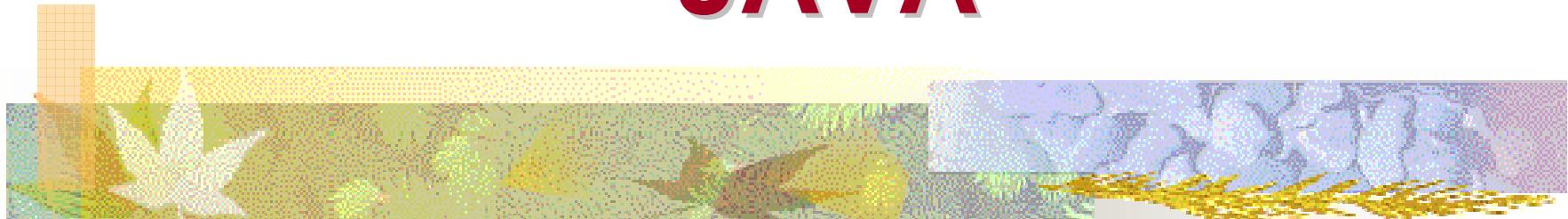




JAVA



Dr. Abderrazak JEMAI

Abderrazak.Jemai@insat.rnu.tn



PLAN

- Chapitre I – Notion Fondamentales en JAVA
- Chapitre II – Programmation Orientée Objets en JAVA
- Chapitre III – Composants AWT en JAVA
- Chapitre IV – Les SWING en JAVA
- Chapitre V – Accès aux bases de données en JAVA
- Chapitre VI – TCP/IP en JAVA

Chapitre I

Notion Fondamentales en JAVA





Tutorial1 : Notions fondamentales de Java

- Introduction
- Java le langage
- Notions élémentaires du langage Java
 - **Instructions, expressions et blocs**
 - **Les commentaires**
 - **Les données**
 - **Les types de base**
 - **Les variables simples**
 - **Blocs et champs**
 - **Expressions et opérateurs**



Tutorial1 : Notions fondamentales de Java

- Le flot de contrôle
 - **if...else**
 - **switch**
 - **while, do...while**
 - **for**
- La programmation orientée objet
- Les objets et les classes Java
- Les tableaux et les chaînes de caractères

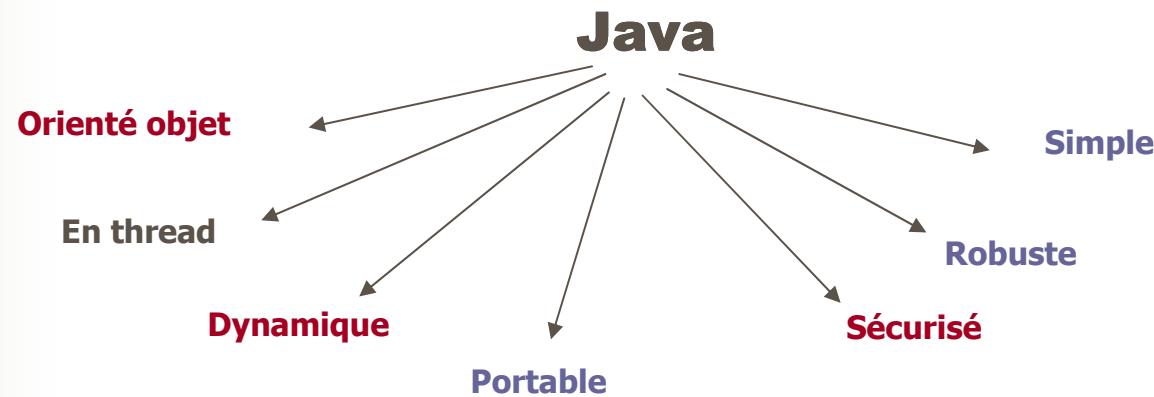


Présentation

- ▶ **C'est un langage qui a été développé par Sun dans l'intention de créer un langage de programmation orienté objet dynamique**
- ▶ **Java est un langage très proche du langage C++ dans lequel les sources d'erreurs ou de confusion les plus courantes ont été supprimées et contrôlées**
- ▶ **Java est tout à fait adapté pour être employé sur le Web**

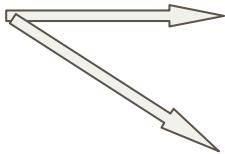


Présentation





Java : langage orienté objet pur ?

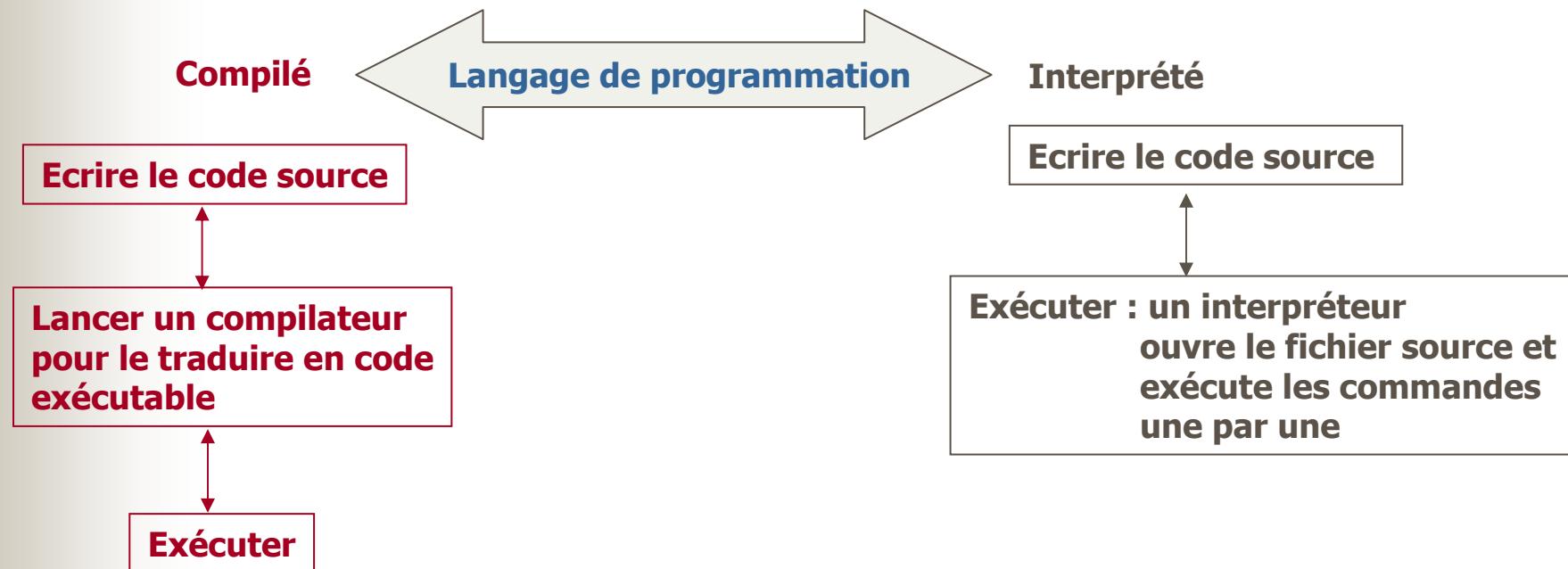
- Exemple : le langage SmallTalk est considéré comme un langage orienté objet pur.
- Java  **types simples (entiers, caractères, ...)**
tout le reste est représenté par des objets



Java est un langage orienté objet pur :



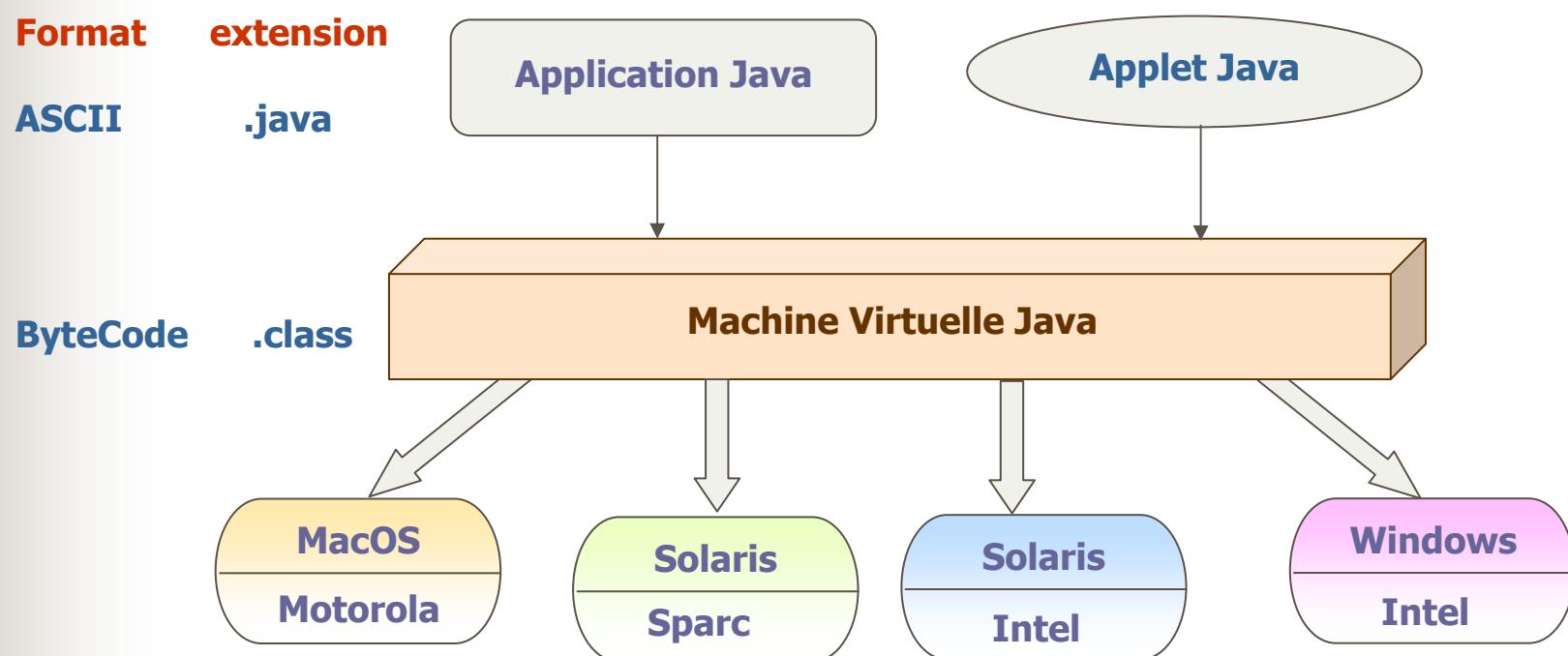
Java : langage compilé ou interprété ?





Java : langage compilé ou interprété ?

- ▶ Java est un langage compilé





Architecture de Java

- **Un programme Java est soit une application classique s'exécutant sur une machine soit une Applet**
- **Le concept de Java : indépendance vis à vis de la CPU**
- **Le fichier source d'un programme Java est un fichier ASCII écrit avec la syntaxe du langage Java qui contient des définitions de classes**
- **Une application peut être constituée de plusieurs fichiers sources**
- **Ces fichiers possèdent l'extension ".java"**



Architecture de Java

- **Les sources Java doivent être compilées afin de générer le ByteCode. Les fichiers générés ont l'extension ".class"**
- **Le ByteCode java est un langage machine dédié à la "pseudo" machine nommée : la Machine Virtuelle Java ou Java Virtual Machine (JVM)**
- **La JVM est un programme écrit pour chaque plate-forme particulière (Solaris, Mac, Windows NT), qui connaît le ByteCode Java et qui est capable d'interpréter les programmes Java compilés**

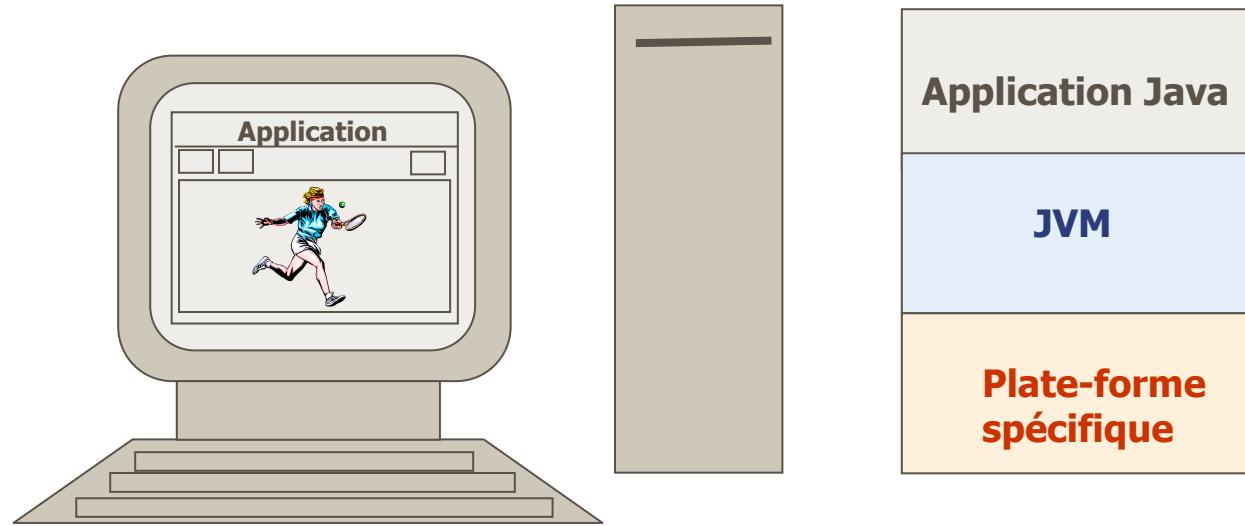


Force de Java

- **Il est facile de programmer sur des plate-formes différentes car cela ne nécessite que l'unique apprentissage de Java**
- **un programme Java s'exécute moins vite qu'un programme écrit en C++.**
- **Java est fourni avec un grand nombre de librairies très complètes pour permettre au programmeur de développer des applications traitant de nombreux thèmes**
- **Cette API (Application Programming Interface) fournit aussi bien des éléments de base du langage (chaînes de caractères, tableaux, ...) que des fonctionnalités plus complexes (accès aux BD, interfaces graphiques, ...)**



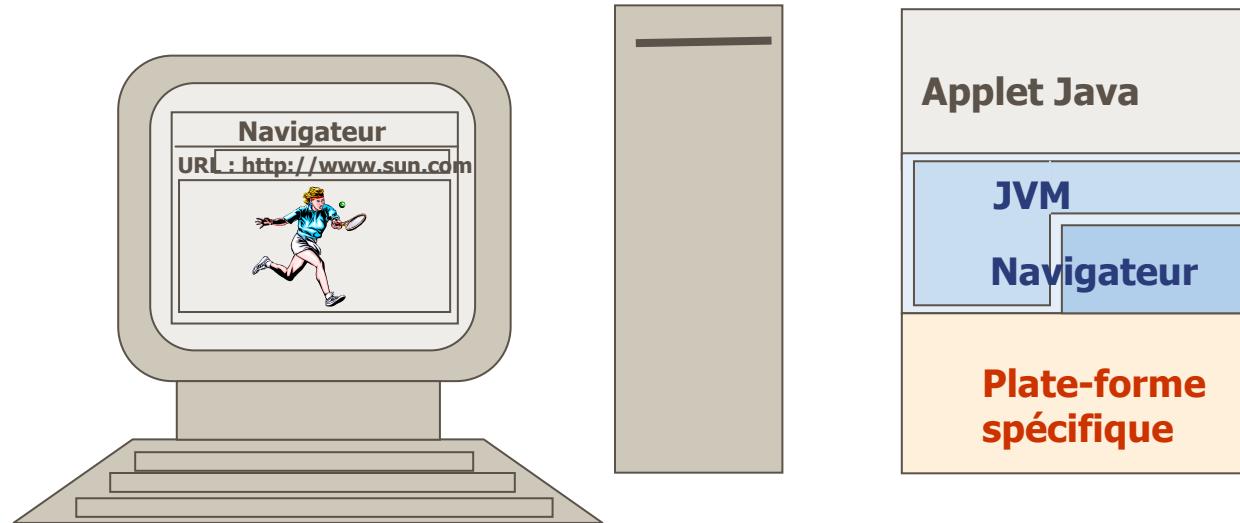
Application Java Standalone



- C'est une application qui s'exécute sur une machine isolée du réseau et qui possède ses propres ressources (disque, CPU, OS)
- Il faut lancer explicitement une JVM pour exécuter l'application
- Les ".class" de l'API Java se trouvent sur le disque local



Applet Java



- **Applet (ou une Appliquette) est une application Java qui s'exécute au sein d'une page HTML dans un navigateur (browser)**

Notions élémentaires du langage Java





Instructions et blocs

- ▶ Soit les expressions suivantes :

```
c = a/b  
k = 2+2  
i = i+1
```

- ▶ Pour que ces expressions deviennent compréhensibles par le compilateurs Java, il faut rajouter à chacune un ; à la fin, ce qui donne une **déclaration d'expression ou une instruction Java** :

```
c = a/b;  
k = 2+2;  
i = i+1;
```

- ▶ Dans un programme Java, les instructions sont organisées en **blocs**. Un bloc est reconnu par les **accolades** qui le délimitent : une accolade ouvrante { au début du bloc et une accolade fermante } à la fin du bloc. Il n'y a pas de ; à la fin d'un bloc



Les commentaires

- ▶ C'est du **texte inclu dans le code source d'un programme Java**
- ▶ Ils fournissent des **renseignements au programmeur ou au lecteur à propos du code**
 - entre **/* et */**, le commentaire peut occuper plusieurs lignes
 - après **//** le commentaire s'arrête à la fin de la ligne

```
/* exemple de commentaire 1 */
/* un exemple de commentaire
sur plusieurs lignes
*/

//un commentaire d'une seule ligne

a = a+1 ; //le code est exécuté et ce commentaire est ignoré

/* un commentaire de plusieurs lignes
a = a+1;
la déclaration de la ligne précédente ne sera jamais exécutée
*/
```



Les identificateurs

- ▶ Ce sont des séquences de caractères alphanumériques utilisables pour nommer une variable, une méthode ou une classe.

```
caractères permis :  
les lettres de a-z et de A-Z  
les chiffres de 0 à 9 sauf au début de l'identificateur  
_ $
```

- ▶ Java distingue minuscules et majuscules
- ▶ Convention : tout nom autre que le nom d'une classe en minuscules

```
a ≠ A  
reponse ≠ Reponse  
choix ≠ choix
```

- ▶ quelques exemples corrects

```
int i;  
double le_perimetre  
boolean $paye  
char lettre_double
```

- ▶ quelques exemples incorrects

~~```
int 2k;
char val%
double int;
```~~



## Les identificateurs

### Liste des interdits (mots clés Java)

|          |            |              |
|----------|------------|--------------|
| abstract | for        | protected    |
| boolean  | future     | public       |
| break    | generic    | rest         |
| byte     | goto       | return       |
| case     | if         | short        |
| cast     | implements | static       |
| catch    | import     | super        |
| char     | inner      | switch       |
| class    | instanceof | synchronized |
| const    | int        | this         |
| continue | interface  | throw        |
| default  | long       | throws       |
| do       | native     | transient    |
| double   | new        | try          |
| else     | null       | var          |
| extends  | operator   | void         |
| final    | outer      | volatile     |
| finally  | package    | while        |
| float    | private    |              |



## Les variables simples

- ▶ **Les variables sont des récipients qui contiennent les données utilisées dans un programme Java**
- ▶ **avant qu'une variable puisse être utilisée dans un programme Java, elle doit avoir été déclarée au préalable**
- ▶ **déclarer une variable est l'action de lui donner un nom et de définir le type des données que l'on peut mettre dedans**
- ▶ **un nom de variable est un identificateur**

```
type identificateur;
```



# Les données littérales

- Une donnée littérale est la représentation la plus explicite d'un type de données dans un code source Java.

```
a = 2 ; // 2 est un entier littéral
s = "Bonjour"; /* le texte entre guillemets est une chaîne de
caractère littérale */
```

- Dans Java, chaque type de données littérales est régi par des règles d'utilisation

# Les données littérales

## Les nombres

- **Les valeurs numériques dans Java peuvent être des entiers, des nombres à virgule flottante et des caractères**
- **Les entiers**
  - ce sont des nombres entiers qui peuvent être représentés en décimal, hexadécimal ou en octal.

```
a = 2 ; // 2 est un entier littéral
```

- **Les nombres à virgule flottante**
  - un tel nombre est n'importe quel nombre à virgule
  - on peut également utiliser la notation scientifique en précédant d'un E la valeur de l'exposant

```
a = 11.23 ; // 11.23 est un littéral à virgule flottante
b = 3.2 E -10;
```

# Les données littérales

- **Les caractères**
  - ce sont des valeurs 16 bits, représentées par un seul caractère entouré de guillements simples
  - certains caractères doivent être représentés par une séquence d'escape

| code escape | caractère              |
|-------------|------------------------|
| \b          | arrière (backspace)    |
| \f          | formfeed               |
| \n          | newline                |
| \r          | retour chariot         |
| \t          | tabulation horizontale |
| \v          | tabulation verticale   |
| \\"         | backslash              |
| \?          | point d'interrogation  |
| \'          | guillemet simple       |
| \"          | guillemet double       |
| \000        | nombre octal           |
| \xhh        | nombre hexadécimal     |

```
r = 'O';
x = 'a';
```

# Les données littérales

## Les booléens

- Ils peuvent avoir la valeur true ou false

```
reponse = true;
quitter = false;
```

## Les chaînes de caractères

- Ce sont des chaînes contenant zéro ou plusieurs caractères entre des guillemets doubles
- une chaîne de caractère n'est pas accessible en tant que tableau de caractères

```
message = "Bienvenue à Tunis";
```

# Les types de base

- ▶ Les types servent à décrire aussi bien le contenu des variables que les valeurs renvoyées par les expressions
- ▶ Ils sont principalement utilisés dans les déclarations des variables

## Les nombres

- ▶ Java implémente quatre sortes de types d'entiers et deux sortes de types de nombres flottants
- ▶ tous ces types se distinguent par leur taille, ce qui signifie que chaque type correspond à un intervalle différent



```
byte b=100;
short i=32000;
int j;
long x;
float p = 2.73;
double t;
```

# Les types de base

## Les caractères : char

- ▶ Ce type est défini comme étant des valeurs Unicode 16 bits non signées
- ▶ rigoureusement parlant, un caractère est un type numérique, équivalent à une valeur entière short (16 bits)
- ▶ ainsi, il est possible d'assigner un littéral numérique à une variable de type caractère et un littéral caractère à une variable entière

```
char caract = 'a';
char c1 = 48;
int code1 = 'a';
```

## Les booléens : boolean

- ▶ les variables de ce type peuvent prendre l'une des deux valeurs : true ou false

```
boolean trouve = false;
boolean reponse = true;
```



# Les affectations et les casts de types

- ▶ la valeur assignée à une variable doit être compatible avec le type de celle-ci

```
type variable_a = (type)variable_b;
```

```
int a = 5;
byte b = (byte)a;
```



# Les initialisations

- Avant qu'une variable puisse être utilisée, il faut qu'elle soit initialisée : une valeur doit être mise dans une variable avant de s'en servir

```
int i;
i = i+1 ; // Interdit, car i n'a pas encore été initialisé
```

```
int i;
i=0;
i = i+1 ;
```

- la méthode la plus pratique est d'utiliser un **initialisateur** : c'est une affectation facultative qui fait partie de la déclaration de la variable. C'est une manière plus pratique d'être certain qu'une variable est initialisée avant son utilisation

```
int i=0;
```



# Expressions et opérateurs

- ▶ une expression est tout ce qui renvoie une valeur. Elle est typiquement constituée de deux valeurs ou opérandes séparées par un opérateur
- ▶ Pour l'ordre de précédence Java garde l'ordre mathématique connu, qui peut être changé en utilisant des parenthèses

## Opérateurs mathématiques

### binaires :

|   |                |
|---|----------------|
| + | addition       |
| - | soustraction   |
| * | multiplication |
| / | division       |
| % | modulo         |

### unaires :

|    |                 |
|----|-----------------|
| -  | négation unaire |
| ++ | incrément       |
| -- | décrément       |

```
a++;
b+=1;
c*=2;
```



# Expressions et opérateurs

Opérateurs binaires : valeurs entières

unaires :

- complément

binaires :

& ET binaire

| OU binaire

^ OU exclusif binaire

<< décalage à gauche

>> décalage à droite arithmétique (propageant le bit de signe)

>>> décalage à droite logique (faisant rentrer des zéros à gauche)

Comparaisons

== égal

!= différent de

< inférieur à

<= inférieur ou égal à

> supérieur à

>= supérieur ou égal à

Opérateurs binaires : valeurs booléennes

! négation : NOT

& ET binaire

| OU binaire

^ OU exclusif binaire

&& ET d'arrêt

|| OU d'arrêt

# Les flots de contrôle





## if ... else

```
if (expression de test) instruction-vrai (ou bloc) ;
[else instruction-faux (ou bloc) ;]
```

? :

```
expression-test ? valeur-vrai : valeur-faux
```

```
plus_petite = (j < k) ? j : k
```

```
if (j < k)
 plus_petite = j;
else
 plus_petite = k;
```



# switch

```
switch (expression) {
 case valeur-littérale1 :
 instruction1;
 [break];
 case valeur-littérale2 :
 instruction2;
 [break];
 ...
 default :
 instruction-default ;
 [break];
}
```



## while

```
while (expression_de_test) instruction_de_la_boucle (ou bloc);
```

## do ... while

```
do instruction_de_la_boucle (ou bloc);
 while (expression_de_test_)
```

## for

```
for (initialisation : expression_de_test : incrément)
 instruction_de_boucle;
```



## break

- ▶ l'instruction **break** peut être utilisée dans la plupart des formes de boucle pour permettre de sortir d'une boucle avant que la condition de sortie ne soit satisfaite

## continue

- ▶ l'instruction **continue** est utilisée pour lancer l'exécution d'une boucle sans avoir à exécuter le reste des instructions de la boucle
- ▶ cela peut être pratique lorsque le programmeur veut sauter certaines instructions dans le code d'une boucle compliquée

# Chapitre II

## La programmation orientée objet

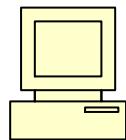




# Qu'est ce qu'un objet ?



- ▶ Notre **monde réel** n'est constitué que d'objets : personnes, animaux, plantes, formes, véhicules, ordinateurs, etc.
- ▶ Ces objets se partagent tous **deux caractéristiques**
  - **un état** (nom, couleur, taille, espèce, etc.)
  - **un comportement** (marcher, aboyer, manger, rouler, etc.)
- ▶ **Exemple : bicyclette      état : poids, couleur, pédales, roues**  
**comportement : rouler, accélérer, ralentir, freiner**
- ▶ A chaque état est associé une variable.
- ▶ A chaque comportement est associé une méthode.



## Définition

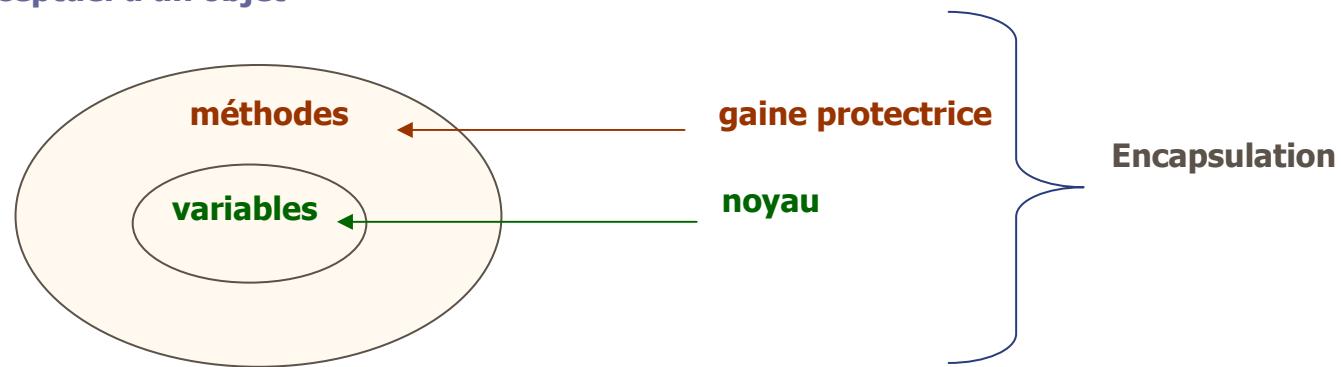
*Les variables et les méthodes sont le seul moyen pour exprimer l'état et le comportement d'un objet.*





# Qu'est ce qu'un objet ?

Schéma conceptuel d'un objet



```
class Bicyclettes
{
 int poids
 Color couleur;
 float rouler()
 {
 for (i=1; i<j; i++) {...}
 }
}
```



## Qu'en est-il d'une classe ?

- ▶ Dans le monde réel nous avons des objets **du même genre** qui ne diffèrent l'un de l'autre que par **quelques détails**
  - ▶ Une bicyclette donnée est **juste une** parmi beaucoup de bicyclettes dans le monde
  - ▶ Cette bicyclette est **une instance** de la classe appelée **Bicyclettes**
- ▶ **Bicyclettes** est appelée **Classe**.
- ▶ **Une classe est un modèle de ces objets, ce modèle est appelé *classe***

### **Définition**

***Une classe est un prototype qui définit les variables et les méthodes communes à tous les objets d'un certain genre.***

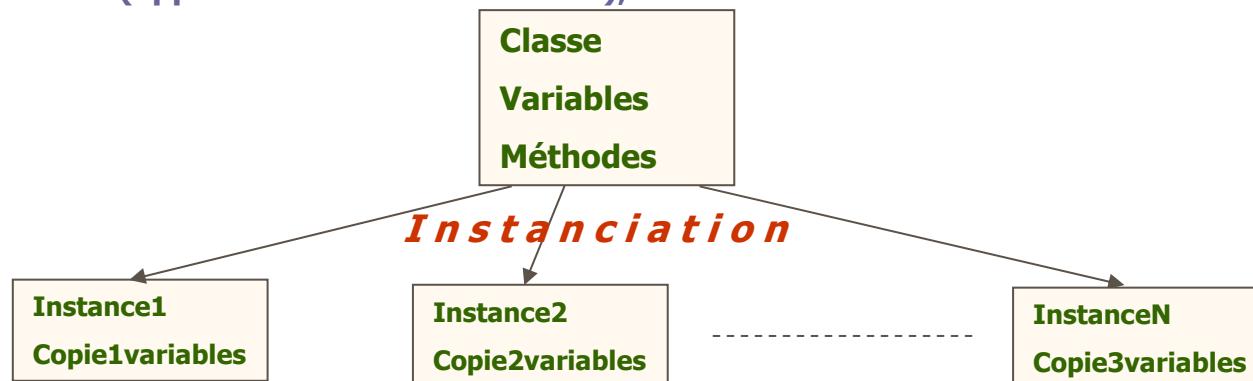


# Qu'en est-il d'une classe ?

- ▶ dans notre exemple :



- ▶ Après la création d'une classe, le développeur peut créer **n'importe quel nombre d'objets de cette classe**
- ▶ Lorsqu'on crée une **instance** d'une classe, le système **alloue autant d'espace mémoire qu'il est nécessaire pour cet objet et toutes ses variables**. Chaque instance aura **sa propre copie des variables** (appelées variables d'instance);



- ▶ Toutefois, si la valeur d'une variable d'instance est **toujours la même** pour tous les objets de cette classe, elle sera définie comme une **variable de classe**
- ▶ On peut également définir des **méthodes de classe**

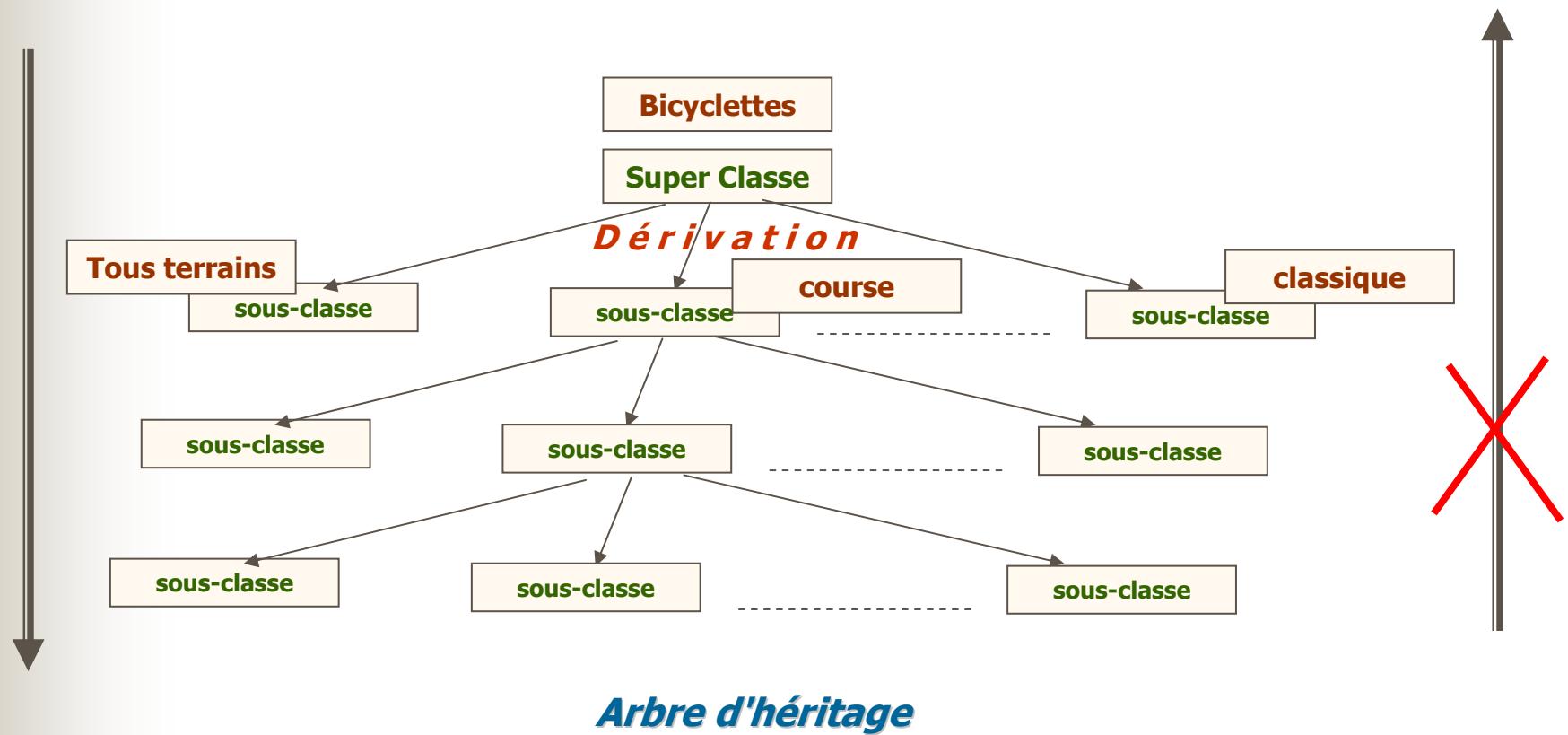


# Sous-classe et Héritage ?

- ▶ Il arrive souvent que l'on veuille **organiser** les objets d'une classe en **sous-groupes** pour mieux les **gérer**
- ▶ chaque sous-groupe réunira des objets qui **se partagent** certaines caractéristiques spécifiques
- ▶ **bicyclettes**
  - ▶ **tous terrains**
  - ▶ **course**
  - ▶ **classique**
- ▶ chaque sous-groupe est appelé **sous-classe** de la classe initiale ou **classe dérivée** ou **classe descendante**
- ▶ la classe initiale est appelée **super classe** ou **classe parente**
- ▶ chaque sous-classe **hérite** les variables et les méthodes de sa classe parente
- ▶ de plus, une sous-classe peut **rajouter** des variables et des méthodes à celle qu'elle hérite
- ▶ une sous-classe peut également **outrepasser** une méthode héritée pour la remplacer par une autre du même type plus adaptée à son comportement spécifique : **Polymorphisme**



# Sous-classe et Héritage ?





# Classe abstraite ?

- ▶ Une **classe abstraite** est une classe qui a des méthodes non encore définies.
- ▶ La classe qui étend cette classe abstraite doit définir les méthodes abstraites (vides). Elle peut bien entendu redéfinir (Polymorphisme) les méthodes déjà définies dans la classe parente.
- ▶ **A quoi sert une classe abstraite :** Parfois, on ne peut pas définir certaines méthodes d'une classe parente.
- ▶ **Exemple :**
- ▶

```
class GenieLogiciel extends Etudiant
{
 int calculMoyenne()
 {
 for (i= 1 ; i< nbMatieres) {...}
 }
}
```

```
abstract class Etudiant
{
 int cin;
 String nom;
 ...
 int calculBourse()
 {
 for (i= 1 ; i< nbMatieres)
 {...}
 }
 abstract int calculMoyenne()
}
```



# Interface ?

- ▶ une interface est tout simplement une classe là où toutes ses méthodes sont non définies.
- ▶ La classe qui implémente (étend) cette interface doit définir toutes les méthodes de l'interface.
- ▶ Une interface n'admet pas de variables membre.

```
abstract class candidat
{
 abstract int calculMoyenne()
}
```



## Syntaxe des classes en Java

### Définition

```
[modificateur] class Nom_de_classe [extends Nom_de_classe] [implements
interface] {

 [déclaration de variables]

 [déclaration de méthodes]

}
```

### Exemple

```
class Lampe {

 boolean allumee;

 void metEtat(boolean allumee) {

 this.allumee=allumee;
 }

 void afficheEtat() {

 if (allumee) { System.out.println("La lampe est allumée");}
 else { System.out.println("La lampe est éteinte"); }
 }
}
```

# Classes

## Les méthodes

### Déclaration

```
[modificateur] type_de_retour nom_de_la_méthode(liste_de_paramètres) {
 [instructions]
}
```

void  
int  
String

int x, float y, ...



# Classes

## Les méthodes

### Méthodes surchargée

- ▶ Créer plusieurs **versions** différentes de la méthode, pour pouvoir l'appeler différemment.
- ▶ Exemple :  
**La méthode print doit pouvoir imprimer des entiers, des chaînes, etc.**

```
Class MyObject {
 float x,y,z;
 public void print(int x) { ... }
 public void print(float x) { ... }
 public void print(String x) { ... }
}
```



# Classes

## Les constructeurs

- ▶ Un constructeur est une fonction qui est appelée lorsqu'un objet est créé comme instance d'une classe
- ▶ Un constructeur est défini comme une méthode qui a le même nom que sa classe
- ▶ Un constructeur est une méthode qui ne retourne aucune valeur

```
class MaDate {
 int jour = 1;
 int mois = 1;
 int annee = 2001

 MaDate() {
 annee = 2002;
 }
 MaDate(int nouveau_mois, int nouveau_jour, int nouvelle_annee) {
 jour = nouveau_jour;
 mois = nouveau_mois;
 annee = nouvelle_annee;
 }
}
```



## Méthodes/Variables de classe (statiques) et d'instance

- Une méthode ou une variable est dite de classe si le mot clé `static` figure dans sa déclaration (le main est une méthode de classe)
- Si le mot clé `static` ne figure pas dans la déclaration d'une méthode ou d'une variable, il s'agit alors d'une méthode ou d'une variable d'instance
- Les variables statiques sont utilisées lorsque le développeur veut que la variable soit commune à toutes les instances de la classe

```
class Alien {
 boolean vivant;

 static int nombre_alien = 0;

 void Alien() {
 vivant = true;
 nombre_alien++;
 }
 void explode() {
 if (vivant) {
 vivant = false;
 nombre_alien--;
 }
 }
}
```

Classe

Variante d'instance  
Ne sera modifiée que dans l'objet  
Auquel elle appartiendra

Variante statique ou de classe  
Partagée par tous les objets  
instance de Alien

## Classes



- Une méthode statique sert lorsque le développeur veut définir des **fonctions globales** qui sont associées à une classe
- Une telle méthode ne peut pas accéder aux variables d'instance de la classe

```
class OutilsMath {
 static int met_au_carre(int x) {
 return x*x;
 }
 static int moitie(int x) {
 return x/2;
 }
}

class Calcul_Math {
 void resultat {
 int i = OutilsMath.met_au_carree(4);
 int j = OutilsMath.moitie(i);
 System.out.println(j);
 }
}
```



### Méthodes/variables privées et publiques

- Une méthode ou une variable est dite de type privé si le modificateur `private` figure dans sa déclaration
- Une variable ou méthode privée n'est visible qu'à l'intérieur de la classe où elle est définie : on ne pourra pas accéder directement à une telle variable ou méthode depuis une instance de la classe qui le contient.

### Variables finales

- Lorsque le développeur veut travailler avec une variable qu'il ne souhaite pas modifier (constante), il suffit qu'il la déclare comme étant final et ce en précédant son identificateur par le mot clé `final`

# Exemple

```
class Classique
{
 private String nom;
 private int nbRepetition;
 String message = "travaillons";
 Classique(String nom, int nbRepetitions)
 {
 this.nom = nom;
 this.nbRepetitions = nbRepetitions;
 }
 void repete()
 {
 for (int i = 0; i < nbRepetitions; i++)
 {
 System.out.println(message);
 }
 }
 public String toString()
 {
 return nom + " dit : " + message;
 }
 public static void main(String[] argv)
 {
 int repetition = Integer.parseInt(argv[0]);
 Classique ecrivain = new Classique("Jean", repetition);
 ecrivain.repete();
 ecrivain.message="reposons-nous";
 System.out.println(ecrivain);
 }
}
```

variables d'instance

méthodes d'instance

méthodes de classe

- Lorsque le programme est lancé, aucune instance de la classe Classique n'existe, ni donc évidemment aucune variable d'une instance de cette classe
- En revanche, les variables et les méthodes statiques existent dès qu'une classe est évoquée
- La méthode main peut donc être lancée sans qu'aucune instance de la classe Classique n'existe
- l'argument de la méthode main est un tableau de chaînes de caractères. Les chaînes contenues dans ce tableau sont des arguments envoyés par la ligne de commande, le nom du programme ne faisant pas partie des arguments



# Objets

## Références

### Définition

- ▶ Une fois un objet créé, sa référence est retournée et peut être utilisée par le développeur pour accéder aux variables et aux méthodes de cet objet
- ▶ Il est interdit de manipuler les références des objets elle-même par exemple en tant que pointeurs comme dans C++

```
ObjetDate laDate;
laDate = new ObjetDate(23,12,65);
laDate.année=1964;
```

### this

- ▶ Java fournit cette variable spéciale pour faire référence à l'objet courant.
- ▶ Elle peut être utilisée pour passer la référence de l'objet **courant** à une méthode d'un autre objet

# Objets

## Références

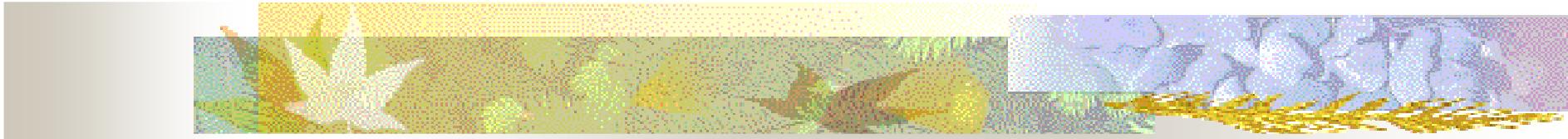
### null

- ▶ Si la valeur d'une référence d'objet est égale à **null**, alors cette référence **n'est pas valide**, c'est à dire qu'elle pointe sur un objet qui **n'existe pas**
- ▶ Elle peut être utilisée pour **tester** la validité d'une référence

```
if (monAmpoule != null) {
 System.out.println("monAmpoule existe !");
}
```

## opérateurs

|    |              |
|----|--------------|
| == | égal à       |
| != | différent de |



# Héritage

## Sous-classement

```
class Nom-de-classe extends Nom-de-classe [implements interface] {
 [déclaration de variables]
 [déclaration de méthodes]
}
```

Classe parente

Classe dérivée

```
class Felins {
 boolean affame = true;
 void parle() {
 }
 void appelle() {
 System.out.println("Ici, petit petit petit ...");
 if (affame) parle();
 }
}
class Chat extends Felins {
 void parle() {
 System.out.println("Miaou...");
 }
}
class Tigre extends Felins {
 void parle() {
 System.out.println("Grrrrr...");
 }
}
```



## Outrepasser des méthodes (*polymorphisme*)

### Outrepasser des méthodes

- ▶ Pour que des objets dans une même hiérarchie de classes répondent à des messages identiques de manière différente, il suffit d'implémenter les méthodes identiques dans les sous-classes de manière différente
- ▶ Une méthode est **outrepassée** lorsqu'une version en est implémentée dans une sous-classe et que cette version ait **les mêmes caractéristiques** (nom, type de valeur de retour, paramètres) que la méthode originale de la classe parent

```
Felins leChat;
leChat=new Chat;
leChat.parle();
```

- ▶ Ce mécanisme est appelé aussi *polymorphisme*

# Les tableaux et les chaînes de caractères





## Déclaration et allocation de tableaux

- Déclaration d'un tableau

```
char[] t;
```

Déclare le tableau T comme un tableau de caractères

- Allocation d'un tableau

```
t=new char[2]
```

Alloue le tableau T pour deux variables de type caractère

- Affectation de valeurs

```
class tableauA
{
public static void main(String[] argv)
{
char[] t;
t=new char[2];
t[0]='h'
t[1]='a'
}
}
```



## Dimensions d'un tableau et dépassement des bornes

- Pour connaître la longueur du tableau T on peut utiliser : **T.length**;
- Length est en quelque sorte un variable du tableau que l'on peut d'ailleurs uniquement lire
- Le plus grand indice dans un tableau est **T.length-1**, ainsi référencer **T.length** provoquerait sûrement une erreur de débordement qu'il aurait fallu gérer

## Tableau d'objets

- L'exemple suivant montre comment manipuler un tableau d'objets, plus spécifiquement un tableau d'Integer. A part pour cela, il n'est pas utile, pour additionner quelques entiers d'utiliser la classe d'objets Integer

```
class tableauC
{
public static void main(String[] argv)
{
 Integer T[] = new Integer[4];
 int somme=0;

 for (int i=0; i<T.length; i++)
 T[i]=new Integer[i];
 for (int i=0; i<T.length; i++)
 somme+=T[i].IntValue();
}
```



## Les chaînes de caractères

- Pour traiter les chaînes de caractères on utilise la classe **Java.lang.String**
- Lorsque le compilateur rencontre une série de caractères entre **doubles apostrophes**, il crée automatiquement **un objet String**.
- Java définit par ailleurs l'opérateur **+** de **concaténation** pour les chaînes de caractères
- Lorsque cet opérateur a une opérande qui est une chaîne de caractère et la deuxième non, alors il convertit cette dernière grâce à la méthode **toString** qui est définie dans **Object**
- Toutefois **toString** peut être redéfinie par l'utilisateur dans sa propre classe pour réaliser les fonctions qu'il souhaite

# Chapitre III

## JAVA et les composants AWT





# Les composants AWT

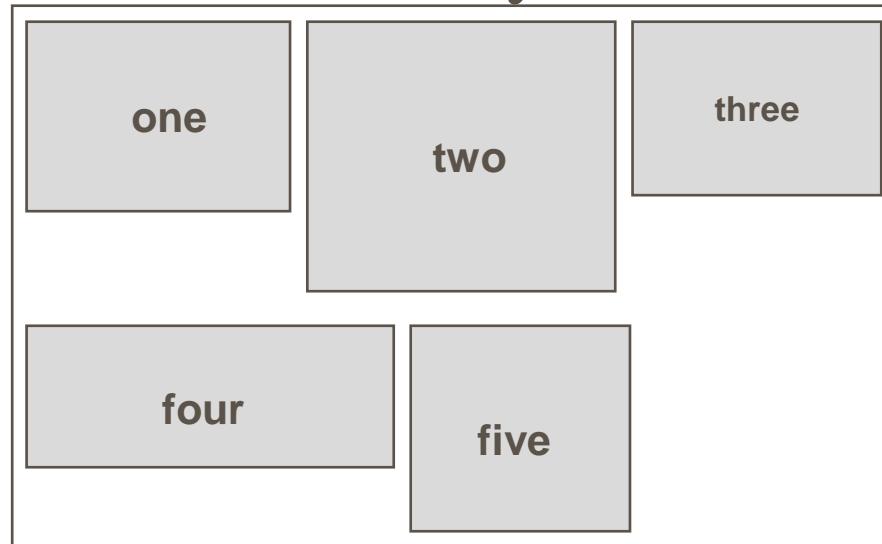
## ■ Les composants AWT comportent :

- Button
- Checkbox
- Choice
- Label
- MenuBar
- MenuItem
- TextArea
- TextField



# FlowLayout

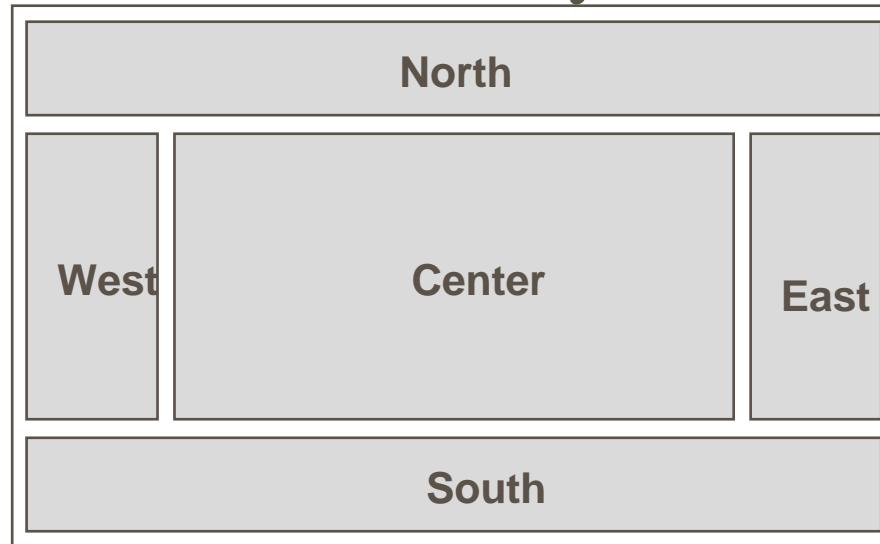
## ■ java.awt.FlowLayout





# BorderLayout

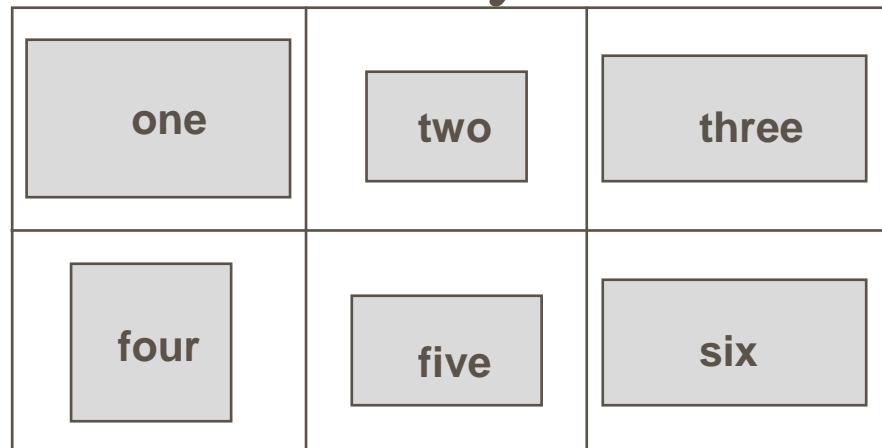
## ■ `java.awt.BorderLayout`





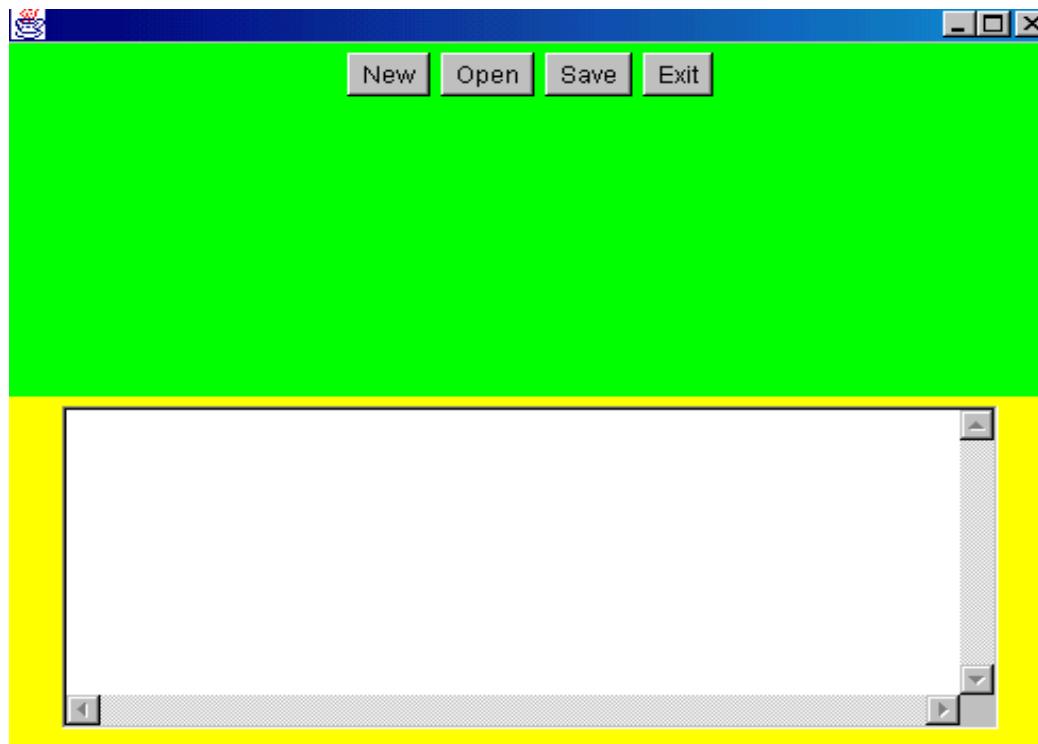
# GridLayout

- `java.awt.GridLayout`





# Exemple



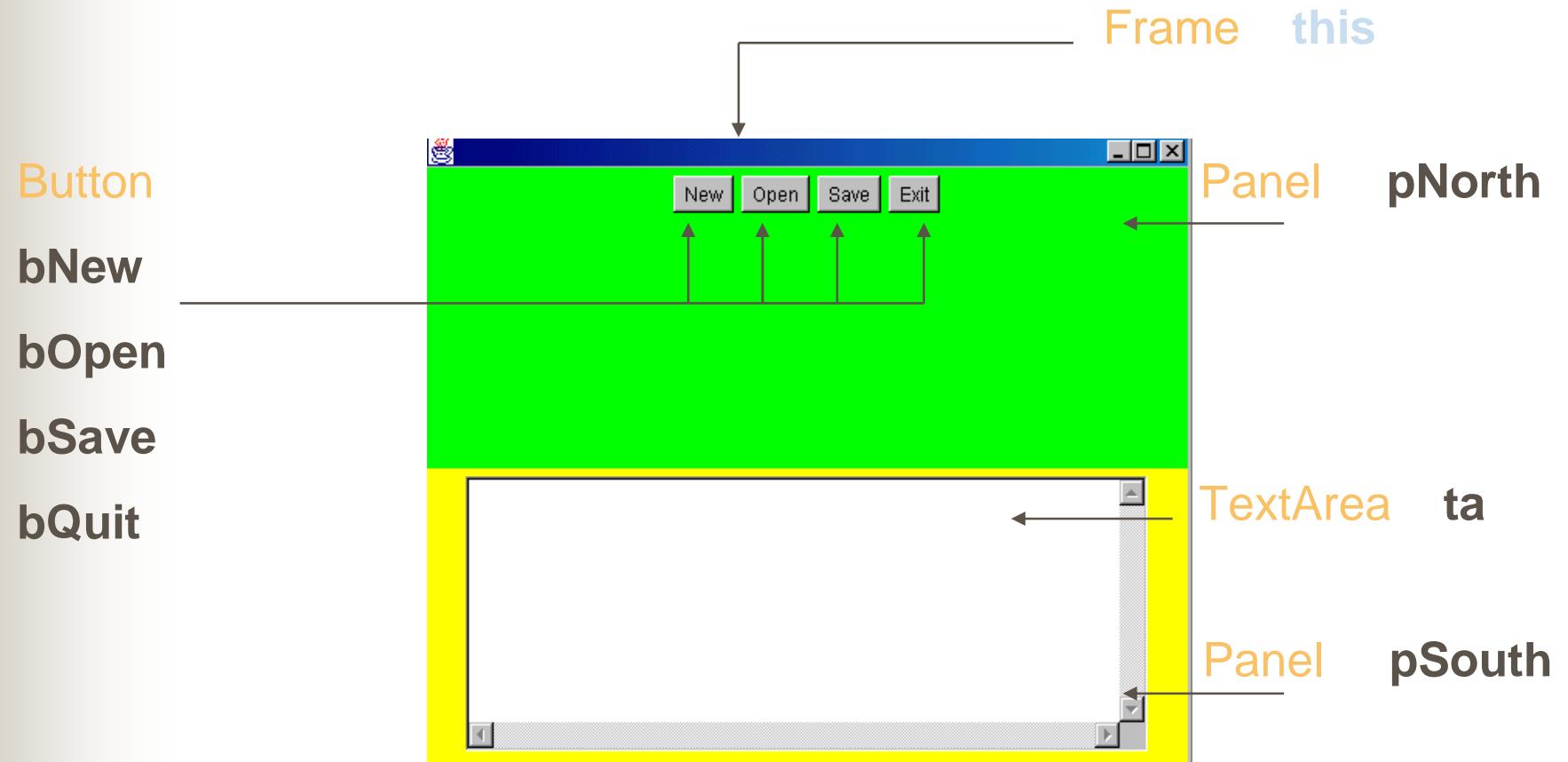


## Approche méthodologique

1. Dessiner l'interface
2. Attribuer des noms à chaque composant
3. Identifier les types de chaque composant
4. Déclarer la classe qui hérite de java.awt.Frame
5. Déclarez le ou les constructeurs
6. Déclarez les composants comme variables membres.
7. Rajoutez les instructions dans le constructeur.



# Application





# Application

1. Le dessin étant effectué
2. Attribuer des noms à chaque composant

- Panel pNorth = new Panel();
- Button bNew = new Button("New");
- Button bOpen = new Button("Open");
- Button bSave = new Button("Save");
- Button bQuit = new Button("Exit");
  
- Panel pSouth = new Panel();
- TextArea ta = new TextArea("");

Ces déclarations font l'objet des variables membres de la classe MyFrameAWT



## Classes

- Classe principale : **MyAppAWT**
- Classe Frame : **MyFrameAWT**
- Autres Classes pour la gestion des événements: **MyActionListenerForOpen**



# Classes

## ■ Classe principale : **MyAppAWT**

```
class MyAppAWT
{
 public static void main (String args [])
 {
 System.out.println("Application AWT");
 MyFrameAWT f = new MyFrameAWT();
 }
}
```



# Classes

## ■ Classe Frame : **MyFrameAWT**

```
import java.awt.*;
class MyFrameAWT extends Frame
{
 Panel pNorth = new Panel();
 Button bNew = new Button("New");
 Button bOpen = new Button("Open");
 Button bSave = new Button("Save");
 Button bQuit = new Button("Exit");

 Panel pSouth = new Panel();
 TextArea ta = new TextArea(" ");
}
```



## Classe MyFrameAWT - Variables Membres -

### ■ Classe Frame : **MyFrameAWT**

```
import java.awt.*;

class MyFrameAWT extends Frame
{
 Panel pNorth = new Panel();
 Button bNew = new Button("New");
 Button bOpen = new Button("Open");
 Button bSave = new Button("Save");
 Button bQuit = new Button("Exit");

 Panel pSouth = new Panel();
 TextArea ta = new TextArea(" ");
}
```



## Classe MyFrameAWT - Constructeurs -

- Classe Frame : **MyFrameAWT**

```
class MyFrameAWT extends Frame
{
... // variables membres
// Constructeurs
Public MyFrameAWT () {
 pNorth.setBackground(Color.green);
 pNorth.add(bNew); pNorth.add(bOpen);
 pNorth.add(bSave); pNorth.add(bQuit);

 pSouth.setBackground(Color.yellow);
 pSouth.add(ta);

 this.setBackground(Color.blue);
this.setLayout(new GridLayout(2,1));
 this.add(pNorth); this.add(pSouth);
 this.setBounds(10,10,500,400);
 this.setVisible(true);
}
}
```



## Gestion des événements - Fermeture de la fenêtre -

- Gestion des événements : fermeture de la fenêtre
- Créer une nouvelle classe appelée MyWindowListener qui hérite de **WindowAdapter**

```
MyWindowListener x1 = new MyWindowListener()
This.addWindowListener(x1);
```

```
public class MyWindowListener extends WindowAdapter
{
 public void windowClosing(WindowEvent evt) {
 System.exit(0);
 }
}
```



# Gestion des événements

## - Bouton Open -

- Gestion des événements : Bouton Open
- Créer une nouvelle classe appelée MyActionListenerForOpen qui implémente **ActionListener**
- Rajouter dans le constructeur MyAWTFrame() les deux lignes suivantes :

```
MyActionListenerForOpen x2 = new MyActionListenerForOpen ()
bOpen.addActionListener(x2);
```

```
public class MyActionListenerForOpen implements ActionListener
{
 public void actionPerformed(ActionEvent e)
 {
 System.out.println("Bouton Open actionné");
 }
}
```



## Entée/Sortie - Bouton Open -

- Classe cible : MyActionListenerForOpen
- Méthode cible : actionPerformed
- Approche :
  1. Activer le FileDialog
    - Créer une instance de FileDialog ayant comme argument un Frame.
    - Sélectionner le fichier à lire
    - Afficher son PATH complet
  2. Créer l'objet instance de File
    - Afficher la taille du fichier
  3. Ouvrir le fichier en mode Lecture :Créer l'objet instance de FileInputStream
  4. Lire les données dans un tableau de bytes[]
  5. Convertir le tableau en String
  6. Affecter le String dans le TextArea.

# Entée/Sortie

## - Ouverture du FileDialog -

- Classe cible : MyActionListenerForOpen
  - Méthode cible : ActionPerformed
  - Instructions :
    1. Créer une instance de FileDialog ayant comme argument un Frame.
      - Faire passer comme argument l'objet « this » dans MyFrameAWT.
- ```
MyActionListenerForOpen x2 = new MyActionListenerForOpen(this);
bOpen.addActionListener(x2);
```
- Rajouter un constructeur dans MyActionListenerForOpen qui admet comme argument un paramètre de type MyFrameAWT.

```
MyFrameAWT f;
public MyActionListenerForOpen(MyFrameAWT f)
{
    this.f=f;
}
```

Entée/Sortie

- Ouverture du FileDialog (suite) -

- Classe cible : MyActionListenerForOpen
- Méthode cible : actionPerformed
- Instructions :
 1. Créer une instance de FileDialog ayant comme argument un Frame.

```
public void actionPerformed(ActionEvent e)
{
    System.out.println("Bouton Open actionné");
    FileDialog fd = new FileDialog(f);
    fd.setVisible(true);
}
```

- N'oublier pas d'importer les paquetages
`import java.awt.*;`

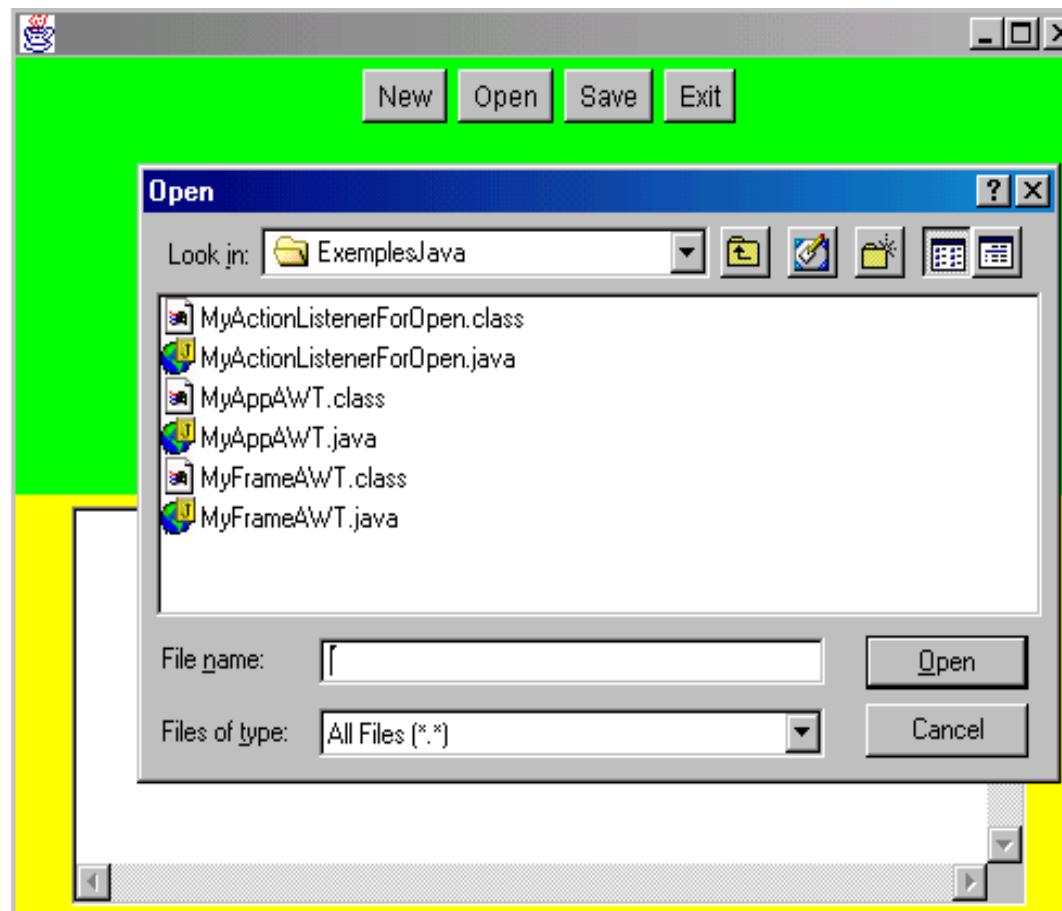
Entée/Sortie

- Ouverture du FileDialog (Solution Complète) -

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

class MyActionListenerForOpen implements ActionListener
{
    MyFrameAWT f;
    public MyActionListenerForOpen(MyFrameAWT f )
    {
        this.f=f;
    }
    public void actionPerformed(ActionEvent e)
    {
        FileDialog fd = new FileDialog(f);
        fd.setVisible(true);
    }
}
```

Ouverture du FileDialog - Résultat -





Entée/Sortie - Bouton Open -

- Classe cible : MyActionListenerForOpen
- Méthode cible : actionPerformed
- Approche :
 1. Activer le FileDialog
 - Créer une instance de FileDialog ayant comme argument un Frame.
 - **Sélectionner le fichier à lire**
 - **Afficher son répertoire**
 - **Afficher son PATH complet**
 2. Créer l'objet instance de File
 - Créer une instance de File
 - Afficher la taille du fichier
 3. Ouvrir le fichier en mode Lecture :Créer l'objet instance de FileInputStream
 4. Lire les données dans un tableau de bytes[]
 5. Convertir le tableau en String
 6. Affecter le String dans le TextArea.

Entée/Sortie

- Sélection du fichier -

```
import java.awt.event.*;
import java.awt.*;
class MyActionListenerForOpen implements ActionListener
{
public void actionPerformed(ActionEvent e)
{
    System.out.println("Bouton Open actionné");
    FileDialog fd = new FileDialog(f);
    fd.setVisible(true);

    String nomFichier = fd.getFile();
    String repFichier = fd.getDirectory();
    String nomComplet = repFichier + nomFichier;
    System.out.println("Nom Fichier : " + nomFichier);
    System.out.println("Répertoire Fichier : " + repFichier);
    System.out.println("Nom complet du Fichier : " +
        nomComplet);
}
}
```



Entée/Sortie - Bouton Open -

- Classe cible : MyActionListenerForOpen
- Méthode cible : actionPerformed
- Approche :
 1. Activer le FileDialog
 - Créer une instance de FileDialog ayant comme argument un Frame.
 - Sélectionner le fichier à lire
 - Afficher son répertoire
 - Afficher son PATH complet
 2. **Créer l'objet instance de File**
 - **Créer l'objet file**
 - **Afficher la taille du fichier**
 3. Ouvrir le fichier en mode Lecture :Créer l'objet instance de FileInputStream
 4. Lire les données dans un tableau de bytes[]
 5. Convertir le tableau en String
 6. Affecter le String dans le TextArea.

Entée/Sortie

- Sélection du fichier -

```
import java.awt.event.*;
import java.awt.*;
import java.io.*;
class MyActionListenerForOpen implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {   // suite du code
        String nomFichier = fd.getFile();
        String repFichier = fd.getDirectory();
        String nomComplet = repFichier + nomFichier;
        System.out.println("Nom Fichier : " + nomFichier);
        System.out.println("Répertoire Fichier : " + repFichier);
        System.out.println("Nom complet du Fichier : " +
                           nomComplet);
        File file = new File(nomComplet);
        int size;
        size = (int) file.length();
        System.out.println("Taille du Fichier : " + size);
    }
}
```



Entée/Sortie - Bouton Open -

- Classe cible : MyActionListenerForOpen
- Méthode cible : actionPerformed
- Approche :
 1. Activer le FileDialog
 - Créer une instance de FileDialog ayant comme argument un Frame.
 - Sélectionner le fichier à lire
 - Afficher son répertoire
 - Afficher son PATH complet
 2. Créer l'objet instance de File
 - Créer l'objet file
 - Afficher la taille du fichier
 3. **Ouvrir le fichier en mode Lecture :Créer l'objet instance de FileInputStream**
 4. Lire les données dans un tableau de bytes[]
 5. Convertir le tableau en String
 6. Affecter le String dans le TextArea.

Entée/Sortie

- Sélection du fichier -

```
import java.awt.event.*;
import java.awt.*;
import java.io.*;
class MyActionListenerForOpen implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {   // suite du code
        File file = new File(nomComplet);
        int size;
        size = (int) file.length();
        System.out.println("Taille du Fichier : " + size);

        try
        {
            FileInputStream in = new FileInputStream(file);
        }catch(FileNotFoundException e2){System.out.println(e2);}
    }
}
```



Entée/Sortie - Bouton Open -

- Classe cible : MyActionListenerForOpen
- Méthode cible : actionPerformed
- Approche :
 1. Activer le FileDialog
 - Créer une instance de FileDialog ayant comme argument un Frame.
 - Sélectionner le fichier à lire
 - Afficher son répertoire
 - Afficher son PATH complet
 2. Créer l'objet instance de File
 - Créer l'objet file
 - Afficher la taille du fichier
 3. Ouvrir le fichier en mode Lecture :Créer l'objet instance de FileInputStream
 4. **Lire les données dans un tableau de bytes[]**
 5. Convertir le tableau en String
 6. Affecter le String dans le TextArea.

Entée/Sortie

- Lecture des données -

```
import java.awt.event.*;
import java.awt.*;
import java.io.*;
class MyActionListenerForOpen implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {   // suite du code
        try
        {
            FileInputStream in = new FileInputStream(file);
            byte data[] = new byte[size];
            in.read(data);
        }catch(FileNotFoundException e2){System.out.println(e2);}
        catch(IOException e3){System.out.println(e3);}
    }
}
```



Entée/Sortie - Bouton Open -

- Classe cible : MyActionListenerForOpen
- Méthode cible : actionPerformed
- Approche :
 1. Activer le FileDialog
 - Créer une instance de FileDialog ayant comme argument un Frame.
 - Sélectionner le fichier à lire
 - Afficher son répertoire
 - Afficher son PATH complet
 2. Créer l'objet instance de File
 - Créer l'objet file
 - Afficher la taille du fichier
 3. Ouvrir le fichier en mode Lecture :Créer l'objet instance de FileInputStream
 4. Lire les données dans un tableau de bytes[]
 5. **Convertir le tableau en String**
 6. **Affecter le String dans le TextArea.**

Entée/Sortie

- Sélection du fichier -

```
// import
class MyActionListenerForOpen implements ActionListener
{
public void actionPerformed(ActionEvent e)
{   // suite du code

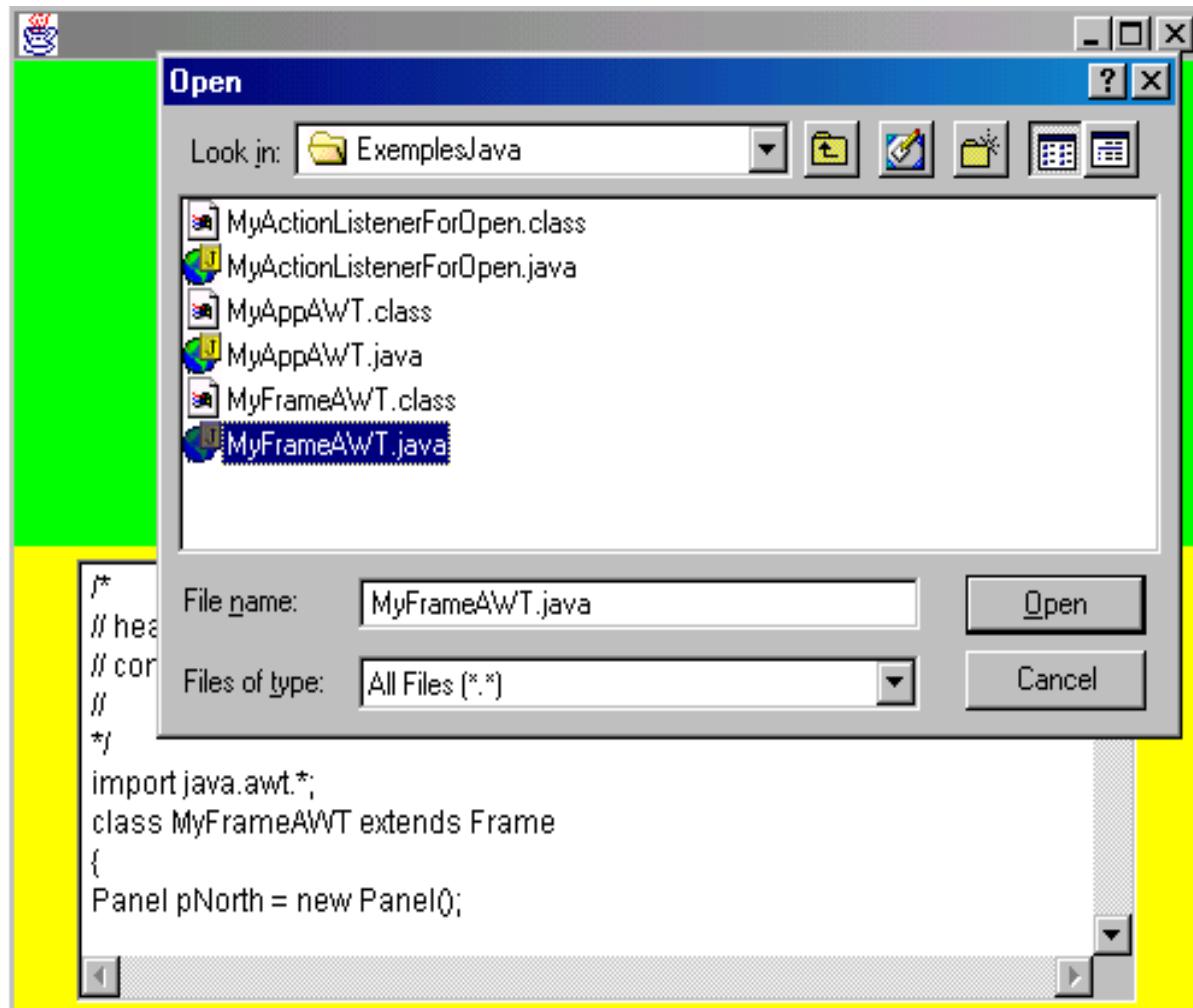
    try
    {
        FileInputStream in = new FileInputStream(file);
        byte data[] = new byte[size];
        in.read(data);

        String s = new String(data);
        f.ta.setText(s);

    }catch(FileNotFoundException e2){System.out.println(e2);}
    catch(IOException e3){System.out.println(e3);}

}
```

Ouverture d'un fichier avec FileDialog



Chapitre IV

Les SWINGs





Généralités

Les composants *Swing* constituent une évolution naturelles des composants AWT. Donc , ajout de nouvelles fonctionnalités pour chacune des classes. Ce package “*javax.Swing.**” comporte les éléments suivants :

- Les conteneurs de plus haut niveau : *JFrame*, *JApplet*
- Les composants légers (*Light-Weight*) : *JButton*, *JCheckBox*, etc.

Les composants Swing se placent dans un conteneur de plus haut niveau “ContentPane” : C'est la fenêtre visible.



Généralités

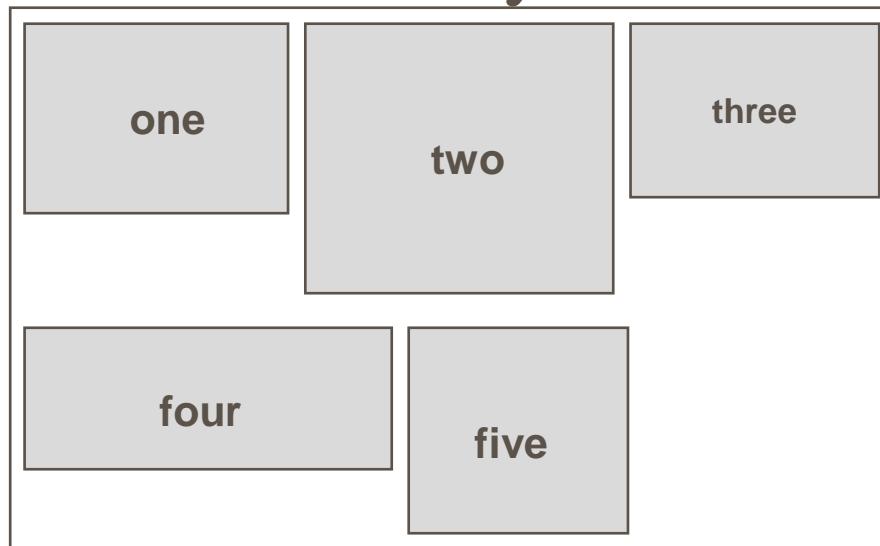
■ Les widgets

- javax.swing.JButton
- javax.swing.JRadioButton
- javax.swing.JCheckBox
- javax.swing.JLabel
- javax.swing.JList
- ...
- javax.swing.JTable
- javax.swing.JSplitPane
- javax.swing.JSlider
- javax.swing.JProgressBar



Rappel des composants AWT

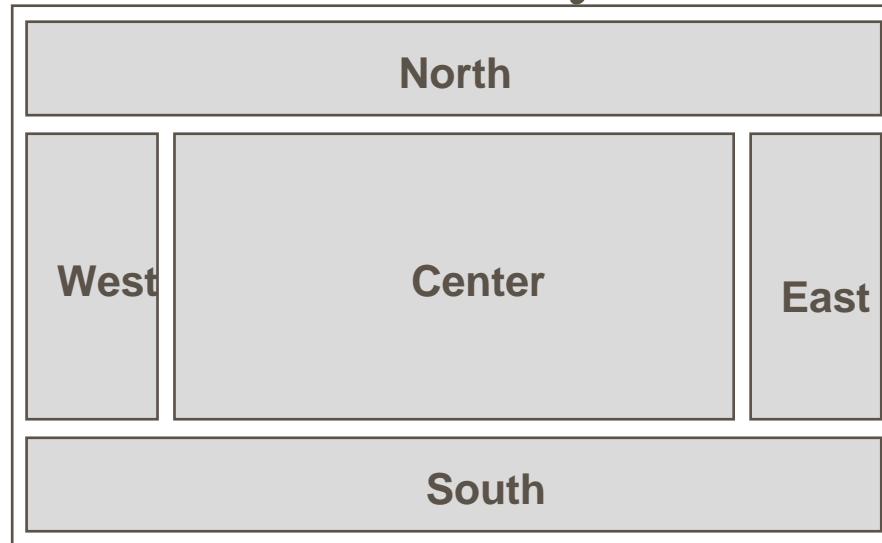
■ `java.awt.FlowLayout`





Rappel des composants AWT

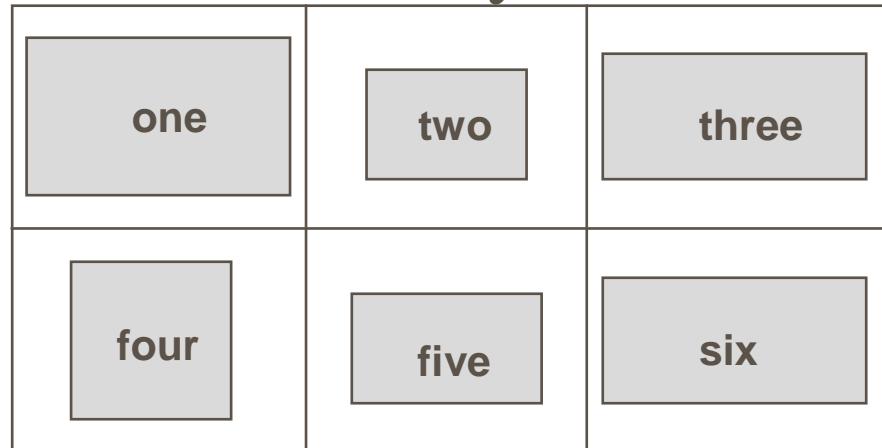
■ `java.awt.BorderLayout`





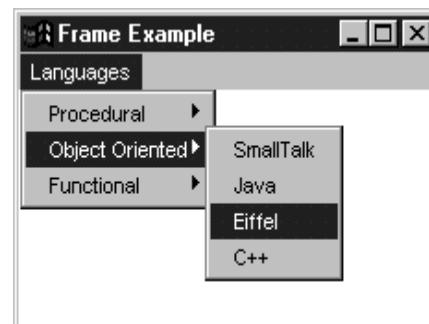
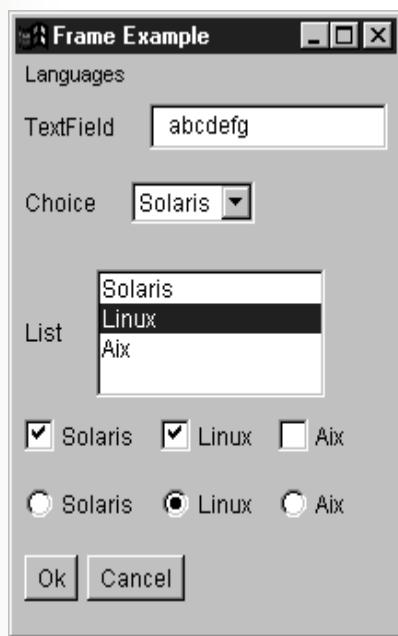
Généralités

■ `java.awt.GridLayout`



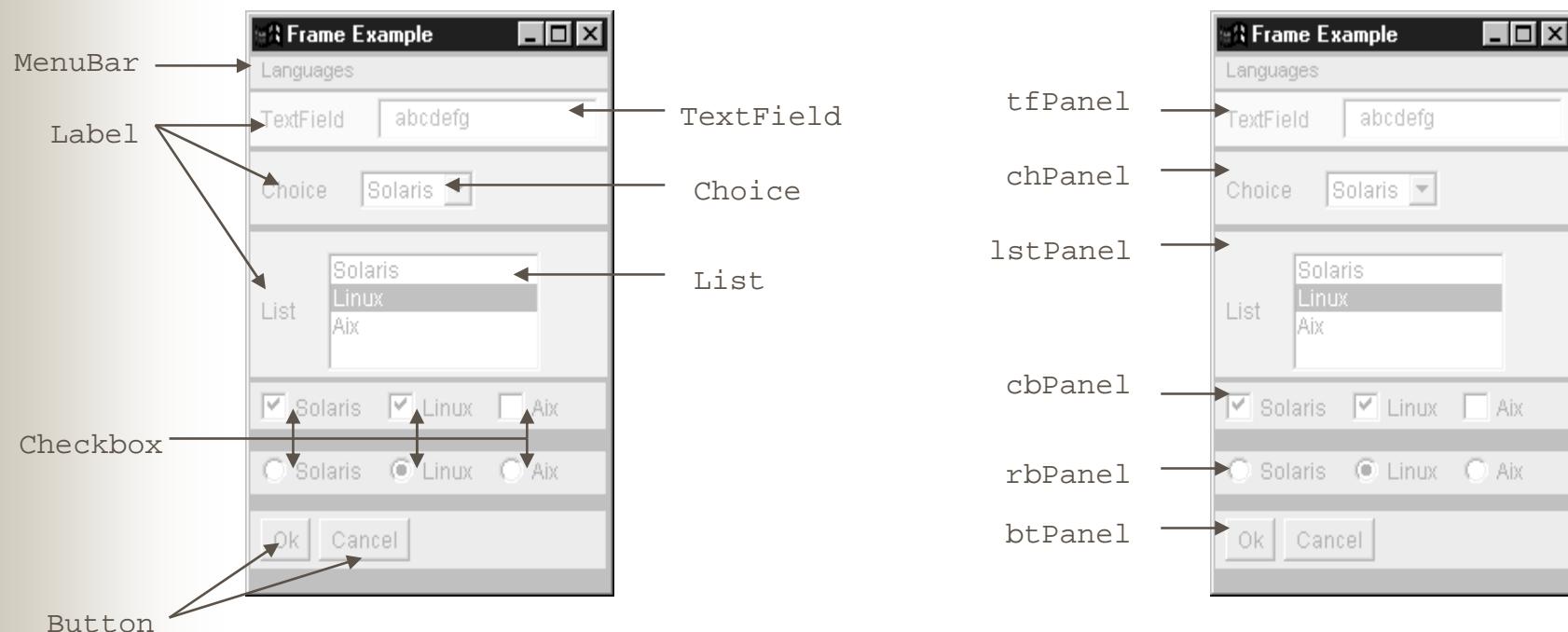


Exemple de composants AWT





Exemple de composants AWT





Exemple AWT

```
// Text Field Panel  
Panel tfPanel = new Panel(new FlowLayout(FlowLayout.LEFT));  
  
label = new Label("TextField");  
tfPanel.add(label);  
  
textField = new TextField(15);  
tfPanel.add(textField);
```



Exemple AWT

```
// Choice Panel  
Panel chPanel = new Panel(new FlowLayout(FlowLayout.LEFT));  
  
label = new Label("Choice");  
chPanel.add(label);  
  
choice = new Choice();  
choice.addItem("Solaris");  
choice.addItem("Linux");  
choice.addItem("Aix");  
  
chPanel.add(choice);
```



Exemple AWT

```
// List Panel  
Panel listPanel = new Panel(new FlowLayout(FlowLayout.LEFT));  
  
label = new Label("List");  
listPanel.add(label);  
  
list = new List();  
list.addItem("Solaris");  
list.addItem("Linux");  
list.addItem("Aix");  
  
listPanel.add(list);
```



Exemple AWT

```
// Checkbox Panel  
Panel cbPanel = new Panel(new FlowLayout(FlowLayout.LEFT));  
  
cbSolaris = new Checkbox("Solaris");  
cbPanel.add(cbSolaris);  
  
cbLinux = new Checkbox("Linux");  
cbPanel.add(cbLinux);  
  
cbAix = new Checkbox("Aix");  
cbPanel.add(cbAix);
```



Exemple AWT

```
// RadioButton Panel  
Panel rbPanel = new Panel(new FlowLayout(FlowLayout.LEFT));  
  
rbGroup = new CheckboxGroup();  
  
rbSolaris = new Checkbox("Solaris");  
rbSolaris.setCheckboxGroup(rbGroup);  
rbPanel.add(rbSolaris);  
  
rbLinux = new Checkbox("Linux");  
rbLinux.setCheckboxGroup(rbGroup);  
rbPanel.add(rbLinux);  
  
rbAix = new Checkbox("Aix");  
rbAix.setCheckboxGroup(rbGroup);  
rbPanel.add(rbAix);  
  
rbGroup.setSelectedCheckbox(rbSolaris);
```



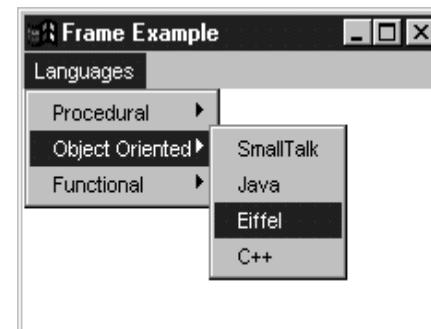
Exemple AWT

```
// Button Panel  
Panel btPanel = new Panel(new FlowLayout(FlowLayout.LEFT));  
  
okButton = new Button("Ok");  
okButton.addActionListener(new OkButtonListener());  
btPanel.add(okButton);  
  
cancelButton = new Button("Cancel");  
cancelButton.addActionListener(new CancelButtonListener());  
btPanel.add(cancelButton);
```



Exemple AWT

```
// Menu Bar  
Menu procMenu = new Menu( "Procedural" );  
procMenu.add(new MenuItem( "Pascal" ));  
...  
  
Menu ooMenu = new Menu( "Object Oriented" );  
ooMenu.add(new MenuItem( "SmallTalk" ));  
...  
  
Menu funcMenu = new Menu( "Functional" );  
funcMenu.add(new MenuItem( "Lisp" ));  
...  
  
Menu languageMenu = new Menu( "Languages" );  
languageMenu.add(procMenu);  
...  
  
MenuBar mb = new MenuBar( );  
mb.add(languageMenu);
```





Exemple AWT

```
public class FrameExample extends Frame {  
  
    public FrameExample(String title) {  
        super(title);  
        // Widget, panel, menu bar creation  
        ...  
        setMenuBar(mb);  
        setLayout(new GridLayout(6,1)); // 6 lignes et 1 colonne  
        add(textFieldPanel);  
        add(choicePanel);  
        add(listPanel);  
        add(checkboxPanel);  
        add(radioButtonPanel);  
        add(buttonPanel);  
        pack();  
        show();  
    }  
}
```



Exemple AWT

■ Gestion des événements : clic sur le bouton Ok

```
okButton.addActionListener(new OkButtonListener());  
  
public class OkButtonListener implements ActionListener {  
    public void actionPerformed(ActionEvent evt) {  
        System.out.println("textField = " + textField.getText());  
        System.out.println("choice = " + choice.getSelectedItem());  
        System.out.println("list = " + list.getSelectedItem());  
        System.out.println("cbSolaris=" + (cbSolaris.getState() ? "Yes" : "No"));  
        System.out.println("cbLinux=" + (cbLinux.getState() ? "Yes" : "No"));  
        System.out.println("cbWindow=" + (cbaix.getState() ? "Yes" : "No"));  
        System.out.println("rbSolaris=" + (rbSolaris.getState() ? "Yes" : "No"));  
        System.out.println("rbLinux=" + (rbLinux.getState() ? "Yes" : "No"));  
        System.out.println("rbWindow=" + (rbAix.getState() ? "Yes" : "No"));  
    }  
}
```



Exemple AWT

- Gestion des événements : sélection d'items du menu

```
MenuItem mi = new MenuItem("Pascal");
mi.addActionListener(new LanguageMenuListener());

public class LanguageMenuListener implements ActionListener {
    public void actionPerformed(ActionEvent evt) {
        System.out.println("Menu: " + evt.getActionCommand());
    }
}
```



Exemple AWT

- Gestion des événements : fermeture de la fenêtre

```
addWindowListener(new MyWindowListener());  
  
public class MyWindowListener extends WindowAdapter {  
    public void windowClosing(WindowEvent evt) {  
        System.out.println("Window closing");  
        System.exit(0);  
    }  
}
```



Les Composants SWING



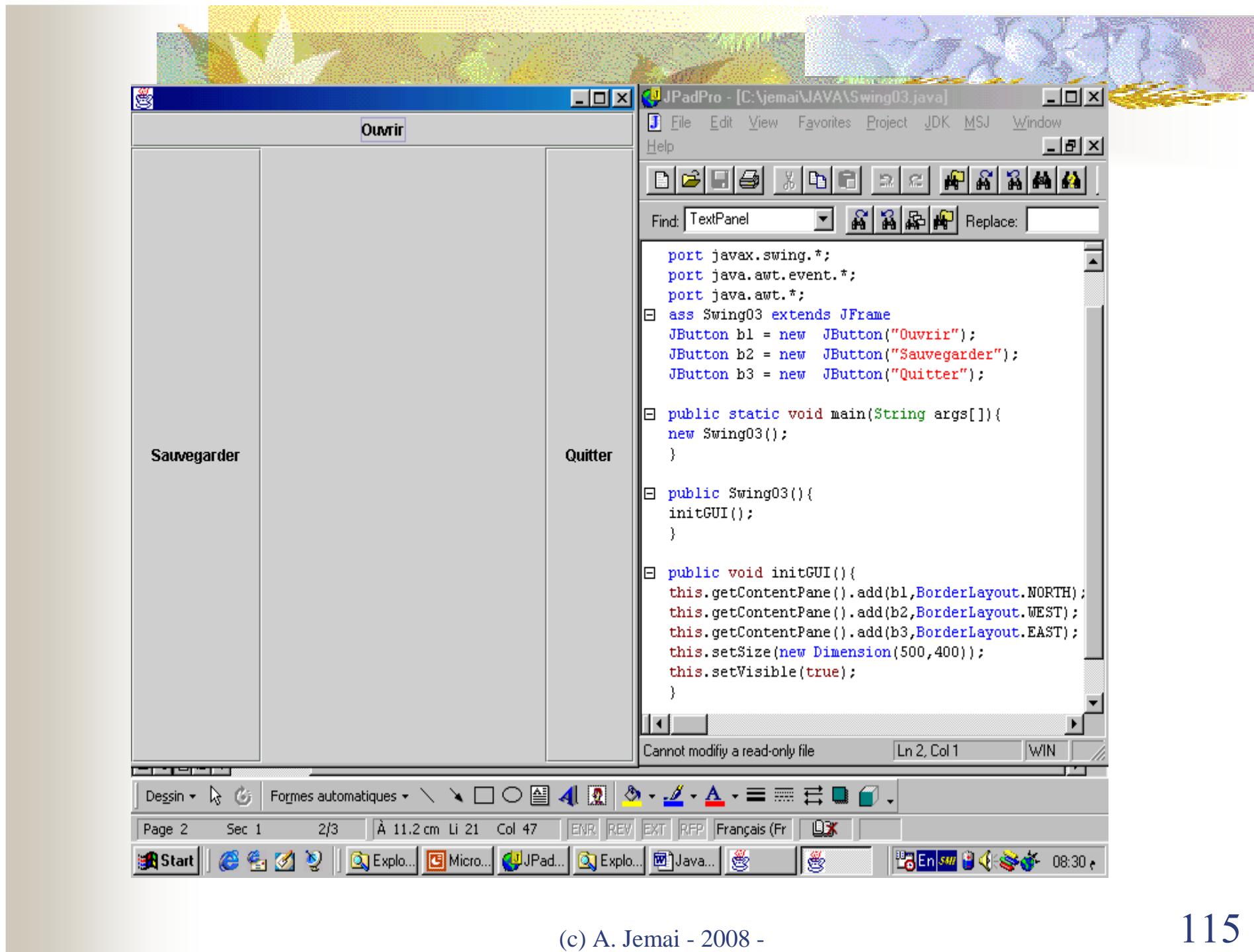
JFrame

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class Swing03 extends JFrame
{ JButton b1 = new JButton("Nord");
  JButton b2 = new JButton("Ouest");
  JButton b3 = new JButton("Est");

  public static void main(String args[]){
  new Swing03();
  }

  public Swing03(){
  initGUI();
  }

  public void initGUI(){
  this.getContentPane().add(b1,BorderLayout.NORTH);
  this.getContentPane().add(b2,BorderLayout.WEST);
  this.getContentPane().add(b3,BorderLayout.EAST);
  this.setSize(new Dimension(500,400));
  this.setVisible(true);
  }
}
```

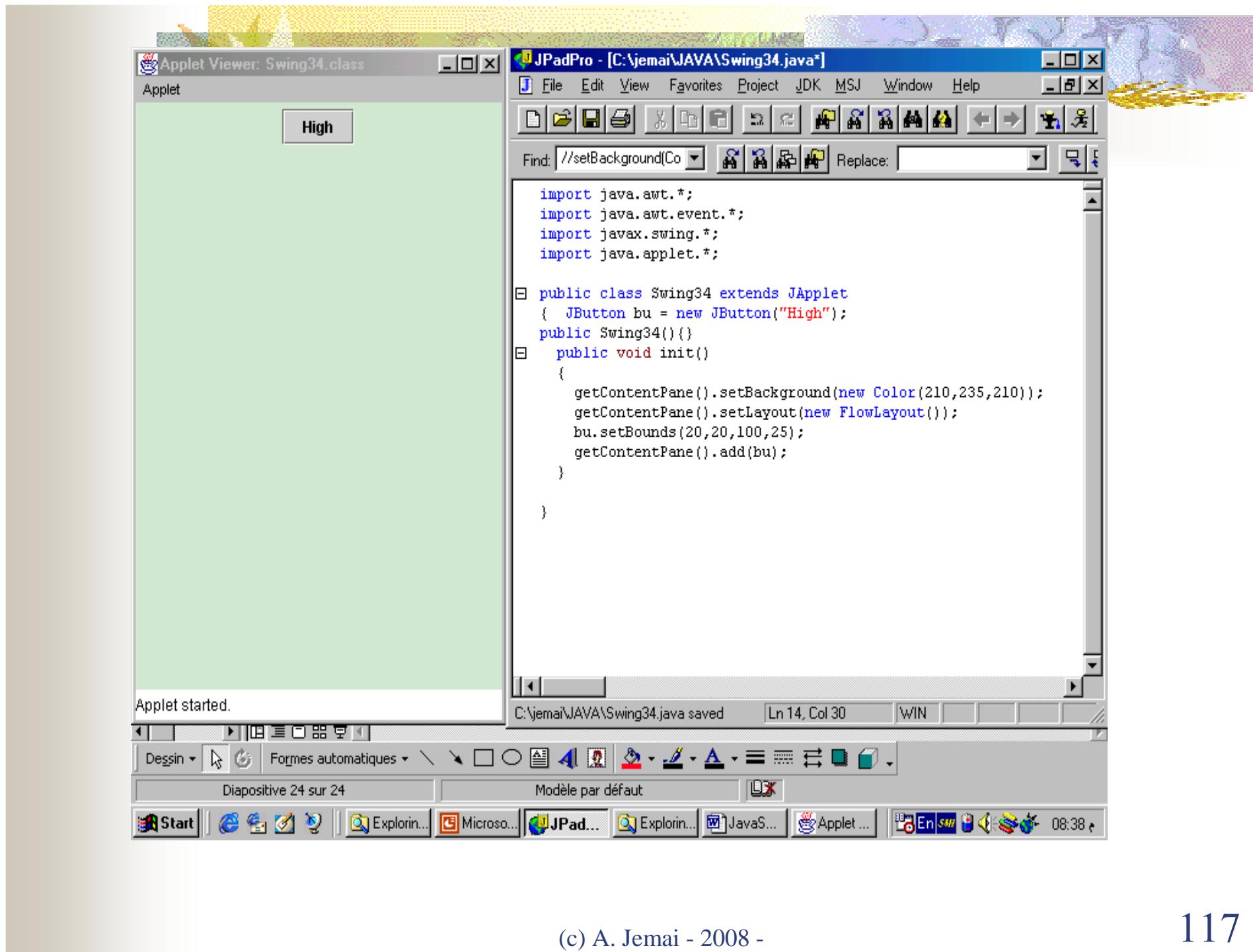




JApplet

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

public class Swing34 extends JApplet
{ JButton bu = new JButton("High");
public Swing34(){}  
public void init()
{
    getContentPane().setBackground(new Color(210,235,210));
    getContentPane().setLayout(new FlowLayout());
    bu.setBounds(20,20,100,25);
    getContentPane().add(bu);
}
}
```





JSplitPane

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

class JSplitPaneEx extends JFrame
{
    JPanel p1 = new JPanel();
    JPanel p2 = new JPanel();
    JPanel p3 = new JPanel();

    JSplitPane sp1 = new JSplitPane();
    JSplitPane sp2 = new JSplitPane();

    Color cp1 = new Color(100,150,200);
    Color cp2 = new Color(150,150,200);
    Color cp3 = new Color(150,200,200);

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

    public JSplitPaneEx(){
        initJSplitPaneEx();
    }
}
```



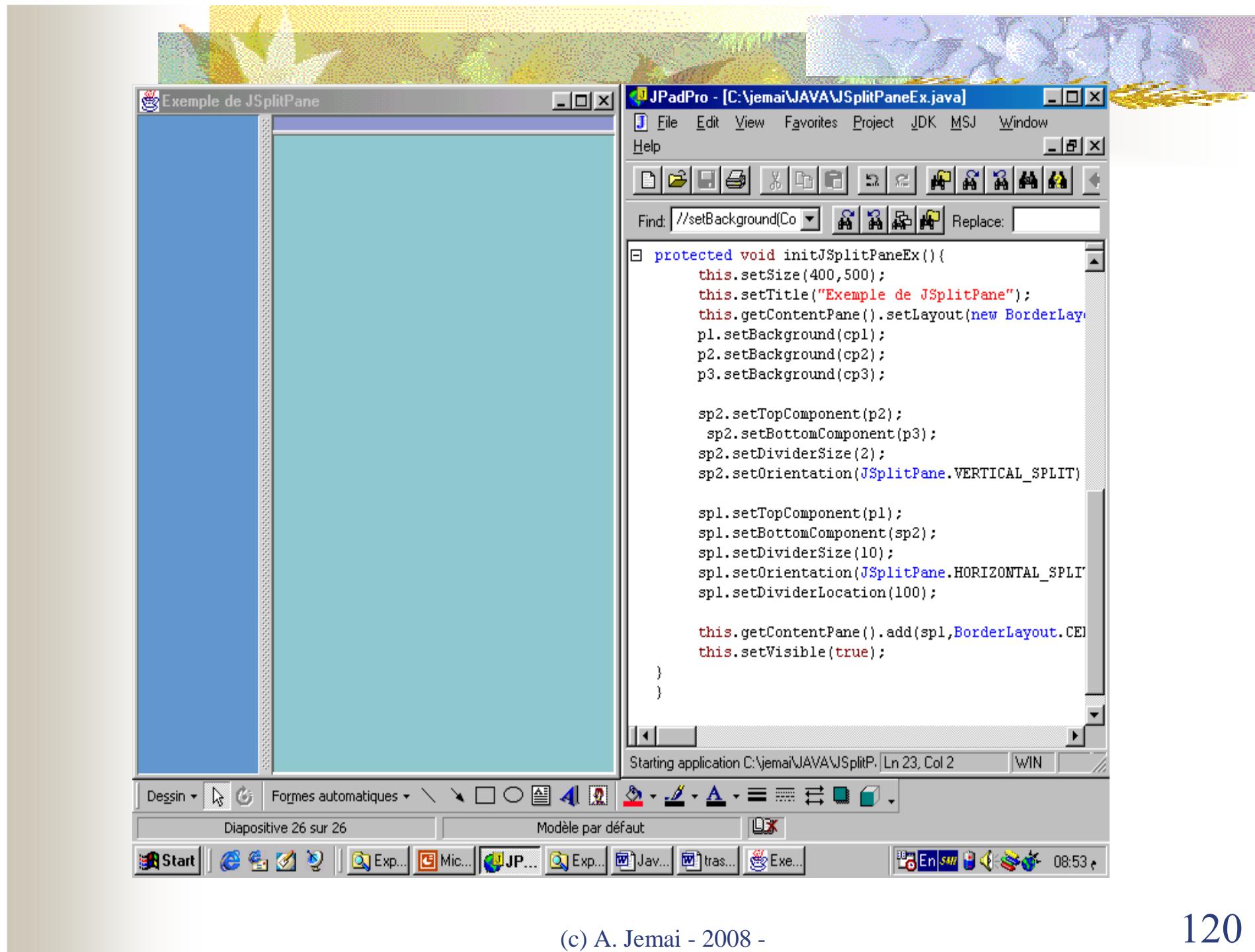
JSplitPane (suite)

```
protected void initJSplitPaneEx(){
    this.setSize(400,500);      this.setTitle("Exemple de JSplitPane");
    this.getContentPane().setLayout(new BorderLayout());
    p1.setBackground(cp1);      p2.setBackground(cp2);
    p3.setBackground(cp3);

    sp2.setTopComponent(p2);      sp2.setBottomComponent(p3);
    sp2.setDividerSize(2);
    sp2.setOrientation(JSplitPane.VERTICAL_SPLIT);

    sp1.setTopComponent(p1);      sp1.setBottomComponent(sp2);
    sp1.setDividerSize(10);
    sp1.setOrientation(JSplitPane.HORIZONTAL_SPLIT);
    sp1.setDividerLocation(100);

    this.getContentPane().add(sp1,BorderLayout.CENTER);
    this.setVisible(true);
}
```





JTabbedPane

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

class OngletEx extends JFrame
{
    JTabbedPane tabbedPane = new JTabbedPane();
    JPanel p1 = new JPanel();
    JPanel p2 = new JPanel();
    JPanel p3 = new JPanel();

    JTextArea textArea = new JTextArea();
    JLabel textAreaLabel = new JLabel();
    JButton benvoyer = new JButton();

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

    public OngletEx(){
        initOngletEx();
    }
```

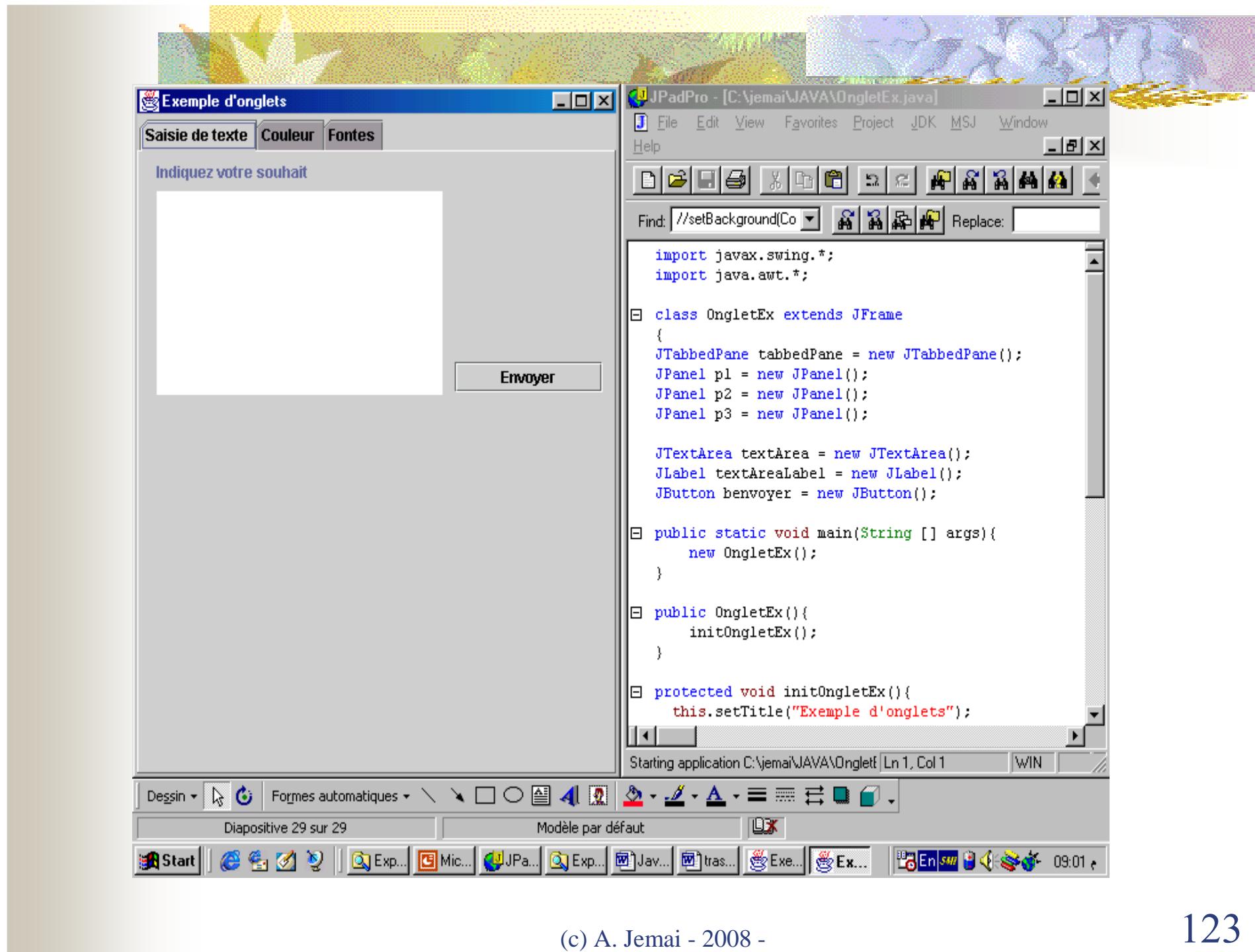


JTabbedPane (suite)

```
protected void initOngletEx(){
    this.setTitle("Exemple d'onglets");
    this.setSize(400,250);
    textArea.setBounds(14,29,234,154);
    textAreaLabel.setText("Indiquez votre souhait");
    textAreaLabel.setBounds(14,7,211,15);
    envoyer.setText("Envoyer");
    envoyer.setBounds(258,158,121,23);
    p1.setLayout(null);
    p1.add(textArea, null);
    p1.add(textAreaLabel, null);
    p1.add(envoyer,null);

    tabbedPane.addTab("Saisie de texte", p1);
    tabbedPane.addTab("Couleur", p2);
    tabbedPane.addTab("Fontes", p3);
    this.getContentPane().add(tabbedPane, BorderLayout.CENTER);
    this.setVisible(true);
}

}
```





JTable

```
import javax.swing.*;
import javax.swing.table.*;

import java.awt.event.*;
import java.awt.Dimension;

class JTableEx extends JFrame
{
    public JTableEx(){

        final Object[][][]lignes = {
            {new Integer(15), "Abderrazak", "JEMAI", "14-03-1963"},
            {new Integer(20), "Mohamed", "BEN SALAH", "12-10-1966"},
            {new Integer(25), "Sami", "Ben ALI", "15-03-1980"},
            {new Integer(30), "Fethi", "MAHJOUB", "21-10-1973}
        };

        final Object[] colonnesNom={"Nins", "Prénom", "Nom", "DN"};
    }
}
```

JTable (suite)

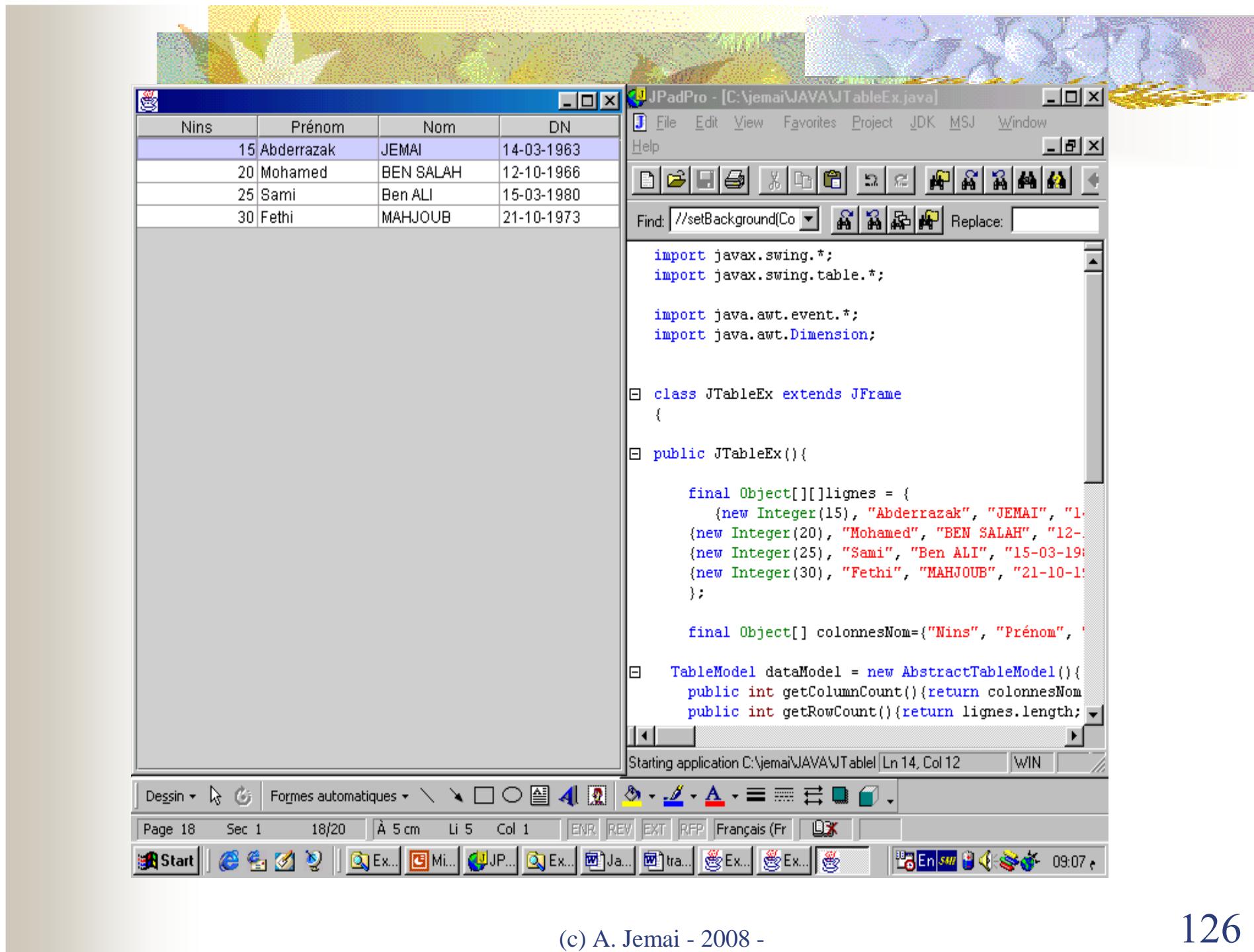
```
TableModel dataModel = new AbstractTableModel(){
    public int getColumnCount(){return colonnesNom.length;}
    public int getRowCount(){return lignes.length;}
    public Object getValueAt(int l, int c)
        {return lignes[l][c];}
    public String getColumnName(int col)
        {return (String)colonnesNom[col];}
    public Class  getColumnClass(int cl)
        {return getValueAt(0,cl).getClass();}

};// fin de définition de AbstractTableModel
JTable table = new JTable(dataModel);

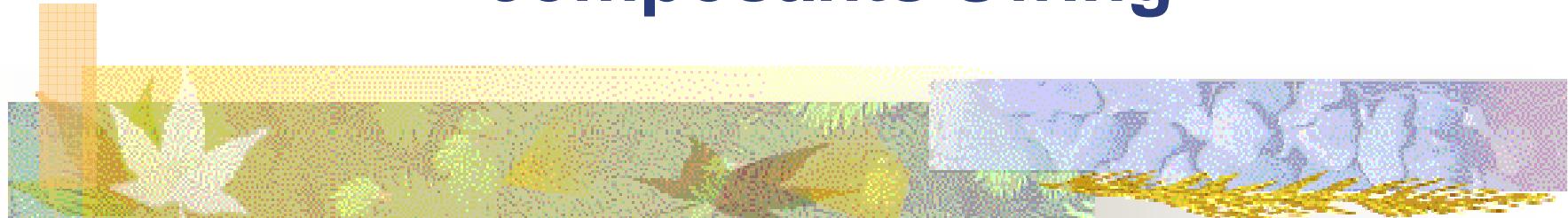
JScrollPane scrollpane = new JScrollPane(table);
scrollpane.setPreferredSize(new Dimension(500, 300));

this.getContentPane().add(scrollpane);
this.pack();
this.setVisible(true);
}

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



Gestion des Evenements avec les composants Swing



Evenement sur JButton

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

class SwingFrameEvent extends JFrame
{ JButton b1 = new JButton("Ouvrir");
 JButton b2 = new JButton("Sauvegarder");
 JButton b3 = new JButton("Quitter");

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

 public SwingFrameEvent(){
 initGUI();
 }

 public void initGUI(){
 b3.addActionListener(new AcLis(this));
 this.getContentPane().add(b1, BorderLayout.NORTH);
 this.getContentPane().add(b2, BorderLayout.WEST);
 this.getContentPane().add(b3, BorderLayout.EAST);
 this.setSize(new Dimension(500,400)); this.setVisible(true);
 }

 public void message(){
 System.out.println("Ceci est un message de SwingFrameEvent");
 }
}
```

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

class AcLis implements ActionListener
{
 SwingFrameEvent obj;

 public AcLis(SwingFrameEvent obj){
 this.obj = obj;
 }

 public void actionPerformed(ActionEvent e){
 obj.message();
 }
}
```

Evenements (suite)

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

class SwingFrameEvent2 extends JFrame
{ JButton b1 = new JButton("Ouvrir");
  JButton b2 = new JButton("Sauvegarder");
  JButton b3 = new JButton("Quitter");
  public static void main(String args[]){new SwingFrameEvent2();}
  public SwingFrameEvent2(){initGUI(); }

  protected void processWindowEvent(WindowEvent e){
    if (e.getID() == WindowEvent.WINDOW_CLOSING){ System.exit(0);}
  }
  public void initGUI(){
    b3.addActionListener(new AcLis2(this));
    this.getContentPane().add(b1,BorderLayout.NORTH);
    this.getContentPane().add(b2,BorderLayout.WEST);
    this.getContentPane().add(b3,BorderLayout.EAST);
    this.setSize(new Dimension(500,400));
    this.setVisible(true);
  }
  public void buttonPressed(ActionEvent e){
    String cmd = e.getActionCommand();
    if ("Quitter".equals(cmd)){ System.exit(0); }
  }
}
```

Chapitre V

Formulaire

et

Accès aux bases de données





Généralités

- Création d'un formulaire avec les composants AWT :
 - Label
 - TextField
 - Checkbox et CheckboxGroup
 - Choice
 - Panels
 - Boutons
 - Un mécanisme de gestion d'événements, ...
 - des *LayoutManager*
- Accès aux bases de données avec JDBC



Les composants AWT

■ Les composants AWT comportent :

- Button
- Checkbox
- Choice
- Label
- MenuBar
- MenuItem
- TextArea
- TextField

Exemple

Nom

Prénom

Adresse

Sexe

Masculin

Féminin

Filière

Clear Envoyer Quit



Approche méthodologique

1. Dessiner l'interface
2. Attribuer des noms à chaque composant
3. Identifier les types de chaque composant
4. Déclarer la classe qui hérite de java.awt.Frame
5. Déclarez le ou les constructeurs
6. Déclarez les composants comme variables membres.
7. Rajoutez les instructions dans le constructeur.



Application

Frame this

Label

INom

IPrenom

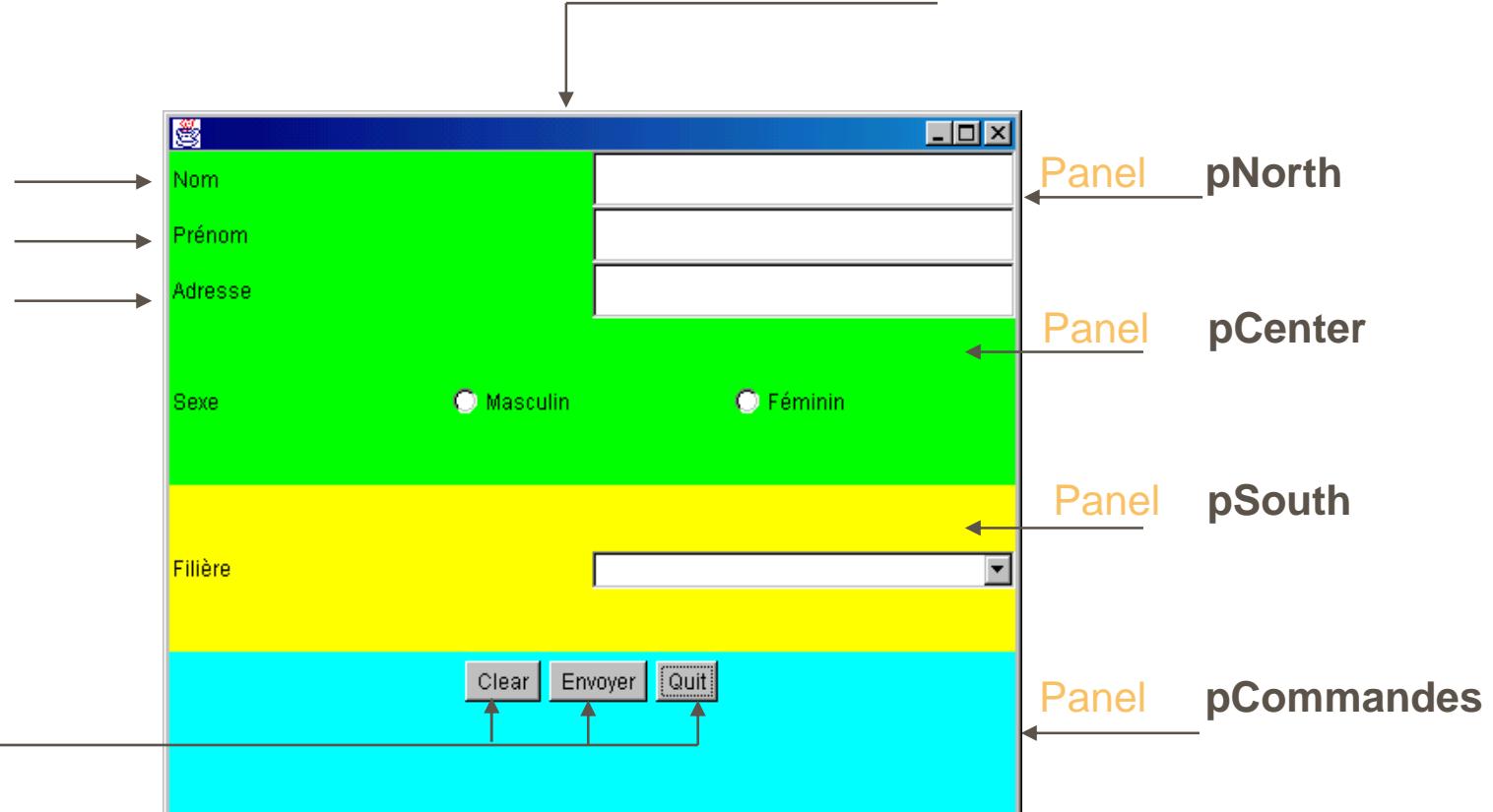
IAdresse

Button

bClear

bEnvoyer

bQuitter





Application

1. Le dessin étant effectué
2. Attribuer des noms à chaque composant

- Panel pNorth = new Panel();
- Label lNom = new Label("Nom");
- TextField tNom = new TextField(" ");
- Label lPrenom = new Label("Prénom");
- TextField tPrenom = new TextField(" ");
- Label lAdresse = new Label("Adresse");
- TextField tAdresse = new TextField(" ");
- Panel pCenter = new Panel();
- Label lSexe = new Label("Sexe");
- CheckboxGroup cbg = new CheckboxGroup();
- Checkbox cbMasculin = new Checkbox("Masculin", cbg, false);
- Checkbox cbFeminin = new Checkbox("Féminin", cbg, false);

Ces déclarations font l'objet des variables membres de la classe MyFrameDataBase



Classes

- Classe principale : **MyAppDataBase**
- Classe Frame : **MyFrameDataBase**
- Autres Classes pour la gestion des événements:
MyActionListenerForEnvoyer



Classes

■ Classe principale : **MyAppDataBase**

```
class MyAppDataBase
{
    public static void main (String args [ ])
    {
        System.out.println("Application : Accès aux
bases de données" );
        MyFrameDataBase f = new MyFrameDataBase( );
    }
}
```



Classe MyFrameDataBase

Variables Membres

- Classe Frame : **MyFrameDataBase**

```
import java.awt.*;  
class MyFrameDataBase extends Frame  
{  
    Panel pNorth = new Panel();  
    Label lNom = new Label("Nom");  
    TextField tNom = new TextField("");  
    Label lPrenom = new Label("Prénom");  
    TextField tPrenom = new TextField("");  
    Label lAdresse = new Label("Adresse");  
    TextField tAdresse = new TextField("");
```



Classe MyFrameDataBase (suite)

```
Panel pCenter = new Panel();
Label lSexe = new Label("Sexe");
CheckboxGroup cbg = new CheckboxGroup();
Checkbox cbMasculin = new Checkbox("Masculin", cbg, false);
Checkbox cbFeminin = new Checkbox("Féminin", cbg, false);
```

```
Panel pSouth = new Panel();
Label lFiliere = new Label("Filière");
Choice cFiliere = new Choice();
```

```
Panel pCommandes = new Panel();
Button bClear = new Button("Clear");
Button bEnvoyer = new Button("Envoyer");
Button bQuitter = new Button("Quit");
}
```



Classe MyFrameAWT - Constructeurs -

- Classe Frame : **MyFrameDataBase**

```
class MyFrameDataBase extends Frame
{
    ...
    // variables membres
    // Constructeurs
    Public MyFrameDataBase () {
        MyActionListenerForEnvoyer x2 = new
            MyActionListenerForEnvoyer(this);
        bEnvoyer.addActionListener(x2);

        pNorth.setBackground(Color.green);
        pNorth.setLayout(new GridLayout(3,2));
        pNorth.add(lNom);
        pNorth.add(tNom);
        pNorth.add(lPrenom);
        pNorth.add(tPrenom);
        pNorth.add(lAdresse);
        pNorth.add(tAdresse);
```



Classe MyFrameAWT - Constructeurs -

- Classe Frame : **MyFrameDataBase**

```
pCenter.setBackground(Color.green);
    pCenter.setLayout(new GridLayout(1,3));
    pCenter.add(lSexe);
    pCenter.add(cbMasculin);
    pCenter.add(cbFeminin);

pSouth.setBackground(Color.yellow);
    pSouth.setLayout(new GridLayout(1,2));
    pSouth.add(lFiliere);
    pSouth.add(cFiliere);

pCommandes.setBackground(Color.cyan);
    pCommandes.add(bClear);
    pCommandes.add(bEnvoyer);
    pCommandes.add(bQuitter);
```



Classe MyFrameAWT - Constructeurs -

- Classe Frame : **MyFrameDataBase**

```
this.setBackground(Color.blue);
this.setLayout(new GridLayout(4,1));
this.add(pNorth);
this.add(pCenter);
this.add(pSouth);
this.add(pCommandes);
this.setBounds(10,10,500,400);

MyWindowListener x1 = new MyWindowListener()
This.addWindowListener(x1);

this.setVisible(true);
}
}
```



Gestion des événements - Fermeture de la fenêtre -

- Gestion des événements : fermeture de la fenêtre
- Créer une nouvelle classe appelée MyWindowListener qui hérite de **WindowAdapter**

```
MyWindowListener x1 = new MyWindowListener()
This.addWindowListener(x1);
```

```
public class MyWindowListener extends WindowAdapter
{
    public void windowClosing(WindowEvent evt) {
        System.exit(0);
    }
}
```



Gestion des événements - Bouton Envoyer -

- Gestion des événements : Bouton Envoyer
- Créer une nouvelle classe appelée MyActionListenerForEnvoyer qui implémente **ActionListener**
- Rajouter dans le constructeur MyFrameDataBase() les deux lignes suivantes :

```
MyActionListenerForEnvoyer x2 = new MyActionListenerForEnvoyer ()  
bEnvoyer.addActionListener(x2);
```

```
public class MyActionListenerForEnvoyer implements ActionListener  
{  
    public void actionPerformed(ActionEvent e)  
    {  
        System.out.println("Bouton Envoyer actionné");  
    }  
}
```

Accès aux bases de données

- Bouton Envoyer -

- Classe cible : MyActionListenerForEnvoyer
- Méthode cible : ActionPerformed
- Approche :
 1. **Définir le nom de base de donnée cible**
 2. **Définir le nom de ou des tables cibles.**
 3. **Formuler la requête**
 4. Rattacher votre base de donnée au driver ODBC. Elle sera accessible via un nouveau nom (ex : MyDataBase).
 5. Définir l'URL : chemin d'accès à la base (ex : jdbc:odbc:MyDataBase).
 6. Faire une instance du driver d'accès à la base
 7. Faire une instance du mode d'accès (statement).
 8. Envoyer la requête.

Formulation de la requête

- Classe cible : MyActionListenerForEnvoyer
- Méthode cible : ActionPerformed
- Instructions :
 1. Formuler la requête conformément à la structure de la table et l'imprimer sur le System.out pour vérification.
 2. Il faut respecter la syntaxe SQL.
 3. Exemple : stockage de **nom** et **prenom** dans une table **etudiant** appartenant à une base de donnée **etudiants**.



Requête Statique

```
import java.awt.event.*;
import java.awt.*;
class MyActionListenerForEnvoyer implements ActionListener
{
    MyFrameDataBase f;
    public MyActionListenerForEnvoyer(MyFrameDataBase f)
    {
        this.f=f;
    }
    public void actionPerformed(ActionEvent e)
    {
        System.out.println("Bouton Envoyer actionné");
        String req= "INSERT INTO etudiant ([nom],[prenom]) VALUES
        ('JEMAI', 'Abderrazak');";
        System.out.println(req);
    }
}
```



Formulation de la requête

- Classe cible : MyActionListenerForEnvoyer
- Méthode cible : ActionPerformed
- Instructions :

1. Utiliser l'opérateur + pour dissocier les champs statique de la requête.

```
String req= "INSERT INTO etudiant ([nom],[prenom]) VALUES ('" + "  
JEMAI " + " ', ' " + " Abderrazak " + " ') ; " ;
```

2. Mettre chaque nom statique sur une ligne à part

```
String req= "INSERT INTO etudiant ([nom],[prenom]) VALUES  
('" +  
" JEMAI "  
+ " ', ' " +  
" Abderrazak "  
+ " ') ; " ;
```



Formulation de la requête

- Classe cible : MyActionListenerForEnvoyer
- Méthode cible : ActionPerformed
- Instructions :
 3. Identifier les méthodes pour lire les données à partir des champs.

```
f.tNom.getText();  
f.tPrenom.getText();
```
 4. Remplacer les données statiques par le nom des variables des champs suivis par les méthodes associées.
 5. Exemple

```
String req= "INSERT INTO etudiant ([nom],[prenom]) VALUES  
('" +  
f.tNom.getText() +  
",'" +  
f.tPrenom.getText() +  
  
+ "');"
```



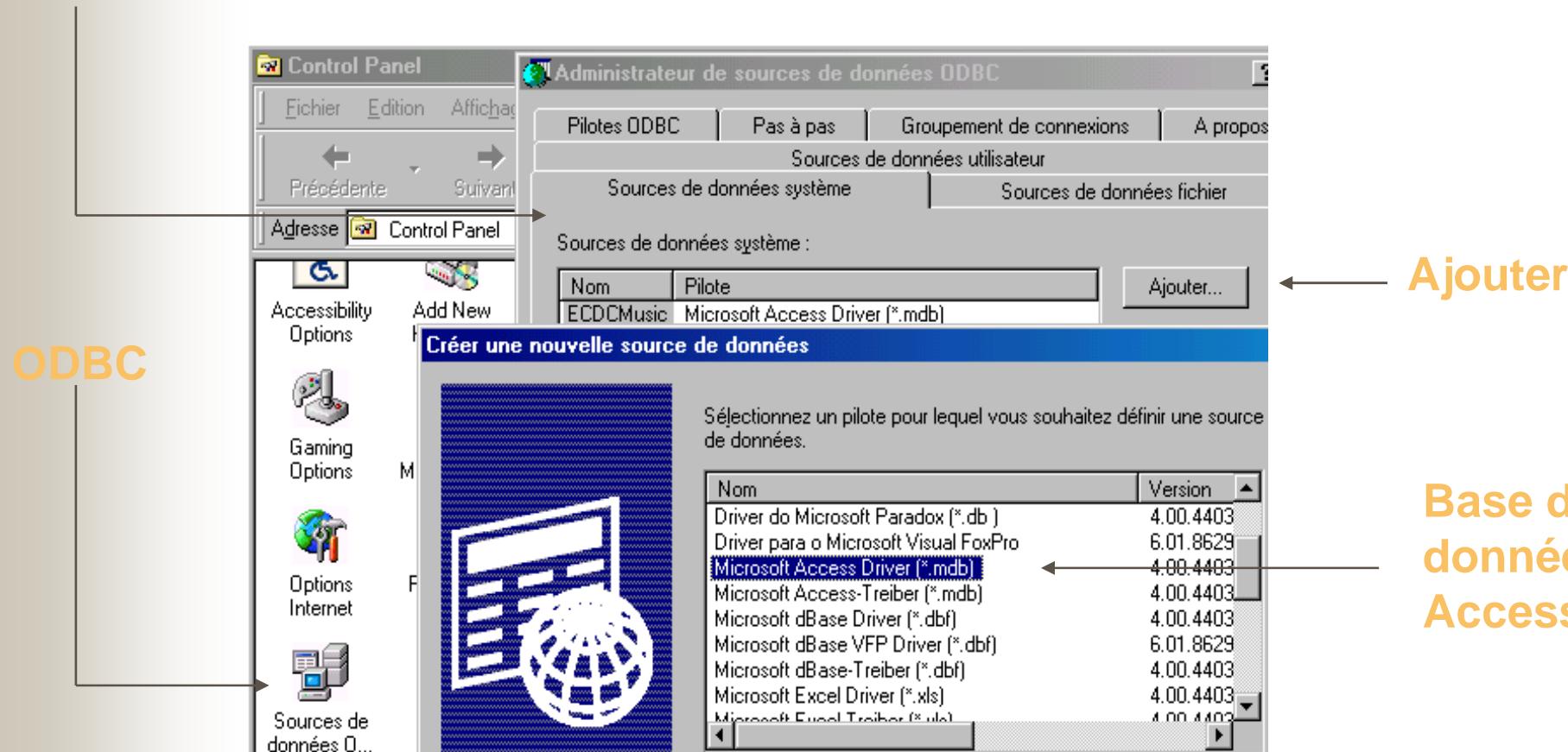
Accès aux bases de données - Bouton Envoyer -

- Classe cible : MyActionListenerForEnvoyer
- Méthode cible : ActionPerformed
- Approche :
 1. Définir le nom de base de donnée cible
 2. Définir le nom de ou des tables cibles.
 3. Formuler la requête
 4. **Rattacher votre base de donnée au driver ODBC. Elle sera accessible via un nouveau nom (ex : MyDataBase).**
 5. Définir l'URL : chemin d'accès à la base (ex : jdbc:odbc:MyDataBase).
 6. Faire une instance du driver d'accès à la base
 7. Faire une instance du mode d'accès (statement).
 8. Envoyer la requête.



ODBC (suite)

Source de données Système



Ajouter

Base de données Access

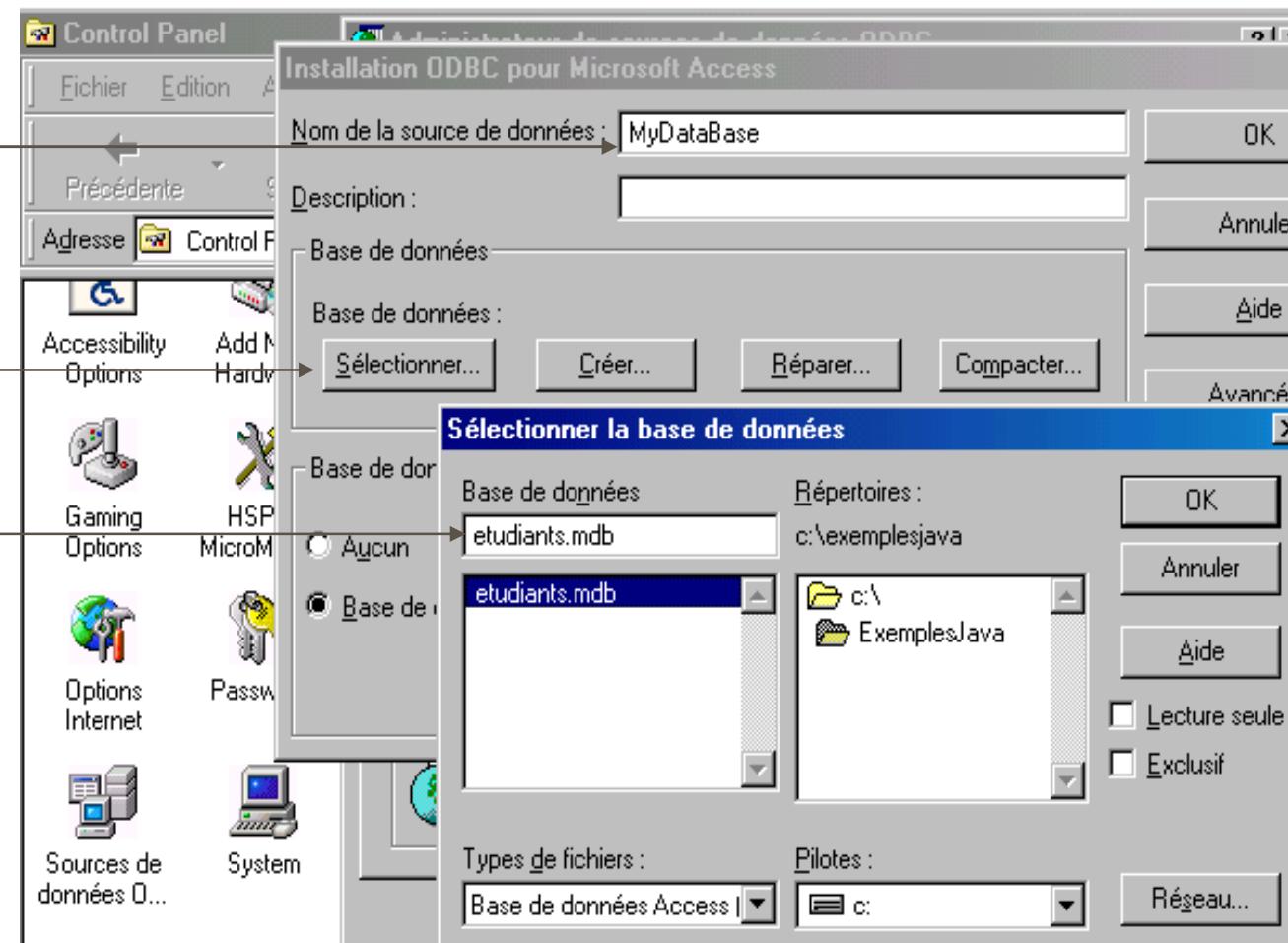
ODBC (suite)

Nom logique de la base

MyDataBase

Sélectionner

Etudiants.mdb





Accès aux bases de données - Bouton Envoyer -

- Classe cible : MyActionListenerForEnvoyer
- Méthode cible : ActionPerformed
- Approche :
 1. Définir le nom de base de donnée cible
 2. Définir le nom de ou des tables cibles.
 3. Formuler la requête
 4. Rattacher votre base de donnée au driver ODBC. Elle sera accessible via un nouveau nom (ex : MyDataBase).
 5. **Définir l'URL : chemin d'accès à la base (ex : jdbc:odbc:MyDataBase).**
 6. **Faire une instance du driver d'accès à la base**
 7. **Faire une instance du mode d'accès (statement).**
 8. **Envoyer la requête.**



Envoie de la requête

```
// suite du code
System.out.println(req);

String url = "jdbc:odbc:MyDataBase";

try
{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
    Connection conn = DriverManager.getConnection(url);
    Statement stmt = conn.createStatement();
    stmt.executeUpdate(req);

    System.out.println ("Bravo ! Vous avez réussi à ajouter un
enregistrement");

    stmt.close();
    conn.close();
}

}catch(Exception e4){System.out.println(e4);}
}
```



Solution complète

```
import java.awt.event.*;
import java.sql.*;
class MyActionListenerForEnvoyer implements ActionListener
{
MyFrameDataBase f;
public MyActionListenerForEnvoyer(MyFrameDataBase f)
{
    this.f=f;
}
public void actionPerformed(ActionEvent e)
{
System.out.println("Bouton Envoyer actionné");
String req= "INSERT INTO etudiant ([nom],[prenom]) VALUES ('"+
f.tNom.getText()
+', "'+
f.tPrenom.getText()
+')";';
System.out.println(req);
String url = "jdbc:odbc:'My DataBase'";

```



Solution Complète (suite)

```
try
{
    Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" ).newInstance();
    Connection conn = DriverManager.getConnection(url);
    Statement stmt = conn.createStatement();
    stmt.executeUpdate(req);
    System.out.println ("Bravo ! Vous avez réussi à ajouter
un enregistrement");
    stmt.close();
    conn.close();
} catch(Exception e4){System.out.println(e4);}
}
```



Accès aux base de donnée - Affichage des résultats -

- Composez la requête
`String req = SELECT nom, prenom FROM etudiant`
- Utilisez la méthode `executeQuery` de Statement pour résupérer les résultats
`ResultSet rs = stmt.executeQuery(req)`
- L'exécution de `rs.next()` permet d'aller vers l'enregistrement suivant.
- Exemple pour afficher toute la table utiliser

```
Boolean b=true;  
while (b)  
{  
    String nom = rs.getString("nom");  
    String prenom = rs.getString("prenom");  
    b=rs.next();  
  
}
```

Accès aux bases de données

- Exercice -

- Ecrire une application « MyVisualiseurDataBase » en java permettant de visualiser la table « etudiant » appartenant à la base « etudiants ».
- La base etudiants admet comme nom logique chez le driver odbc « MyDataBase ».
- Approche :
 - Une application principale : MyAppVisaliseurDataBase
 - Un Frame de visualisation : MyVisualiseurDataBase
 - Un formulaire pour afficher les données:
 - Nom
 - Prenom
 - Trois boutons principaux
 - Début : pour revenir au début de la base
 - Suivant : pour aller à l'enregistrement suivant
 - Quit : pour fermer la base et quitter l'application.

Résultat attendu

The screenshot shows a Windows application window titled "Résultat attendu". The window contains the following fields:

- Nom: JEMAI
- Prénom: Abderrazak
- Adresse: (empty)
- Sexe:
 - Masculin (radio button selected)
 - Féminin
- Filière: (dropdown menu)

At the bottom of the window, there are three buttons: "Début", "Suivant" (highlighted with a dotted border), and "Quit".



Chapitre 6 :Java-TCP/IP

- **Introduction à TCP/IP**
 - **Le protocole TCP**
 - **Le protocole UDP**
- **Les ports de communication**
- **Les Sockets**
 - **Définition**
 - **types de Sockets**
 - **Les classes réseau dans Java**
 - **Opérations sur les Sockets**
 - **Lire et écrire une Socket**



Introduction à TCP/IP

- ▶ Les ordinateurs qui fonctionnent dans l'environnement Internet communiquent par le protocole **TCP** (Transmission Control Protocol) ou **UDP** (User Datagram Protocol) et le protocole **IP** (Internet Protocol)
- ▶ Cette architecture de protocoles TCP/IP se présente selon un modèle en couches :



- Écrire des programmes Java : travailler au niveau **application**
- Pour écrire des programmes de communication sur le réseau, il faut utiliser les classes du package **java.net**

Le protocole TCP

- Lorsque deux machines veulent communiquer à travers un réseau de manière **fiable**, elle doivent suivre les étapes suivantes :
 - établir une connexion
 - communiquer
 - libérer la connexion
- Ces phases peuvent être réalisées grâce au protocole TCP
- Le protocole TCP assure la livraison **sûre** des données : contrôle d 'erreur
- ftp et telnet sont des exemples d 'applications qui nécessitent un transfert fiable

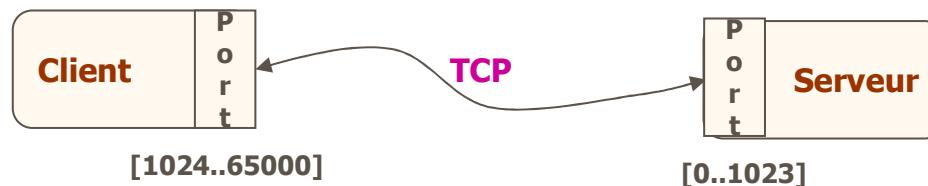
Le protocole UDP

- C 'est un service **non garanti** (best effort)
- Ce mode s 'adapte à certaines applications qui n 'ont pas des contraintes fortes sur la perte des données mais qui nécessitent le **maximum de débit disponible** (multimédia)



Les ports de communication

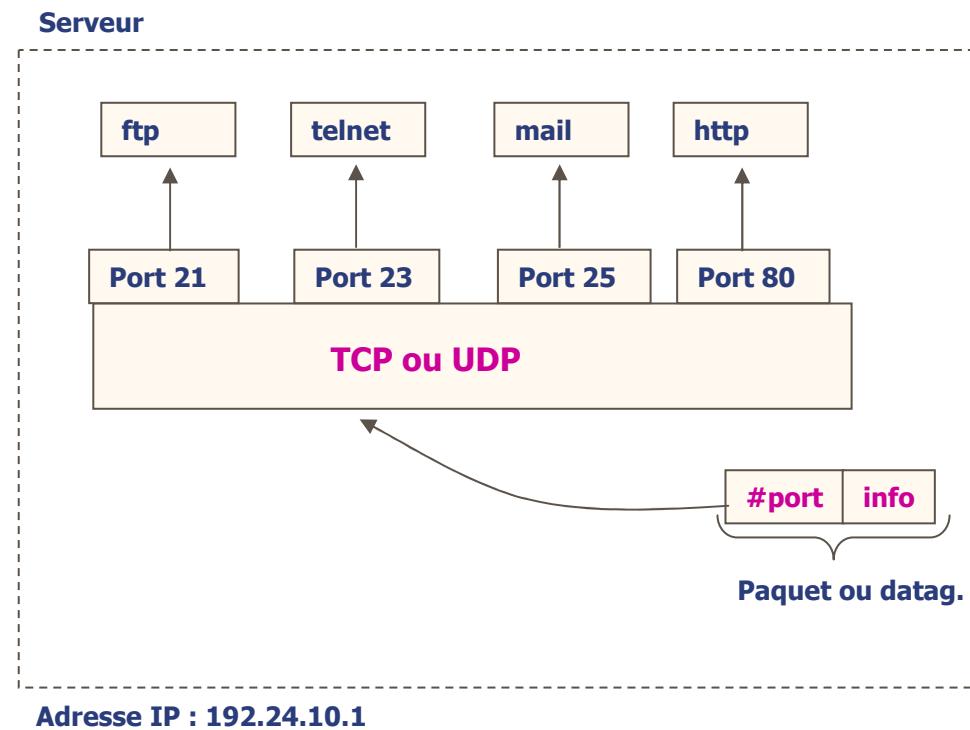
- **Un ordinateur est relié au réseau par un seul câble**
- **Toutefois, plusieurs applications peuvent tourner en parallèle sur cette machine : Chaque service (application) attend ses données sur un port.**
- **Un port est un numéro logique [0..6500]**
- **Les port [0..1023] sont des ports systèmes ou port serveur.**
- **Les port [1024 .. 6500] sont des ports applicatifs ou ports clients.**
- **Donc une requête client sort du porte client via un port client [1024..6500] et se loge chez le serveur dans le port serveur [0..1023].**
-





Les ports de communication

- Pour identifier un service [numéro de port] placé chez un serveur ayant une adresse IP, on utilise la notion de **Socket**.
- Une **Socket** est l'association <adresse IP, numéro de port>
- Un serveur offre plusieurs services (ex ftp, telnet, mail, http, etc.).

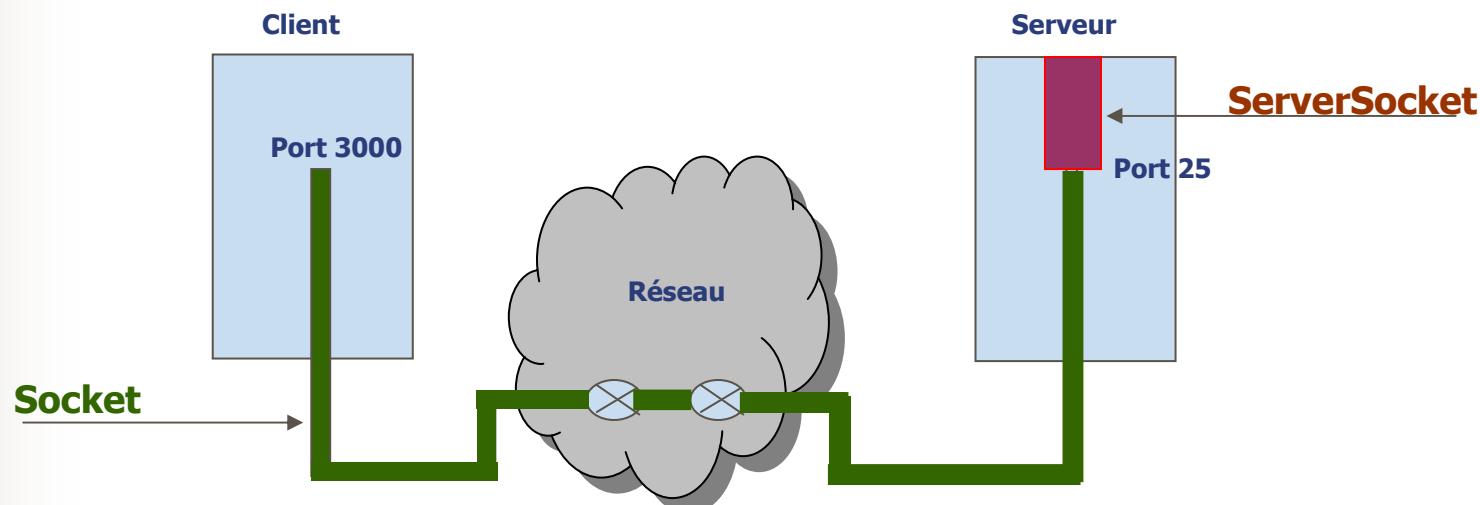


Les Sockets en Java

Définition

***Une Socket est une classe qui permet d'établir une connection entre une machine A et une machine B. On trouve deux types de classe Socket :
Socket (client) et
ServerSocket (serveur)***

- **Un serveur installe son service (application ou programme) en utilisant la classe ServerSocket.**
- **Un Client établit une connexion avec le serveur en utilisant la classe Socket.**





Les Sockets coté client : Socket

- Pour établir une connexion avec un serveur d'adresse IP et demander le service Port, le client exécute :
`new Socket(adresseIP, port)`
- ***Exemple : new Socket(192.168.24.1, 25)***
- Cet objet est assimilable à un canal de communication fiable du client vers le serveur



Les Sockets côté serveur : ServerSocket

- **Le serveur installe son application en utilisant une instance de la classe ServerSocket. Il a besoin que du numéro du port.**
L'instruction est :
- **new ServerSocket(port)**
- ***Exemple : new ServerSocket(25)***



Les Sockets coté Client : Socket

- La connexion avec le serveur étant établie, il fallait maintenant formuler la requête et l'envoyer via le canal (Socket). Ce Canal est assimilable à un dispositif d'entrée/sortie.

```
s=new Socket(args[0], port);
```

- Pour envoyer (émettre, write, etc.) on doit ouvrir le canal en mode « Write

```
DataOutputStream sout = new DataOutputStream(s.getOutputStream());
```

Ou bien:

```
PrintStream sout = new PrintStream(s.getOutputStream());
```

- Pour recevoir les réponses du serveur on doit ouvrir le canal en mode « read »

```
DataInputStream sin = new DataInputStream(s.getInputStream());
```

Solution Socket Client

```
import java.io.*;
import java.net.*;
class Client {
    public static void main (String[ ] args){
        int port = 6061;
        Socket s = null;
        try {
            s=new Socket(args[0], port);
            DataInputStream sin = new DataInputStream(s.getInputStream());
            PrintStream sout = new PrintStream(s.getOutputStream());
            System.out.println("connected to " + s.getInetAddress() +
                               ":" + s.getPort());
        }
    }
}
```

Solution Socket Client (suite...)

```
public static void main (String[] args){  
    // suite du code  
    String line ;  
    sout.println(" Bonjour "); // envoie du message BONJOUR  
    line=sin.readLine(); // attente de la réponse  
    if (line == null) {  
        System.out.println("Connection closed by server.");  
        break;  
    }  
    System.out.println(line);  
}  
}  
catch(IOException e) {System.err.println(e);}  
finally { try {if (s != null) s.close();}  
    catch (IOException e){}  
}  
}
```

- **Faire une instance de ServerSocket**
- **ServerSocket listen_socket = new ServerSocket(6061)**

- **Code complet de la création d'une socket :**

```
try { listen_socket = new ServerSocket(port); }
catch (IOException e)
{System.out.println("Exception creating server socket" + e);}
System.out.println("Server : listening... " + port);
```

Traitement d'une requête d'un client

Socket client_socket = listen_socket.accept();

- **Code complet de traitement d'une requête**

```
try{  
    Socket client_socket = listen_socket.accept();  
    DataInputStream in;  
    PrintStream out;  
    in = new DataInputStream(client.getInputStream());  
    out = new PrintStream(client.getOutputStream());  
}catch (Exception e)  
{ System.out.println("Exception while listening for... " +e);}
```



Les Sockets coté client code complet

```
import java.io.*;
import java.net.*;
class Client {
    public static final int DEFAULT_PORT = 6061;
    public static void usage(){
        System.out.println("Usage...Client <hostname> [<port>]");
        System.exit(0);
    }
    public static void main (String[ ] args){
        int port = DEFAULT_PORT;
        Socket s = null;

        if ((args.length != 1) && (args.length != 2)) usage();
        if (args.length == 1) port =DEFAULT_PORT;
        else{
            try { port = Integer.parseInt(args[1]);}
            catch (NumberFormatException e) {usage();}
        }
        try {
            s=new Socket(args[0], port);
            DataInputStream sin = new DataInputStream(s.getInputStream());
            PrintStream sout = new PrintStream(s.getOutputStream());
            sout.println("Hello from Client");
            sout.flush();
            String str=sin.readLine();
            System.out.println(str);
        } catch (IOException e) {e.printStackTrace();}
    }
}
```



Les Sockets côté client code complet (suite)

```
DataInputStream in = new DataInputStream(System.in);
System.out.println("connected to " + s.getInetAddress() +
                     ":" + s.getPort());
String line;
while(true) {
    System.out.println(">");
    System.out.flush();
    line = in.readLine();
    if(line == null) break;
    sout.println(line);
    line=sin.readLine();
    if (line == null) {
        System.out.println("Connection closed by
server.");
        break;
    }
    System.out.println(line);
}
}
catch(IOException e) {System.err.println(e);}
finally { try {if (s != null) s.close();} catch
(IOException e){}}
}
```



Les Sockets coté Serveur code complet

```
import java.io.*;
import java.net.*;
class Server extends Thread{
    protected int port;
    protected ServerSocket listen_socket;
    public static final int DEFAULT_PORT = 6061;
    // Termine le programme si une exception est levée.
    public static void fail(Exception e, String msg) {
        System.err.println(msg + ": " + e);
        System.exit(1);
    }

    // Crée un objet ServerSocket pour surveiller... et
    // active la tâche du serveur (Elle s'auto-active).
    public Server(int port){
        if (port == 0) port = DEFAULT_PORT;
        this.port = port;
        try { listen_socket = new ServerSocket(port);}
        catch (IOException e)
        {fail(e, "Exception creating server socket");}
        System.out.println("Server : listening... " + port);
        this.start();
    }
}
```



Les Sockets coté Serveur code complet

```
public void run(){
    try{
        while (true) {
            Socket client_socket = listen_socket.accept();
            Connection c = new Connection(client_socket);
        }
    }catch (IOException e)
        { fail(e, "Exception while listening for...");}
    }
public static void main(String[] args) {
    int port = 0;
    if (args.length == 1){
        try { port = Integer.parseInt(args[0]);}
        catch (NumberFormatException e) {port = 0;}
    }
    new Server(port);
}
}// fin de la classe Server
```

```
// La classe ci-dessous Connection est une tâche gérant toutes les
// communications avec un client
class Connection extends Thread{
    protected Socket client;
    protected DataInputStream in;
    protected PrintStream out;

    // Initialise le flot et s'auto-active
    public Connection(Socket client_socket){
        client = client_socket;
        try { in = new
DataInputStream(client.getInputStream());
            out = new
PrintStream(client.getOutputStream());
        }
        catch (IOException e){
            try {client.close();} catch (IOException e2) {}
            System.err.println("Exception while....stream : " +
e);
            return;
        }
        this.start();
    }
}
```



Les Sockets coté Serveur code complet

```
// Fournit les services du serveur
// Lit une ligne, l'inverse et la renvoie au client
public void run(){
    String line;
    StringBuffer revline;
    int len;
    try {
        for(;;) {
            line = in.readLine(); // Lit une line
            if (line == null) break;
            // L'inverse
            len = line.length();
            revline = new StringBuffer(len);
            for (int i = len - 1; i >= 0; i--)
                revline.insert(len - 1 - i, line.charAt(i));
            // Renvoie la ligne inversée
            out.println(revline);
        }
    }catch (IOException e) {}
    finally {
        try {client.close(); }
        catch (IOException e2) {}
    }
}
```