ENSICAEN
ÉCOLE NATIONALE SUPÉRIEURE D'INGÉNIEURS DE CAEN
& CENTRE DE RECHERCHE

JAVA
programmation objet

Patrick Ducrot

dp@ensicaen.fr

http://www.ducrot.org/java/PolycopieJAVA.pdf

Plan du document (1/2)

Plan du document (2/2)

Généralités

Généralités

- Langage conçu par les ingénieurs de « SUN Microsystem » (société rachetée par Oracle en avril 2009)

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

Généralités

- ***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 interprétateurs java avant exécution.
  - Un débordement de tableau déclenchera automatiquement une exception.
  - L'absence d'arithmétique de pointeur évite les malversations.

Généralités

- ***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***
  - Java est un langage pseudo interprété
  - Techniques de "Just in Time" (JIT) améliorent 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.

Les différentes version de java

- **Java 1.0 (23 janvier 1996)**
  - **8 Packages**
  - **212 Classes et Interfaces**
  - **1545 Méthodes**
- **Java 1.1 (19 février 1997)**
  - **23 Packages**
  - **504 Classes et Interfaces**
  - **3 851 Méthodes**
- **Java 1.2 (Java 2) (9 décembre 1998)**
  - **60 Packages**
  - **1 781 Classes et Interfaces**
  - **15 060 Méthodes**
- **Et bien plus dans les versions 1.3 (8 mai 2000), 1.4 (6 février 2002), 1.5 (tiger, 30 septembre 2004) ,1.6 (mustang, 11 décembre 2006)**
- **Dernière version: 1.7, le 28 juillet 2011 (OpenSource et sous l'ère Oracle).**

Les outils de développement

- Différents environnements java :
  - Java SE (Standard Edition)
  - Java ME (Mobile Edition)
  - Java EE (Enterprise Edition)

L'environnement J2SE

- L'outil de base : le JDK (Java Development Kit) de SUN :
  - http://java.sun.com (lien redirigé sur le site d'Oracle).
    - 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:
  - http://www.eclipse.org
  - http://www.netbeans.org/

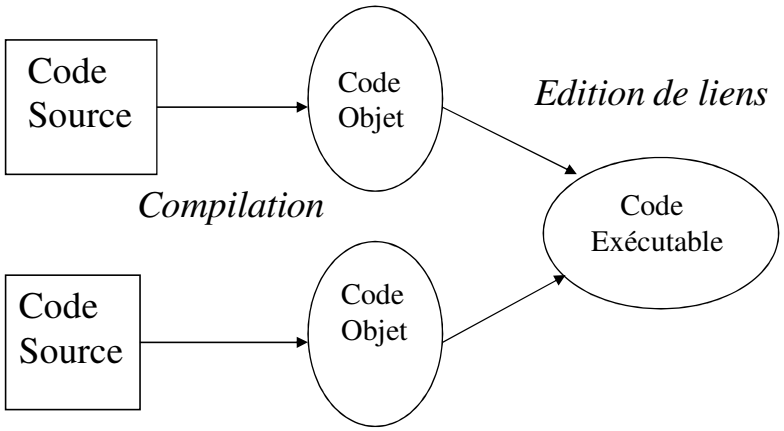
Utilisation de JAVA

- **Le langage Java peut générer :**
  - **des applications**
  - **des applets**
  - **des servlets**
  - **des midlets**
  - **etc.**

Génération de code exécutable dans les langages de programmation

- **Le code est généré par un compilateur en plusieurs étapes :**
  - **Vérification syntaxique.**
  - **Vérification sémantique (typage).**
  - **Production de code dans un langage plus proche de la machine.**
  - **Production de « briques » de code assemblables.**

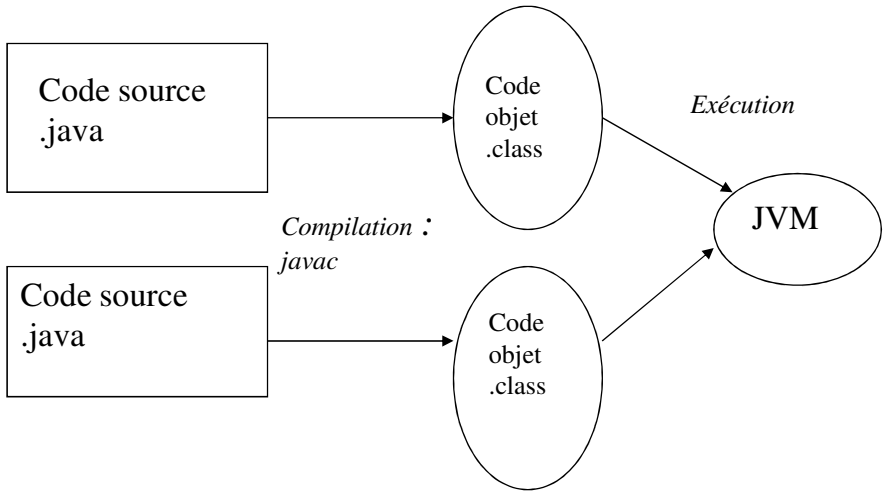
Génération de code exécutable dans les langages de programmation



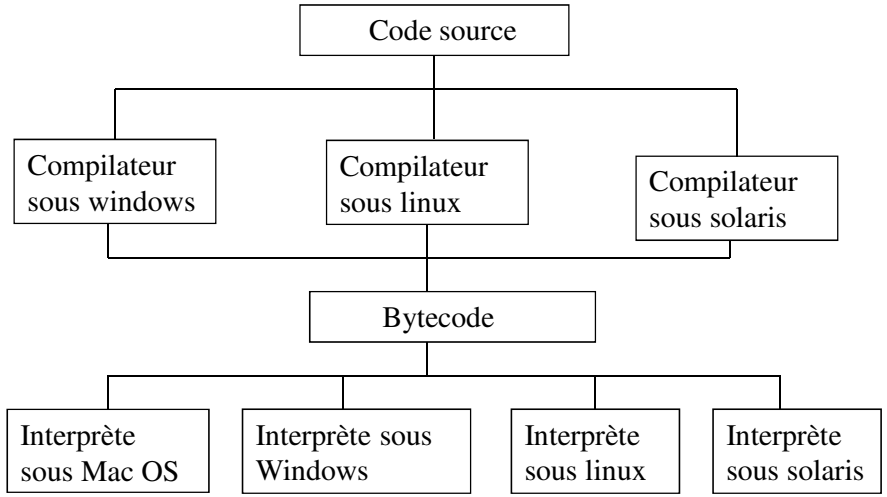
Génération de code exécutable dans les langages de programmation

- Avantages/inconvénients du code natif
  - Rapidité d'exécution
  - Nécessité de recompiler lors du portage d'un logiciel sur une autre architecture/système d'exploitation
  - Choix de la distribution du logiciel : source ou binaire ?

Génération de code en java



Principe Machine Virtuelle



Avantages/Inconvénients du bytecode

- Code portable au niveau binaire
- Moins efficace que du code natif (mais compensé par la technologie JIT)

Exemple d'application

- **Fichier: First.java**

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

- **Compilation :** *javac First.java*

  Création de First.class

- **Exécution :** *java First*

Syntaxe de base

Syntaxe de base du langage Java

- Les commentaires existent sous plusieurs formes:
  - Commentaires multi lignes

    /*

    */
  - Commentaires sur une seule ou fraction de ligne

    //
  - Commentaires destinés au générateur de documentation 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).**

Type de données prédéfinis

- **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 754Un suffixe *f* ou *d* après une valeur numérique permet de spécifier le type.**
    - **Exemples :**

```
double x = 145.56d ;
float  y = 23.4f ;
float f = 23.65 ;   // Erreur
```

Type de données prédéfinis

- **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)		

Types de données prédéfinis

- **Chaînes de caractères**
  - Les chaînes de caractères sont manipulées par la *classe* String ; ce n'est donc pas un type scalaire, mais il s'en approche beaucoup dans son utilisation.

    Exemple :

    ```
    String str = "exemple de chaîne de caractères" ;
    String chaine = "Le soleil " + "brille" ;   // Opérateur de concaténation
    ```

  - Remarque: Depuis la version 1.7, il est possible d'utiliser les chaînes de caractères dans les structures *switch/case*

    Exemple:

    ```
    String chaine= "… " ;
    switch (chaine)
    {
        case  "Bonjour" :       System.out.println (" Je suis poli ") ;
                                break ;
        default:                System.out.println(" Je suis impoli ") ;
    }
    ```

Les tableaux

- **Les tableaux peuvent être 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} ;

Les tableaux

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

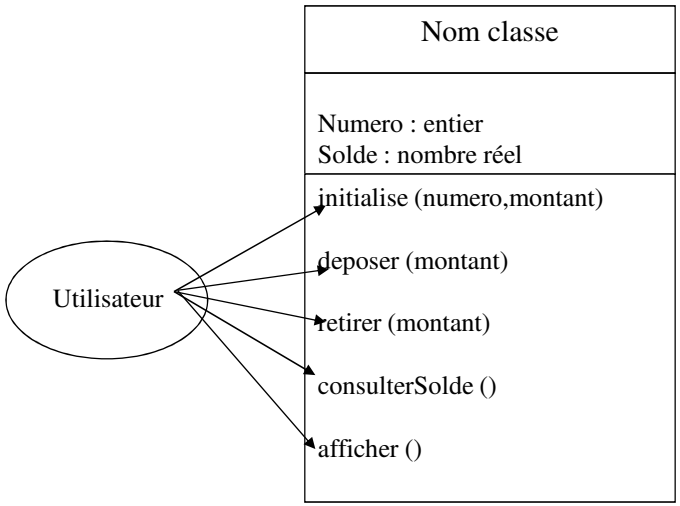
```
System.out.println (table3.length) ;

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

Concepts objet, classes, constructeurs, destructeurs, surcharge

Concepts des langages objet

- **Concept de base de la programmation orientée objet : la classe**
- **Une classe modélise la structure statique (*données membres*) et le comportement dynamique (*méthodes*) des objets associés à cette classe.**
- **Un objet d'une classe est appelé une *instance*.**
- **Une classe est la description d'un objet. Chaque objet est créé à partir d'une classe (avec l'opérateur *new*).**

Exemple


Nom classe
Numero : entier
Solde : nombre réel
initialise (numero,montant)
deposer (montant)
retirer (montant)
consulterSolde ()
afficher ()
Utilisateur

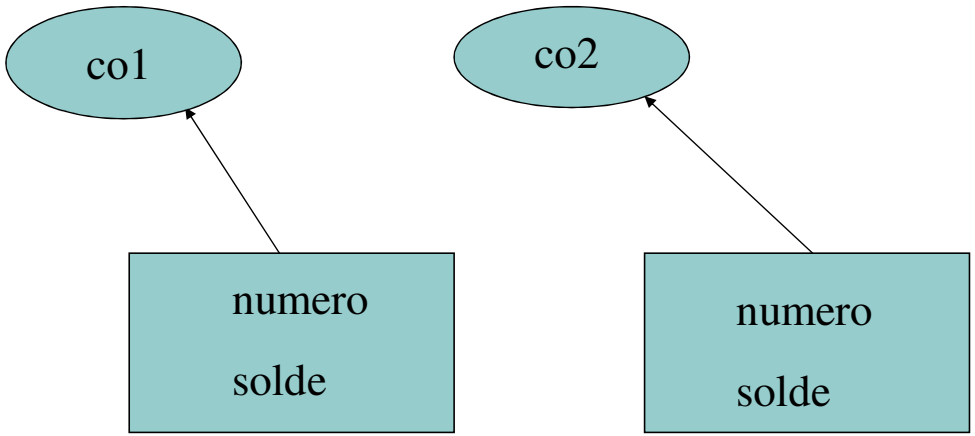
Exemple d'écriture de la classe Compte

```
class Compte
{
    private int numero ;
    private float solde ;

    public void initialise (int n, float s)  { numero = n ; solde = s ; }
    public void deposer (float montant) { solde = solde + montant ; }
    public void retirer (float montant)    {  solde = solde – montant ; }
    public float consulterSolde ( )          { return solde ; }
    public void afficher()
    {
       System.out.println ("Compte : " + numero + " solde: " + solde) ;
    }
}
```

Utilisation de la classe Compte

```
public class Banque
{
    static public void main (String args [])
    {
        Compte co1 = new Compte () ;
        Compte co2 = new Compte () ;

        co1.initialise (1234,1000f) ; co2.initialise (5678,500f) ;
        co1.deposer (2100.95f) ; co1.afficher () ;
        co2.retirer (1000.0f) ; co2.afficher () ;
    }
}
```

Représentation mémoire



Représentation mémoire

- **co1 et co2 contiennent l'adresse des zones mémoires allouées par l'opérateur *new* pour stocker les informations relatives à ces objets.**
- **co1 et co2 sont des *références*.**
- **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).**

Constructeur de classe

- **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 de constructeur

```
class Compte
{
   public Compte (int num,float s)
   {
      numero = num ;
      solde = s ;
   }
....
}
```

```
Compte co1 = new Compte (1234, 1000,00f) ;
```

Destructeur de classe

- **Un destructeur peut être appelé lorsqu'un objet est détruit.**
- **Le destructeur peut être utilisé pour libérer des ressources spécifiques(déconnexion d'une base de données, fermeture d'un fichier, …). Il sera appelé lorsque le garbage collector récupèrera la mémoire.**
- **Un destructeur est une méthode:**

  public void finalize ()

- **Remarque: on peut forcer l'appel du garbage collector:**

```
Runtime r = Runtime.getRuntime();
r.gc();
```

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++)**

Exemples de surcharge de méthodes

- **Exemple 1:**

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

Exemples de surcharge de méthodes

- **Exemple 2 :**

```
public class Circle
{
      private 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 1 :**

    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**

Comparaison d'objets

- **Exemple 2:**

  Comparaison de chaînes de caractères:

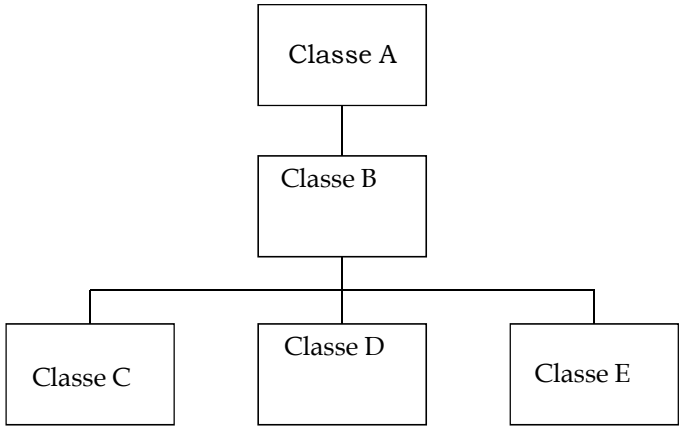
  ```
  String s1 = "Bonjour" ;
  String s2 = "Bonjour" ;

  if (s1.equals (s2))    // Compare le contenu de s1 et s2.

  if (s1.equalsIgnoreCase (s2))      // Compare le contenu de s1 et s2
                                     // sans tenir compte des majuscules
                                     // et minuscules.
  ```

- Remarque: dans le cas des chaînes de caractères, il est malgré tout possible d'utiliser: *if (s1 == s2)*

Héritage, interface, package, droits d'accès

Héritage de classe

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



Héritage de classe

- **Le mot clé pour déclarer une classe dérivée est *extends*.**
- **Java ne supporte que l'héritage simple.**

Exemple d'héritage (1/3)

```
// Animal.java
   public class Animal
  {
     private 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");}
  }
```

Exemple d'héritage (2/3)

```
// 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");}
  }
```

Exemple d'héritage (3/3)

```
// 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

La classe de base

- Toute classe java hérite implicitement de la classe *java.lang.Object*.

- Quelques méthodes de la classe *java.lang.Object*:
  - *public boolean equals(Object obj) ;*
  - *public String toString() ;*

Cas particuliers de l'héritage

- **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.**

Héritage: mot clé "*super*"

- **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 ( String title)
    {
     super ( title) ;          // Appel du constructeur de Frame
                               // Si cet appel est utilisé, c'est toujours
                               //  la première instruction du constructeur
    ...
    }
}
```

Héritage: mot clé "*super*"

- **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.**
- **Le mot clé *interface* remplace le mot clé *class* en tête de déclaration.**
- **Une interface ne peut contenir que des variables constantes ou statiques et des entêtes de méthodes.**
- **Toutes les signatures de méthodes d'une interface ont une visibilité publique.**
- **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 d'imposer un comportement à une classe**
- **Une classe peut implémenter autant d'interfaces qu'elle le souhaite.**

Exemple d'interface

```
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 classes java stockées dans la bibliothèques de classe ne sont pas automatiquement disponibles.**
- **Les packages sont des collections de classes et d'interfaces regroupées par thème.**
- **Une application peut utiliser des classes et interfaces prédéfinies par importation du package concerné.**
- **Le mot clé pour importer un package est *import***
- **De nouveaux packages peuvent être définis rendant le langage très extensible (exemple : package java3d pour des classes graphiques en 3 dimensions).**

Quelques packages de base

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

Exemple de manipulation de packages

Exemple d'utilisation de packages:

```
// Le package java.lang.* est importe implicitement
import java.awt.* ;
import java.util.* ;
```

Exemple de création d'un nouveau package :

```
package monpackage ;
import java.awt.* ;

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

Droits d'accès

- **Toutes les méthodes et données membres définies au sein d'une classe sont utilisables par toutes les méthodes de la classe.**
- **Lors de la conception d'une classe, il faut décider des méthodes/variables qui seront visibles à l'extérieur de cette classe.**
- **Java implémente la protection des 4 P (public, package, protected, private).**

La protection des 4 P

- *private* : visible uniquement au sein de la classe.
- *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.

Variables de classe

- **Une variable de classe est une variable associée à une classe et non aux instances de cette classe.**
- **Un seul exemplaire d'une variable de classe ne peut exister, indépendamment du nombre d'instances de cette classe créées.**
- **Une variable *static* est partagée par toutes les instances de la classe.**
- **Les variables de classe sont préfixées par le mot clé *static*.**
- **On accède à une variable de classe en faisant précéder le nom de cette variable par le nom de la classe suivi d'un point.**

Exemples de variables de classe

Exemple 1:

```
class Alien
{
        static int counter ;

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

Exemple 2:

La classe java.lang.Math contient la valeur de PI

```
class Math
{
        public static final double PI=3.141592653589793d ;
        ...
}
```

Utilisation: ***Math.PI***

Méthodes de classe

- **Une méthode de classe est une méthode associée à une classe et non à une instance de cette classe.**
- **Les méthodes de classe sont préfixées par le mot clé *static*.**
- **Une méthode de classe est appelable sans avoir à créer d'objet de cette classe.**
- **Une méthode de classe est appelée en préfixant le nom de la méthode par le nom de la classe suivi d'un point.**

Exemples de méthodes de classe

Exemple 1 :
La classe java.lang.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 ; }
}
```

Utilisation: *MathStuff.halfInt (10) ;*

Conversion de types, gestion des exceptions

Conversions de type

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

  class java.lang.String
  class java.lang.Integer
  class java.lang.Long
  class java.lang.Float
  class java.lang.Double
  ...

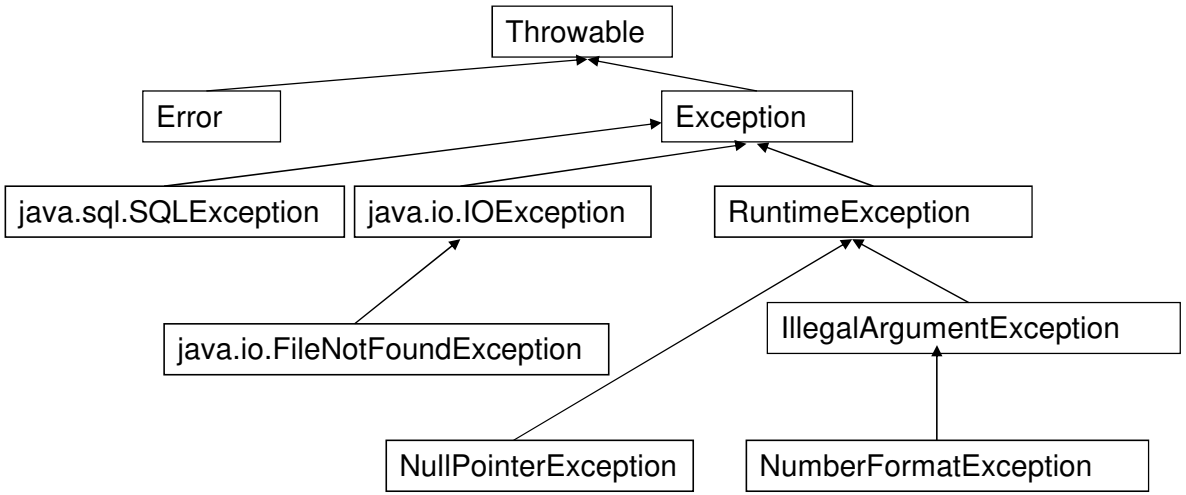
Exemples de conversion de type

```
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 clause *try/catch* .**
- **L'utilisation d'une clause *try/catch* est obligatoire pour les exceptions sous contrôles (une exception sous contrôle est une sous classe de *Exception* mais pas de *RunTimeException*).**
- **Le concepteur d'un programme java a un devoir de captage des exceptions pour ne pas désemparer les utilisateurs.**

Aperçu des classes d'exception



La clause try/catch

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

Exemples de gestion d'exceptions

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

```
String chaine ;
int valeur ;
      ...
try {
       valeur = Integer.parseInt (chaine) ;
       FileInputStream fichier = new FileInputStream ("c:\\texte.txt " );
     } catch (NumberFormatException e1)
       {
              System.err.println ("mauvais format ") ;
       }
       catch (FileNotFoundException e2)
      {
         System.err.println ("Fichier non trouve " );
      }
```

Gestion d'exceptions

- Depuis la version 1.7, on peut gérer plusieurs exceptions dans une seule clause catch:

```
try {
                valeur = Integer.parseInt (chaine);
                FileInputStream fichier = new FileInputStream ("c:\\texte.txt") ;
    } catch (NumberFormatException | FileNotFoundException e)
      {
                System.err.println (« Une erreur est survenue ");

      }
```

Génération d'exception

- **On indique qu'une méthode m peut générer une exception *MyException* par le mot clé *throws* (obligatoire pour les exceptions sous contrôle)**

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

- **On peut déclencher une exception grâce au mot clé *throw***

Exemple:

```
if (x<0)
{
   throw new IllegalArgumentException ("x doit etre positif");
   // ne mettre aucune instruction en dessous du throw
}
```

Exemple de génération d'exception

```
class Test
{
   public String getNom (String key) throws NullPointerException
   {
      if (key == null)  throw new NullPointerException ("cle nulle") ;
      else return "OK" ;
   }
}
public class Exemple
{
   public static void main (String args [])    { new Exemple () ;  }
   public  Exemple ()
   {
      Test test = new Test () ;
      try {
            System.out.println (test.getNom (null)) ;
         } catch (NullPointerException e)
         {
            System.err.println (e.getMessage ()) ;
         }
   }
}
```

Création de ses propres exceptions

```
public class Pile
{
   private int table [] ;
   private int hauteur = 0 ;

   public Pile ()         {  table = new int [3] ; }
   public Pile (int h)   { table = new int [h] ; }

   public void insertValue (int valeur) throws PileException
   {
     if (hauteur == table.length) throw new PileException ("Pile pleine") ;
     else table [hauteur++] = valeur ;
   }
   public int removeValue () throws PileException
   {
     if (hauteur == 0) throw new PileException ("Pile vide") ;
     else return table [--hauteur] ;
   }
}
```

Classe PileException et utilisation

```
public class PileException extends Exception
{
      public PileException(String m)
      {
         super (m) ;
      }
}
```

Utilisation:

```
  Pile pile = new Pile () ;
    try {
                System.out.println (pile.removeValue()) ;
        } catch (PileException e)
       {
          System.out.println (e.getMessage()) ;
       }
```

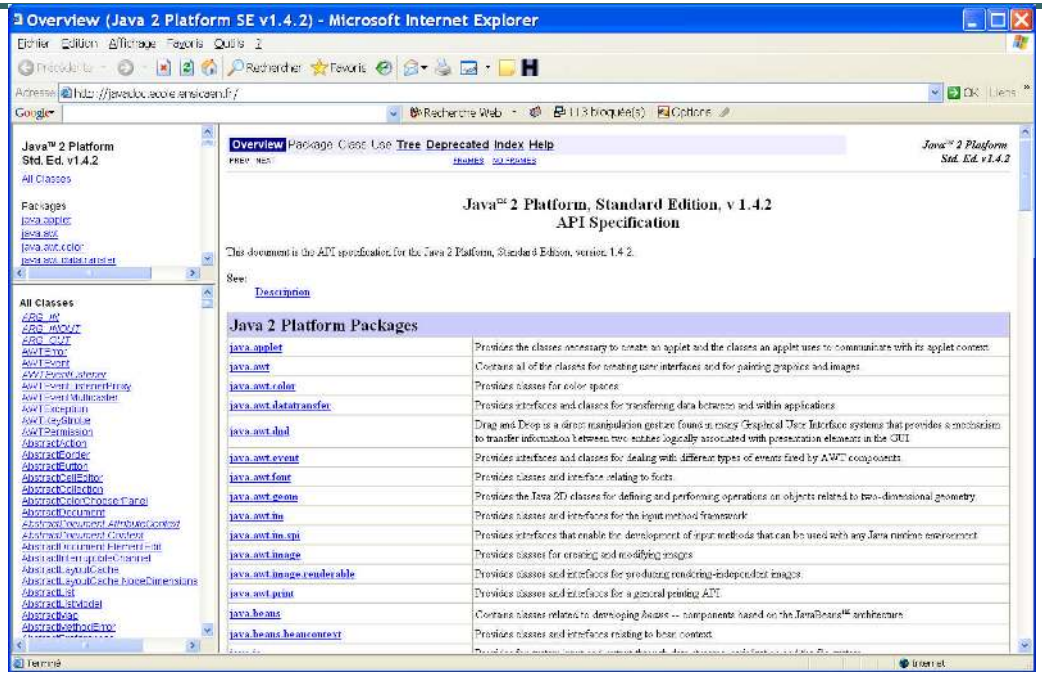
javadoc:

le générateur de documentation

Généralités sur javadoc

- Outils présent dans la jdk de Sun.
- Par défaut, la documentation générée est au format HTML.
- Toute la documentation des API java de SUN a été générée grâce à javadoc.
- La documentation générée contient les fichiers suivants:
  - Un fichier html par classe ou interface contenant le détail de chaque classe ou interface.
  - Un fichier html par package
  - Un fichier *overview-summary.html*
  - Un fichier *deprecated.html*
  - Un fichier *serialized-form.html*
  - Un fichier *overview-frame.html*
  - Un fichier *all-classes.html*
  - Un fichier *package-summary.html* pour chaque package.
  - Un fichier *package-frame.html* pour chaque package.
  - Un fichier *package-tree.html* pour chaque package.

Exemple de génération



Syntaxe des commentaires javadoc

- Commentaire javadoc:

```
/**
 *  Voilà un exemple de <B> commentaire </B>
 *  javadoc
 */
```

- javadoc utilise des "tags" pour préciser le rôle de certains composants d'un élément:
  - "tag" standard: *@tag*
  - "tag" remplacé par une valeur: *{@tag}*
  - Liste complète des tags et l'utilisation de javadoc à l'adresse *http://java.sun.com/j2se/javadoc/*

Quelques "tags" de javadoc

@author	Spécifie l'auteur de l'élément	classe et interface	1.0
@deprecated	Spécifie que l'élément est déprécié	package, classe, interface, champ	1.1
{@docRoot}	Représente le chemin relatif du répertoire de génération de la documentation		1.3
@exception	Spécifie une exception qui peut être levée par l'élément	méthode	1.0
{@link}	Spécifie un lien vers un élément de la documentation dans n'importe quel texte	package, classe, interface, méthode, champ	1.2
@param	Spécifie une paramètre de l'élément	constructeur, méthode	1.0
@see	Spécifie un élément en relation avec l'élément documenté	package, classe, interface, champ	1.0
@since	Spécifie depuis quelle version l'élément a été ajouté	package, classe, interface, méthode, champ	1.1
@throws	identique à @exception	méthode	1.2
@version	Spécifie le numéro de version de l'élément	classe et interface	1.0
@return	Spécifie la valeur de retour d'un élément	méthode	1.0

Exemple d'utilisation de "tags"

```
/**
 * Commentaire sur le role de la methode
 * @param val la valeur a traiter
 * @since 1.0
 * @return Rien
 * @deprecated Utiliser la nouvelle methode XXX
 */
  public void maMethode(int val) { }
```

---

maMethode

```
public void maMethode(int val)
```

Deprecated. *Utiliser la nouvelle methode XXX*

Commentaire sur le role de la methode

Parameters:
`val` - la valeur a traiter
Returns:
Rien
Since:
1.0

Le format d'archive JAR

Le format d'archive jar

- Les fichiers archives rassemblent et compressent plusieurs classes java dans un seul fichier.
- Les fichiers archives peuvent être signés numériquement.
- 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

- Utilisation d'un fichier jar pour une applet:

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

- Utilisation d'un fichier jar pour une application:

  - Il faut spécifier le nom de la classe principale dans un fichier *META-INF/MANIFEST.MF*

    Manifest-Version: 1.0
    Main-Class: bibliotheque.Main

    ligne blanche obligatoire

  - Archiver le fichier manifest.mf et toutes les ressources et classes de l'application:
    jar cvfm bibli.jar META-INF/MANIFEST.MF bibliotheque/.class*

  - Lancement de l'application: java -jar bibli.jar

Gestion de fichiers, flux d'entrées/sortie, sérialisation

Gestion de fichiers

- La gestion de fichiers se fait par l'intermédiaire de la classe *java.io.File*.
- Cette classe possède des méthodes qui permettent d'interroger ou d'agir sur le système de fichiers du système d'exploitation.
- Un objet de la classe *java.io.File* peut représenter un fichier ou un répertoire.

Quelques méthodes de la classe *java.io.File*

File (String name)
File (String path, String name)
File (File dir, String name)
boolean isFile ()
boolean isDirectory ()
boolean mkdir ()
boolean exists ()
boolean delete ()
boolean canWrite ()
boolean canRead ()
File getParentFile ()
long lastModified ()
String [] list ()

Exemple d'utilisation de la classe *java.io.File*

```
import java.io.* ;
public class ExempleFile
{
    static public void main (String args [])  {  new ExempleFile () ; }
    ExempleFile () {    liste (new File ("c:\\")) ; }
    private void liste (File dir)
    {
       if (dir.isDirectory () == true)
       {
          String fichiers [] = dir.list () ;
          for (int i = 0 ; i != fichiers.length ; i++) System.out.println (fichiers [i]) ;
       }
      else
      {
         System.err.println (dir + " n'est pas un repertoire") ;
      }
   }
}
```

Les flux

- **Difficulté d'un langage d'avoir un bon système d'entrées/sorties.**
- **Beaucoup de sources d'E/S de natures différentes (console, fichier, socket,…).**
- **Beaucoup d'accès différents (accès séquentiel, accès aléatoire, mise en mémoire tampon, binaire, caractère, par ligne, par mot, etc.).**
- **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**

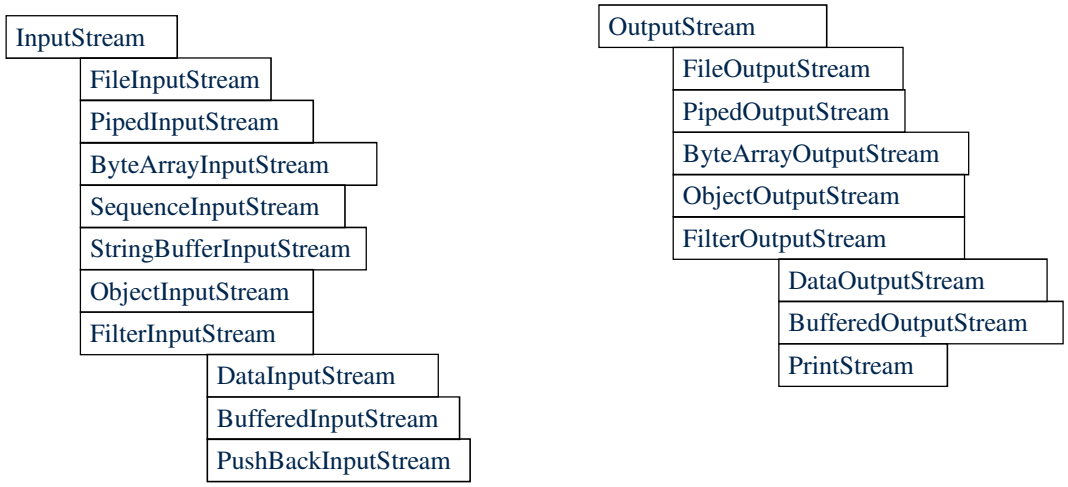
Les flux proposés par java

- **Flux d'entrée/sortie de bytes.**
- **Flux d'entrée/sortie de caractères depuis la version 1.1 de java.**
- **Toutes les classes d'entrée/sortie sont dans le package *java.io***
- **Toutes les méthodes peuvent générer une *java.io.IOException***

Classes de base abstraites des flux

	Flux d'octets	Flux de caractères
Flux d'entrée	*java.io.InputStream*	*java.io.Reader*
Flux de sortie	*java.io.OutputStream*	*java.io.Writer*

Classes de flux de bytes



La classe java.io.InputStream

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

La classe java.io.InputStream

- **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 markSupported () ;***
- **Marquage d'un flux :** ***public void mark (int readlimit) ;***
- **Revenir sur la marque:** ***public void reset () ;***
- **Fermer un flux :** ***public void close () ;***

Exemple de flux d'entrée

```
import java.io.* ;
public class LitFichier
{
  public static void main (String args [])
  {
    try {
          InputStream s = new FileInputStream ("c:\\temp\\data.txt") ;
          byte buffer [ ] = new byte [s.available()] ;
          s.read (buffer) ;
          for (int i = 0 ; i != buffer.length ; i++)
                System.out.print ( (char) buffer [i]) ;
         s.close () ;
       } catch (IOException e)
         {
           System.err.println ("Erreur lecture") ;
         }
  }
}
```

La classe java.io.OutputStream

- **Les méthodes d'écriture :**

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

- **Nettoyage d'un flux, forçant l'écriture des données bufférisées :**

  public void flush () ;

- **Fermeture d'un flux**

  public void close () ;

Exemple de flux de sortie

```
import java.io.* ;
public class EcritFichier
{
  static public void main (String args [])
  {
        String Chaine = "Bonjour" ;
        try {
              FileOutputStream f = new FileOutputStream ("c:\\temp\\data.txt") ;
              f.write (Chaine.getBytes ()) ;
              f.close () ;
           } catch (IOException e)
         {
              System.err.println ("Erreur ecriture") ;
         }
  }
}
```

Les classes FilterInputStream/ FilterOutputStream

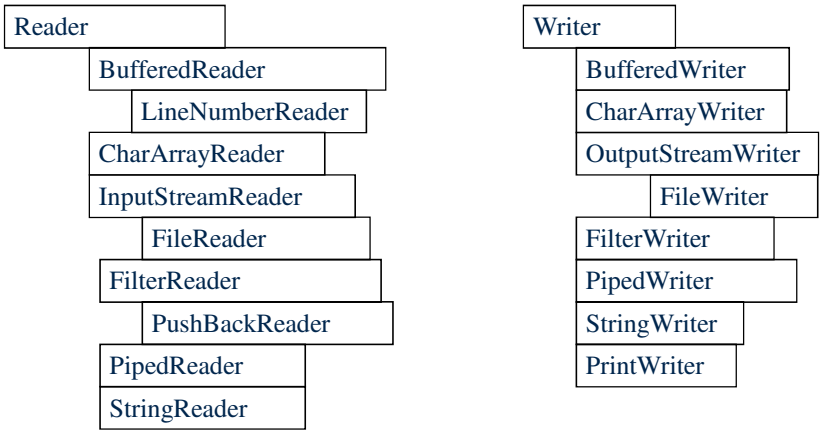
- **Ces deux classes servent de classes de base à des classes de gestion d'entrées/sorties plus évoluées:**
  - ***BufferedInputStream*** **et** ***BufferedOutputStream*** **permettent de lire et écrire des données à travers un tampon de lecture/écriture pour améliorer les performances.**
  - ***DataInputStream*** **et** ***DataOutputStream*** **permettent de lire/écrire des données formatées (byte, int, char, float, double, etc.)**
  - **etc.**

Exemples de lecture/écriture évoluée

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

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

Les classes de flux de caractères



Exemple de BufferedReader

```
import java.io.*;
public class TestBufferedReader
{
   public static void main(String args[])
   {
      try {
               String ligne ;
                BufferedReader fichier = new BufferedReader (new FileReader("data.txt"));
                while ((ligne = fichier.readLine()) != null)  System.out.println(ligne);
                fichier.close();
          } catch (IOException e)
            {
               System.err.println ("Erreur lecture") ;
            }
   }
}
```

La sérialisation

- La sérialisation est un mécanisme permettant de rendre un objet persistant. Il peut être ensuite:
  - Stocké dans un fichier
  - Transmis sur le réseau (exemple: RMI)
  - …
- Le processus inverse est la désérialisation.
- Un objet sérialisé est dit persistant.
- Cette fonctionnalité est apparue dans la version 1.1 de Java.

Pourquoi sérialiser ?

- **Rendre un objet persistant nécessite une convention de format pour la lecture/écriture (cela peut être une opération complexe et difficile à maintenir) ou transmettre un objet via le réseau à une application distante.**
- **La sérialisation permet de rendre un objet persistant de manière simple et naturelle.**
- **Si un objet contient d'autres objets sérialisables, ceux-ci seront automatiquement sérialisés.**
- **La plupart des classes de base (mais pas toutes) du langage Java sont sérialisables.**
- **Si la classe a été modifiée entre la sérialisation et la désérialisation, l'exception *java.io.InvalidClassException* est déclenchée.**

Comment sérialiser

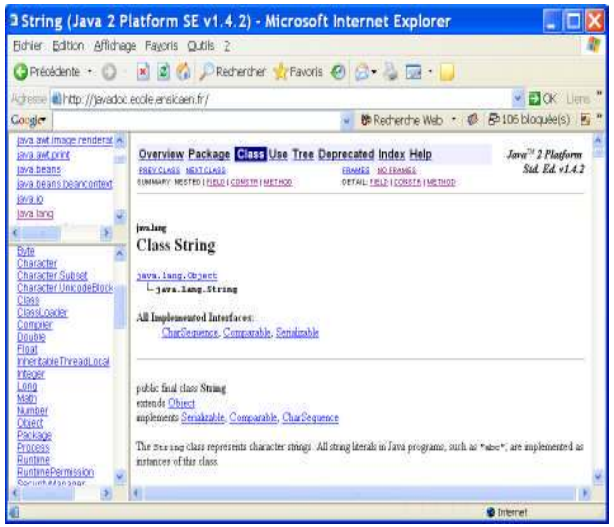
- **Une classe est sérialisable si elle implémente l'interface *java.io.Serializable*.**
- **Des objets de type *java.io.ReadObjectStream* et *java.io.WriteObjectStream* vont permettre de sérialiser/désérialiser.**
- **Les données membres que l'on ne souhaite pas sauvegarder doivent être déclarées *transient*.**
- **Des méthodes de lecture/écriture peuvent être redéfinies le cas échéant:**

  private void writeObject (java.io.ObjectOutputStream out) throws IOException ;

  private void readObject (java.io.ObjectInputStream in) throws IOException,ClassNotFoundException ;

Exemple
Sérialisation/Désérialisation

```
import java.io.* ;

class Info implements Serializable
{
   private String Nom = "" ;
   private String MotPasse = "" ;

   public Info(String n, String m)
   {
     Nom=n ; MotPasse = m ;
   }

   public String getNom () { return
     Nom ; }
   public String getPassword () {
     return MotPasse ; }
}
```



Exemple
Sérialisation/Désérialisation

```
public class ExempleSerialisation
{
  static public void main (String args [])
  {
    new ExempleSerialisation () ;
  }

  public ExempleSerialisation ()
  {
    Info User = new Info ("Pierre","password") ;
    Ecrire (User) ;
    User = Lire () ;
    if (User != null)
    System.out.println ("nom = " +
        User.getNom () + " mot de passe = " +
        User.getPassword ()) ;
  }
```

```
void Ecrire (Info user)
 {
   try {
         FileOutputStream file = new
           FileOutputStream ("c:\\travail\\info.txt") ;
         ObjectOutputStream out = new
           ObjectOutputStream (file) ;
         out.writeObject (user) ;
         out.flush () ;
         out.close () ; file.close () ;
       } catch (IOException ex)
         {
           System.err.println ("Erreur d'ecriture " + ex) ;
         }
  }

// Fin classe ExempleSerialisation transparent
// suivant
```

Exemple
Sérialisation/Désérialisation

```
        Info Lire ()
        {
          Info User = null ;
          try {
                FileInputStream file = new FileInputStream ("c:\\travail\\info.txt") ;
                ObjectInputStream in = new ObjectInputStream (file) ;
                User = (Info) in.readObject() ;
             } catch (Exception ex)
               {
                 System.err.println ("Erreur de lecture " + ex) ;
               }
           return User
        }
} // Fin classe ExempleSerialisation
```

Le package java.nio

- Nouveau package de gestion des entrées/sorties introduit par la version 1.4.
- NIO permet d'utiliser des entrées/sorties plus rapides en gérant des blocs plutôt que des bytes ou caractères.
- Les objets de base pour ce type d'entrée sortie sont: *Buffer* et *Channel.*
- Un "*Buffer*" contient les données à lire ou écrire, un "*Channel*" désigne la source ou la destination de l'information.
- Une entrée/sortie sur un objet "*Channel*" transite obligatoirement par un objet "*Buffer*".

L'objet "*Buffer*"

- Un "*Buffer*" est un objet contenant les informations à écrire ou recevant les informations lues.
- Le même objet "*Buffer*" peut être utilisé en entrée et en sortie.
- Outre le stockage, un objet "Buffer" fournit des informations sur l'opération d'entrée sortie.

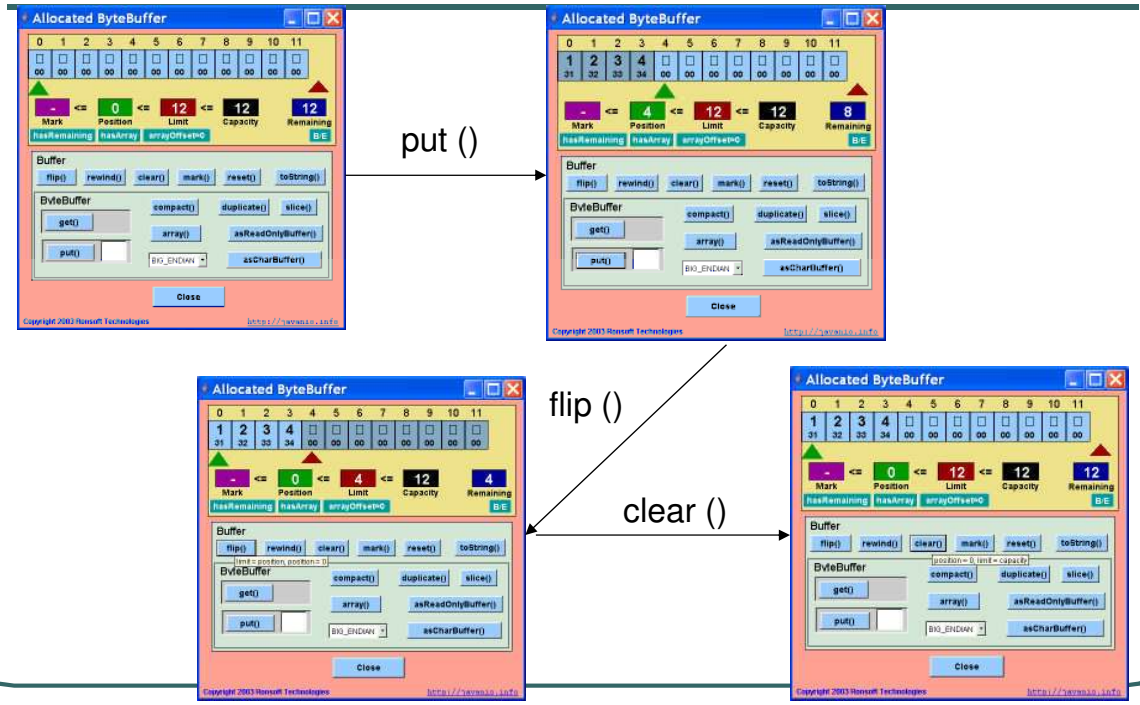
Les variables d'état d'un "*Buffer*"

- position
  - indique la 1ère position libre dans la zone de stockage.
- limit
  - quantité d'information restant à envoyer (écriture) ou espace restant disponible (lecture)
- capacity
  - taille maximale d'information pouvant être stockée dans un objet buffer

Quelques méthodes de "*Buffer*"

- *flip ()*
  - positionne "limit" à "position" et remet "position" à 0.
- *clear ()*
  - positionne "limit" à capacity, remet "position" a 0 et efface la marque.
- mark ()
  - Utilise "position" comme la marque courante
- reset ()
  - Place "position" à la marque courante
- rewind ()
  - Place 0 dans "position" et efface la marque
- Les différentes forme des méthodes *get()* et *put ()* permettent d'obtenir ou d'initialiser la zone de stockage d'un objet "*Buffer*".

Fonctionnement d'un "ByteArrayBuffer"



Quelques méthodes de manipulation d'un objet "*Buffer*"

- Allocation d'un buffer:

  ByteBuffer buffer = ByteBuffer.allocate (512) ;
  ou
  byte array [] = new byte [512] ;
  ByteBuffer buffer = ByteBuffer.wrap (array) ;

- Decoupage d'un buffer

  ByteBuffer buffer = ByteBuffer.allocate (10) ;
  buffer.position (3) ; buffer.limit (7)
  ByteBuffer slice = buffer.slice () ;

Exemple java.nio

```
import java.io.*;
import java.nio.*;
import java.nio.channels.*;

public class Test1Nio
{
    Test1Nio()
    {
       try
          {
                FileInputStream fin = new FileInputStream("d:\\travail\\image.jpg");
                FileOutputStream fout = new FileOutputStream("d:\\travail\\image2.jpg");
                ByteBuffer buffer = ByteBuffer.allocate(512);
                FileChannel fcin = fin.getChannel();  FileChannel fcout = fout.getChannel();
                while (fcin.read(buffer) != -1)
                {
                     buffer.flip() ;  fcout.write(buffer); buffer.clear () ;
                 }
                fin.close () ; fout.close () ;
                } catch (Exception e) { System.err.println(e); }
    }
    public static void main(String[] args) { new Test1Nio(); }
}
```

Différents types de buffer

- ByteBuffer
- CharBuffer
- IntBuffer
- LongBuffer
- FloatBuffer
- DoubleBuffer

Quelques "*Channel*"

- FileChannel
- ServerSocketChannel
- SocketChannel
- DatagramChannel
- SelectableChannel

Les threads

Les threads

- **Un thread est une unité d'exécution au sein d'un même processus (ce n'est pas un autre processus).**
- **Tous les threads d'un même processus partagent la même zone mémoire.**
- **La programmation multithreads donne l'illusion de la simultanéité.**
- **La gestion des threads est dépendante de la JVM (problème pour la portabilité). Les threads peuvent être préemptifs ou coopératifs.**
- **Un thread possède un nom et une priorité.**
- **Un thread s'exécute jusqu'au moment où:**
  - **Un thread de plus grande priorité devient exécutable.**
  - **Une méthode *wait ()*, *yield ()* ou *sleep ()* est lancée.**
  - **Son quota de temps a expiré dans un système préemptif.**

Création d'un thread

- **Une classe est un thread si elle remplit une des deux conditions:**
  - **Elle étend la classe *java.lang.Thread***
  - **Elle implémente l'interface *java.lang.Runnable***
- **Le corps du thread est contenu dans une méthode:**

  public void run ()
- **Un thread est lancé par appel d'une méthode *start ()***

Premier exemple de création d'un thread

```
class MyThread extends Thread
{
        // Constructeur,données membres, méthodes éventuels
        ...
        public void run ()
        {
            // corps du thread
        }
}


MyThread thread = new MyThread () ;      // Création du thread
thread.start () ;                        // Appelle la méthode run ()
```

Deuxième exemple de création d'un thread

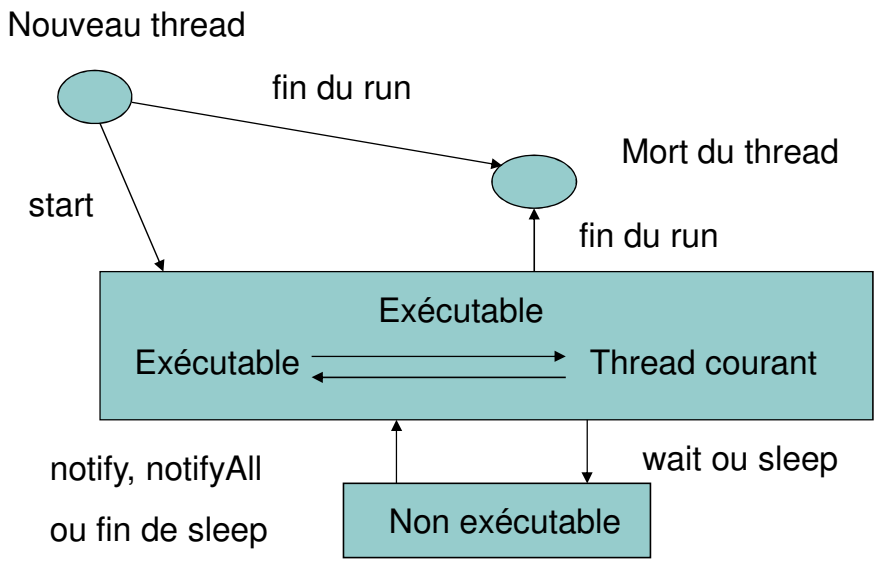
```
class MyClass extends classe implements Runnable
{
   // Constructeur,données membres, méthodes éventuels
    ...
   public void run ()
   {
            // corps du thread
   }
}


MyClass objet = new MyClass () ;        // Création d'un objet MyClass
Thread thread = new Thread (objet) ;    // Récupération du thread
thread.start () ;                       // Appelle la méthode run ()
```

Quelques méthodes de la classe java.lang.Thread

start ()	*Rend un thread exécutable en lançant la méthode run ().*
sleep (i)	*Endort le thread pour i millisecondes.*
*wait ()**	*Suspend le thread.*
*notify ()**	*Place le thread dans un état exécutable.*
*notifyAll ()**	*Réveille tous les threads en attente.*
yield ()	*Place le thread de l'état « en cours d'exécution » à l'état « exécutable ».*
setPriority (i)	*Modifie la priorité d'un thread (i est compris entre MIN_PRIORITY et MAX_PRIORITY).*
join () *join (long)*	*Pour qu'un deuxième thread attende la fin d'exécution d'un premier thread, il suffit d'appeler la méthode join sur le premier thread. Un paramètre de temps (en millisecondes) peut être spécifié.*

* Méthodes héritées de la classe *java.lang.Object*

Cycle de vie d'un thread



Arrêt d'un thread

- La méthode *stop ()* est dépréciée.
- Un thread s'arrête lorsqu'il n'y a plus d'instruction à exécuter dans la méthode *run ()*.
- Une solution possible:

```
public class ThreadTest extends Thread
    {
        private boolean bKillThread = false;

        public void run()
        {
            while (bKillThread == false)
                    System.out.println( getName() ) ;
        }

        public void stopThread()
        {
            bKillThread = true;
        }
    }
```

Synchronisation

- **Plusieurs threads accédant à une même donnée doivent être synchronisés**
- **La synchronisation peut se faire sur un objet (pas de synchronisation possible pour une variable d'un type de base).**
- **Mot clé: *synchronized***
- **Si un thread invoque une méthode "*synchronized*" d'un objet, celui-ci est verrouillé pour toutes ses méthodes "*synchronized*".**

Premier exemple de synchronisation

```
class Compteur
{
   private int valeur ;
  // Si plusieurs threads dispose d'une référence sur un
   // objet de classe « Compteur », un seul thread à la fois
   // pourra pénétrer dans la méthode incrémente de cet objet.
   synchronized void incremente ()
   {
      valeur += 1 ;
   }

   int Combien ()
   {
      return valeur ;
   }
}
```

Deuxième exemple de synchronisation

```
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 : " + safeX + safeY) ;
   }
}
```

Synchronisation sur 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.
- Il faut dans ce cas protéger une classe et pas seulement une instance de classe.

Exemple de synchronisation sur une variable de classe

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

- *public final Class getClass():* renvoie la classe de l'objet.
- *Tous* les objets de classe C*ompteur* seront bloqués dans la méthode *incremente().*

Les collections

Les collections

- Les collections sont des objets permettant de gérer des ensembles d'objets avec éventuellement la possibilité de gérer les doublons, les ordres de tri, etc.

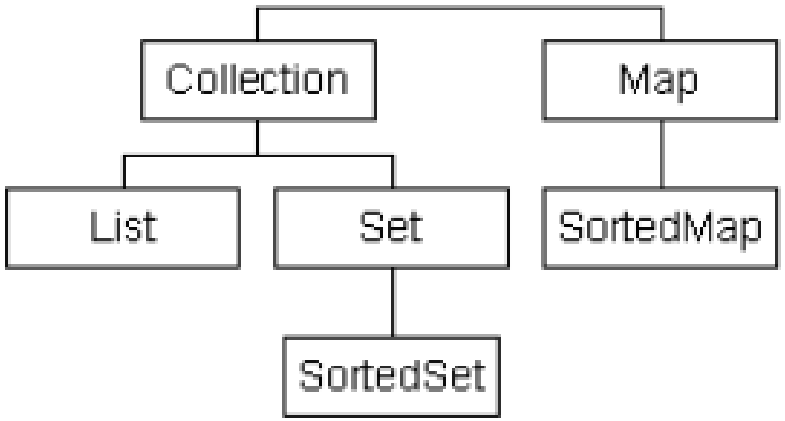
- La version 1 de Java proposait:
  - *java.util.Vector, java.util.Stack, java.util.Hashtable*
  - Une interface *java.util.iterator* permettant de parcourir ces objets

Exemple de la collection "stack"

```
package teststack;
import java.util.*
public class ExempleStack
{
	private Stack pile ;

	public ExempleStack ()
	{
		pile = new Stack () ;
		pile.push("Je suis ") ; pile.push("Un exemple ") ; pile.push("de pile") ;
		Iterator iter = pile.iterator () ;
		while (iter.hasNext())
		{
			System.out.println (iter.next()) ;
		}
	}
	public static void main(String[] args)
	{
		new ExempleStack () ;
	}
}
```

```
Je suis
Un exemple
de pile
```

Interfaces de collections



Collection à partir de java 2

- *Collection* : interface qui est implémentée par la plupart des objets qui gèrent des collections.
- *Map* : interface qui définit des méthodes pour des objets qui gèrent des collections sous la forme clé/valeur
- *Set* : interface pour des objets qui n'autorisent pas la gestion des doublons dans l'ensemble
- *List* : interface pour des objets qui autorisent la gestion des doublons et un accès direct à un élément
- *SortedSet* : interface qui étend l'interface *Set* et permet d'ordonner l'ensemble
- *SortedMap* : interface qui étend l'interface *Map* et permet d'ordonner l'ensemble

Implémentation des interfaces

Interface	*Implémentation*
Set	HashSet
SortedSet	TreeSet
List	ArrayList, LinkedList, Vector
Map	HashMap, Hashtable
SortedMap	TreeMap

Exemple "TreeMap"

```
package exempletreemap;
import java.util.* ;
public class ExempleTreeMap
{
	public ExempleTreeMap ()
	{
		TreeMap tree = new TreeMap () ;
		tree.put ("zzzz",new Integer (26));
		tree.put ("aaaa", new Integer (1)) ;
		tree.put ("bbbb", new Integer (2)) ;

		Iterator itercle = tree.keySet().iterator() ;
		Iterator itervaleurs = tree.values().iterator() ;
;		while (itercle.hasNext())
		{
			System.out.println (itercle.next() + " --> " + itervaleurs.next()) ;
		}
	}
	public static void main(String[] args)
	{
		new ExempleTreeMap () ;
	}
}
```

```
aaaa --> 1
bbbb --> 2
zzzz --> 26
```

Collections et threads

- Si plusieurs threads peuvent accéder à un objet collection, il y a nécessité de synchroniser avec une des méthodes statiques de la classe java.util.Collections:

```
static Collection  synchronizedCollection (Collection c)
static List  synchronizedList (List list)
static Map  synchronizedMap (Map m)
static Set  synchronizedSet (Set s)
static SortedMap  synchronizedSortedMap (SortedMap m)
static SortedSet  synchronizedSortedSet (SortedSet s)
```

- Les méthodes précédentes ne synchronisent pas les itérateurs. Il faut donc le faire manuellement:

```
synchronized (objet)
{
        Iterator iter = objet.iterator () ;
        {
            // travailler avec l'iterateur
        }
}
```

Collections et threads

- Modifications de l'exemple précédent:

```
SortedMap tree = Collections.synchronizedSortedMap(new TreeMap ()) ;

…..

synchronized (tree)
{
        Iterator itercle = tree.keySet().iterator() ;
        Iterator itervaleurs = tree.values().iterator() ;
        while (itercle.hasNext())
        {
          System.out.println (itercle.next() + " --> " + itervaleurs.next()) ;
        }
}
```

Une table particulière: *java.util.Properties*

- La classe *java.util.Properties* est une table de hachage pour définir des variables d'environnement sous la forme (nom_variable, valeur)
- Exemple:

```
Properties props = new Properties () ;
props.put ("monApp.xSize","50") ;
```

- La méthode statique *System.getProperties ()* retourne les variables d'environnement définies telles que:

java.vendor
java.home
file.separator
path.separator
user.name
user.home
user.dir
…

Collection et java 1.5

- Jusqu'à la version 1.4, on stockait et récupérait des "*Object*" d'une collection.
- Exemple:

```
ArrayList liste = new ArrayList ()
liste.add (new MaClasse ()) ;
MaClasse obj = (MaClasse) liste.get (0) ;
```

- Depuis la version 1.5, il est recommandé de spécifier la nature des objets stockés.
- Exemple:

```
ArrayList<MaClasse> liste = new ArrayList<MaClasse> () ;
liste.add (new MaClasse ())
MaClasse obj = liste.get (0) ;
```

Les nouveautés de la version 1.5

Nouveautés de la version 1.5

- La version 1.5 (nom de code Tiger) est une évolution majeure du langage Java.
- Les évolutions précédentes (hormis la 1.1) n'apportaient en général que des nouvelles classes internes.

Autoboxing/Unboxing

- L'autoboxing permet de transformer automatiquement une variable de type scalaire en un objet du type correspondant (exemple: int → Integer).
- L'Unboxing est l'opération inverse (exemple: Integer → int).
- Ces propriétés sont définies dans la JSR 201.

Exemple AutoBoxing/UnBoxing

```
import java.util.* ;

public class AutoBoxing
{       public static void main (String args [])
        {
                        new AutoBoxing () ;
        }

        public AutoBoxing ()
        {
             Vector liste = new Vector () ;
             for (int i = 1 ; i <= 5 ; i++) { liste.add (i) ; }  // liste.add (new Integer (i))
             for (int i = 0 ; i < liste.size () ; i++) System.out.println (liste.get (i)) ;
        }
}
```

L'importation statique

- L'importation statique allège l'écriture pour l'accès aux données et aux méthodes statiques d'une classe. Elle est définie par la JSR 201

```
import static java.lang.Math.* ;    // Nouveauté: mot clé static

public class TestStaticImport
{
   public static void main (String args [])
   {
      new TestStaticImport () ;
   }

   public TestStaticImport ()
   {
      System.out.println (PI) ;    // Remplace Math.Pi grâce à l'importation statique
   }
}
```

Boucles évoluées

- La syntaxe des boucles a été simplifiée pour le parcours des éléments d'un tableau ou d'une collection.
- Exemple:

```
int tableau [] = {0,1,2,3,4,5} ;
for (int i : tableau) System.out.println (i) ;
```

Les arguments variables

- Cette fonctionnalité, définie par la JSR 201,permet de transmettre un nombre variable d'arguments d'un type donné à une fonction.

- Les arguments seront transmis comme un tableau. On peut transmettre soit un tableau, soit une liste unitaire mais pas les deux en même temps.

Exemples de liste variable d'arguments

```
public class VarArgs
{

	public static void main(String[] args)
	{
		new VarArgs () ;
	}
	public VarArgs ()
	{
		System.out.println(additionner (1,2,3)) ;
		System.out.println (additionner (4,5,6,7,8,9)) ;
	}
	public int additionner (int ... valeurs)  // …    indique une liste variable d'entiers
	{
		int total = 0 ;
		for (int val : valeurs) total += val ;
		return total ;
	}
}
```

Le type énuméré

- Défini par la JSR 201, le type énuméré permet de définir un ensemble fini de valeurs.

- Exemple de type énuméré:

  public enum MaCouleur { BLEU , BLANC , ROUGE } ;

- Le compilateur créera une classe avec les caractéristiques suivantes:
  - Un champ *static* pour chaque élément de la déclaration.
  - Une méthode *values()* qui renvoie un tableau avec les éléments définis.
  - Une méthode *valueOf(String)* qui retourne la valeur correspondante à la chaîne.
  - La classe implémente les interfaces *Comparable* et *Serializable*
  - Les méthodes *toString(), equals(), hashCode() et compareTo()* sont redéfinies.

Exemple de type énuméré

```
public class TestEnum
{
	private String Objet ;
	private enum MaCouleur { BLEU , BLANC , ROUGE } ;
	private MaCouleur Couleur ;
	public static void main(String[] args)  { new TestEnum ("voiture",MaCouleur.ROUGE) ; }
	public TestEnum (String obj, MaCouleur c)
	{
		Objet = obj ; Couleur = c ;
		afficherObjet () ;
		System.out.println (c) ;   // Affichera "ROUGE"
	}
	void afficherObjet ()
	{
		switch (Couleur)
		{
			case BLEU : System.out.println (Objet + " de couleur bleue") ; break ;
			case BLANC: System.out.println (Objet + " de couleur blanche") ; break ;
			case ROUGE: System.out.println (Objet + " de couleur rouge") ; break ;
			default: System.out.println (Objet + " de couleur inconnue") ;
		}
	}
}
```

Affichage formaté

- La classe *System.out* dispose maintenant d'une méthode *printf* analogue à celle du langage C.
- Les caractères de formatage sont similaires au langage C à l'exception du \n remplacé par %n (pour des questions de portabilité).
- Exemple:

```
public class IOFormatted
{

    public static void main(String[] args)
    {
            new IOFormatted () ;
    }
    public IOFormatted ()
    {
            String chaine = "Bonjour" ;
            int i = 1 ;
            System.out.printf ("la chaine vaut %s%n",chaine) ;
            System.out.printf ("l'entier vaut %10d%n",i) ;
    }
}
```

Saisie formatée

- La classe *java.util.Scanner* permet des entrées formatées similaires à la fonction *scanf* du langage C.
- Exemple:

```
import java.util.* ;
public class IOFormatted
{
      public static void main(String[] args)  { new IOFormatted () ; }
      public IOFormatted ()
      {
          String chaine ;
          int i = 0 ;
          Scanner clavier = new Scanner (System.in) ;
          chaine = clavier.next () ;
          try { i = clavier.nextInt() ;
              } catch (InputMismatchException e)
                {
                        System.err.println("l'entier saisi est incorrect") ;
                }
          System.out.printf ("la chaine vaut %s%n",chaine) ;
          System.out.printf ("l'entier vaut %d%n",i) ;
      }
}
```

Quelques utilisations de *java.util.Scanner*

- Exemple 1 :

```
String chaine ;
Scanner clavier = new Scanner (System.in) ;
try {
        chaine = clavier.next (java.util.regex.Pattern.compile("[Oo]")) ;
   } catch (InputMismatchException e)
   {
        System.err.println ("O ou o attendu") ;
   }
```

- Exemple 2: *Scanner* versus *StringTokenizer*

```
String chaine ;
chaine = "1 test 2 test rouge test bleu test " ;
Scanner s = (new Scanner (chaine)).useDelimiter ("\\stest\\s") ;
System.out.println (s.nextInt ()) ;
System.out.println (s.nextInt ()) ;
System.out.println (s.next ()) ;
System.out.println (s.next ()) ;
```

Les types génériques

- Les types génériques, définis par la JSR 14, permettent de spécifier le type d'objets que l'on va placer dans une collection d'objets (List, Vector)
- Avantages:
  - meilleure lisibilité: on connaît à la lecture du programme quel type d'objets seront placés dans la collection.
  - La vérification peut être fait à la compilation.
  - Le cast pour récupérer un objet de la collection est devenu implicite (sans cette fonctionnalité, il fallait faire un cast explicite, sachant que celui-ci peut échouer mais cela n'était détectable qu'à l'exécution).
- La syntaxe pour utiliser les types génériques utilise les symboles < et >.

Exemple de type générique

```
import java.util.* ;
public class TestGenerique
{
	public static void main(String[] args)  { new TestGenerique () ; }
	public TestGenerique ()
	{
		String chaine,str ;
		boolean bFinBoucle = false ;
		List<String> liste = new ArrayList () ;
		Scanner clavier = new Scanner (System.in) ;
		while (bFinBoucle == false)
		{
			chaine = clavier.next () ;
			if (chaine.equalsIgnoreCase("quit") == false)
				liste.add (chaine) ;    // on ne peut stocker que des Strings
			else bFinBoucle = true ;
		}
		for (Iterator<String> iter = liste.iterator (); iter.hasNext () ;)
		{
			str = iter.next () ; // Pas de cast ici
			System.out.println (str) ;
		}
	}
}
```

Les classes génériques

```
public class TestClasseGenerique
{
	public static void main(String[] args)
	{
		new TestClasseGenerique () ;
	}
	public TestClasseGenerique ()
	{
		new MaClasseGenerique<String,Integer> ("Dupont",33) ;
	}
}

class MaClasseGenerique<T1,T2>
{
	private T1 param1 ;
	private T2 param2 ;
	public MaClasseGenerique (T1 param1,T2 param2)
	{
		this.param1 = param1 ;
	}
	public T1 getParam1 () { return param1 ; }
	public T2 getParam2 () { return param2 ; }
}
```

Les annotations

- Une annotation permet de marquer certains éléments du langage.
- Une annotation pourra ensuite être utilisée à la compilation ou à l'exécution pour automatiser certaines tâches.
- Création d'une annotation:

```
public @interface MonAnnotation
{
}
```

- Utilisation d'une annotation:

```
@MonAnnotation
```

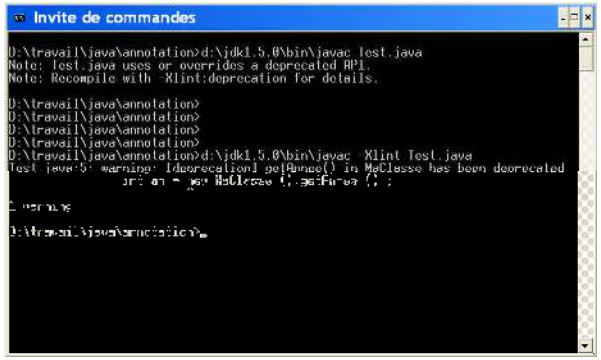
- Une annotation peut être utilisée sur différents types d'éléments du langage: package, class, interface, enum, annotation, méthodes paramètre, champs d'une classe, variables locales

Les annotations standards

@Deprecated	Indique au compilateur que l'élément marqué est déprécié et ne devrait plus être utilisé.
@Override	Indique que l'élément marqué est une surcharge d'une méthode héritée
@SuppressWarnings	Indique au compilateur de ne pas afficher certains messages d'avertissement.

Exemple @Deprecated

```
public class MaClasse
{
        private int annee ;
        public MaClasse () { annee = 2007 ; }

        public int getYear () { return annee ; }

        @Deprecated
        public int getAnnee () { return annee ; }
}

public class Test
{
        static public void main (String args [])
        {
                System.out.println ( new MaClasse ().getAnnee ()) ;
        }
}
```



Exemple @SuppressWarnings

- Liste des avertissements:

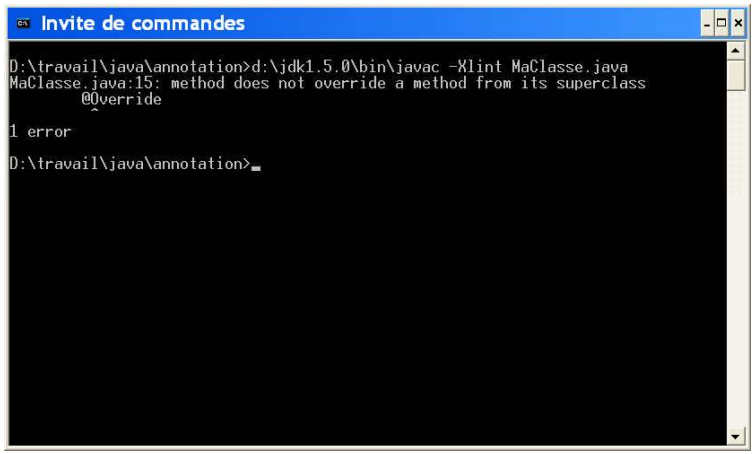
  all,deprecation,unchecked,fallthrough,path,serial,finally,
  deprecation -unchecked,-fallthrough,-path,-serial,-finally

- Exemples:

```
@SuppressWarnings ("deprecation")
public class VieilleClasse { … }

@SuppressWarnings ("deprecation")
public int methode () { … }
```

Exemple @Override

```
public class MaClasse
{
	private int annee ;

	public MaClasse ()
	{
		annee = 2007 ;
	}

	public int getYear ()
	{
		return annee ;
	}

	@Override
	public String ToString ()
	{
		return String.valueOf (annee) ;
	}
}
```



Les meta-annotations

- Une méta annotation permet d'annoter une autre annotation afin d'indiquer au compilateur des informations supplémentaires.

- Les méta annotations sont dans le package *java.lang.annotation*:
  - @Documented
  - @Inherited
  - @Retention
  - @Target

Les méta annotations

@Documented	Indique au générateur de documentation que l'annotation doit être présente dans la documentation.
@Inherited	L'annotation sera héritée par tous les descendants de l'élément marqué (classe uniquement).
@Retention	Indique la durée de vie de l'annotation: *RetentionPolicy.SOURCE* *RetentionPolicy.CLASS (défaut)* *RetentionPolicy.RUNTIME*
@Target	Limite le type d'éléments sur lesquels l'annotation peut être utilisée.

Exemple @Documented

```
public @interface Annotation1
{
}

import java.lang.annotation.Documented ;
@Documented
public @interface Annotation2
{
}

public class MaClasse
{
      …
      @Annotation1
      public int getYear () { return annee ; }

      @Annotation2
      public int getAnnee () { return annee ; }
}
```

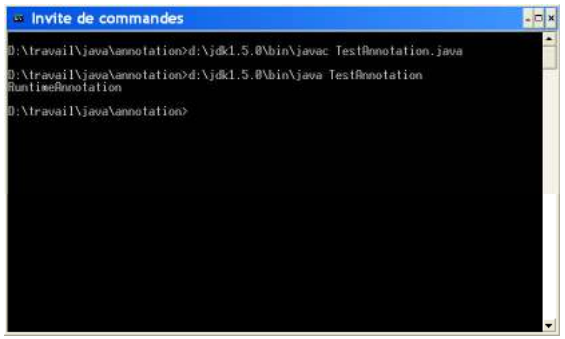
Exemple @Inherited

```
import java.lang.annotation.Inherited ;
@Inherited
public @interface Annotation1
{
}

@Annotation1
public class MaClasse
{
…
}
```

Toutes les classes étendant *MaClasse* hériteront de l'annotation

Exemple @Retention

```
import java.lang.annotation.* ;
@Retention(RetentionPolicy.SOURCE)
@interface SourceAnnotation {}

@Retention(RetentionPolicy.CLASS)
@interface ClassAnnotation {}

@Retention(RetentionPolicy.RUNTIME)
@interface RuntimeAnnotation {}

@SourceAnnotation
@ClassAnnotation
@RuntimeAnnotation

public class TestAnnotation
{
	public static void main (String args [])
	{
		for (Annotation a : TestAnnotation.class.getAnnotations ())
		{
			System.out.println (a.annotationType().getSimpleName ()) ;
		}
	}
}
```



Exemple @Target

- Exemple d'une annotation ne pouvant être utilisée que sur un constructeur:

```
@Target(ElementType.CONSTRUCTOR)
public @interface ConstructeurAnnotation {}
```

- Liste des éléments pouvant être annotées:

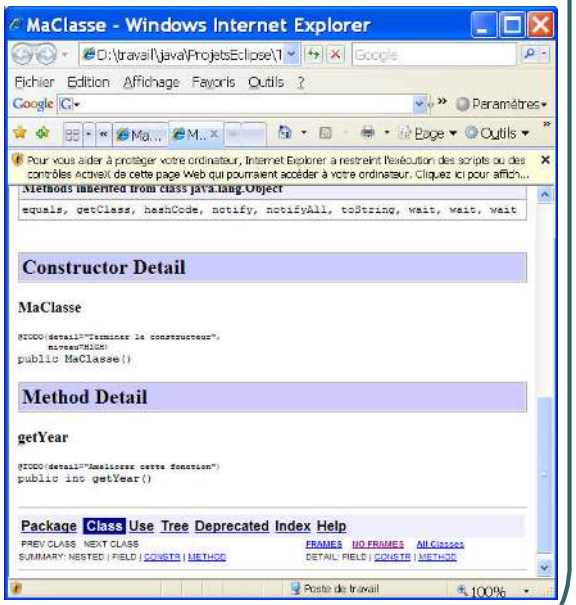
ElementType.ANNOTATION
ElementType.CONSTRUCTOR
ElementType.FIELD
ElementType.LOCAL_VARIABLE
ElementType.METHOD
ElementType.PACKAGE
ElementType.PARAMETER
ElementType.TYPE

Exemple d'annotation personnalisée (1/2)

```
package testannotation ;

import java.lang.annotation.* ;

@Documented
@Retention(RetentionPolicy.SOURCE)
public @interface TODO
{
    public static enum Level { LOW, MEDIUM, HIGH } ;
    String detail () ;
    Level niveau () default Level.MEDIUM ;
}
```

Exemple d'annotation personnalisée(2/2)

```
package testannotation;

import static testannotation.TODO.Level.* ;

public class MaClasse
{
	private int annee ;

	@TODO (detail="Terminer le constructeur",niveau=HIGH)
	public MaClasse ()
	{
		annee = 2007 ;
	}
	@TODO (detail="Ameliorer cette fonction")
	public int getYear ()
	{
		return annee ;
	}

}
```



Les applets

Applet

- **Une applet sera téléchargée à partir d'un site web et sera exécutée dans une machine virtuelle java incorporée aux navigateurs (netscape, internet explorer, mozilla, opera, …)**
- **Pas de méthode *main()***
- **La classe principale d'une applet doit étendre la classe *java.applet.Applet.***
- **Quelques fonctions importantes de la classe *java.applet.Applet*:**
  - ***public void init()***
  - ***public void start ()***
  - ***public void stop ()***
  - ***public void destroy ()***
  - ***public void paint (Graphics g)***

Exemple d'Applet

```
package hello ;

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

public class Hello extends Applet
{
   public void paint (Graphics g)
   {
            g.drawString ("Hello World",10,30) ;
   }
}
```

La balise <APPLET>

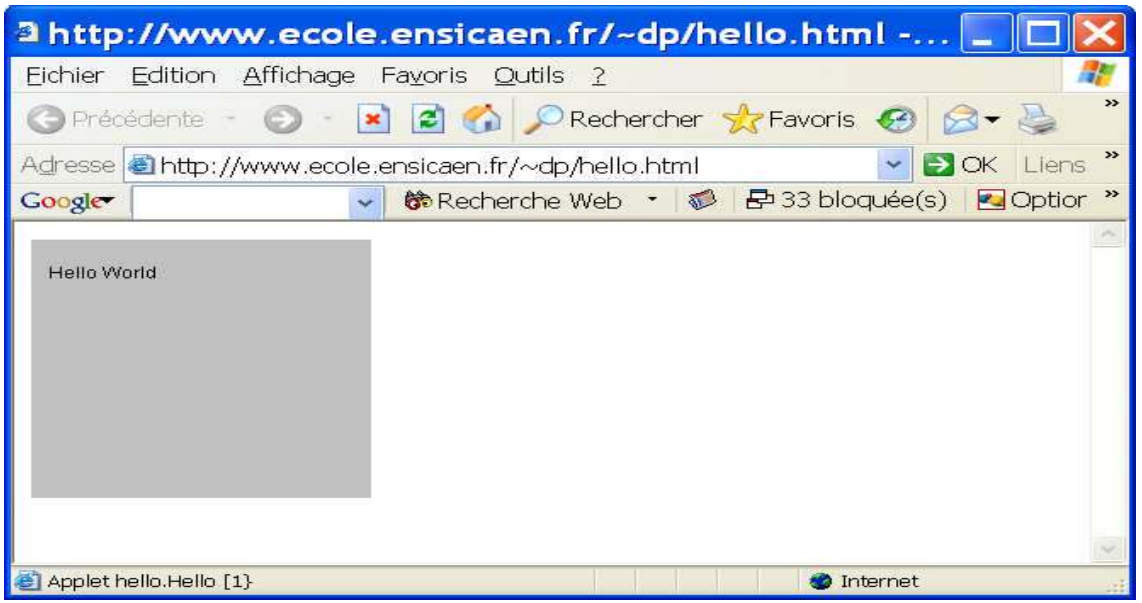
- **L'applet nécessite un fichier HTML contenant une balise <APPLET> pour être exécutée.**

- **Exemple :**

```
<HTML>
<APPLET CODE = "hello.Hello.class" WIDTH=200 HEIGHT=200>
</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

Exécution de l'applet



La balise <APPLET>

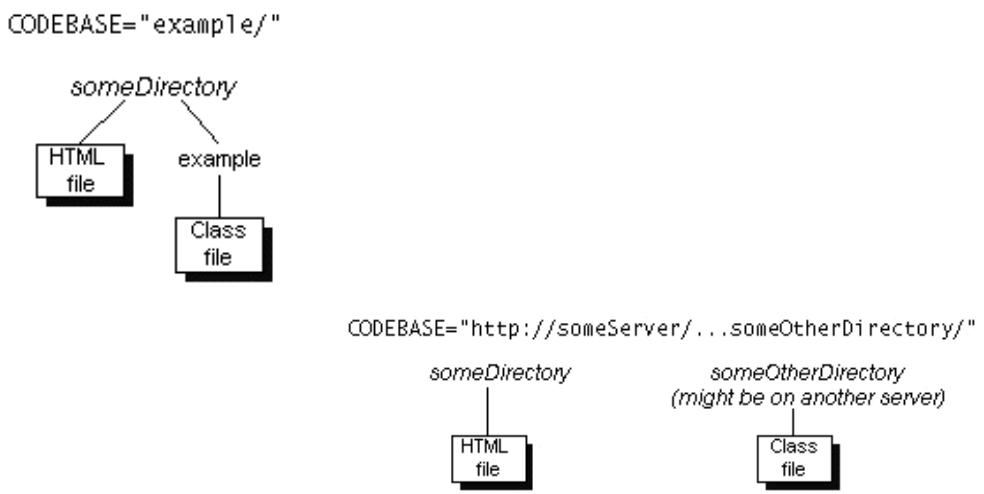
- D'autres attributs pour la balise <APPLET> existent :
  - ***ALIGN*** Définit comment l'applet doit être alignée sur la Page. Les valeurs possibles sont:

    LEFT,RIGHT,TOP,TEXTTOP,MIDDLE,ABSMIDDLE, BASELINE,BOTTOM,ABSBOTTOM

  - ***CODEBASE*** Permet de définir le répertoire ou le serveur

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

Exemple de *CODEBASE*



Exemple d'applet

```
package hello ;

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

public class Hello extends Applet
{
  private String chaine ;

  public void init ()
  {
    chaine = getParameter ("welcome") ;
  }

  public void paint (Graphics g)
  {
    g.drawString (chaine, 10,30) ;
  }
}
```

Exemple d'applet

```
<HTML>

<APPLET CODEBASE="http://www.ecole.ensicaen.fr/~dp"
        CODE="hello.Hello.class"
        ALIGN=LEFT
        WIDTH=200 HEIGHT=200>
<PARAM NAME="welcome" VALUE="Bonjour a tous">
</APPLET>

A la gauche de ce texte, vous pouvez voir une applet affichant un texte
<BR CLEAR=ALL>
Ce texte s'affiche desormais en dessous de l'applet et a gauche de la fenetre du
 navigateur.

</HTML>
```

Exécution de l'applet

La classe java.awt.Graphics

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

Exemples de méthodes de la classe *java.awt.Graphics*

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 **)**

public void setColor (Color c)

….

Gestion de l'affichage

- **L'affichage graphique est géré par un thread "Screen Updater"**

- **Le thread d'affichage appelle la méthode *update (Graphics g)* des composants qui doivent être redessinés. Tous les composants graphiques possèdent cette méthode.**

- **Il est possible de forcer un composant graphique à être redessiné en appelant la méthode *repaint ()* Cela positionne une variable dans le composant à destination du "screen updater" qui appellera la méthode *update (Graphics g)***

Fonctionnement par défaut de la méthode *update (Graphics g)*

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

La classe java.awt.Color

- **La gestion des couleurs est basée sur un modèle à 24 bits**

- **Une couleur est définie par ses composantes RGB**
- **<u>Exemple :</u>**

  Color BleuPale = new Color (0,0,80) ;

- **Certaines couleurs ont un nom symbolique (membres statiques)**
- **<u>Exemples :</u>**

Color.white	***255,255,255***
Color.red	***255,0,0***
Color.orange	***255,200,0***

Manipulation des couleurs

- **Quelques méthodes de gestion de couleurs héritées de la classe *java.awt.Component*:**

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

- **Quelques méthodes de gestion de couleurs appartenant à la classe *java.awt.Graphics*:**

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

La classe java.awt.Font

- **La classe *java.awt.Font* permet de définir des objets représentants des polices de caractères caractérisées par leur nom, leur style et leur taille en points**
- **Exemple :**

  Font f = new Font ("Monospaced", font.BOLD, 24) ;
- **Les spécifications du langage java fournissent 5 polices universelles:**

  Dialog
  SansSerif (anciennement Helvetica)
  Serif (anciennement TimesRoman)
  Monospaced (anciennement Courier)
  Symbol
- **En fonction de la plate forme utilisée, d'autres polices peuvent être disponibles.**
- **Styles disponibles:**

  Font.BOLD
  Font.ITALIC
  Font.PLAIN

Exemple d'utilisation de la classe java.awt.Font

```
package exemplefont ;

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)) ;
        }
    }
}
```

Les interfaces utilisateurs avec le package java.awt

Les interfaces utilisateurs avec le package java.awt

- **Le package *java.awt* propose des classes graphiques permettant de concevoir facilement une interface utilisateur**

  java.awt.Button
  java.awt.CheckBox
  java.awt.Choice
  java.awt.Label
  java.awt.List
  java.awt.Scrollbar
  java.awt.TextArea
  java.awt.TextField

- **Tous les objets s'appuient sur les objets natifs de la plate forme utilisée.**

Exemple de création d'une interface utilisateur

```
package exempleawt ;

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

public class Composants extends Applet
{
        private Label label ;
        private Button button ;
        private Checkbox checkbox ;
        private Choice choice ;
        private Scrollbar h_scrollbar ;
        private TextArea textarea ;
        private TextField textfield ;
        private 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.add ("chaine 1") ;  list.add ("chaine 2") ;
        add (list) ;
        }
}
```

Exemple d'affichage des composants

Les classes de mise en page

- La fenêtre du navigateur sert de container
- Pour disposer les composants graphiques, on utilise des classes de mise en page :
  - *java.awt.FlowLayout*
  - *java.awt.BorderLayout*
  - *java.awt.CardLayout*
  - *java.awt.GridLayout*
  - *java.awt.GridBagLayout*

La classe java.awt.FlowLayout

- **Cette classe dispose les objets par ordre d'ajout. C'est le layout par défaut pour les containers de type *java.awt.Panel*.**
- **Plusieurs constructeurs:**
  - ***public FlowLayout () ;***
  - ***public FlowLayout (int align) ;***
  - ***public FlowLayout (int align, int hgap, int vgap) ;***
    - **Le paramètre *align* peut prendre les valeurs:**

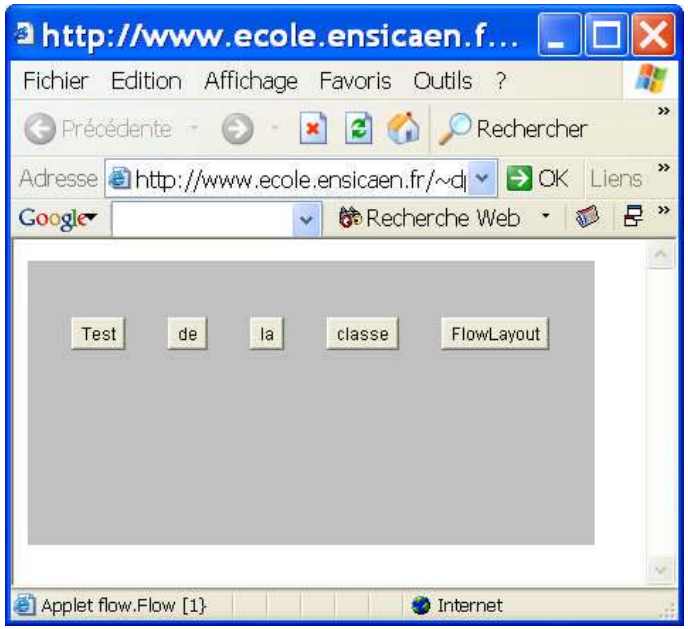
      FlowLayout.LEFT
      FlowLayout.CENTER
      FlowLayout.RIGHT

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

Exemple de FlowLayout

```
package flow ;

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

public class Flow extends Applet
{
      public void init()
      {
            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")) ;
      }
}
```

Exécution de l'applet "Flow"



La classe java.awt.BorderLayout

- **Les composants graphiques sont ajoutés en spécifiant une position géographique :**

  North, South, East, West, Center

- **C'est le layout par défaut des containers *java.awt.Frame*.**
- **Plusieurs constructeurs:**

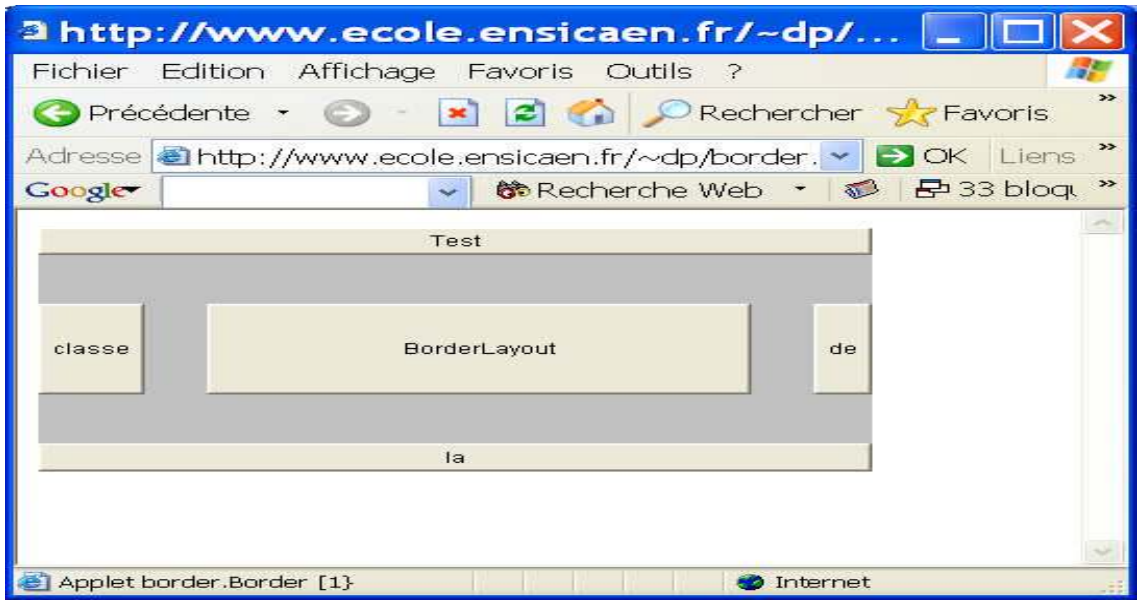
  public BorderLayout () ;

  public BorderLayout (int hgap, int vgap) ;

Exemple de BorderLayout

```
package border;

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


public class Border extends Applet
{
	public void init()
	{
		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")) ;
	}
}
```

Exécution de l'applet "Border"



La classe java.awt.GridLayout

- La classe *java.awt.GridLayout* définit un quadrillage dans lequel les composants graphiques seront placés de la gauche vers la droite et du haut vers le bas.

- Plusieurs constructeurs:

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

Exemple de GridLayout

```
package grid ;

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


public class Grid extends Applet
{
	public void init()
	{
		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")) ;
	}
}
```

Exécution de l'applet "Grid"

La classe java.awt.GridBagLayout

- La classe *java.awt.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.
- Un seul constructeur:
  - *public GridBagLayout () ;*
- Chaque composant graphique sera ajouté en spécifiant les contraintes souhaitées grâce à un objet de classe *java.awt.GridBagConstraints.*
- 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

Principales données membres de java.awt.GridBagConstraints

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

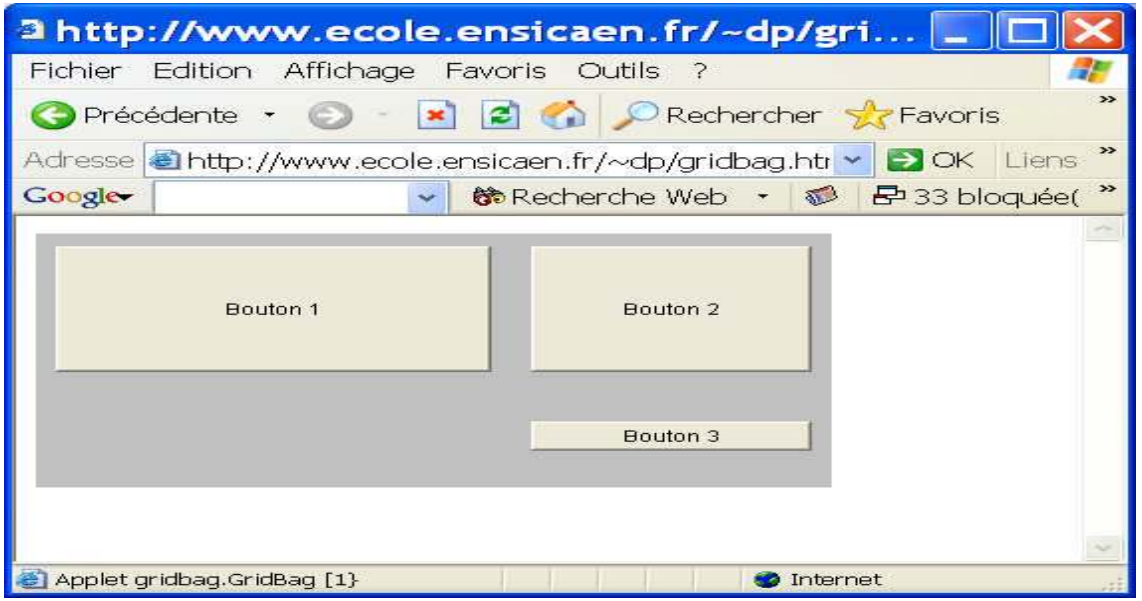
Principales données membres de java.awt.GridBagConstraints

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

Exemple de GridBagLayout

```
package gridbag ;
import java.applet.*;
import java.awt.*;
public class GridBag extends Applet
{
      public void init()
      {
        Button b1 = new Button ("Bouton 1") ; Button b2 = new Button ("Bouton 2") ;
        Button b3 = new Button ("Bouton 3") ;
        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) ;
      }
}
```

Exécution de l'applet "GridBag"



La classe java.awt.CardLayout

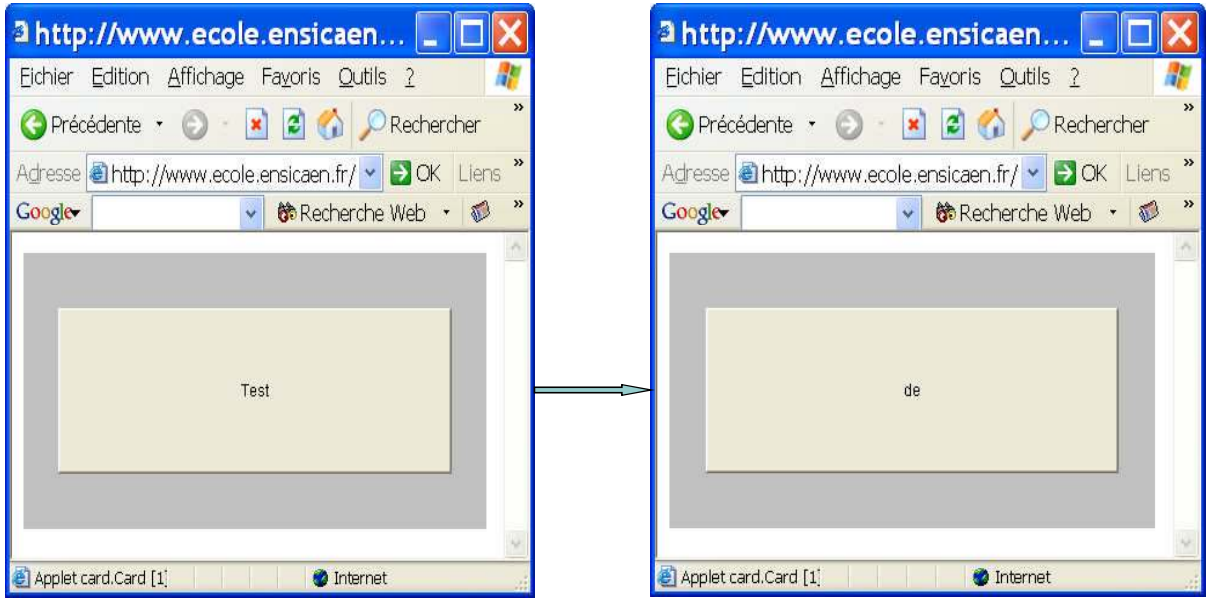
- La classe *java.awt.CardLayout* définit des objets qui ne sont pas visibles simultanément mais consécutivement.
- Plusieurs constructeurs:
  - *public CardLayout () ;*
  - *public CardLayout (int hgap, int vgap) ;*
- Quelques méthodes pour passer d'un composant à un autre :

first ()	Affiche le premier composant
last ()	Affiche le dernier composant
previous ()	Affiche le composant précédent
next ()	Affiche le composant suivant
show ()	Affiche le composant spécifié dans le 2ème argument

Exemple de CardLayout

```
package card ;
import java.applet.*;
import java.awt.*;
import java.awt.event.* ;
public class Card 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) ;
      }
      public void actionPerformed (ActionEvent evt)  { cl.next (this) ;  }
}
```

Exécution de l'applet "Card"



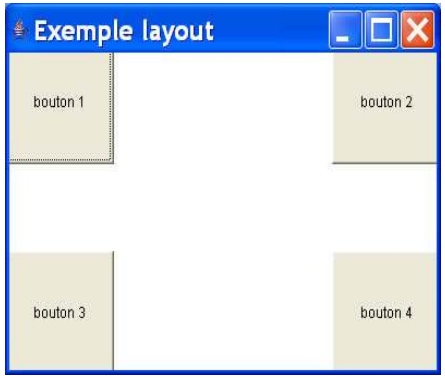
Layout personnalisé

- On peut ne pas utiliser de layout (*setLayout (null)*) et positionner les composants en fonction de leurs coordonnées. Cette méthode est adaptée uniquement pour les conteneurs non redimensionnables.

- On peut définir une classe de layout en implémentant l'interface *java.awt.LayoutManager.*

Exemple de layout personnalisé

- Layout pour disposer au maximum quatre composants de taille 100x100 aux quatre coins d'un conteneur.



Exemple de layout personnalisé

```
import java.awt.* ;
public class MyLayout implements LayoutManager
{
	public void addLayoutComponent(String name, Component comp)  { }
	public void layoutContainer(Container parent)
	{
		Component comp [] = parent.getComponents() ;
		Rectangle rect = parent.getBounds() ;
		for (int i = 0 ; i != comp.length && i != 4 ; i++)
		{
			switch (i)
			{
			   case 0: comp [i].setBounds(0,30,100,100) ; break ;
			   case 1: comp [i].setBounds(rect.width-100, 30, 100, 100) ; break ;
			   case 2: comp [i].setBounds(0, rect.height-100, 100, 100) ; break ;
			   case 3: comp [i].setBounds(rect.width-100, rect.height-100, 100, 100) ; break ;
			}
		}
	}
	public Dimension minimumLayoutSize(Container parent)  { return parent.getSize() ;}
	public Dimension preferredLayoutSize(Container parent)  { return parent.getSize () ; }
	public void removeLayoutComponent(Component comp)  { }
}
```

Exemple de layout personnalisé

```
import java.awt..*;
public class TestLayout extends Frame
{
        public static void main(String[] args)
        {
            new TestLayout().setVisible (true) ;
        }
        public TestLayout()
        {
            super("Exemple layout"); initGUI();
        }
        private void initGUI()
        {
           try {
                    setLayout(new MyLayout ()) ;
                    add (new Button ("bouton 1")) ;
                    add (new Button ("bouton 2")) ;
                    add (new Button ("bouton 3")) ;
                    add (new Button ("bouton 4")) ;
                    pack(); setSize(400, 300);
              } catch (Exception e) { }
         }
}
```

Les conteneurs

Panel	**Conteneur sans fenêtre propre. Permet d'ordonner les composants graphiques.**
Window	**Fenêtre principale sans cadre ni menu.**
Frame	**Fenêtre possédant toutes les fonctionnalités (barre de titre, barre de menus, forme du curseur, etc.)**
Dialog	**Permet de réaliser des boîtes de dialogue. Nécessite une frame**

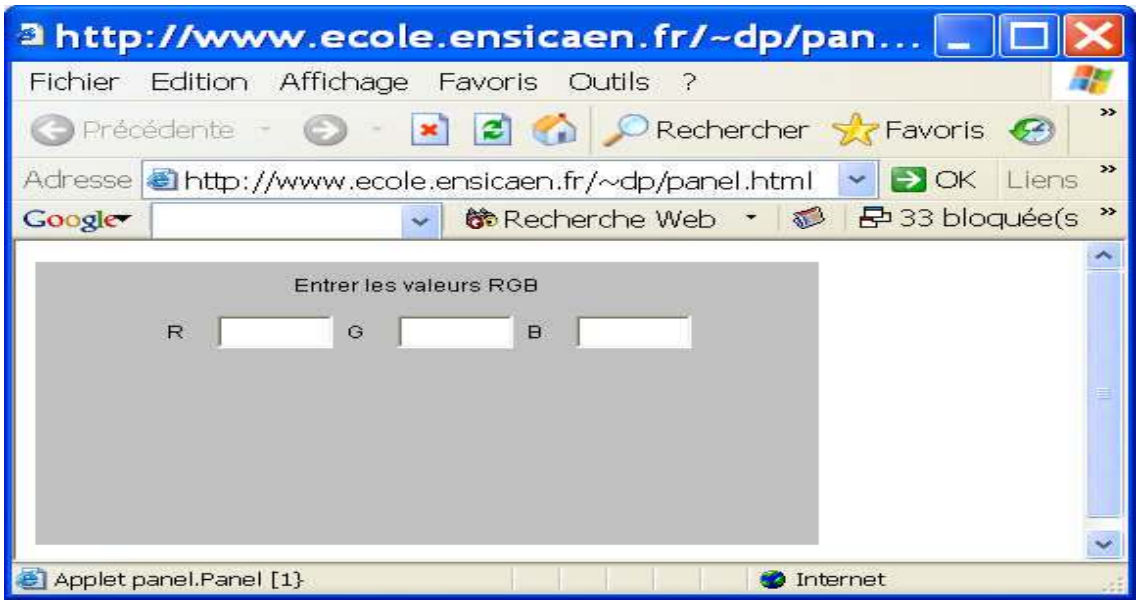
La classe java.awt.Panel

- **Un "Panel" est un container sans fenêtre propre.**
- **Il dispose de son propre layout (*java.awt.FlowLayout* par défaut).**
- **Son utilisation facilite la conception d'interface utilisateur.**

Exemple de Panel

```
package panel;
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 ()) ; // Valeur par defaut
     Haut.add (new Label ("Entrer les valeurs RGB")) ;

     Milieu.setLayout (new FlowLayout ()) ;  // Valeur par defaut
     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) ;
  }
}
```

Exécution de l'appel "Panel"



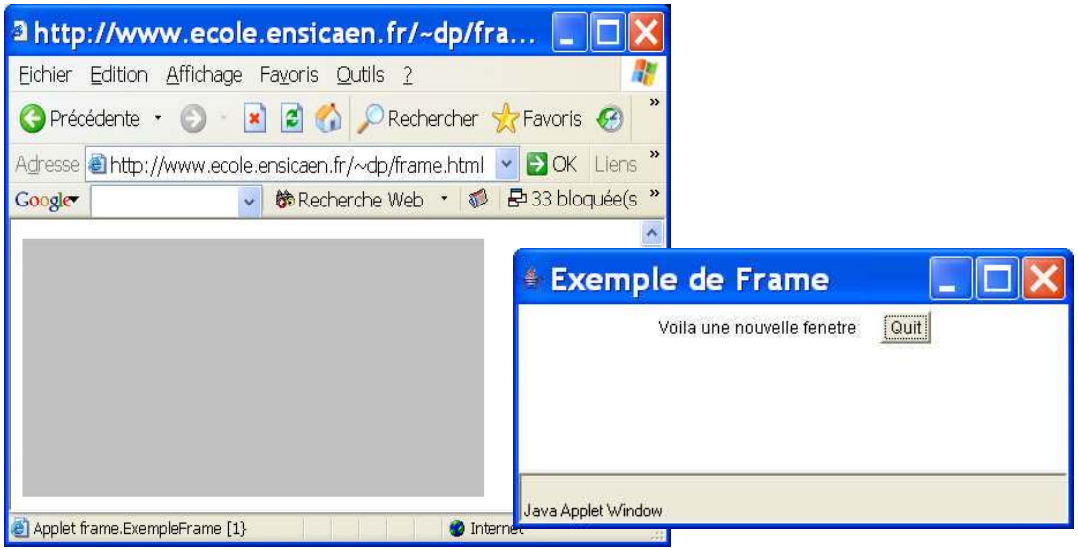
La classe java.awt.Frame

- **Un objet "Frame" est une fenêtre flottante (indépendante de la fenêtre du navigateur dans le cas particulier des applets).**
- **La fenêtre dispose de ses propres caractéristiques (barre de titre, barre de menu, curseur, etc.)**
- **La fenêtre dispose de son propre layout (*java.awt.BorderLayout* par défaut).**

Exemple de Frame

```
package frame ;
import java.applet.* ;
import java.awt.* ;

public class ExempleFrame extends Applet
{
        public void init ()
        {
                new MyFrame ("Exemple de Frame") ;
         }
}

class MyFrame extends Frame
{
   public MyFrame (String title)
  {
       super (title) ;
       setLayout (new FlowLayout ()) ;
       add (new Label ("Voila une nouvelle fenetre")) ; add (new Button ("Quit")) ;
       setSize (400,200) ; setVisible (true) ;
 }
}
```

Exécution de l'applet "ExempleFrame"



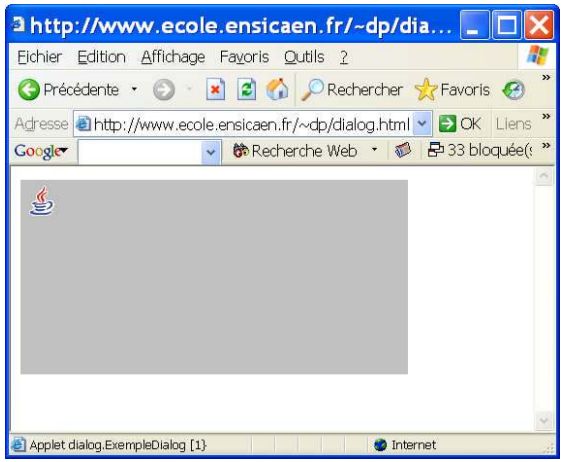
Les boîtes de dialogues

- **Les boîtes de dialogue s'appuie sur les frames pour disposer de leur propre fenêtre.**
- **Une boîte de dialogue peut être modale ou non**
- **S'appuyant sur une frame, une boîte de dialogue dispose de son propre layout (*java.awt.BorderLayout* par défaut).**
- **Les constructeurs de la classe *java.awt.Dialog*:**
  - ***public Dialog (Frame parent, boolean modal)***
  - ***public Dialog (Frame parent, String title ,boolean modal)***

Exemple de Dialog

```
package dialog;
import java.applet.*;
import java.awt.*;
public class ExempleDialog extends Applet
{
    public void init ()
    {
        ErrorDialog error = new ErrorDialog ("Permission denied") ;
        error.setVisible (true) ;
    }
}
class ErrorDialog extends Dialog
{
    ErrorDialog (String message)
    {
       super (new Frame (),"Error",true) ;
       add ("Center",new Label (message)) ;
       add ("South", new Button ("OK")) ;   // Il faudrait armer l'evenement du bouton
      setSize (200,200) ;
    }
}
```

Exécution de l'applet "ExempleDialog"



Les menus

- **Le package *java.awt* propose des classes permettant de créer des barres de menu dans des frames et des "Popup Menu"**
- **Classes du package *java.awt* à utiliser :**
  Menu, MenuItem, MenuBar, PopupMenu,CheckBoxMenuItem, MenuShortCut
- **Une barre de menu est positionnée par la méthode *setMenuBar ()* de la classe *java.awt.Frame.***
- **Un "popup menu" est ajouté à un composant grâce à la méthode *void add (Popup menu)* de la classe *java.awt.Component* et est rendu visible grâce à la méthode *void show (Component origin, int x, int y)* de la classe *java.awt.PopupMenu.***
- **La méthode *setEnabled (boolean)* de la classe *java.awt.Component* permettent de rendre sensitif/insensitif une option ou sous option de menu.**
- **Les sous options de menu peuvent être séparées par un trait horizontal par ajout d'un MenuItem spécial (*new MenuItem ("-") ;)* ou par les méthodes *void addSeparator ()* ou *void insertSeparator (int index)* de la classe *java.awt.Menu.***

Exemple de barre de menu

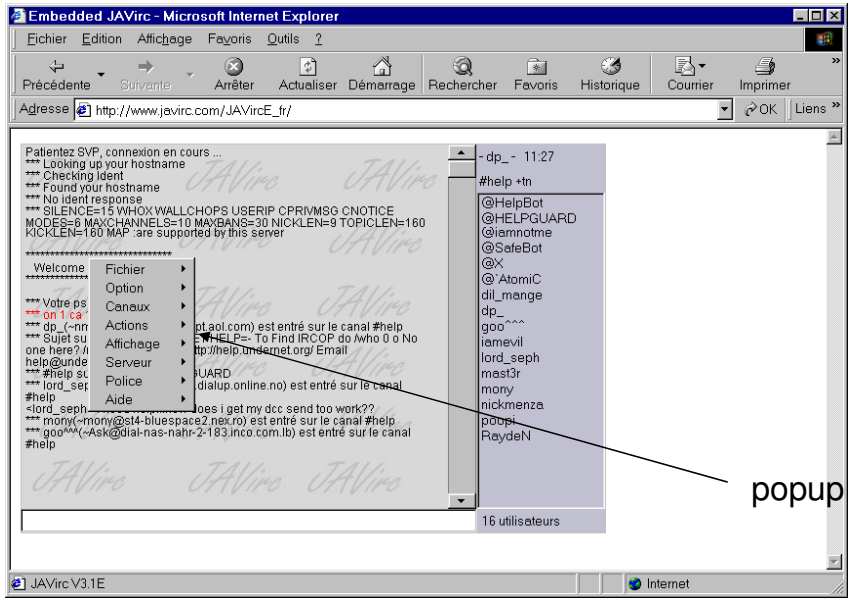
```
package menubar ;

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

Exemple de barre de menu

```
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 (400,100) ; setMenuBar (mb) ; setVisible (true) ;
  }
}
```

Exécution de l'applet "ExempleMenuBar"

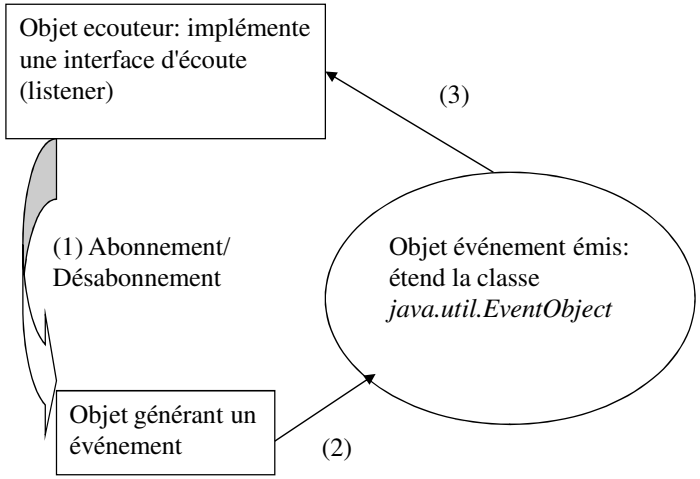
Exemple de popup menu



Gestion des événements

Gestion des événements

- **Permet de définir un comportement en réponse à une action d'un utilisateur ou du système.**



Les Méthodes de *java.util.EventObject*

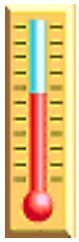
Méthodes	Description
public EventObject (Object source)	Constructeur; prend en paramètre l'objet source de l'événement.
public Object getSource ()	Renvoie l'objet qui a généré l'événement.
public String toString ()	Renvoie une chaîne de caractères caractérisant l'objet.

Gestion des Evénements

- La réception et le traitement des événements sont clairement séparés :
- Certains objets sont à même d'émettre des événements
- Un objet écouteur pourra recevoir ces événements si:
  - il implémente l'interface *listener* correspondant au type de l'événement.
  - il s'est enregistré auprès d'un émetteur.

Exemple d'événement

L'objet thermomètre s'enregistre auprès de l'objet température pour être informé des changements.



Objet thermomètre; actualise son affichage à chaque événement *TempChangeEvent* reçu

Objet température; génère des événements *TempChangeEvent*

La classe "TempChangeEvent"

```
package thermometre;

public class TempChangeEvent extends java.util.EventObject
{
   protected double temperature;
   public TempChangeEvent(Object source, double temperature)
  {
    super(source);
    this.temperature = temperature;
  }

  public double getTemperature()
  {
    return temperature;
  }
}
```

L'interface "TempChangeListener"

- Un objet récepteur de l'événement TempChangeEvent doit implémenter cette interface.

```
package thermometre;

public interface TempChangeListener extends java.util.EventListener
{
   void tempChange(TempChangeEvent evt);
}
```

La classe "*Temperature*" 1/3

```
package thermometre;
import java.util.*;

public class Temperature extends Thread
{
  protected double TemperatureCourante = 0 ;
  // Stockage des récepteurs
  private Vector<TempChangeListener> TempChangeListeners = new Vector<TempChangeListener> ();

  public synchronized void addTempChangeListener(TempChangeListener l)
  {
    if ( ! TempChangeListeners.contains(l)) TempChangeListeners.addElement(l);
  }

  public synchronized void removeTempChangeListener(TempChangeListener l)
  {
    if (TempChangeListeners.contains(l)) TempChangeListeners.removeElement(l);
  }
```

La classe « *Temperature* » 2/3

```
protected void notifyTemperatureChange()
 {
    TempChangeEvent evt = new TempChangeEvent (this,TemperatureCourante);

    Vector<TempChangeListener> recepteursClone;
    synchronized(this)
    {
       recepteursClone = (Vector<TempChangeListener>) TempChangeListeners.clone();
    }
    Iterator<TempChangeListener> Iter = recepteursClone.iterator();
    while (Iter.hasNext())
    {
          TempChangeListener Thermometre =  Iter.next();
          Thermometre.tempChange(evt);
    }
 }
```

La classe "*Temperature*" 3/3

```
//Thread pour modifier la température et
//générer des événements TempChangeEvent

public void run ()
 {
   while (true)
   {
      try {
           TemperatureCourante = 20 * Math.random() ;
            notifyTemperatureChange () ;
           Thread.sleep (60000) ;
        } catch (InterruptedException e) {}
      TemperatureCourante += 1 ;
  }
 }
}
```

La classe "*Thermometre*"

```
package thermometre;

public class Thermometre implements TempChangeListener
{
   public Thermometre ()
   {
      Temperature Temp = new Temperature () ;
      Temp.addTempChangeListener (this);
      Temp.start () ;  // On lance le Thread de Temperature
   }

   public void tempChange (TempChangeEvent evt)
   {
      System.out.println (evt.getTemperature()) ;
   }

   static public void main (String args [])
   {
      new Thermometre () ;
   }
}
```

Gestion des événements pour l'AWT

- Tous les événements et les interfaces listener sont déjà répertoriés et dépendent du type de composants graphiques utilisés.

- Toutes les classes et interfaces sont définies dans le package *java.awt.event*.

Les listeners disponibles dans le package *java.awt*

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

Les méthodes définies dans les interfaces listeners

ActionListener	***public void actionPerformed (ActionEvent)***
AdjustmentListener	***public void adjustmentValueChanged (AdjustmentEvent)***
FocusListener	***public void focusGained (FocusEvent)*** *public void focusLost (FocusEvent)*
ItemListener	***public void itemStateChanged (ItemEvent)***
KeyListener	***public void keyPressed (KeyEvent)*** ***public void keyReleased (KeyEvent)*** ***public void keyTyped (KeyEvent)***

Les méthodes définies dans les interfaces listeners

MouseListener	***public void mouseClicked (MouseEvent)*** ***public void mouseEntered (MouseEvent)*** ***public void mouseExited (MouseEvent)*** ***public void mousePressed (MouseEvent)*** ***public void mouseReleased (MouseEvent)***
MouseMotionListener	***public void mouseDragged (MouseEvent)*** ***public void mouseMoved (MouseEvent)***
WindowListener	***public void windowActivated (WindowEvent)*** ***public void windowClosed (WindowEvent)*** ***public void windowClosing (WindowEvent)*** ***public void windowDeactivated (WindowEvent)*** ***public void windowDeiconified (WindowEvent)*** ***public void windowIconified (WindowEvent)*** ***public void windowOpened (WindowEvent)***

Association source-récepteur

- **L'association source-récepteur se fait par l'une des méthodes suivantes :**
  - **addActionListener (java.awt.event.ComponentListener)**
  - **addAdjustmentListener (java.awt.event.AdjustmentListener)**
  - **addFocusListener (java.awt.event.FocusListener)**
  - **addItemListener (java.awt.event.ItemListener)**
  - **addKeyListener (java.awt.event.KeyListener)**
  - **addMouseListener (java.awt.event.MouseListener)**
  - **addMouseMotionListener (java.awt.event.MouseMotionListener)**
  - **addWindowListener (java.awt.event.WindowListener)**

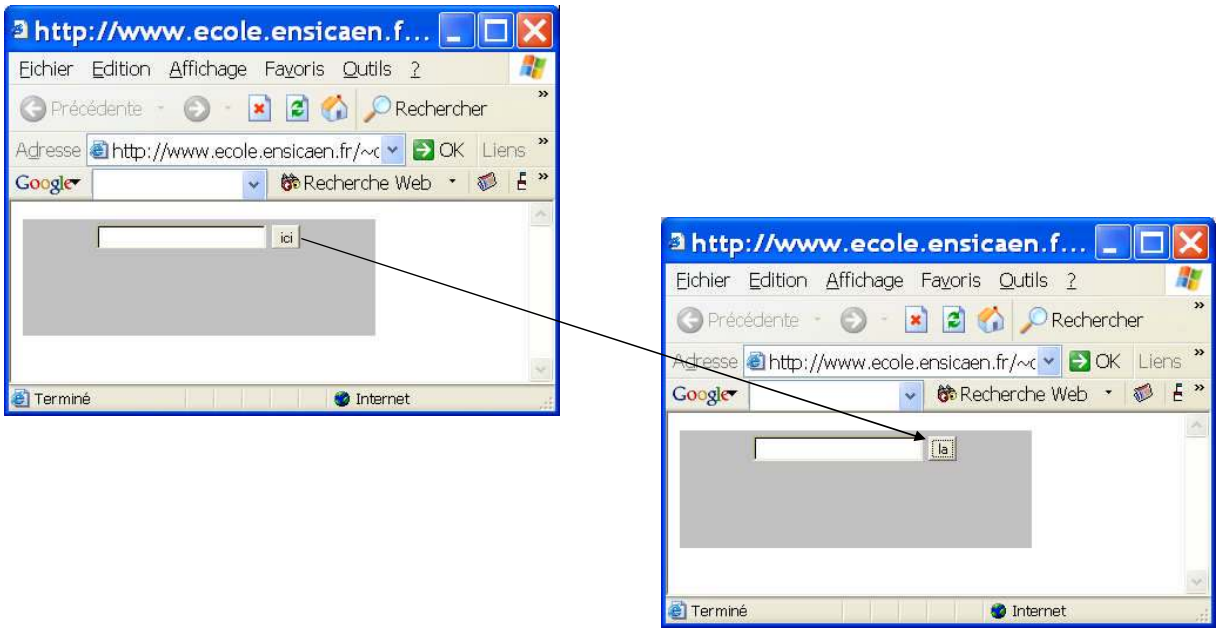
Exemple de gestion d'événement 1/2

```
package evenement ;

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

public class ExempleEvenement extends Applet
{
      Button IciOuLa = new Button ("ici") ; TextField saisie = new TextField (20) ;
      Recepteur recepteur = new Recepteur () ;

      public void init ()
      {
            add (saisie) ; add (IciOuLa) ;
            saisie.addActionListener (recepteur) ;  IciOuLa.addActionListener (recepteur) ;
      }
}
```

Exemple de gestion d'événement 2/2

```
class Recepteur 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") ;
        }
}
```

Exécution de l'applet "ExempleEvenement"



Adaptateurs 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
}
```

Les adaptateurs

- ***Le package java.awt.event contient des adaptateurs (adapters) qui sont des classes implémentant les écouteurs et fournissant des amorces vides des méthodes :***
  - ***MouseAdapter implémente MouseListener***
  - ***MouseMotionAdapter implémente MouseMotionListener***
  - ***KeyAdapter implémente KeyListener***
- ***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

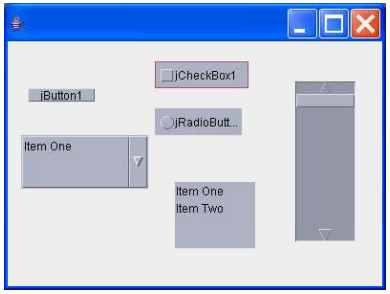
```
import java.applet.*;
import java.awt.event.*;

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") ;
  }
}
```

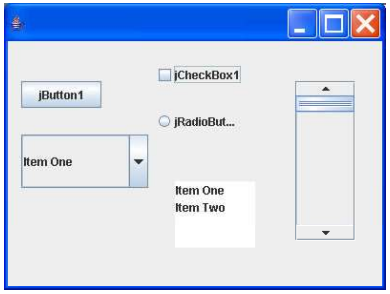
Le package javax.swing

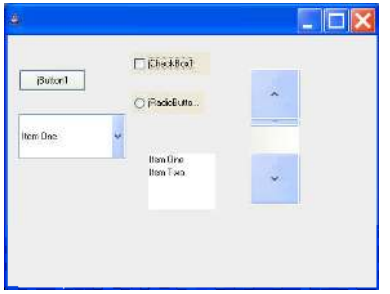
Présentation de Swing

- Swing propose de nouvelles classes et interfaces pour construire des interfaces graphiques.
- Le package *javax.swing* est inclus dans la jdk depuis la version 1.2 de Java.
- Swing utilise le même mécanisme de gestion d'événement que le package *java.awt*.
- Swing propose de très nombreux composants et une apparence modifiable à la volée (Windows, Motif, Metal).

Quelques apparences





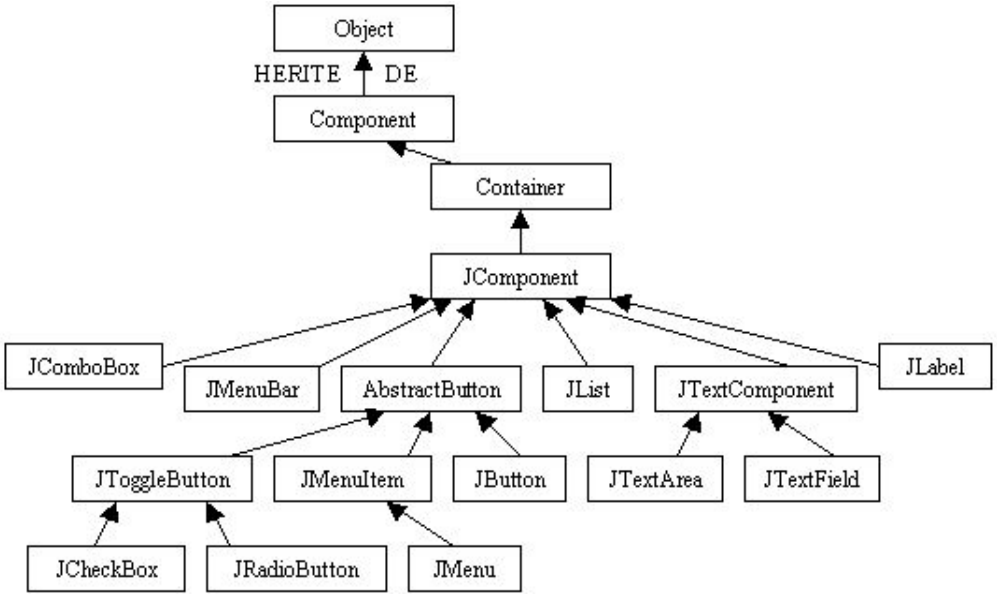



```
UIManager.setLookAndFeel ("com.sun.java.swing.plaf.motif.MotifLookAndFeel") ;
UIManager.setLookAndFeel ("javax.swing.plaf.metal.MetalLookAndFeel") ;
UIManager.setLookAndFeel ("com.sun.java.swing.plaf.windows.WindowsLookAndFeel") ;
```

Connaître les "look and feel" disponibles sur la plate forme:

```
UIManager.LookAndFeelInfo [] info = UIManager.getInstalledLookAndFeels () ;
for (int i = 0 ; i != info.length ; i++) System.out.println (info [i]) ;
```

Extrait de l'arborescence swing



Caractéristiques des composants

- La racine de l'arborescence des classes et interfaces de Swing est la classe *JComponent* (analogue et héritant de *java.awt.Component*).
- Les composants sont des beans.
- Les composants n'ont pas de partie native (sauf *JApplet, JDialog, JFrame, JWindow*).
- Le bord des composants peut être changé.

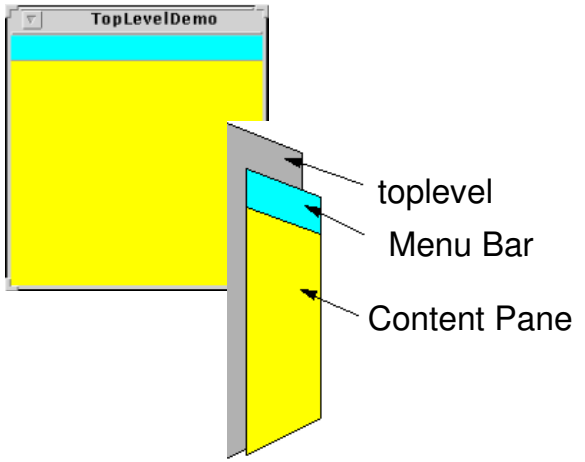
Architecture d'une interface Swing

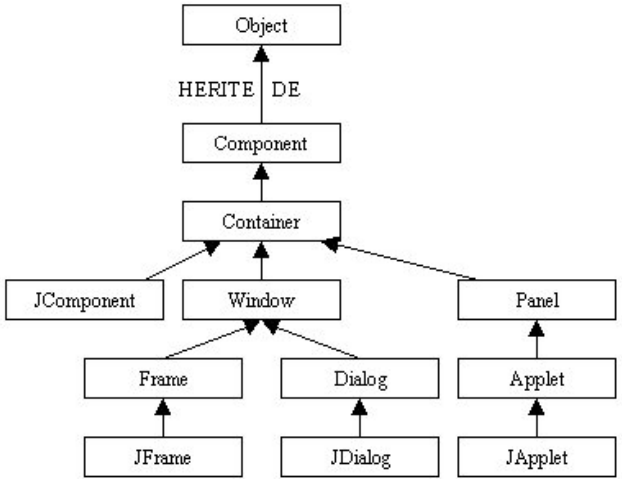
- Une interface sera composée:
  - D'un composant racine (toplevel)
  - D'un ou de plusieurs containers
  - De composants au sein des containers

Composants racines de Swing

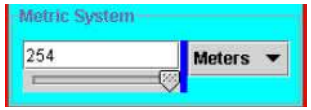
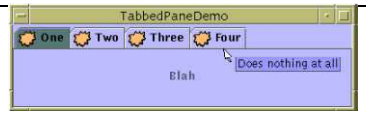
- Il existe trois composants racines principaux:

  JApplet, JFrame, JDialog





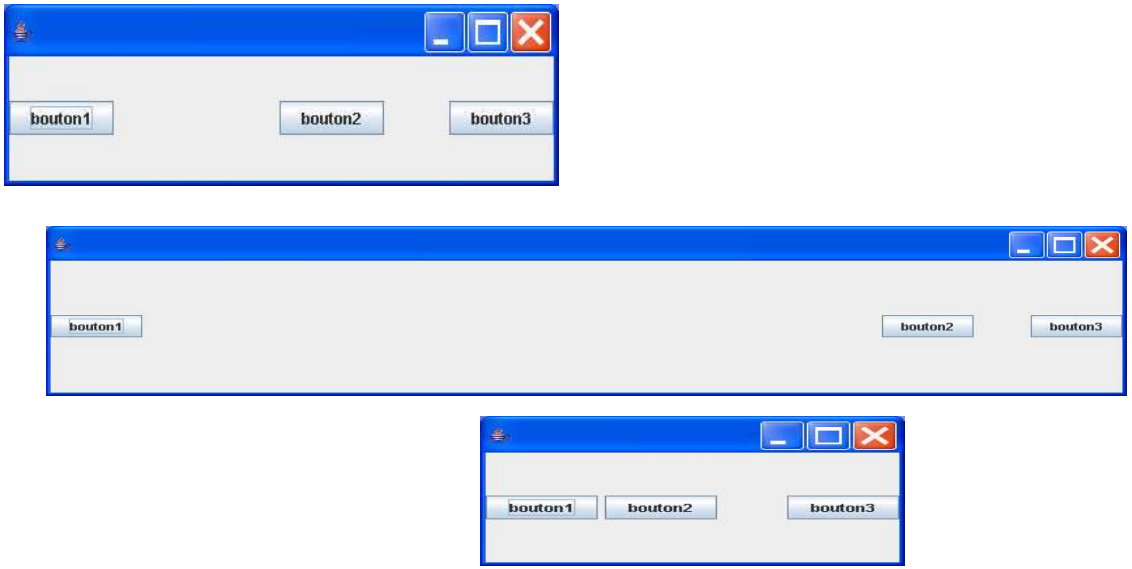
Quelques Containers sans fenêtre propre

JPanel	
JScrollPane	
JSplitPane	
JTabbedPane	
JToolBar	

Disposition des composants

- Les composants utilisent les mêmes classes de layout que les composants AWT.
- Un nouveau Layout: *javax.swing.BoxLayout* qui permet de placer des composants en ligne ou en colonne, chaque composant pouvant avoir sa propre largeur et sa propre hauteur.
- Un nouveau container: *javax.swing.box* utilisant le layout précédent proposant des méthodes statiques de création de composant invisible redimensionnable ("glue") ou non redimensionnable ("strut").

Exemple Container Box

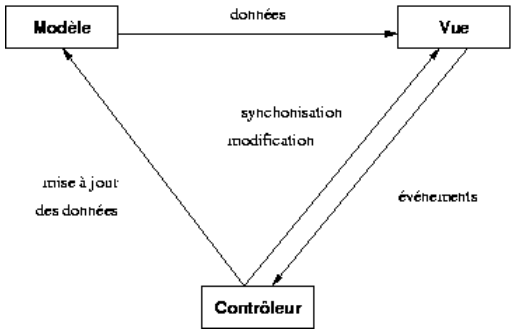


Exemple container Box

```
public class NewJFrame extends javax.swing.JFrame
{
	public static void main(String[] args) {
		NewJFrame inst = new NewJFrame(); inst.setVisible(true);
	}
	public NewJFrame()
	{
		super(); initGUI();
	}
	private void initGUI()
 {
		try {
			setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
			pack(); setSize(400, 300);
			Container box = Box.createHorizontalBox () ;
			box.add (new JButton ("bouton1")) ;
			box.add(Box.createHorizontalGlue()) ;
			box.add (new JButton ("bouton2")) ;
			box.add(Box.createHorizontalStrut(50)) ;
			box.add (new JButton ("bouton3"))  ;
			this.getContentPane().add("Center",box) ;
		} catch (Exception e) { }
	}
}
```

Le modèle MVC

- Swing est bâti sur une architecture **M**odel/**V**iew/**C**ontroller:
  - Model: contient les données et fournit pour y accéder en consultation et modification.
  - View: représentation graphique des données et réception d'événements
  - Controler: gestion des événements



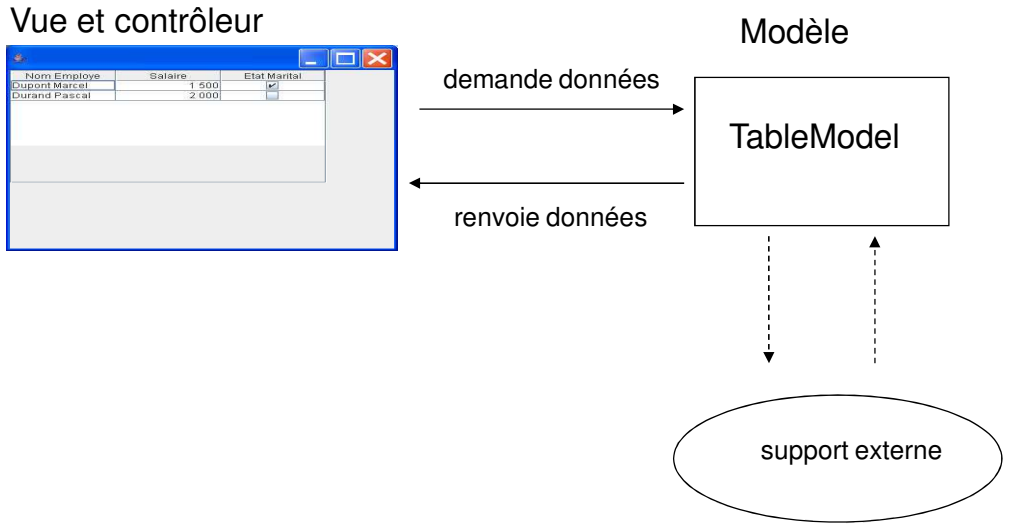
Le modèle MVC

- Les composants Swing vont souvent proposer des interfaces spécifiques pour stocker les données.

- Exemples:
  - *JList* propose *ListModel*
  - *JTextComponent* et ses sous-classes proposent *Document*
  - *JTree* propose *TreeModel*
  - *JTable* propose *TableModel*

Exemple de modèle avec JTable

tutorial à l'adresse: http://java.sun.com/docs/books/tutorial/uiswing/components/table.html#data



Exemple de TableModel

```
package testswing;

public class Employe
{
	private String nom ;
	private float salaire ;
	private boolean marital ;
	public Employe(String nom, float salaire,boolean marital) {
		super();
		this.nom = nom;
		this.salaire = salaire;
		this.marital = marital ;
	}
	public boolean getMarital() { return marital; }
	public void setMarital(boolean marital) { this.marital = marital; }
	public String getNom() { return nom; }
	public void setNom(String nom) { this.nom = nom;}
	public float getSalaire() { return salaire;}
	public void setSalaire(float salaire) { this.salaire = salaire; }
}
```

Exemple de TableModel

```
package testswing;
import java.util.ArrayList;
import javax.swing.table.AbstractTableModel;
public class TableauModel extends AbstractTableModel {
        ArrayList<Employe> donnees ;
        String [] ColumnNames = { "Nom Employe", "Salaire", "Etat Marital" } ;
        Class ColumnTypes [] = { String.class, Float.class, Boolean.class } ;
        public TableauModel () { donnees = new ArrayList<Employe> () ; initialiser () ; }
        private void initialiser (){
            donnees.add (new Employe ("Dupont Marcel",1500.0f,true)) ;
            donnees.add (new Employe ("Durand Pascal",2000.0f,false)) ;
        }
        public int getColumnCount()  { return ColumnNames.length; }
        public int getRowCount()  { return donnees.size () ; }
        public String getColumnName (int col) { return ColumnNames [col] ; }
        public Class getColumnClass (int col)  { return ColumnTypes [col] ; }
        public Object getValueAt(int row, int col) {
            switch (col)
            {
            case 0: return donnees.get(row) .getNom() ;
            case 1: return donnees.get(row) .getSalaire () ;
            case 2: return donnees.get(row).getMarital () ;
            }
        }
}
```

Exemple de TableModel

- Inclusion de la JTable dans une JFrame (extrait):

```
private void initGUI() {
	try {
		setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
		getContentPane().setLayout(null);
		{
			TableauModel jTable1Model = new TableauModel () ;
			jTable1 = new JTable(jTable1Model);
			jTable1.setBounds(63, 14, 245, 161);
			jTable1.setPreferredSize(new java.awt.Dimension(252, 105));
			JScrollPane scrollpane = new JScrollPane (jTable1) ;
			getContentPane().add(scrollpane);
			scrollpane.setBounds(0, 0, 329, 182);

		}
		pack();
		this.setSize(351, 240);
	} catch (Exception e) {
		e.printStackTrace();
	}
}
```

TableModel: modification à partir de la vue

- Modification de la classe *TableauModel* pour rendre la colonne "Etat Marital" modifiable et répercuter la valeur saisie à l'objet *Employe*:

```
public boolean isCellEditable (int row, int col)
{
        if (col == 2) return true ;
        return false ;
}
public void setValueAt (Object value, int row, int col)
{
        if (col == 2) donnees.get(row).setMarital( (Boolean) value) ;
}
```

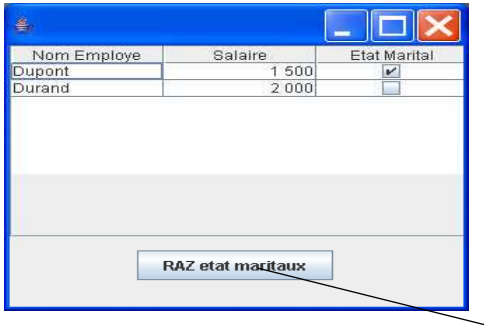
TableModel: Actualisation de la vue à partir d'une modification externe

- Un objet *TableModel* peut avoir des listeners implémentant *TableModelListener*.

- L'interface *TableModelListener* possède les méthodes suivantes:

fireTableCellUpdated	Mise à jour de la cellule spécifiée
fireTableRowsUpdates	Mise à jour de la ligne spécifiée
fireTableDataChanged	Mise à jour de la table complète
fireTableRowsInserted	Nouvelles lignes insérées
fireTableRowsDeleted	Lignes existantes supprimées
fireTableStructureChanged	Invalide la table entière (données et structures)

Modification de la classe TableauModel




```
public TableauModel ()
{
   donnees = new ArrayList<Employe> () ;
   initialiser () ;
   this.addTableModelListener(new RAZEtatMarital ()) ;
}
```


```
public void actionPerformed(ActionEvent arg0)
 {
    for (int i = 0 ; i != donnees.size() ; i++)
   {
        donnees.get(i).setMarital (false) ;
        this.fireTableChanged(new TableModelEvent (this,i));
    }
}
```

Gestion d'une modification externe

```
package testswing;

import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.TableModel;

public class RAZEtatMarital implements TableModelListener
{
    public void tableChanged(TableModelEvent evt) {
            int row = evt.getFirstRow() ;
            int col = evt.getColumn() ;
            TableModel model = (TableModel) evt.getSource() ;
            model.setValueAt(false, row, col) ;

    }
}
```

Personnaliser l'apparence avec les "renderers"

- Les vues permettent de dessiner des données.
- Exemple précédent:

- Pour des raisons de performances on n'alloue pas n composants graphiques pour un tableau de n cellules.
- Le tableau va consulter un renderer pour savoir comment dessiner la cellule (i,j).

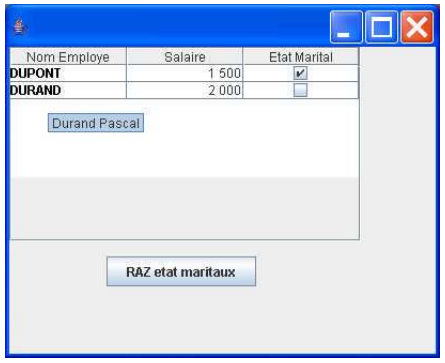
Les renderers par défaut

- Les composants complexes JList, JTable, JTree possèdent un renderer par défaut:
  - JList: DefaultListCellRenderer (implémentant ListCellRenderer)
  - JTable: DefaultTableCellRenderer (implémentant TableCellRenderer)
  - JTree: DefaultTreeCellRenderer (implémentant TreeCellRenderer).
- Les principaux types ont un renderer par défaut:

Boolean	JCheckBox
Number	JLabel justifié à droite (ou JTextField si la cellule est éditable)
Double, Float	Comme Number avec vérification du format
Date	JLabel avec vérification du format
ImageIcon, Icon	JLabel centré
Object	JLabel affichant la chaîne de caractères représentant l'objet

Définition d'un renderer personnalisé

- Sur la 1ère colonne du tableau, on souhaite n'afficher que le nom de famille en majuscule, le nom complet dans une infobulle et des caractères rouge quand la cellule est sélectionnée.



Définition d'un renderer personnalisé

```
package testswing;
import java.awt.* ;
import java.util.StringTokenizer;
import javax.swing.*;
import javax.swing.table.TableCellRenderer;
public class NomRenderer extends JLabel implements TableCellRenderer
{
	public NomRenderer () { super () ; setOpaque (true) ; }
	public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected,
	boolean hasFocus, int row, int col)
	{
		if (isSelected) { setBackground (SystemColor.textHighlight) ; setForeground (Color.red) ; }
		else { setBackground (Color.white) ; setForeground (Color.black) ; }
		String nom = (String) value ;
		StringTokenizer st = new StringTokenizer (nom," ") ; this.setText(st.nextToken ().toUpperCase()) ;
		this.setToolTipText(nom) ;
		return this;
	}
}
```

Application du renderer à l'objet JTable:

```
TableColumn col0 = jTable1.getColumnModel().getColumn(0) ;
col0.setCellRenderer(new NomRenderer ()) ;
```

Swing et Threads

- Une application/applet java utilisant des interfaces graphiques met en œuvre plusieurs threads:
  - Thread initial (Initial thread) en charge de la construction de l'interface.
  - Thread de travail (worker thread) tournant en arrière plan pour la récupération de la mémoire (garbage collector).
  - Thread de gestion d'événements et de réaffichage des composants (event dispatch thread).

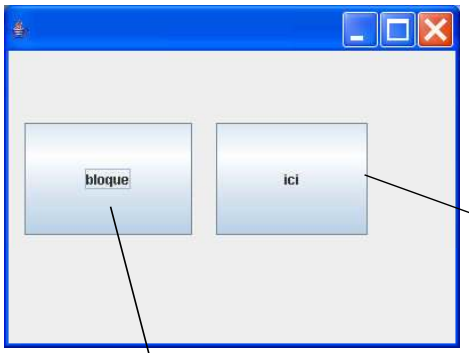
Swing et threads

- Un composant swing peut être manipulé par n'importe quel thread tant qu'il n'a pas été réalisé (pack ou setVisible (true)).
- Ensuite toutes les modifications doivent être effectuées par le thread de gestion d'événements (la plupart des méthodes de swing ne sont pas "thread safe").
- Problèmes:
  - Une application peut être nativement multithread. Comment un thread peut modifier un composant swing ?
  - Si un événement déclenche un traitement très long, il faut le déporter dans un thread pour ne pas geler l'interface; ce thread peut ensuite avoir besoin de modifier un composant swing.

Swing et threads

- La classe *javax.swing.SwingUtilities* propose des méthodes statiques permettant d'exécuter du code dans le thread de gestion d'événement:
  - *invokeLater (Runnable)*
  - *invokeAndWait (Runnable)*

Exemple Swing et threads




```
public void actionPerformed(ActionEvent evt)
{
  if (jChange.getText().equals("ici")) jChange.setText("la") ;
  else jChange.setText("ici") ;
}
```

```
public void actionPerformed(ActionEvent evt)
{
    jBloque.setText("en cours") ;
    try { Thread.sleep (3000) ; } catch (Exception e) {}
    jBloque.setText("bloque") ;
}
```

bloque interface

Exemple Swing et threads

- On modifie l'événement du bouton "jBloque":

```
jBloque.addActionListener(new ActionListener()  {
        public void actionPerformed(ActionEvent evt) {
                                        jBloque.setText("en cours") ; new Traite ().start () ;
                                        }
                                });


private class Traite extends Thread {
        public void run ()
        {

                try { Thread.sleep (3000) ; } catch (Exception e) {}
                Runnable modifiejBloque = new Runnable () {
                        public void run () {
                                        jBloque.setText("bloque") ;
                        }
                } ;
                SwingUtilities.invokeLater(modifiejBloque) ;

        }
}
```

Traitement dans un thread externe

La modification du texte de jBloque se fera par le thread de gestion d'événements

La classe SwingWorker

- La version 6 de Java propose la classe abstraite *SwingWorker* proposant 2 types paramétrés pour faciliter l'écriture des applications swing multithread. Quelques méthodes intéressantes:

Méthode	Description
protected abstract T doInBackground ()	***à redéfinir.*** La seule méthode abstraite. C'est dans cette méthode qu'il faut définir le code à exécuter **dans un thread séparé** (un calcul long par exemple). Cette méthode retourne un résultat, du type **T** (type passé en paramètre de la classe), que l'on peut récupérer grâce à la méthode **get()** une fois le traitement terminé.
protected void done()	***à redéfinir (éventuellement).*** Permet d'effectuer des actions dans l'**EDT** une fois que le traitement (effectué par **doInBackground**) est terminé.
protected final void publish(V... chunks)	Permet de transmettre des résultats partiels du traitement. Le paramètre utilise l'ellipse, permettant d'avoir un nombre quelconque d'arguments (**en savoir plus sur l'ellipse**), ici de type **V**. **V** est le second type passé en paramètre de la classe, et définit le type des résultats partiels à transmettre à la méthode **process**.
protected void process(List<V> chunks)	***à redéfinir (éventuellement).*** Permet de récupérer les résultats partiels du traitement dans l'**EDT** publiés par la méthode **publish**. Cette méthode prend en paramètre une liste de **V**, et non uniquement un **V**, car pour des raisons d'efficacité, plusieurs appels à **publish** peuvent résulter en 1 seul appel à **process**, en transmettant donc plusieurs résultats à la fois.
public final T get()	Permet de récupérer le résultat renvoyé par **doInBackground**, en attendant éventuellement que le traitement se termine s'il n'est pas terminé. Cette méthode étant bloquante, il faut donc éviter de l'appeler à partir de l'**EDT** (sauf si vous savez ce que vous faites).
protected void setProgress(int progress)	Indique le nouvel état d'avancement du traitement (à appeler donc dans la méthode **doInBackground**). Comme nous le verrons plus tard, ceci est bien pratique pour mettre à jour une barre de progression.
public final void addPropertyChangeListener (PropertyChangeListener listener)	Ajoute un écouteur de propriétés, permettant notamment d'écouter l'avancement du traitement, provoqué par **setProgress**.
public final void execute()	Démarre l'exécution de **doInBackground** dans un thread séparé.

Utilisation de SwingWorker

```
class MySwingWorker extends SwingWorker <Void,Void>
{
	@Override
	protected Void doInBackground() throws Exception
	 {
		jBloque.setText("en cours") ;
		try { Thread.sleep (3000) ; } catch (Exception e) {}
		jBloque.setText("bloque") ;
		return null ;
	}
}

jBloque.addActionListener(new ActionListener()
{
	public void actionPerformed(ActionEvent evt)
	 {
		new MySwingWorker ().execute() ;
	}
});
```

Java et la sécurité

Java et la sécurité

- **La sécurité a toujours été présente dans la conception de 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
  6) ...

Java et la sécurité

- 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

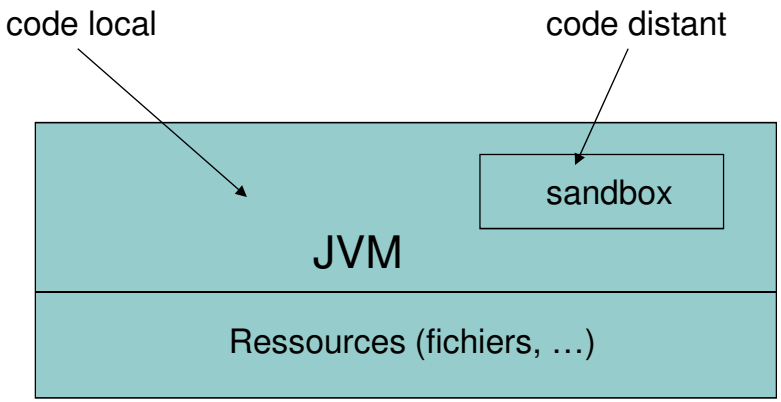
La "sandbox" java

- 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.

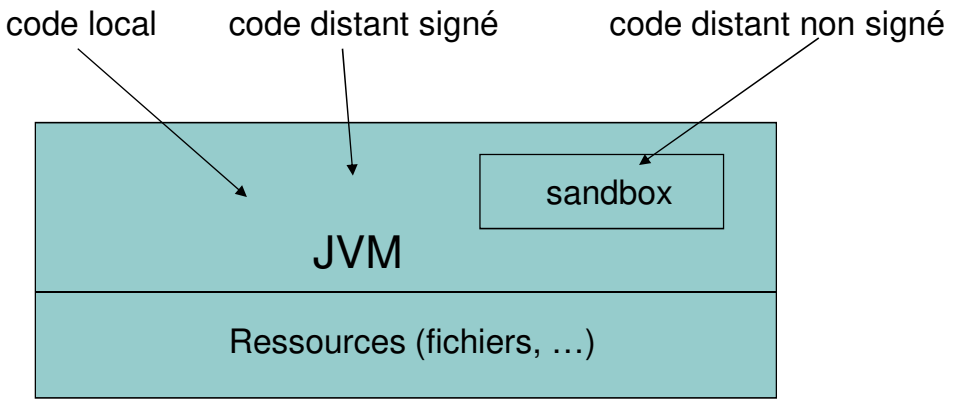
La sécurité des applets java

- 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).

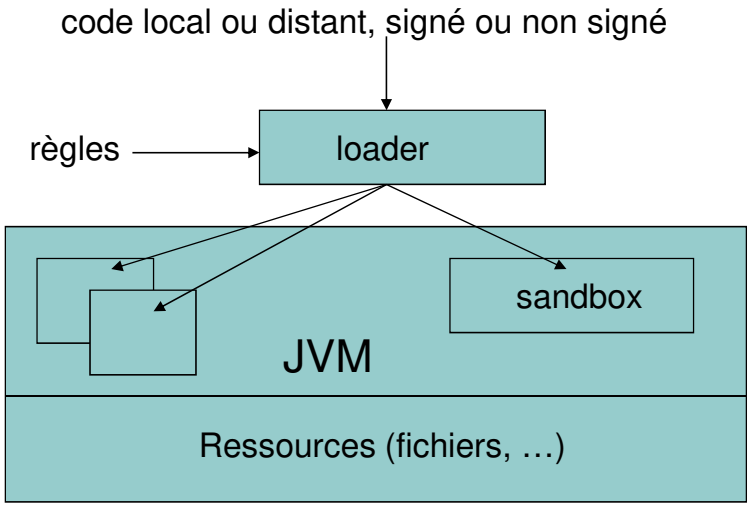
Le modèle de sécurité dans java 1.0



Le modèle de sécurité dans java 1.1



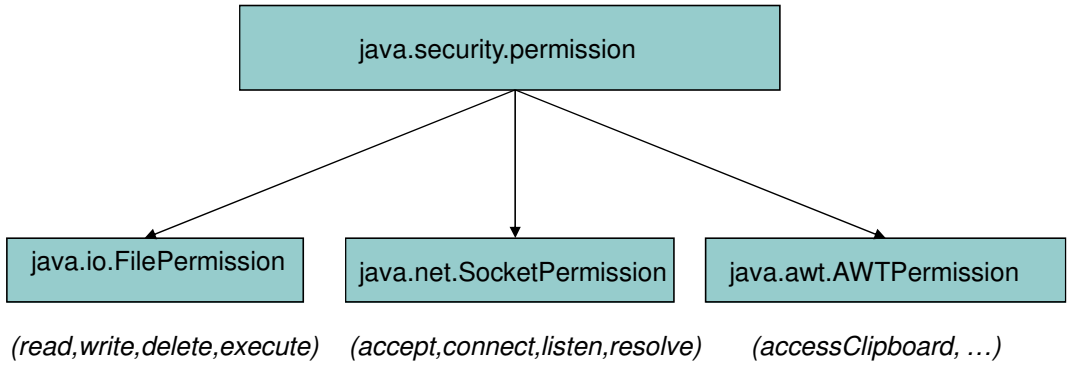
Le modèle de sécurité dans java 1.2



Résumé des différents modèles

- java 1.0: sandbox très restrictive.
- java 1.1: principe du tout ou rien selon que la signature électronique est utilisée ou non.
- java 1.2: principe du moindre privilège. Une stratégie de sécurité pourra être appliquée à une application ou à une applet en fonction de son origine, de l'identité du tiers certificateur.
  - Exemple:
    - Accorder à toutes les applets de http://www.trusted.com la permission de lire les fichiers du répertoire c:\temp.
    - Accorder à toutes les applets la permission de se connecter sur n'importe quelle machine.
    - etc.

Quelques types de permissions



Spécification de la stratégie de sécurité

- Créer ou modifier le fichier de stratégie système *<java.home>\lib\security\java.policy*.
- Donner à la propriété système *java.policy* le nom d'un autre fichier de stratégie de sécurité.
- Créer ou modifier le fichier de stratégie utilisateur dans *<user.home>\java.policy*.
- Définir une autre valeur pour la propriété *java.policy* en utilisant l'option -D de la ligne de commande:

  java -Djava.policy="test.policy" Test
- Changer la classe utilisée dans le fichier *<java.home>\lib\security\java.security* en changeant la ligne *policy.provider=java.security.PolicyFile* en *policy.provider=AutreClasse*.

Contenu du fichier de stratégie de sécurité

grant [SignedBy "nom_signataires] [, CodeBase "URL] {
rubriques permissions
} ;

Exemple:

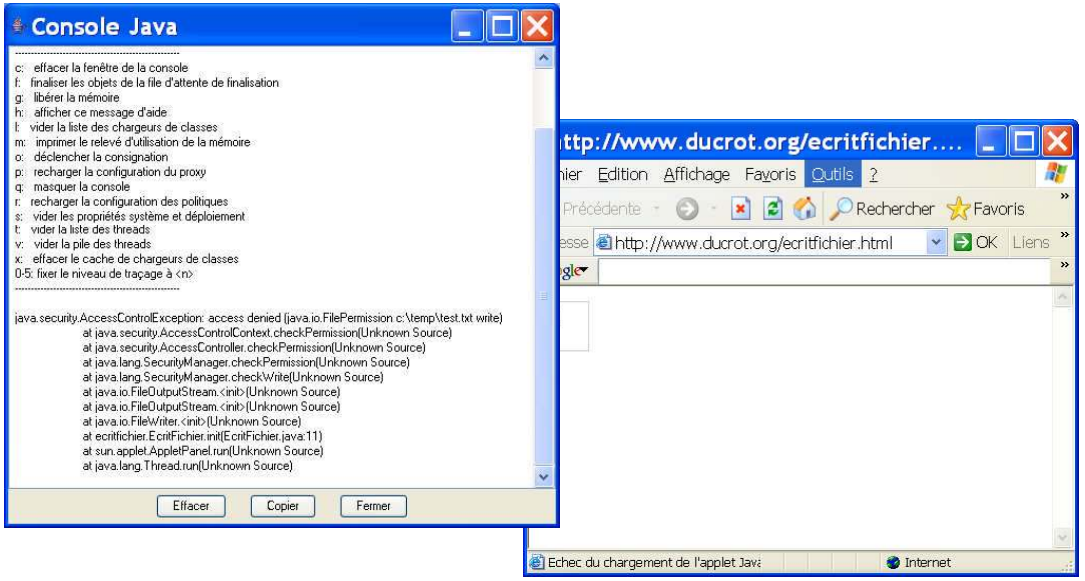
```
grant SignedBy "Jean,Fred", CodeBase http://www.trusted.com
{
        permission java.io.permission "c:\\temp\\*", "read" ;
} ;
```

Exemple d'applet

```
package ecritfichier;
import java.applet.Applet;
import java.io.* ;
public class EcritFichier extends Applet
{
   public void init ()
   {
         try {
                   FileWriter fichier = new FileWriter ("c:\\temp\\test.txt") ;
                  BufferedWriter buffer = new BufferedWriter (fichier) ;
                  fichier.write("test ecriture") ;
                 fichier.flush() ; fichier.close () ;

         } catch (IOException e)  { System.err.println ("Erreur fichier: " + e.getMessage () ) ; }
   }
}
```

```
<HTML>
<APPLET CODE="ecritfichier.EcritFichier.class" ARCHIVE="ecritfichier.jar" WIDTH=50 HEIGHT=50>
</APPLET>
</HTML>
```

Exécution de l'applet



Création d'une stratégie de sécurité

- Création du fichier jar:
  jar cvf ecritfichier.jar ecritfichier.class*
- Dans *<java.home>\lib\security\java.security*, ajout de la ligne:
  policy.url.3=file:${java.home}/lib/security/ecritfichier.policy
- Création du fichier *<java.home>\lib\security\ecritfichier.policy*, soit manuellement, soit avec l'outil *policytool de la jdk:*

```
/* AUTOMATICALLY GENERATED ON Mon Apr 04 16:52:09 CEST 2005*/
/* DO NOT EDIT */

grant {
        permission java.io.FilePermission "c:\\temp\\*", "write";
};
```

Signature de l'applet

- Génération d'une paire de clés:

  keytool -genkey -alias TestCle -keystore trousseau

- Signature de l'applet:
  - Signature de la clé publique par un tiers certificateur (moyennant finance), par exemple:

    Verisign **http://www.verisign.com**

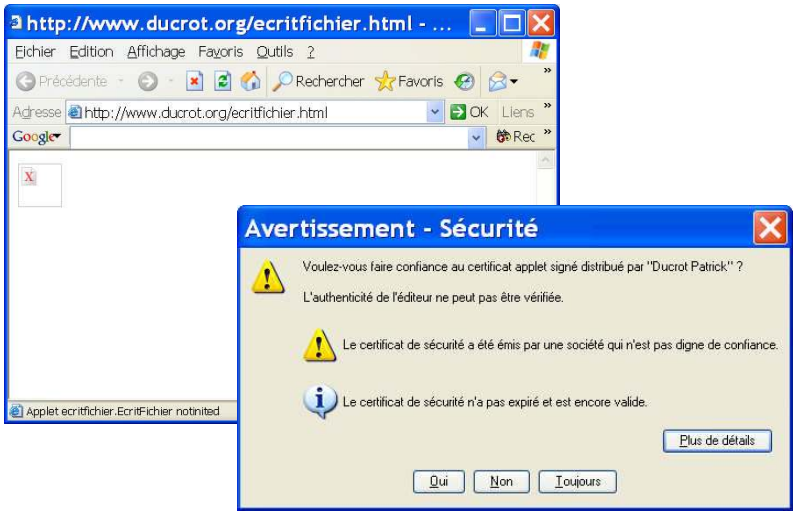
    Thawte **http://www.thawte.com**

    On extrait le certificat qui pourra être authentifié:

    keytool -export -keystore trousseau -alias TestCle -file certificat.cer
  - Auto signature du fichier jar à des fins de test:

    jarsigner -keystore trousseau ecritfichier.jar TestCle

Exécution de l'applet signée



Obfuscation de code

Obfuscation de code

- Après compilation d'un fichier source java, beaucoup d'informations sont stockées dans les fichier de bytecode.
- La décompilation permet de récupérer le code source intégralement (sans les commentaires ;)
- Il est donc nécessaire de brouiller le code avant diffusion de celui-ci.
- Des exemples de décompilateurs:
  - Historiquement, mocha en 1996 par Hanpeter Van Vliet
  - DJ Java Decompiler:
    http://members.fortunecity.com/neshkov/dj.html

Exemple code source Java

```
/**
  * Classe Thermometre:
  *     - Affiche la temperature a chaque mofigication de la temperature
  *     - Implemente l'interface TempChangeListener pour s'abonner aupres des objets Temperature
 */
public class Thermometre implements TempChangeListener
{
   public Thermometre ()
   {
      /**
       * Constructeur:
       *             - Creation d'un objet temperature
       *             - Enregistrement de l'ecouteur d'evenement
       *             - Lancement du thread pour test
       */
      Temperature Temp = new Temperature () ; Temp.addTempChangeListener (this);  Temp.start () ;  // On
        lance le Thread de Temperature
   }
   /**
    * Methode appelee a chaque modification de la temperature
   */
   public void tempChange (TempChangeEvent evt) {  System.out.println (evt.getTemperature()) ; }
}
```

Code après décompilation

```
public class Thermometre
    implements TempChangeListener
{

    public Thermometre()
    {
        Temperature Temp = new Temperature();
        Temp.addTempChangeListener(this);
        Temp.start();
    }

    public void tempChange(TempChangeEvent evt)
    {
        System.out.println(evt.getTemperature());
    }
}
```

Des exemples d'obfuscateur

- Zelix Class Master: *http://www.zelix.com*
  - Obfuscation des noms et du code.
  - Cryptage des chaînes.
  - Suppression des classes, méthodes, données non utilisées.
  - Interface graphique et langage de script
  - Support de la J2ME et J2EE
- yGuard: *http://www.yworks.com*
- ProGuard: *http://proguard.sourceforge.net*
- CodeShield: *http://www.codingart.com*

Classe après obfuscation (zkm)

```
public class b
    implements a
{

    public b()
    {
        int i = c.c;
        super();
        c c1 = new c();
        c1.a(this);
        c1.start();
        if(d.b != 0)
            c.c = ++i;
    }

    public void a(d d1)
    {
        System.out.println(d1.a());
    }
}
```

Programmation réseau

Modèle Client/Serveur en mode connecté

Serveur	Client
- Création d'un objet *ServerSocket* pour l'ouverture du service - Attente d'une demande de connexion (méthode *accept ()* qui retourne une socket de service) - Facultativement: création d'un thread pour gérer les échanges avec le client - Echange d'informations avec le client (InputStream,OutputStream) avec le client. - Fermeture socket (méthode *close()*).	- Création d'un objet Socket. - Connexion sur le serveur. - Echange d'informations avec le serveur (InputStream, OutputStream) - Fermeture socket (méthode *close ()*).

Les sockets en mode connecté côté client

- **Utilisation de la classe *java.net.Socket.***
- **Quelques constructeurs:**
  - ***public Socket (String host, int port) throws UnknownHostException, IOException ;***
  - ***public Socket (InetAddress address, int port) throws IOException ;***
    - host : Nom du serveur
    - port : Numéro de port
    - address : Classe contenant l'adresse IP
- **Quelques méthodes:**
  - ***public void close();***
  - ***public InetAddress getInetAddress();***
  - ***public InputStream getInputStream();***
  - ***public int getLocalPort();***
  - ***public OutputStream getOutputStream();***
  - ***public int getPort();***

Exemple de client en mode connecté

```
package socket;
import java.net.* ;
import java.io.* ;
public class Heure
{
   public static void main (String args [])
   {
      Socket s = null;    PrintStream flux = null;
      try {
            s = new Socket ("e450c.ecole.ensicaen.fr",2000) ;
            flux = new PrintStream (s.getOutputStream (),true) ;  flux.println ("heure") ;
           BufferedReader reponse = new BufferedReader (new InputStreamReader (s.getInputStream ()));
          System.out.println (reponse.readLine ()) ;

         }  catch (IOException e)
            {
               System.err.println (""Erreur: " + e.getMessage ()) ;
               System.exit (1) ;
            }
   }
}
```

Les sockets en mode connecté côté serveur

- **Utilisation la classe *java.net.ServerSocket.***

- **Quelques constructeurs:**
  - *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:**
  - *public Socket accept() ;*
  - *public void close() ;*
  - *public InetAddress getInetAddress() ;*
  - *public int getLocalPort() ;*

Exemple de serveur en mode connecté 1/2

```
package socket;

import java.net.* ;
import java.io.* ;
import java.util.Date ;

public class ServeurHeure
{
  public static void main (String args [])
  {
    ServerSocket s = null;
    PrintStream flux = null;
    try {
          s = new ServerSocket (2000) ;
        }  catch (IOException e)
           {
             System.err.println ("Erreur socket " + e) ;
             System.exit (1) ;
           }
```

Exemple de serveur en mode connecté 2/2

```
while (true)
{
    try {
            Socket service = s.accept () ;
            BufferedReader requete =
                    new BufferedReader (new InputStreamReader (service.getInputStream ()));

            if (requete.readLine ().equals ("heure"))
            {
                Date date = new Date () ;
                flux = new PrintStream (service.getOutputStream (),true) ;
                flux.println (date) ;
            }
        } catch (IOException e) { System.err.println (e.getMessage()); System.exit (1) ; }
  }
 }
}
```

Les sockets en mode non connecté

- Utilisation de la classe *java.net.DatagramSocket*
- Les constructeurs:
  - *public DatagramSocket () ;*
  - *public DatagramSocket (int port) ;*
  - *public DatagramSocket (int port, InetAddr laddr) ;*
- Quelques méthodes:
  - *public void receive (DatagramPacket p) ;*
  - *public void send (DatagramPacket p) ;*
- La classe *java.net.DatagramPacket* contient les données reçues ou à envoyer (tableau de bytes) et l'adresse (*java.net.InetAddress*) de l'expéditeur ou du destinataire.

Exemple de client en mode non connecté 1/2

```
public class HeureUDP {

  static public void main (String args [])
  {
     final int port = 2000 ;
     DatagramSocket socket = null ; DatagramPacket PacketRequest,PacketResponse ;
     InetAddress address = null ;
     byte [] reponse = new byte [30] ; String requete = "heure";

     try {
          socket = new DatagramSocket () ;
        } catch (SocketException e)
          {
            System.err.println ("Erreur socket " + e) ;
            System.exit (1) ;
          }
     try {
          address = InetAddress.getByName ("localhost") ;
        } catch (IOException e)
          {
             System.err.println ("serveur inconnu") ;
             System.exit (1) ;
          }
```

Exemple de client en mode non connecté 2/2

```
PacketRequest = new DatagramPacket (requete.getBytes(),requete.length(),address,port);
PacketResponse = new DatagramPacket (reponse,reponse.length) ;
try {
    socket.send (PacketRequest) ;
    socket.receive(PacketResponse);
    System.out.println ( new String (PacketResponse.getData())) ;
  } catch (IOException e)
    {
      System.err.println ("Erreur  de lecture ou d'ecriture " + e) ;
      System.exit (1) ;
    }
 }
}
```

Exemple de serveur en mode non connecté 1/2

```
package socket;

import java.net.* ;
import java.io.* ;
import java.util.Date ;

public class ServeurHeureUDP {

  public static void main (String args [])
  {
     DatagramSocket socket = null ;
     byte buffer [] = new byte [30] ;
     DatagramPacket packet = new DatagramPacket (buffer,buffer.length) ;
     try {
           socket = new DatagramSocket (2000) ;
        } catch (SocketException e)
          {
            System.err.println ("Erreur socket " + e.getMessage ()) ;
            System.exit (1) ;
          }
```

Exemple de serveur en mode non connecté 2/2

```
while (true)
    {
      try {
           socket.receive(packet);
           String requete = new String (packet.getData());
           requete = requete.substring (0,packet.getLength()) ;
           if ( requete.equalsIgnoreCase("heure") == true)
           {
              String date = (new Date ()).toString() ;
              packet.setData (date.getBytes()); packet.setLength(date.length());
              socket.send (packet) ;
           }
          } catch (IOException e)
           {
              System.err.println ("Erreur reception " + e) ;
              System.exit (1) ;
           }

    }
  }
}
```

Une classe évoluée: java.net.URL

- **Quelques Constructeurs:**
  - *public URL (String spec) ;*
  - *public URL (String protocol, String host, int port, String file) ;*

spec	Protocole + Nom machine + Nom fichier
protocol	http,ftp,gopher,file
host	Nom de la machine distante
port	Port de communication
file	Nom complet du document

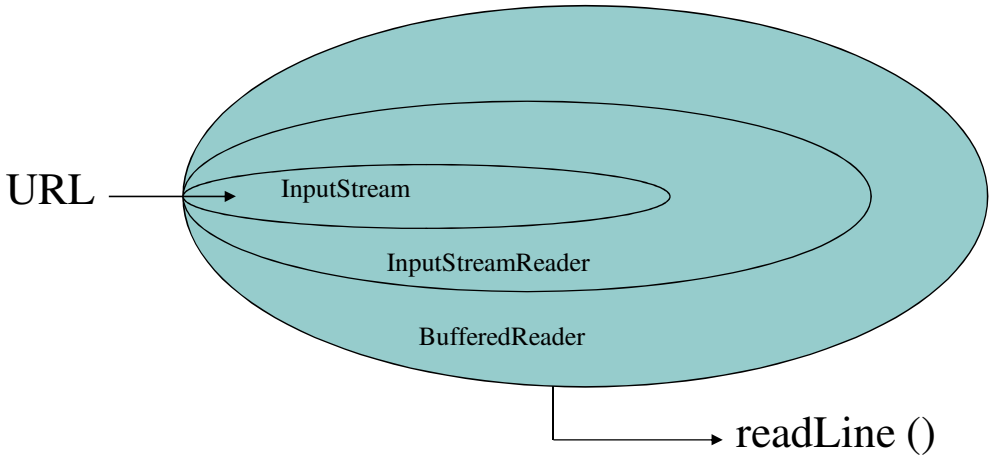
- **Quelques méthodes:**
  - **public InputStream openStream () ;**
  - **public URLConnection openConnection () ;**

    java.net.URLConnection* est une classe abstraite qui est utilisée comme classe de base de *java.net.HttpURLConnection
- **<u>Remarque:</u> Les méthodes *getCodeBase ()* et *getDocumentBase ()* de la classe *java.applet.Applet* retournent des objets *java.net.URL*.**

Exemple d'utilisation de la classe URL

```
import java.io.* ;
import java.net.* ;
public class LireURL
{
   static public void main (String args [])
   {
      try {
            URL url = new URL ("http://www.ensicaen.fr") ;
            InputStream is = url.openStream () ;
             InputStreamReader ir = new InputStreamReader (is) ;
            BufferedReader br = new BufferedReader (ir) ;
            while (br.ready () == true)
                  System.out.println (br.readLine ()) ;
            br.close () ; ir.close () ; is.close () ;
        } catch (Exception e)
          {
                  System.err.println ("Erreur " + e) ;
          }
   }
}
```

Flux mis en œuvre dans l'exemple



Extrait de l'exécution de l'application "LireURL"

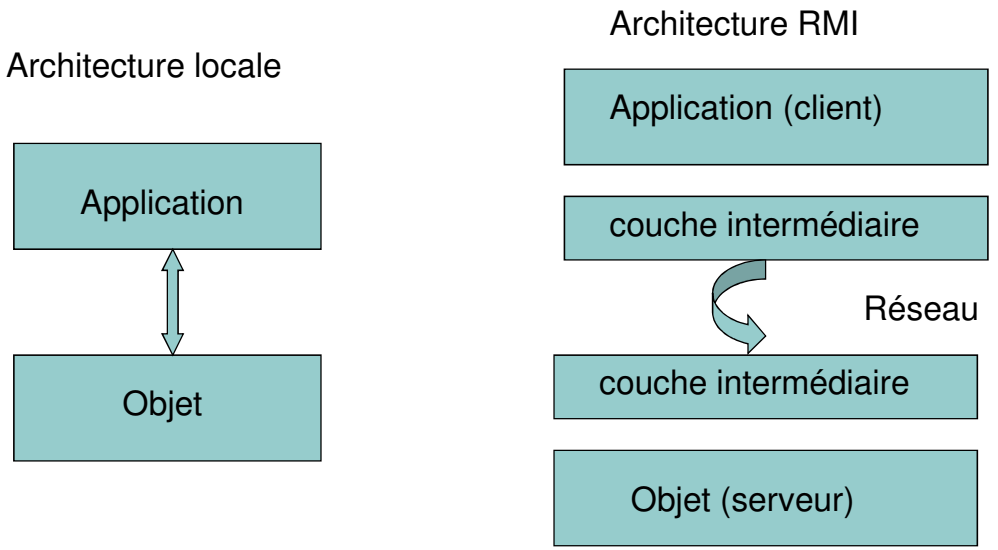
```
D:\travail>java LireURL
<html>
<script language="JavaScript">
</script>
<head>
<meta name="description"
content="Sommaire: ENSICAEN, Ecole Nationale Superieure d'Ingenieurs de Caen et Centre de
      Recherche">
<meta name="keywords"
 content="ENSICAEN,ISMRA,I.S.M.R.A,ENSI CAEN,ENSI CAEN
      ISMRA,CAEN,NORMANDIE,ensicaen,ismra,ENSI CAEN
      ISMRA,ENSI,Informatique,Microelectronique,Instrumentation,Chimie fine,Genie
      Informatique,latems,monetique,Instru,Micro,Genie Chimique,Ecole,Ingenieur,Generaliste,Ensi
      Caen,CAEN,Normandie,Ecole Nationale Superieure d'Ingenieurs de
      Caen,ISMRA,ENSICAEN,ISMRA-ENSICAEN,CAEN ENSI,I.S.M.R.A.,Institut desSciences de la
      Matiere et du Rayonnement,ENSI CAEN,ISMRA,ENSI CAEN,ISMRA,ENSI CAEN,ISMRA,ENSI
      CAEN,ISMRA,ENSI CAEN,ISMRA,ENSI CAEN,ISMRA,ENSI CAEN,ISMRA,ENSI
      CAEN,ISMRA,ENSI CAEN,ISMRA,ENSI CAEN,ISMRA,ENSI CAEN,Engineer,Engineering
      school,Computer Science,Electronics,Chemistry,Instrumentation,France">
<title>ENSICAEN</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
...
```

Remote Method Invocation

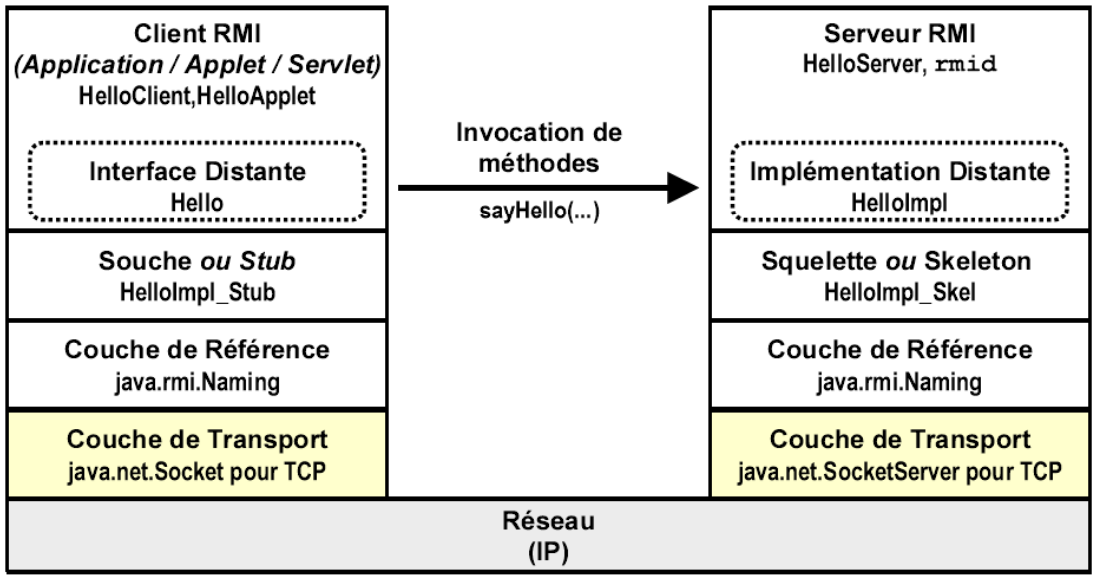
Généralités

- Les approches traditionnelles pour développer des applications réseaux à base de sockets sont lourdes.
- Les RMI vont permettre de distribuer une application sur plusieurs machines.
- Une application fera appel à des méthodes sur des objets localisés sur des machines distantes.
- RMI se charge de faire transiter les paramètres et la valeur de retour.
- Les communications pourront être sécurisées grâce à un objet RMISecurityManager.

Architecture



Architecture RMI



Stub

- Classe spéciale générée par la commande rmic.
- Transforme un appel de méthode en une suite d'octets à envoyer sur le réseau (marshaling).
- Reconstruit le résultat reçu sous le même format (unmarshaling)
- Format d'un appel de méthode:
  - identificateur de l'objet distant
  - identificateur de la méthode
  - paramètres sérialisés

Skeleton

- Classe spéciale générée par la commande rmic (cette classe n'est plus générée depuis la version 1.2 du protocole).
- Reçoit la demande d'invocation distante.
- Reconstruit les paramètres.
- Trouve l'objet distant et appelle la méthode.
- Retourne le résultat.

Propriétés d'un objet distant

- Un objet distant se manipule comme un objet local.
- L'invocation d'une méthode distante est identique à l'invocation d'une méthode locale.
- Les paramètres d'un appel distant et le résultat renvoyé sont transmis:
  - par valeur pour les types scalaires
  - par recopie sous format sérialisés pour les objets
- Si le fichier .class correspondant à un type de paramètre ou de la valeur de retour n'est pas disponible localement, il est chargé dynamiquement (RMIClassLoader).

Chargement dynamique des classes

- Si le client est une applet, toutes les classes apparaissant dans le code du client sont chargées depuis le codebase spécifié par la propriété *java.rmi.server.codebase*.
- Si le client est une application, les classes seront recherchées:
  - dans le CLASSPATH
  - à l'URL spécifiée par *java.rmi.server.codebase*
- Une classe peut être chargée explicitement:
  - class c = RMIClassLoader.loadClass (http://xxxxx,"nom_classe) ;

Interface *java.rmi.Remote*

- L'interface distante doit être publique et hériter de *java.rmi.Remote*
- Chaque méthode de l'interface distante doit déclarer *java.rmi.RemoteException* dans sa clause *throws.*
- Un objet distant passé en argument ou en valeur de retour doit être déclaré en tant qu'interface distante.

Interface *java.rmi.Remote*

- Définition d'une interface déclarant les méthodes distantes exposées:

```
import java.rmi.* ;
public interface Hello extends Remote
{
        public String envoieHello () throws RemoteException;
}
```

- L'interface étendant l'interface *java.rmi.Remote* permet de marquer les méthodes distantes mises à disposition du client.

RMI côté serveur

- Le serveur doit contenir une classe qui étende *java.rmi.server.UnicastRemoteObject* (qui utilise les classes *Socket* et *ServerSocket*) et qui implémente l'interface précédente.
- Le constructeur de cet objet distant doit être défini explicitement (il doit émettre *java.rmi.RemoteException*).
- Une ou plusieurs instances de l'objet distant doivent être créés.
- Les instances créées doivent être enregistrées auprès du registre d'objets distants.
- La classe *java.rmi.Naming* permet d'archiver, de lister, et de récupérer des objets auprès d'un registre. Lorsque qu'un client transmet une URL « *rmi://domaine/ObjetDistant* », une référence est transmise en retour (en fait une référence sur une portion de code local capable de communiquer à travers le réseau).
- Création des « stubs » et « skeletons » .

RMI côté serveur

- Définition d'une classe implémentant cette interface:

```
import java.rmi.server.* ;
import java.rmi.* ;
import java.net.* ;
public class serHello extends UnicastRemoteObject implements Hello
{
      public serHello () throws RemoteException { super () ; }
      public String envoieHello () throws RemoteException { return "Hello World" ; }
     public static void main (String args [])
     {
     System.setSecurityManager(new RMISecurityManager ()) ;
     try {
                              serHello h = new serHello () ;
                              Naming.rebind ("bonjour",h) ;
                              System.out.println ("Serveur pret") ;
               } catch (RemoteException e) { System.err.println ("RemoteException "+e) ; }
               catch (MalformedURLException e) { System.err.println ("Malformed "+e) ;}
     }
}
```

RMI côté serveur

- Compilation:
  javac serHello.java
- Génération du stub et du squelette:
  rmic serHello
- Fichiers générés:
  Hello.class serHello.class [serHello_Skel.class] serHello_Stub.class
- Lancement du registre Naming (port par défaut: 1099):
  start rmiregistry (windows)
  rmiregistry & (unix)
  Par programme: *LocateRegistry.createRegistry (port)*
- Lancement de l'application:
  java -Djava.security.policy=java.policy serHello
- Fichier java.policy:

```
grant {
  permission java.net.SocketPermission "*:1024-65535", "accept, listen, connect, resolve";
} ;
```

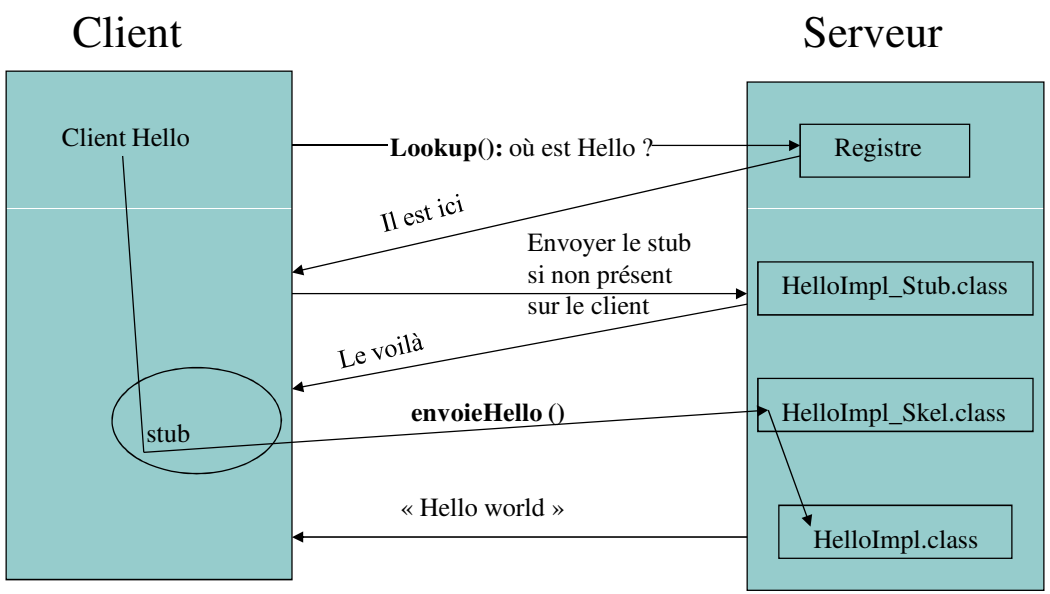
RMI côté client

- Le programme client doit rechercher et rapatrier l'interface distante.
- Les méthodes distantes peuvent ensuite être exécutées.

RMI côté client

```
import java.rmi.* ;
public class cliHello
{
        public static void main (String args [])

      {

              System.setSecurityManager(new RMISecurityManager ()) ;
              try {
               Hello h = (Hello) Naming.lookup (“rmi://e450c.ecole.ensicaen.fr/bonjour") ;
               String message = h.envoieHello () ;
              System.out.println ("recu : " + message) ;
         } catch (Exception e)
         {
              System.err.println ("Exception : " + e) ;
         }
    }
}
```

Interaction client/serveur



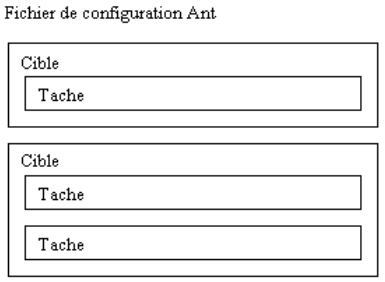
ANT

ANT

- Projet du groupe Apache-Jakarta pour permettre la construction d'applications (compilation, déploiement, …).
- Site officiel: *http://jakarta.apache.org/ant*
- Ant s'inspire des Makefile d'unix mais est multi plateforme et ne propose pas le "syndrome de la tabulation" en utilisant des fichiers de configuration au format XML.
- Exécution de ant:
  ant [-buildfile fichier.xml] [cible]
- Variables d'environnement nécessaires à l'exécution de ant:
  ANT_HOME *JAVA_HOME* *PATH*

Ant: fichier de configuration

- Le fichier de configuration propose un ensemble de cibles.
- Une cible contient une ou plusieurs tâches à exécuter.
- Les cibles peuvent être dépendantes entre elles.



ANT: fichier de configuration

- Le fichier de configuration commence par le préfixe:

  <?xml version="1.0">

- La balise racine est le projet: *<project>*
- A l'intérieur du projet on trouve:
  - Les cibles
  - Les propriétés
  - Les tâches

La balise <project>

- La balise *<project>* contient des attributs:
  - *name*: nom du projet
  - *default*: détermine la cible par défaut
  - *basedir*: indique le répertoire racine pour tous les répertoires utilisés par leur nom relatif

- Exemple:

```
<project name="nomprojet" default="compile" basedir=".">
```

Les commentaires

- Les commentaires sont inclus dans les balises <!-- et -->

- Exemple:

<!-- ces deux lignes sont

des commentaires -->

Les propriétés

- Les propriétés permettent de définir des variables qui pourront être utilisées dans le projet
- Les propriétés peuvent être définies sur la ligne de commande (option *-D*) ou par la balise *<property>*
- Exemple:

  <property name="repertoire" value="travail" />
  <property file="proprietes.properties />

- Une propriété s'utilise avec la syntaxe *${nompropriété}*

Propriétés prédéfinies

basedir	chemin absolu du répertoire de travail (défini dans la balise <project>)
ant.file	chemin absolu du fichier de configuration
ant.java.version	numéro de version de la JVM exécutant ant
ant.project.name	nom du projet en cours d'exécution

Les cibles

- La balise *<target>* permet de définir une cible constituée par un certain nombre de tâches.

- La balise <target> possède plusieurs attributs:
  - *name* obligatoire. nom de la cible
  - *description* optionnel. description de la cible
  - *if* optionnel. conditionne l'exécution à l'existence d'une propriété

Les tâches

- Une tâche est une unité de traitement à exécuter.
- Une tâche est une classe Java implémentant l'interface *org.apache.ant.Task*
- De nombreuses tâches sont définies dans ant (pour en connaître la liste il faut se référer à la documentation fournie avec ant ou à l'adresse *http://ant.apache.org/manual/index.html*).
- Les tâches prédéfinies permettent le développement java, la manipulation du système de fichiers, des archives, etc.

Exemple de fichier de configuration

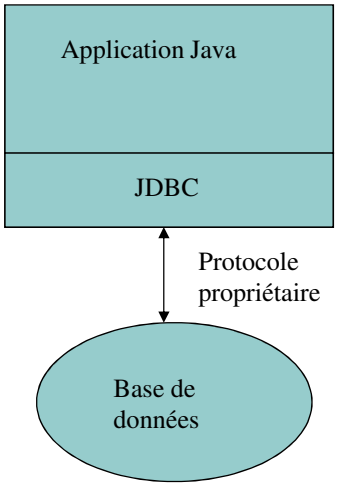
```
<?xml version="1.0"?>
<project name="project" default="runchatclient">
    <description>
            test ant
    </description>
   <property name="src" value="src" />
   <property name="dst" value="classes" />

   <target name="chatclient"  description="Client chat RMI">
        <javac srcdir="${src}/rmichat" destdir="${dst}"  />
        <rmic classname="rmichat.ClientRMI" base="${dst}" />
    </target>

    <target name="runchatclient">
        <java classname="rmichat.ChatClient" classpath="${dst}" fork="true" >
                    <arg value="localhost" />
                    <sysproperty key="java.security.policy"  value="java.policy" />
        </java>
    </target>

    <target name="archive" >
         <jar destfile="chat.jar" basedir="${dst}" />
    </target>
</project>
```

Interaction Java/bases de données

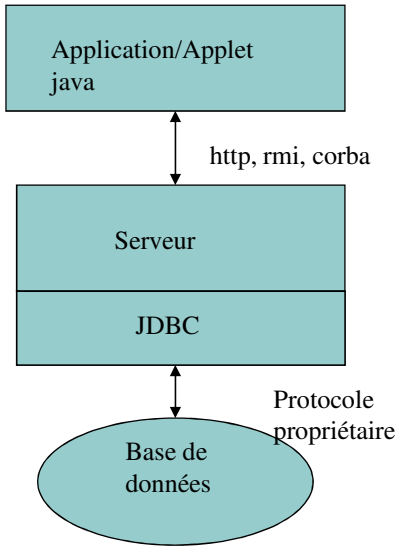
JDBC: Java DataBase Connectivity

- C'est une API java qui permet aux applications java de communiquer avec les gestionnaires de base de données dans un langage universel (comparable à ODBC).
- Les applications peuvent ainsi être indépendantes de la base de données utilisées.
- Un pilote JDBC permet:
  - Etablir une connexion avec une base de données.
  - Envoyer des requêtes SQL.
  - Traiter les résultats.

Architecture d'utilisation



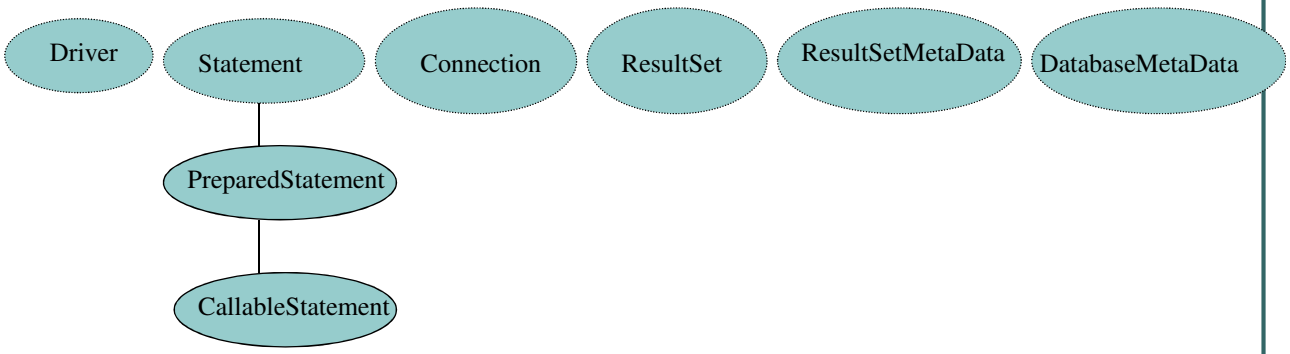
Modèle 2/3



Modèle 3/3

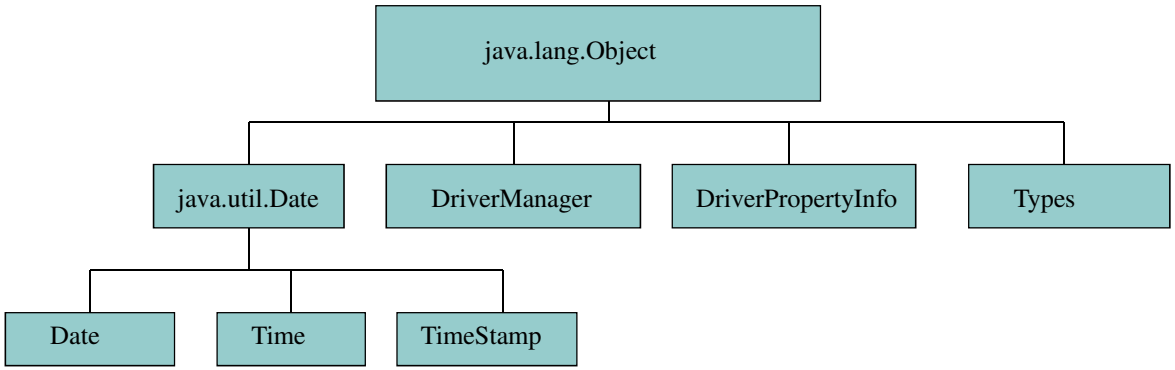
Le package java.sql

- Les interfaces



Le package java.sql

- Les classes



Pilotes JDBC

- Il est nécessaire de disposer du pilote JDBC pour interagir avec la base de données.
- Un pilote JDBC peut être commercial (exemple: Oracle) ou gratuit.
- Une liste de pilotes est disponible à l'adresse:

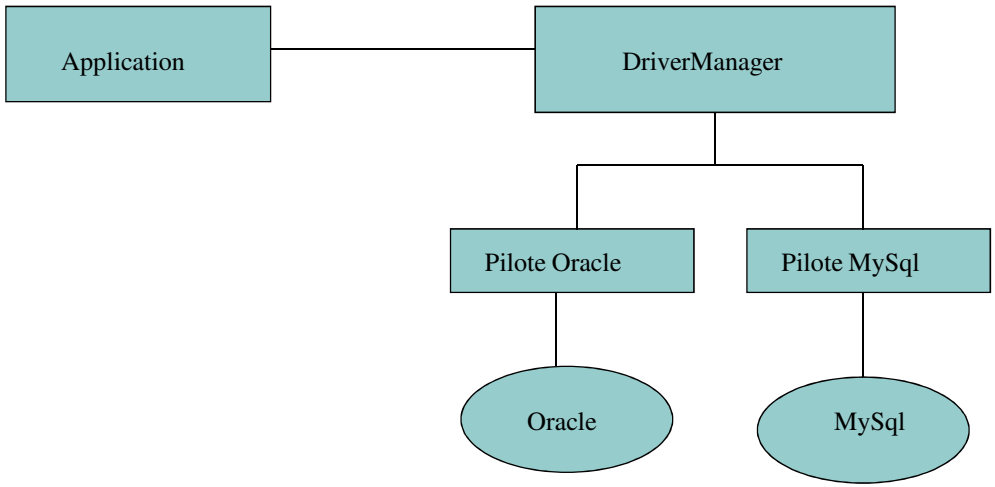
  http://industry.java.sun.com/products/jdbc/drivers

Pilotes JDBC

- Il existe 4 types de pilote JDBC:

type 1	Pilotes accédant aux bases de données grâce à une technologie de ponts. Exemple: le pont ODBC. Cela requiert en général d'installer du code natif sur le poste client.
type 2	Le code Java appellent les méthodes C/C++ natives livrées par les éditeurs de base de données. Cela requiert d'installer du code natif sur le poste client.
type 3	Ces pilotes fournissent au client une API générique. Le pilote JDBC sur le client communique au moyen de sockets avec une application intermédiaire sur le serveur qui convertit les requêtes du client en appel API spécifique du pilote souhaité.
type 4	Via des sockets java, ces pilotes interagissent directement avec le gestionnaire de la base de données.

Connexion à la base de données



Classes de connexion

java.sql.Driver	Interface devant être implémentée par les classes de chargement des pilotes JDBC.
java.sql.DriverManager	Un objet *DriverManager* va tenter de localiser le pilote JDBC et charger les classes correspondantes.
java.sql.Connection	Un objet *Connection* représente le lien entre l'application et la base de données. Toutes les requêtes SQL transmises et le retour des résultats s'effectueront à travers cet objet.

Exemple de connexion

```
private Connection Conn ;

try {
            Class.forName("org.gjt.mm.mysql.Driver").newInstance();
    }
        catch (Exception e)
        {
            System.err.println(" Probleme avec le driver JDBC: "  + e);
            return ;
        }

      try {
            Conn = DriverManager.getConnection("jdbc:mysql://e450c.ecole.ensicaen.fr/",<login>,<password>) ;
        }
          catch (SQLException e)
         {
            System.err.println("Probleme ouverture: "  + e) ;
            return ;
         }
```

Classes d'accès à la base de données

java.sql.Statement	Classe à utiliser pour les requêtes SQL élémentaires. Quelques méthodes: *public ResultSet executeQuery (String sql) throws SQLException* *public int executeUpdate (String sql) throws SQLException* *public boolean execute(String sql) throws SQLException*
java.sql.ResultSet	Une instance de cette classe contient une rangée de données extraite de la base par une requête SQL et offre plusieurs méthodes chargées d'en isoler les colonnes. La notation suivante est utilisée: <type> get<type> (int | String) Exemple: *String getString (« title »)* A un instant donné, un objet *ResultSet* ne peut contenir plus d'une rangée mais propose une méthode *next()* permettant de référencer la rangée suivante.
java.sql.PreparedStatement	Cette classe est utilisée pour pouvoir envoyer au gestionnaire de base de données une requête SQL pour interprétation mais non pour exécution. Cette requête peut contenir des paramètres qui seront renseignés ultérieurement.

Exemple de code

```
Statement stmt = conn.createStatement () ;

ResultSet rs = stmt.executeQuery (« SELECT a,b,c FROM Table1 ») ;

while (rs.next ())
{
    int x = rs.getInt ("a") ;
    String s = rs.getString ("b") ;
    float f = rs.getFloat ("c") ;
}
```

Exemple de code

```
PreparedStatement inst = con.preparedStatement ("UPDATE comptes
    SET solde = ? Where id = ?") ;

for (int i = 0 ; i < comptes.length ; i++)
{
  inst.setFloat (1,comptes [i].extraitSolde ()) ;
  inst.setFloat (2,comptes [i].extraitIdf ()) ;
  inst.execute ()
}
```

Aspect transactionnel

- Par défaut, les opérations sur la base de données sont en mode *auto-commit*. Dans ce mode, chaque opération est validée unitairement pour former la transaction.

- Pour rassembler plusieurs opérations en une seule transaction:
  - *connection.setAutoCommit(false);*
  - *connection.commit () ;*

- Retour en arrière:
  - *connection.rollback ();*

JavaBeans

Programmation Traditionnelle

- Limites:
  - Tout est à la charge du programmeur
  - Peu d'abstraction pour limiter la complexité
  - Maintenance et évolution difficiles
- Conséquences:
  - Besoins de compétences techniques
  - Fiabilité incertaine
  - Accroissement du temps de développement et de maintenance

Programmation par objet

- La programmation par objet améliore la situation:
  - Modules réutilisables
  - Les constituants sont représentés en classes
  - Héritage entre classes
- Certaines limites demeurent:
  - La réutilisation des modules demandent des connaissances techniques
  - Le code explicite les interconnexions entre modules

Programmation par composant

- Un composant est une pièce logicielle autonome.
- Une application pourra être conçue par assemblage de composants.
- Une application pourra être construite par des non-informaticiens.
- Évolution plus rapide des applications.

Programmation par composant

- Son implémentation n'est pas requise (boîte noire).
- Il exporte les interfaces fournies et requises.
- Interconnectable avec d'autres composants (inconnus à priori).
- Configurable, diffusable.
- Autodescriptif.

Le modèle JavaBean

- Un JavaBean est un modèle de composants logiciels pour plate-formes JAVA.
- Un Javabean est réutilisable et peut être manipulé visuellement par un outil de construction de programmes.
- Un JavaBean est simple à créer et à utiliser.
- Un JavaBean est portable sur toute plate-forme supportant l'environnement JAVA.
- Ce modèle est disponible depuis la JDK 1.1

Type de beans

- Bean Visuel
  - Boutons, icônes, …
- Bean non visuel
  - Accès à une base de données, …
- Bean composite
  - Grapheur, feuille de calcul …

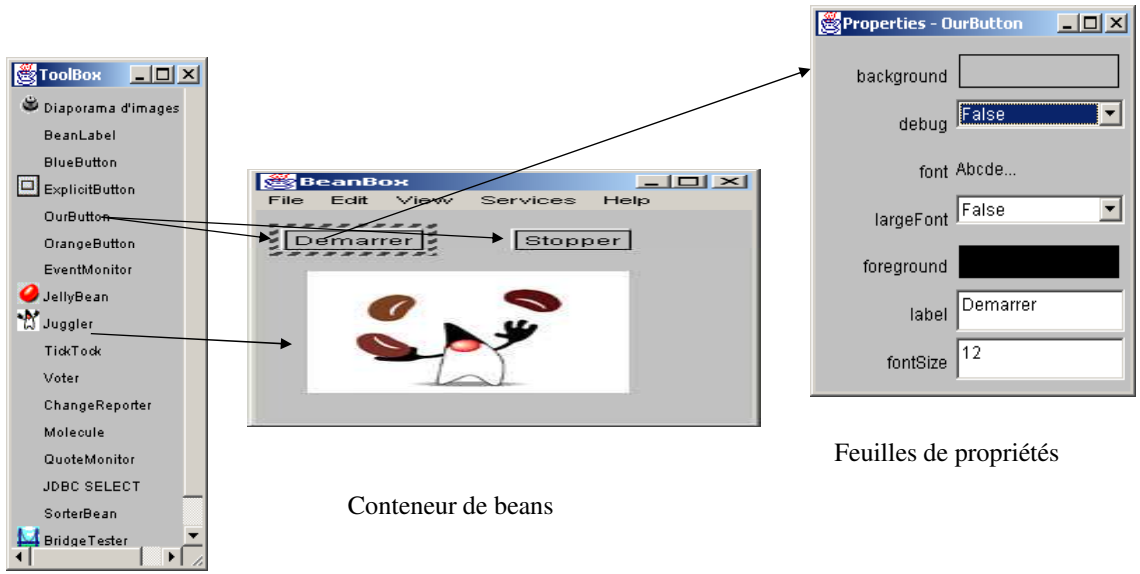
Caractéristiques d'un bean

- Un bean est caractérisé par:
  - Ses propriétés
  - Les méthodes exportées
  - Les événements qu'il peut émettre et recevoir
- Un bean peut être connecté à d'autres beans
- La communication entre beans repose sur le modèle événement/action

Caractéristiques d'un bean

- Un bean doit être:
  - Introspectable
  - Sérialisable (persistance)
  - Distribuable
  - Editable visuellement
- Un bean doit:
  - Respecter les règles de sécurité
  - S’adapter au multithreading

Exemple: la beanbox



Palette de beans

Conteneur de beans

Feuilles de propriétés

Descriptif d'un bean

- Un bean doit au minimum implémenter:
  - Une classe java respectant quelques conventions d'écriture.
  - Éventuellement une classe sans convention d'écriture mais implémentant l'interface *java.beans.BeanInfo*
- Un bean visuel doit hériter de *java.awt.Component* (ou une sous classe).

Exemple de bean (1/2)

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

public class CounterBean extends Canvas implements Serializable
{
    protected boolean stopped;
    protected int value;
    public CounterBean()
    {
          stopped = true; setSize(60,40); setBackground(Color.white);
    }
    public int value() { return value; }
    public void reset() { value = 0; }
    public void start() { stopped = false ; }
    public void stop() {stopped = true; }
```

Exemple de bean (2/2)

```
    public void step() {
          if (stopped == false) {
                    value++;
                    repaint () ;
          }
    }
    public void paint(Graphics g) {
          FontMetrics fm;
          Dimension dim;
          int strWidth, strAscent, centerBoxX, centerBoxY;
          String str = Integer.toString(value);
          dim = getSize(); fm = g.getFontMetrics();
          strWidth = fm.stringWidth(str); strAscent = fm.getAscent();
          centerBoxX = dim.width/2 - strWidth/2;
          centerBoxY = dim.height/2 + strAscent/2;
          g.drawString(str, centerBoxX, centerBoxY);
    }
}
```

Empaquetage des beans

- Un bean peut être constitué de plusieurs fichiers:
  - Fichiers .class (classe du bean, classe beaninfo, …)
  - Fichiers d'aide en html
  - Fichiers de ressources (icônes, sons, images, …)
  - …
- Les beans sont distribués sous forme d'archives java.

Archive java

- Une archive java est fichier .jar au format Zip contenant des fichiers compressés de natures diverses et un fichier MANIFEST.MF.
- Caractéristiques des archives java:
  - Multi-plateforme
  - Auto-descriptif
  - Sécurité et authentification
  - Téléchargeable par navigateur
- Il existe des API pour gérer des fichiers archives java (depuis la version 1.1)

Le fichier MANIFEST.MF

- Le fichier MANIFEST.MF est constitué:
  - D'un numéro de version
  - D'une liste d'informations pour chaque version respectant la structure suivante:

    Name : Nom_Fichier

    Attribut1 : valeur1

    Attribut2 : valeur2

    …
- Le fichier MANIFEST.MF est construit de façon automatique par l'outil d'archivage, éventuellement à partir d'un autre fichier MANIFEST.MF renseignant des attributs spécifiques.

Exemple de fichier MANIFEST.MF

```
Manifest-Version: 1.0
Created-By: Signtool (signtool 1.1)
Comments: PLEASE DO NOT EDIT THIS FILE. YOU WILL BREAK IT.

Name: k.class
Digest-Algorithms: MD5 SHA1
MD5-Digest: Gqz1Sya8TSAtHZPTEJomXg==
SHA1-Digest: f6KORQq/qBWAoblMsRzJUgzs4Qo=

Name: b.class
Digest-Algorithms: MD5 SHA1
MD5-Digest: wB5eyVTEAbVXU1IRykAK9g==
SHA1-Digest: ydSJ2/aK8Oz0BZY1gTUy2D98trw=

Name: y.class
Digest-Algorithms: MD5 SHA1
MD5-Digest: VIlRykROkOnofxP5Hdgawg==
SHA1-Digest: gsFMaM16H/L0TmTCEp959N6ZVIQ=
```

Fichier MANIFEST.MF & Beans

- 3 Attributs supplémentaires:
  - Java-Bean: attribut booléen permettant d'indiquer si la classe représente un bean.
  - Design-Time-Only: attribut booléen permettant d'indiquer si le fichier est seulement utile pour les environnements.
  - Depends-On: attribut permettant de spécifier une liste de fichiers associés avec un bean.

Exemple: CounterBean

Fichier: CounterBean.mf

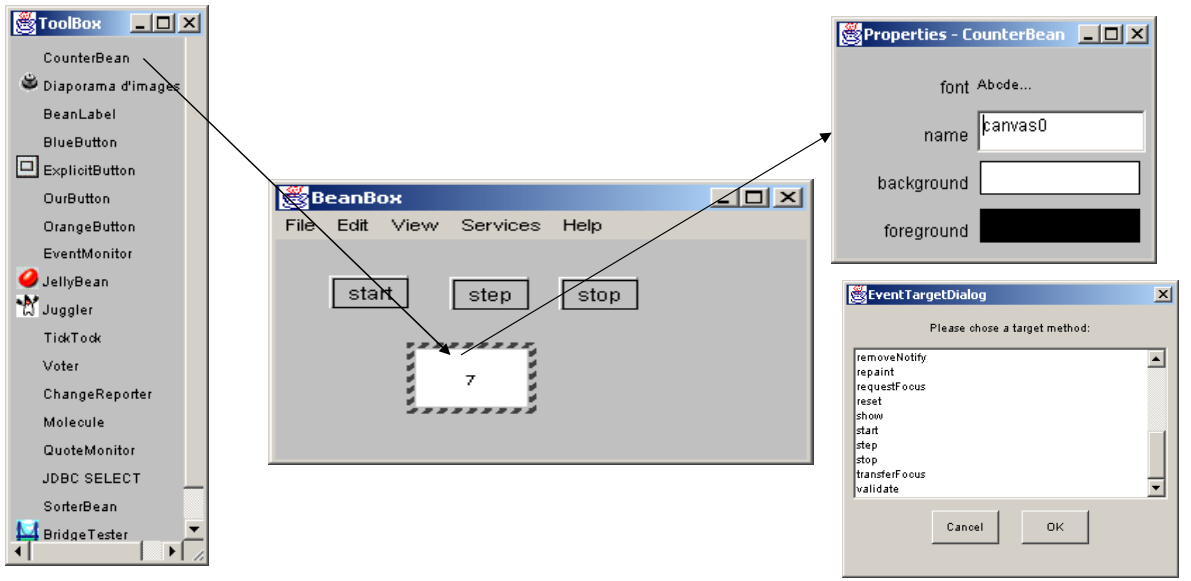
Manifest-Version: 1.0

Name: counterbean/CounterBean.class
Java-Bean: True

Commande d'archivage:

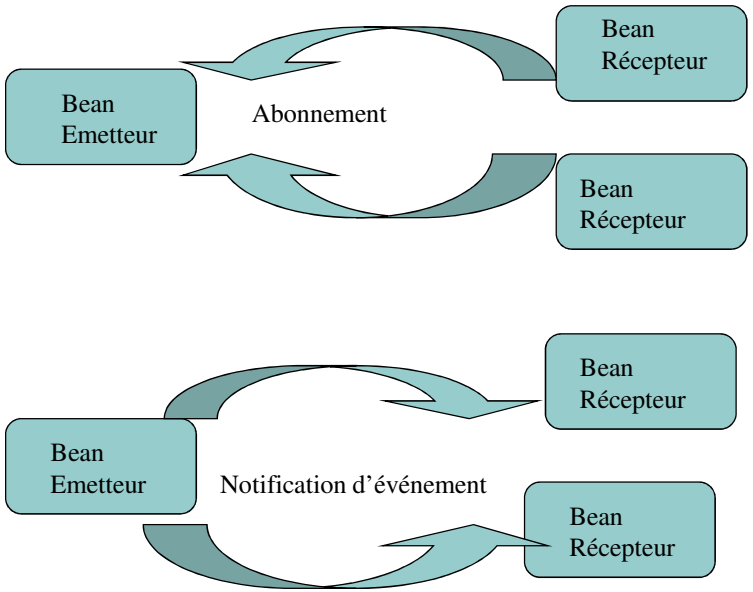
jar cvfm Counter.jar CounterBean.mf counterbean/.class*

Manipulation de CounterBean



Méthodes exposées par CounterBean

Communication entre beans



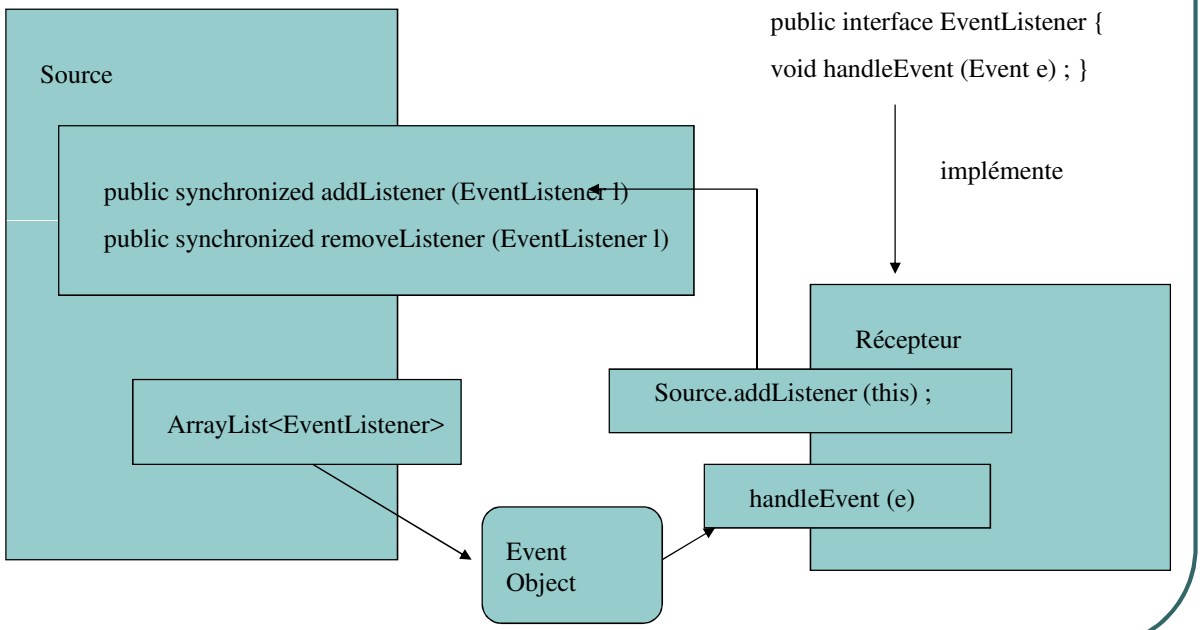
Communication entre beans

- Possibilités offertes:
  - Un bean peut émettre plusieurs types d'événements différents.
  - Un bean récepteur peut s'abonner auprès de plusieurs beans.
  - Un bean peut être à la fois émetteur et récepteur d'événements.
- Deux types de communications:
  - Unicast: un émetteur et un seul récepteur
  - Multicast: un émetteur et plusieurs récepteurs

Communication entre beans

- Les événements émis par un bean sont définis par l'existence de méthodes d'abonnement/désabonnement respectant des conventions précises d'écriture.
- Les événements sont représentés par des objets.
- La notification d'un événement à un bean récepteur est réalisée par envoi de messages.
- La signature de la méthode invoquée lorsqu'un événement survient est spécifiée par une interface Java que doivent implanter les beans intéressés.

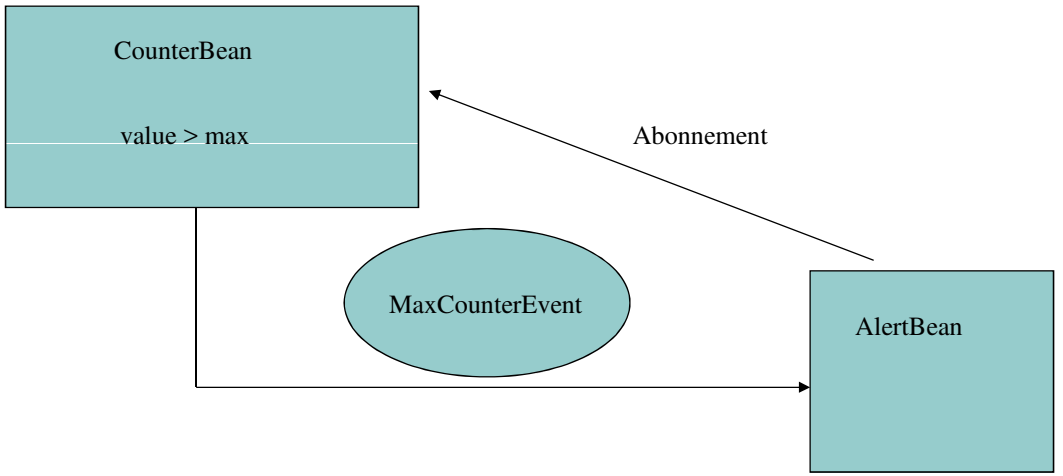
Mise en œuvre



Déclaration d'événements émis par un bean

- La classe du bean doit comporter deux méthodes publiques pour l'abonnement et le désabonnement des autres beans.
- La signature de ces deux méthodes doit respecter les conventions d'écritures suivantes:
  - Si E est un événement multicast:
    - void addEListener (EListener l)
    - void removeEListener (EListener l)
  - Si E est un événement monocast:
    - void addEListener (EListener l) throws java.util.TooManyListenerException
    - void removeEListener (EListener l)

Exemple sur CounterBean



Méthodes d'abonnement/désabonnement

```
private MaxCounterEventListener maxCounterListener ;


public void addMaxCounterEventListener (MaxCounterEventListener mcel) throws
       java.util.TooManyListenersException
    {
      if (maxCounterListener != null)
        throw new java.util.TooManyListenersException ("Un listener est deja en place") ;
      else
        maxCounterListener = mcel ;
    }

    public void removeMaxCounterEventListener (MaxCounterEventListener mcel) throws
      IllegalArgumentException
    {
      if (maxCounterListener == null || maxCounterListener != mcel)
        throw new IllegalArgumentException ("Le listener ne peut etre detruit") ;
      else
        maxCounterListener = null ;
    }
```

Les événements

- Un événement émis par un bean est représenté par un objet
- Règles pour la définition d'une classe d'événement:
  - Le nom de la classe est suffixé par *Event*
  - La classe doit hériter de *java.util.EventObject*
- Dans le cas d'un bean visuel, les événements peuvent être ceux de l'awt.

Exemple d'événement

```
import java.util.EventObject;

public class MaxCounterEvent extends EventObject
{
  private int CurrentValue ;

  public MaxCounterEvent(Object source,int cv)
  {
    super (source) ;
    CurrentValue = cv ;
  }

  public int getValue ()
  {
    return CurrentValue ;
  }
}
```

Notification de l'événement

```
public void step() {
            if (stopped == false)
            {
                value++;
                  if (value >= max) fireMaxCounterEvent () ;
                repaint () ;
            }
    }
private void fireMaxCounterEvent ()
    {
      if (maxCounterListener != null)
      {
        MaxCounterEvent evt = new MaxCounterEvent (this,value) ;
        maxCounterListener.maxReached (evt);
      }
    }
```

Interface de notification

- Une interface de notification sert à spécifier la signature de la méthode invoquée par le bean émetteur lorsque l'événement se produit.
- Règles pour la définition de l'interface:
  - Le nom de l'interface est composé du nom de l'événement suffixé par le mot *Listener*
  - L'interface doit hériter de *java.util.EventListener*
  - L'événement correspondant peut être transmis en paramètre (pas obligatoire).

Exemple d'interface

```
import java.util.EventListener;

public interface MaxCounterEventListener extends EventListener
{
  public void maxReached (MaxCounterEvent e) ;
}
```

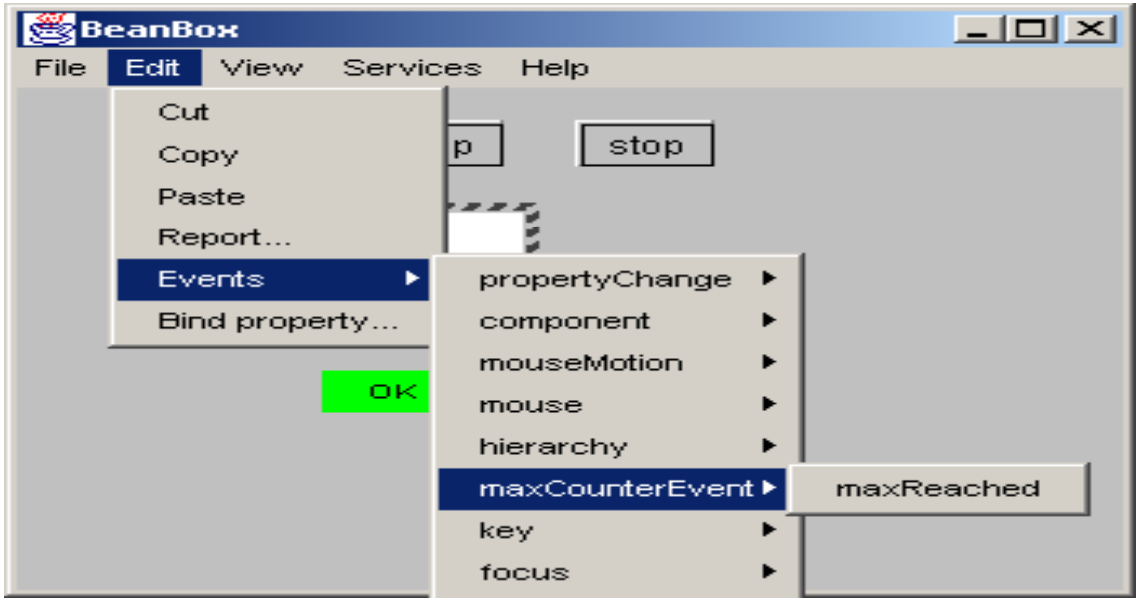
Bean récepteur

- Un bean intéressé par un type d'événement doit:
  - Implémenter l'interface de notification
  - S'enregistrer auprès du bean émetteur

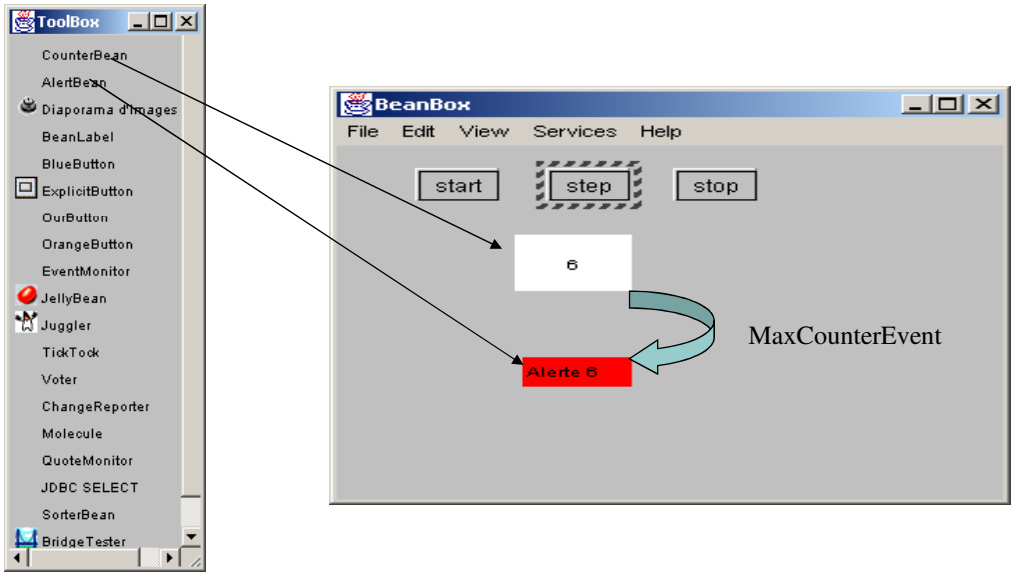
Exemple de bean récepteur

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

public class AlertBean extends Label implements Serializable, MaxCounterEventListener
{
  public AlertBean()
  {
    setBackground(Color.green);
    setText ("    OK     ") ;
  }

  public void maxReached (MaxCounterEvent evt)
  {
     setBackground (Color.red) ;
     setText ("Alerte " + evt.getValue()) ;
  }
}
```

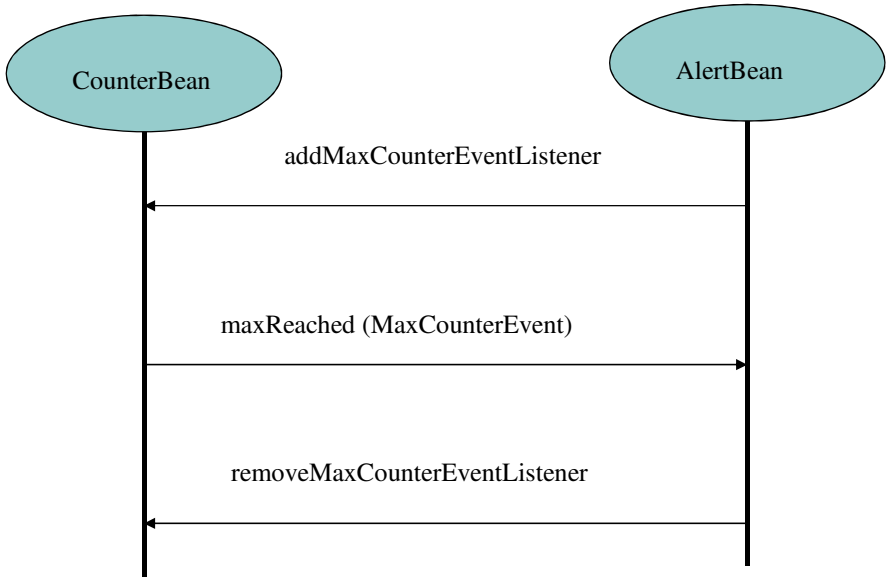
CounterBean



CounterBean



Interactions



Propriétés de beans

- Les propriétés d'un bean sont des attributs
  - Affectant son comportement ou son apparence
  - Reconnus par les environnements d'assemblage visuel
  - Manipulables par programme en invoquant les méthodes
- Les propriétés font généralement partie de l'état persistant d'un objet.
- Une propriété est définie par l'existence de méthodes publiques respectant des conventions précises d'écriture.
- 3 modes d'accès possibles aux propriétés: lecture, écriture, lecture/écriture.
- 4 types de propriétés disponibles:
  - Propriétés scalaires, propriétés indexées
  - Propriétés liées, propriétés contraintes

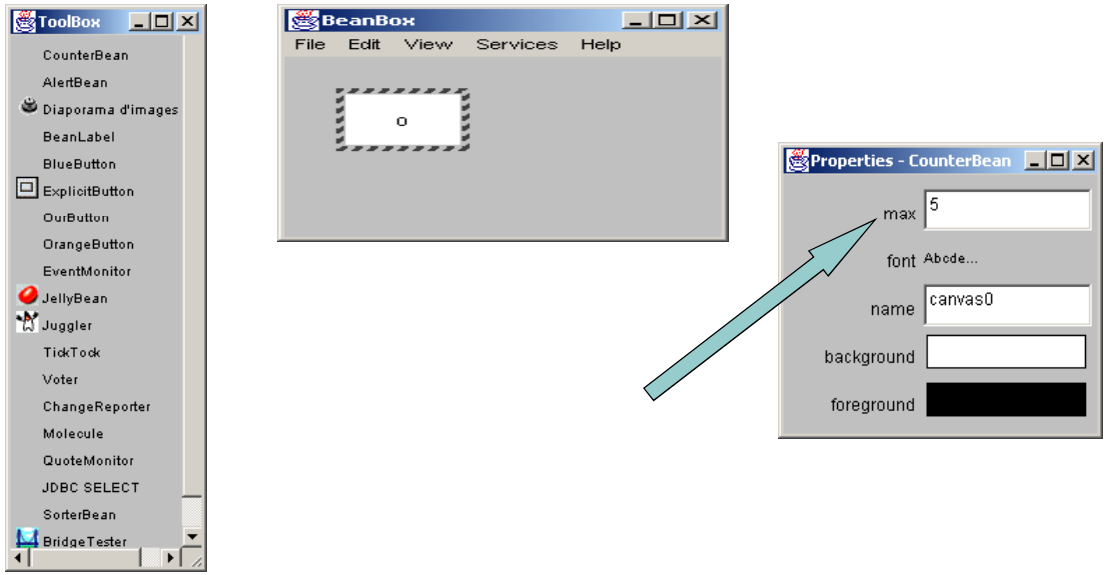
Propriété scalaire

- Une propriété scalaire représente une valeur simple d'un certain type.
- Méthodes d'accès pour une propriété P de type T:
  - *public T getP ( )* Lecture
  - *public void setP (T valeur)* Ecriture
  - Si T est boolean, la méthode de lecture peut s'écrire: *public boolean isP ( )*

Exemple CounterBean

```
private int max = 5 ;

public int getMax () { return max ; }
 public void setMax (int m) { max = m ; }
```

Exemple CounterBean



Propriété indexée

- Une propriété indexée représente un tableau de valeurs ayant le même type
- Méthodes d'accès pour une propriété P de type T:
  - *public T getP (int index)*
  - *public void setP (int index, T valeur)*
  - *public T [ ] getP ( )*
  - *public void setP (T [] valeurs)*

Propriété liée

- Une propriété liée est une propriété d'un bean dont le changement de valeur est notifié par événement à des beans abonnés.
- L'événement émis est *PropertyChangeEvent*
- Un bean supportant des propriétés liées doit implanter les méthodes d'abonnement et de désabonnement:
  - *void addPropertyChangeListener (PropertyChangeListener l)*
  - *void removePropertyChangeListener (PropertyChangeListener l)*
- Les beans notifiés doivent implanter l'interface suivante:

```
public interface PropertyChangeListener extends java.util.EventListener
{
        public void propertyChange (PropertyChangeEvent e)
}
```

Notification des changements

- Lors du changement d'une propriété, les méthodes *propertyChange* des beans abonnés est invoquée avec en argument un objet *PropertyChangeEvent*.
- Cet événement encapsule le nom de la propriété, l'ancienne et la nouvelle valeur.

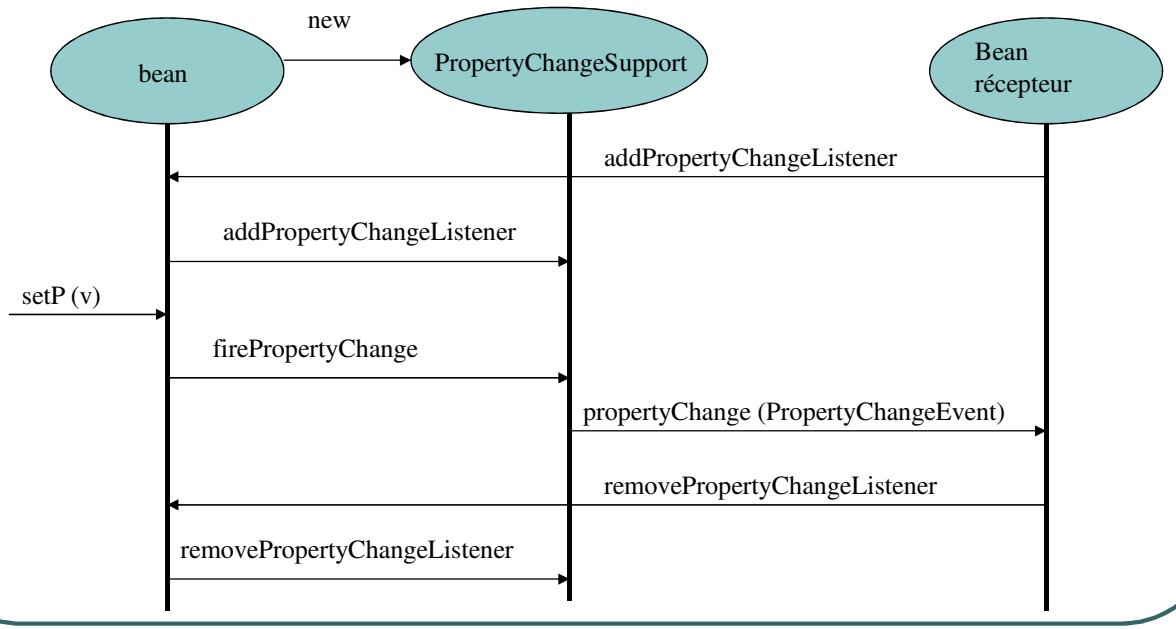
```
public class PropertyChangeEvent extends EventObject
{
        public PropertyChangeEvent(Object source, String propertyName,
                                   Object oldValue, Object newValue)
        public String getPropertyName();
        public Object getNewValue();
        public Object getOldValue();
}
```

Gestion de la notification

- Au niveau du bean émetteur, la notification peut être déléguée à une instance de la classe *PropertyChangeSupport.*
- Aspects gérés:
  - Méthodes d'abonnement/désabonnement
  - Création de l'événement et invocation des beans abonnés.
- Aperçu de la classe:

```
public class PropertyChangeSupport extends Object implementsSerializable
{
   public PropertyChangeSupport(Object sourceBean) ;
   public synchronized voidaddPropertyChangeListener(PropertyChangeListenerl);
   public synchronized voidremovePropertyChangeListener(PropertyChangeListener 1);
  public void firePropertyChange(String propertyName, Object oldValuObject newValue);
}
```

Interactions



CounterBean

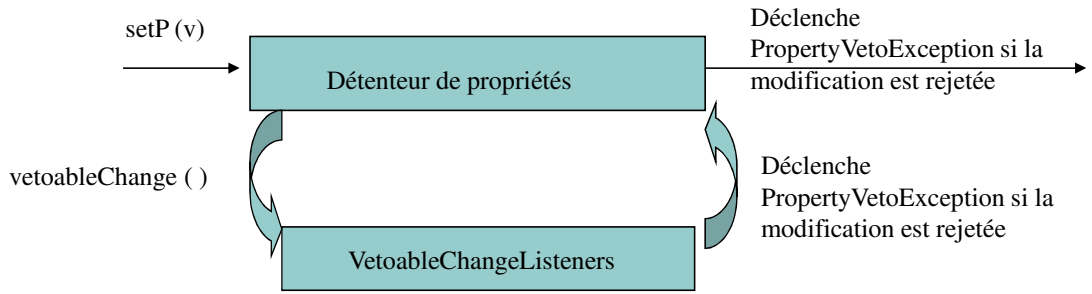
- La propriété *Max* devient une propriété liée.
- Si *Max* change, le compteur (variable value) revient à 0.
- Dans cet exemple, le bean *CounterBean* est à la fois émetteur et récepteur.

CounterBean

```
public class CounterBean extends Canvas implements Serializable,PropertyChangeListener

 private PropertyChangeSupport changes ;

 changes = new PropertyChangeSupport (this) ;  // dans constructeur

 public void setMax (int m)
 {
        changes.firePropertyChange("Changement de Max",max,m);
        max = m ;
 }

 public void propertyChange (PropertyChangeEvent evt)
 {
       value = 0 ;
       repaint () ;
 }
```

Propriété contrainte

- Une propriété contrainte est une propriété d'un bean dont le changement de valeur est soumis à d'autres beans abonnés ayant un droit de véto.
- Le propriétaire doit déclencher un *VetoableChangeEvent* avant de modifier la propriété. Si aucun bean ne la rejette, la propriété peut être changée.



Propriété contrainte

- Un bean supportant les propriétés contraintes doit implanter les méthodes d'abonnement/désabonnement:
  - *void addVetoableChangeListener (VetoableChangeListener l)*
  - *void removeVetoableChangeListener (VetoableChangeListener l)*
- Les beans notifiés doivent implémenter l'interface PropertyChangeListener et l'interface:

```
public interface VetoableChangeListener extendsEventListener
{
 public void vetoableChange(PropertyChangeEvent ev) throws PropertyVetoException
}
```

Demande de changement

- Avant le changement de valeur, la méthode *vetoableChange* des beans abonnés est invoquée avec un événement de type *PropertyChangeEvent.*
- Si l'un des beans lève une exception de type *PropertyVetoException*, la propriété n'est pas changée.
- Classe PropertyVetoException:

```
public class PropertyVetoException extends Exception
 {
    public PropertyVetoException(String msg,PropertyChangeEvent e);
   public PropertyChangeEvent getPropertyChangeEvent();
}
```

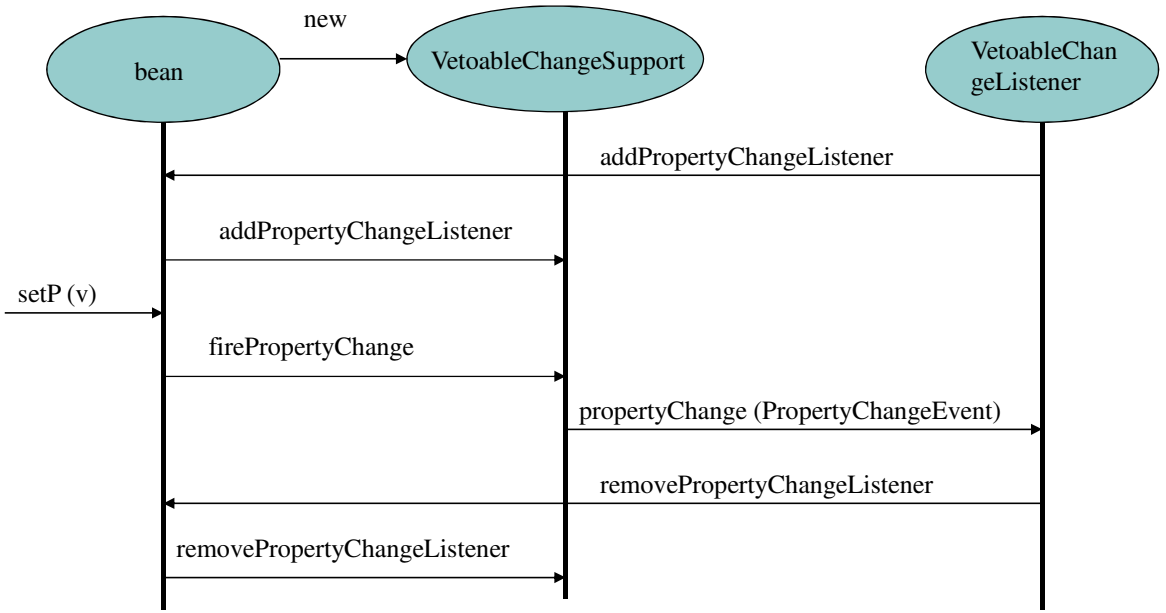
- Dans le cas contraire, la valeur de la propriété est changée et la méthode *propertyChange* des beans abonnés est invoquée.

Implémentation de la demande

- Le bean détenant la propriété contrainte peut déléguer à une instance de *VetoableChangeSupport*.
- Aspects gérés:
  - Méthodes d'abonnement/désabonnement.
  - Création de la classe et invocations des beans abonnés.
- Aperçu de la classe:

```
public class VetoableChangeSupport extends Object implementsSerializable
{
  public VetoableChangeSupport(Object sourceBean) ;
  public synchronized voidaddVetoableChangeListener(PropertyChangeListenerl);
   public synchronized voidremoveVetoableChangeListener(PropertyChangeListener 1);
   public void fireVetoableChange(String propertyName, ObjectoldValue,
                                  Object newValue) throws PropertyVetoException;
}
```

Aperçu des interactions



Introspection

- Un bean expose ses propriétés, méthodes et événements.
- Une instance de la classe *java.beans.Introspector* va inspecter le bean sur 2 niveaux:
  - Recherche d'une classe nommée *<classe_bean>BeanInfo* (par exemple, *CounterBeanBeanInfo*)
  - Sinon, le mécanisme de la réflexion de Java va être utilisé pour obtenir la liste des méthodes du bean.

L'interface BeanInfo

- La classe *<classe_bean>BeanInfo* doit implémenter l'interface *BeanInfo*.
- Cette classe ne sert qu'à décrire la classe *<classe_bean>*.
- Les méthodes de *BeanInfo*

Méthode	Description
getAdditionalBeanInfo ()	Retourne tous les objets concernant le bean associé
getBeanDescriptor ()	Retourne l'objet descripteur du bean
getDefaultEventIndex ()	Retourne l'index des événements par défaut
getDefaultPropertyIndex ()	Retourne l'index des propriétés par défaut
getEventSetDescriptors ()	Retourne les descripteurs de l'ensemble des événements
getIcon ()	Retourne l'icône spécifiée pour le bean
getMethodDescriptors ()	Retourne les descripteurs de méthodes
getPropertyDescriptors ()	Retourne les descripteurs de propriétés

Les servlets

Les servlets Java

- **Programmes java s'exécutant sur le serveur www et non pas sur la machine cliente.**
- **Nécessite un serveur www incluant un moteur de servlets (exemple: apache + tomcat).**
- **Une servlet est chargée lorsque le serveur est mis en route ou lorsque le premier client fait appel aux services de la servlet. Une fois chargée, une servlet reste active dans l'attente de nouvelles requêtes.**
- **Une servlet va pouvoir utiliser des ressources du serveur (base de données, ...) et renvoyer une page HTML au navigateur.**

Les servlets java

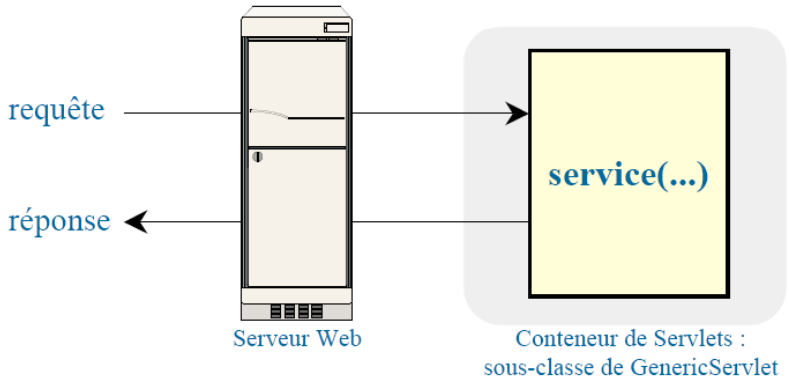
- Les servlets utilisent des classes et interfaces issues des packages *javax.servlet* (servlets indépendantes d'un protocole) et *javax.servlet.http* (servlets spécifiques au protocole http).
- Une servlet doit soit implémenter l'interface *javax.servlet.Servlet* ou étendre soit la classe *javax.servlet.GenericServlet* soit *javax.servlet.http.HttpServlet.*
- Une servlet n'a ni de méthode main () ni constructeur.
- Les initialisations peuvent se faire dans une des méthodes init () héritées de *javax.servlet.GenericServlet*:

  public void init () throws ServletException
  public void init (ServletConfig) throws ServletException

- La méthode *destroy ()* permet de libérer les ressources acquises et éventuellement d'écrire des informations persistantes qui pourront être lues au prochain chargement de la servlet par l'une des méthodes *init ()*.

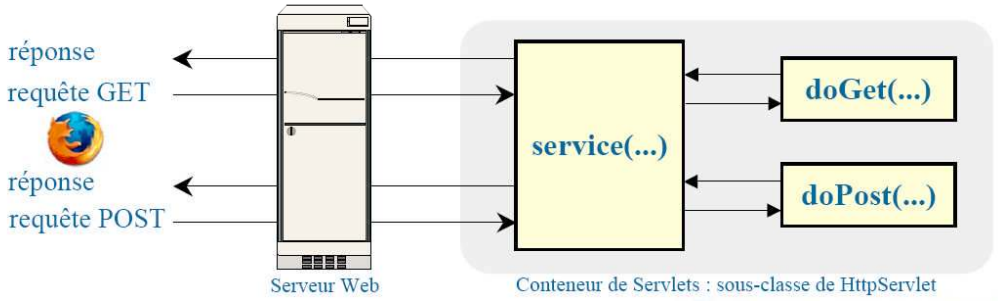
Servlet étendant la classe *javax.servlet.GenericServlet*

- **Une servlet générique doit surcharger une méthode *service ()*, méthode abstraite de la classe *javax.servlet.GenericServlet*.**



Servlet étendant la classe *javax.servlet.http.HttpServlet*

- **Une servlet http doit surcharger une méthode *doGet()* ou *doPost()* en fonction du type de requête qu'elle aura à traiter.**



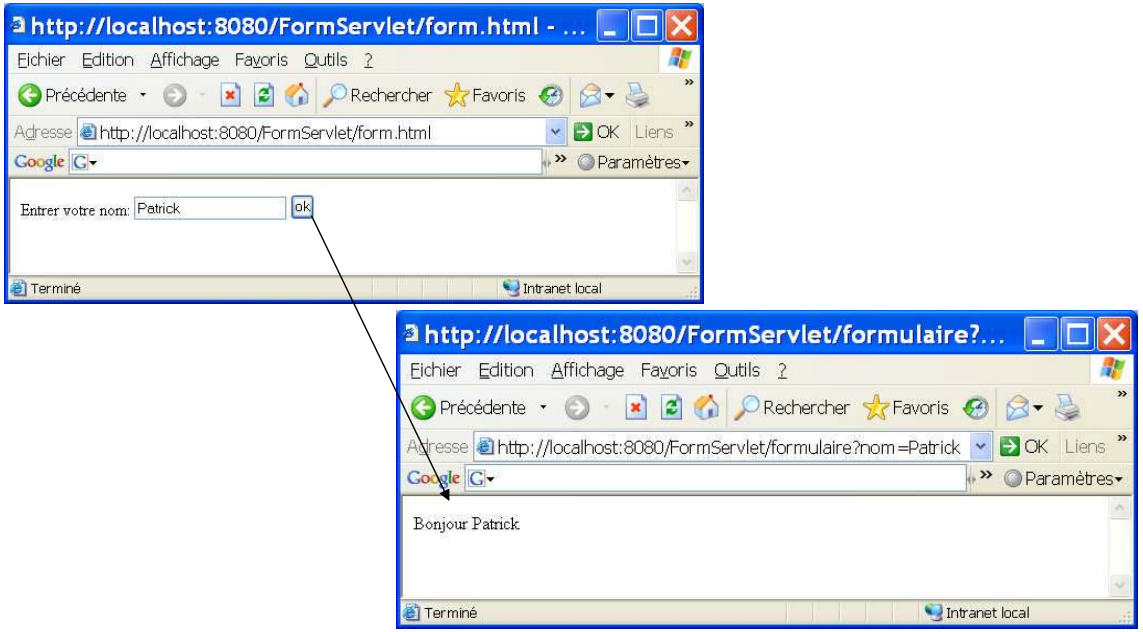
Exemple de servlet: formulaire HTML

```
<html>
<head>
<title>
ExempleServlet
</title>
</head>
<body>

<form action="http://localhost:8080/FormServlet/formulaire">
Entrer votre nom:
<input type=text name=nom>
<input type=submit value="ok">
</form>

</body>
</html>
```

Exemple de servlet: le code JAVA

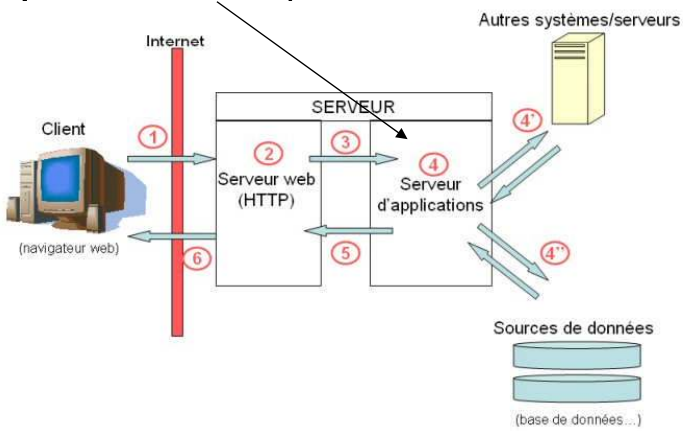
```
package exempleservlet;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class ExempleServlet extends HttpServlet
{
  private static final String CONTENT_TYPE = "text/html";

  public void init(ServletConfig config) throws ServletException { super.init(config); }

  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
       ServletException, IOException
  {
    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();
    out.println("<p>Bonjour " + request.getParameter("nom") + ".</p>");
}
}
```

Exécution de la servlet "ExempleServlet"



Serveur d'application

- Une servlet est une application web devant être déployée sur un serveur d'application (exemple: Tomcat).

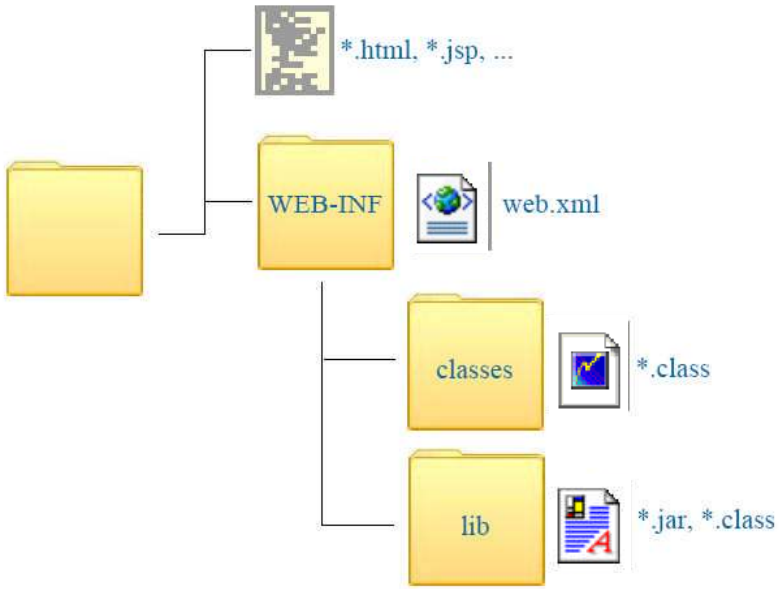


Application web

- Quand une application web est finalisée, on la place dans un fichier d'archive web (extension war).

- Un fichier d'archive est créé avec la commande jar et possède une architecture précise.

Architecture d'un fichier war



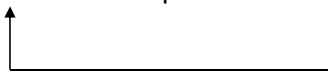
web.xml: le fichier de déploiement

- Le descripteur de déploiement contient toutes les informations de configuration du fichier archive.

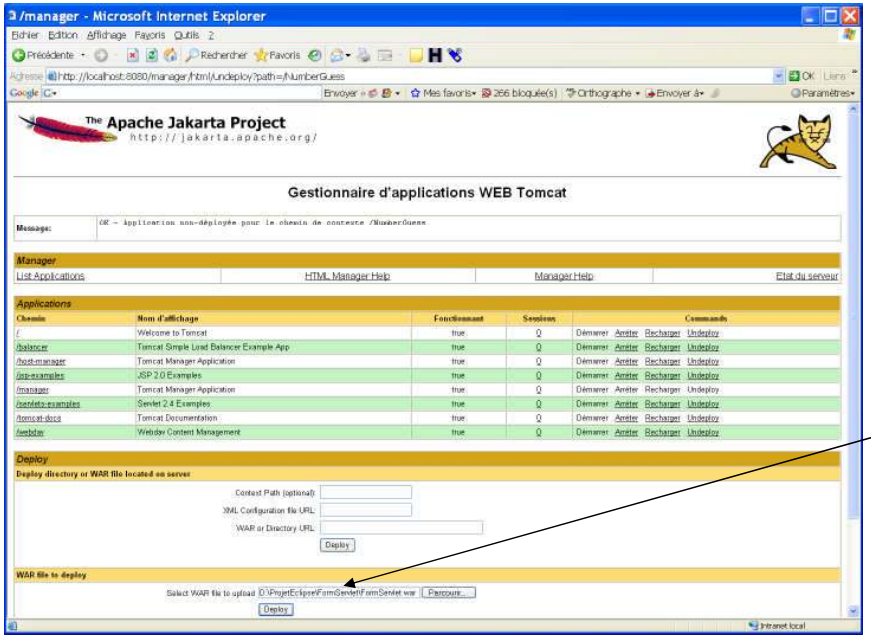
- Dans le cas des servlets, il va permettre de définir la classe contenant la servlet, le nom de la servlet , les paramètres d'initialisation, le chemin virtuel d'accès, …

Exemple web.xml

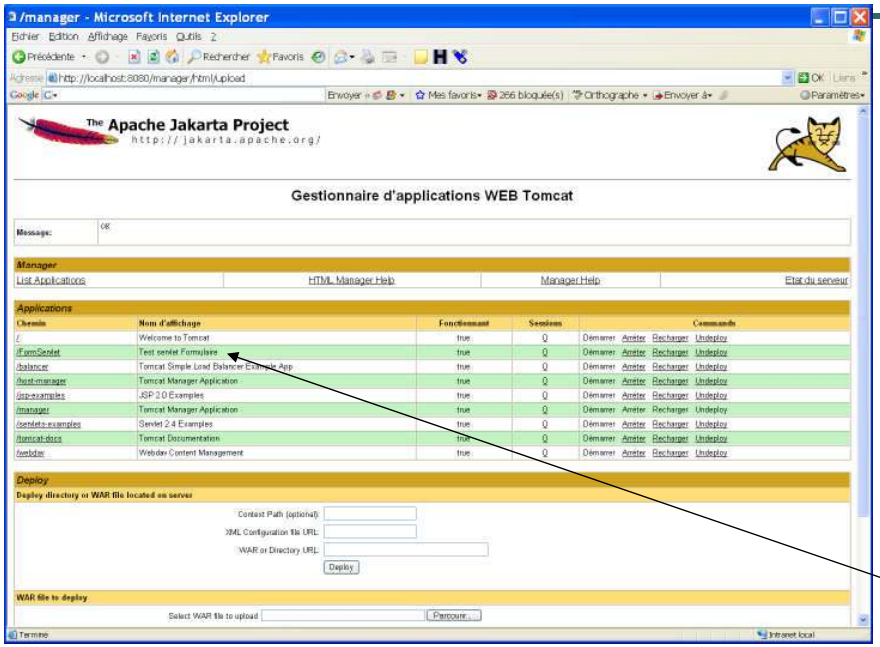
```
<web-app>

    <display-name> Test servlet Formulaire  </display-name>
    <description> A web app  </description>

    <servlet>
        <servlet-name> FormServlet </servlet-name>
        <servlet-class> formservlet.FormServlet </servlet-class>
        </servlet>

    <servlet-mapping>
        <servlet-name> FormServlet </servlet-name>
        <url-pattern> /formulaire  </url-pattern>
    </servlet-mapping>

</web-app>
```



Déploiement de l'archive avec tomcat



Déploiement de l'archive avec tomcat



Quelques balises pour web.xml

- Paramètres d'initialisation consultable par la méthode:

```
getInitParameter ("nom_parametre") ;
<init-param>
        <param-name>nom_parametre</param-name>
        <param-value>valeur_parametre</param-value>
        <description> description du parametre </description>
</init-param>
```

- Gestion des erreurs:

```
<error-page>
        <error-code> 404 </error-code>
        <location> /errors/404.html </location>
</error-page>
<error-page>
        <exception-type>javax.servler.ServletException</exception-type>
        <location> /errors/exception.jsp </location>
</error-page>
```

Suivi de session

- Le protocole http est sans état.
- Le suivi de session peut être simulé:
  - cookies (classe *javax.servlet.http.Cookie*)
  - utilisation de la classe j*avax.servlet.http.HttpSession* qui va permettre de stocker des objets plutôt que des chaînes de caractères comme les cookies.
- La durée de session peut être définie dans le fichier de déploiement (valeur en minute):

```
<session-config>
        <session-timout> 10  </session-timout>
</session-config>
```

ou par appel de la méthode de *HttpSession setMaxInactiveInterval (int time)* définissant le temps maximal en secondes entre deux requêtes avant que la session n'expire.

Cookies

- Quelques méthodes de *javax.servlet.http.Cookie*:
  Cookie (String name, String Value)
  String getName ()
  String getValue ()
  setValue (String value)
  setMaxAge (int expiry)

- Création d'un cookie: méthode de HttpServletResponse
  response.addCookie (Cookie cookie)

- Récupération de cookies: méthode de HttpServletRequest
  Cookie [] getCookies ()

La classe javax.servlet.http.HttpSession

- Création, méthodes de HttpServletRequest:
  HttpSession getSession ()
  HttpSession getSession (boolean p)

- Destruction, méthode de HttpSession:
  invalidate ()

- Gestion de la session, méthodes de HttpSession:
  Enumeration getAttributNames ()
  Object getAttribut (String name)
  setAttribut (String name, Object value)
  removeAttribut (String name)

Exemple HttpSession

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
        response.setContentType(CONTENT_TYPE);
        HttpSession session = request.getSession() ;
        Integer count = (Integer) session.getAttribute("count") ;
        if (count == null) count = new Integer (1) ;
        else count = new Integer (count.intValue() + 1) ;

        if (count.intValue () == 5) session.invalidate() ;
        else session.setAttribute("count", count) ;

        PrintWriter out = response.getWriter();
        out.println (chaine + "  " + count) ;
}
```

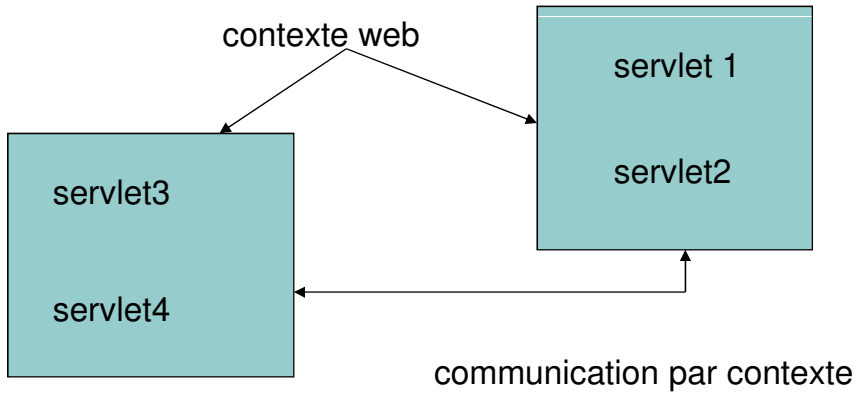
Collaboration entre servlets

- Des servlets s'exécutant sur le même serveur web peuvent collaborer:
  - Par partage d'informations
  - Par partage du contrôle (une servlet peut recevoir une requête et laisser une autre servlet la traiter).

Partage d'informations entre servlets

- Les servlets peuvent partager de l'information:
  - à travers un conteneur externes (base de données).
  - à travers l'utilisation des contextes:

    ServletContext getContext (String uri)



Exemple de communication: servlet1

```
package servletscommunication;

import javax.servlet.* ;
import javax.servlet.http.*;
import java.io.* ;

public class Servlet1 extends HttpServlet
{

        public void doGet (HttpServletRequest request, HttpServletResponse response)
        {
                PrintWriter out = null ;
                response.setContentType("text/plain") ;
                try {
                                                out = response.getWriter () ;
                } catch (IOException e) {}

           ServletContext contexte = this.getServletContext() ;
           contexte.setAttribute ("chaine1", "chaine deposee par servlet1") ;
           out.println ("la chaine est deposee") ;
        }
}
```

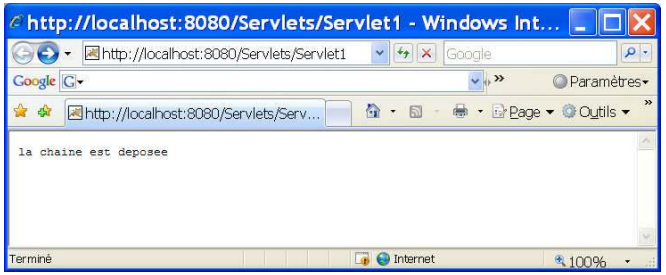
Exemple de communication: servlet2

```
package servletscommunication;

import javax.servlet.* ;
import javax.servlet.http.*;
import java.io.* ;

public class Servlet2 extends HttpServlet
{
	public void doGet (HttpServletRequest request, HttpServletResponse response)
	{
		PrintWriter out = null ;
		response.setContentType("text/plain") ;
		try { out = response.getWriter () ; } catch (IOException e) {}
		ServletContext moncontexte = this.getServletContext() ;
		ServletContext servlet1 = moncontexte.getContext("/Servlets/servlet1") ;
		if (servlet1 == null) out.println ("Pas de contexte trouve") ;
		else
		{
			String chaine = (String) servlet1.getAttribute ("chaine1") ;
			if (chaine == null) out.println ("Aucune chaine trouvee") ;
			else out.println ("chaine trouvee:" + chaine) ;
		}
	}
}
```

Exemple de communication: web.xml

```
<web-app>

        <display-name> Collaboration servlets  </display-name>
        <description> Collaboration de servlets  </description>

        <servlet>
                <servlet-name> Servlet1 </servlet-name>
                <servlet-class> servletscommunication.Servlet1 </servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name> Servlet1 </servlet-name>
                <url-pattern> /Servlet1  </url-pattern>
        </servlet-mapping>

        <servlet>
                <servlet-name> Servlet2 </servlet-name>
                <servlet-class> servletscommunication.Servlet2 </servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name> Servlet2 </servlet-name>
                <url-pattern> /Servlet2  </url-pattern>
        </servlet-mapping>

</web-app>
```

Exemple de communication: résultat



Collaboration de servlets par partage du contrôle

- Les servlets peuvent partager ou distribuer le contrôle d'une requête grâce à l'interface javax.servlet.*RequestDispatcher*
  - par renvoi: une servlet peut renvoyer une requête entière sur une servlet, page jsp ou html par la méthode:
    void forward (ServletRequest req, ServletResponse res)
  - par inclusion: une servlet peut inclure du contenu généré.

Exemple de distribution de renvoi

```
package servletscollaboration;

import javax.servlet.* ;
import javax.servlet.http.*;

public class ServletMain extends HttpServlet
{
	public void doGet (HttpServletRequest request, HttpServletResponse response)
	{
		response.setContentType("text/plain") ;
		request.setAttribute("chaine1","argument1") ;  // on transmet un objet
		try {
			RequestDispatcher dispat = // on transmet une chaine
				request.getRequestDispatcher("/process.jsp?chaine2=argument2") ;
			dispat.forward(request, response) ;
		} catch (Exception e) {}
	}
}
```

Exemple de distribution de renvoi

```
<HTML>
<%= request.getAttribute ("chaine1") %><br>
<%= request.getParameter ("chaine2") %><br>
</HTML>
```

fichier: process.jsp

Résultat

Exemple d'inclusion

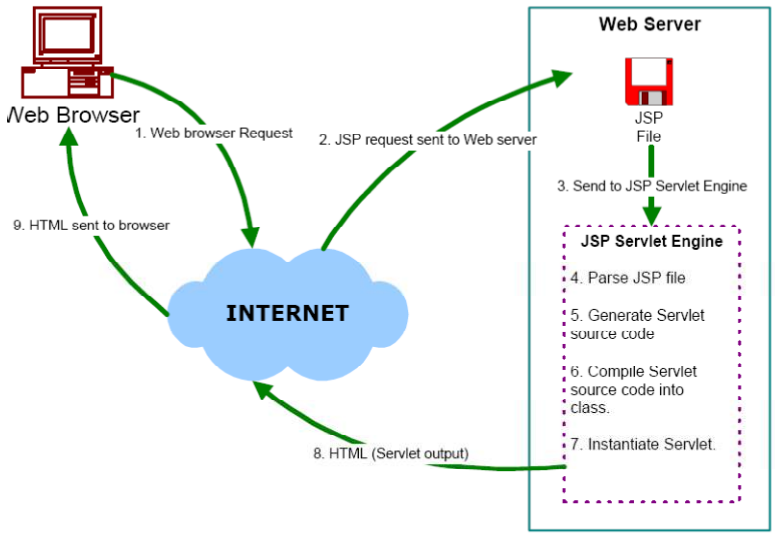

```
package servletscollaboration;

import javax.servlet.* ;
import javax.servlet.http.*;
import java.io.* ;

public class ServletInclude extends HttpServlet
{
	public void doGet (HttpServletRequest request, HttpServletResponse response)
	{
		PrintWriter out = null ;
		response.setContentType("text/plain") ;
		request.setAttribute("chaine1","argument1") ;  // on transmet un objet
		try {
			out = response.getWriter() ;
			RequestDispatcher dispat = // on transmet une chaine
				request.getRequestDispatcher("/process.jsp?chaine2=argument2") ;
			dispat.include(request, response) ;
		} catch (Exception e) {}
		out.println ("inclusion effectuee") ;
	}
}
```

Java Server Pages

Avantages JSP

- Les JSP permet de créer des pages webs dynamiques en mélangeant:
  - du code HTML.
  - des balises JSP spéciales.
  - du code java (scriptlet) directement dans la page.
- Les JSP sont multi plate formes (Write Once, Run Anywhere).
- Les parties traitements statiques (accès à des bases de données, …) peuvent être déportées dans des java beans.
- Les JSP permettent de retourner le code HTML aux navigateurs de manière plus élégante que les servlets.
- Une balise JSP est une balise XML associée à une classe java.

Principe JSP



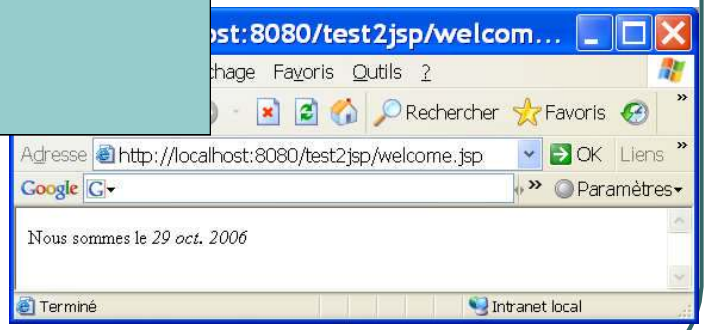
Exemple code JSP

```
<%@page import="java.util.*,java.text.*" %>

<HTML>
<BODY>
<%
Date d = new Date () ;
String today = DateFormat.getDateInstance().format (d) ;
%>
Nous sommes le
<em> <%= today %> </em>
</BODY>
</HTML>
```

conteneur de servlet

compilation

servlet



Commentaires

- Syntaxe: <%-- ……. --%>

- Les commentaires JSP ne seront pas visibles par l'option "affichage source" des navigateurs, contrairement aux commentaires HTML: <!-- … ->

Balise de déclaration

- Syntaxe: <%! ... %>
- Cette balise permet de déclarer des variables et des méthodes.
- Exemple:

```
<%!
        private int counter = 0 ;
        private String getAccount (int accountNo) ;
 %>
```

Balise d'expression

- Syntaxe: <%= … %>
- Cette balise d'évaluer et d'afficher la valeur d'une expression (appel simplifié de *out.println ()*)
- Exemple:

La date du jour est <%= new java.util.Date () %>

Balise de directive

- Syntaxe: <%@directive ... %>
- La balise de directive donne des informations concernant la page jsp au moteur de jsp.
- Trois directives possibles:
  - include — fichier à inclure
  - page — information concernant cette page
  - tag library — définition de balises personnalisées

La directive include

- Cette directive permet d'inclure un fichier.

- Exemples:

<%@include file="exemple.html %>

<%@include file="menu.jsp" %>

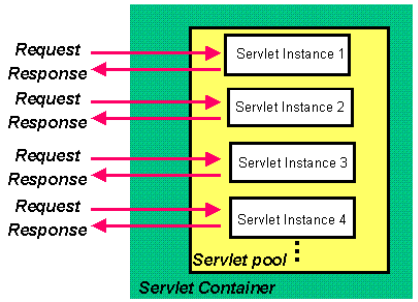
La directive "page"

language	langage utilisé: <%@page language="java" %>
extends	super classe utilisée pour la servlet générée: <%@page extends="com.taglib..." %>
import	importation d'un package. <%@page import java.util.*; %>
session	Par défaut, toutes les données sont disponible le temps de la session. Cette valeur peut être mise à "false" pour des raisons de performances.
buffer	Définit la taille du cache en sortie (8kb par défaut).
autoFlush	Vide le cache de sortie lorsqu'il est plein.
isThreadSafe	Si active, un thread sera créé pour gérer la requête autorisant ainsi la servlet générée de traiter de multiples requêtes simultanément (voir transparent suivant).
info	Permet de mettre des informations sur la page (auteur, copyright, date, …).
errorPage	Indique l'url d'une page à afficher en cas d'exception non traitée (voir exemple)
isErrorPage	Si true, la page pourra accéder à l'objet implicite "exception" (voir exemple)
contentType	Indique le type mime et le jeu de caractères.

Les problèmes de synchronisation

- Les servlets générées sont multi threads par défaut; cela peut engendrer des conflits d'accès à des données partagées.

- Solution 1: *<%@page isThreadSafe="false" %>*



- Solution 2:

```
<% synchronized (application) {
        SharedObject foo = (SharedObject) application.getAttribute("sharedObject");
        foo.update(someValue);
        application.setAttribute("sharedObject",foo);
          } %>
```

Gestion des exceptions

- Il est important d'intercepter les exceptions non traitées, notamment les "run time exceptions":

```
<%@page isErrorPage="false" errorPage="exception.jsp" %>
```

- Pour accéder à l'objet implicite "exception", la page exception.jsp doit implémenter la balise:

```
<%@ page isErrorPage="true" %>
```

Exemple de gestion d'exceptions

- Fichier jsp:

```
<%@page isErrorPage="false" errorPage="error.jsp" %>
<%= java.lang.Integer.parseInt ("2.34")  %>
```

- Fichier error.jsp:

```
<%@page isErrorPage = "true" %>
<H1> Une erreur fatale est intervenue: <%= exception.getMessage () %> </H1>
```

- Résultat:

Scriplets

- Syntaxe: <% …… %>
- Exemple:

```
<% for (int i = 1 ; i < 4 ; i++) { %>
<H<%= i %>>Bonjour </H<%= i %>>
<% } %>
```

Balise de redirection

<jsp:forward page=« nom_page" />

- Exemple:

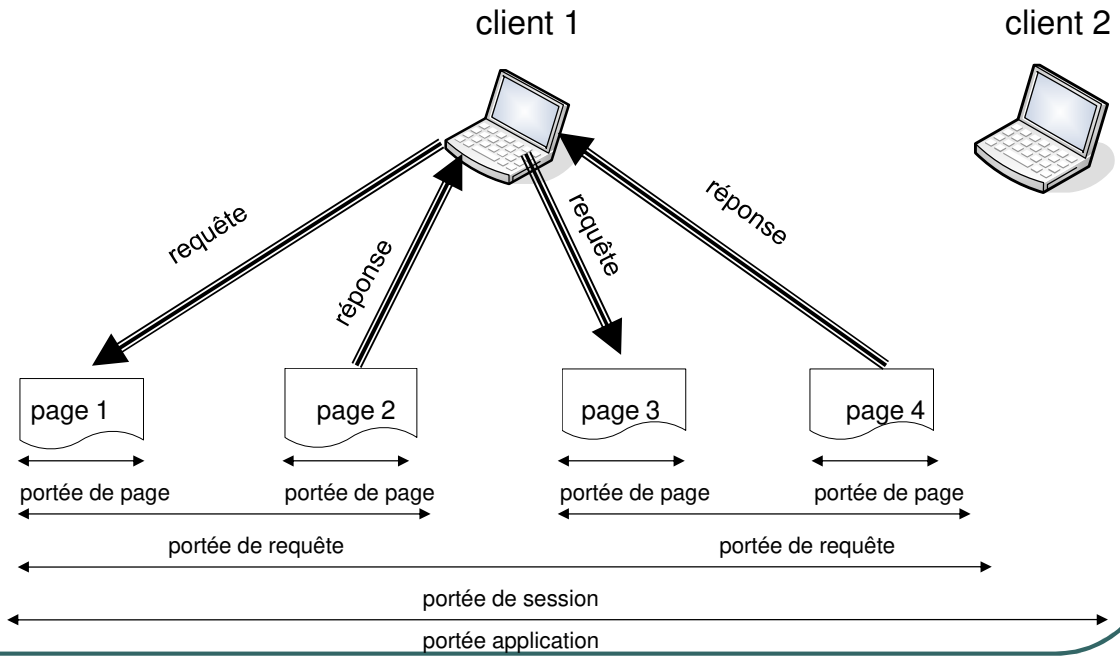
<jsp:forward page="retry.jsp" />

Portées des objets

- Les objets créés au sein d'une page JSP peuvent avoir différentes portées:

application	Objets accessibles aux pages appartenant à la même application
session	Objets accessibles aux pages appartenant à la même session que les pages où ils ont été créés.
request	Objets accessibles uniquement dans les pages exécutant la requête qui les ont créées.
page	objets accessibles uniquement dans la page où ils ont été créés.

Portée des objets

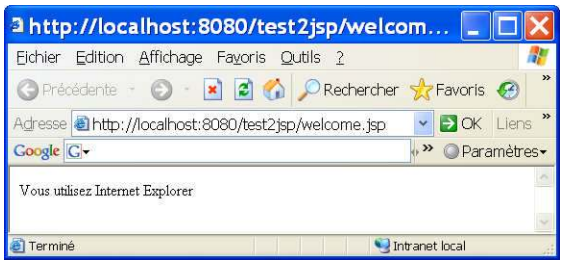


Les objets JSP implicites

Nom objet	classe java	Portée
request	HttpServletRequest	request
response	HttpServletResponse	page
pageContext	PageContext	page
application	ServletContext	application
out	JspWriter	page
config	ServletConfig	page
page	HttpJspPage	page
session	HttpSession	page
exception	Throwable	page

Exemple d'objet implicite

```
<HTML>
<BODY>

<% if(request.getHeader ("User-Agent").indexOf ("MSIE") != -1) { %>
Vous utilisez Internet Explorer
<% } else if(request.getHeader ("User-Agent").indexOf ("Mozilla") != -1) {
%>
Vous utilisez Netscape
<% } else { %>
Le navigateur m'est inconnu
<% } %>

</BODY>
</HTML>
```



JSP et les JavaBeans

- Rappel: un javabean est une classe java:
  - sérialisable
  - disposant de méthodes get/set pour accéder à des propriétés.

- Exemple:

```
package testpackage ;
public class InfoBean implements java.io.Serializable
{
    private String nom ;
    public InfoBean ()   {  nom = null ; }
    public InfoBean (String n) { nom = n  ; }
    public String getNom() { return nom; }
    public void setNom(String nom) { this.nom = nom; }
}
```

La balise <jsp:useBean>

- Association d'une instance de bean avec un identificateur et définition de la portée:

<jsp:useBean id="beanName" scope="page|request|session|application" typespec>
body
</jsp:useBean>

La balise <jsp:setProperty>

- <jsp:setProperty name="beanName" prop_expr />

  où prop_expr vaut une des valeurs:
  - property="*"
  - property="propertyName"
  - property="propertyName" param="parameterName"
  - property="propertyName" value="propertyValue"

- <jsp:getProperty name="beanName" property="propertyName" />

Exemple

```
<HTML>
<FORM ACTION="form.jsp">
<INPUT TYPE="Text" NAME="nom" />
<INPUT TYPE="Submit" />
</FORM>
</HTML>
```

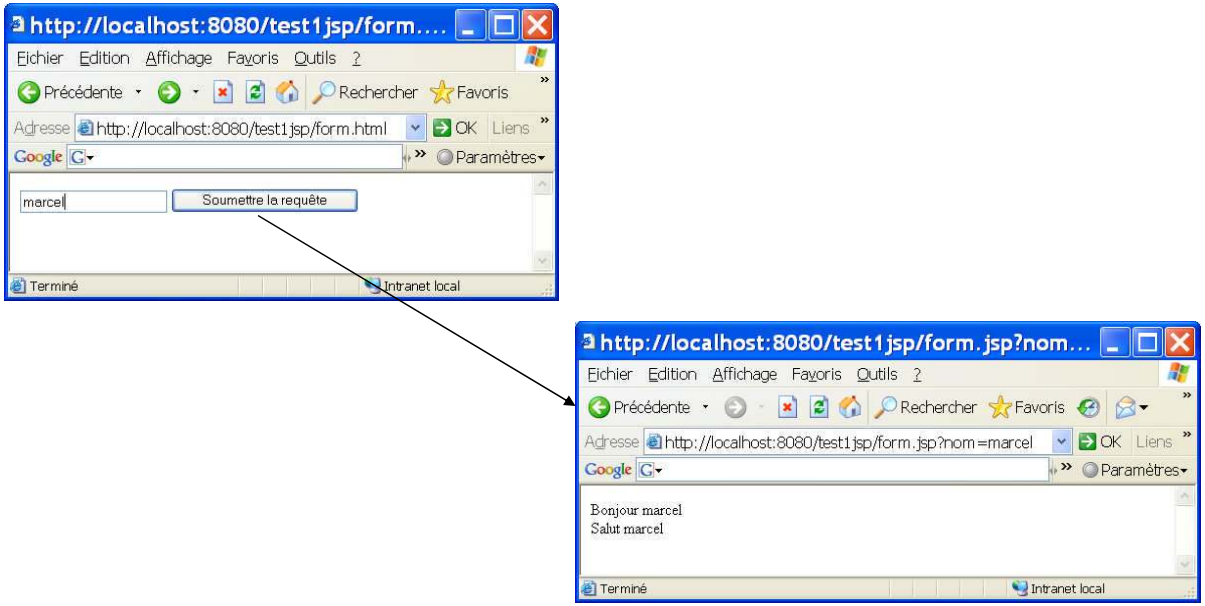
form.html

```
<HTML>
<jsp:useBean id="InfoBeanId" scope="request" class="testpackage.InfoBean">
<jsp:setProperty name="InfoBeanId" property="*"/>
</jsp:useBean>
Bonjour <%= InfoBeanId.getNom () %>
<br>
Salut <jsp:getProperty name="InfoBeanId" property="nom"/>
</HTML>
```

Identification du champ nom du formulaire avec la propriété nom de InfoBean

form.jsp

Exécution de l'exemple



Remarque sur la syntaxe

```
<jsp:useBean id="InfoBeanId" scope="session" class="testpackage.InfoBean">
<jsp:setProperty name="InfoBeanId" property="*"/>
</jsp:useBean>
```

Le setProperty sera exécuté uniquement au moment de l'instanciation du bean (dans cet exemple une fois au cours de la session compte tenu de la valeur de scope).

```
<jsp:useBean id="InfoBeanId" scope="session" class="testpackage.InfoBean" />
<jsp:setProperty name="InfoBeanId" property="*"/>
```

Le setProperty est exécuté ici inconditionnellement.

Java Native Interface

Pourquoi JNI

- Certaines fonctionnalités peuvent être inaccessibles à JAVA:
  - Les bibliothèques java ne supporte pas les fonctionnalités dépendantes de la plate forme et requises par l'application (adressage physique, accès au matériel, interruption, …).
  - Développer en C/C++ tout en bénéficiant de l'IHM java.
  - Rendre accessible au code java une bibliothèque existante dans un autre langage.
  - Utiliser du code natif pour accélérer le temps d'exécution.
- Quelques conséquences:
  - La portabilité est annulée.
  - La sécurité et la robustesse deviennent moindres.

Présentation de JNI

- Il est possible:
  - D'appeler des fonctions C/C++ depuis Java.
  - D'accéder à des objets Java depuis le C/C++.
- JNI permet:
  - D'écrire des méthodes natives
  - D'appeler des méthodes java
  - De capturer et de déclencher des exceptions
  - De charger des classes
  - …

De Java vers C/C++

- Méthodologie:
  - Utilisation du mot clé *native*
  - Génération d'un fichier d'entête .h (avec javah)
  - Ecrire du code natif et génération d'une bibliothèque (.dll, .so)

Exemple: Java vers C/C++

```
public class JavaVersC
{
	public native void bonjour () ;

	public static void main (String args [])
	{
		new JavaVersC ().bonjour () ;
	}

	static
	{
		try {
			System.loadLibrary ("JavaVersC") ;
		    } catch (UnsatisfiedLinkError e)
		    {
			System.err.println ("Erreur bibliotheque " + e) ;
			System.exit (1) ;
		    }
	}
}
```

JavaVersC.java

Exemple: Java vers C/C++

- javah -jni JavaVersC
- Génération fichier JavaVersC.h:

```
/* Header for class JavaVersC */

#ifndef _Included_JavaVersC
#define _Included_JavaVersC
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     JavaVersC
 * Method:    bonjour
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_JavaVersC_bonjour  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

Exemple: Java vers C/C++

```
#include <stdio.h>
#include "JavaVersC.h"

JNIEXPORT void JNICALL Java_JavaVersC_bonjour (JNIEnv *env,
    jobject j)
{
      printf ("Bonjour tout le monde") ;
}
```

bonjour.c

- **JNIEnv *env**
  Ce pointeur est un point d'entrée dans le thread courant au sein de la machine Java

- **jobject j**
  Référence sur l'objet qui a appelé la méthode native.

Exemple: Java vers C/C++

- Compilation sous Solaris:

javac JavaVersC.java

gcc -c bonjour.c -I /usr/java1.2/include -I /usr/java1.2/include/solaris -o libJavaVersC.so

- Compilation sous Linux

javac JavaVersC.java

gcc -shared bonjour.c -I /usr/local/java/include -I /usr/local/java/include/linux -o libJavaVersC.so

Exemple: Java vers C/C++

- Compilation sous Windows (Visual C++ 6.0)

```
@ECHO OFF
SET DEVSTUDIO=c:\Program Files\Microsoft Visual Studio\VC98
SET JDK13=c:\JDK1.3
@ECHO ON
%JDK13%\bin\javac JavaVersC.java
%JDK13%\bin\javah -classpath . JavaVersC
@ECHO OFF
SET COMPILE_CMD="%DEVSTUDIO%\bin\cl"
SET COMPILE_CMD=%COMPILE_CMD% bonjour.c
SET COMPILE_CMD=%COMPILE_CMD% -I"%JDK13%\INCLUDE"
SET COMPILE_CMD=%COMPILE_CMD% -I"%JDK13%\INCLUDE\WIN32"
SET COMPILE_CMD=%COMPILE_CMD% -I"%DEVSTUDIO%\Include"
SET COMPILE_CMD=%COMPILE_CMD% -FeJavaVersC.dll
SET COMPILE_CMD=%COMPILE_CMD% -MD -LD /link
SET COMPILE_CMD=%COMPILE_CMD% /libpath:"\"%JDK13%\lib\""
SET COMPILE_CMD=%COMPILE_CMD% /libpath:"\"%DEVSTUDIO%\lib""
SET COMPILE_CMD=%COMPILE_CMD% user32.lib gdi32.lib
@ECHO ON
%COMPILE_CMD%
```

Exemple: Java vers C/C++

- Exécution:

  java –cp . -Djava.library.path=. JavaVersC

Résultat: → Bonjour tout le monde

Correspondance type de données

- JNI établit une correspondance entre les types java et les types natifs :

Type Java	Type natif	Description
boolean	jboolean	unsigned 8 bits
byte	jbyte	signed 8 bits
char	jchar	unsigned 16 bits
short	jshort	signed 16 bits
int	jint	signed 32 bits
long	jlong	signed 64 bits
float	jfloat	32 bits
double	jdouble	64 bits
void	void	---

Exemple échange de types de données

- Soit la fonction native suivante:

```
public class test
{
        public native long calcul (long valeur) ;
}
```

- Le code natif sera défini par:

```
JNIEXPORT jlong JNICALL Java_test_calcul (JNIEnv *, jobject, jlong);
```

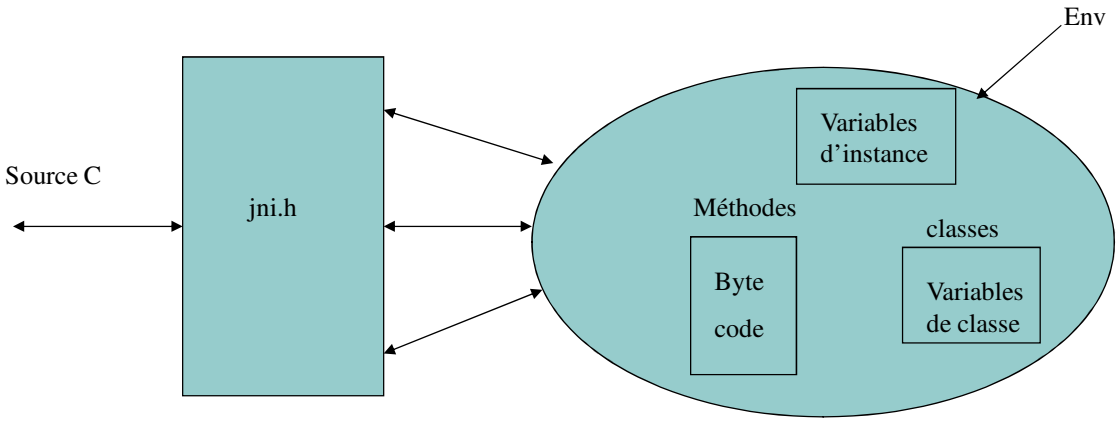
De C/C++ vers Java

- Accéder aux variables d'instance
- Accéder aux variables de classe
- Accéder aux méthodes d'instance
- Accéder aux méthodes de classe

Le pointeur JNIEnv

- *JNIEnv *env*

Ce pointeur permet d'accéder à l'environnement de la machine java associée.



Accès aux variables d'instance

- jclass GetObjectClass (JNIEnv *env, jobject obj) ;
- jfieldID GetFieldID (JNIEnv *env,jclass clazz,const char *name,const char *sig) ;
- *NativeType* Get<*type*>Field (JNIEnv *env, jobject obj, jfieldID fieldID) ;
- void Set<*type*>Field (JNIEnv *env, jobject obj, jfieldID fieldID, *NativeType* value) ;

```
class Exemple
{
   int x ;
   public native void setX (int val) ;
}


JNIEXPORT void JNICALL Java_Exemple_setX (JNIEnv *env, jobject obj,jint valeur)
{
   jclass classe = (*env)->GetObjectClass (env,obj) ;
   jfieldID fid = (*env)->GetFieldID (env,classe,"x","I") ;
   (*env)->SetIntField (env,obj,fid,valeur) ;
}
```

Accès aux variables de classe

- jfieldID GetStaticFieldID (JNIEnv *env, jclass clazz, const char *name, const char *sig) ;
- *NativeType* GetStatic<*type*>Field (JNIEnv *env, jclass clazz, jfieldID fieldID) ;
- void SetStatic<*type*>Field (JNIEnv *env, jclass clazz, jfieldID fieldID, *NativeType* value) ;

```
class Exemple
{
   static int x ;
   public native void setX (int val) ;
}

JNIEXPORT void JNICALL Java_Exemple_setX (JNIEnv *env, jobject obj,jint valeur)
{
   jclass classe = (*env)->GetObjectClass (env,obj) ;
   jfieldID fid = (*env)->GetStaticFieldID (env,classe,"x","I") ;
   (*env)->SetStaticIntField (env,obj,fid,valeur) ;
}
```

Appels de méthodes d'instance

- jmethodID GetMethodID (JNIEnv *env, jclass clazz, const char *name, const char *sig) ;
- *NativeType* Call<*type*>Method (JNIEnv *env, jobject obj, jmethodID methodId, …) ;
- *NativeType* Call<*type*>MethodA (JNIEnv *env, jobject obj, jmethodID methodId, jvalue *args) ;
- *NativeType* Call<*type*>Method (JNIEnv *env, jobject obj, jmethodID methodId, va_list args) ;

```
class Exemple
{
   int x ;
   public void afficheX () { System.out.println ("X vaut : " + x) ; }
   public native void setX (int val) ;
}

JNIEXPORT void JNICALL Java_Exemple_setX (JNIEnv *env, jobject obj,jint valeur)
{
   jclass classe = (*env)->GetObjectClass (env,obj) ;
  jfieldID fid = (*env)->GetFieldID (env,classe,"x","I") ;
   jmethodID mid = (*env)->GetMethodID (env,classe,"afficheX","()V") ;
   (*env)->SetIntField (env,obj,fid,valeur) ;

   (*env)->CallVoidMethod (env,obj,mid) ;
}
```

Appel de méthodes de classe

- jmethodID GetStaticMethodID (JNIEnv *env, jclass clazz, const char *name, const char *sig) ;
- *NativeType* CallStatic<*type*>Method (JNIEnv *env, jobject obj, jmethodID methodId, …) ;
- *NativeType* CallStatic<*type*>MethodA (JNIEnv *env, jobject obj, jmethodID methodId, jvalue *args) ;
- *NativeType* CallStatic<*type*>Method (JNIEnv *env, jobject obj, jmethodID methodId, va_list args) ;

```
class Exemple
{
public static void affiche () { System.out.println ("Affiche est une methode statique") ; }
}

JNIEXPORT void JNICALL Java_Exemple_setX (JNIEnv *env, jobject obj,jint valeur)
{
   jclass classe = (*env)->GetObjectClass (env,obj) ;
   jfieldID fid = (*env)->GetFieldID (env,classe,"x","I") ;
   jmethodID mid = (*env)->GetStaticMethodID (env,classe,"affiche","()V") ;
   (*env)->CallStaticVoidMethod (env,obj,mid) ;
}
```

Type signatures

- JNI utilise la représentation de la JVM pour les types de signature:

Type Signature	Java Type
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
L fully-qualified-class;	fully-qualified-class
[type	type [ ]
(arg-types) ret-type	method type

Type Signature

- Signature de la méthode java suivante:

  long f (int n, String s, int [] arr) ;

  (ILjava.lang.String;[I)J

- Obtention des signatures avec le désassembleur javap:

  javap –s –private JavaVersC

J2ME

Présentation J2ME

- Java 2 Micro Edition
- Plate forme destinée aux machines disposant de ressources réduites (téléphone portable, pda, …).
- La plate forme JavaCard n'est pas incluse dans J2ME.
- Compte tenu de la grande disparité des appareils mobiles, J2ME a une structure modulaire composée de configurations et de profiles.

Les configurations

- CLDC (Connected Limited Device Configuration)
  - Définie par la JSR 030
  - concerne les appareils à ressources faibles comme par exemple un téléphone portable (moins de 512 ko de RAM, processeur lent, connexion réseau limitée et intermittente, interface utilisateur limitée).
  - Nécessite une machine virtuelle de type KVM.
- CDC (Connected Device Configuration)
  - Définie par la JSR 036
  - concerne les appareils possédant des ressources plus importantes (2Mb de ram ou plus, processeur 32 bits, connexion réseau)
  - Nécessite une machine virtuelle de type CVM.

Les profiles

- Les profiles sont un ensemble d'API adapté à une configuration.

MIDP1.0	CDLC	JSR 37
Foundation Profile	CDC	JSR 46
Personal Profile	CDC	JSR 62
MIDP 2.0	CDLC	JSR 118
Personnal Basis Profile	CDC	JSR 129
RMI Optional Profile	CDC	JSR 66
Mobile Media API (MMAPI)	CLDC	JSR 135
PDA		JSR 75
JDBC Optional Profile	CDC	JSR 169
Wireless Messaging API (WMA)	CLDC	JSR 120

APIs CDLC

- 4 packages:
  - java.lang (classes de bases).
  - java.io (gestion des flux d'entrées/sorties).
  - java.util (classes utilitaires).
  - javax.microedition.io (classes de gestion de connexions).

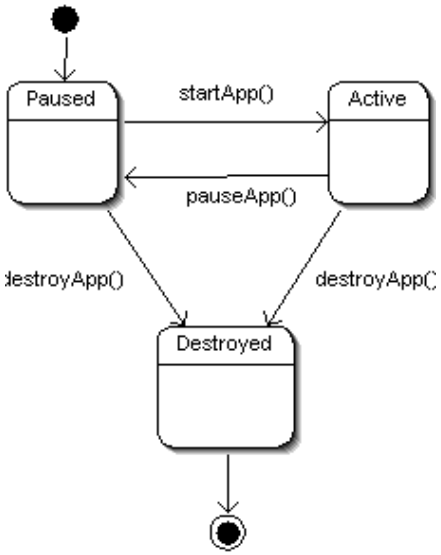
APIs CDC

- 6 packages conformes à la J2SE 1.3
  - java.lang
  - java.util
  - java.net
  - java.io
  - java.text
  - java.security

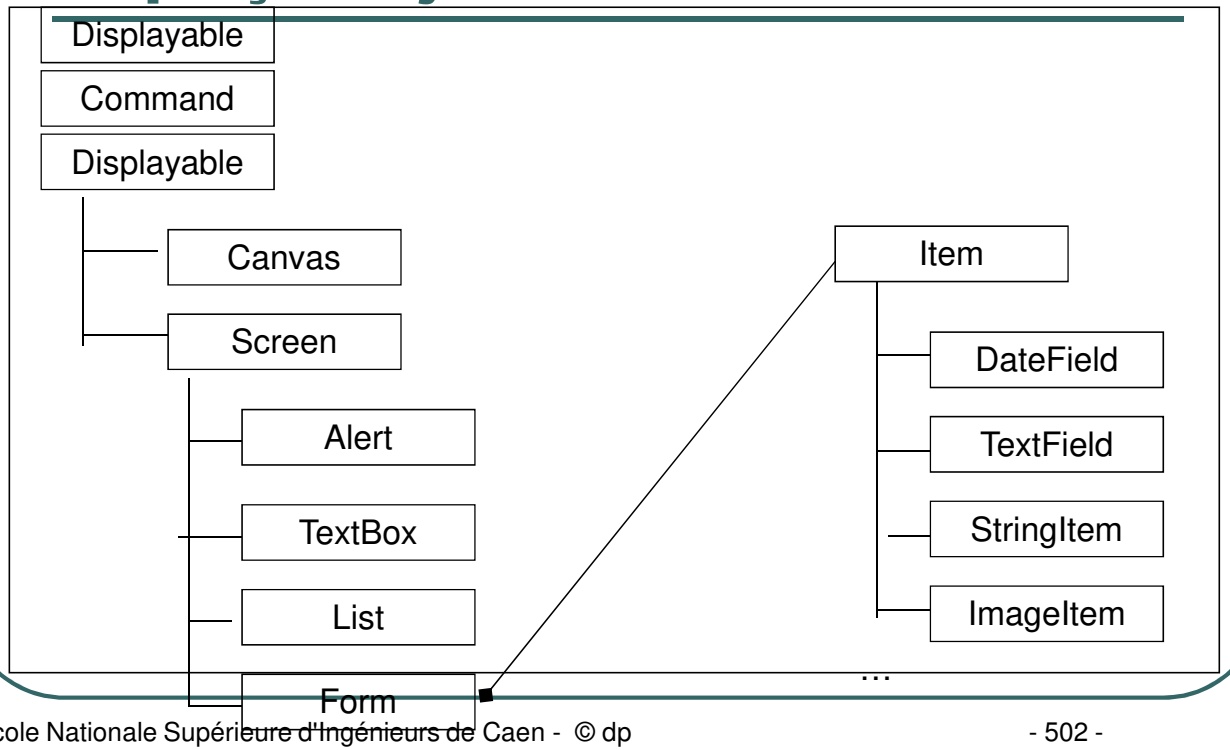
Le profile MIDP

- Le profile MIDP ajoute 3 packages aux packages de la configuration CDLC:
  - javax.microedition.midlet (cycle de vie de la midlet)
  - javax.microedition.lcdui (interface utilisateur)
  - javax.microedition.rms (persistance des données)
- Le profile MIDP 1.0 est conforme à la JSR 37
- Le profile MIDP 2.0 est conforme à la JSR 118

Cycle de vie d'une midlet



Aperçu de javax.microedition.lcdui



Exemple de midlet

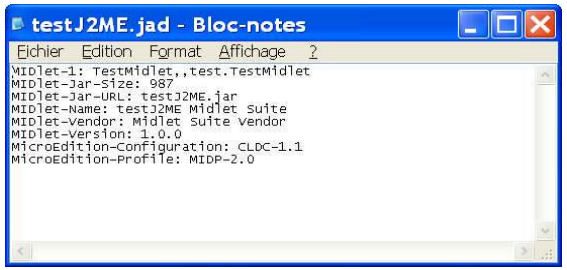
```
package test;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class TestMidlet extends MIDlet  {

  private Display display;
  private TextBox textbox ;

  public TestMidlet() {
    display = Display.getDisplay(this);
    textbox = new TextBox("Boite de saisie texte", "Bonjour tout le monde", 30, 0);
  }

  public void startApp() {
  display.setCurrent(textbox);
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
  }
}
```

Outils de développement

- J2ME Wireless toolkit (SUN) qui comprend:
  - un outil de développement (ktoolbar)
  - des émulateurs
- Plugin j2me pour eclipse pour le développement sous Eclipse (nécessite l'outil précédent).

Packaging d'une application

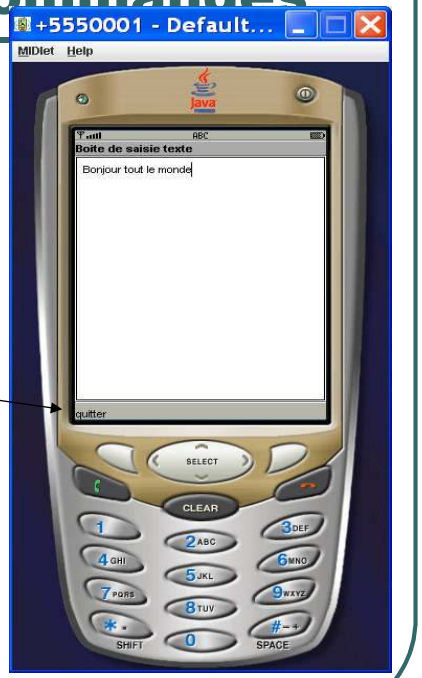
- Toutes les classes et ressources sont dans un fichier jar.
- Un descriptif se trouve dans un fichier jad (**J**ava **A**pplication **D**escriptor) et peut contenir la taille du jar, l'URL de téléchargement du jar, des paramètres éventuels, etc.



- Pour les Systèmes sous Palm OS, nécessité de convertir le couple JAD/JAR en PRC

Gestion des boutons de commandes

```
package test;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class TestMidlet extends MIDlet implements CommandListener {

  private Display display;
  private TextBox textbox ;

  public TestMidlet() {
    display = Display.getDisplay(this);
    textbox = new TextBox("Boite de saisie texte", "Bonjour tout le monde", 30, 0);
    Command quit = new Command ("quitter" , Command.EXIT, 1);
    textbox.addCommand(quit) ; textbox.setCommandListener(this) ;
  }
  public void startApp() { display.setCurrent(textbox); }
  public void pauseApp() {
  }
  public void destroyApp(boolean unconditional) {
  }
  public void commandAction(Command arg0, Displayable arg1) {
  System.exit(0) ;
  }
}
```



Bibliographie

Titre	*Auteurs*	*Editeur*
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**	**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**
Java Servlets	**Jason Hunter, William Crawford**	**O'Reilly**
Programmation java côté serveur	**Andrew Patzer**	**Eyrolles**
Le dictionnaire officiel Java 2	**Patrick Chan**	**Eyrolles**