Chapitre 1

Introduction à la programmation Java

Historique

- James Gosling (entré en 1984 chez Sun)
- WebRunner écrit en Oak : HotJava
- Une première version de Java (1.0) fut proposée en 1995
- Une nouvelle version de Java (1.1) est apparue en décembre 1996

Qu'est ce que Java?

- Un langage orienté objet
- Une machine virtuelle
- Une API
- Un environnement de programmation :

```
Logiciel de mise au point
Outil de génération de la documentation
Mini browser pour le développement des applets
. . .
```

API

API = ensemble de librairies pour la programmation

`Manipulation de chaîne de caractères`

```
String urlName = “ftp:://ftp.” + getCompany() +“.com” ;
```

Réseau

```
URL url = new URL (urlName) ;
```

API .

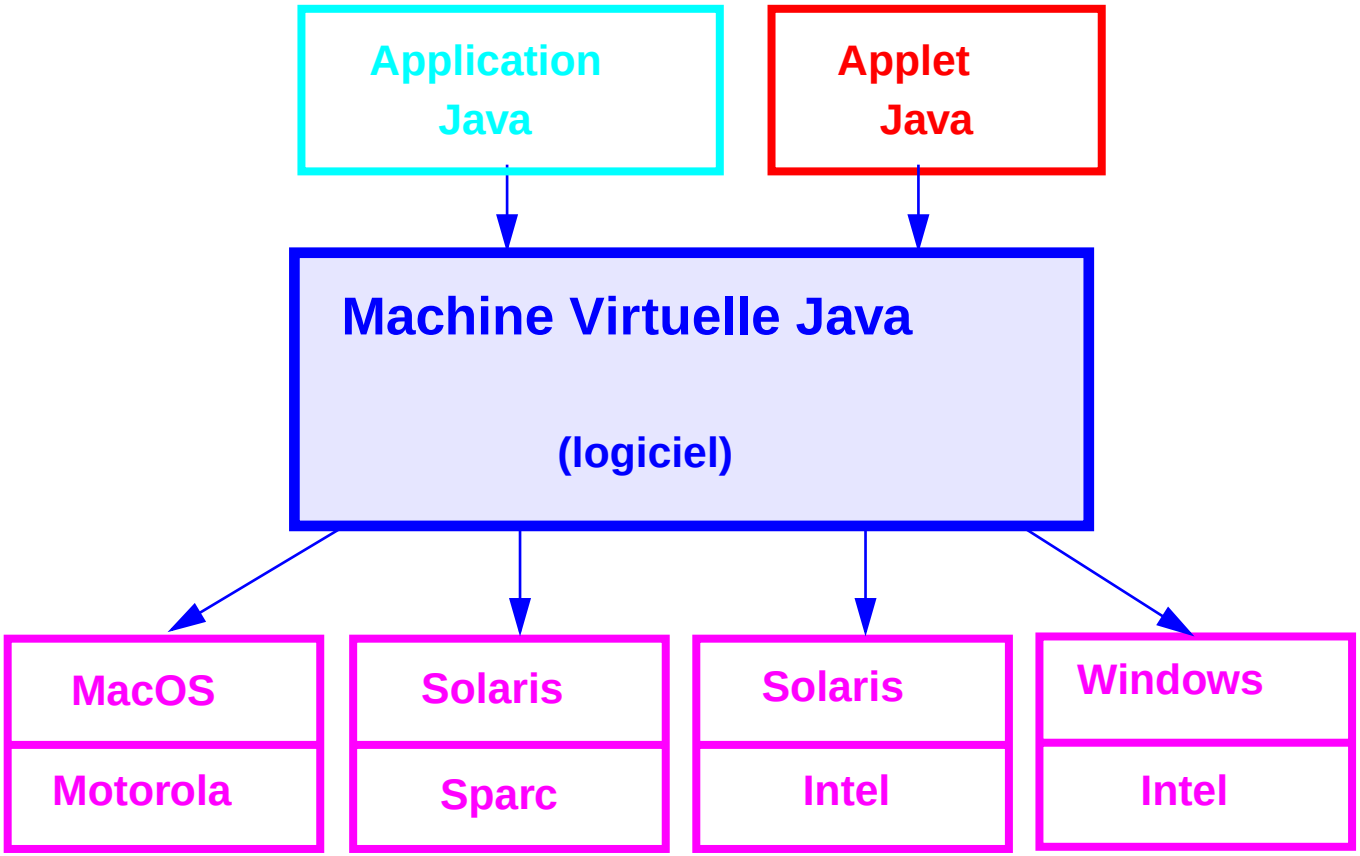
`Entrée / Sortie`

```
donnees = new DataInputStream (url.openStream()) ;
byte valeur= donnees.readByte () ;
```

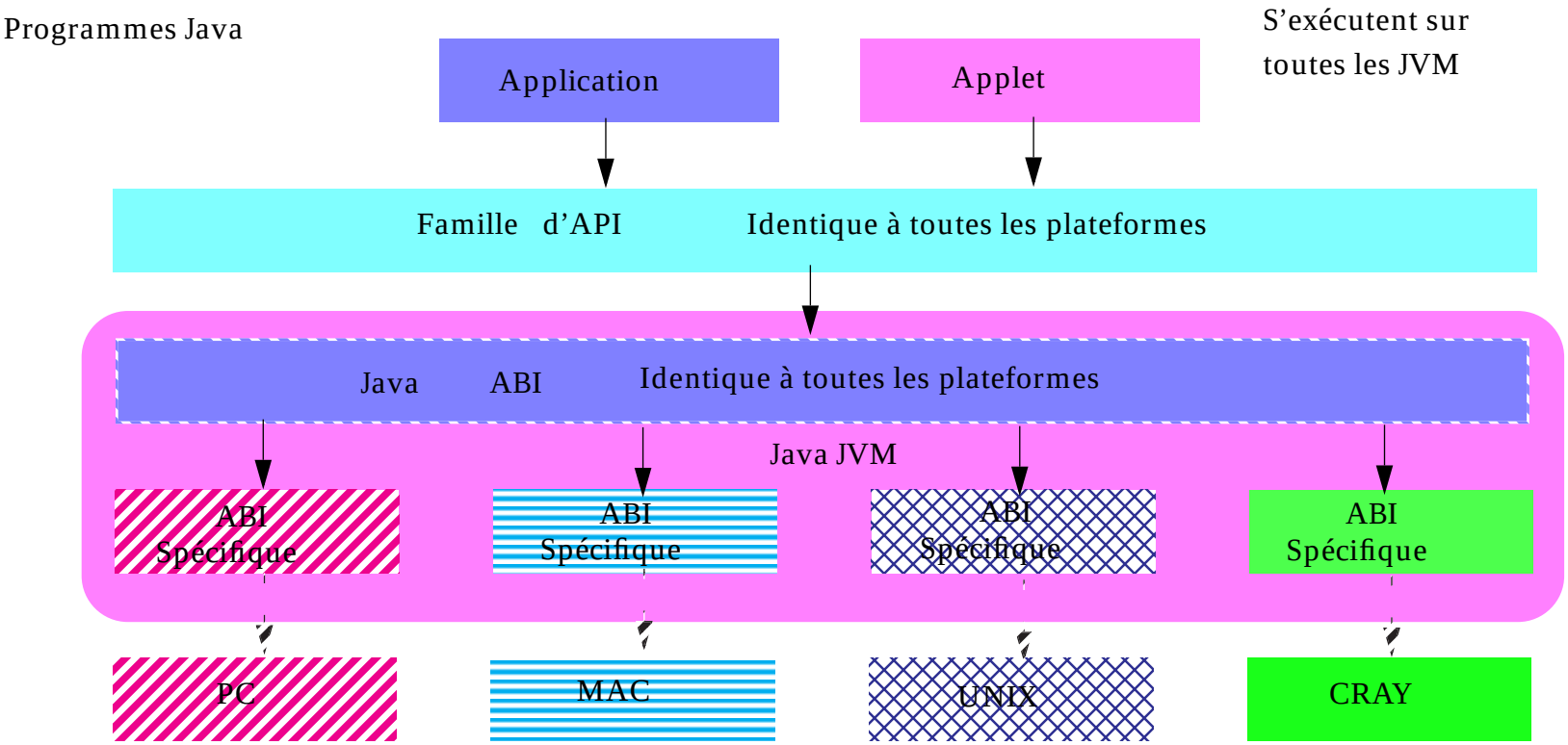
`Environnement Graphique`

```
Frame fenetre = new Frame (“Java Fenetre”) ;
fenetre.show() ;
```

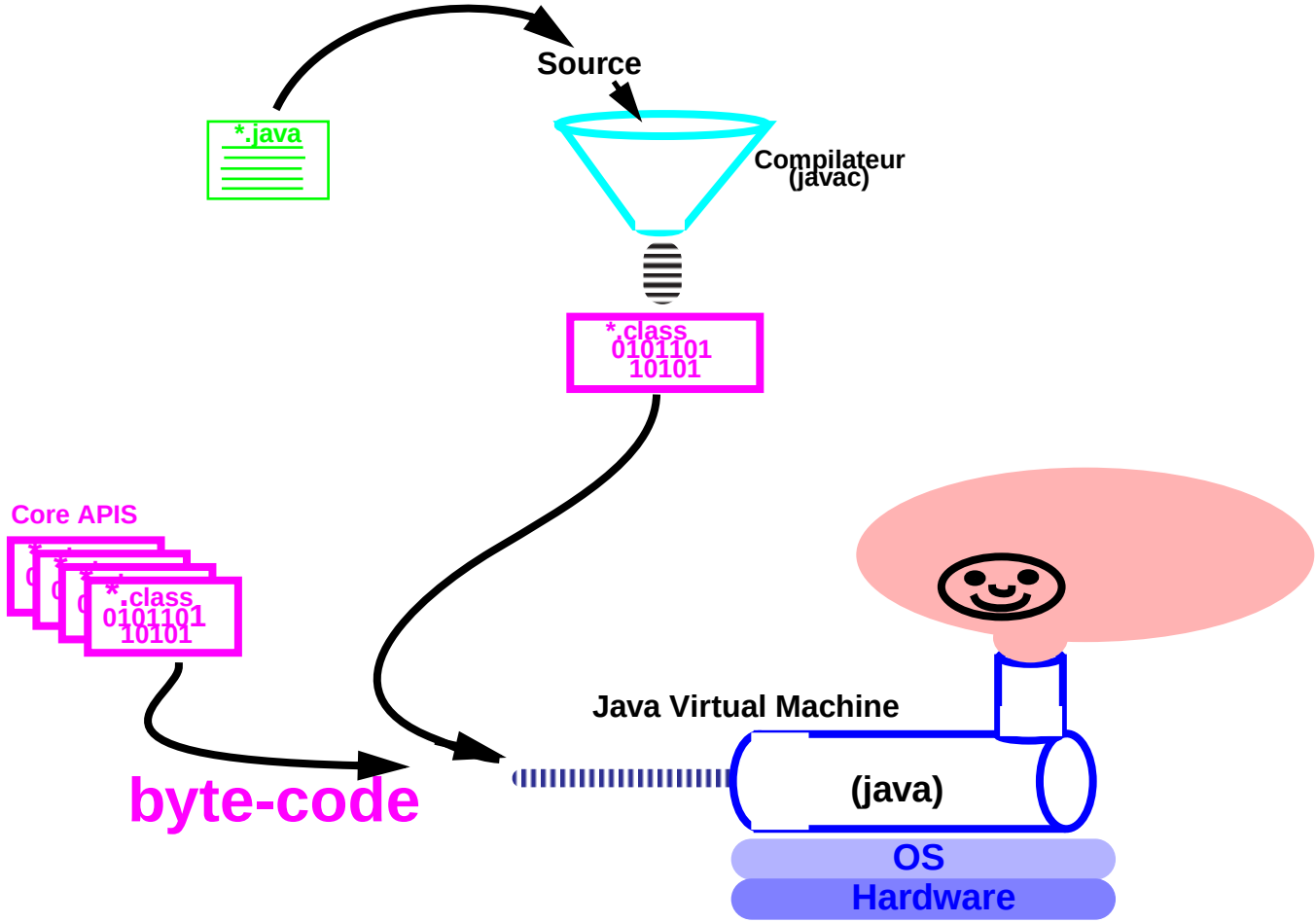
L´architecture Java



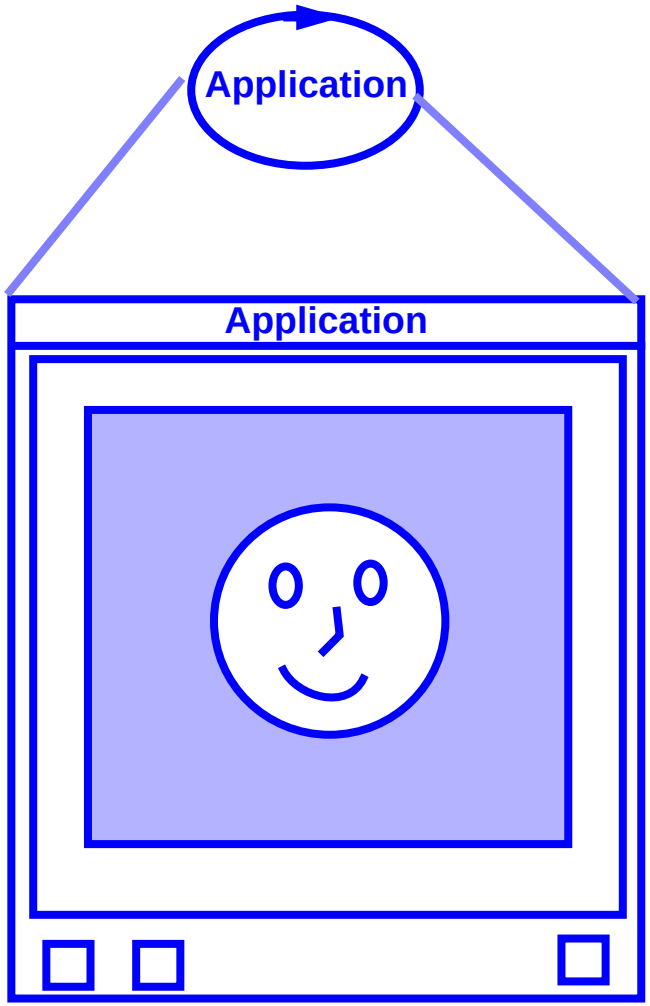
Architecture Java



L´exécutable



Application Java standalone

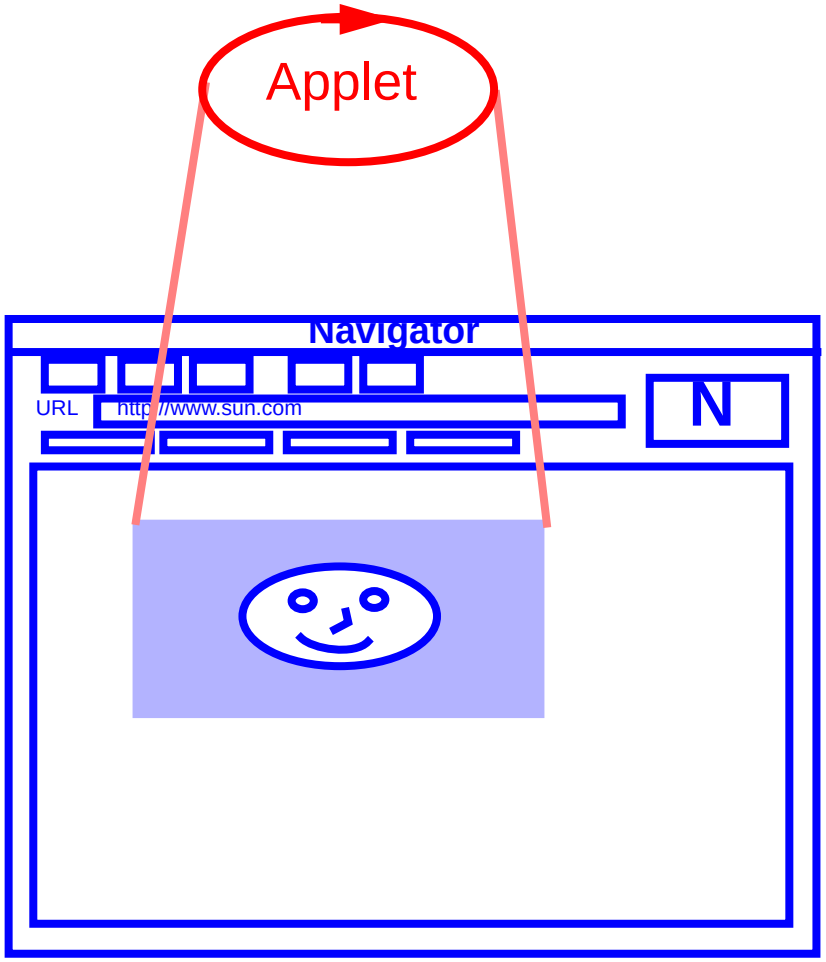


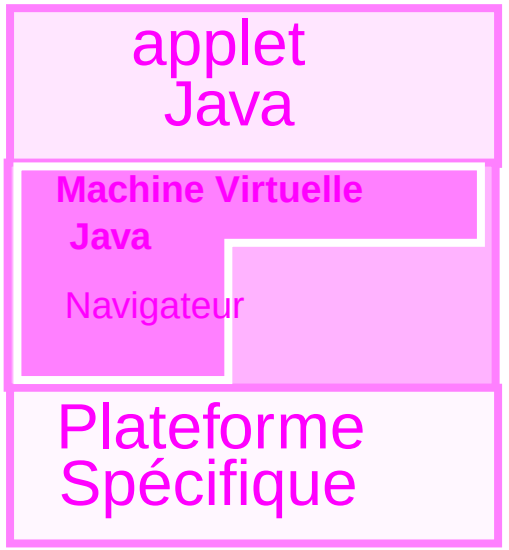
Application Java

Machine Virtuelle Java

Plateforme Spécifique

L'applet Java

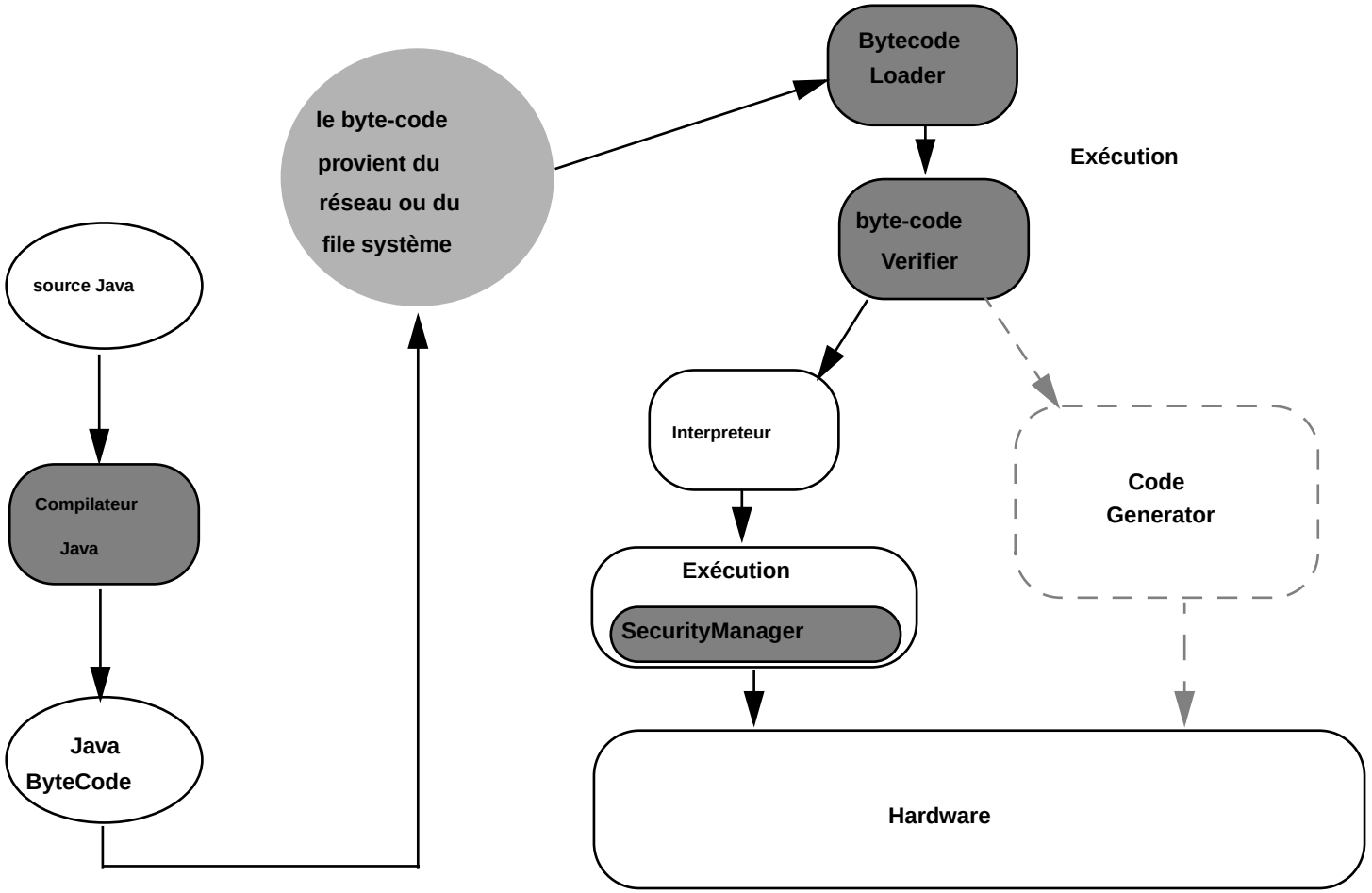




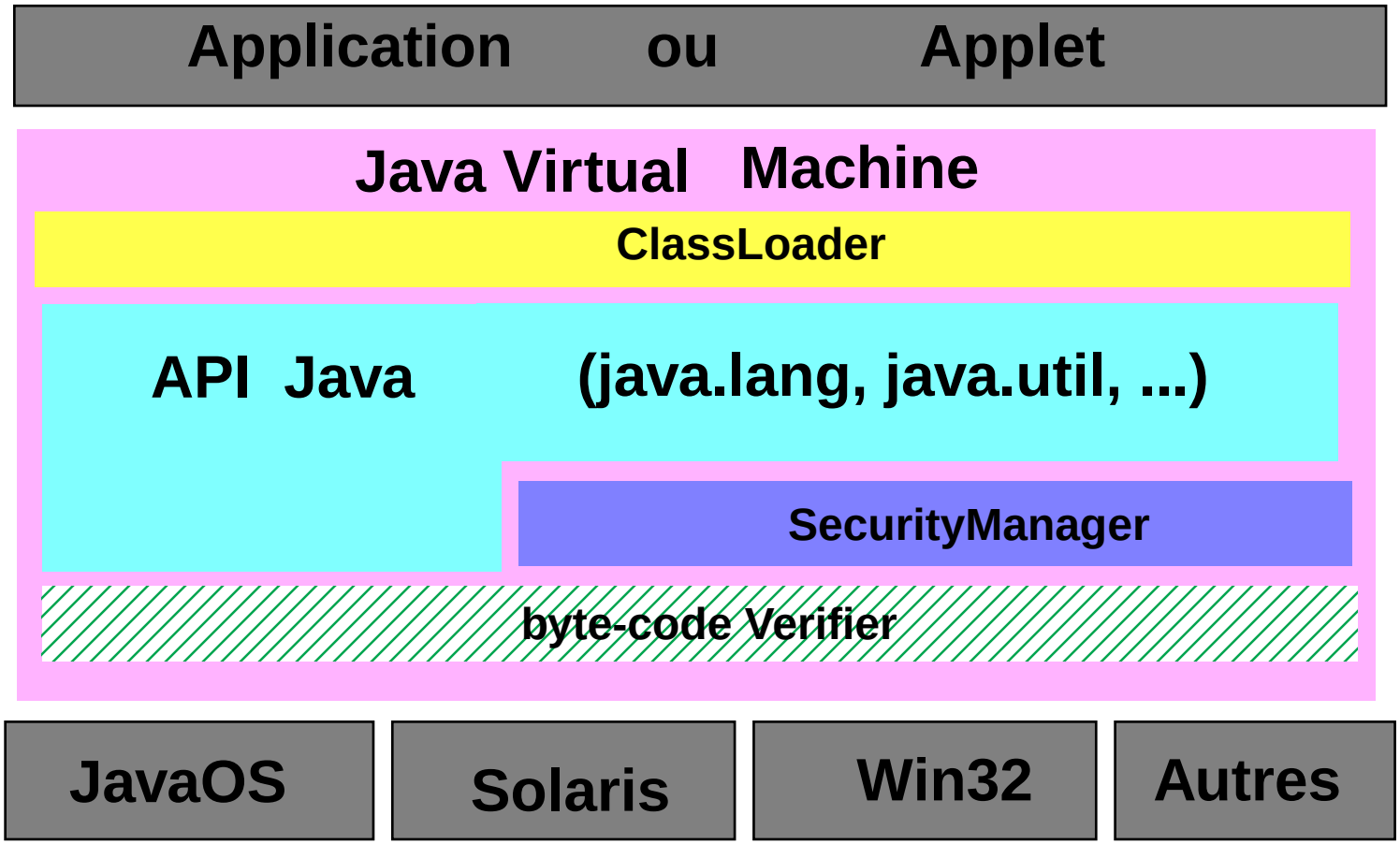
Caractéristiques Java

- Indépendant de la plate-forme
- Dynamique
- Orienté objet
- Simple
- Robuste
- Sécurisé
- Multithreadé

Sécurisé au niveau langage



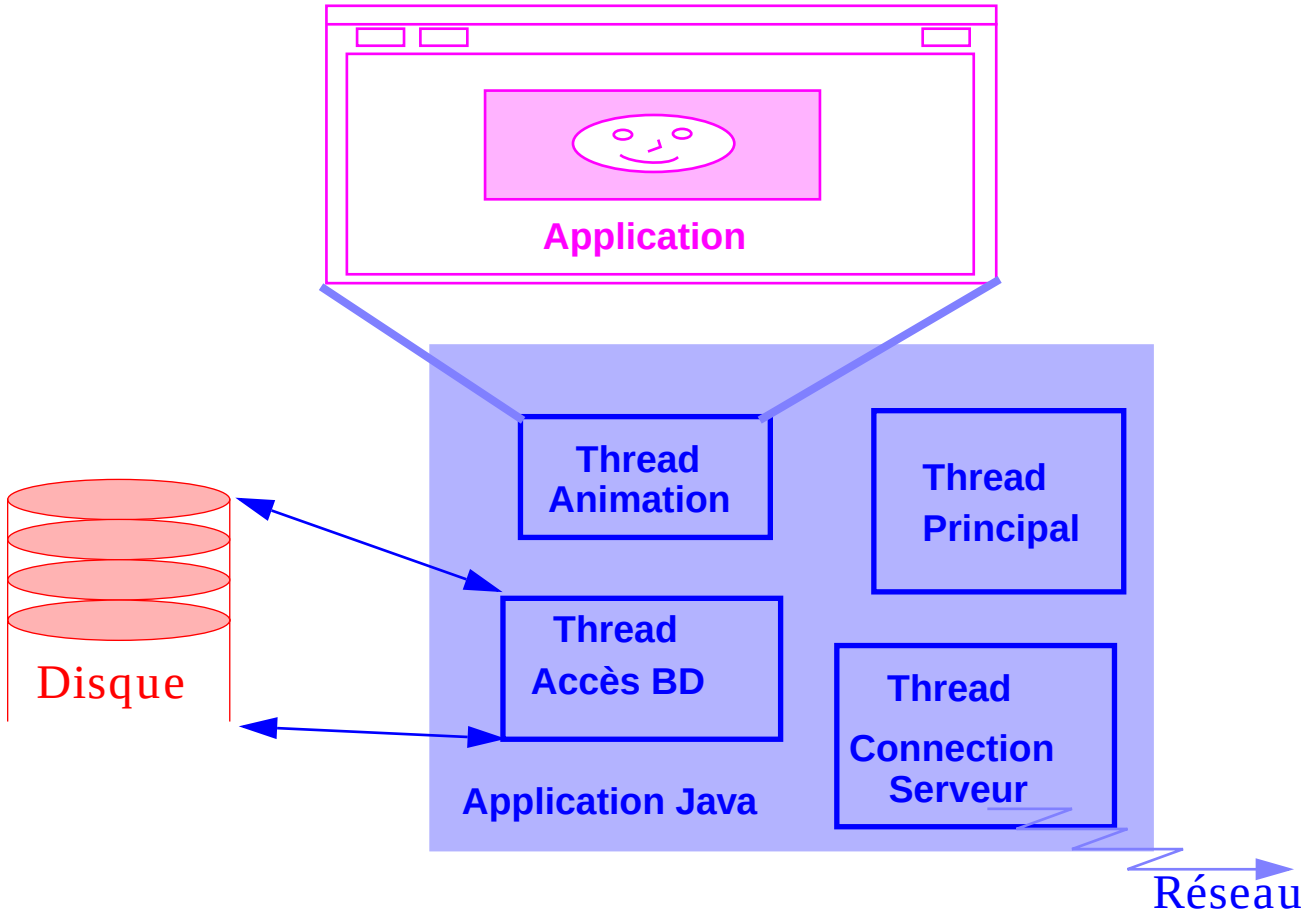
Sécurisé au niveau langage .



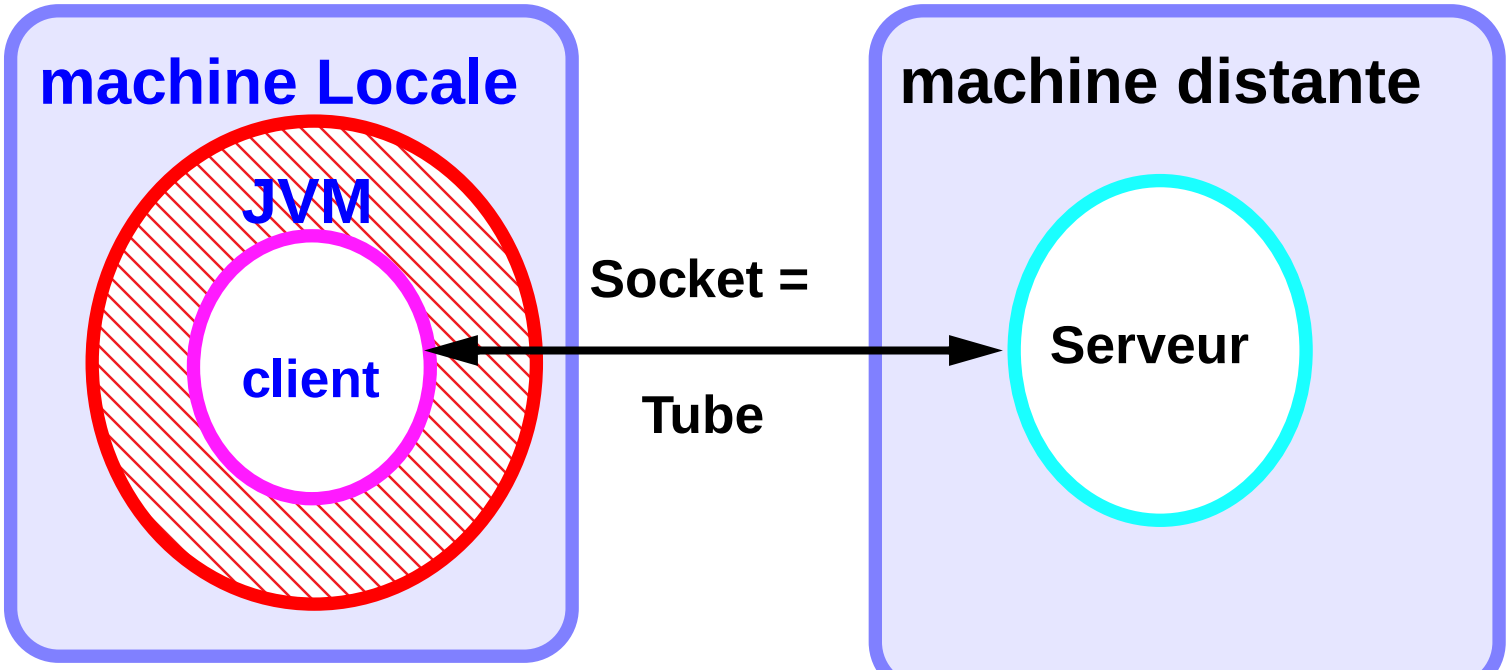
Sécurisé au niveau langage . .

- Limitation des erreurs
  Langage Java + Compilateur
- Vérification du byte-code avant exécution
  byte-code Verifier
- Chargement des classes dans un espace de noms fonction de sa provenance
  Class Loader
- Vérification pour l'accès aux ressources locales
  Security Manager

Langage Multi-Thréadé

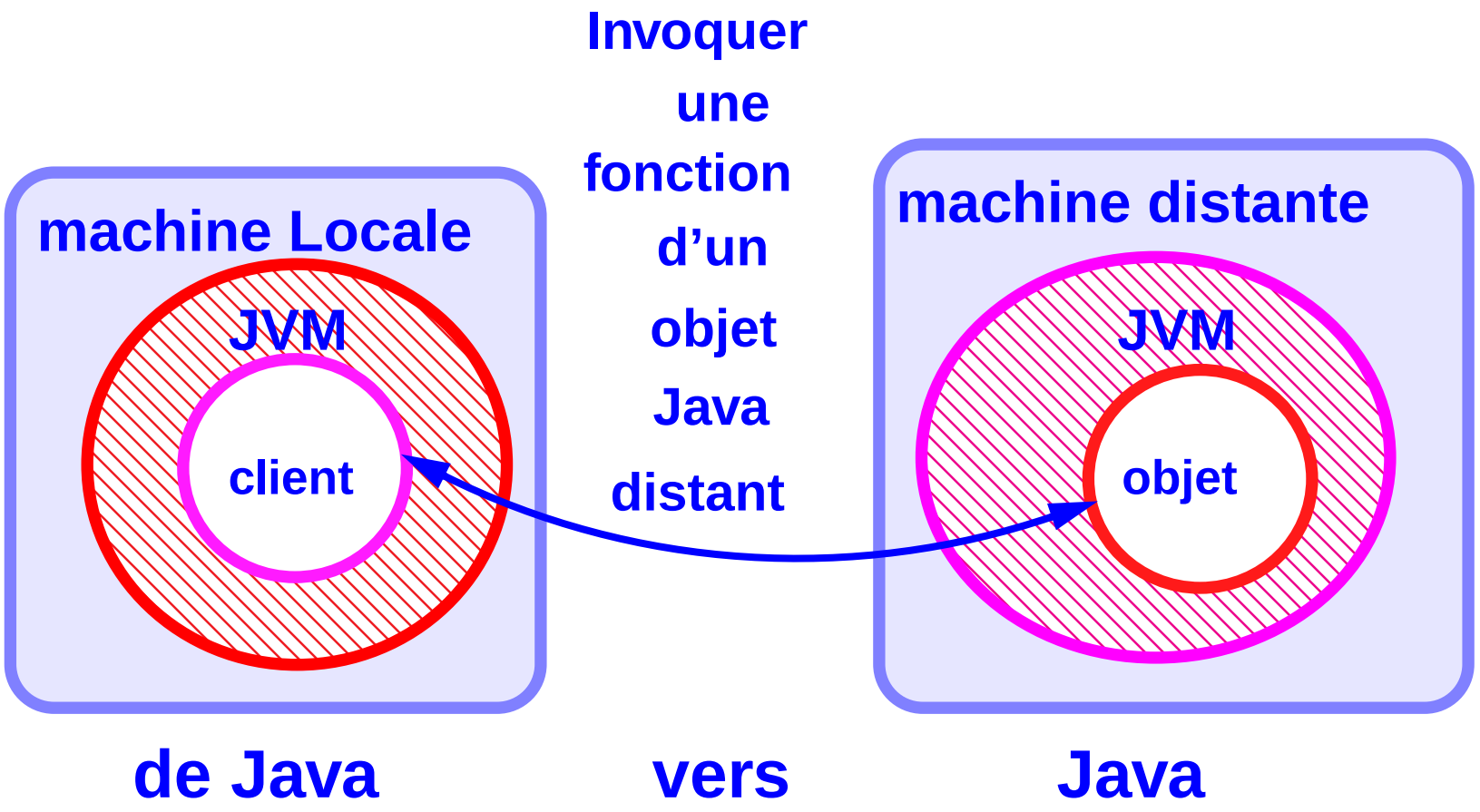


Le réseau - Java et les sockets

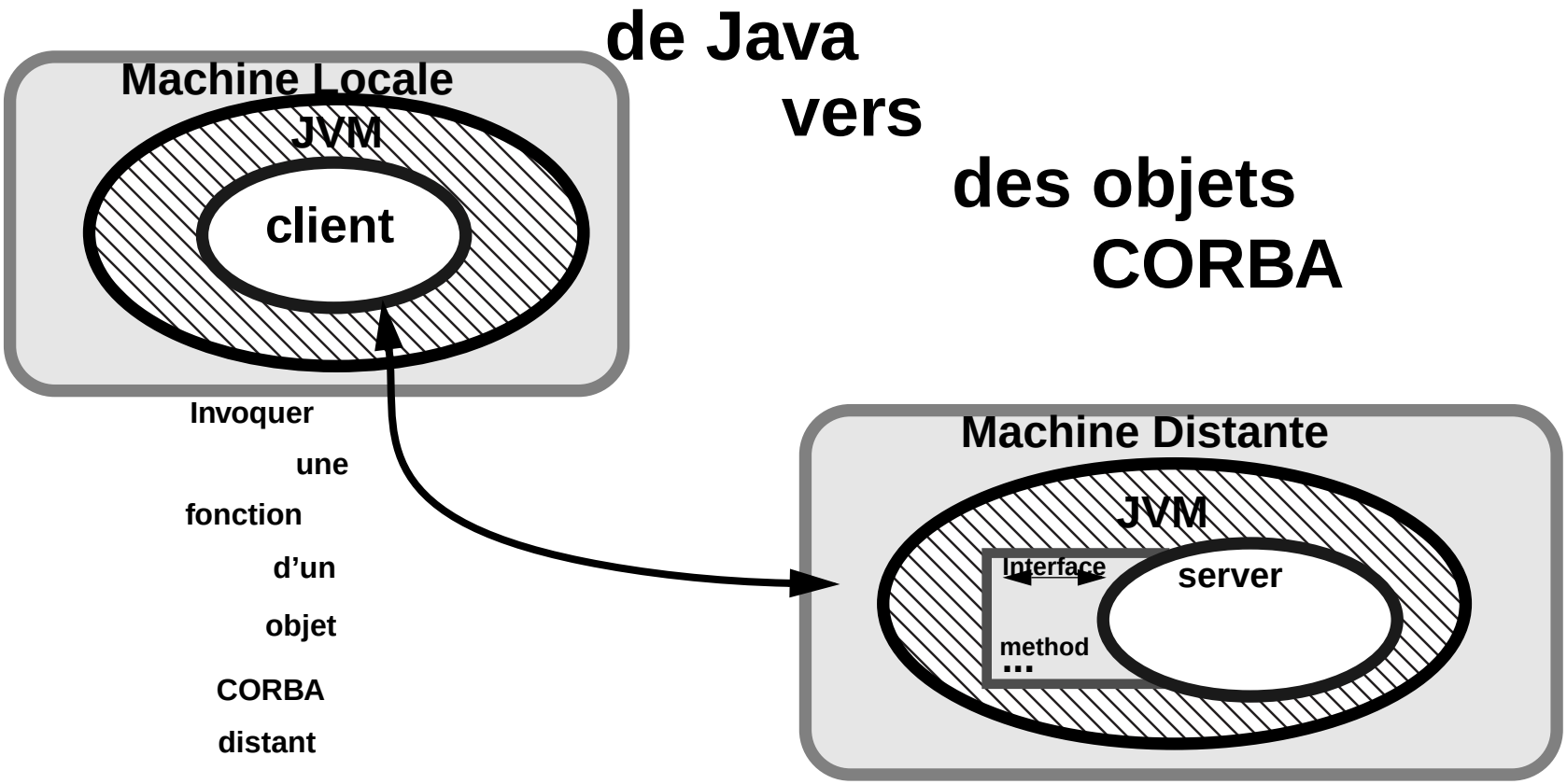


de Java vers Serveur

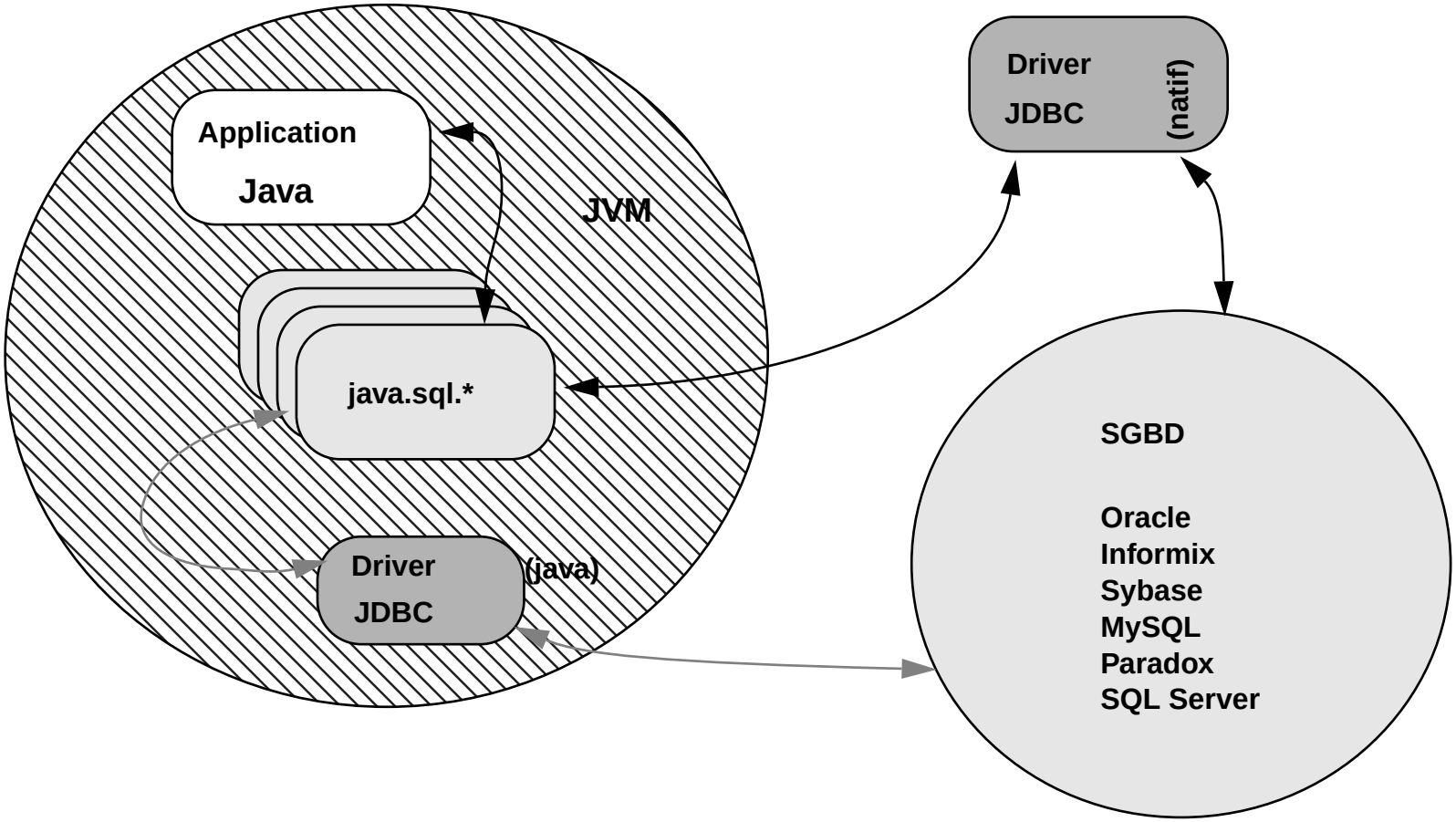
Le réseau - Java et RMI



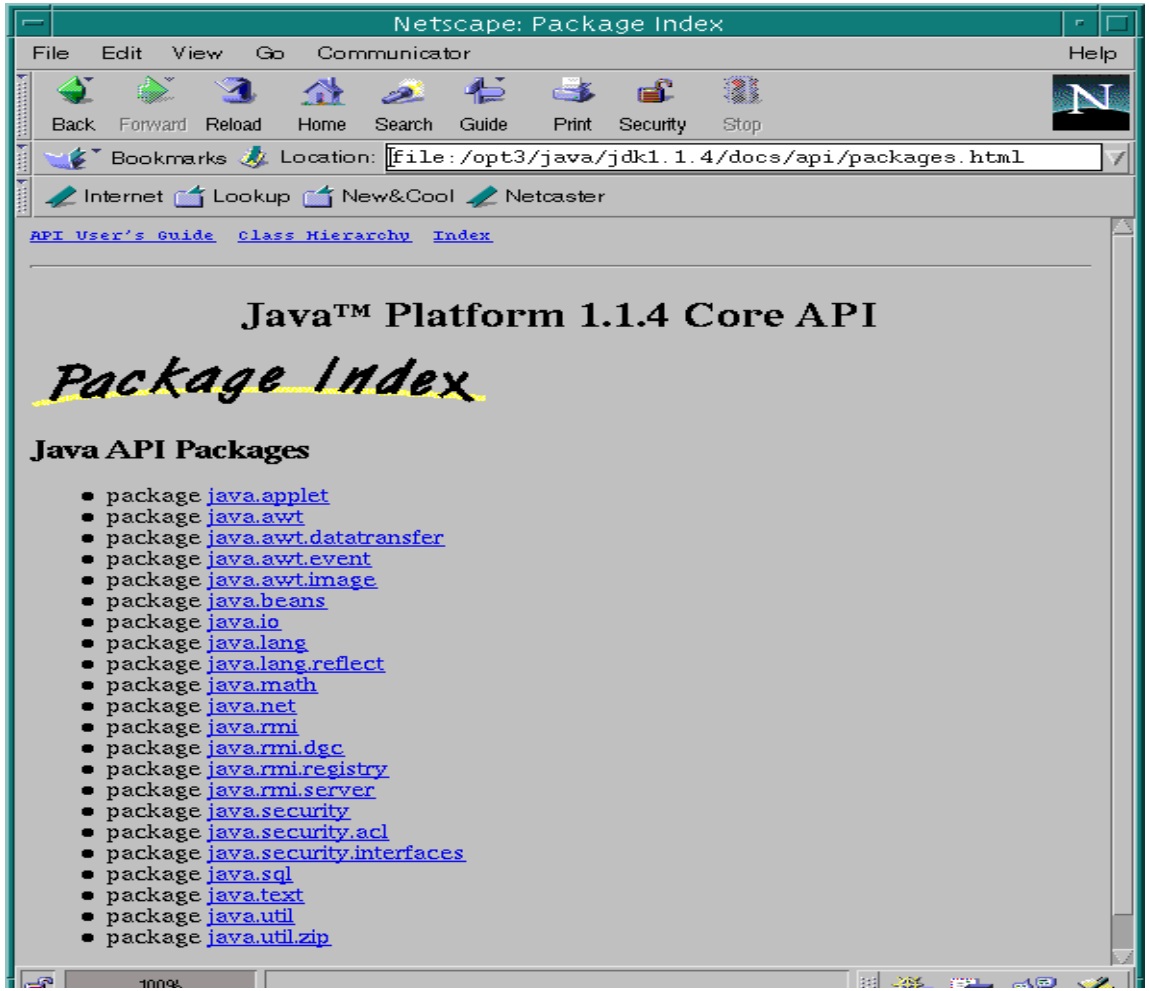
Hétérogénéité - Java IDL



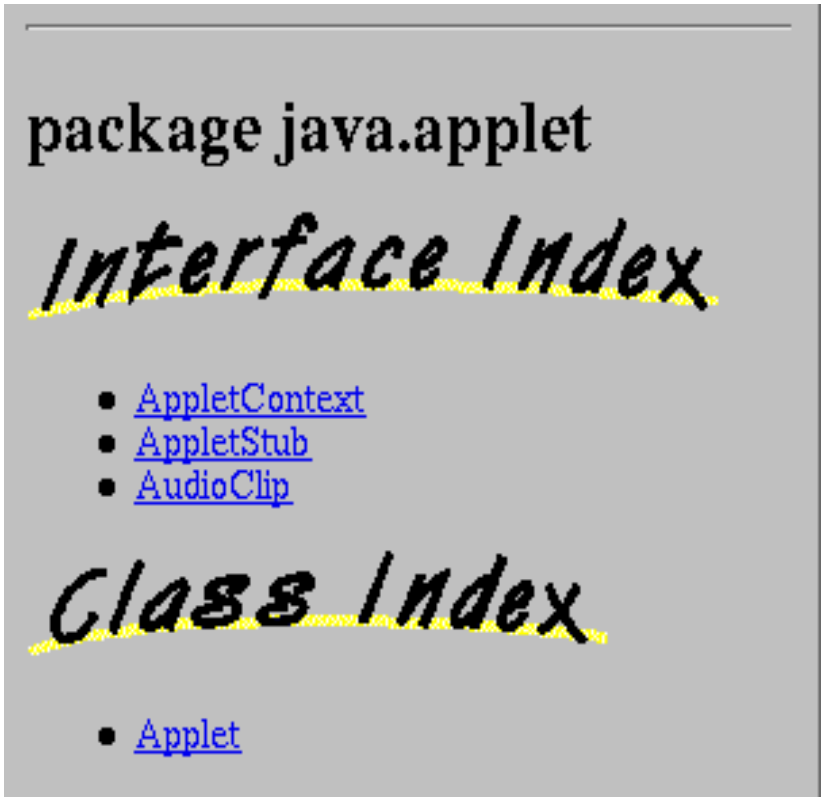
Accès aux bases de données JDBC



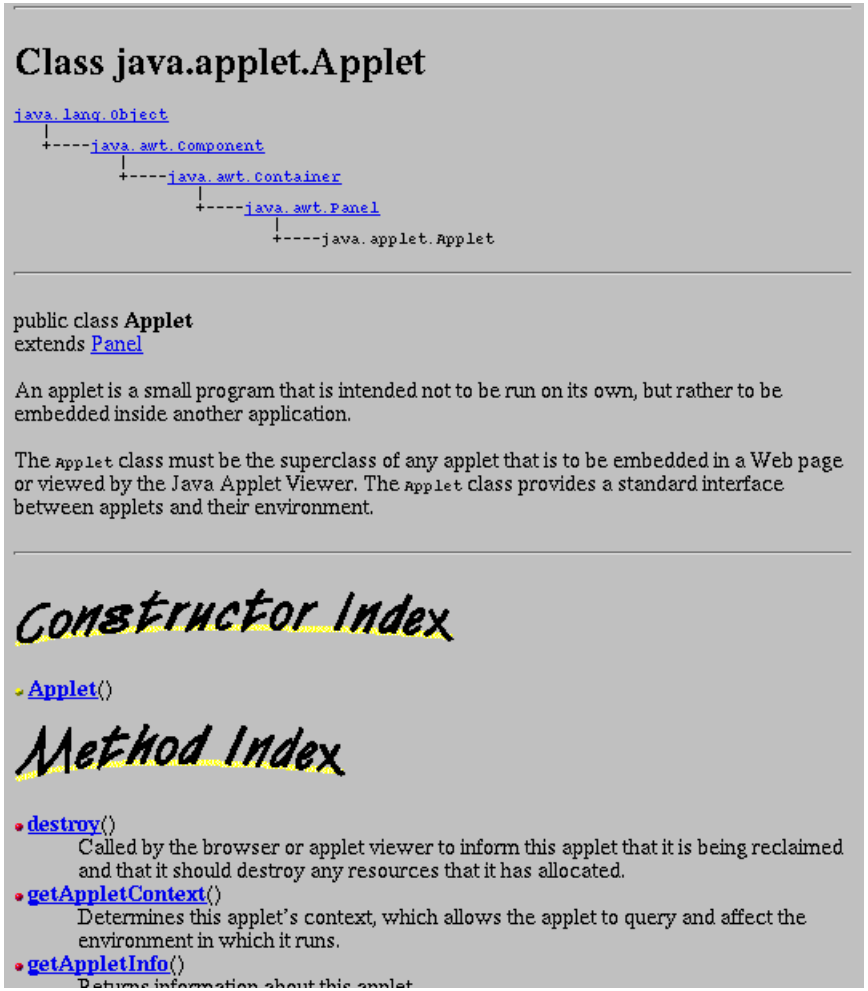
API Java : JDK



API Java : JDK .



API Java : JDK . .



Class java.applet.Applet

java.lang.Object
  |
  +----java.awt.Component
         |
         +----java.awt.Container
                |
                +----java.awt.Panel
                       |
                       +----java.applet.Applet

public class **Applet**
extends Panel

An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application.

The `Applet` class must be the superclass of any applet that is to be embedded in a Web page or viewed by the Java Applet Viewer. The `Applet` class provides a standard interface between applets and their environment.

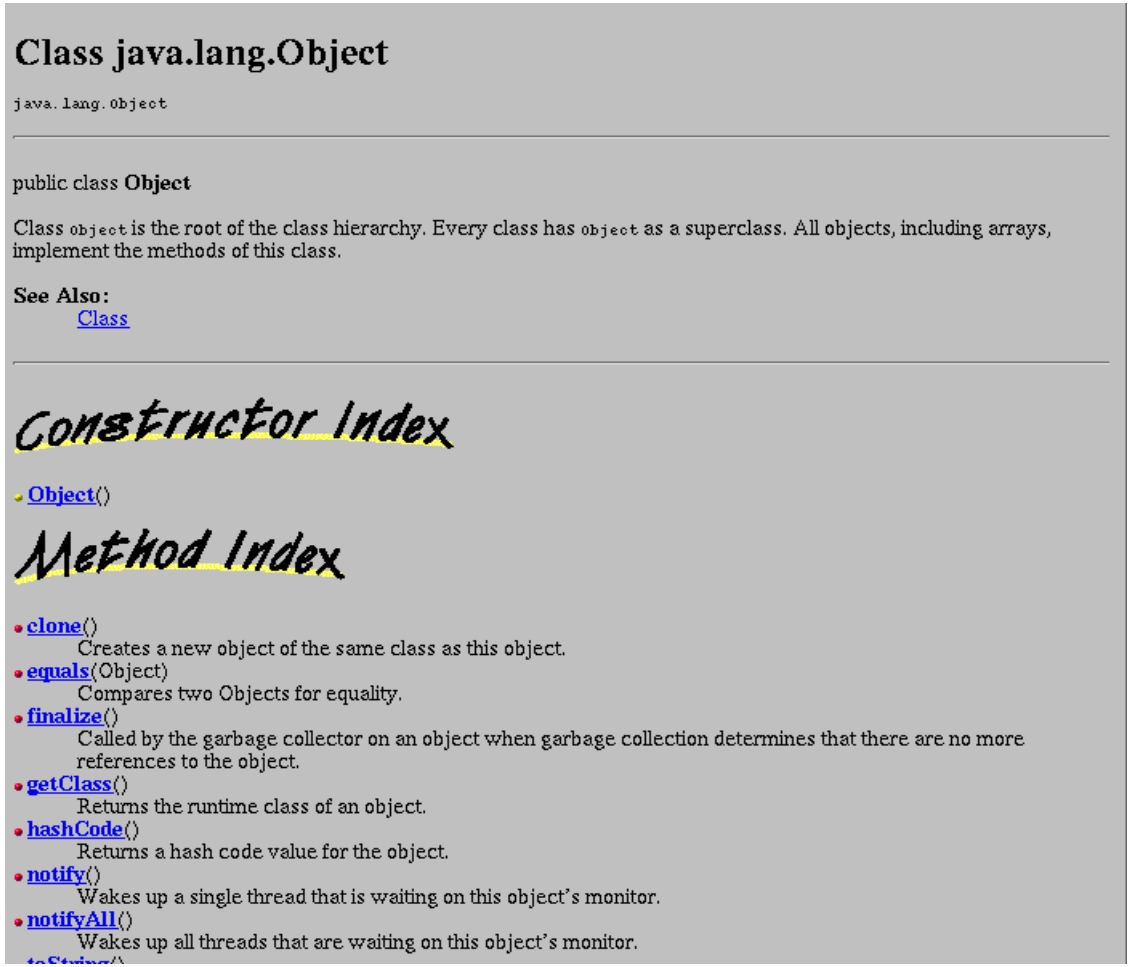
Constructor Index

- **Applet**()

Method Index

- **destroy**()
  Called by the browser or applet viewer to inform this applet that it is being reclaimed and that it should destroy any resources that it has allocated.
- **getAppletContext**()
  Determines this applet's context, which allows the applet to query and affect the environment in which it runs.
- **getAppletInfo**()
  Returns information about this applet

Utilisation de la documentation Java



Class java.lang.Object

java.lang.Object

---

public class **Object**

Class `object` is the root of the class hierarchy. Every class has `object` as a superclass. All objects, including arrays, implement the methods of this class.

See Also:
Class

Constructor Index

- **Object**()

Method Index

- **clone**()
  Creates a new object of the same class as this object.
- **equals**(Object)
  Compares two Objects for equality.
- **finalize**()
  Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
- **getClass**()
  Returns the runtime class of an object.
- **hashCode**()
  Returns a hash code value for the object.
- **notify**()
  Wakes up a single thread that is waiting on this object's monitor.
- **notifyAll**()
  Wakes up all threads that are waiting on this object's monitor.

Exercice 1

Utilisation de la documentation en ligne:

▼ **- Lancer netscape**

1. En utilisant le choix "Options"
2. Choisir une taille police qui vous convienne.

▼ **- Charger le fichier : /apps/Java/jdk1.1.2/docs/index.html**

1. Sélectionner sur Java Platform API pour visualiser l'ensemble des packages de l'API
2. En utilisant le choix "Bookmarks"
3. Mémoriser la référence de cette page
4. Sélectionner le package "java.lang"
5. Citer 3 classes appartenant à ce package.
6. Sélectionner la classe "System"
7. Quelle est la nature de "out" ?

Exercice 2

Votre première application standalone :

En vous inspirant de l'exemple ci-dessous, écrire un programme minimum permettant d'afficher le message :

"Bonjour de mon premier programme Java"

- **Lancer sa compilation**
- **lancer son exécution**

Code Source: dans le fichier PremiereApplication.java

```
public class PremiereApplication {
   public static void main(String args[]) {
      System.out.println( "Hello World");
   }
}
```

Exercice 3

Votre première applet:

A partir du code ci dessous générer un fichier source :

- **Compiler ce source**
- **Générer un fichier HTML associé :**

PremiereApplet.html en prévoyant une largeur de 300 et une hauteur de 100 pour la fenêtre d'affichage.

- **Lancer l'exécution de votre applet**
- `% appletviewer PremiereApplet.html`

Code Source:dans le fichier PremiereApplet.java

```
import java.applet.*;
import java.awt.*;
public class PremiereApplet extends Applet {
   public void paint(Graphics g) {
      g.drawString("Bonjour de ma premiere applet Java", 30, 30);
   }
}
```

Chapitre 2

APIs Java

La famille d'API Java

- Définition d'une API
- Définition d'une famille d'API
- Contenu d'une API
- Contenu d'un paquetage.

Librairies fournies avec le JDK

- Java language
- Gestionnaire graphique 'Windowing (GUI)'
- Système distribué 'Networking'
- Entrée / Sortie fichiers 'File I/O'
- Utilitaires 'Utilities'
- Java et le commerce
- Java et la sécurité

Librairies fournies avec le JDK (suite)

- Java entreprise comporte cinq APIs
  - Accès aux bases de données JDBC
  - Intéropérabilité Java IDL
  - Invocation de méthodes distantes RMI
  - Sérialisation d'objets
  - Interfaces de nommage NDI - JNDI

Librairies fournies avec le JDK (suite)

- Modèle de composants réutilisables Beans
- Java Media 2D, video et téléphonie.

Extensions standards (APIs)

- Java Media 3D, video, téléphonie
  - Son, Image et gestion de MIDI : Media Framework
  - Conférence, travail en répartie : Java Share
  - Graphisme 3D : Java 3D
  - Téléphonie : Java Tel

Extensions standards (APIs).

- Serveur et client en Java : Java Server
- Gestion - administration réseau : JMAPI

Extensions standards (APIs) . .

- Micro et Pico applications : Java embedded JEAPI

Chapitre 3

Le éléments du Langage

Identificateurs, mots-clés et type

- Commentaires

```
// commentaire sur une ligne
/* commentaire sur une ou plusieurs lignes */
/** commentaire d'explication */
```

- Points-virgules, blocs et blancs

```
{
 int x;
 x = 23 * 54;
}
```

- Identificateurs

```
nomUtilisateur

MaClasse var_sys1
```

Identificateurs, mots-clés et type

- Identificateurs

  `identificateurt, nomUtilisateur, MaClasse, var_sys1, $change.`
- Mots-réservés

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

Conventions de codage Java

- Conventions

  Classes :
  Interfaces :
  Constantes :
  Variables :

- Autres conventions

  Structures de contrôle :
  Espacement :

Identificateurs, mots-clés et types

- Types de base Java

  Catégorie "Logique" - type boolean
  Catégorie "Texte" - char *et* String
  String s ="Une souris verte qui sautait dans l'herbe."
  Catégorie "Entier" - byte, short, int et long

Longueur de l'entier	**Nom ou type**	**Plage**
8 bits	`byte`	-2^{7} ... 2^{7} -1
16 bits	`short`	-2^{15} ... 2^{15} -1
32 bits	`int`	-2^{31} ... 2^{31} -1
64 bits	`long`	-2^{63} ... 2^{63} -1

Catégorie Flottant - float *et* double

Longueur du flottant	**Nom ou Type**
32 bits	`float`
64 bits	`double`

Identificateurs, mots-clés et type

- Variables, déclarations et affectations

```
public class Assign {
     public static void main(String args []) {
      int x = 0, y = 0;// déclare et affecte les variables int
      float z = 3.414f;// déclare et affecte la variable flottante
      double w = 3.1415;// déclare et affecte la variable double
      boolean truth = true;// déclare et affecte la variable booléenne
      char c = 'a';// déclare la variable caractère
      String str = null;// déclare la variable chaîne
      String str1 = "bye";// déclare et affecte la variable chaîne
      c = 'A';  // affecte une valeur à la variable
                 // caractère
      str = "Hi out there!";// affecte une valeur à la variable chaîne
      x = 6;
      y = 1000;  // affecte des valeurs aux variables
                 // entières
     ...
     }
}
```

Expressions

- Variables et Cycle de vie

  `int, float, classe, ... , automatic, local, temporary, stack.`

- Initialisation de variable

`byte`	`0`
`short`	`0`
`int`	`0`
`long`	`0L`
`float`	`0.0f`
`double`	`0.0d`

char	'\u0000' (NULL)
boolean	false
Tout autre type	null

Expressions

- Opérateurs

R to L	. [] ()
R to L	++ -- + - ~ ! (cast_operator)
L to R	* / %
L to R	+ - (unaire)
L to R	<< >> >>>
L to R	< > <= >= instanceof
L to R	== !=
L to R	&
L to R	^
L to R	|
L to R	&&
L to R	||
R to L	?:

Expressions

- Expressions Logiques ont un résultat *boolean*
- Concatenation de String avec +

```
String handle = "Prof.";
String name = "First " + "Last";
String title = handle + name;
```

- Short Circuit Logical Operators

```
String unset = null;
if ((unset != null) && (unset.length() > 5)) {
     // do something with unset
}
```

Expressions

- Opérateurs de décalage à droite >> et >>>

  `128 >> 1` donne `64`

  `-256 >> 4` donne `-32`

  `1010 ... >> 2` donne `111010 ...`

  `1010 ... >>> 2` donne `001010 ...`

- Promotion et forçage (cast) des expressions

```
long bigval = 6;// 6 is an int type, OK
int smallval = 99L;// 99L is a long, illegal

float z = 12.414F;// 12.414F is float, OK
float z1 = 12.414;// 12.414 is double, illegal
```

- Conversion de type de donnée (Cast)

```
long bigValue = 99L;
int squashed = (int)(bigValue);
```

Contrôle de flux

- Spécification de branchement
  `if`,`else` *Statements*
  `switch` *Statement*

- Exemple :

```
int count;
count = getCount();// a method defined in the program
if (count < 0) {
     System.out.println("Error: count value is negative!");
}
else {
     System.out.println("There will be " + count +
     " people for lunch today.");
}
```

Contrôle de flux

- `switch` Statement

```
switch (colorNum) {
     case 0:
      setBackground(Color.red);
      break;
     case 1:
      setBackground(Color.green);
      break;
     default:
      setBackground(Color.black);
      break;
}
```

Contrôle de flux

- Spécifications de boucles

`for` *Loops*

Exemple

```
for (int i = 0; i < 10; i++) {
     System.out.println("Are you finished yet?");
}
System.out.println("Finally!");
```

`while` *Loops*

Exemple

```
int i = 0;
while (i < 10) {
     System.out.println("Are you finished yet?");
     i++;
}
System.out.println("Finally!");
```

Contrôle de flux

- Spécifications de boucle (suite)
  - do *Loops*

```
Exemple
int i = 0;
do {
     System.out.println("Are you finished yet?");
     i++;
} while (i < 10);
System.out.println("Finally!");
```

Contrôle de flux

- Spécification pour le contrôle de flux des boucles.

```
break [label];
continue [label];
label: statement;
```

Contrôle de flux

- Exemples

```
loop: while (true) {
      for (int i=0; i < 100; i++) {
       switch (c = in.read()) {
        case -1:
        case ‘\n‘:
         // jumps out of while-loop to line #12
         break loop;
        ...
       }
      } // end for
     }// end while

test:for (...) {
      ...
      while (...) {
       if (j > 10) {
        // jumps to next iteration of for-loop at line #13
        continue test;
```

```
            }
          }
        }
      }
```

Les tableaux

- Déclaration de tableaux

```
char s[];
Point p[];
```

- Création de tableaux

```
p = new Point[100];
```

- Initialisation de tableaux

```
String names[] = {
 "Georgianna",
 "Jen",
 "Pete",
 "Tom"
};
```

Les tableaux

- Tableau multi dimensionnel

```
int twoDim [][] = new int [4][];
twoDim[0] = new int[5];
twoDim[1] = new int[5];
```

- Limites de tableau

```
int list[] = new int [10];
```

- Copie de tableau

```
int elements[] = new int[6];// tableau d'origine
int hold[] = new int[10];// nouveau tableau

System.arraycopy(elements, 0, hold, 0,
        elements.length);
elements = hold;//la référence éléments pointe
        //maintenant sur le  nouveau
        //tableau hold.,
```

Exercice

Illustration des tableaux:

▼ **- Ecrire un programme `Tab1Prog.java`**

1. Définir un tableau de caractères initialisé avec le mot "Bonjour"
2. Définir un String contenant le texte "Bonsoir Madame"

▼ **- Afficher en utilisant println() les 2 données**

▼ **- Afficher ensuite la taille de ces 2 données**

Note – Un tableau est objet ayant une donnée length que l'on peut exploiter.

Note – Utiliser une méthode indiquée par la documentation pour la classe String.

`Code source`:

```
public class Tab1Prog{
    public static void main(String [] args) {
                ...
      }
}
```

Exercice

Illustration des tableaux et passage d'arguments:

▼ **- Ecrire un programme Calc.java**

Ce programme se lance en lui passant 3 arguments:

1er arg = 1er opérande (entier)
2ième arg = 2ième opérande (entier)
3ième arg = opérateur (+ - * /)

Exemple: %java Calc 5 8 +
ou : %java Calc 5 8 '*' (Attention)

`Il affiche le résultat sur la sortie standard`

Note – Utiliser la méthode de classe Integer.parseInt()
Utiliser la methode de classe String.charAt()

Note – Contrôler le nombre de paramètres reçus = 3

Note – Lorsque votre programme fonctionne . . .
Tester la division par zéro . . .

Chapitre 4

Les notions orientées objets

Compréhension des objets

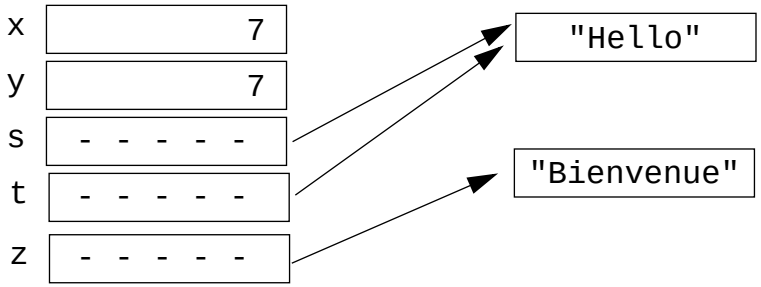
- Regroupement de types de données

```
class Individu {
   String nom;
   int age;
   float salaire;
}


Individu un, deux;

un.nom = "Jean";
un.age = 24;
un.salaire = 10000;
```

Compréhension des objets

- Initialisation par défaut et valeur de référence `null`
- Création d'un objet

```
Exemple :
int x = 7;
int y = x;
String s = "Hello";
String t = s;
String z = "Bienvenue";
```



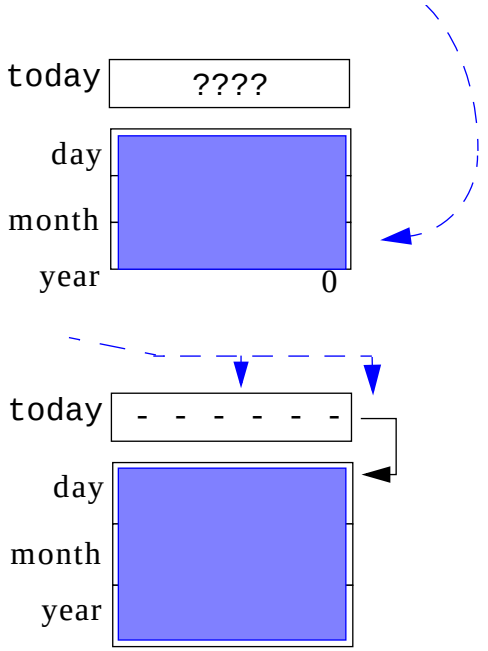
Compréhension des objets

```
class MaDate {
     int day;
     int month;
     int year;
}
MaDate mybirth;
mybirth = new MaDate();
```

MaDate today;

today ????

Compréhension des objets



Récapitulatif de la terminologie

- Regroupement de types de données
- Classe
- Objet
- Membre
- Référence

Méthode

- Introduction

```
void tomorrow(MaDate d);

MaDate d = ?????;// Objet date initialisé quelconque
d.tomorrow();

d.day
```

- Définition de méthodes

```
<modificateur> <type_retour> <nom> ( <liste_arguments> ) <bloc>


public void addDays(int days)
```

Classe

- Contrôle d'accès aux membres

```
class MaDate {
     private int day;
     private int month;
     private int year;
}

class MaDateUser {
     public static void main(String args[]) {
      MaDate mydate = new MaDate();
      mydate.day = 21; // incorrect !
     }
}



MaDate d = new MaDate();
d.day = 32;
d.month = 2; d.day = 30; // plausible mais faux
d.month = d.month + 1; // omettre le contrôle pour
                  //retour au début
```

Classe

- Encapsulation
- Récapitulatif terminologique

```
Méthode
Transmission de message
Contrôle d'accès
Encapsulation
```

Construction et initialisation des objets

- Initialisation explicite de donnée membre

```
public class Initialized {
     public int x = 5;
     public String name = “Fred”;
     public MaDate created = new MaDate();
}
```

- Constructeurs

```
1 - Le nom de la méthode doit correspondre exactement au nom de la classe.
2 - Aucun type de retour ne doit être déclaré pour la méthode.
Exemple :
public class Xyz {
     // variables membres
     public Xyz() {
      // définition de l’objet
     }
     public Xyz(int x) {
      // définition de l’objet avec un paramètre
     }
}
```

- Constructeur par défaut

Héritage

- Relation “est un(e)

```
public class Employee {
     String name;
     Date hireDate;
     Date dateOfBirth;
     String jobTitle;
     int grade;
}
```

- Mot-réservé *extends*

```
public class Manager extends Employee {
   String department;
   Employee [] subordinates;
}
```

- Héritage simple

Héritage

- Spécialisation des méthodes

```
class Employee {
     …
}

class Manager extends Employee {
     …
}
```

```
class Employee {
     void calculPrime() {
      // une prime pour un employe
      …
     }
}

class Manager extends Employee {
     void calculPrime() {
      // une prime plus importante
      // pour un manager
      …
     }
}
```

Polymorphisme

```
Employee e = new Employee();
e.calculPrime(); // lance la methode calculPrime()
     //de la classe Employee
mais aussi
Employee e = new Manager();
e.calculPrime(); // lance la methode calculPrime()
     //de la classe Manager
```

- Polymorphisme = liaison dynamique

```
Voulez vous creer un Manager ou un Employe ?
lire reponse

si reponse = Manager alors
     e = reference a un Manager
fsi
si reponse = Employe alors
     e = reference a un Employe
fsi

La prime de la personne est :
e.calculPrime();
```

- Conclusion

Exercice

Illustration de la création de la classe personnelle:
Variables privées
Méthodes publiques.
 - Créer un fichier indépendant Pile.java
Correspondant à une classe Pile indépendante:

Cette classe possède 2 variables privées:
Un tableau d'entiers correspondant à la pile.
Un entier correspondant à la position courante du sommet de pile.
Cette classe à 3 méthodes publiques:
public void empile(int i)
public int depile()
public int nb_elem()
Créer un programme PileProg.java qui utilise cette classe
Créer une instance p1 de Pile
Empiler 3 entiers
Demander le nombre d'eléments contenus dans la pile
Afficher en la vidant le contenu de la pile.

Exercice

Illustration de la création de la classe personnelle:
Mise en oeuvre de constructeurs
Définir une nouvelle classe Pile2
Créer un fichier independant Pile2.java à partir du fichier Pile.java
Prévoir 2 constructeurs:
1 ne recevant aucun paramètre (Taille par defaut = 100)
1 recevant un entier indiquant la taille initiale de la pile.
Ajouter une méthode taille() permettant de connaître la taille de la pile.
Créer un programme Pile2Prog.java qui utilise cette classe:
Créer une instance p1 de Pile2 avec une taille initiale de 500.
Afficher la taille de la pile.
Empiler 3 entiers.
Demander le nombre d'éléments contenus dans la pile.
Afficher en la vidant le contenu de la pile.

Exercice

Illustration de sous-classement: surcharge

1. Créer une classe Point (int x , int y).
2. Créer une classe Figure (Point p , String type_fig)
3. Creer une sous classe de Figure --> Rect
   (avec en plus int larg, int haut)
4. Créer une classe Fenetre correspondant a un tableau de Figures.

▼ **Créer une première version du programme JavaDraw.java**

Qui crée un objet Fenetre f capable de recevoir 20 Figures.

a. affecter à l'élément 0 de la fenêtre un rectangle x = 5 ,
   y = 10 , type_fig = Rectangle , largeur = 200 , hauteur = 300.

b. Afficher par la méthode toString() la description de l'objet Fenetre f.

c. Afficher par la méthode toString() la description de l'elément 0 de la fenêtre f
(On aura redéfini la méthode toString() des objets
Rect et Figure pour la rendre plus parlante).

Chapitre 5

Les applets Java

Qu'est-ce-qu'une applet ?

- Qu'est-ce-qu'une applet

  Une portion de code Java
- Chargement d'une applet

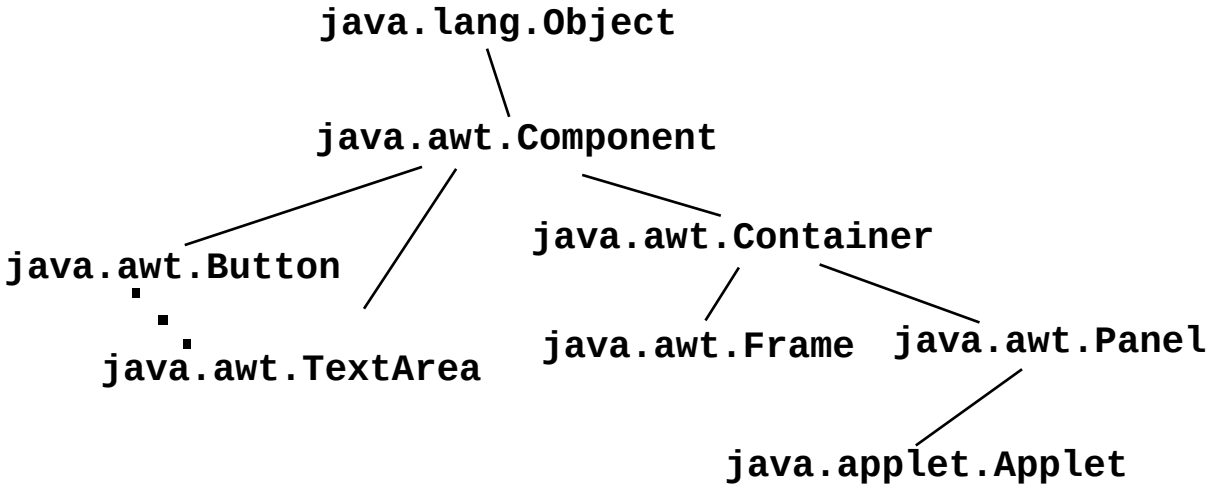
  Exécution dans un navigateur
- Balise applet

```
<applet code=HelloWorld.class width=100 height=100>
</applet>
```

Ecriture d'une applet

```
import java.applet.*;
public class HelloWorld extends Applet {
```

- Hiérarchie de la classe Applet



Ecriture d'une applet .

- Ordre d'appel

1. `init()` pour effectuer les initialisations
2. `start()` pour démarrer le traitement
3. `paint()`pour dessiner ou afficher des images

Ecriture d'une applet . .

- Affichage d´une Applet

  Ecriture de la méthode paint() associée à l'applet.

- Méthode *paint()* et graphiques

```
import java.awt.*;
import java.applet.*;
public class HelloWorld extends Applet {
  public void paint(Graphics g){
    g.drawString("Hello World!", 25, 25);
  }
}
```

Méthodes et cycle de vie d'une applet

- Ce sont les méthodes `init()`, `start()`, `stop()`, `destroy()` :

  Plus précisément :

  `public void init() { ... }`

  est lancée au moment où l'applet est chargée par le browser

  `public void start() { ... }`

  est lancée :

  - après `init()`
  - lorsque l'applet est "visitée" i.e. lorsque le navigateur revient sur la page HTML contenant l'applet (par le bouton "back" de browser) ou bien après désiconification du navigateur .

Méthodes et cycle de vie d'une applet .

`public void stop() { ... }`
est lancée lorsque l'applet n'est plus à l'écran : par exemple quand le
navigateur quitte la page HTML contenant l'applet ou lorsque le navigateur
est iconifié.

`public void destroy() { ... }`
est lancée lorsque l'applet est déchargée du cache du browser

en résumé :

```
init()
    start()
    stop()
destroy()
```

Méthodes et cycle de vie d'une applet

- `init()`
- `start()`

```
1  public void start() {
2     musicClip.play();
3  }
```

- `stop()`

```
1  public void stop() {
1     musicClip.stop();
2  }
```

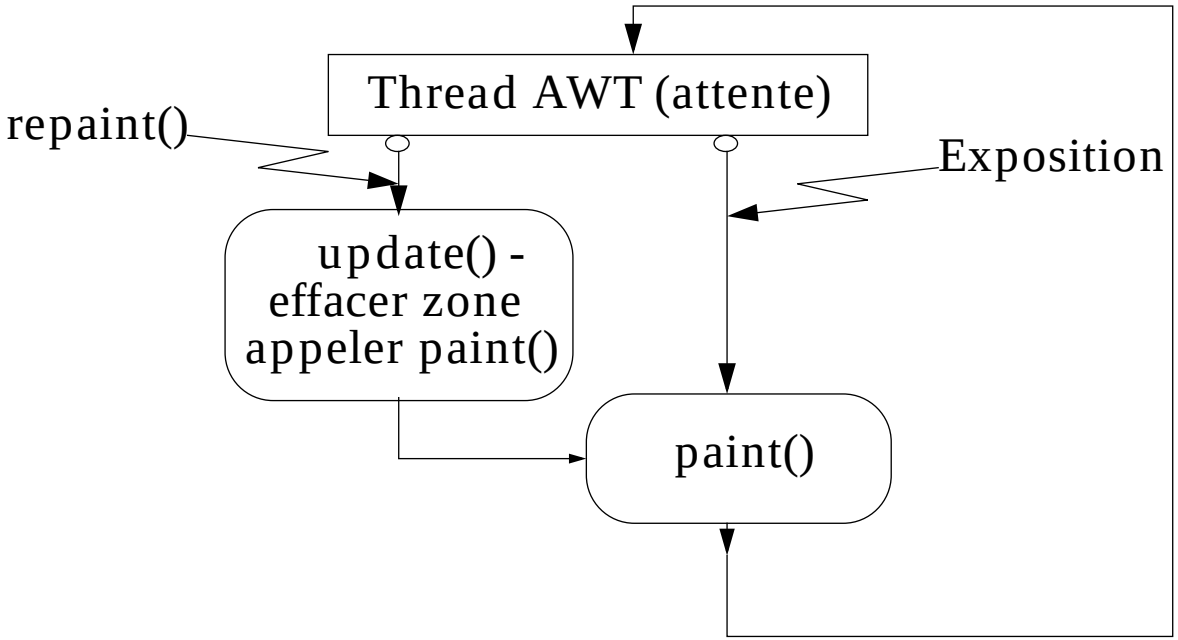
- `destroy()`

Dessin AWT

- Méthode `repaint()`
- Méthode `update(Graphics g)`

Dessin AWT .

- Diagramme des méthodes `paint()` `update()` `repaint()`



AppletViewer

- Synopsis

  ```
  appletviewer [-debug] urls . . .
  ```

- Fonctions du menu AppletViewer
  - Restart

    ```
    stop() ; start() ; paint()
    ```

  - Reload

    ```
    stop() ; destroy() ; init() ; start() ; paint()
    ```

- Utilisation d'AppletViewe

  ```
  Clone
  Tag
  Info
  Propriétés
  Quit
  ```

Cycle de vie d´une Applet

- Appel d'applets avec l'outil AppletViewer

```
<html><head><title> example </title></head>
<body>
<applet code=HelloWord.class width=100 height=100>
</applet>
</body>
</html>
```

1. Une instance de la sous classe Applet est créée. Dans cet exemple, il s'agit de la classe `HelloWorld`.
2. L'applet s'auto-initialise, c'est à dire que la méthode `init()` est appelée et exécutée sur cette instance.

Cycle de vie d´une Applet .

3. L'exécution de l'applet commence, c'est à dire que la méthode `start()` est appelée et exécutée sur cette instance.

   - Vous pouvez voir dans la fenêtre :

     ```
     status: applet loaded
       status: applet initialized
       status: applet started
     ```

   - Puis, lorsque vous quittez la fenêtre autonome, vous voyez :

     ```
     status: applet stopped
       status: applet destroyed
       status: applet disposed
     ```

Balise Applet

- Syntaxe
  - Voici la syntaxe complète de la balise `applet` :

```
<applet
  code=appletFile.class
  width=pixels height=pixels
  [codebase=codebaseURL]
  [alt=alternateText]
  [name=appletInstanceName]
  [align=alignement]
  [vspace=pixels]  [hspace=pixels]
>
  [<param name=appletAttribute1 value=value>]
  [<param name=appletAttribute2 value=value>]
  . . . . .
  [alternateHTML]
</applet>
```

- Description

Test d'image simple

- Applet

```
//Extension de HelloWorld pour tracer une image
//Suppose l'existence de
//"graphics/joe.surf.yellow.small.gif"
//
import java.awt.*;
import java.applet.Applet;

public class HwImage extends Applet {
   Image duke;

     public void init() {
      duke = getImage(getDocumentBase(),
         "images/joe.surf.yellow.small.gif");
     }

     public void paint(Graphics g) {
      g.drawImage(duke, 25, 25, this);
     }
}
```

Clips audio

- Ecoute d´un clip

```
play(URL soundDirectory, String soundfile);
```

ou plus simplement :

```
play(URL soundURL);

play(getDocumentBase(), "bark.au");
```

- Test audio simple

```
//
// Extension de HelloWorld pour restituer un son audio
// Suppose l'existence du fichier "sounds/cuckoo.au"
//
import java.awt.Graphics;
import java.applet.Applet;

public class HwAudio extends Applet {
     public void paint(Graphics g) {
      g.drawString("Audio Test", 25, 25);
     }
}
```

Boucle d'un clip audio

- Chargement d´un clip audio

```
AudioClip sound;
sound = getAudioClip(getDoumentBase(), "bark.au");
```

- Exécution d'un clip audio

```
sound.play();
sound.loop();
```

- Arrêt d'un clip audio

```
sound.stop()
```

Test simple de boucle

Voici un exemple de boucle automatique d'un clip audio :

```
//
// Extension de HelloWorld pour générer une boucle
     // d'une piste audio
// Suppose l'existence de "sounds/cuckoo.au"
//
import java.awt.Graphics;
import java.applet.*;
public class HwLoop extends Applet {
     AudioClip sound;
     public void init() {
      sound = getAudioClip(getDocumentBase(),
       "sounds/cuckoo.au");
     }
     public void paint(Graphics g) {
      g.drawString("Audio Test", 25, 25);
     }
     public void start () {
      sound.loop();
     }
     public void stop() {
      sound.stop();
     }
}
```

Lecture de paramètres

```
<applet code=configureMe.class width=100 height=100>
<param name=image value=duke.gif>
</applet>


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

public class DrawAny extends Applet {
  Image im;

  public void init() {
    URL myPage = getDocumentBAse();
    String imageName = getParameter("image");
    im = getImage (myPage, imageName);
  }
}


int speed = Integer.parseInt (getParameter ("speed"));
```

Exercice

une simple applet: SimpleApplet.java
Cette applet affiche un rectangle plein de couleur rouge en 50, 10
larg = 200 haut = 50.
et un texte en vert = "BONJOUR DE MON APPLET"
Consulter la documentation pour pouvoir utiliser les méthodes
fillRect() setColor() et drawString().
- Compilation
% javac SimpleApplet.java
- Générer un fichier HTML associé : SimpleApplet.html
En prevoyant une largeur de 300 et une hauteur de 100 pour la fenêtre
d'affichage.
- Lancer l'exécution de votre applet:
% appletviewer SimpleApplet.html

Exercice

Illustration du passage de paramètres:

Ecrire une applet AutreApplet qui reçoit en paramètres:

d. La référence d'une image à charger.

e. La référence d'un son à charger.

Le son doit démarrer à l'affichage de l'applet.

S'arrêter si on change de page et reprendre si on revient sur la page.

Chapitre 6

Exceptions dans le langage Java

Exceptions

- Implémentation des exceptions

```
                        java.lang.Object
                              |
                        java.lang.Throwable
                          |          |
java.lang.Error----+      |          +----java.lang.Exception
```

- Exceptions courantes

```
int i = 12 / 0;
```

```
Image im [] = new Image [4];
System.out.println(im[0].toString());
```

- Autres Exceptions

```
ClassCastException . .
NegativeArraySizeException . .
ArrayIndexOutOfBoundsException . .
```

Exceptions .

- Exemple d´exception

```
class HelloWorld {
     public static void main (String args[]) {
      int i = 0;
      String greetings [] = {
       "Hello world!",
       "No, I mean it!",
       "HELLO WORLD!!"
      };
      while (i < 4) {
       System.out.println (greetings[i]);
       i++;
      }
     }
}
```

Exceptions . .

- Importance des exceptions

```
$ java HelloWorld
Hello world!
No, I mean it!
HELLO WORLD!!
java.lang.ArrayIndexOutOfBoundsException: 3
        at HelloWorld.main(HelloWorld.java:12)
```

Traitement d´exception

- Instructions `try` et `catch`

```
try {
     // code pouvant lever une exception particulière
     . . .
} catch (MyExceptionType e) {
     // code à exécuter si MyExceptionType est levée
     . . .
}
```

- Instruction `finally`

```
try {
     startFaucet();
     waterLawn();
}
catch (MyExceptionType e) {
     // code à exécuter si MyExceptionType est levée
     . . .
}
finally {
     stopFaucet();
}
```

Traitement d´exception .

- Exemple

```
public static void main (String args[]) {
 int i = 0;
 String greetings [] = {
  "Hello world!",
  "No, I mean it!",
  "HELLO WORLD!!"
 };
 while (i < 4) {
  try {
   System.out.println (greetings[i]);
  } catch (ArrayIndexOutOfBoundsException e){
   System.out.println("Re-setting Index Value");
   i = -1;
  } catch (Exception e) {
   System.out.println(e.toString());
  } finally {
   System.out.println("This is always printed");
  }
  i++;
 } // end while()
} // end main()
```

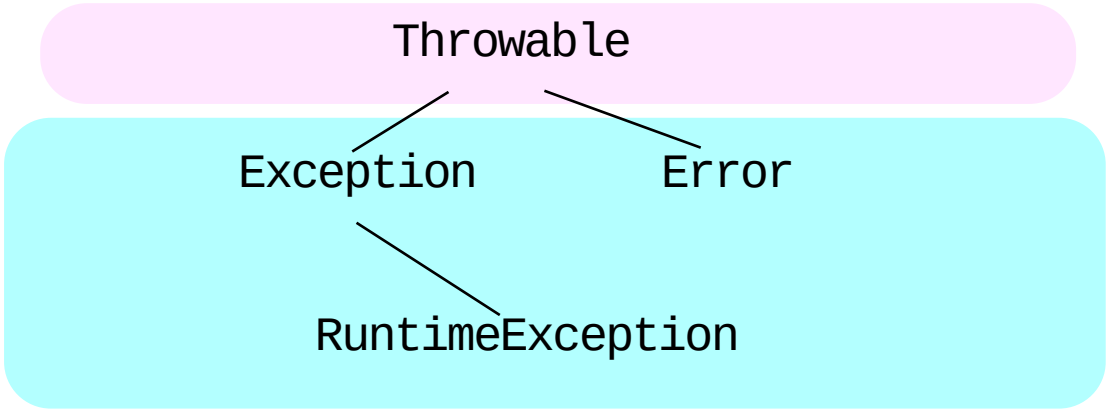
Traitement d´exception . .

- Exemple : impression résultante de l’exécution

```
Hello world!
This is always printed
No, I mean it!
This is always printed
HELLO WORLD!!
This is always printed
Re-setting Index Value
This is always printed
```

Trois catégories d´exceptions

- En Java les exceptions sont des instances de classe dont les principales sont celles ci-dessous rangées en arborescence d'héritage.



Règle : Déclarer ou traiter

- Java exige que si une méthode est susceptible de générer une exception (classe `Exception`), elle doit clairement indiquer quelle action doit être entreprise si le problème survient. Cette action est :
  - soit déclarer que la méthode lève cette exception
  - soit la traiter
- Cette règle ne s'applique pas pour les exceptions instances de sous classes `Error` ou `RuntimeException`

Déclarer ou traiter : syntaxe

- Soit

```
leve () throws MonException{... throw new MonException(
... );... }
```

  - avec

```
class MonException extends Exception{ . . . }
```

- Soit

```
lance() ... { ... leve(); ... }
```

Dans ce cas lance() doit être écrite de l'une des deux manières :

1)
```
lance() { ... try{ ... leve(); ... } ... }
                catch(MonException e){ ... }
```

2)
```
lance() throws MonException { ... leve(); ... }
```

Exemple de manipulation des exceptions

- Déclaration

```
public class DelaiDeGardeDepasse extends Exception{ }
```

- Lancement d'une exception créée

```
throw new DelaiDeGardeDepasse( );
```

- Utilisation

```
public void connectMe(String serverName) throws DelaiDeGardeDepasse {
     int success;
     int portToConnect = 80;
     success = open(serverName, portToConnect);
     if (success == -1) {
      throw new DelaiDeGardeDepasse( );
     }
}
```

Exemple de manipulation des exceptions .

- Traiter votre exception, en utilisant `catch` :

```
public void findServer() {
          . . .
          try {
           connectMe(defaultServer);
          } catch(DelaiDeGardeDepasse e) {
           g.drawString("Server timed out, trying alternate", 5, 5);
           try {
                connectMe(alternateServer);
           } catch (DelaiDeGardeDepasse e1) {
            g.drawString("No server currently available", 5, 5);
           }
          }
          . . .
```

Exercice

Illustration du traitement des exceptions:

▼ **Créer un programme ExcepProg.java:**

Qui reçoit en argument le nom d'un fichier.

Ce programme appelle sous surveillance une méthode
trait_fic(String nomfic) qui va traiter le fichier.

Cette méthode doit vérifier que le fichier existe et qu'il correspond à un répertoire.

Si le fichier n'existe pas elle doit lever une exception personnelle "NonRepException".

Si le fichier n'est pas un répertoire elle doit lever une exception personnelle "NonFicException".

Créer une classe pour chacune de ces exceptions:

On utilisera un objet de la classe File.

Exercice (suite)

On utilisera les methodes de cette classe :

```
exists()
isDirectory()
list()
```

Cette dernière méthode permet de récupérer sous la forme d'un tableau de String la liste des fichiers de ce répertoire.

Note – Il est conseillé de donner le chemin d'accès au fichier au format unix (Reconnu également sous NT). On peut utiliser un chemin relatif. Dans ce cas il est relatif au chemin courant.

Chapitre 7

Fonctions évoluées du langage

Package

- Définition
- Diversité
- Organisation
- Packages inclus dans le JDK

```
java.lang: classes fondamentales
java.util : structures de données fondamentales
java.io : entrées-sorties
java.net : réseau
java.awt : objets graphiques
java.awt.event : traitement des événements graphiques
java.applet : traitement des  Applets
. . .
Voir la documentation du JDK.
```

Package

- L'instruction `import`
  - Première utilisation :

```
import java.util.Vector;
```

en début de fichier permet par la suite d'écrire :

```
Vector monVector;
```

au lieu de :

```
java.util.Vector monVector;
```

  - Deuxième utilisation :

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

en début de fichier permet d'utiliser toutes classes ou interfaces du package `java.awt` par son nom :

```
import java.awt.*;
  . . .
  public void paint(Graphics g) {
  . . .
  }
```

Package

- L'instruction package
  - Création de package
  - Par exemple :

```
package figure;
class Cercle {
   private java.awt.Point  origine;
}
Phase de compilation :
On utilise l'option -d de javac :
javac -d $HOME/. . ./<nom_package> <classe>.java
Par exemple :
javac -d $HOME/mes_classes/figure Cercle.java
```

Classes abstraites

```
public abstract class Drawing {
    public abstract void drawDot (int x, int y);
    public void drawLine(int x1, int y1,
                         int x2, int y2) {
     // dessiner avec la méthode drawDot() de façon
        // répétée.
    }
}
```

Interfaces

```
public class ReactToButton
    implements ActionListener {
      public void actionPerformed(ActionEvent e) {
      // réagir à l'événement
      }
    }
```

Classes internes imbriquées

- Exemple:

```
//la classe Pile gère une pile d'objets
public class Pile {
  private Object[] tableExtensible ;
  private int  sommetDePile ;
     ....
     class ParcoursPile implements
                java.util.Enumeration {
      int index = sommetDePile ;
      public boolean hasMoreElements() {
       return index >= 0 ;
      }
      public Object nextElement() {
       return tableExtensible[index--] ;
      }
     }// ParcoursPile

     public java.util.Enumeration elements() {
      return new ParcoursPile() ;
     }
     ...
}
```

Classes anonymes

```
// exemple dérivé d´AWT création d'un bouton fermant une fenêtre
class Fenetre extends Frame {
  ....
     Button quit = new Button("QUIT") ;
     add(quit, "South");
     quit.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
                                dispose() ;}
     });
}

class MonVeilleur implements ActionListener {
     Frame myFrame ;
     MonVeilleur( Frame thisFrame) {
      myFrame=thisFrame ;
     }
     public void actionPerformed(Actionevent e) {
      myFrame.dispose() ;
     }
} //monVeilleur


Invocation:

// code de l'appel
quit.addActionListener(new monVeilleur(this)) ;
```

Surcharge des méthodes

```
public void print(int i)

public void print(float f)

public void print(String s)
```

surcharge vs.polymorphisme

	Surcharge	**Polymorphisme**
Héritage	nul besoin	nécessite une arborescence de classes ou d'interfaces
signature des méthodes	doivent différer	doivent-être les mêmes
résolu à	la compilation	l'exécution

Passage par valeur

```
public class PassTest  {

float ptValue;

     public static void main (String args[]) {

      String str;
      int val;

      // Créer une instance de la classe
      PassTest pt = new PassTest ();

      // Affecter l'entier
      val = 11;

      // Tenter de le modifier
      pt.changedInt ( val );

      // Quelle est la valeur actuelle ?
      System.out.println ("Int value is:"+val);

      // Affecter la chaîne
      str = new String ("hello");
```

```
        // Tenter de la modifier
        pt.changeStr (str);


        // Quelle est la valeur actuelle ?
        System.out.println ("Str value is:"+str)
        // Maintenant définir ptValue
        pt.ptValue = 101f;

        // Changer la valeur du flottant
        // via la référence à l'objet
        pt.changeObjValue (pt);

        // Quelle est la valeur actuelle ?
        Ssytem.out.println ( "Valeur actuelle :
         " + pt.ptValue);
      }

      // Méthode pour changer les valeurs actuelles
      public void changeInt (int value) {
        value = 55;
      }

      public void changeStr (String value) {
        value = new String ("differente");
      }

      public void changeObjValue (PassTest ref) {
        ref.ptValue = 99f;
      }
}
```

Variable de classe

```
public class Count {
    private int serialNumber;
    private static int counter = 0;
    public Count () {
       counter++;
       serialNumber = counter;
    }
}
```

Variable de classe

- Exemple:

```
public class StaticVar {
    public static int number;
}

public class OtherClass {
    public void method() {
        int x = StaticVar.number;
    }
}
```

Méthode de classe

```
public class GeneralFunction {
    public static int addUp( int x, int y) {
        return x + y;
    }
}
public class UseGeneral {
    public void method() {
        int a = 9;
        int b = 10;
        int c = GeneralFunction.addUp(a , b);
        System.out.println("addUp() gives" + c);
    }
}
```

Méthode de classe

```
public class Wrong {
    int x;
    public static void main(String args[]) {
        x = 9;  // ERREUR DE COMPILATION !
    }
}
```

Opérateur *instanceof*

```
public class Employee extends Object
public class Manager extends Employee
public class Contractor extends Employee
```

```
public void method(Employee e) {
   if (e instanceof Manager) {
      // l'appeler 'Monsieur'
   }
   else if (e instanceof Contractor) {
      // garder les secrets de la société
   }
   else {
      // employé de plein droit, parler librement
   }
}
```

Contrainte temporaire de type : Cast

```
public class Employee {
   String name;
   Date hireDate;
   Date dateOfBirth;
   String jobTitle;
   int grade;
}

public class Manager extends Employee {
   String department;
   Employee [] subordinates;
}
```

****

```
public void method(Employee e) {
   if (e instanceof Manager) {
      Manager m = (Manager)e;
      System.out.println("C'est le manager de " +
                         m.department);
   }
   // reste de l'opération
}
```

Référence this

- Premier cas :
  - **passage de la référence de l´objet courant à la méthode** :

```
// les méthodes drawImage() de la classe Graphics
drawImage( . . . , this);
....
// doivent informer régulièrement l'objet courant
// de l'état d'avancement du chargement de l'image.
```

  - **Levée d'une ambiguité**:

```
class Client {
     String nom ;
     String clientID ;
     ....
     Client(String nom, String ID) {
        this.nom = nom ;
      clientID = ID ; // this implicite
     }
     ....
}
```

Appel de constructeur par this()

- Deuxième cas :
  - **Appel d'un constructeur de la même classe** :

```
public MaClasse(int a, int b) {
     ...
}

public MaClasse(int c) {
     this(c, 0);
}

public MaClasse() {
     this(10);
}
```

Référence super

- Premier cas :

```
class ClientRegulier extends Client {
     double ristourne;
     public String tostring() {
      return super.toString() + "ristourne=" + ristourne;
     . . .
     }
}
```

```
class Client {

     String nom ;
     String clientID ;
     public String toString() {
      return "nom=" + nom + " id=" + clientID;
     }
}
```

Référence super (suite)

- Deuxième cas :
  - Appel d'un constructeur de la classe de base :

```
class ClientRegulier extends Client {
   ClientRegulier( String le_nom, String le_id) {
      super ( le_nom, le_id);
      . . .
   }
}
```

Classes et méthodes finales

- Classe : dans le paquetage `java.lang` :

```
public final class Integer extends
Number {
   . . .
}
```

- Méthodes

```
// dans java.awt.Window
public final String getWarningString() {
   ....
}
```

Champ de donnée `final`

```
// dans java.lang.Math
public static final double PI =
                    3.14159265358979323846;
// dans java.awt.Color
public static final Color white
                    = new Color(255, 255, 255);
// dans un champ d'instance
private final Date dateCreationInstance = new Date() ;

// java 1.1 "final" comme controle
                        d'initialisation unique
public class Client {
     public final String clientID;
     ...
     public Client( String nom, String ID) {
      clientID = ID ;
      ....
     }
}
```

Contrôle d´accès

Modificateurs	Utilisé sur	Signification

`abstract`	`class`
	
	`interface`
	méthode
		Contient des méthodes non implémentées (seulement déclarée) et ne peut pas être instanciée.
		Ils sont `abstract`, le modificateur est optionnel .
		Elle est seulement déclarée (elle n'a pas de corps). Une classe contenant une méthode `abstract` doit être aussi déclarée `abstract`.

`final`	`class` donnée méthode variable	On ne peut pas en hériter. Le champ ne doit être modifié. Elle ne doit pas être spécifier (i.e. ne peut être réécrite dans une classe dérivée). En Java 1.1, ils ne peuvent être modifiés.
`native`	méthode	Est une déclaration pour une méthode importée d'un autre langage.

pas de modificateur	`class`
	`interface`
	membre
		Est seulement accessible dans son package.
		Est seulement accesible dans son package.
		Est seulement accesible dans son package.

`private`	membre	Est seulement accessible dans la classe qui le définit.

Contrôle d´accès (suite)

`protected`	membre	Est seulement accessible dans le package dans lequel il est définit ainsi que dans les sous-classes.

`public`	`class`
	`interface`
	membre
		Est accessible partout où son package l´est.
		Est accessible partout où son package l´est.
		Est accessible partout où sa classe est accessible.

`static`	`class` donnée bloc méthode	En Java1.1, elle est non imbriquée. Est un champ de classe. Un seul exemplaire quel que soit le nombre d'instance de la classe. Est exécuté une seule fois lorsque la classe est chargée et non à la création de chaque instance. Est une méthode de classe. Un seul exemplaire quel que soit le nombre d'instance.

Contrôle d´accès (suite)

Modificateurs	Utilisé sur	Désignation
`synchronized`	méthode	Pour une méthode `static` un verrou est affecté à la classe (aux données `static` de la classe). Pour une méthode `non-static` on affecte un verrou à l´ instance.
`transient`	donnée	Ne fait pas partie de l´état persistant de l´objet, ne sera pas être sérialisé avec cet objet.
`volatile`	donnée	On indique au compilateur de ne pas réaliser d'optimisation d'accès. La valeur de ce champ peut-être modifié par d´autres threads.

Chapitre 8

Interface graphique utilisateur

Package java.awt

- Instruction Import

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

Interfaces graphiques utilisateurs

- Conteneurs et éléments
- Positionnement de composants
- Taille de composant

```
import java.awt.*;
public class ExGui extends Frame{
     private Button b1;
     private Button b2;

     public ExGui () {
      super ("GUIExample");
      setLayout(new FlowLayout());
      b1 = new Button(“Press Me”);
      b2 = new Button(“Don’t press Me”);
      add(b1);
      add(b2);
     }
     public static void main(String[] args) {
      ExGui that = new ExGui();
      that.pack();
      that.setVisible(true);
     }
}
```

Exemple simple

Méthode main()

super(“GUI Example”)

setLayout(new FlowLayout())

new Button(“Press Me”)

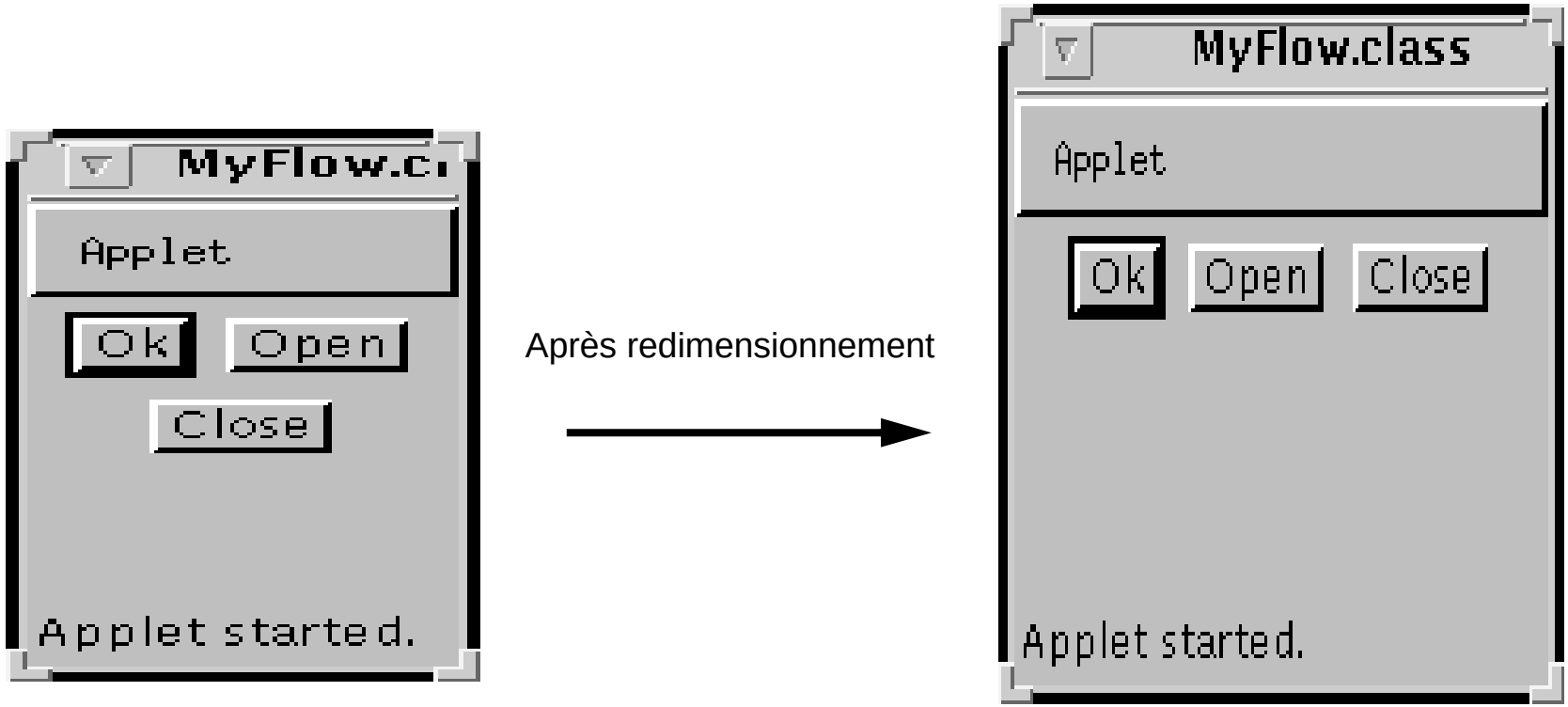
add(b1)

that.pack()

that.setVisible(true)

Gestionnaire de présentration

- Gestionnaire de présentation de FlowLayout



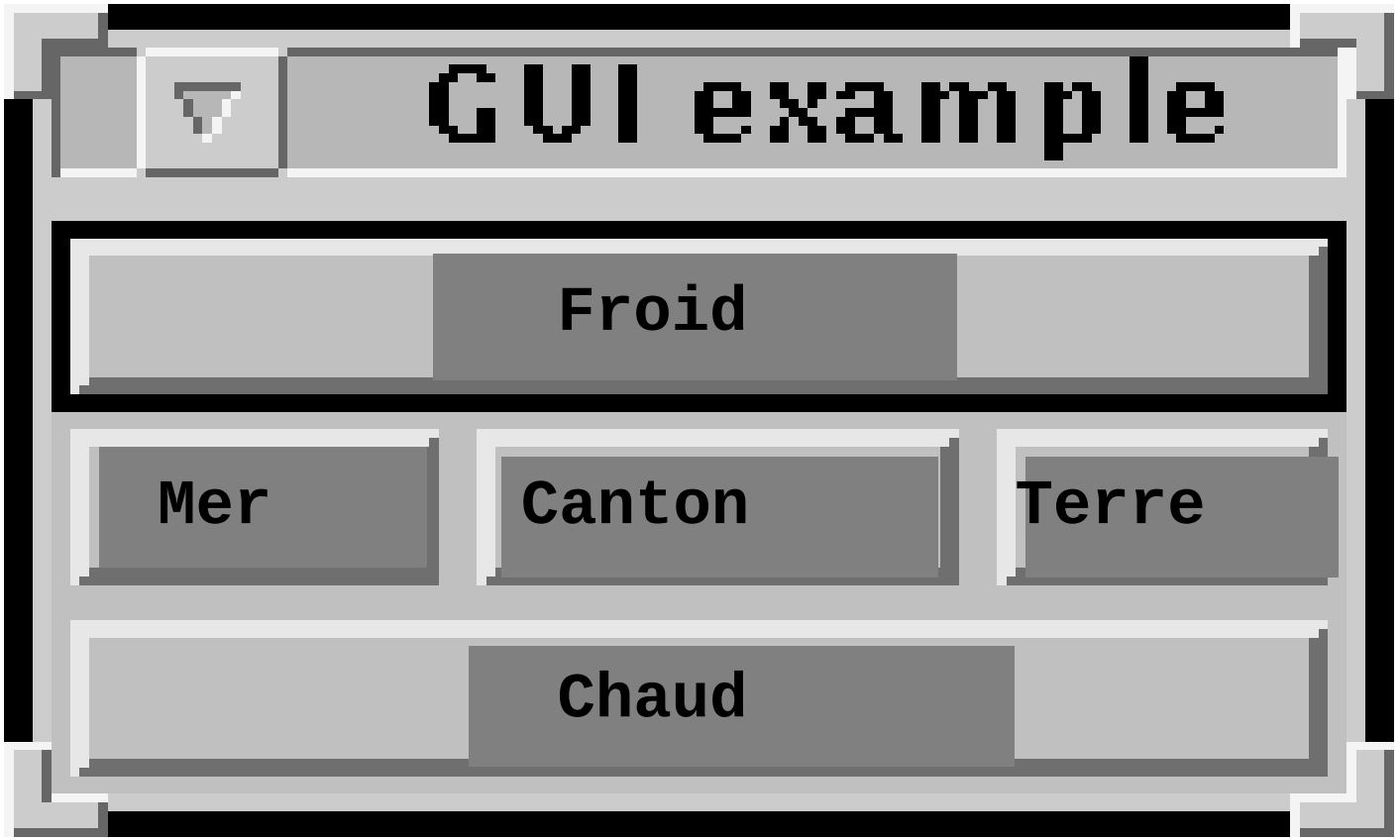
Gestionnaires de présentration

- Gestionnaire de présentation de BorderLayout

```
import java.awt.*;
public class ExGui2 {
      private Frame f;
      private Button bn, bs, bw, be, bc;
      public static void main(String args[]) {
       ExGui2 that = new ExGui2();
       that.go();
      }
      public void go() {
       f = new Frame("GUI example");
       bn = new Button("Froid");
       bs = new Button("Chaud");
       bw = new Button("Mer");
       be = new Button("Terre");
       bc = new Button("Canton");
       f.add("North", bn);
       f.add("South", bs);
       f.add("West", bw);
       f.add("East", be);
       f.add("Center", bc);
       f.pack();
       f.setVisible(true);
       }
      }
}
```

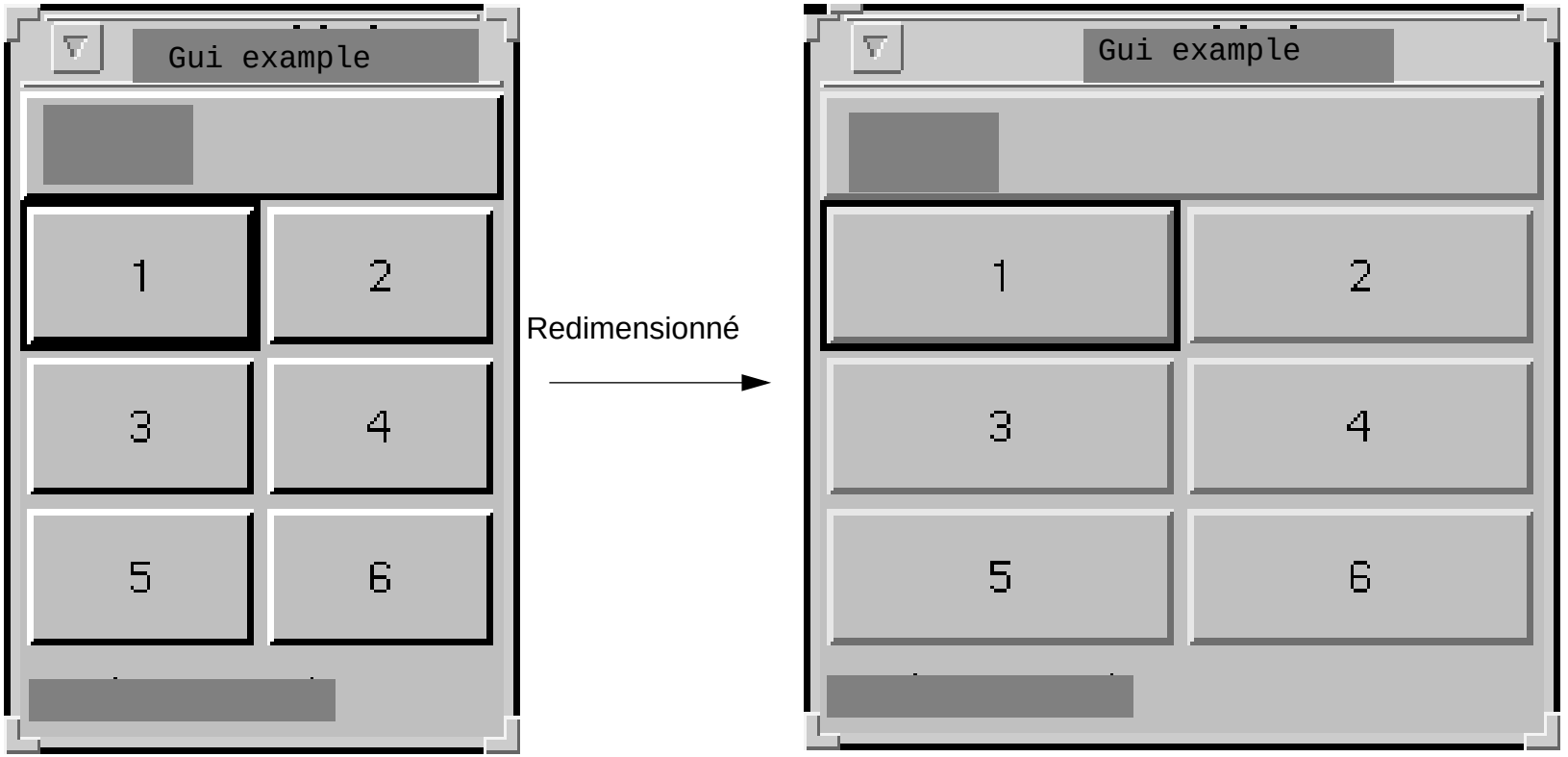
Gestionnaires de présentation

- Gestionnaire de présentation `BorderLayout`



Gestionnaire de représentation

- Gestionnaire de présentation `GridLayout`



Conteneurs

- Frame
- Panel
- Création de Panel et de présentation complexes

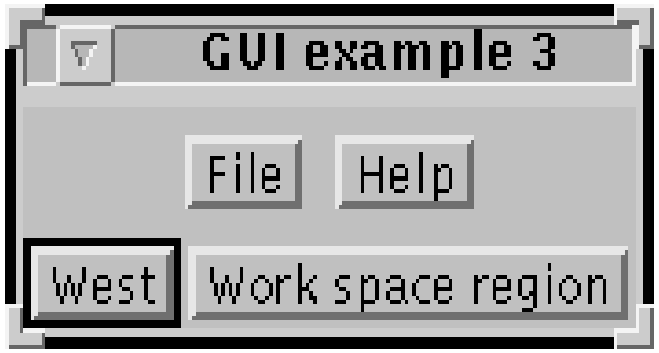
```
import java.awt.*;
public class ExGui3 {
   private Frame f;
   private Panel p;
   private Button bw, bc;
   private Button bfile, bhelp;

   public static void main(String args[]) {
      ExGui3 that = new ExGui3();
      that.go();
   }
```

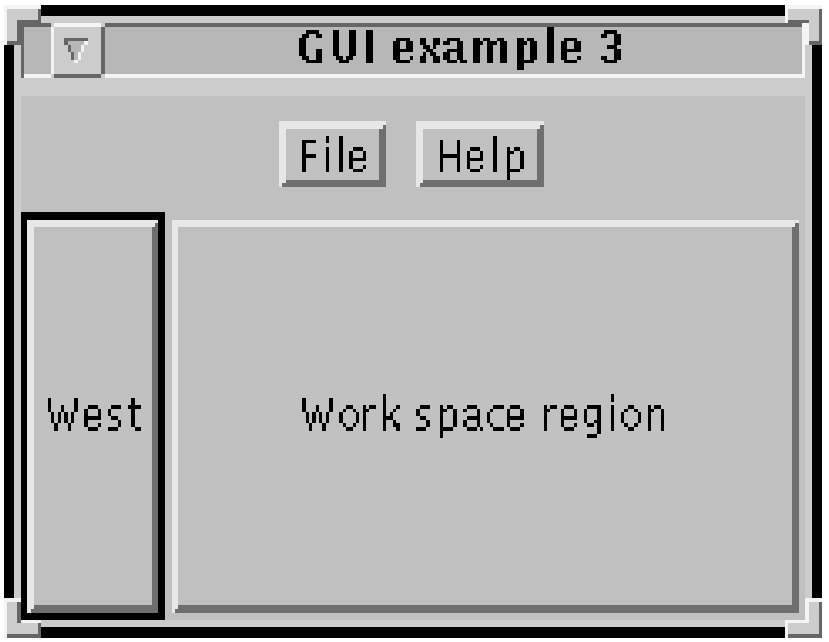
Conteneurs .

```
   public void go() {
      f = new Frame(“GUI example 3”);
      bw = new Button(“West”);
      bc = new Button(“Work space region”);
      f.add(“West”, bw);
      f.add(“Center”, bc);
      p = new Panel();
      f.add(“North”, p);
      bfile = new Button(“File”);
      bhelp = new Button(“Help”);
      p.add(bfile);
      p.add(bhelp);
      f.pack();
      f.setVisible(true);
   }
}
```

Conteneurs . .

- Création de Panel et de présentation complexes





Exercice

Écrire un programme qui fait afficher :

remarque : il n'y a que des boutons poussoirs.

S'en servir pour construire ensuite :

où un objet de la classe `TextField` a été ajouté en haut (euh au Nord !!) de la Frame. On pourra utiliser un Panel pour regrouper tous les boutons.

remarque : dans un emplacement d'un copnteneur géré par un BorderLayout, on ne peut mettre qu'un seul composant graphique (qui lui, peut en contenir plusieurs).

Chapitre 9

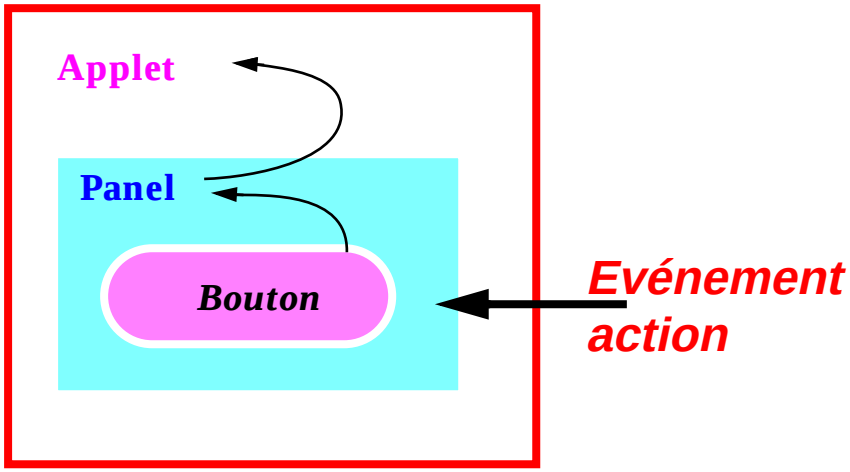
Modèle d'événements AWT

Qu´est-ce-qu´un événement?

- Evénements sources
  - ActionCommand : nom de commande associé à l'action.
  - modifiers : tous modificateurs mobilisés au cours de l'action
- Traitements d'événements
- Mode de traitement des événements
  - Modèle JDK1.0 et JDK1.1

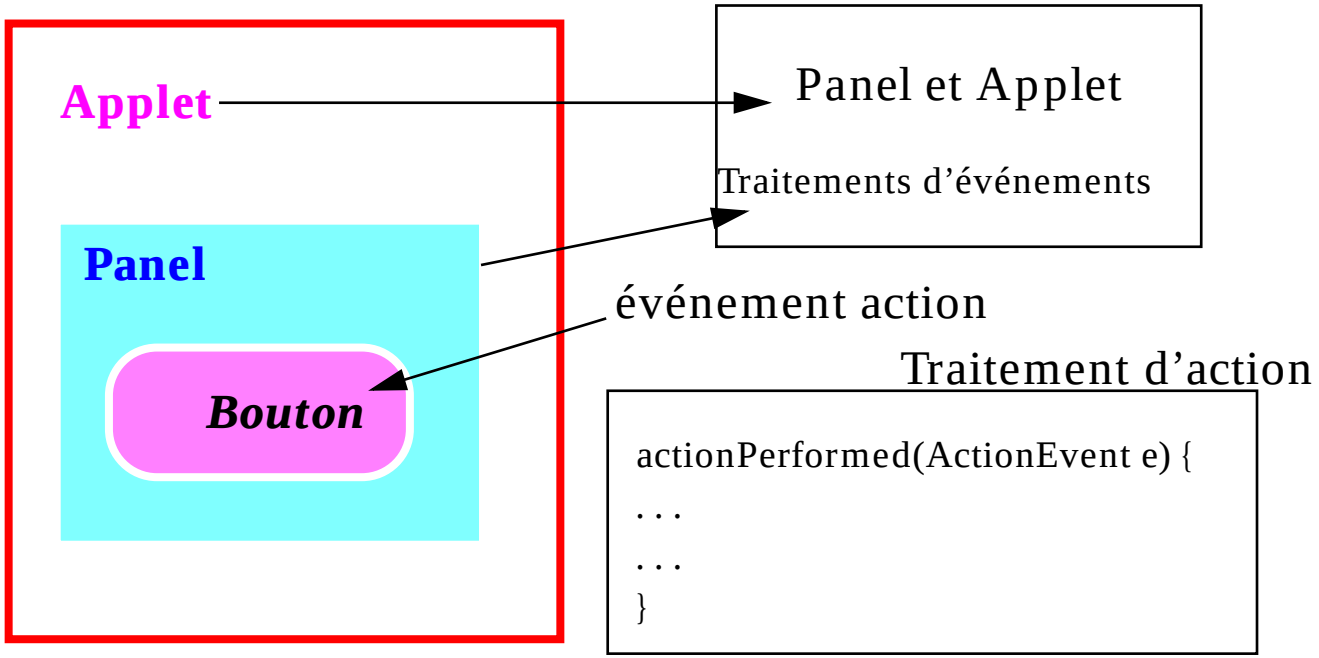
Comparaison entre les modèles d'évènements

- Modèle hiérarchisé (JDK 1.0)



Comparaison entre les modèles d'évènements . .

- Modèle de délégation (JDK 1.1)



Comparaison entre les modèles d'événements . . .

- Modèle de délégation (JDK 1.1(suite))

```
import java.awt.*;
public class TestButton {
   public static void main (String args[]){
      Frame f = new Frame ("Test");
      Button b = new Button("Press Me!");
      b.addActionListener(new ButtonHandler());
      f.add("Center", b);
      f.pack();
      f.setVisible(true);
   }
}
import java.awt.event.*;
public class ButtonHandler implements
                                ActionListener{
   public void actionPerformed(ActionEvent e) {
      System.out.println("Action occured");
   }
}
```

Comparaison entre les modèles d'événements . . . .

- Méthode simple et existante
- Implémentation d'un récepteur d'événement par objet
- Classification les actions d'événements (filtres)
- Modèle par délégation est le plus adapté
- Support de JavaBeans
- Passage de 1.0 à 1.1
- Pas de mixité dans le traitement des événements

Comportement de l'interface graphique utilisateur Java

Catégorie	Nom de l'interface	Méthodes
Action	ActionListener	actionPerformed(ActionEvent)
Item	ItemListener	ItemStateChanged(ItemEvent)
Mouse	MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
Mouse	MouseListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mouseClicked(MouseEvent)
Key	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent)keyTyped(KeyEvent)
Focus	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
Adjustment	AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)

Comportement de l'interface graphique utilisateur Java

Component	ComponentListener	componentMoved(ComponentEvent) componentHidden(ComponentEvent) componentResize(ComponentEvent) componentShown(ComponentEvent)
Window	WindowListener	windowClosing(WindowEvent) windowOpened(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)
Container	ContainerListener	componentAdded(ContainerEvent) componentremoved(ContainerEvent)
Text	TextListener	textValueChanged(TextEvent)

Comportement de l'interface graphique utilisateur Java

- Catégories d'événements
  - Tableau

Category	Interface Name	Methods
Action	ActionListener	actionPerformed(ActionEvent)
Item	ItemListener	ItemStateCanged(ItemEvent)
Mouse Motion	MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
Mouse Button	MouseListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mouseClicked(MouseEvent)
Key	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)

Focus	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
Adjustement	AdjustmentListener	adjustementValueChanged(AdjustementEvent)
Component	ComponentListener	componentMoved(ComponentEvent) componentHidden(ComponentEvent) componentResize(ComponentEvent) componentShown(ComponentEvent)
Window	WindowListener	windowClosing(WindowEvent) windowOpened(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)
Container	ContainerListener	componentAdded(ContainerEvent) componentremoved(ContainerEvent)
Text	TextListener	textValueChanged(TextEvent)

Comportement de l'interface graphique utilisateur Java

- Evénements générés par composants AWT

Composant AWT	Action	adjust	component	container	focus	item	key	mouse	mouse motion	text	window
Button	●		●		●		●	●	●		
Canvas			●		●		●	●	●		
Checkbox			●		●	●	●	●	●		
CheckboxMenuItem						●					
Choice			●		●	●	●	●	●		
Component			●		●		●	●	●		
Container			●	●	●		●	●	●		
Dialog			●	●	●		●	●	●		●
Frame			●	●	●		●	●	●		●

Composant AWT	Action	adjust	component	container	focus	item	key	mouse	mouse motion	text	window
Label			●		●		●	●	●		
List	●		●		●	●	●	●	●		
MenuItem	●										
Panel			●	●	●		●	●	●		
Scrollbar		●	●		●		●	●	●		
ScrollPane			●	●	●		●	●	●		
TextArea			●		●		●	●	●	●	
TextComponent			●		●		●	●	●	●	
TextField	●		●		●		●	●	●	●	
Window			●	●	●		●	●	●		●

Comportement de l'interface graphique utilisateur Java

- Exemple plus complexe :

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

public class TwoListen implements
       MouseMotionListener, MouseListener {
     private Frame f;
     private TextField tf;

     public static void main ( String args[]) {
      TwoListen that = new TwoListen();
      that.go();
     }
Comportement de l'interface graphique utilisateur Java
public void go() {
     f = new Frame("Exemple Deux écouteurs");
     f.add ("North", new Label ("Cliquer et déplacer la souris"));
     tf = new TextField (30);
     f.add ("South", tf);

     f.addMouseMotionListener (this);
     f.addMouseListener (this);
```

```
        f.setSize(300, 200);
        f.setVisible(true);
        //f.requestFocus90;
}
//These are MouseMotionListener events
public void mouseDragged (MouseEvent e) {
        String s =
          "Mouse dragging: X = " +e.getX() + " Y = " +e.getY();
        tf.setText (s);
}
public void mouseMoved (MouseEvent e) {
}
//These are MouseListener events
public void mouseClicked (MouseEvent e) {
}
public void mouseEntered (MouseEvent e){
        String s = "La souris est entrée";
        tf.setText (s);
}
public void mouseExited (MouseEvent e) {
        String s = "La souris a quitté l'immeuble";
        tf.setText (s);
}
public void mousePressed (MouseEvent e) {
}
public void mouseReleased (MouseEvent e) {
}
}
```

Comportement de l'interface graphique utilisateur Java . . . . .

- Déclaration de plusieurs interfaces

```
implements MouseMotionListener, MouseListener
```

- Ecoute de plusieurs sources

```
f.addmouseListener(this);
f.addMouseMotionListener(this);
```

- Obtention d'informations sur un événement
- Récepteurs multiples

Adaptateurs d´événements

- `mouseClicked (Mouseevent)`
- `mouseEntered (MouseEvent)`
- mouseExited (MouseEvent)
- mousePressed (MouseEvent)
- mouseReleased(MouseEvent)
- Par exemple :

```
import java.awt.*;
import.awt.event.*;
public class MouseClickHandler extends MouseAdapter {

  //Nous avons seulement besoin du traitement mouseClick,
  //nous utilisons donc l'adaptateur pour ne pas avoir à
  //écrire toutes les méthodes de traitement d'événement

  public void mouseClicked (MouseEvent e) {
    //Faire quelque chose avec le clic de la souris . . .
  }
}
```

Test simple de la souris

Le programme ci-dessous affiche l'emplacement du clic de la souris au sein de l'applet.

```
//Extension de HelloWorld pour surveiller les entrées
     //souris
//"HelloWorld!" est réimprimé à l'emplacement du clic
     // de la souris.
//

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

public class HwMouse extends Applet
     implements MouseListener {

     int mouseX=25;
     int mouseY=25;

     //Enregistrement de l'événement mousePressed
     public void init() {
      addMouseListener (this);
     }

     public void paint(Graphics g) {
      g.drawString("Hello World!", mouseX, mouseY);
```

```
        }

        public void mousePressed(MouseEvent evt) {
         mouseX = evt.getX();
         mouseY = evt.getY();
         repaint();
        }

        //Nous n'utilisons pas les autres événements souris
        public void mouseClicked (MouseEvent e) {}
        public void mouseEntered (MouseEvent e) {}
        public void mouseExited  (MouseEvent e) {}
        public void mouseReleased(MouseEvent e) {}

}
```

Exercice

Reprendre l'interface :

Lorsqu'on clique sur chaque bouton "à chiffre", le chiffre du bouton est affiché dans le `TextField`.

Lorsqu'on clique sur le bouton Reset tout est effacé dans le `TextField`.

Lorsqu'on clique sur le bouton Bis les chiffres déjà saisis sont réaffichés.

Chapitre 10

Librairie de composants AWT

Fonctions de AWT

- Button


```
public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("Sample") {
        . . .
        }
        if (e.getSource() == b) {
        . . .
        }
}
```

Fonctions de AWT

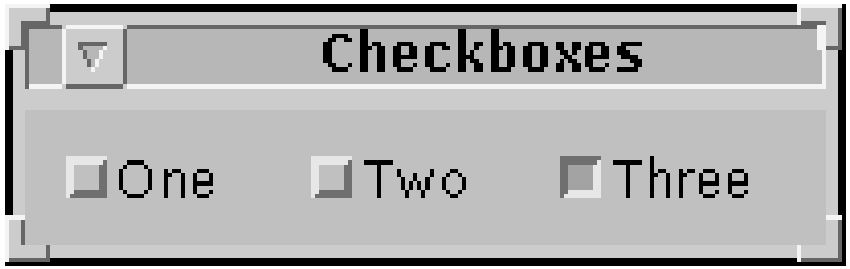
- Button

Autre méthode en indiquant directement les actions associées au bouton :

```
b.addActionListener( new ActionListener() {
     public void actionPerformed(ActionEvent e) {
      // traitement de l'action . .
     }
} );
```

Checkbox

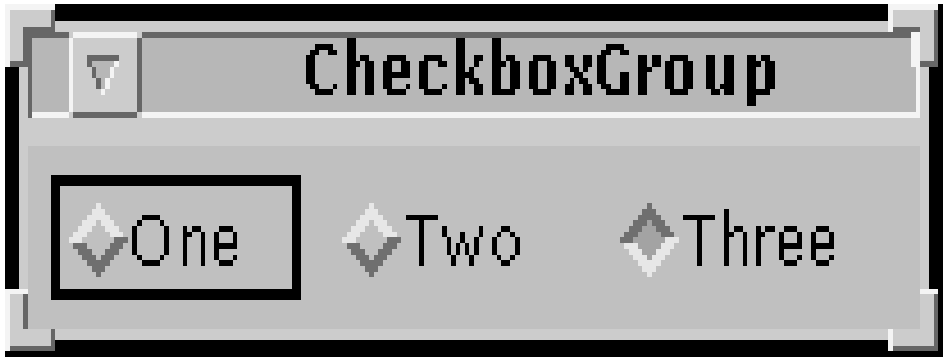
```
Checkbox one = new Checkbox("One", false);
Checkbox two = new Checkbox("Two", false);
Checkbox three = new Checkbox("Three", true);
add(one); add(two); add(three);
one.addItemListener(new Handler());
two.addItemListener(new Handler());
three.addItemListener(new Handler());
```




```
class Handler implements ItemListener {
     public void itemStateChanged(ItemEvent ev) {
      String state = “deselected”;
      if (ev.getStateChange() == ItemEvent.SELECTED){
       state = “selected”;
      }
      System.out.println(ev.getItem() + “ “ + state);
     }
}
```

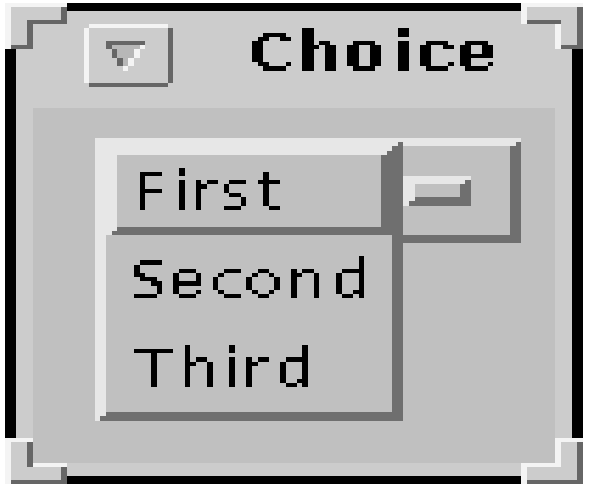
CheckboxGroup

```
CheckboxGroup cbg = new CheckboxGroup();
Checkbox one = new Checkbox("One", cbg, false);
Checkbox two = new Checkbox("Two", cbg, false);
Checkbox three = new Checkbox("Three", cbg, true);
add(one);
add(two);
add(three);
```

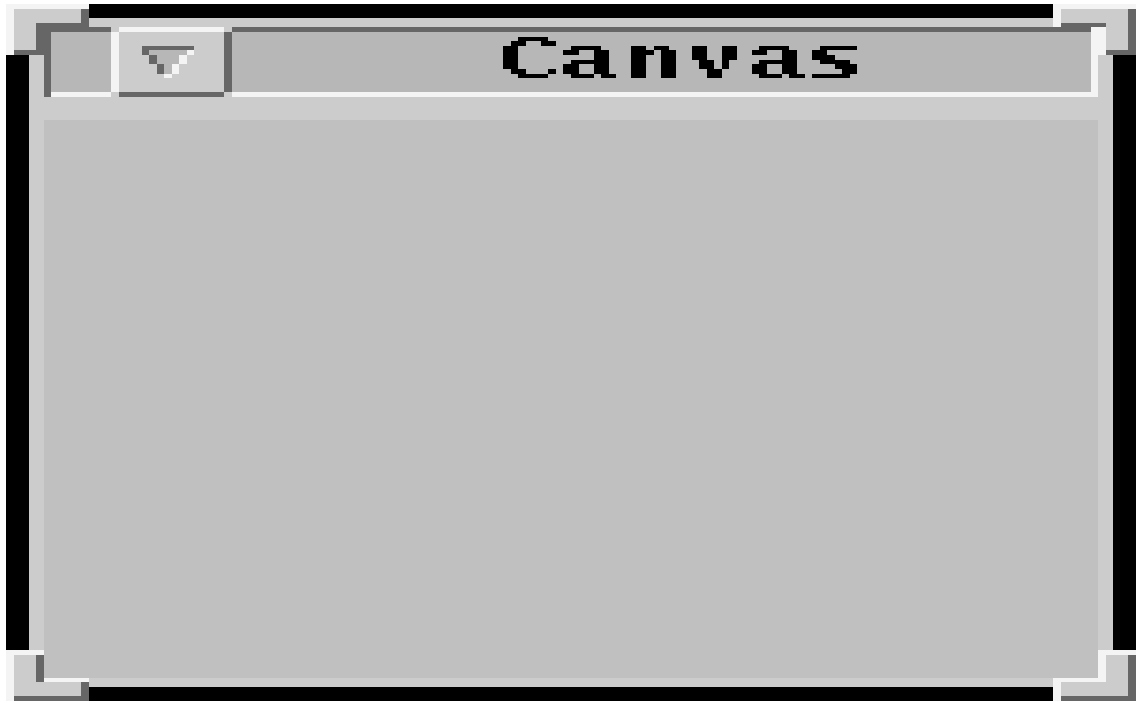


Choice

```
Choice c = new Choice();
c.addItem("First");
c.addItem("Second");
c.addItem("Third");
c.addItemListener(. . .);
```



Canvas



Canvas . .

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

public class Ca
     implements KeyListener, MouseListener {
     Canvas c;
     String s = "";

     public static void main(String args[]) {
      Frame f = new Frame("Canvas");
      Ca that = new Ca();
      that.c = new Canvas();
      that.c.setsize(100,100);
      that.c.addMouseListener(that);
      that.c.addKeyListener(that);
      f.add("Center", that.c);
      f.setSize(150, 150);
      f.setVisible(true);
     }
```

Canvas . . .

```
public void keyTyped(KeyEvent ev) {
      System.out.println("keyTyped");
      s += ev.getKeyChar();
      // Not a good drawing technique!!!
      c.getGraphics().drawString(s, 0, 20);
     }

     public void mouseClicked(MouseEvent ev) {
      System.out.println("mouseClicked");
      // force the focus onto the canvas
      c.requestFocus();
     }
     public void keyPressed(KeyEvent ev) {
      System.out.println("keyPressed");
     }

     public void keyReleased(KeyEvent ev) {
      System.out.println("keyReleased");
     }
```

Canvas . . .

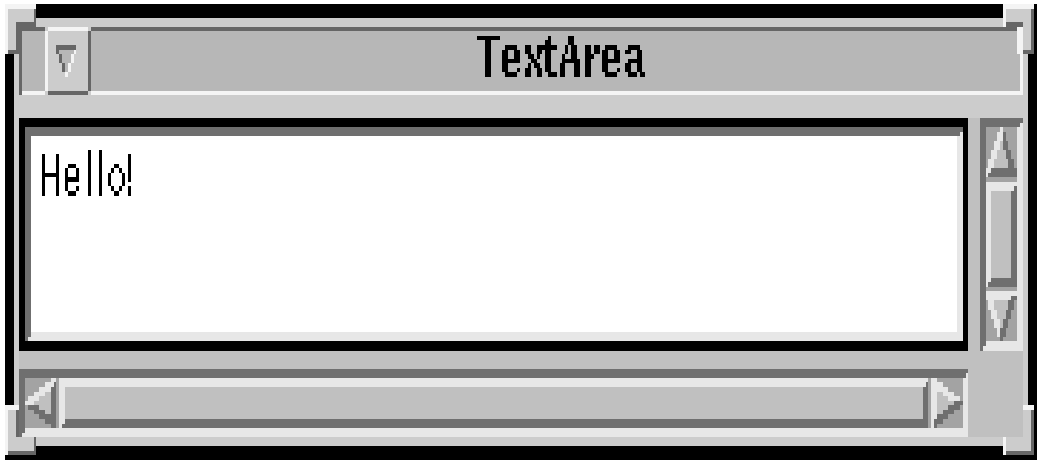
```
public void mousePressed(MouseEvent ev) {
     System.out.println("mousePressed");
   }

   public void mouseReleased(MouseEvent ev) {
     System.out.println("mouseReleased");
   }

   public void mouseEntered(MouseEvent ev) {
     System.out.println("mouseEntered");
   }

   public void mouseExited(MouseEvent ev) {
     System.out.println("mouseExited");
   }
}
```

Label

```
Label l = new Label(“Hello”);
add(l);
```

TextArea

```
TextArea t = new TextArea(“Hello!”, 4, 30);
add(t);
```



- Appui sur touche "Enter" provoque le passage à la ligne suivante

TextField

```
TextField f = new TextField(“Single line”, 30);
add(f);
```

- Appui sur touche "Enter" génère un événement de la classe `ActionEvent`.

TextComponent

- Classe de base TextArea et TextField
- Contient les méthodes fondamentales et communes pour ces classes : `getText()`, `setText()`, . . .
- La méthode `setEditable(boolean)` rend un TextComponent éditable ou non. Dans le cas non éditable, seul le programme (et non l'utilisateur) peut changer sa zone de texte.

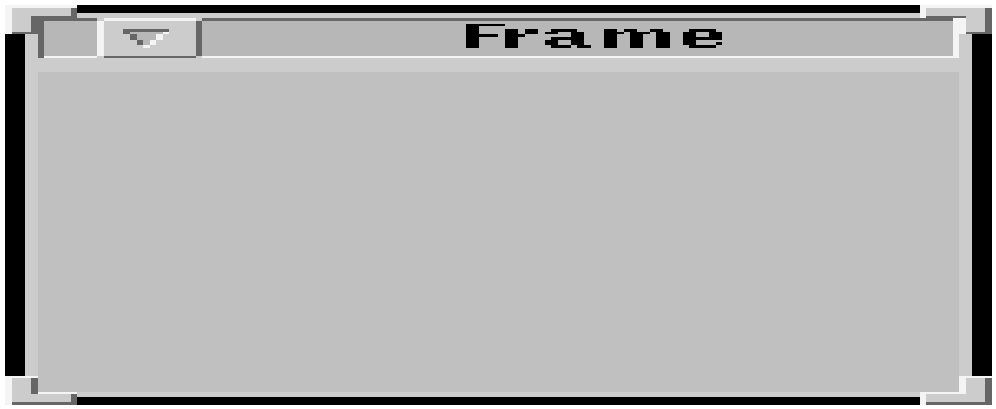
List

```
List l = new List(4, true);
```

- Un `ActionEvent` est généré par la liste dans les modes de sélection simple et multiple.

Frame

```
Frame f = new Frame(“Frame”);
```



- Un `WindowListener` est utilisé pour reconnaître, via la méthode `windowClosing()`, que le bouton Quit a été activé dans le menu du gestionnaire de fenêtres.

Panel

```
Panel p = new Panel();
```

- Conteneur important, servant à regroupper des composants graphiques.
- Seul conteneur qui modélise une fenêtre qui n'est pas une sous fenêtre du fond d'écran.

Dialog

- Semblable à un Frame dans ses décorations : c'est une fenêtre sous la fenêtre fond d'écran, habillé par le gestionnaire de fenêtres.
- Un Dialog diffère d'une Frame par les faits :
  - - elle est destinée à afficher des messsages fugitifs
  - - elle peut être modale: elle recevra systématiquement toutes les formes d'entrées jusqu'à sa fermeture.
  - - elle ne peut-être supprimer ou iconifier par les boutons du gestionnaire de fenêtre, elle contient habituellement un bouton de validation

Dialog

- Création du Dialog

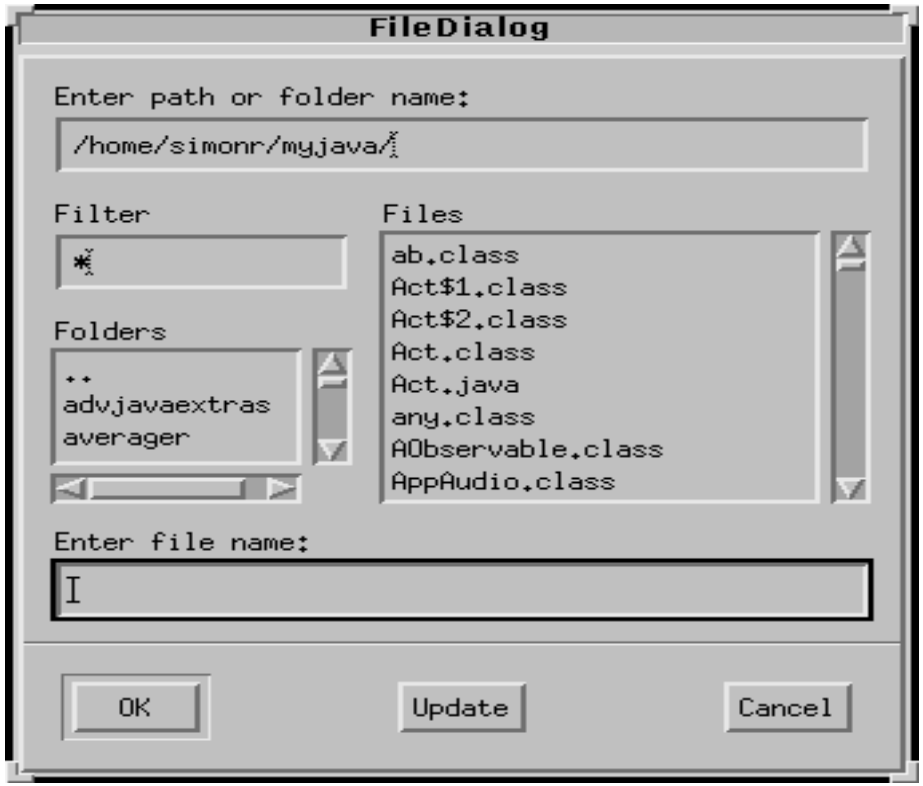
```
Dialog d = new Dialog(f, "Dialog", false);
d.add("Center",
      new Label("Hello, I'm a Dialog"));
d.pack();
```

- Affichage du Dialog

```
public void actionPerformed(ActionEvent ev) {
     d.setVisible(true);
}
```

FileDialog

```
FileDialog d = new FileDialog(f, "FileDialog");
d.setVisible(true);
String fname = d.getFile();
```



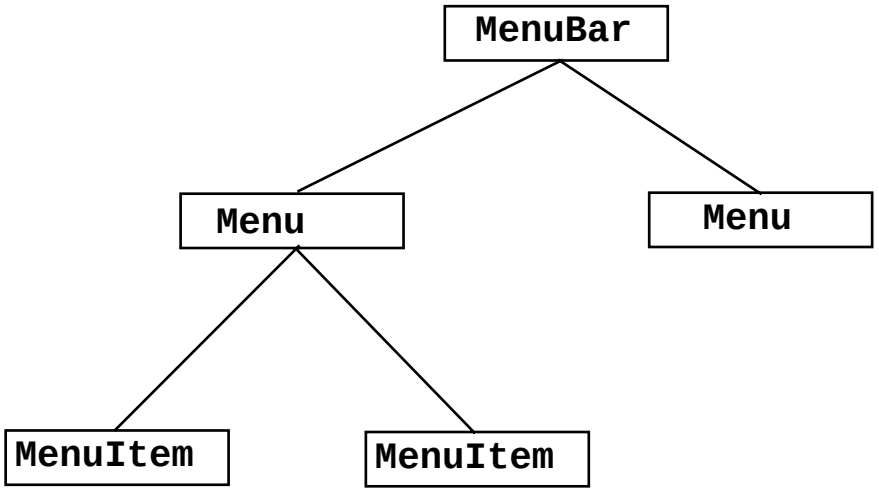
ScrollPane

```
Frame f = new Frame("ScrollPane");
Panel p = new Panel();
ScrollPane sp = new ScrollPane();
p.setLayout(new GridLayout(3, 4));
sp.add(p);
f.add(sp);
f.setSize(200, 200);
f.setVisible(true);
```

Menus

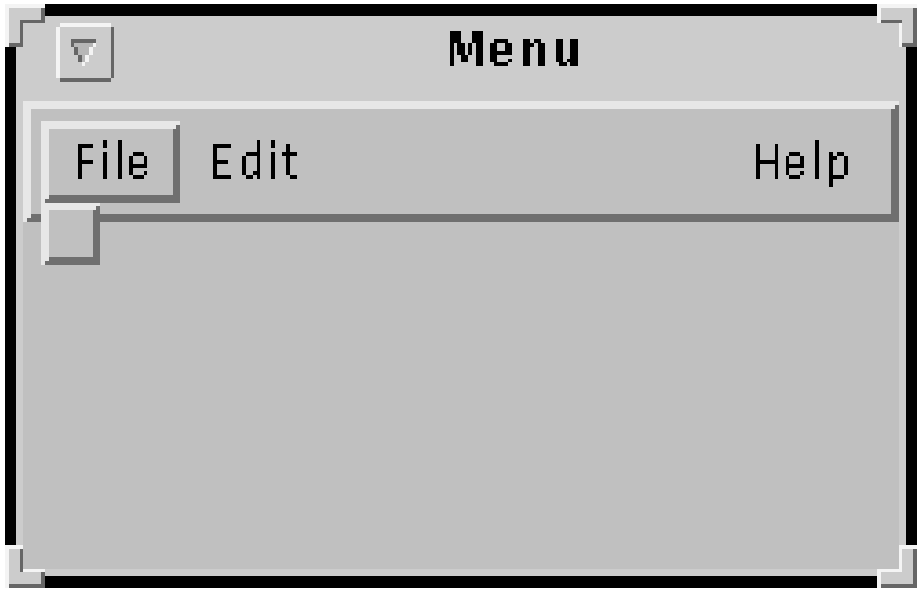
- Menu Aide
- MenuBar

```
Frame f = new Frame("MenuBar");
MenuBar mb = new MenuBar();
f.setMenuBar(mb);
```



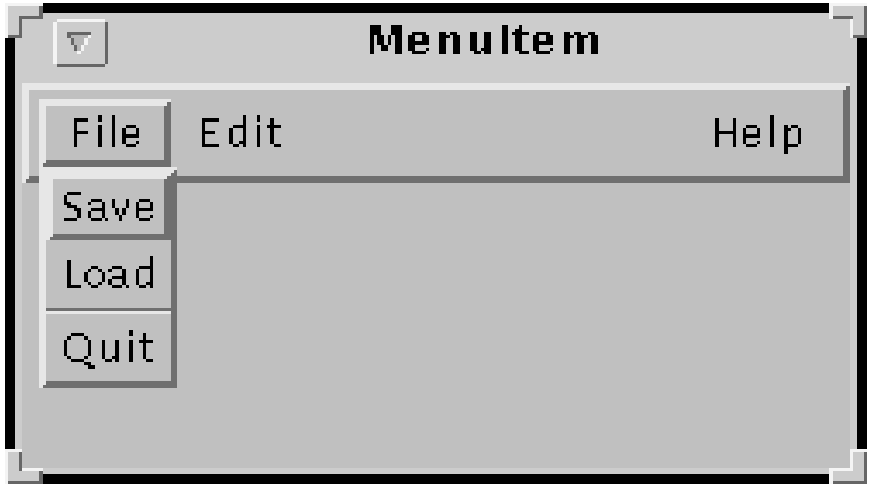
Menu

```
MenuBar mb = new MenuBar();
Menu m1 = new Menu("File");
Menu m2 = new Menu("Edit");
Menu m3 = new Menu("Help");
mb.add(m1);
mb.add(m2);
mb.add(m3);
mb.setHelpMenu(m3);
```



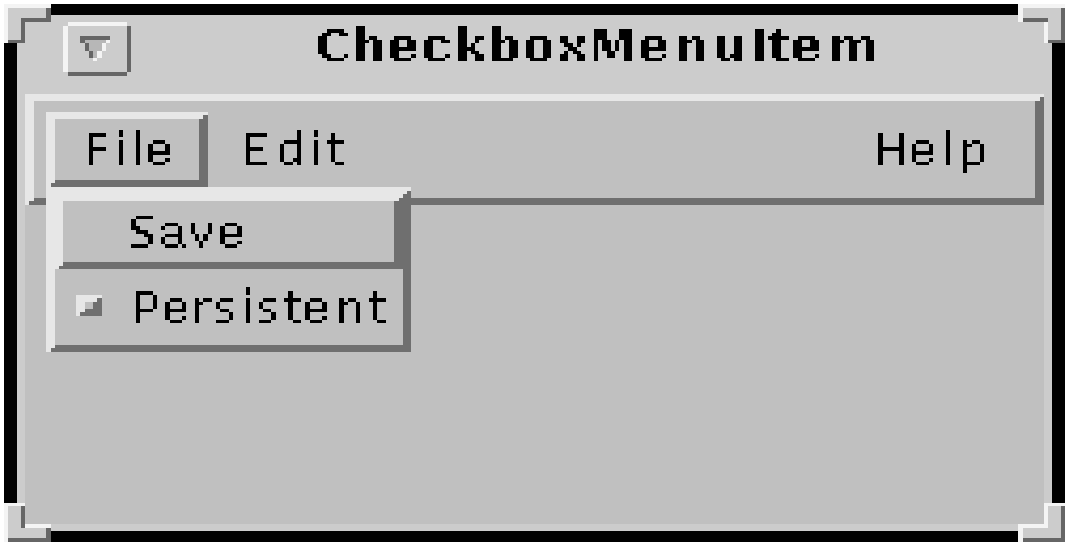
MenuItem

```
Menu m1 = new Menu("File");
MenuItem mi1 = new MenuItem("Save");
MenuItem mi2 = new MenuItem("Load");
MenuItem mi3 = new MenuItem("Quit");
m1.add(mi1);
m1.add(mi2);
m1.addSeparator();
m1.add(mi3);
```



CheckboxmenuItem

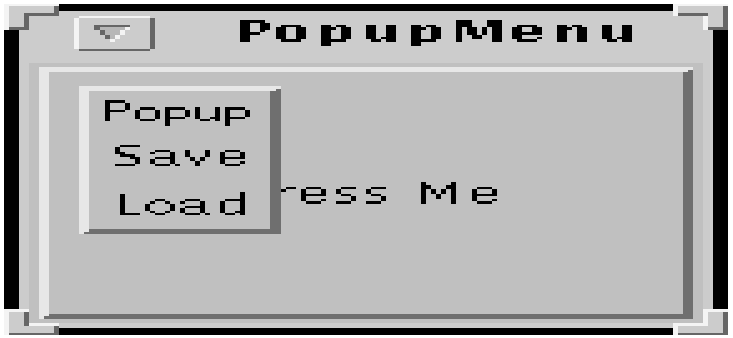
```
Menu m1 = new Menu("File");
MenuItem mi1 = new MenuItem("Save");
CheckboxMenuItem mi2 =
   new CheckboxMenuItem("Persistent");
m1.add(mi1);
m1.add(mi2);
```



PopupMenu

- Création du Popup

```
Frame f = new Frame("PopupMenu");
Button b = new Button("Press Me");
b.addActionListener(this);
PopupMenu p = new PopupMenu("Popup");
MenuItem s = new MenuItem("Save");
MenuItem l = new MenuItem("Load");
s.addActionListener(this);
l.addActionListener(this);
f.add("Center", b);
p.add(s);p.add(l);
f.add(p);
```



PopupMenu .

- Affichage de la Popup


```
public void actionPerformed(ActionEvent ev) {
   p.show(b, 10, 10);
}
```

Composant de référence pour l'affichage du popup menu.

Coordonnées % composantes d'origines

Contrôle des aspects visuels

- Couleurs

```
setForeground(...)
setBackground(...)
int r = 255, g = 255, b = 0;
Color c = new Color(r, g, b);
```

- Polices

```
Font f = new Font(“TimesRoman”, Font.PLAIN, 14);
```

- Impression

```
Frame f = new Frame("Print test");
Toolkit t = f.getToolkit();
PrintJob job = t.getPrintJob(f, "MyPrintJob", null);
Graphics g = job.getGraphics();

f.printAll(g);

g.dispose();
job.end();
```

Exercice

Illustration interface graphique et événements:

▼ **Ecrire une application autonome de nom MyDraw.java:**

Définir dans la partie supérieure un Panel de contrôle.

Définir dans la partie inférieure un Canvas de dessin.

Dans le panneau de contrôle définir :

1. un choix couleur
2. un choix exclusif Figure
   (Rectangle , Cercle , Ligne).

Prévoir la gestion des événements pour ces 2 choix.

Prévoir également la gestion des événements souris dans le canvas:

1. Mémorisation des coordonnées X Y à l'appui sur un bouton de la souris.

Exercice (suite)

2. Mémorisation des coordonnées X Y au relâchement du bouton de la souris et ...

   a. Création d'une figure de type actuellement sélectionné dans un tableau de figures (100 figures maxi).

   b. Appel de la méthode paint() du canvas.

   c. Cette méthode parcourt le tableau de figure et appelle les méthodes dessine() de tous les objets creés jusqu'à maintenant.

Note – De par le principe retenu, le rafraîchissement de la fenêtre en cas de recouvrement puis de nouvelles expositions sont automatiquement implémentées.