, l'écriture de ce qui est dans la colonne de gauche permet de calculer une valeur (approchée dans le cas des types réels) de ce qui est indiqué dans la colonne de droite :

$abs(x)$	valeur absolue de x,
$acos(x)$	Arc cosinus de x,
$asin(x)$	Arc sinus de x,
$atan(x)$	arc tangente de x,
$cos(x)$	cosinus de x,
$sin(x)$	sinus de x,
$tan(x)$	tangente de x,
$exp(x)$	exponentielle de x,
$log(x)$	logarithme (népérien) de x,
$log10(x)$	logarithme décimal de x,
$pow(x, y)$	x à la puissance y, pour des réels,
$cosh(x)$	cosinus hyperbolique de x,
$sinh(x)$	sinus hyperbolique de x,

$tanh(x)$	tangente hyperbolique de x,
$ceil(x)$	plus petit entier supérieur ou égal à x, considéré comme réel,
$floor(x)$	partie entière de x, considérée comme un réel,
$sqrt(x)$	racine carrée de x (pour *squareroot*),
$max(x, y)$	maximum de x et de y,
$min(x, y)$	minimum de x et de y.

Il existe aussi les constantes `Math.PI` (égale à 3.14159265358979323846) et `Math.E` (égale à 2.7182818284590452354).

Exemple.- Pour calculer $\sin(\pi/6)$, on utilisera l'application :

```
class Sinus
   {
   public static void main(String args[])
      {
      System.out.println(Math.sin(Math.PI/6));
      }
   }
```

2.3.5 Conversion de type

Règle générale.- Il n'y a aucune conversion de type implicite en langage Java. La conversion (explicite, en anglais *cast*) entre types élémentaires se fait, comme en C++, en faisant précéder l'expression du nom du type entre parenthèses.

Exemple.- Pour obtenir la partie entière de π, on utilisera l'application suivante :

```
class Pi
   {
   public static void main(String args[])
      {
      System.out.println((int) Math.PI);
      }
   }
```

2.4 Les chaînes de caractères en Java

Constantes chaîne de caractères.- Nous avons vu que l'on place les constantes chaîne de caractères entre guillements verticaux.

Séquences d'échappement.- Les séquences d'échappement sont les mêmes que celles du C++, à savoir :

l'apostrophe '	doit être écrite \',
le guillemet "	doit être écrit \",
la barre oblique inverse \	doit être écrite \\,

le signe de pourcentage % doit être doublé %%,

\n permet d'aller à la ligne (*newline*),
\t pose une tabulation (horizontale),
\b permet le retour d'un caractère en arrière (*backspace*),
\r provoque un retour chariot sans aller à la ligne (*carriage return*),
\f provoque un saut de page (*form feed*),
\a déclenche un signal sonore (*alarm*).

Code unicode.- Contrairement aux langages C et C++, pour lesquels le codage des caractères n'est pas normalisé, bien que l'on utilise le plus souvent le code ASCII sur les micro-ordinateurs, le Java utilise le code unicode. Chaque caractère est codé sur deux octets.
Pour écrire un caractère unicode on utilise :

`\u####`

où #### est la représentation hexadécimale de la valeur unicode de ce caractère.

Exercice.- *Déterminer le code unicode de 'π' ou, mieux, faites afficher les 500 premiers caractères unicode.*

[On utilise une boucle, dont la syntaxe est la même qu'en C, et une conversion explicite `(char) i`.]

Affichage.- Rappelons l'existence des deux méthodes d'affichage :

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

et :

```
System.out.print()
```

la différence entre les deux étant que la première génère automatiquement un passage à la ligne à la fin de l'affichage.

2.5 Affectation

La syntaxe de l'affectation est la même qu'en langage C.

Variable.- Toute *variable* a un nom, qui est un *identificateur*.

Identificateurs.- En Java, un identificateur est une chaîne de caractères sur l'alphabet unicode, ne contenant pas d'espace, commençant par une lettre, un dollar ou un blanc souligné et n'étant pas un mot réservé.

Voici la liste des mots réservés en Java :

```
abstract     boolean   break      byte      case
catch        char      class      continue  const
default      do        double     else      extends
false        final     finally    float     for
goto         if        implements import    instanceof
int          interface long       native    new
null         package   private    protected public
return       short     static     super     switch
synchronised this      throw      throws    transient
```

```
true          try       void       volatile  while
```

Déclaration d'une variable d'un type élémentaire.- La déclaration d'une variable d'un type élémentaire se fait comme en langage C :

```
type nom ;
```

par exemple :

```
int n ;
```

Initialisation des variables lors de la déclaration.- En Java, comme en langage C, on peut initiliser une variable lors de la déclaration, par exemple :

```
int n = 103 ;
```

Remarques.- 1^o) Contrairement aux langages C et C++, toute variable se voit attribuer une valeur par défaut au moment de sa déclaration. Mais il vaut mieux ne pas s'y fier.

2^o) Rappelons qu'il n'existe aucune conversion implicite de type. La déclaration suivante :

```
float x = 3.05 ;
```

donnera donc lieu à une erreur (alors que le langage C s'en accomode), puisque '3.05' est une constante de type `double`. Il est indispensable d'écrire :

```
float x = 3.05f ;
```

en Java.

Affectation.- La syntaxe de l'affectation simple est, comme en langage C :

```
Identificateur = Expression ;
```

où `Identificateur` est une variable (déclarée) et `Expression` une expression dont le type est le même que celui de `Identificateur`.

Comme en langage C également, on a l'affectation en chaîne, les affectations composées '`+=`', '`-=`', '`*=`', '`/=`' et '`%=`' ainsi que les incrémentations ('`n++`' et '`++n`') et les décrémentations ('`n--`' et '`--n`').

Exemple.- Nous allons écrire plusieurs programmes dont le but est d'évaluer la fonction f, définie par $f(x) = \frac{sin(x)+ln(x)}{exp(x)+2}$, en plusieurs points.

On peut écrire un programme par valeur de x. C'est ce que nous allons faire pour commencer. Pour ne pas avoir à répéter la valeur de x trois fois nous utilisons ici une affectation.

```
class Eval_1
   {
   public static void main(String args[])
      {
      double x, y;
      x = 2;
      y = (Math.sin(x) + Math.log(x))/(Math.exp(x) + 2);
      System.out.println("f(" + x + ") = " + y);
      }
   }
```

2.6 Ordres de lecture

Les ordres de lecture simple sont plus difficiles à mettre en place en Java qu'en langage C, comme nous allons le voir.

2.6.1 Cas d'une application

Avec le kit JDK 1.0 on ne peut lire que caractère par caractère dans une application. Le kit JDK 1.1 (ainsi que les suivants) permettent une lecture de chaîne de caractères, mais de façon un peu compliquée. Le kit JDK 1.2 permet une lecture aisée, mais en ayant recours à une autre fenêtre. Nous ignorerons ici la façon de faire du JDK 1.0.

2.6.1.1 Méthode du kit 1.1

Un exemple.- Écrivons une application Java demandant un prénom et écrivant 'Bonjour' suivi de ce prénom.

```
import java.io.*;

public class Saisie
   {
   public static void main(String args[]) throws IOException
      {
      InputStreamReader isr;
      BufferedReader br;
      String Prenom;
      isr = new InputStreamReader(System.in);
      br = new BufferedReader(isr);
      System.out.print("Quel est votre pr\u00E9nom : ");
      Prenom = br.readLine();
      System.out.println("Bonjour " + Prenom);
      }
   }
```

Commentaires.- 1^o) On doit importer le paquetage 'java.io' (pour *Input/Output*) portant, comme son nom l'indique, sur les entrées-sorties.

2^o) Nous avons déjà dit que Java est un langage plus sécurisé que le langage C. En particulier il traite les **exceptions**, c'est-à-dire les erreurs (prévues) qui peuvent intervenir. Comme on va lire dans un fichier, on doit lui indiquer comment il doit traiter les exceptions engendrées lors des entrées-sorties. Le compilateur exige donc la présence de '`throws IOException`'.

Nous ne verrons pas cependant, pour l'instant, comment récupérer ces exceptions.

3^o) La classe `InputStreamReader`, du paquetage `java.io`, est celle des objets permettant de manipuler des fichiers d'une espèce particulière, des flots (d'octets) en entrée uniquement.

4^o) La classe `BufferedReader`, du paquetage `java.io`, est celle des objets permettant de manipuler des fichiers d'une espèce particulière, des tampons en entrée.

5^o) La classe `String`, du paquetage `java.lang`, est celle des objets chaînes de caractères.

6^o) Nous commençons par déclarer un objet de chacune de ces classes.

7^o) En Java, il est indispensable de créer les objets, on dit aussi d'*instantier les classes*, explicitement grâce à un constructeur. Il ne suffit pas de les déclarer ; c'était déjà le cas des entités dynamiques en C++. La syntaxe est :

```
objet = new classe(paramètres) ;
```

où `objet` est un identificateur, le nom de l'objet tel qu'il a été déclaré, et `classe` est le nom de la classe. Le nombre de paramètres varie suivant les constructeurs.

8^o) Le fichier physique que nous choisissons est le clavier, l'entrée standard, dont le nom est '`System.in`' en Java, d'où l'explication de la façon dont nous avons instantier la classe `InputStreamReader`.

9^o) Le tampon proviendra de ce fichier physique, d'où l'instantiation de la classe `BufferReader`.

10^o) La classe BufferReader contient une méthode, la méthode `readLine()`, permettant de lire une ligne de texte dans le fichier et renvoyant une chaîne de caractères.

2.6.1.2 Méthode du kit JDK 1.2

Depuis le kit 1.2, on peut utiliser une méthode de saisie plus conviviale, mais chaque saisie se fait alors nécessairement dans une fenêtre distincte.

Syntaxe.- On utilise la classe `JOptionPane` du paquetage `javax.swing`, et plus exactement sa méthode statique :

```
String showInputDialog(String)
```

Sémantique.- À l'appel de cette méthode, une nouvelle fenêtre s'ouvre, appelée **boîte de dialogue d'entrée** : elle porte dans le coin supérieur gauche une tasse de café fumant (le logo du navigateur `hotjava` ; le nom de la fenêtre est 'Input' ; il y a à gauche un logo qui est un point d'interrogation ; l'argument chaîne de caractères est écrit et en-dessous on voit une surface d'une autre couleur pour entrer la donnée ; en-dessous apparaissent deux boutons dénommés 'OK' et 'Cancel'.

Lorsqu'on a entré la chaîne de caractères à saisir dans la zone adéquate (après avoir cliqué éventuellement sur la fenêtre avec la souris, pour la rendre active), on clique sur le bouton 'OK' et cette chaîne de caractères est renvoyée par la méthode.

Exemple.- Réécrivons notre programme précédent en utilisant cette méthode :

```
import javax.swing.JOptionPane;

public class Saisie2
   {
   public static void main(String args[])
      {
      String Prenom;
      Prenom = JOptionPane.showInputDialog("Quel est votre pr\u00E9nom : ");
      System.out.println("Bonjour " + Prenom);
      }
   }
```

2.6.2 Cas des applets

Comme dans le cas des applications, la façon de saisir une donnée a variée suivant les versions du kit de développement. Voyons comment adapter notre exemple à une applet.

2.6.2.1 Cas de JDK 1.0

La méthode que nous allons utiliser, valable pour le JDK 1.0, est désavouée ('`deprecated`' en anglais) dès la version 1.1 du JDK : cela signifie qu'elle est encore implémentée mais qu'il n'est pas sûr qu'elle le sera encore dans les versions ultérieures.

Programme source.- On peut utiliser le programme source suivant pour l'applet :

```
import java.applet.Applet;
import java.awt.*;

public class Prenom extends Applet
   {
   Label prompteur;
   TextField entree;
   String Prenom;

   public void init()
      {
      resize(100, 100);
      prompteur = new Label("Entrez votre pr\u00E9nom : ");
      add(prompteur);
      entree = new TextField(10);
      add(entree);
      Prenom = new String();
      }
   public boolean action(Event e, Object o)
      {
      Prenom = entree.getText();
      Prenom = "Bonjour " + Prenom;
      repaint();
      return true;
      }
   public void paint (Graphics g)
      {
      g.drawString(Prenom, 10, 75);
      }
   }
```

Compilation.- On remarquera la présence d'un 'warning' (avertisement) à propos de la méthode désavouée lors de la compilation. La méthode en cause est `action()`.

Exécution.- Lorsqu'on fait appel à cette applet, on voit apparaître dans la fenêtre une ligne avec « `Entrez votre prénom :` » et, en-dessous, un cadre blanc. Lorsqu'on clique dans ce cadre, un curseur apparaît. Vous pouvez alors écrire votre prénom (tout au moins les dix premières lettres). Lorsque vous terminez par un retour chariot, le message voulu s'affiche en-dessous.

Vous pouvez recommencer autant de fois que vous voulez en commençant par cliquer dans le cadre blanc.

2.6.2.2 Champ de texte

La notion essentielle qui nous intéresse ici est celle de champ de texte (`TextField` en anglais).

Notion.- Un champ de texte est la façon de saisir une chaîne de caractères d'au plus une ligne en Java. Nous verrons plus tard `TextArea` pour plus d'une ligne. C'est une classe du package `awt`, à savoir `java.awt.TextField`.

Déclaration.- La déclaration d'un objet de la classe `TextField` se fait comme d'habitude :

```
TextField nom ;
```

où `nom` est un identificateur non utilisé pour autre chose.

Initialisation.- On peut initialiser un champ de texte de quatre façons :

```
nom = new TextField() ;
nom = new TextField(int) ;
nom = new TextField(String) ;
nom = new TextField(String, int) ;
```

où l'entier éventuel est la longueur maximum du texte à saisir et la chaîne de caractères éventuelle un texte par défaut. C'est bien sûr la deuxième façon que nous utiliserons le plus souvent.

Affichage du champ de texte.- Ni la déclaration, ni l'instantiation d'un champ de texte ne suffisent pour le faire afficher. Pour afficher le cadre du champ de texte à un emplacement par défaut il suffit de l'instruction :

```
add(nom) ;
```

Nous verrons plus tard comment faire pour le faire afficher à un endroit que nous voulons dominer.

Récupération du texte.- La récupération de la chaîne de caractères placée dans le champ de texte s'effectue grâce à la méthode `getText()` de la classe `TextField`. Nous avons par exemple ici :

```
Prenom = entree.getText() ;
```

la valeur de retour étant une chaîne de caractères.

2.6.2.3 Étiquette comme prompteur

Introduction.- Nous avons beaucoup insisté, lors de l'initialisation à la programmation, sur le fait que toute demande de saisie doit être précédée d'un prompteur. On peut utiliser pour cela une méthode d'écriture `drawString()` mais cela peut aussi être fait avec une étiquette.

La classe des étiquettes.- La classe des étiquettes est la classe `Label` ('étiquette' en anglais), du paquetage `awt`.
La déclaration se fait de façon habituelle :

```
Label prompteur ;
```

L'initialisation peut se faire par :

```
prompteur = new Label(String) ;
```

où la chaîne de caractères est ce que l'on veut voir apparaître.

On fait apparaître explicitement l'étiquette grâce à la même méthode `add()` que ci-dessus :

```
add(prompteur) ;
```

2.6.2.4 Programmation événementielle

Notion.- Le langage Java suit les règles de la programmation structurée, de la programmation orientée objet mais aussi de la **programmation événementielle**. Dans cette dernière on attend qu'un **événement** se produise (clic de souris à un certain endroit, frappe au clavier...) pour déclencher alors une certaine action.

Implémentation en JDK 1.0.- La mise en place de la programmation événementielle en Java, avec le JDK 1.0, se fait grâce à la surcharge des méthodes `handleEvent()` et `action()`, méthodes de la classe `java.awt.Component`. C'est cette façon de faire qui est désavouée par les versions suivantes de Java.

La méthode `action()`.- L'en-tête de cette méthode est :

```
public boolean action(Event e, Object o)
```

où `e` et `o` nous permettrons plus tard de répondre différemment suivant l'événement qui s'est produit (parmi plusieurs prévisibles).

La valeur de retour est à définir par l'utilisateur qui en fera ce qu'il veut (il n'y a pas d'interprétation par défaut).

La méthode `repaint`.- On aura remarqué l'utilisation de la nouvelle méthode `repaint()` dans le corps de la méthode `action()`. Comme son nom l'indique, elle indique d'appeler à nouveau la méthode `paint()` (qui ne peut être définie qu'une seule fois).

2.6.2.5 Cas du JDK 1.1

Nous verrons plus tard comment la programmation événementielle a évoluée et la méthode qu'il faut utiliser au lieu de la méthode `action()`.

2.6.2.6 Cas du JDK 1.2

On peut utiliser la classe `JOptionPane` comme dans le cas d'une application, comme le montre l'exemple suivant :

```
import java.applet.Applet;
import javax.swing.*;
import java.awt.*;

public class Prenom2 extends Applet
   {
   String prenom;
   public void init()
      {
      prenom = JOptionPane.showInputDialog("Entrez votre pr\u00E9nom : ");
      }
   public void paint(Graphics g)
      {
      g.drawString("Bonjour " + prenom,10 , 75);
```

```
    }
}
```

Remarque.- Lors de l'exécution de l'applet, on remarquera le message dans la boîte de dialogue, indiquant qu'il s'agit d'une fenêtre liée à une applet.

2.7 Conversion de types en chaînes de caractères

Nous avons vu comment lire et afficher des chaînes de caractères. Lorsqu'il s'agit d'une entité d'un autre type, essentiellement d'un type numérique pour l'utilisateur, il faut nécessairement passer par un objet chaîne de caractères, et donc faire des conversions.

Nous avons vu comment convertir les entités de type numérique élémentaire entre elles grâce à la notion de *cast*, déjà présente en langage C. Dans le cas de la conversion d'une entité d'un type élémentaire en une chaîne de caractères, ou vice-versa, il faut passer par des méthodes associées à des classes correspondant aux types élémentaires.

2.7.1 Les classes élémentaires

Notion de classe élémentaire.- À chaque type élémentaire, dont nous avons donné la liste plus haut, correspond une **classe élémentaire** associée, encapsulant le type élémentaire associé.

Liste des classes élémentaires.- Ces classe se trouvent dans le paquetage principal `java.lang`. Les classes élémentaires principales sont les suivantes :

- la classe `Integer` correspondant au type `int` ;
- la classe `Long` correspondant au type `long` ;
- la classe `Float` correspondant au type `float` ;
- la classe `Double` correspondant au type `double`.

Déclaration.- La déclaration d'un objet d'une de ces classes se fait de la façon habituelle, par exemple :

```
String mot ;
Integer N ;
```

Instantiation.- Comme tout objet, ces objets doivent être instantiés avant de pouvoir être utilisés. On dispose pour cela de plusieurs constructeurs, par exemple :

```
N = new Integer(String) ;
```

qui permet d'initialiser par l'entier représenté par la chaîne de caractères (formée de chiffres, bien sûr).

2.7.2 Conversions

Nous avons donc les types élémentaires et les classes élémentaires. Encore faut-il pouvoir passer de l'un à l'autre. De même on a intérêt à pouvoir passer d'une entité à une chaîne de caractères.

2.7.2.1 Conversion entité à objet

Principe.- Chaque classe élémentaire possède un constructeur permettant de passer d'une entité d'un type élémentaire à un objet de la classe élémentaire correspondante.

Exemple.- On pourra créer l'objet de type `Double` dont la valeur encapsulée est `1.06` de la façon suivante :

```
Double X ;
double x ;
x = 1.06 ;
X = new Double(x) ;
```

2.7.2.2 Conversion objet à entité

Principe.- Chaque classe élémentaire possède une méthode permettant d'extraire l'entité du type élémentaire correspondant d'un objet de cette classe.

Exemple.- La méthode `intValue()` de la classe `Integer` (de même, par exemple, la méthode `doubleValue()` de la classe `Double`) permet de convertir un objet de la classe `Integer` (respectivement un objet de la classe `Double`) en une entité du type élémentaire `int` (respectivement `double`).

Ce qui suit est un extrait d'un programme Java :

```
int n ;
Integer N ;
----------
n = N.intValue() ;
```

2.7.2.3 Conversion d'une entité en chaîne de caractères

Principe.- La méthode `valueOf()` de la classe `String` permet de convertir une entité d'un des types élémentaires `boolean`, `int`, `long`, `float`, `double` ou `char` en une chaîne de caractères. Pour un objet, on peut commencer par le transformer en une entité.

Exemple.- On peut avoir ainsi :

```
int i ;
String mot ;
i = 12 ;
mot = new String() ;
mot = mot.valueOf(i) ;
g.drawString(mot, 10, 25) ;
```

2.7.2.4 Conversion d'une chaîne de caractères en un objet

Principe.- Chaque classe élémentaire possède un constructeur permettant de passer d'une chaîne de caractères à un objet de la classe élémentaire correspondante.

Exemple.- On pourra créer l'objet de type `Double` dont la valeur encapsulée est `1.06` de la façon suivante :

```
String mot ;
Double X ;
mot = "1.07" ;
X = new Double(mot) ;
```

2.7.2.5 Conversion d'une chaîne de caractères en une entité

Principe.- Les classes `Integer` et `Double`, entre autres, possèdent les méthodes statiques respectives `parseInt(String)` et `parseDouble(String)`, permettant de convertir une chaîne de caractères en une entité de type `int` ou `double`.

Exemple.- Ceci est une façon (compliquée) d'obtenir le réel `1.06` :

```
double x ;
String mot ;
mot = "1.06" ;
x = Double.parseDouble(mot) ;
```

2.7.3 Un exemple d'application

Écrivons un programme Java permettant de saisir un réel et d'afficher la valeur de $f(x)$ pour notre fonction habituelle f définie par :

$$f(x) = \frac{sin(x) + ln(x)}{e^x + 2}.$$

On a par exemple :

```
import java.io.*;

class Eval_2
   {
   public static void main(String args[]) throws IOException
      {
      InputStreamReader fich;
      BufferedReader ligne;
      String mot;
      double x, y;
      fich = new InputStreamReader(System.in);
      ligne = new BufferedReader(fich);
      System.out.print("x = ");
      mot = ligne.readLine();
      x = Double.parseDouble(mot);
      y = (Math.sin(x) + Math.log(x))/(Math.exp(x) + 2);
      System.out.println("f(" + x + ") = " + y);
      }
   }
```

2.7.4 Un exemple d'applet

Écrivons une applet permettant de saisir des entiers et d'afficher, après chaque saisie d'entier, la somme cumulée.

```
import java.applet.Applet;
import java.awt.*;
```

```
public class Add_3 extends Applet
   {
   Label prompteur;
   TextField entree;
   int n, sum;
   String s;

   public void init()
      {
      prompteur = new Label("Entrez un entier : ");
      add(prompteur);
      entree = new TextField(10);
      add(entree);
      sum = 0;
      }
   public boolean action(Event e, Object o)
      {
      s = entree.getText();
      n = Integer.parseInt(s);
      sum = sum + n;
      repaint();
      return true;
      }
   public void paint(Graphics g)
      {
      g.drawString("" + sum,10,75);
      }
   }
```

Exercice.- *Écrire une applet Java* `Eval_3` *permettant d'effectuer la même chose que notre exemple d'application.*

2.8 Les structures de contrôle

Le langage Java utilise les structures de contrôle du langage C avec la même syntaxe. Rappelons-les rapidement.

2.8.1 Le séquencement

Chaque instruction élémentaire se termine par un point virgule ';'. Une suite d'instructions à effectuer successivement l'une après l'autre, dans l'ordre indiqué, se place entre accolade ouvrante '{' et accolade fermante '}'. Ceci constitue une première instruction composée qu'il est inutile de terminer par un point virgule.

2.8.2 Les conditions

2.8.2.1 Les expressions booléennes

Les booléens.- Nous avons déjà vu que, contrairement aux langages C et C++, le langage Java possède le type élémentaire `boolean`, avec les deux constantes `true` (pour vrai) et `false` (pour

faux).

Les connecteurs logiques.- En langage Java sont implémentés les mêmes trois connecteurs logiques qu'en langage C ou C++. Si `P` et `Q` sont des **expressions booléennes**, c'est-à-dire des expressions dont la valeur est un booléen, on obtient la *négation* de `P`, la *conjonction* de `P` et de `Q` (le *et* logique) et, enfin, la *disjonction* de `P` et de `Q` (le *ou* inclusif logique) en écrivant respectivement :

```
  !P
P && Q
P || Q
```

dont le résultat est un booléen.

2.8.2.2 Relations prédéfinies sur les types numériques élémentaires

Les relations prédéfinies sur les types numériques élémentaires (entiers et réels) en Java sont, comme en langage C, la relation d'égalité, la relation de différence et les relations d'inégalité, notées de la façon suivante :

- `==` pour la relation d'égalité,
- `!=` pour la relation de différence,
- `<` pour strictement plus petit,
- `<=` pour plus petit ou égal,
- `>` pour strictement plus grand,
- `>=` pour plus grand ou égal.

Les deux opérandes doivent être du même type élémentaire. Le résultat d'une opération de comparaison est `true` ou `false`, suivant que la relation est réalisée ou non.

2.8.3 Les itérations

Les trois types de répétition vues en langage C sont implémentées en Java, avec la même syntaxe (à l'existence du type `boolean` près).

Boucle *while*.- La forme de cette instruction est :

```
while (condition) instruction
```

où `instruction` est une instruction (sic), appelée le **corps de la boucle**, et où `condition` est une expression booléenne. Ce qui précède le corps de la boucle est appelé **en-tête de la boucle**.

Boucle *do*.- La forme de cette instruction est :

```
do instruction while (condition) ;
```

où `instruction` est une instruction, encore appelée **corps de la boucle**, et `condition` est une expression booléenne.

Boucle *Pour*.- La forme de cette instruction est :

```
for (init ; condition ; increment) instruction
```

où :

- `init` est une instruction initialisant la (ou les) **variable**(s) **de contrôle** ;
- `condition` est la **condition de terminaison** ;

- `increment` est une instruction permettant de réinitialiser la (ou les) variable(s) de contrôle après le passage dans le **corps de la boucle** `instruction`.

La partie qui précède le corps de la boucle est l'**en-tête de la boucle**.

2.8.4 Le test et l'alternative

Le langage Java reprend la syntaxe du langage C.

Le test.- Cette instruction a la forme suivante :

```
if (condition) instruction
```

où `condition` est une expression booléenne et `instruction` une instruction.

L'alternative.- Cette instruction a la forme suivante :

```
if (condition) instruction1 else instruction2
```

où `condition` est une expression booléenne et `instruction1` et `instruction2` des instructions (préfixées et suffixées éventuellement par { et }, respectivement, pour bien les délimiter si c'est un séquencement).

L'alternative multiple.- Le langage Java reprend également l'alternative multiple :

```
switch(expression)
{
case const_1 : instruction_1 ; break ;
case const_2 : instruction_2 ; break ;
- - - - - - - - - - - - - - - - - -
case const_n : instruction_n ; break ;
}
```

ou :

```
switch(expression)
{
case const_1 : instruction_1 ; break ;
case const_2 : instruction_2 ; break ;
- - - - - - - - - - - - - - - - - -
case const_n : instruction_n ; break ;
default : instruction ;
}
```

où `expression` est une expression d'un des types élémentaires *byte*, *short*, *int* et *char*, où `const_1`, ... , `const_n` sont des constantes, toutes différentes, de ce type et où `instruction_1`, ... , `instruction_n`, `instruction` sont des instructions.

2.8.5 Exemple

Écrivons une application Java permettant d'évaluer notre fonction habituelle en un certain nombre de points, en terminant sur la valeur sentinelle 1000.

```
import java.io.*;

class Eval_4
   {
   public static void main(String args[]) throws IOException
      {
      InputStreamReader fich;
      BufferedReader ligne;
      String mot;
      double x, y;
      fich = new InputStreamReader(System.in);
      ligne = new BufferedReader(fich);
      System.out.print("x = ");
      mot = ligne.readLine();
      x = Double.parseDouble(mot);
      while (x != 1000)
         {
         if (x > 0)
            {
            y = (Math.sin(x) + Math.log(x))/(Math.exp(x) + 2);
            System.out.println("f(" + x + ") = " + y);
            }
         else System.out.println("f(" + x + ") non dfini");
         System.out.print("x = ");
         mot = ligne.readLine();
         x = Double.parseDouble(mot);
         }
      }
   }
```