

Introduction au langage

JAVA

P. Ducrot

Table des matières

GÉNÉRALITÉS	3
SYNTAXE DU LANGAGE.....	11
APPLET.....	34
LE PACKAGE AWT.....	42
GESTION DES ÉVÉNEMENTS	65
LES THREADS	72
LES FLUX	80
LES SOCKETS : CÔTÉ CLIENT	89
LES SOCKETS : CÔTÉ SERVEUR	91
LA CLASSE URL.....	94
LA SÉCURITÉ.....	97

Généralités

Définition de SUN : "Java est un langage simple, orienté objet, distribué, robuste, sûr, indépendant des architectures matérielles, portable, de haute performance, multithread et dynamique"

Simple

Inspiré du C++, Fortran, Lisp, Smalltalk

Pas de pointeur; pas de surcharge d'opérateurs; pas d'héritage multiple

Présence d'un "garbage collector"

Orienté objet

La programmation objet modélise des objets ayant un état (ensemble de variables) et des méthodes (fonctions) qui leur sont propres. L'unité de base en Java est la *classe*. Un des intérêts de Java est de disposer de nombreuses classes déjà faites. Un objet créé à partir d'une classe est une *instance*.

Distribué

Les fonctions d'accès au réseau et les protocoles Internet les plus courants sont intégrés.

Généralités

Robuste

Typage des données très strict

Pas de pointeur

Sûr

Java n'est pas compilé à destination d'un processeur particulier mais en « byte code » qui pourra être ensuite interprété sur une machine virtuelle Java (JVM = Java Virtual Machine). Le "byte code" généré est vérifié par les interpréteurs java avant exécution.

Un débordement de tableau déclenchera automatiquement une exception.

L'absence d'arithmétique de pointeur évite les malversations.

Portable

Les types de données sont indépendants de la plate forme (par exemple les types numériques sont définis indépendamment du type de plate forme sur laquelle le byte code sera interprétée).

Haute performance

Discutable car java est un langage pseudo interprété

Techniques de "Just in Time" (JIT) peuvent améliorer ces performances

Généralités

Multi thread

Une application peut être décomposée en unités d'exécution fonctionnant simultanément

Dynamique

Les classes Java peuvent être modifiées sans avoir à modifier le programme qui les utilise.

Politique

Java est actuellement totalement contrôlé par SUN.

Les différentes version de java

Java 1.0

8 Packages
212 Classes et Interfaces
1545 Méthodes

Java 1.1

23 Packages
504 Classes et Interfaces
3 851 Méthodes

Java 1.2 (Java 2)

60 Packages
1 781 Classes et Interfaces
15 060 Méthodes

Les outils de développement

- **L'outil de base : le JDK (Java Development Kit) de SUN :**
<http://java.sun.com>
 - . **Dernière version : Java 2 SDK 1.3 (comprenant la jdk 1.3)**
 - . **gratuit**
 - . **comprend de nombreux outils :**
 - le compilateur : javac**
 - l'interpréteur d'application : java**
 - l'interpréteur d'applet : appletviewer**
 - le débogueur : jdb**
 - le générateur de documentation : javadoc**
 - etc.**
- **Des environnements de développement commerciaux :**
 - . **Visual J++ (Microsoft)**
 - . **JBuilder (Borland/Inprise)**
 - . **Visual Café (WebGain)**
 - . **CodeWarrior (Metrowerks)**
 - . **etc.**

Généralités

Le langage Java peut générer :

- **des applications**
- **des applets**
- **des servlets**
- **etc.**

Exemple d'application : First.java

```
public class First  
{  
    public static void main (String args [])  
    {  
        System.out.println ("Premier exemple") ;  
    }  
}
```

Cette application s'exécute avec un interpréteur java

java First

Exemple d'applet

Exemple d'applet java : FirstApplet.java

```
public class FirstApplet extends java.applet.Applet  
{  
    public void init ()  
    {  
        System.out.println ("Je passe ici quand la page html est chargée la  
premiere fois");  
    }  
  
    public void start ()  
    {  
        System.out.println ("Je passe ici quand la page html a ete chargee");  
        System.out.println ("puis chaque fois que l'on revient sur cette page");  
    }  
  
    public void stop ()  
    {  
        System.out.println ("Je passe ici quand on quitte la page html");  
    }  
  
    public void destroy ()  
    {  
        System.out.println ("Je passe ici quand l'applet est stoppee");  
    }  
}
```

Utilisation d'une applet

Cette applet s'exécute dans une machine virtuelle proposée par les navigateurs WWW (Netscape, Internet Explorer, ...) et nécessite l'écriture d'un fichier html :

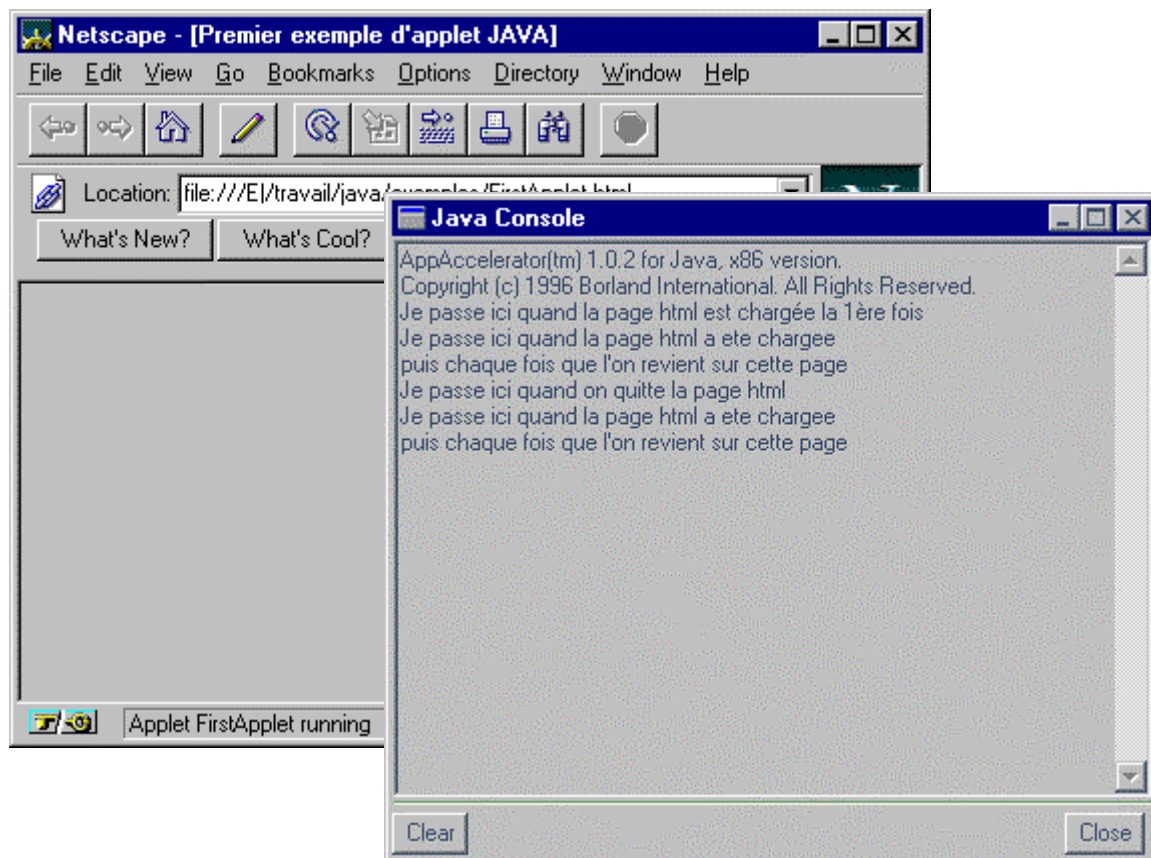
<HTML>

<TITLE> Premier exemple d'applet JAVA </TITLE>

<APPLET code = "FirstApplet.class" WIDTH=200 HEIGHT=200>

</APPLET>

</HTML>



Syntaxe du langage

- Les commentaires existent sous plusieurs formes
- Commentaires sur plusieurs lignes

```
/*  
    */
```

- Commentaires sur une seule ou fraction de ligne

```
//
```

- Commentaires destinés au générateur de documentation

```
/**  
 *  
 *  
 */
```

Exemple : fichier Essai.java

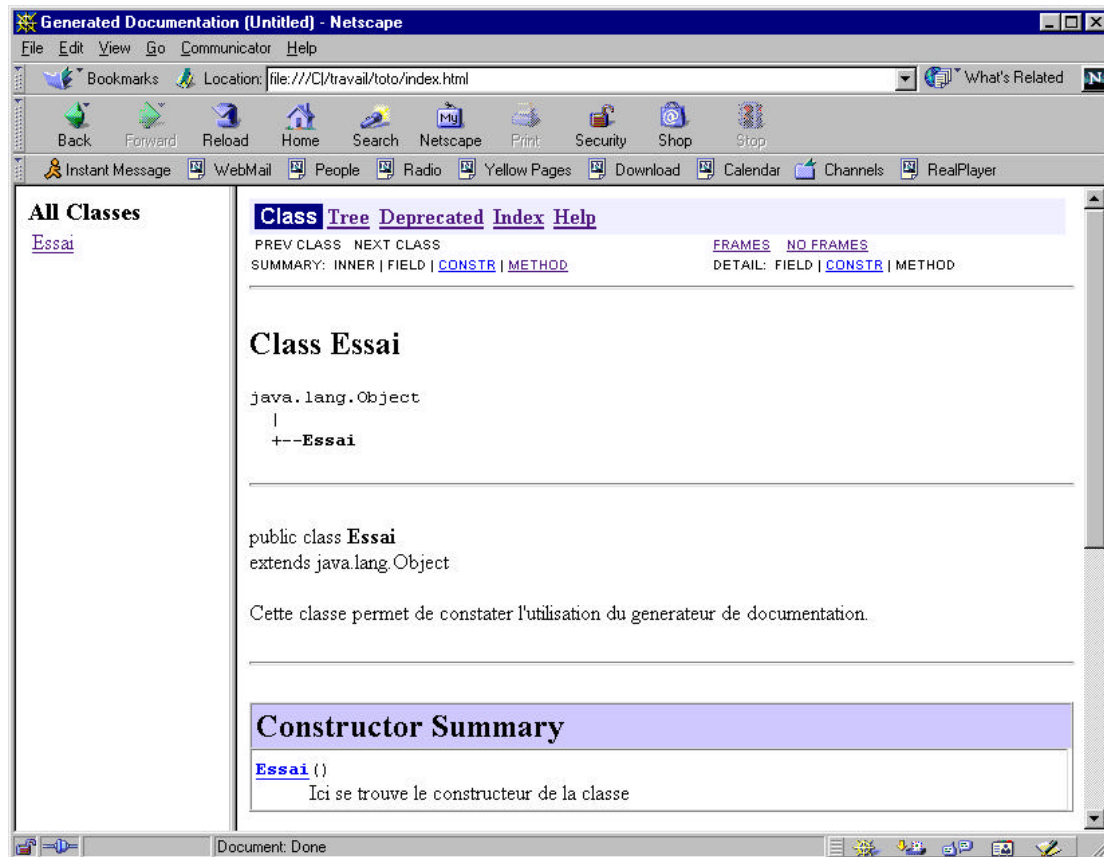
```
/**  
 * Cette classe permet de constater l'utilisation du generateur  
 * de documentation.  
 */
```

```
public class Essai  
{  
    /**  
     * Ici se trouve le constructeur de la classe  
     */  
    public Essai () {}  
    void Methode1 () {}  
    void Methode2 () {}  
}
```

On génère la documentation : *javadoc Essai.java*

Les commentaires

Résultat de la commande javadoc :



Type de données prédéfinis

- Nombres entiers

byte	-2^7 , $(2^7)-1$	-128,127
short	-2^{15} , $(2^{15}) - 1$	-32768,32767
int	-2^{31} , $(2^{31}) - 1$	-2147483648, 2147483647
long	-2^{63} , $(2^{63}) - 1$	9223372036854775808, 9223372036854775807

Les entiers peuvent être exprimés en octal (0323), en décimal (311) ou en hexadécimal (0x137).

- Nombres réels

float	simple précision sur 32 bits 1.4023984 e-45 3.40282347 e38
double	double précision sur 64 bits 4.94065645841243544 e-324 1.79769313486231570 e308

Représentation des réels dans le standard IEEE 754

Un suffixe *f* ou *d* après une valeur numérique permet de spécifier le type.

Exemples :

double x = 145.56d ;
float y = 23.4f ;

Sans suffixe, le réel est assimilé à un double.

float f = 23.65 ; // Erreur

Type de données

- boolean

Valeurs *true* et *false*

Un entier non nul est également assimilé à *true*

Un entier nul est assimilé à *false*

- char

Une variable de type char contient un seul caractère codé sur 16 bits (jeu de caractères 16 bits Unicode contenant 34168 caractères).

Des caractères d'échappement existent :

\b	Backspace
\t	Tabulation horizontale
\n	Line Feed
\f	Form Feed
\r	Carriage Return
\"	Guillemet
\'	Apostrophe
\\	BackSlash
\xdd	Valeur hexadécimale
\ddd	Valeur octale
\u00xx	Caractère Unicode (xx est compris entre 00 et FF)

- Chaînes de caractères

Les chaînes de caractères sont manipulées par la *classe* String.

Exemples :

String str = "exemple de chaîne de caractères" ;

String chaine = "Le soleil " + "brille" ; // Opérateur de concaténation

Les tableaux

Les tableaux sont déclarés suivant les syntaxes suivantes :

```
type [ ] nom ;  
type nom [ ] ;
```

Exemples :

```
int table [ ] ;  
double [ ] d1,d2 ;
```

- Pas de tableau statique.
- La taille d'un tableau est allouée dynamiquement par l'opérateur *new*

```
table = new int [10] ;  
int table2 [ ] = new int [20] ;  
int table3 [ ] = {1,2,3,4,5} ;
```

- La taille n'est pas modifiable et peut être consultée par la propriété *length*

```
System.out.println (table3.length) ;
```

```
int [ ] [ ] Matrice = new int [10][20] ;  
System.out.println (Matrice.length) ; // 1ère dimension  
System.out.println (Matrice[0].length) ; // 2ème dimension
```

Les classes

- Donnée de base de la programmation objet
- Un objet est la réunion au sein d'une même entité de code (fonctions) et de données (variables).
- Les fonctions sont appelées méthodes ou fonctions membres
- Les données sont appelées propriétés ou données membres
- Une classe est la description d'un objet. Chaque objet est créé à partir d'une classe (avec l'opérateur *new*).
- Une classe peut être créée à partir d'une classe existante (héritage).
- Un objet est créé (instancié) à partir d'une classe. Lorsqu'un objet est créé, la mémoire est allouée pour les données membres.
- La création d'un objet retourne une référence sur cet objet qui est typiquement stockée dans une variable
- La référence d'un objet est utilisée pour accéder aux données et fonctions membres de l'objet.
- Un objet peut accéder à sa propre référence grâce à la valeur *this* (variable en lecture seule).
- Une référence contenant la valeur *null* ne désigne aucun objet.
- Quand un objet n'est plus utilisé (aucune variable du programme ne contient une référence sur cet objet), il est automatiquement détruit et la mémoire récupérée (garbage collector)

Exemple de classe

```
class Lampe
{
    private boolean allume ;

    public void changeEtat (boolean etat)
    {
        allume = etat ;           // équivalent à this.allume = etat
    }

    public boolean envoieEtat ()
    {
        return allume ;         // équivalent à return this.allume
    }

    public void afficheEtat ()
    {
        if (allume) System.out.println ("La lampe est allumee ") ;
        else System.out.println ("La lampe est eteinte ") ;
    }
}
```

Utilisation de la classe Lampe : Eclairage.java

```
public class Eclairage
{
    static public main (String args [ ])
    {
        Lampe lampe = new Lampe () ; // lampe contient une référence
        Lampe.changeEtat (true) ;
        Lampe.afficheEtat () ;
    }
}
```

Modularité/Encapsulation

- **Modularité**

- . **Technique de décomposition de systèmes**

- . **Réduire la complexité d'un système par assemblage de sous-systèmes plus simples.**

- . **Les sous systèmes doivent être aussi indépendants les uns des autres que possible.**

- **Encapsulation**

- . **Séparation de l'interface d'un module de son implémentation.**

- . **Interface (partie publique): liste des services offerts.**

- . **Implémentation: (partie privée): réalisation des services (structures de données, instructions, algorithmes,...).**

Constructeur / Destructeur

- Un constructeur est une méthode automatiquement appelée au moment de la création de l'objet.
- Un constructeur est utile pour procéder a toutes les initialisations nécessaires lors de la création de la classe.
- Le constructeur porte le même nom que le nom de la classe et n'a pas de valeur de retour.

Exemple :

```
class Lampe  
{  
    private boolean allume ;  
  
    Lampe ()  
    {  
        allume = false ;  
    }  
    ....  
}
```

- On peut utiliser un destructeur appelé lorsqu'un objet est détruit.
- La présence du garbage collector rend le destructeur peu utile dans la plupart des cas.
- Un destructeur est une méthode **void finalize ()**.

Surcharge de méthodes

- Une méthode (y compris le constructeur) peut être définie plusieurs fois avec le même nom à condition de se différencier par le nombre et/ou le type des paramètres transmis (polymorphisme).
- Le compilateur décidera de la bonne méthode à utiliser en fonction des paramètres d'appel.
- Java ne supporte pas la surcharge des opérateurs (différence avec le C++)

Exemple 1:

```
class BarreDeProgression
{
    float pourcent ;
    ...
    void setPourcent (float valeur) { pourcent = valeur ;}
    void setPourcent (int effectue, int total)
    {
        pourcent = total/effectue ;
    }
    ...
}
```

Exemple 2 :

```
public class Circle {
    public double x, y, r;
    public Circle(double x, double y, double r) {
        this.x = x; this.y = y; this.r = r;
    }
    public Circle(double r) { x = 0.0; y = 0.0; this.r = r; }
    public Circle(Circle c) { x = c.x; y = c.y; r = c.r; }
    public Circle() { x = 0.0; y = 0.0; r = 1.0; }
    public double circumference() { return 2 * 3.14159 * r; }
    public double area() { return 3.14159 * r*r; }
}
```

Comparaison d'objets

- On ne peut comparer 2 objets en comparant les variables d'instance.

Exemple :

```
r1 = new Rectangle (10,20) ;  
r2 = new rectangle (30,40) ;  
r3 = new Rectangle (10,20) ;
```

Comparaison des variables d'instance:

```
r1 == r2            → false  
r1 == r3            → false
```

Comparaison avec une méthode equals incluse dans la classe Rectangle

```
r1.equals (r2)      → false  
r1.equals (r3)      → true
```

Duplication d'objets

- La duplication d'objet peut se faire par une copie de surface ou par une copie profonde.

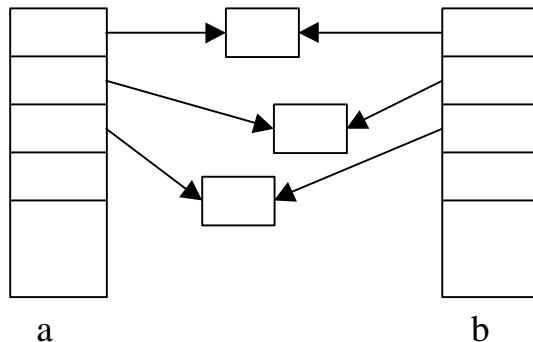
Exemple :

```
class ListeRectangles  
{  
    Rectangle rectangles [ ] ;  
    int nbRectangles ;  
    ...  
}
```

ListeRectangles a,b ;

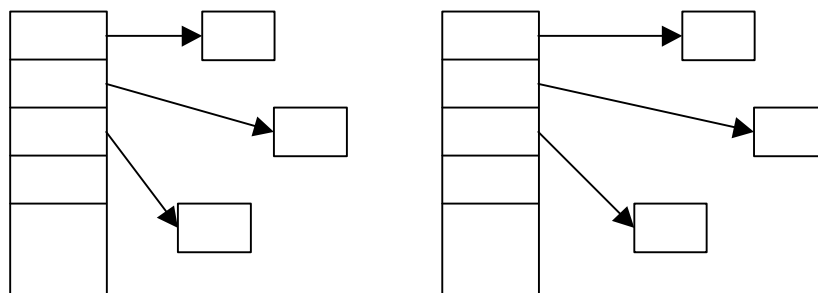
Copie de surface :

```
for (int i = 0 ; i < nbRectangles ; i++)  
    a.rectangles [i] = b.rectangles [i] ;
```



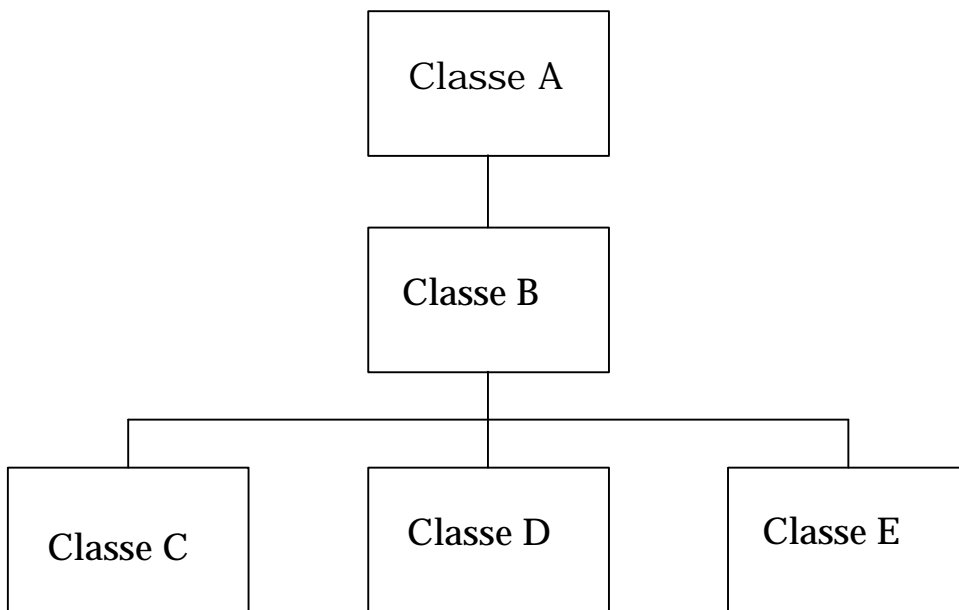
Copie profonde:

```
for (int i = 0 ; i < nbRectangles ; i++)  
    a.rectangles [i] = b.rectangles [i].clone () ;
```



Héritage

- **Concept très important dans la programmation objet.**
- **Une classe peut hériter d'une autre classe et apporter ses propres spécificités.**



- **Le mot clé pour déclarer une classe dérivée est *extends*.**
- **Java ne supporte que l'héritage simple.**
- **Une méthode peut être préfixée par *abstract*. Dans ce cas, la classe est abstraite. Aucun objet ne peut être instancié d'une classe abstraite et les classes dérivées devront définir complètement les méthodes abstraites.**
- **Le mot clé *final* interdit la dérivation d'une classe (par exemple, la classe String est *final*) ; appliqué à une variable, celle ci ne peut pas être modifiée (constante) ; appliqué à une méthode, celle ci ne peut pas être surchargée.**

Exemple d'héritage

```
// Animal.java
public class Animal
{
    public int poids;
    public void dormir () { System.out.println ("Méthode dormir de Animal"); }
    public void jouer () { System.out.println ("Méthode jouer de Animal"); }
    public void seReproduire () { System.out.println ("Méthode sereproduire de Animal"); }
}

// Mammifere.java
public class Mammifere extends Animal
{
    public void seReproduire () { System.out.println ("Méthode seReproduire de Mammifère"); }
}

// Chat.java
public class Chat extends Mammifere
{
    public void jouer () { System.out.println ("Méthode jouer de Chat"); }
    public void miauler () { System.out.println ("Méthode miauler de Chat"); }
}

// RunChat.java
import Chat;

public class RunChat
{
    public static void main ( String []arguments)
    {
        Chat minet = new Chat();

        minet.dormir();
        minet.seReproduire();
        minet.jouer();
    }
}
```

L'exécution de RunChat donnera :

Méthode dormir de Animal
Méthode seReproduire de Mammifère
Méthode jouer de Chat

Exemple extrait du cours de Stéphane Bortzmeyer, bortzmeyer@pasteur.fr

Héritage

- Il est possible d'accéder aux données/méthodes de la classe de base grâce au mot clé *super*.

Exemple 1 :

```
class MaFrame extends Frame
{
    // Constructeur
    MaFrame ()
    {
        super () ; // Appel du constructeur de Frame
                // Si cet appel est utilisé, c'est toujours
                // la première instruction du constructeur
        ...
    }
}
```

Exemple 2 :

```
class HouseCat extends Feline
{
    void speak ()
    {
        System.out.println ("Meow !!!") ;
    }
}
class MagicCat extends HouseCat
{
    boolean people_present ;
    ...
    void speak ()
    {
        if (people_present) super.speak () ;
        else System.out.println ("Hello") ;
    }
}
```

Interface

- Les interfaces compensent un peu l'absence d'héritage multiple.
- Une interface est comme une classe abstraite mais sans aucun code de méthodes.
- Une interface peut contenir des variables constantes et des entêtes de méthodes.
- Le mot clé pour implémenter une interface est *implements*.
- Une classe implémentant une interface s'engage à surcharger toutes les méthodes définies dans cette interface (contrat).
- Une interface permet donc d'imposer un comportement à une classe
- Une classe peut implémenter autant interfaces qu'elle le souhaite.

Exemple :

```
Interface Drawable
{
    void drawMe (int x, int y) ;
}

class GraphicObject implements Drawable
{
    ...
    void drawMe (int x,int y)
    {
        // Code de la fonction drawMe
    }
    ...
}
```

Packages

- Les packages sont des collections de classes regroupées par thème.
- Une application peut utiliser des classes prédéfinies par importation du package concerné.
- Des packages de base sont :

<i>java.lang</i>	Principales classes du langage Java
<i>java.io</i>	E/S vers différents périphériques
<i>java.util</i>	Utilitaires (vecteur, hashtables, ...)
<i>java.net</i>	Support du réseau (socket, URL, ...)
<i>java.awt</i>	Interface graphique
<i>java.applet</i>	Classes de base pour la réalisation d'une applet

- De nouveaux packages peuvent être définis rendant le langage très extensible (exemple : package java3d pour des classes graphiques en 3 dimensions).
- Exemple d'utilisation

```
import java.lang.* ; // Ce package est implicite  
import java.awt.* ;
```

- Exemple de création d'un nouveau package :

```
Package MonPackage ;
```

```
public class MaClasse  
{  
    void test ()  
    {  
        System.out.println ("test ") ;  
    }  
}
```

Droits d'accès

- **Toutes les méthodes et variables définies au sein d'une classe sont utilisables par la classe.**
- **Lors de la conception d'une classe, il faut décider des méthodes/variables/classes qui seront visibles à l'extérieur de cette classe.**
- **Java implémente la protection des 4 P (public, package, protected, private).**
- ***public* : visible partout**
- **Le droit par défaut est une visibilité des classes/données/membres pour toutes les classes au sein d'un même package. Il n'y a hélas pas de mot clé pour préciser explicitement cet accès.**
- ***protected* : visible uniquement dans la classe et dans les classes dérivées de cette classe.**
- ***private* : visible uniquement au sein de la classe.**

Variables de classe

- Les variables de classe sont préfixées par le mot clé *static*.
- Une variable *static* est partagée par toutes les instances de la classe.

Exemple 1:

```
class Alien
{
    static int counter ;

    Alien ()
    {
        counter += 1 ;
    }
    ...
}
```

Exemple 2:

La classe Math contient la valeur de PI

```
class Math
{
    public static final double PI ;
    ...
}
```

Utilisation: *Math.PI*

Méthodes de classe

- Les méthodes de classe sont préfixées par le mot clé *static*.
- Une méthode de classe est appellable sans avoir à créer d'objet de cette classe.

Exemple 1 :

La classe **Math** contient beaucoup de méthodes *static*

```
class Math
{
    ...
    public static int max( int a, int b ) ;
    public static double sqrt( double a ) ;
    public static double sin( double a ) ;
    ...
}
```

Utilisation: ***Math.sqrt (3.678) ;***

Exemple 2 :

```
class MathStuff
{
    static int HalfInt (int x)
    {
        return x/2 ;
    }
}
```

MathStuff.HalfInt (10) ;

Conversions de type

Les méthodes de conversion de type sont stockées dans des classes :

```
class String  
class Integer  
class Long  
class Float  
class Double
```

Exemples :

```
int i = 10 ;  
String chaine = String.valueOf (i) ;  
  
String chaine="123" ;  
int entier = Integer.parseInt (chaine) ;
```

```
class Integer  
{  
.....  
    public static int parseInt(String s) throws NumberFormatException;  
.....  
};  
  
try {  
    entier = Integer.parseInt (chaine) ;  
} catch (NumberFormatException e)  
    {  
        // Si la variable chaine n'est pas convertible on vient ici  
    }
```

Les exceptions

- Le traitement des exceptions permet à une application d'avoir un comportement adéquat à une situation inattendue.
- Beaucoup de méthodes sont susceptibles de déclencher une exception (comme par exemple l'exemple précédent).
- Une exception peut être levée par une séquence try/catch :

```
String chaine ;  
int valeur ;  
...  
try {  
    valeur = Integer.parseInt (chaine) ;  
} catch (NumberFormatException e)  
{  
    System.out.println ("mauvais format ") ;  
}
```

```
try a = tab [i] ;  
catch (ArrayIndexOutOfBoundsException ex) a = 0 ;
```

- Structure :

```
try <instruction> ;  
catch (<type exception1> ex1) instruction ;  
catch (<type exception2> ex2) instruction ;  
....  
finally <instruction> ;
```


Exceptions

- Une méthode pouvant produire une exception doit être appelée dans un bloc try/catch.

- On indique qu'une méthode *m* peut générer une exception *E* :

```
void m () throws E  
{  
    ...  
}
```

- Le concepteur d'un programme java a un devoir de captage des exceptions pour ne pas désemparer les utilisateurs.
- Il faut cependant éviter de capter les erreurs non récupérables (par exemple *NullPointerException*); les exceptions doivent rester exceptionnelles ;-)

Applet

- Pas de fonction "main"
- s'exécute dans une machine virtuelle (Navigateurs, ...)
- Utilise la classe *Applet* du package *applet*
- Les fonctions les plus importantes de cette classe :
init, start, stop, destroy (vues dans les généralités)

```
public void paint (Graphics g)  
{  
    // Appelée à l'initialisation puis chaque fois  
    // que l'affichage doit être rafraîchi  
}
```

Exemple d'Applet

```
import java.awt.Graphics ;  
import java.awt.Font ;  
import java.awt.Color ;  
//  
// On aurait pu faire un seul import : import java.awt.* ;  
//  
import java.applet.* ;  
  
public class Hello extends Applet  
{  
    Font f = new Font ("Serif", Font.BOLD,36) ;  
  
    public void paint (Graphics g)  
    {  
        g.setFont (f) ;  
        g.setColor (Color.red) ;  
        g.drawString (getParameter ("Chaine"),10,30) ;  
    }  
}
```



La balise APPLET

L'applet nécessite un fichier HTML contenant une balise <APPLET> pour être exécutée.

Exemple :

```
<HTML>  
<TITLE> Applet Hello </TITLE>  
  
  <APPLET CODE = "Hello.class" WIDTH=200 HEIGHT=200>  
    <PARAM NAME=Chaine VALUE="Hello World">  
  </APPLET>  
  
</HTML>
```

- **CODE** indique le nom du fichier qui chargera l'applet
- **WIDTH, HEIGHT** Taille nécessaire pour l'applet en pixels dans la fenêtre du navigateur

Ces 3 attributs sont indispensables.

La balise APPLET

D'autres attributs pour la balise <APPLET> existent :

- **ALIGN** Définit comment l'applet doit être alignée sur la Page
Valeurs possibles : **LEFT,RIGHT,TOP,TEXTTOP, MIDDLE,ABSMIDDLE,BASELINE,BOTTOM,ABSBOTTOM**

```
<APPLET CODE="Hello.class" ALIGN=LEFT WIDTH=200 HEIGHT=50>  
</APPLET>
```

A la gauche de ce texte, vous pouvez voir une applet affichant le texte "Hello World" en rouge.

```
<BR CLEAR=ALL>
```

Ce texte s'affiche désormais en dessous de l'applet et a gauche de la fenêtre du navigateur.



- **CODEBASE** Permet de définir le répertoire ou le serveur contenant l'applet:

```
<APPLET CODE="Hello.class" CODEBASE="http://www.hello.com" WIDTH=200 HEIGHT=50>
```

- **PARAM** Permet de définir des paramètres dans le fichier HTML qui seront récupérés dans l'Applet grâce à la méthode `getParameter()` (méthode de la classe `applet`).

```
<PARAM NAME=font VALUE="Serif">
```

```
String FontName = GetParameter ("font") ;
```

Gestion affichage

- Géré par un thread "Screen Updater"
- Appelle la méthode *update ()* des composants qui doivent être redessiné
- Un composant graphique peut demander à être redessiné en appelant la méthode *repaint ()*
 - Positionne une variable dans le composant à destination du screen updater qui appellera la méthode *update ()*
- Fonctionnement par défaut de la méthode *update* :

```
public void update (Graphics g)  
{  
    g.setColor (getBackground ()) ;  
    g.fillRect (0,0,getSize ().width, getSize().height) ;  
    g.setColor (getForeground ()) ;  
    paint (g) ;  
}
```

La classe graphique

- Les fonctions graphiques sont utilisables à travers un objet "Graphics"
- L'objet Graphics gère un contexte graphique (même notion que sous X-Window, comparable au contexte de périphérique sous Windows)
- L'objet Graphics est transmis en argument des fonctions *update()* ou *paint ()* et peut être créé par les méthodes *getGraphics ()* ou *create ()*
- Un objet Graphics manipule une surface spécifique de l'application
- Une surface peut être manipulée par plusieurs objets Graphics
- La classe Graphics contient les fonctions classiques de gestion de tracés de formes, de remplissage, d'utilisation des couleurs et de fontes, ...

Exemples:

```
public void drawLine( int x1, int y1, int x2, int y2 )  
public void drawPolygon( int xPoints[], int yPoints[], int nPoints )  
public void drawRect( int x, int y, int width, int height )  
public void fillOval( int x, int y, int width, int height )  
public void fillRect( int x, int y, int width, int height )
```

La classe Color

- La gestion des couleurs est basée sur un modèle à 24 bits
- Une couleur est définie par ses composantes RGB

Exemple :

```
Color BleuPale = new Color (0,0,80) ;  
g.setColor (BleuPale) ;
```

- Certaines couleurs ont un nom symbolique (membres statiques)

Exemples :

<i>Color.white</i>	<i>255,255,255</i>
<i>Color.red</i>	<i>255,0,0</i>
<i>Color.orange</i>	<i>255,200,0</i>

Quelques méthodes de gestion de couleurs héritées de la classe *Component*:

```
public void setBackground(Color c )  
public Color getBackground()
```

ou de la classe *Graphics*:

```
public void setColor (Color c )  
public Color getColor()
```


Les polices de caractères

- La classe *Font* permet de définir des objets fontes caractérisés par leur nom, leur style et leur taille en points

Exemple :

Font f = new Font ("Monospaced", font.BOLD, 24) ;

- Java fournit 5 fontes universelles

Dialog

SansSerif (anciennement Helvetica)

Serif (anciennement TimesRoman)

Monospaced (anciennement Courier)

Symbol

- D'autres fontes peuvent être disponibles (dépendant de la plateforme utilisée)

```
import java.applet.* ;  
import java.awt.* ;  
public class Fontes extends Applet  
{  
    public void paint (Graphics g)  
    {  
        String FontListe [] ;  
        FontListe = getToolkit().getFontList () ;  
  
        for (int i = 0 ; i != FontListe.length ; i++)  
        {  
            g.setFont (new Font (FontListe[i],Font.PLAIN,12)) ;  
            g.drawString (FontListe [i],0,20*(i+1)) ;  
        }  
    }  
}
```

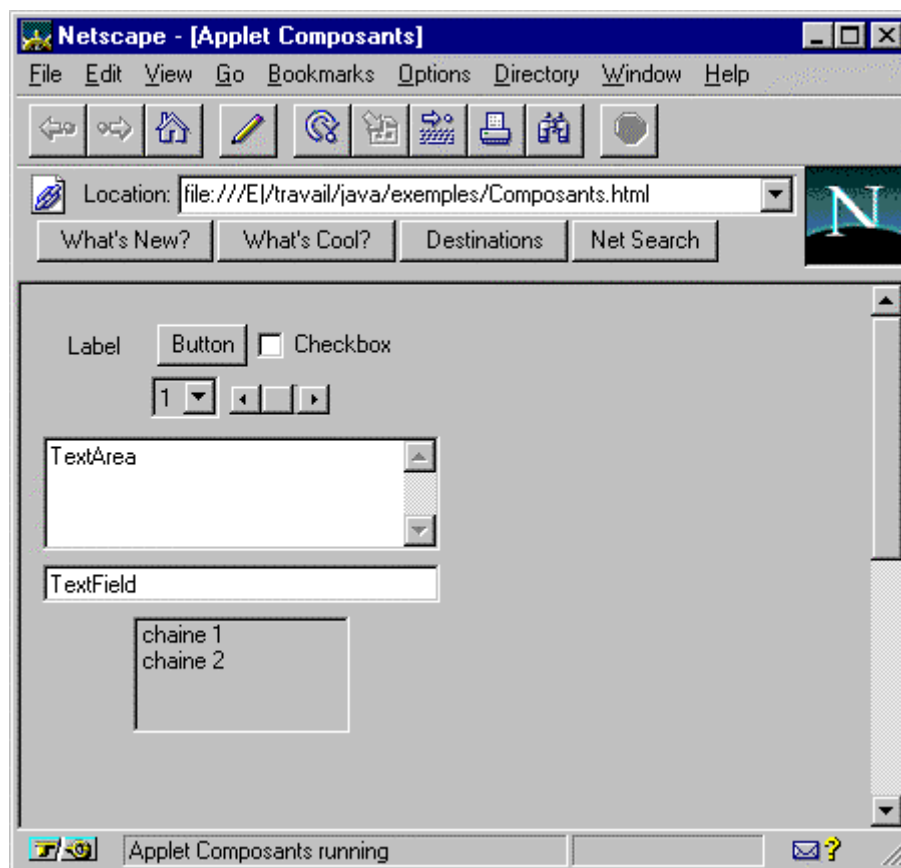
- Styles disponibles: **Font.BOLD**
 Font.ITALIC
 Font.PLAIN

Le package AWT

- **Java propose des composants graphiques permettant de concevoir facilement une interface utilisateur**

Button
CheckBox
Choice
Label
List
Scrollbar
TextArea
TextField

- **Java 1.2 apporte la classe Swing permettant d'enrichir considérablement l'aspect des interfaces**



Interface utilisateur

```
import java.applet.* ;  
import java.awt.* ;  
public class Composants extends Applet {  
    Label label ;  
    Button button ;  
    Checkbox checkbox ;  
    Choice choice ;  
    Scrollbar h_scrollbar ;  
    TextArea textarea ;  
    TextField textfield ;  
    List list ;  
  
    public void init ()  
    {  
        label = new Label ("Label") ; add (label) ;  
  
        button = new Button ("Button") ; add (button) ;  
  
        checkbox = new Checkbox ("Checkbox") ; add (checkbox) ;  
  
        choice = new Choice () ;  
        choice.addItem ("1") ;  
        choice.addItem ("2") ;  
        add (choice) ;  
  
        h_scrollbar = new Scrollbar(Scrollbar.HORIZONTAL,50,10,0,1000) ;  
        add (h_scrollbar) ;  
  
        textarea = new TextArea ("TextArea",3,30) ; add (textarea) ;  
  
        textfield = new TextField ("TextField",30) ; add (textfield) ;  
  
        list = new List () ;  
        list.addItem ("chaine 1") ; list.addItem ("chaine 2") ;  
        add (list) ;  
        setSize (300,300) ;  
    }  
}
```

Mise en Page

- Les objets awt apparaissent dans l'ordre de création
- La fenêtre du navigateur sert de container
- On utilise des classes de mise en page :

FlowLayout (layout par défaut)

BorderLayout

CardLayout

GridLayout

GridBagLayout

FlowLayout

public FlowLayout () ;

Les objets sont centrés dans l'espace qui leur est réservé

public FlowLayout (int align) ;

align = FlowLayout.LEFT

align = FlowLayout.CENTER

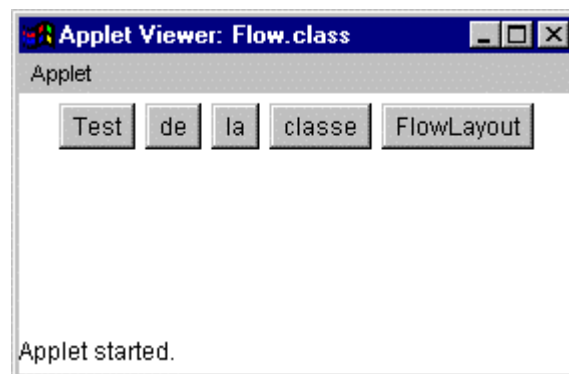
align = FlowLayout.RIGHT

public FlowLayout (int align, int hgap, int vgap) ;

hgap et vgap correspondent à l'espacement horizontal et vertical des objets en pixels.

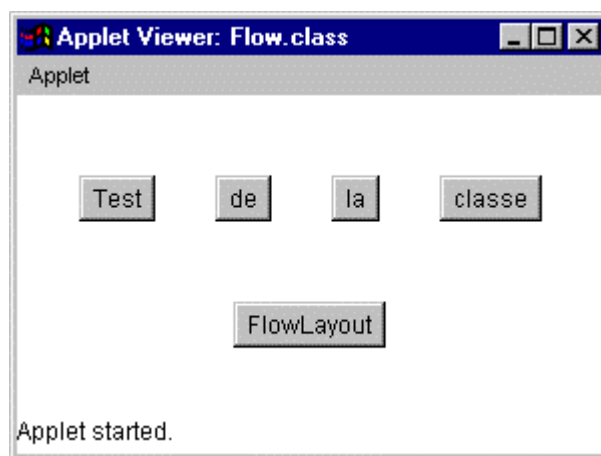
Exemple de FlowLayout

```
//*****  
// Flow.java: Applet  
//  
//*****  
  
import java.applet.*;  
import java.awt.*;  
  
public class Flow extends Applet  
{  
    public void init()  
    {  
        setSize(320, 240);  
        setLayout (new FlowLayout ());  
        add (new Button ("Test"));  
        add (new Button ("de"));  
        add (new Button ("la"));  
        add (new Button ("classe"));  
        add (new Button ("FlowLayout"));  
    }  
}
```



Exemple de FlowLayout

```
//*****  
// Flow.java:      Applet  
//  
//*****  
  
import java.applet.*;  
import java.awt.*;  
  
public class Flow extends Applet  
{  
    public void init()  
    {  
        setSize(320, 240);  
        setLayout (new FlowLayout (FlowLayout.CENTER,30,40)) ;  
        add (new Button ("Test")) ;  
        add (new Button ("de")) ;  
        add (new Button ("la")) ;  
        add (new Button ("classe")) ;  
        add (new Button ("FlowLayout")) ;  
    }  
}
```



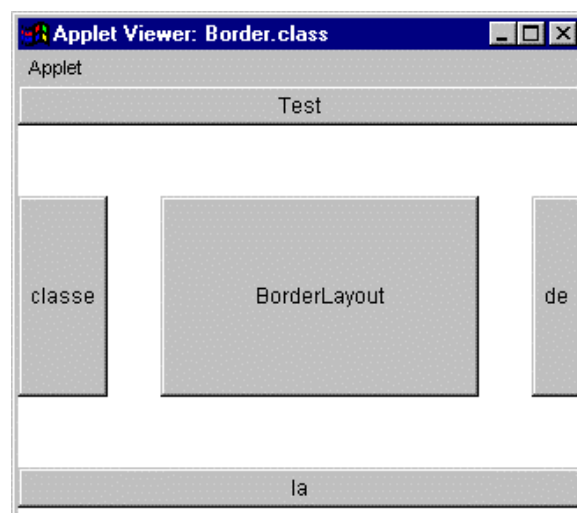
BorderLayout

On définit la mise en page des objets en indiquant la position géographique :

North, South, East, West, Center

public BorderLayout () ;
public BorderLayout (int hgap, int vgap) ;

```
//*****  
// Border.java: Applet  
//  
//*****  
import java.applet.*;  
import java.awt.*;  
  
public class Border extends Applet  
{  
    public void init()  
    {  
        setSize(320, 240);  
        setLayout (new BorderLayout (30,40)) ;  
        add ("North",new Button ("Test")) ;  
        add ("East",new Button ("de")) ;  
        add ("South",new Button ("la")) ;  
        add ("West",new Button ("classe")) ;  
        add ("Center",new Button ("BorderLayout")) ;  
    }  
}
```



CardLayout

Définit des objets qui ne sont pas visibles simultanément mais consécutivement.

```
public CardLayout () ;  
public CardLayout (int hgap, int vgap) ;
```

Quelques méthodes pour passer d'un composant à un autre :

<i>first ()</i>	Affiche le premier composant
<i>last ()</i>	Affiche le dernier composant
<i>previous ()</i>	Affiche le composant précédent
<i>next ()</i>	Affiche le composant suivant
<i>show ()</i>	Affiche le composant spécifié dans le 2^{ème} argument

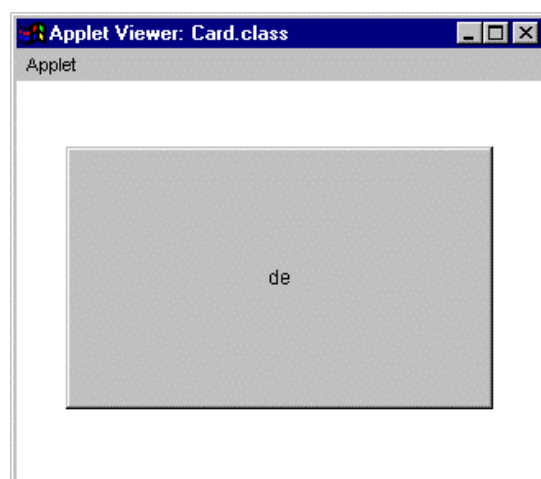
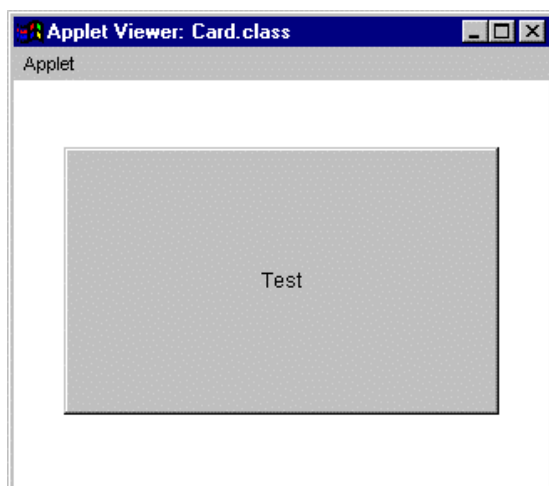
Exemple de CardLayout

```
//*****
// Card.java: Applet
//
//*****
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Hello extends Applet implements ActionListener
{
    CardLayout cl      = new CardLayout (30,40) ;
    Button Test        = new Button ("Test") ;
    Button De           = new Button ("de") ;
    Button La           = new Button ("la") ;
    Button Classe       = new Button ("classe") ;
    Button Cardlayout   = new Button ("CardLayout") ;

    public void init()
    {
        setLayout (cl) ;
        add ("Test",Test) ; Test.addActionListener (this) ;
        add ("de",De) ;      De.addActionListener (this) ;
        add ("la",La) ; La.addActionListener (this) ;
        add ("classe",Classe) ; Classe.addActionListener (this) ;
        add ("CardLayout",Cardlayout);Cardlayout.addActionListener (this) ;
        setSize(320, 240);
    }

    public void actionPerformed (ActionEvent evt)
    {
        cl.next (this) ;
    }
}
```



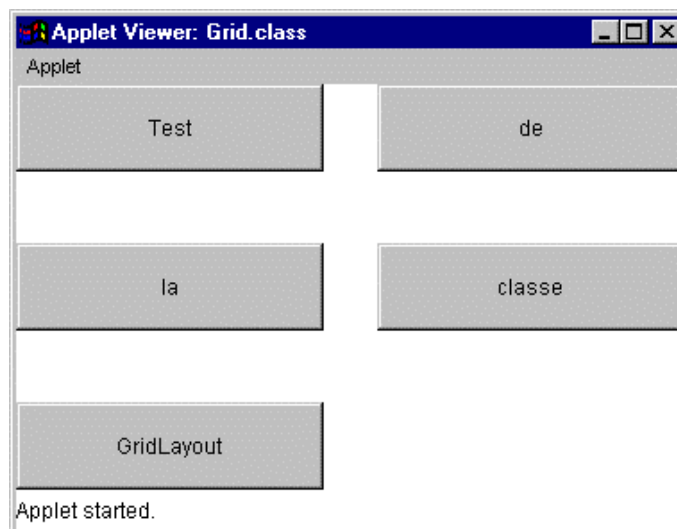
GridLayout

Définit un quadrillage dans lequel sont placés les objets qui seront placés de la gauche vers la droite et du haut vers le bas.

public GridLayout (int rows, int cols) ;
public GridLayout (int rows, int cols, int hgap, int vgap) ;

```
/**
// Grid.java:  Applet
//
//
import java.applet.*;
import java.awt.*;

public class Grid extends Applet
{
    public void init()
    {
        setSize(320, 240);
        setLayout (new GridLayout (3,2,30,40)) ;
        add (new Button ("Test")) ;
        add (new Button ("de")) ;
        add (new Button ("la")) ;
        add (new Button ("classe")) ;
        add (new Button ("GridLayout")) ;
    }
}
```



GridBagLayout

Définit un quadrillage analogue à GridLayout mais les composants n'ont pas forcément une taille identique et peuvent occuper une ou plusieurs cases de la grille.

public GridBagLayout () ;

Cette mise en forme s'utilise avec une ou plusieurs variables de type ***GridBagConstraints*** dont les principaux champs sont :

<i>public int gridx ; public int gridy ;</i>	Définissent les coordonnées de la cellule dans la partie supérieure gauche de la zone d'affichage. La valeur par défaut est <i>GridBagConstraints.RELATIVE</i>
<i>public int gridwidth; public int gridheight;</i>	Nombre de cellules en colonnes et en ligne du composant courant. La valeur par défaut est 1.
<i>public int fill ;</i>	Détermine comment utiliser l'espace libre disponible lorsque la taille du composant ne correspond pas à celle qui est offerte. <i>GridBagConstraints.NONE</i> <i>GridBagConstraints.HORIZONTAL</i> <i>GridBagConstraints.VERTICAL</i> <i>GridBagConstraints.BOTH</i>

GridBagLayout

<i>public int ipadx ; public int ipady ;</i>	Définit la taille horizontale et verticale (internal padding) à ajouter aux composants si la valeur <i>fill</i> n'est pas spécifiée.
<i>public Insets insets</i>	Définit l'espacement autour du composant (external padding). La classe <i>Insets</i> est défini par : <i>public Insets (int top,int left, int bottom,int right)</i>
<i>public int anchor</i>	Positionne le composant lorsque la taille de la cellule est plus grande que la taille du composant. Valeurs possibles : <i>NORTH, NORTHWEST, NORTHEAST, SOUTH, SOUTHWEST, SOUTHEAST, WEST, EAST</i>
<i>public double weightx public double weighty</i>	Définit la répartition de l'espace en cas de changement de dimension (en proportion)

Mise en œuvre :

- Création d'un objet GridBagLayout
- Création d'un objet GridBagConstraints
- Fixation des contraintes d'un composant
- Enregistrement des contraintes auprès du gestionnaire
- Ajout du composant

Exemple de GridBagLayout

```

//*****
// GridBag.java: Applet
//
//*****

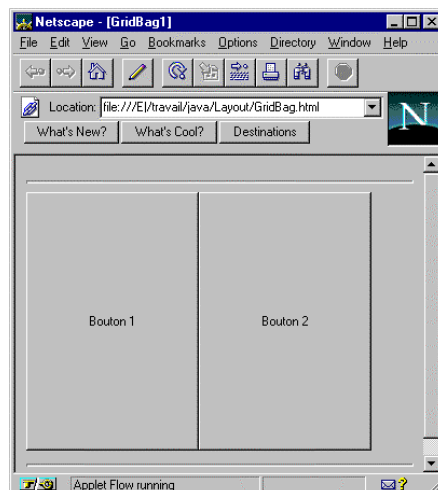
import java.applet.*;
import java.awt.*;

public class GridBag1 extends Applet
{
    public void init()
    {
        Button b1 = new Button ("Bouton 1");
        Button b2 = new Button ("Bouton 2");

        setSize (200,200) ;
        GridBagLayout gbl = new GridBagLayout () ;
        GridBagConstraints gbc = new GridBagConstraints () ;

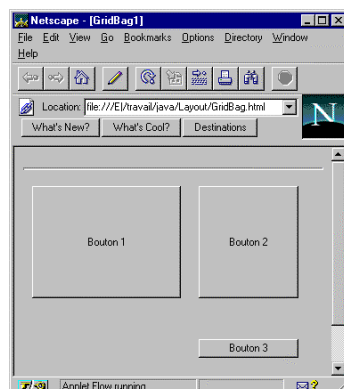
        setLayout (gbl) ;
        gbc.fill = GridBagConstraints.BOTH ;
        gbc.weightx = 1 ; gbc.weighty = 1 ;
        gbl.setConstraints (b1,gbc) ; add (b1) ;
        gbl.setConstraints (b2,gbc) ; add (b2) ;
    }
}

```



Exemple de GridBagLayout

```
//*****  
// GridBag.java:    Applet  
//  
//*****  
import java.applet.*;  
import java.awt.*;  
  
public class GridBag1 extends Applet  
{  
    public void init()  
    {  
        Button b1 = new Button ("Bouton 1") ;  
        Button b2 = new Button ("Bouton 2") ;  
        Button b3 = new Button ("Bouton 3") ;  
  
        setSize (200,200) ;  
  
        GridBagLayout gbl = new GridBagLayout () ;  
        GridBagConstraints gbc = new GridBagConstraints () ;  
  
        setLayout (gbl) ;  
        gbc.insets = new Insets (10,10,10,10) ;  
        gbc.fill = GridBagConstraints.BOTH ;  
        gbc.weightx = 2 ; gbc.weighty = 2 ;  
  
        gbl.setConstraints (b1,gbc) ; add (b1) ;  
        gbc.weightx = 1 ; gbc.weighty = 1 ;  
        gbl.setConstraints (b2,gbc) ; add (b2) ;  
  
        gbc.gridx = 1 ; gbc.gridy = 1 ;  
        gbc.fill = GridBagConstraints.HORIZONTAL ;  
        gbl.setConstraints (b3,gbc) ; add (b3) ;  
    }  
}
```



Les conteneurs

<i>Panel</i>	Conteneur sans fenêtre propre. Permet d'ordonner les composants graphiques.
<i>Window</i>	Fenêtre principale sans cadre ni menu. Les objets de cette classe peuvent servir à implémenter des menus
<i>Frame</i>	Fenêtre possédant toutes les fonctionnalités (barre de titre, barre de menus, etc.)
<i>Dialog</i>	Permet de réaliser des boîtes de dialogue. Nécessite une frame

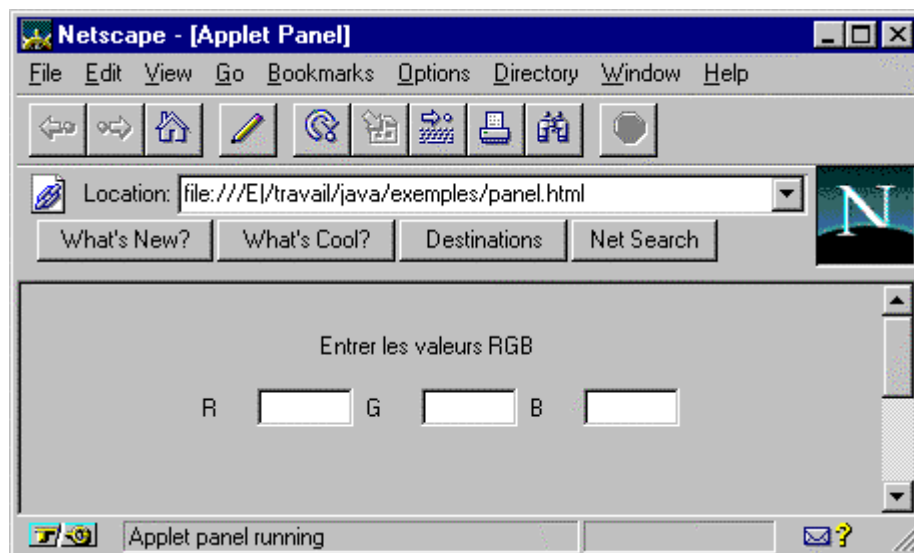
Panel

- Container sans fenêtre propre
- Il définit son propre layout (FlowLayout par défaut)
- Il peut aider à la mise en place de l'interface

Exemple :

```
import java.applet.* ;  
import java.awt.* ;  
  
public class panel extends Applet  
{  
    public void init ()  
    {  
        setLayout (new BorderLayout ()) ;  
        Panel Haut = new Panel () ;  
        Panel Milieu = new Panel () ;  
  
        Haut.setLayout (new FlowLayout ()) ;  
        Haut.add (new Label ("Entrer les valeurs RGB")) ;  
  
        Milieu.setLayout (new FlowLayout ()) ;  
        Milieu.add (new Label ("R")) ;  
        Milieu.add (new TextField (5)) ;  
        Milieu.add (new Label ("G")) ;  
        Milieu.add (new TextField (5)) ;  
        Milieu.add (new Label ("B")) ;  
        Milieu.add (new TextField (5)) ;  
  
        add ("North",Haut) ;  
        add ("Center",Milieu) ;  
    }  
}
```

Panel



L'alignement ne dépend pas de la taille de la fenêtre

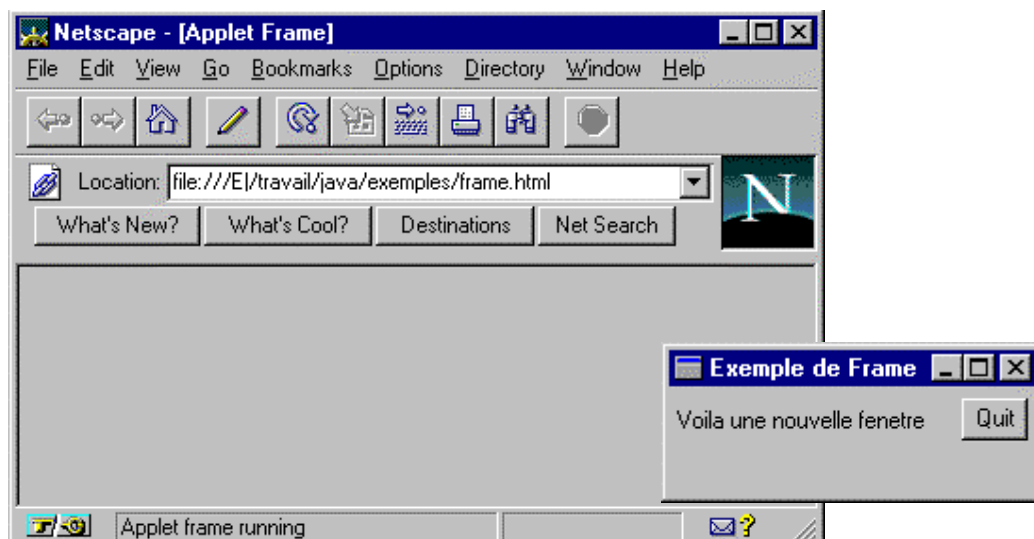
Frame

- Permet de définir des fenêtres indépendantes de la fenêtre du navigateur
- La fenêtre dispose de ses propres caractéristiques (barre de titre, barre de menu, curseur, etc.)
- *BorderLayout* est le layout par défaut

```
public Frame () ;  
public Frame (String title)
```

Exemple :

```
import java.applet.* ;  
import java.awt.* ;  
  
public class frame extends Applet  
{  
    public void init ()  
    {  
        Frame frm = new Frame ("Exemple de Frame") ;  
        frm.setLayout (new FlowLayout ()) ;  
        frm.add (new Label ("Voila une nouvelle fenetre")) ;  
        frm.add (new Button ("Quit")) ;  
        frm.show () ;  
    }  
}
```



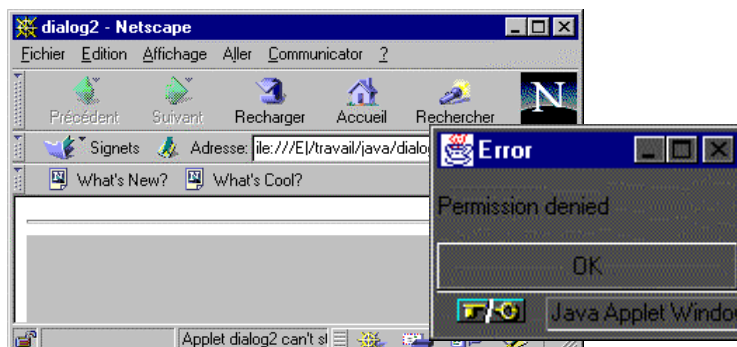
Les boîtes de dialogues

- Disposent de leur propre fenêtre (ont besoin d'une frame)
- peuvent être modale ou non
- Ont leur propre layout (*BorderLayout* par défaut)

public Dialog (Frame parent, boolean modal)
public Dialog (Frame parent, String title ,boolean modal)

Exemple :

```
import java.applet.*;  
import java.awt.*;  
public class dialog2 extends Applet  
{  
    public void init () {  
        AlertDialog error = new AlertDialog ("Permission denied") ;  
        error.setVisible (true) ;  
    }  
}  
class AlertDialog extends Dialog  
{  
    AlertDialog (String message) {  
        super (new Frame (), "Error",true) ;  
        add ("Center",new Label (message)) ;  
        add ("South", new Button ("OK")) ;  
        setSize (200,100) ;  
    }  
}
```



Les menus

- Utilisation de barres de menu (uniquement dans des Frames)
- Classes à utiliser : *Menu, MenuItem, MenuBar, CheckBoxMenuItem*
- Barre de menu positionnée par la méthode *setMenuBar ()* de la classe *Frame*
- Les méthodes *enable()* / *disable ()* permettent de rendre sensitif/insensitif une option ou sous option de menu (*setEnabled ()* en java 1.1)
- Les sous options de menu peuvent être séparées par un trait horizontal :

new MenuItem ("-") ;
- Java 1.1 apporte :

des popups menus (classe *PopupMenu* et méthode *add()* dans la classe *Component*)

des raccourcis claviers (classe *MenuShortCut*)

La barre de menu

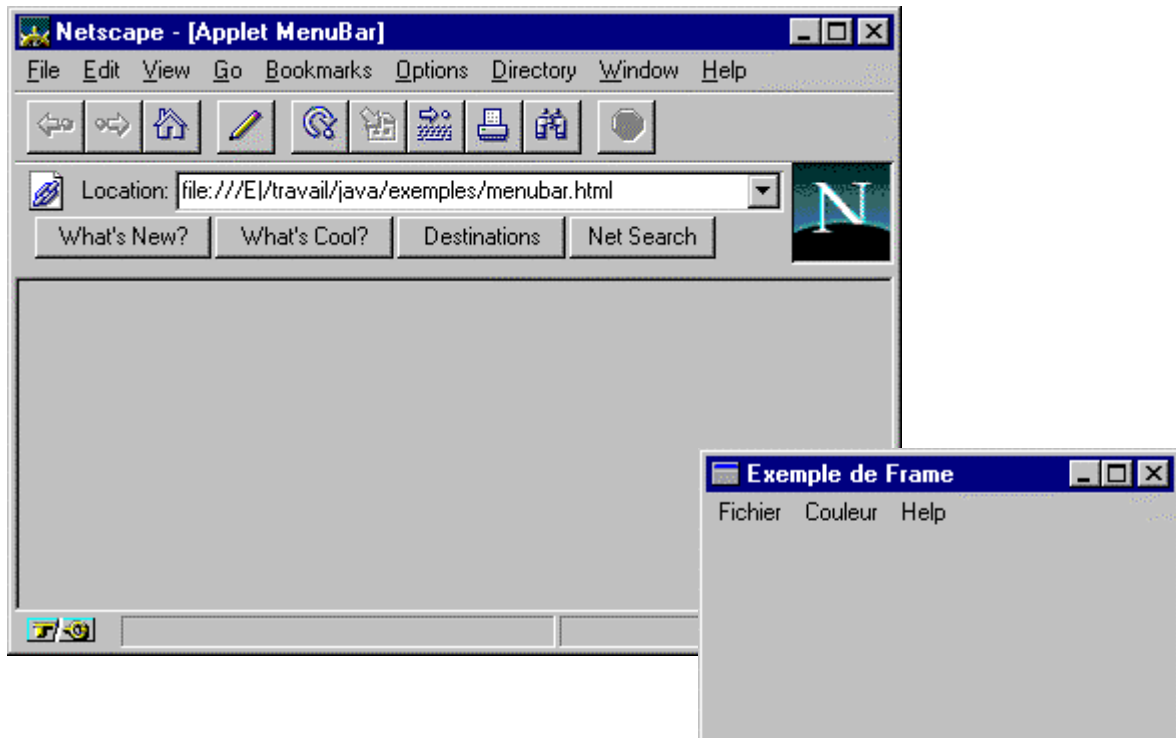
```
import java.applet.* ;
import java.awt.* ;
public class menubar extends Applet
{
    public void init () {
        MyFrame frm = new MyFrame ("Exemple de Frame") ;
    }
}

class MyFrame extends Frame
{
    MyFrame (String title)
    {
        super (title) ;
        MenuBar mb = new MenuBar () ;
        Menu fichier = new Menu ("Fichier") ;
        MenuItem ouvrir = new MenuItem ("Ouvrir") ;
        MenuItem quitter = new MenuItem ("Quitter") ;
        fichier.add (ouvrir) ;
        fichier.add (new MenuItem ("-")) ; // Separateur
        fichier.add (quitter) ;
        mb.add (fichier) ;

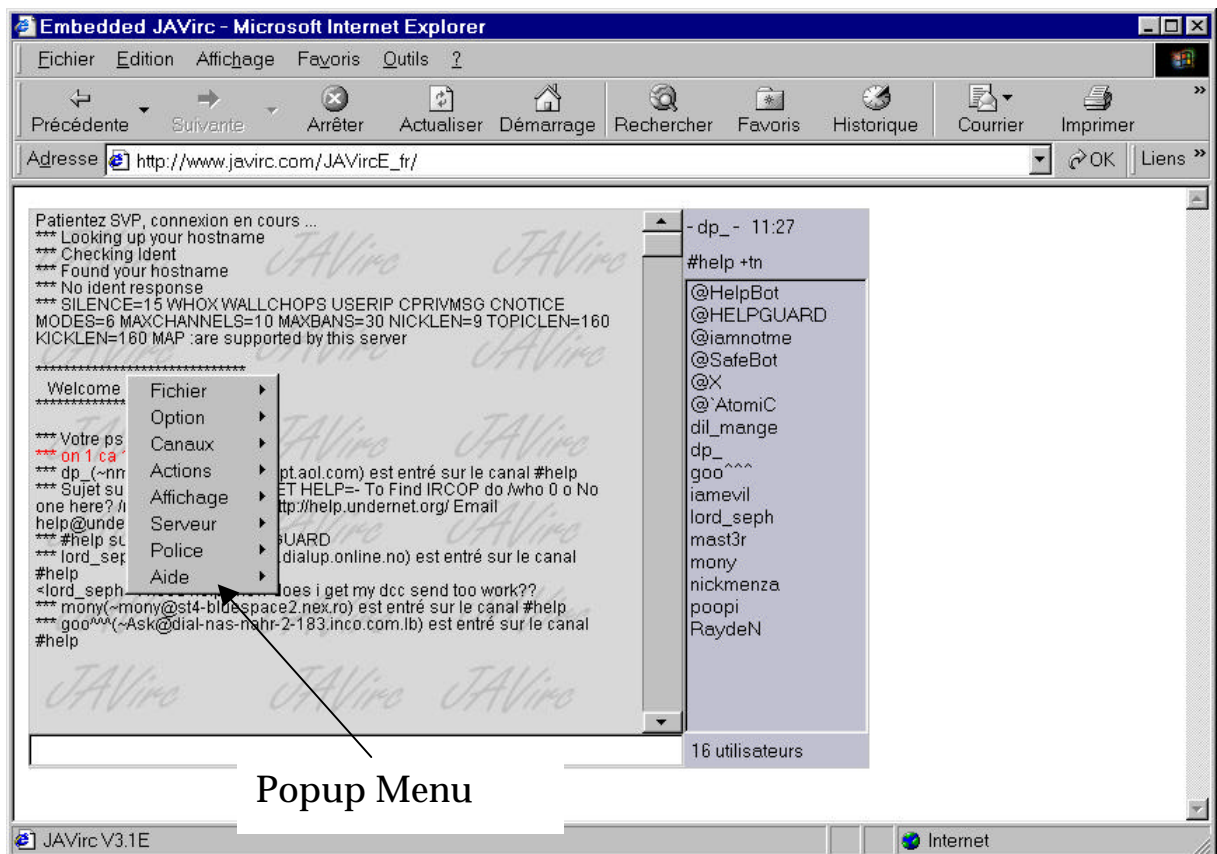
        Menu couleur = new Menu ("Couleur") ;
        CheckboxMenuItem rouge = new CheckboxMenuItem ("Rouge") ;
        couleur.add (rouge) ;
        CheckboxMenuItem noir = new CheckboxMenuItem ("Noir") ;
        couleur.add (noir) ;
        noir.setState (true) ;
        mb.add (couleur) ;

        Menu help = new Menu ("Help") ;
        mb.setHelpMenu (help) ;
        MenuItem apropos = new MenuItem ("A Propos") ;
        help.add ("A Propos") ;
        mb.add (help) ;
        setSize (200,100) ;
        setMenuBar (mb) ; show () ;
    }
}
```

Exemple de barre de menu



Exemple de popup menu



Gestion des événements

- Permet de définir un comportement en réponse à une action d'un utilisateur ou du système.
- Gestion très différente entre la version 1.0x et les versions 1.1x et supérieures de Java (sans compatibilité ascendante).
- Surtout ne pas mélanger les versions.

Dans la version 1.0x :

- La gestion des événements est simple à mettre en œuvre mais est inadaptée :
 - pour la mise en œuvre d'interfaces complexes
 - pour les évolutions futures (javabeans)
- Tous les événements sont des instances de la classe *Event*; il est donc nécessaire de tester manuellement la nature des événements survenus.
- Si l'on souhaite inclure la fonctionnalité (feel) à l'intérieur d'un composant graphique, il faut créer une classe dérivée.

A partir de la version 1.1

- Les événements seront envoyés à un composant uniquement si celui-ci en a manifesté l'intention.
- Les différents événements sont répartis en classes (package `java.awt.events`).
- La gestion des événements peut être déportés vers une classe séparée pouvant être utilisée par de multiples objets et modifiée indépendamment de l'interface utilisateur.

Evénements à partir de la version 1.1

- La réception et le traitement des événements sont clairement séparés :

- un objet reçoit un événement (écouteur)**
- l'événement est traité par un "listener"**
- le lien entre les deux est réalisé par l'appel d'une méthode spéciale d'enregistrement.**

- Un écouteur donné n'aura à traiter qu'un certain types d'événements. On trouvera un écouteur de touches, un écouteur de souris, etc.

La gestion des événements dans la version 1.1x se décompose en 3 phases :

- Détermination des événements du composant et attribution de ces événements à des écouteurs.**
- Ecriture du code de chaque écouteur.**
- Enregistrement des écouteurs pour le composant.**

Ecouteurs d'événements

- Une classe devant répondre à la gestion d'un événement doit implémenter l'interface qui gère cet événement (écouteur)
- Un écouteur d'événements peut être soit une classe séparée soit être incluse dans la classe.
- Un écouteur doit définir toutes les fonctions contenues dans l'interface (éventuellement vide)
- Liste des interfaces:

ActionListener	Action spécifique effectuée sur un composant
AdjustmentListener	Événement généré quand un composant est ajusté (barre de défilement, ...)
FocusListener	Focus clavier. Généré lorsqu'un composant reçoit ou perd le focus
ItemListener	Événement généré quand un élément tel qu'une case à cocher a été modifiée
KeyListener	Événement quand un utilisateur entre du texte au clavier
MouseListener	Événement généré par la souris
MouseMotionListener	Événement générés quand la souris se déplace sur un composant
WindowListener	Événement de gestion fenêtre

Création des composants

- **Les composants créés doivent indiquer les événements qui les intéressent et la classe dans laquelle se trouve les méthodes de gestion de ces événements.**
- **L'association composant-ecouteur se fait par l'une des méthodes suivantes:**

addActionListener ()

addAdjustmentListener ()

addFocusListener ()

addItemListener ()

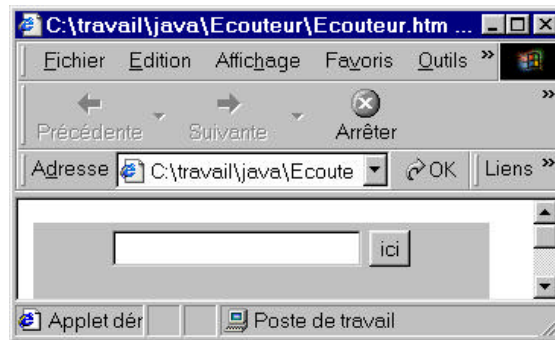
addKeyListener ()

addMouseListener ()

addMouseMotionListener ()

addWindowListener ()

Exemple



```
import java.applet.* ;  
import java.awt.* ;  
import java.awt.event.* ;  
  
public class Ecouteur extends Applet  
{  
    Button IciOuLa = new Button ("ici") ;  
    TextField saisie = new TextField (20) ;  
    Evenement evenement = new Evenement () ;  
  
    public void init ()  
    {  
        add (saisie) ; add (IciOuLa) ;  
        saisie.addActionListener (evenement) ;  
        IciOuLa.addActionListener (evenement) ;  
    }  
}  
  
class Evenement implements ActionListener  
{  
    public void actionPerformed (ActionEvent evt)  
    {  
        Object src = evt.getSource () ;  
        if (src instanceof TextField) ((TextField) src).setText ("") ;  
        else if (src instanceof Button)  
            if ( ((Button) src).getLabel ().equals ("ici"))  
                ((Button) src).setLabel ("la") ;  
            else ( (Button) src).setLabel ("ici") ;  
    }  
}
```

Adaptateur d'événements

- Implémenter un écouteur oblige à surcharger toutes les méthodes de l'interface

Exemple:

```
class MonApplet extends Applet implements MouseListener, KeyListener  
{  
  
// 8 fonctions événements à écrire  
}
```

- Le package `java.awt.event` contient des adaptateurs (adapters) qui sont des classes implémentant les écouteurs et fournissant des amorces vides des méthodes :

<i>MouseAdapter</i>	implémente	<i>MouseListener</i>
<i>MouseMotionAdapter</i>	implémente	<i>MouseMotionListener</i>
<i>KeyAdapter</i>	implémente	<i>KeyListener</i>

Exemple :

```
import java.awt.event.* ;  
  
class GestionSouris extends MouseAdapter  
{  
    public void mousePressed (MouseEvent e) {...}  
  
// On n'est pas obligé d'écrire les autres méthodes  
}
```

- L'absence d'héritage multiple limite l'utilisation des adaptateurs

Exemple d'utilisation d'un adaptateur

```
import java.applet.* ;  
import java.awt.event.* ;  
  
public class event11 extends Applet  
{  
    public void init ()  
    {  
        GestionSouris mouse = new GestionSouris () ;  
        addMouseListener (mouse) ;  
    }  
}  
  
class GestionSouris extends MouseAdapter  
{  
    public void mouseClicked (MouseEvent e)  
    {  
        System.out.println ("je suis dans mouseClicked") ;  
    }  
}
```

Les threads

- Thread = flot d'exécution à l'intérieur d'un processus
- Multi Processing != Multi Threading
- L'interpréteur Java exécute alternativement des instructions de chaque thread d'un processus pour donner l'impression de simultanéité

Thread A	Thread B
void runA () { System.out.println ("Début A") ; yield () ;	<i>Attente du processeur</i>
<i>Attente du processeur</i>	void runB () { System.out.println ("Début B"); yield () ;
System.out.println ("A:1") ; yield () ;	<i>Attente du processeur</i>
<i>Attente du processeur</i>	System.out.println ("B:1") ; yield () ;
System.out.println ("A:2") ; yield () ;	<i>Attente du processeur</i>
<i>Attente du processeur</i>	System.out.println ("B:2") ; yield () ;
System.out.println ("Fin A") ; }	<i>Attente du processeur</i>
	System.out.println ("Fin B") ; }

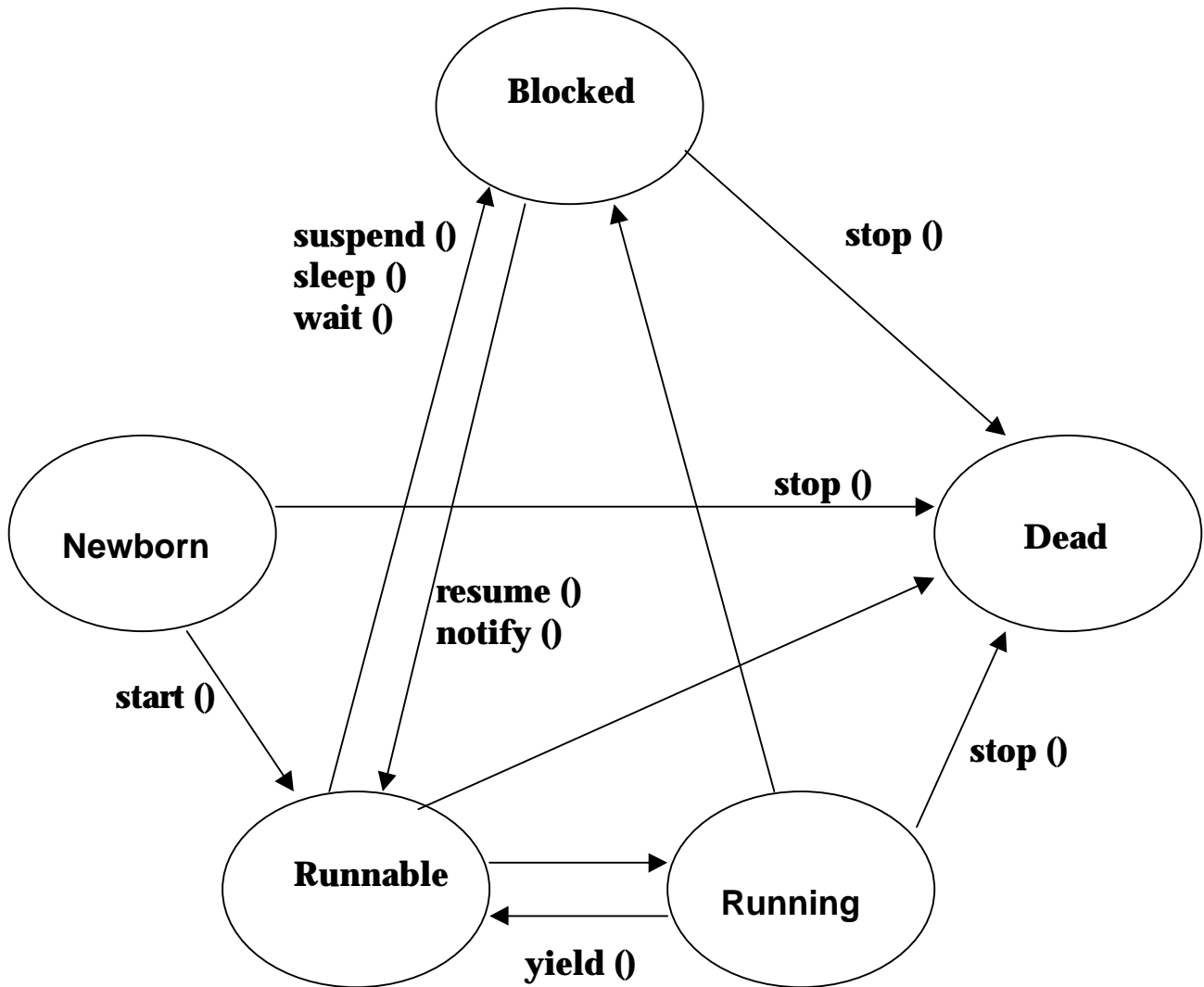
Affichage :

Début A
Début B
A:1
B:1
A:2
B:2
Fin A
Fin B

Priorités des threads

- Une priorité est attribuée à chaque thread permettant au "scheduler" java de déterminer le thread à exécuter
- Quand les threads ont la même priorité, ils partagent le processeur sur la base "premier entré / premier servi"
- Un thread de haute priorité prendra immédiatement le contrôle du processeur (en préemptant éventuellement celui-ci)
- Un thread de basse priorité devra attendre la fin (stop ()) ou le sommeil d'un thread de haute priorité (sleep(), wait ()).

Vie d'un thread



Création et utilisation d'un thread

- Utilisation de la classe *java.lang.Thread*

```
class MyThread extends Thread  
{  
    // Constructeur éventuel  
    ...  
    public void run ()  
    {  
        // corps du thread  
    }  
}  
  
MyThread thread = new MyThread () ;  
thread.start () ; // Appelle la méthode run ()
```

- Utilisation de l'interface *Runnable*

```
class MyThread extends classe implements Runnable  
{  
    public void run ()  
    {  
        // corps du thread  
    }  
}  
  
MyThread objet = new MyThread () ;  
Thread thread = new Thread (objet) ;  
thread.start () ;
```

Gestion de l'état d'un thread

start ()	Lance la méthode run () du thread
stop ()	Détruit le thread
sleep (long)	Endort le thread pour un temps approximatif dont la valeur en millisecondes est transmis en argument. En cas d'erreur, l'exception <i>InterruptedException</i> est générée
sleep (long,int)	Le deuxième argument affine le temps du sommeil en spécifiant le nombre de nanosecondes
suspend ()	Suspend le thread jusqu'au moment où il sera relancé par la méthode <i>resume ()</i>
resume ()	Reprend l'exécution d'un thread préalablement suspendu par <i>suspend ()</i>
yield ()	Indique à l'interpréteur Java qu'il peut passer au thread suivant
destroy ()	Destruction brutale du thread

La classe ThreadGroup

```
public ThreadGroup (String nom) ;  
public ThreadGroup (ThreadGroup parentGroup,String nom) ;
```

- Plusieurs threads peuvent appartenir à un groupe (regroupement par thème par exemple).

```
ThreadGroup groupe = new ThreadGroup ("Groupe") ;  
Thread first = new Thread (groupe,"premier") ;  
Thread second = new Thread (groupe,"deuxieme") ;
```

- Un objet ThreadGroup peut contenir des Threads mais aussi d'autres objets ThreadGroup

- Un thread ne peut manipuler que des threads de son groupe ou d'un groupe subordonné

- On peut agir simultanément sur tous les threads d'un groupe

Exemple : ***groupe.stop ()***

Synchronisation

- 2 threads accédant à une même donnée doivent être synchronisés
- La synchronisation peut se faire sur une instance de classe

Exemples :

```
class compteur
{
    private int valeur ;
    synchronized void incremente ()
    {
        valeur += 1 ;
    }
    int Combien ()
    {
        return valeur ;
    }
}
```

//-----

```
class Point
{
    private float x,y ;
    float x () { return x ; }           // ne nécessite pas de
                                        //synchronized
    float y () { return y ; }         // idem
    void print ()
    {
        float safeX,safeY ;
        synchronized (this)
        {
            safeX = x ; safeY = y ;
        }
        System.out.print ("voilà x et y : " + x + y) ;
    }
}
```

Synchronisation

- Protection d'une variable de classe :

. dans les exemples précédents, *synchronized* ne protégeait que l'instance d'une classe

. une variable de classe peut appartenir à plusieurs instances

Exemple :

```
class compteur
{
    private static int valeur ;
    void incremente ()
    {
        synchronized (getClass ())
        {
            valeur += 1 ;
        }
    }
    int Combien ()
    {
        return valeur ;
    }
}
```

- ***public final Class getClass()***

Determines the run-time class of an object.

- ***Tous*** les objets de classe *compteur* pourront être bloqués dans la méthode *incremente*

Les flux

- Un flux (stream) est un chemin de communication entre la source d'une information et sa destination

- Un processus consommateur n'a pas besoin de connaître la source de son information; un processus producteur n'a pas besoin de connaître la destination

- Une interprétation de haut niveau peut s'ajouter indépendamment des mécanismes de transmission des octets (exemple: flux de sérialisation d'objets dans Java 1.1)

- Java propose :

- . des flux d'entrée (création, utilisation, détection de fin)
- . des flux de sorties

Java 1.1 propose en plus :

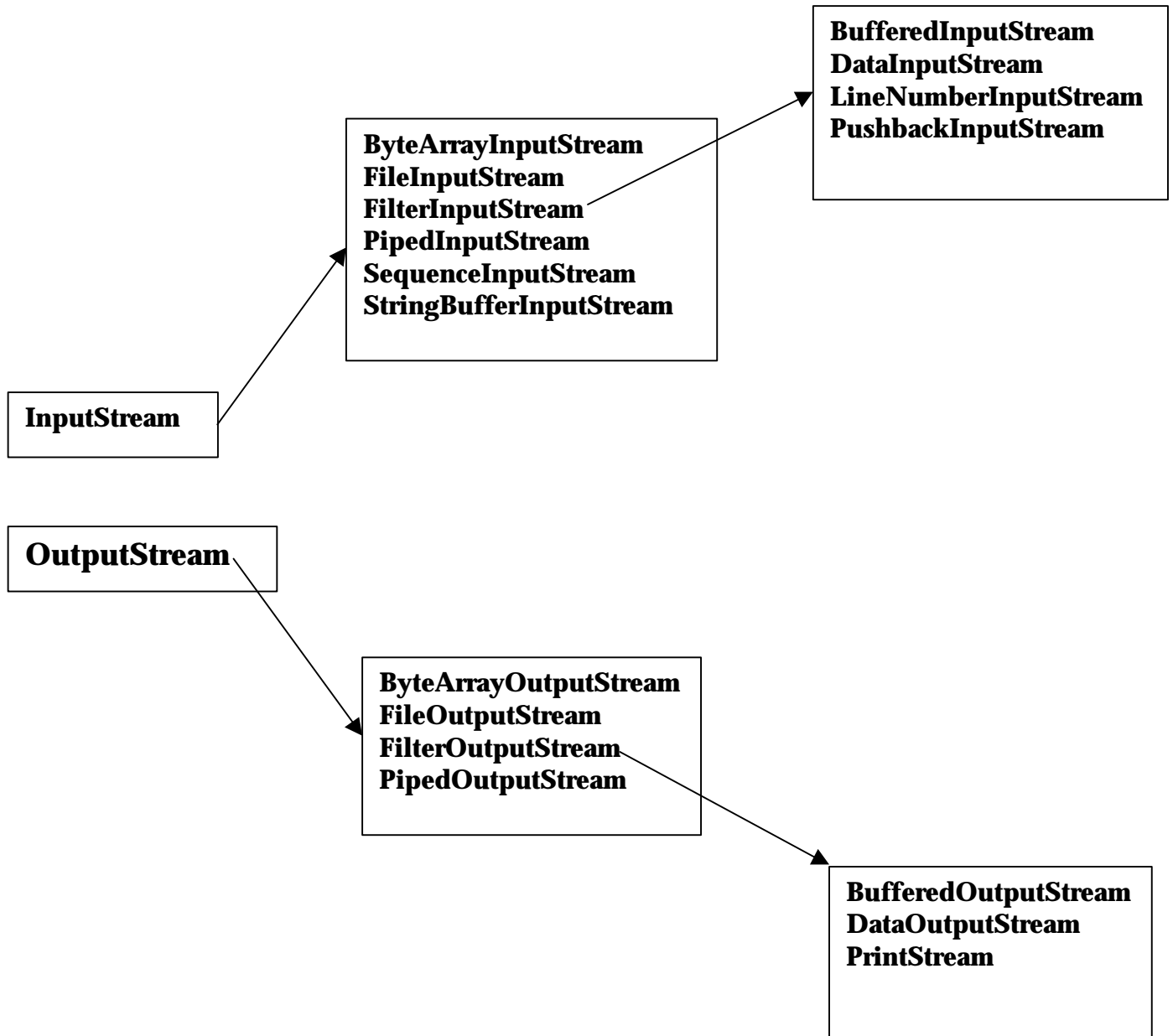
- . des classes d'E/S de caractères (readers, filtered readers, writers)

- La structure des flux est articulée autour de 2 classes abstraites du package *java.io* :

InputStream
OutputStream

- Toutes les méthodes peuvent générer une *IOException*

InputStream / OutputStream



InputStream

- **InputStream** est une classe abstraite qui définit les moyens grâce auxquels le consommateur peut lire un flux d'octets

- Les méthodes de lecture :

```
public int read () ;  
public int read (byte b [ ]) ;  
public int read (byte b [ ], int off, int len) ;
```

Exemple :

```
InputStream s = ..... ;  
byte b [ ] = new byte [1024] ;  
try {  
    s.read (buffer) ;  
} catch (IOException e)  
    {  
    }
```

- Sauter des octets : **public long skip (long n) ;**

- Combien d'octets dans le flux : **public int available () ;**

- Le flux supporte-t'il le marquage ?

```
public boolean s.markSupported () ;
```

Si oui :

. marquage d'un flux :

```
public void mark (int readlimit) ;
```

. revenir sur la marque

```
public void reset ()
```

- Fermer le flux : **public void close () ;**

OutputStream

- **OutputStream** est une classe abstraite qui définit les moyens par lesquels un producteur peut écrire un flux d'octets à une destination donnée

- Les méthodes d'écriture :

```
public void write (int b) ;  
public void write (byte b [ ]) ;  
public void write (byte b [ ], int off, int len) ;
```

- Nettoyage du flux, forçant l'écriture des données bufférisées :

```
public void flush () ;
```

- Fermeture du flux

```
public void close () ;
```

Exemples d'InputStream

- ByteArrayInputStream

Crée un flux à partir d'un tableau de bytes

```
public ByteArrayInputStream (byte buf [ ] ) ;  
public ByteArrayInputStream (byte buf [ ], int off, int len) ;
```

- FileStream

```
public FileInputStream (String name) ;  
public FileInputStream (File file) ;  
public FileInputStream (int fd) ;
```

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

```
public class filestream  
{  
  
    public static void main (String args []) throws IOException  
    {  
        FileDialog f = new FileDialog (new Frame(), "Ouvrir",  
                                         FileDialog.LOAD) ;  
        f.show () ;  
  
        String fichier = f.getFile () ;  
        String Dir     = f.getDirectory () ;  
  
        InputStream s = new FileInputStream (Dir + fichier) ;  
  
        byte buffer [] = new byte [s.available()] ;  
        s.read (buffer) ;  
        for (int i = 0 ; i != buffer.length ; i++)  
            System.out.print ( (char) buffer [i]) ;  
    }  
}
```

Exemples d'OutputStream

- ByteArrayOutputStream

Envoie un flux dans un tableau de bytes

```
public ByteArrayOutputStream () ;  
public ByteArrayOutputStream (int size) ;
```

Exemple :

```
FileDialog f = new FileDialog (new Frame(), "Ouvrir", FileDialog.LOAD) ;  
f.show () ;
```

```
String fichier = f.getFile () ;  
String Dir    = f.getDirectory () ;
```

```
InputStream s = new FileInputStream (Dir + fichier) ;
```

```
byte buffer [] = new byte [s.available ()] ;  
s.read (buffer) ;
```

```
ByteArrayOutputStream b = new ByteArrayOutputStream () ;  
b.write (buffer) ; b.write (buffer) ;  
System.out.println (b.size () ) ;  
byte result [] = b.toByteArray () ; // contient 2 fois le contenu du fichier  
System.out.println (result.length) ;
```

- FileOutputStream

```
public FileOutputStream(File file);  
public FileOutputStream(FileDescriptor fdObj);  
public FileOutputStream(String name);
```

Exemple :

```
File Fichier = new File ("resultat") ; // dans le répertoire courant  
FileOutputStream f = new FileOutputStream (Fichier) ;  
f.write (result) ; // result est un tableau de bytes  
f.close () ;  
System.out.println (Fichier.length()) ; // Affiche la taille du fichier
```

FilterInputStream / FilterOutputStream

Ces deux classes donnent accès aux méthodes des classes *InputStream* et *OutputStream* à travers 7 sous classes :

- *BufferedInputStream* et *BufferedOutputStream* permettent de lire et écrire des données à travers un tampon de lecture/écriture pour améliorer les performances.

```
public BufferedInputStream (InputStream in) ;  
public BufferedInputStream (InputStream in, int size) ;
```

```
public BufferedOutputStream (OutputStream out) ;  
public BufferedOutputStream (OutputStream out, int size) ;
```

- *DataInputStream* et *DataOutputStream* permettent de lire des données formatées (byte, int, char, float, double, etc.)

```
public DataInputStream (InputStream in) ;  
  
public DataOutputStream (OutputStream out) ;
```

Exemple :

```
InputStream s = new FileInputStream (Dir + fichier) ;  
DataInputStream data = new DataInputStream (s) ;  
  
double valeur = data.readDouble () ;
```

FilterInputStream / FilterOutputStream

- *LineNumberInputStream* numérote automatiquement les lignes lues.

```
public LineNumberInputStream (InputStream s) ;  
public int getLineNumber () ;  
public void setLineNumber () ; // Positionne No de ligne
```

Exemple :

```
FileDialog f = new FileDialog (new Frame (), "Ouvrir", FileDialog.LOAD) ;  
f.show () ;
```

```
String fichier = f.getFile () ;  
String Dir    = f.getDirectory () ;
```

```
InputStream s = new FileInputStream (Dir + fichier) ;  
LineNumberInputStream line = new LineNumberInputStream (s) ;  
DataInputStream data = new DataInputStream (line) ;
```

```
String buffer ;  
buffer = data.readLine () ;  
System.out.println (buffer) ;           // Affiche ligne 0  
System.out.println (line.getLineNumber()) ; // Affiche 1
```

- La classe *PushbackInputStream* possède une méthode *unread()* qui permet de relire le dernier caractère lu.

- La classe *PrintStream* possède des méthodes qui permettent d'afficher des éléments de différents types (comparables aux méthodes *System.out.print* et *System.out.println*)

Exemple :

```
PrintStream s = new PrintStream (new FileOutputStream ("resultat")) ;  
s.println ("On écrit dans le fichier resultat") ;
```

PipedInputStream / PipedOutputStream

Ces 2 classes permettent d'établir une connexion entre deux threads

```
public PipedInputStream();  
public PipedInputStream(PipedOutputStream src);  
  
public PipedOutputStream();  
public PipedOutputStream(PipedInputStream snk);
```

Exemple :

```
import java.io.* ;  
  
public class pipe  
{  
    public static void main (String args []) throws IOException  
    {  
        PipedOutputStream sOut = new PipedOutputStream () ;  
        DataOutputStream envoie = new DataOutputStream (sOut) ;  
        MyThread thrd = new MyThread (sOut) ;  
        thrd.start () ;  
        envoie.writeChars ("envoi de caracteres\n") ;  
    }  
}  
  
class MyThread extends Thread  
{  
    DataInputStream recoit ;  
    MyThread (PipedOutputStream sOut) throws IOException  
    {  
        recoit = new DataInputStream (new PipedInputStream (sOut)) ;  
    }  
  
    public void run ()  
    {  
        try {                // On affiche ce qu'on recoit du pipe  
                            System.out.println (recoit.readLine () ;  
                            } catch (IOException e) {}  
    }  
}
```


Les sockets : côté client

- Utilise la classe *Socket* dans le package *java.net*.

public Socket (String host, int port)
throws UnknownHostException, IOException ;

public Socket (String host, int port, boolean stream)
throws IOException ;

public Socket (InetAddress address, int port) throws IOException ;

***public Socket (InetAddress address, int port, boolean stream)
throws IOException ;***

host :	Nom du serveur
port :	Numéro de port
stream :	booléen true = stream (défaut) false = datagram
address :	adresse IP

Quelques méthodes de la classe Socket :

```
public void close();  
public InetAddress getInetAddress();  
public InputStream getInputStream();  
public int getLocalPort();  
public OutputStream getOutputStream();  
public int getPort();
```

Exemple de client

```
import java.net.* ;  
import java.io.* ;  
  
public class heure  
{  
    public static void main (String args []) throws IOException  
    {  
        Socket s = null;  
        PrintStream flux = null;  
        try {  
            s = new Socket ("ibm580",2000) ;  
        }  
        catch (IOException e)  
        {  
            System.out.println ("Connection failed") ;  
            System.exit (1) ;  
        }  
  
        flux = new PrintStream (s.getOutputStream (),true) ;  
        flux.println ("heure") ;  
        DataInputStream reponse = new DataInputStream  
                                (s.getInputStream () ) ;  
        System.out.println (reponse.readLine () ) ;  
  
    }  
  
}
```

Les sockets : côté serveur

- Utilise la classe *ServerSocket* dans le package *java.net*.

public ServerSocket (int port) throws IOException ;
public ServerSocket (int port, int count) throws IOException ;

port :	port d'écoute
count :	taille de la file d'attente (50 par défaut)

Quelques méthodes de la classe Socket :

public Socket accept() ;

public void close() ;

public InetAddress getInetAddress() ;

public int getLocalPort() ;

Exemple de serveur

```
import java.net.* ;  
import java.io.* ;  
import java.util.Date ;  
  
public class Serveur  
{  
    public static void main (String args []) throws IOException  
    {  
        ServerSocket s = null;  
        PrintStream flux = null;  
        try {  
            s = new ServerSocket (2000) ;  
        }  
        catch (IOException e)  
        {  
            System.out.println ("Connection failed") ;  
            System.exit (1) ;  
        }  
  
        while (true)  
        {  
            Socket service = s.accept () ;  
            DataInputStream requete = new DataInputStream  
                (service.getInputStream () ) ;  
  
            if (requete.readLine ().equals ("heure"))  
            {  
                Date date = new Date () ;  
                flux = new PrintStream  
                    (service.getOutputStream (),true) ;  
                flux.println (date) ;  
            }  
        }  
    }  
}
```

Les sockets

- **Les navigateurs imposent des restrictions d'utilisation des sockets au sein des applets (voir le chapitre consacré à la sécurité java).**
- **En java 1.0x, les classes *Socket* et *ServerSocket* sont des classes finales. Pour créer ses propres classes Socket, il faut dériver la classe *SocketImpl***
- **En Java 1.1, les classes *Socket* et *ServerSocket* sont dérivables**

La classe URL

- classe incluse dans le package *java.net*.
- Constructeurs:

public URL(String spec) throws MalformedURLException

***public URL(String protocol, String host, int port, String file)
throws MalformedURLException***

***public URL(String protocol, String host, String file)
throws MalformedURLException***

public URL(URL context, String spec) throws MalformedURLException

<i>protocol</i>	http,ftp,gopher,file
<i>host</i>	Nom de la machine distante
<i>port</i>	Port de communication
<i>file</i>	Nom complet du document
<i>spec</i>	Protocole + Nom machine + Nom fichier
<i>context</i>	Chemin d'accès de base

Exemples:

URL www = new URL ("http","www.ismra.fr",80,"ecole.htm") ;

URL www = new URL ("http://www.ismra.fr/ecole.htm") ;

URL www = new URL (getCodeBase (), "ecole.htm") ;

URL www = new URL (getDocumentBase (), "ecole.htm") ;

2 Méthodes de la classe *applet*:

getCodeBase () **Renvoie l'adresse de base du site à partir duquel l'applet a été téléchargée.**

getDocumentBase () **Renvoie l'adresse de base du site sur lequel se trouve le document html.**

Exemple d'utilisation de la classe URL

```
import java.net.* ;
import java.awt.* ;
import java.io.* ;

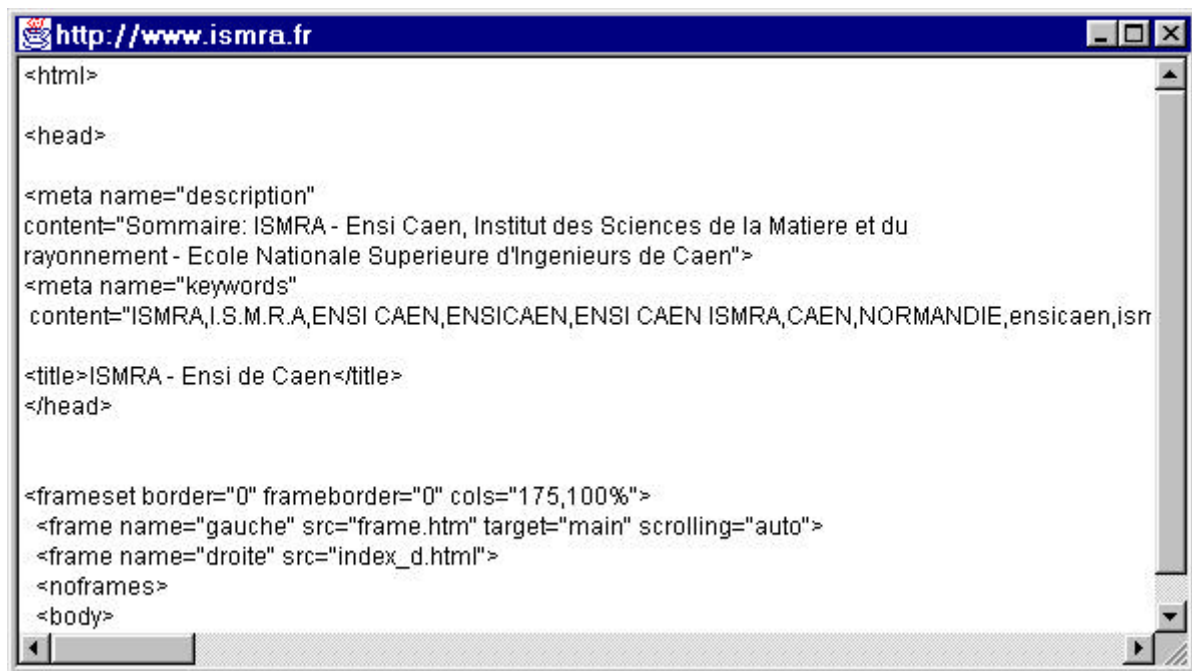
public class URLdemo
{
    static public void main (String args []) throws MalformedURLException
    {
        if (args.length == 0) System.err.println ("Usage: java URLdemo url") ;
        else URLlire l = new URLlire (args [0]) ;
    }
}
class URLlire extends Frame
{
    URL url ;
    InputStream content ;
    int length ;
    byte buffer [] ;

    public URLlire (String sURL)
    {
        setTitle (sURL) ;
        try {
            url = new URL (sURL) ;
            content = url.openStream () ;
            length = content.available () ;
            buffer = new byte [length] ;
            content.read (buffer,0,length) ;
        } catch (MalformedURLException e1)
        {
            System.err.println ("Erreur URL " + e1) ;
            System.exit (1) ;
        }
        catch (IOException e2)
        {
            System.err.println ("Erreur IO " + e2) ;
            System.exit (1) ;
        }

        add ("Center",new TextArea (new String (buffer),10,80)) ;
        setSize (600,500) ;
        setVisible (true) ;
    }
}
```

Classe URL : résultat de l'exemple précédent

java URLdemo <http://www.ismra.fr>



```
<html>

<head>

<meta name="description"
content="Sommaire: ISMRA - Ensi Caen, Institut des Sciences de la Matiere et du
rayonnement - Ecole Nationale Superieure d'Ingenieurs de Caen">
<meta name="keywords"
content="ISMRA,I.S.M.R.A,ENSI CAEN,ENSICAEN,ENSI CAEN ISMRA,CAEN,NORMANDIE,ensicaen,isr

<title>ISMRA - Ensi de Caen</title>
</head>

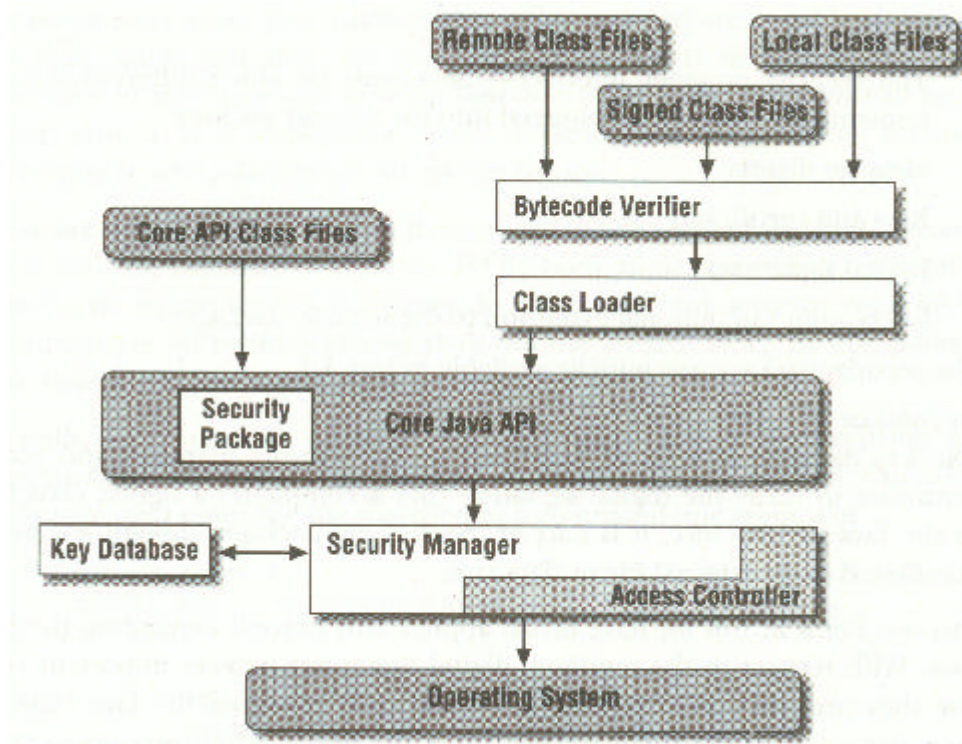
<frameset border="0" frameborder="0" cols="175,100%">
  <frame name="gauche" src="frame.htm" target="main" scrolling="auto">
  <frame name="droite" src="index_d.html">
<noframes>
<body>
```


La sécurité

- **La sécurité est un point sensible en java**
- **Quelques points de sécurité à prendre en compte:**
 - 1) Se prémunir des programmes malveillants (virus, chevaux de troie)**
 - 2) Pas d'intrusion (pas d'accès à des informations privées)**
 - 3) Authentification des parties en cours**
 - 4) Cryptage**
 - 5) Audit**
- ...
- **Les points 1 et 2 sont pris en compte dès la norme 1.0 de Java**
- **Le point 3 a été pris en compte par la norme 1.1**
- **Le point 4 a été pris en compte par la norme 1.2**
- **Le point 5 peut être pris en compte dans la norme 1.2 par ajout d'un module**

java sandbox

- La sécurité java est axée autour d'une "sandbox" qui va établir le contour de l'environnement auquel peut accéder l'application.
- La notion de sécurité dans les applications et les applets est très différente:
 - . Une application peut définir sa politique de sécurité
 - . Une applet est tributaire de la politique de sécurité définie par le navigateur qui l'a chargée.
- Une "sandbox" peut être le CPU et la mémoire centrale de la machine cliente et le serveur web de téléchargement de l'applet.
- Anatomie d'une application/applet java



La sécurité des applets java

- **Imposée par les navigateurs**
- **Les applets sont soumises à de nombreuses restrictions:**
 - . **Pas d'accès au disque dur local de l'utilisateur.**
 - . **Pas de connexion sur une machine autre que le serveur WWW d'origine de l'applet.**
 - . **Pas de lancement de programme sur la machine de l'utilisateur.**
 - . **Pas de chargement de programmes stockés sur la machine de l'utilisateur (exécutable, bibliothèque partagée).**
- **Impossible de passer outre ces restrictions dans la norme 1.0.**
- **La norme 1.1 a introduit le concept d'applet signée.**
- **La signature utilise la technologie RSA:**
 - . **Génération d'un couple clé privée/clé publique**
 - . **Signature de la clé publique par un tiers certificateur (moyennant finance).**
- **Exemples de tiers certificateurs:**
 - . **Verisign** **<http://www.verisign.com>**
 - . **Thawte** **<http://www.thawte.com>**

Les fichiers archives

- Les fichiers archives sont utilisées pour accélérer les temps de téléchargement des applets.
- Les fichiers archives peuvent être signés.
- Format différent pour Netscape et Internet Explorer.
- Netscape utilise les fichiers jar (Java ARchive)
- Un fichier jar peut être créé avec la commande `jar` du JDK; la syntaxe est inspirée de la commande `tar` d'unix.

Exemples:

```
jar cvf applet.jar *.class  
jar tvf applet.jar  
jar xvf applet.jar
```

- Utilisation d'un fichier jar:

```
<APPLET CODE="app.class" ARCHIVE="applet.jar" ...>
```

- Un fichier jar peut être généré et signé par la commande ***signtool*** (téléchargeable sur <http://developer.netscape.com/software/signtool/jarpack.html>).

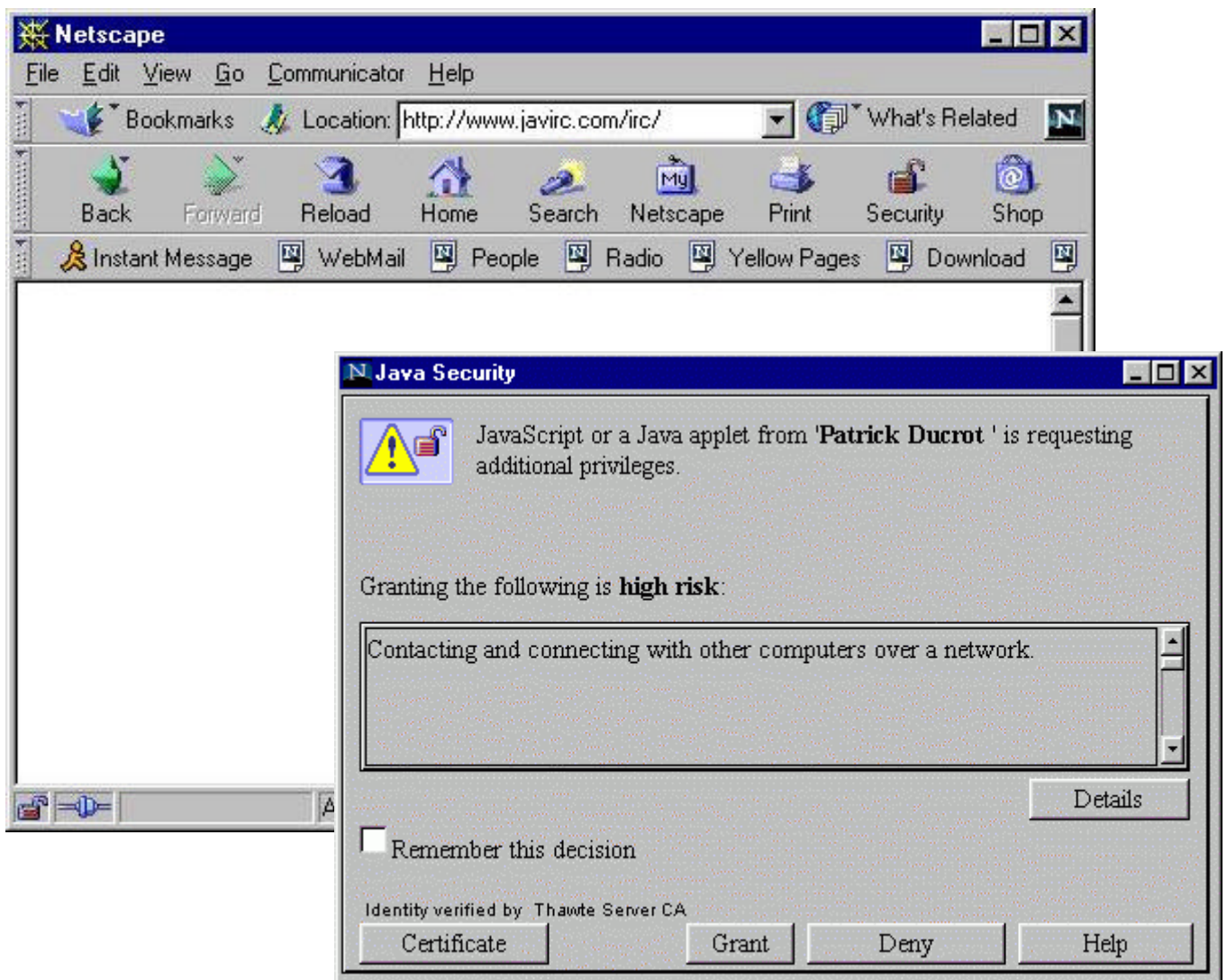
Exemple :

```
signtool -d c:\progra~1\netscape\users\default -k "Patrick  
Ducrot" -c9 -Z JAVirc.jar signdir
```

- L'applet java indique les privilèges dont elle aura besoin:

```
import netscape.security.PrivilegeManager;  
try {  
    if (System.getProperty ("java.vendor").startsWith ("Netscape"))  
        PrivilegeManager.enablePrivilege ("UniversalConnect");  
} catch (Exception e)  
{  
    System.err.println ("Netscape violation " + e.getMessage () );  
}
```

Netscape et les applets signées



Les archives signées d'Internet Explorer

- Internet Explorer utilise les fichiers cab (CABinet file).
- L'archivage est effectué par la commande *cabarc* et la signature par la commande *signcode*.
- Ces deux commandes font partie de la SDK-Java de Microsoft (<http://www.microsoft.com/downloads>).
- Exemple:

```
cabarc N JAVirc.cab *.class  
signcode -spc mycert.spc -v mykey.pvk -n JAVirc -i  
http://www.javirc.com JAVirc.cab
```

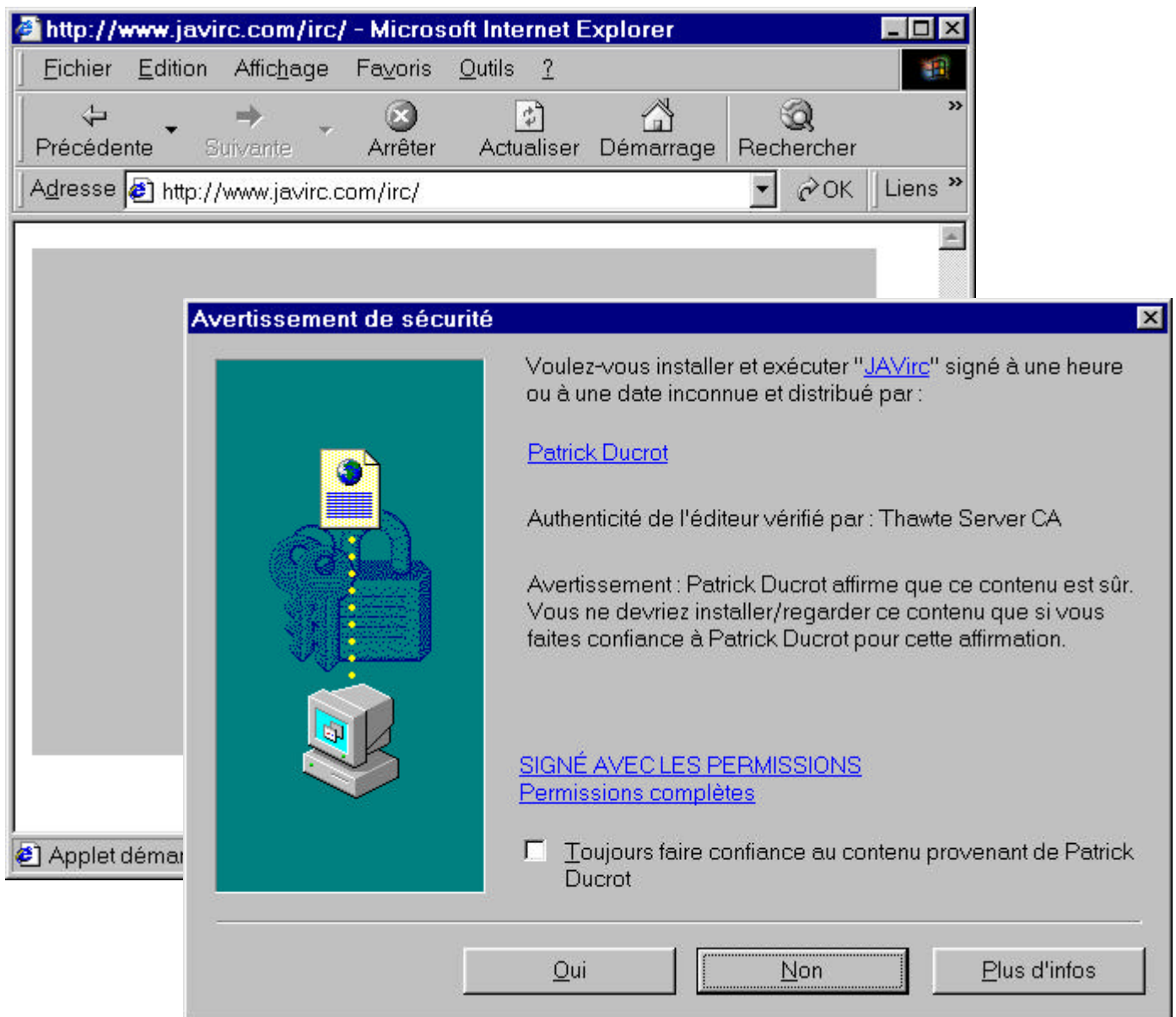
- Utilisation d'un fichier cab:

```
<PARAM NAME=cabbase VALUE="JAVirc.cab">
```

- L'applet java indique les privilèges dont elle aura besoin:

```
import com.ms.security.* ;  
  
try {  
    if (System.getProperty ("java.vendor").startsWith ("Microsoft"))  
        PolicyEngine.assertPermission ( PermissionID.SYSTEM) ;  
    }  
    catch (Exception e)  
    {  
        System.err.println ("Microsoft violation " + e.getMessage () ;  
    }
```

Internet Explorer et les applets signées



Bibliographie

<i>Titre</i>	<i>Auteurs</i>	<i>Editeur</i>
Java source book	Ed Anuff	Wiley computer publishing
Teach yourself Java in 21 days	Laura Lemay Charles L. Perkins	Samsnet
Livre d'or de Java	Patrick Longuet	Sybex
PC Poche JAVA	Rolf Maurers Kai Baufeldt	Micro application
Apprenez Java 1.1 en 21 jours	Laura Lemay Charles L. Perkins	Simon & Schuster Macmillan
Java 1.2	Laura Lemay Roger Cadenhead	Simon & Schuster Macmillan
Java Security	Scott Oaks	O'Reilly
Le dictionnaire officiel Java 2	Patrick Chan	Eyrolles