
JAVA

Abdellah EZZATI

Introduction & Historique

- Java trouve ses origines dans les années 1990. Les premières versions de Java datent de 1995.
- Java est développé par Sun Microsystems. Il est fortement inspiré des langages C et C++.
- Java est multi plates-formes : les programmes tournent sans modification sur Windows, Unix et Mac OS. Portable grâce à l'exécution par une machine virtuelle.
- Java est presque un pur langage Orienté Objet:
 - Un programme Java est constitué uniquement de classes
 - Les types primitifs : int, float... qui ne sont pas des objets

Introduction & Historique

- Java est doté, en standard, de bibliothèques de classes comprenant la gestion des interfaces graphiques (fenêtres, boîtes de dialogue, menus, graphisme), la programmation multi-threads (multitâches), la gestion des exceptions, les accès aux fichiers et au réseau ...
- Contrairement au C/C++, le compilateur Java ne génère pas de fichier exécutable. Il crée pour chacune des classes (tp.java) d'un fichier Java un fichier .class (tp.class) qui sera interprété par la Machine Virtuelle Java.
- Ensemble d'outils (java, javac, jdb, javadoc, jar...)

Introduction & Historique

- En Java, tout se trouve dans une classe. Il ne peut y avoir de déclarations ou de code en dehors du corps d'une classe.
- La classe elle même ne contient pas directement du code.
Elle contient des déclarations de variables globales, que l'on appelle des «attributs» et des méthodes.
- Le code se trouve exclusivement dans le corps des méthodes, mais ces dernières peuvent aussi contenir des déclarations de variables locales (visibles uniquement dans le corps de la méthode).
- Java développe deux genres de programmes :
 - Les applications (programmes ou logiciels classiques).
 - Les applets qui sont des programmes java insérés dans un document HTML.

Introduction & Historique

- **JRE**: Java Runtime Environment. Désigne la machine virtuelle et l'ensemble des outils nécessaires pour exécuter une application Java sur votre machine.
- **JDK**: Java Development Kit. Ancien terme SDK.
- **SDK**: Standard Development Kit ↔ L'ensemble d'outils nécessaires au développeur Java. Celui-ci contient:
 - JRE
 - Les bibliothèques de développement
 - Le compilateur
- IDE:
 - Eclipse
 - Netbeans
 - Jbuilder
 - IntelliJ

Mon premier programme en Java

Soit le programme suivant qui affiche à l'écran :

Bonjour c'est mon premier programme en Java



TP1.java

```
public class TP1 {  
    // déclaration de données globales  
    //définition des méthodes  
    public static void main(String args[]) {  
        // déclaration des données locales;  
        System.out.println("Bonjour voici mon premier programme en Java");  
    }  
}
```

Mon premier programme en Java

→ Un programme autonome destiné à être exécuté doit contenir une méthode particulière nommée **main()** définie de la manière suivante:

```
public static void main(String args[]) {  
    /* corps de la méthode main */  
}
```

→ Le paramètre **args** de la méthode **main()** est un tableau d'objets de type **String**. Il n'est pas utilisé mais, il est exigé par le compilateur Java.

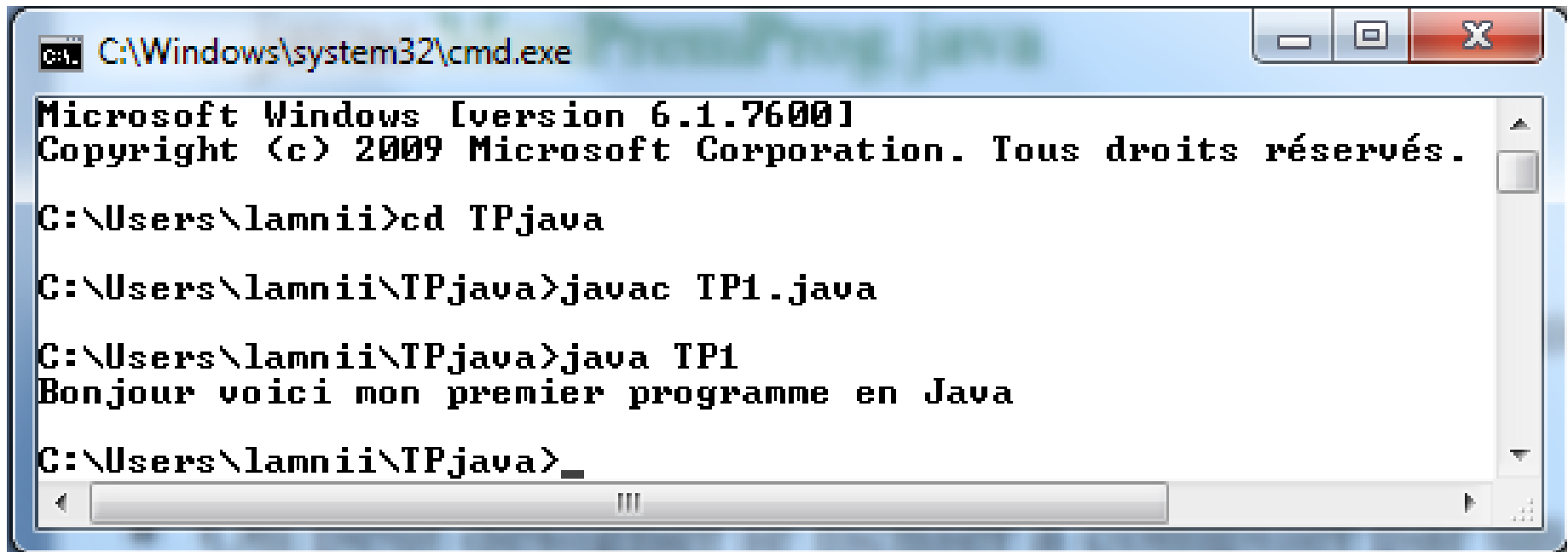
→ La classe contenant la méthode **main()** doit obligatoirement être **public** afin que la machine virtuelle y accède. Notre premier exemple **TP1** est réduit à la définition d'une méthode **main()**.

Mon premier programme en Java

- Un fichier source peut contenir plusieurs classes mais une seule doit être public (Dans notre premier programme par exemple TP1 est public).
- Le nom du fichier source est identique au nom de la classe publique qu'il contient, suivi du suffixe .java.
- Dans l'exemple : **TP1.java**

Compilation & Exécution

La compilation d'un programme Java ne traduit pas directement le code source en fichier exécutable. Elle traduit d'abord le code source en un code intermédiaire appelé « **bytecode** ». C'est le bytecode qui sera ensuite exécuté par une machine virtuelle (JVM ; Java Virtual Machine). Ceci permet de rendre le code indépendant de la machine qui va exécuter le programme.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Tous droits réservés.

C:\Users\lamnii>cd TPjava

C:\Users\lamnii\TPjava>javac TP1.java

C:\Users\lamnii\TPjava>java TP1
Bonjour voici mon premier programme en Java

C:\Users\lamnii\TPjava>_
```

Java : Eléments de bases

- Les identificateurs commencent par une lettre ou par _
- Les mots-clés et les signes (-, +, @, ...) sont interdits dans les identificateurs qui soient des attributs ou des méthodes.
- Un identificateur ne commence jamais avec un chiffre.
- Java distingue entre minuscules et majuscules
- Les instructions terminent par un point-virgule ;
- Les blocs sont délimités par un couple d'accolades {...}
- Les caractères // définissent un commentaire s'étend jusqu'à la fin de ligne.
- /* commentaires sur plusieurs lignes */

Identificateurs et mots réservés

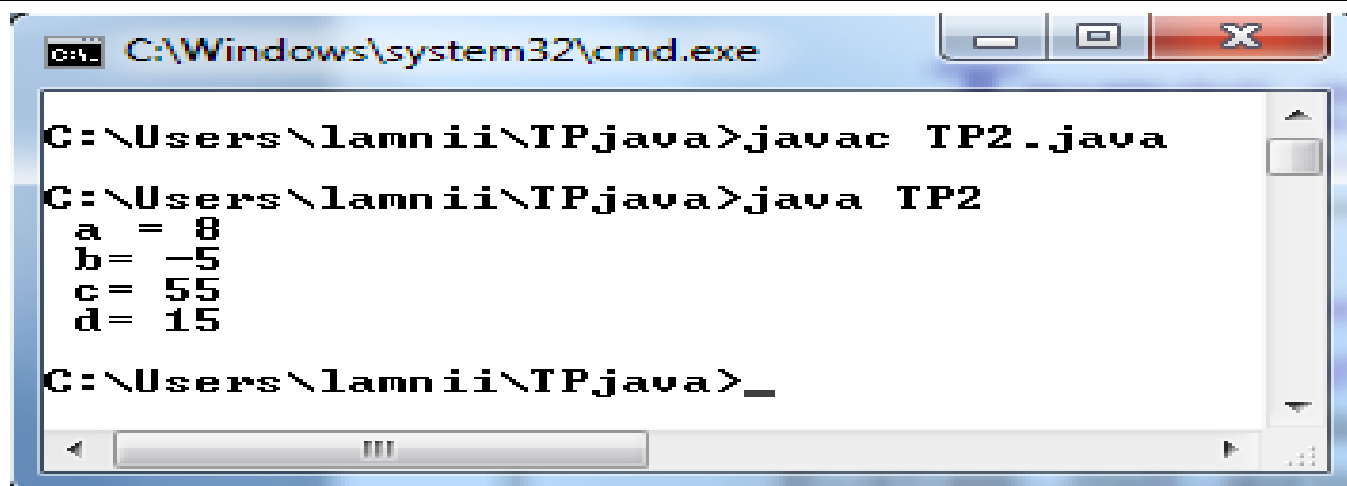
| | | | | |
|----------|----------|------------|-----------|--------------|
| abstract | continue | for | new | switch |
| assert | default | goto | package | synchronized |
| boolean | do | if | private | this |
| break | double | implements | protected | throw |
| byte | else | import | public | throws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

Eléments de bases : types entiers

| Type | Taille (oct) | Valeur minimale | Valeur maximale |
|-------|--------------|---------------------|---------------------|
| byte | 1 | -128 | 127 |
| short | 2 | -32 768 | 32 767 |
| int | 4 | -2 147 483 648 | 2 147 483 647 |
| long | 8 | -923372036854775808 | -923372036854775807 |

Types entiers : Exemple

```
public class TP2 {  
    public static void main(String args[]) {  
        int a=8;    byte b=-5; short c=55; long d=15;  
        System.out.println(" a = " + a);  
        System.out.println(" b= " + b);  
        System.out.println(" c= " + c);  
        System.out.println(" d= " + d);  
    }  
}
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The prompt is at "C:\Users\lannii\TPjava>". The user has entered "javac TP2.java" and "java TP2". The output of the program is displayed as follows:

```
C:\Users\lannii\TPjava>javac TP2.java  
C:\Users\lannii\TPjava>java TP2  
a = 8  
b = -5  
c = 55  
d = 15  
C:\Users\lannii\TPjava>_
```

Eléments de bases : types réels

| Type | Taille(oct) | Précision | Valeur minimale | Valeur maximale |
|--------|-------------|-----------|-------------------------|-----------------------|
| float | 4 | 7 | -1.40239846E-45 | 3.40282347E38 |
| double | 8 | 15 | 4.9406564584124654E-324 | 1.797693134862316E308 |

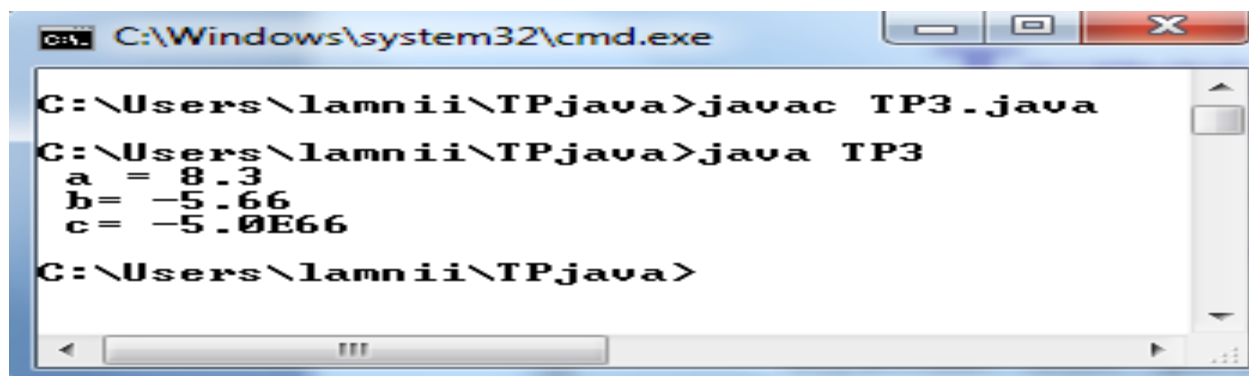
f ou **d** après une valeur numérique spécifier le type.

double a = 3.14d ;

double b = 3.14f ;

Types réels : Exemple

```
public class TP3 {  
    public static void main(String args[]) {  
        float a=8.3f;    double b=-5.66; double c=-5e66;  
        System.out.println(" a = "+ a);  
        System.out.println(" b= " + b);  
        System.out.println(" c= " + c);  
    }  
}
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The user has navigated to the directory "C:\Users\lamnii\TPjava". The commands and their outputs are as follows:

```
C:\Users\lamnii\TPjava>javac TP3.java  
C:\Users\lamnii\TPjava>java TP3  
a = 8.3  
b= -5.66  
c= -5.0E66  
C:\Users\lamnii\TPjava>
```

Eléments de bases : types caractère

| Notation | Code Unicode | Abréviation usuelle | Signification |
|----------|--------------|---------------------------|------------------------|
| \b | 0008 | BS(BackSpace) | Retour arrière |
| \t | 0009 | HT(Horizontal Tabulation) | Tabulation horizontale |
| \n | 000a | LF(LineFeed) | Saut de ligne |
| \f | 000c | FF(FormFeed) | Saut de page |
| \r | 000d | CR(CariageReturn) | Retour chariot |
| \" | 0022 | | |
| \' | 0027 | | |
| \\ | 005c | | |

Java permet de manipuler des caractères

Java représente les caractères sur 2 octets

Types caractères: Exemple

```
public class TP4 {  
    public static void main(String args[]) {  
        char a='A';      char b='T';  
        System.out.println(" a = " + a);  
        System.out.println(" b = " + b);  
    }  
}
```

Constante et mot clé final

Java permet de déclarer des variables et des expressions dont la valeur est inchangeable en utilisant le mot clé **final**.

```
public class TP5 {  
    public static void main(String args[]) {  
        final int a=5;          final int b= 2*a-7;  
        System.out.println(" a = "+ a);  
        System.out.println(" b = " + b);  
        // a=a+3; erreur  
        // b=2*b; erreur  
    }  
}
```

Lecture au clavier

L'affichage (Java) ne pose pas de problème on peut toujours utiliser des instructions telles que :

```
System.out.println(" valeur = " + a+b); //Concaténation
```

Ou bien:

```
System.out.print(" valeur = " + (a+b)); //Calcul
```

Mais pour la lecture java ne prévoit rien. Mais, il existe deux solutions (voir www.editions-eyrolles.com) :

➔ Clavier.java

➔ Scanner

Clavier.java

```
import java.io.* ;
public class Clavier
{ public static String lireString ()    // lecture d'une chaine
  { String ligne_lue = null ;
    try{InputStreamReader lecteur = new InputStreamReader(System.in);
      BufferedReader entree = new BufferedReader (lecteur) ;
      ligne_lue = entree.readLine() ;      }
    catch (IOException err)
      { System.exit(0) ;      }      return ligne_lue ;    }
  public static float lireFloat ()    // lecture d'un float
  { float x=0 ;    // valeur a lire
    try { String ligne_lue = lireString() ;
      x = Float.parseFloat(ligne_lue) ;      }
    catch (NumberFormatException err)
      { System.out.println ("*** Erreur de donnee ***") ;
        System.exit(0) ;      }
    return x ;
  }
}
```

Clavier.java (suite)

```
public static double lireDouble ()    // lecture d'un double
{ double x=0 ;    // valeur a lire
  try { String ligne_lue = lireString() ;
    x = Double.parseDouble(ligne_lue) ;    }
  catch (NumberFormatException err)
  { System.out.println ("*** Erreur de donnee ***") ;
    System.exit(0) ;    }
  return x ;    }

public static int lireInt ()          // lecture d'un int
{ int n=0 ;    // valeur a lire
  try    { String ligne_lue = lireString() ;
    n = Integer.parseInt(ligne_lue) ;    }
  catch (NumberFormatException err)
  { System.out.println ("*** Erreur de donnee ***") ;
    System.exit(0) ;    }
  return n ;    }
}
```

Exemple de lecture (Clavier)

```
public class TP6 {  
    public static void main(String args[]) {  
        int a; float b; String c;  
        System.out.print(" Veuillez saisir un entier ");  
        a=Clavier.lireInt() ;;  
        System.out.println(" a = " + a);  
        System.out.print(" Veuillez saisir un reel ");  
        b=Clavier.lireFloat();  
        System.out.println(" b = " + b);  
        System.out.print(" Veuillez saisir un caractere "); // ou chaine  
        c=Clavier.lireString();  
        System.out.println(" c = " + c);  
        System.out.println(" valeur = " + a+b); //Concaténation  
        System.out.print(" valeur = " + (a+b)); //Calcul  
    }  
}
```

C:\Windows\system32\cmd.exe

```
C:\Users\lannii\TPjava>javac Clavier.java
```

```
C:\Users\lannii\TPjava>javac TP6.java
```

```
C:\Users\lannii\TPjava>java TP6
```

```
  Veuillez saisir un entier 2
```

```
  a = 2
```

```
  Veuillez saisir un reel 3.56
```

```
  b = 3.56
```

```
  Veuillez saisir un caractere ok
```

```
  c = ok
```

```
  valeur = 23.56
```

```
  valeur = 5.56
```

Exemple de lecture (Scanner)

```
import java.util.Scanner;
public class TP7 {
    public static void main(String args[]) {
        Scanner alire = new Scanner(System.in);
        int a; float b; String c;
        System.out.print(" Veuillez saisir un caractere ");
        c=alire.nextLine();
        System.out.println(" c = " + c);
        System.out.print(" Veuillez saisir un entier ");
        a=alire.nextInt();
        System.out.println(" a = " + a);
        System.out.print(" Veuillez saisir un reel ");
        b=alire.nextFloat();
        System.out.println(" b = " + b);
    }
}
```


Les opérateurs arithmétiques

| <i>symbole</i> | <i>description</i> | <i>Nombre d'opérandes</i> | <i>exemple</i> |
|----------------|-----------------------|-------------------------------|----------------|
| $-$ | <i>soustraction</i> | <i>2</i> | $a - b$ |
| $+$ | <i>addition</i> | <i>2</i> | $a + b$ |
| $*$ | <i>multiplication</i> | <i>2</i> | $a * b$ |
| $/$ | <i>division</i> | <i>2</i> | a / b |
| $\%$ | <i>modulo</i> | <i>2</i> | $a \% b$ |

En Java l'opérateur d'exponentiation n'existe pas : pour calculer a^b on utilise la fonction prédéfinie :

`Math.pow(a, b)`

Opposé de a est $-a$.

```
public class TP8 {  
    public static void main(String args[]) {  
        double a=4, b=2;  
        System.out.println( a + " + " + b + " = " + (a+b));  
        System.out.println( a + " - " + b + " = " + (a-b));  
        System.out.println( a + " / " + b + " = " + (a/b));  
        System.out.println( a + " * " + b + " = " + (a*b));  
        System.out.println( a + " % " + b + " = " + (a%b));  
        System.out.println( a + " ^ " + b + " = " + Math.pow(a,b));  
    }  
}
```

Les opérateurs relationnelles

| Opérateur | Signification |
|-----------|----------------------|
| < | Inférieur à |
| <= | inférieur ou égale à |
| > | Supérieur à |
| >= | Supérieur ou égale à |
| == | Égal à |
| != | différent |

Les opérateurs logiques

| Opérateur | Signification |
|-----------|--------------------------------|
| ! | Négation |
| & | Et |
| ^ | ou exclusif |
| | Ou inclusif |
| && | Et avec court circuit |
| | Ou inclusif avec court circuit |

Les opérateurs d'incrémentation / décrémentation

| <i>symbole</i> | <i>description</i> | <i>Nb opérande</i> | <i>exemple</i> |
|----------------|---------------------------|--------------------|----------------|
| <i>++</i> | <i>préincrémentation</i> | <i>1</i> | <i>++i</i> |
| <i>++</i> | <i>postincrémentation</i> | <i>1</i> | <i>i++</i> |
| <i>--</i> | <i>prédécrémentation</i> | <i>1</i> | <i>--i</i> |
| <i>--</i> | <i>postdécrémentation</i> | <i>1</i> | <i>i--</i> |

Les opérateurs d'affectation élargie

| <i>symbole</i> | <i>exemple</i> | <i>symbole</i> |
|----------------|----------------|---------------------------|
| $+=$ | $i+=4$ | $\wedge=$ |
| $-=$ | $i-=4$ | $\&=$ |
| $*=$ | $i*=4$ | $<<=$ (décalage) $>>=$ |
| $/=$ | $i/=4$ | $>>>=$ (décalage) |

Instructions de contrôle

Instruction `if`

Syntaxe:

`if (condition)`

`bloc_instruction1`

`else`

`bloc_instruction2`

Instructions de contrôle

instruction `switch`

Syntaxe:

`switch(expression)`

`{case constante1 : bloc d'instructions1; break;`

`case constante2 : bloc d'instructions2; break;`

`.....`

`case constanteN : bloc d'instructionsN; break;`

`default: bloc d'instructions;`

`}`

Instructions de contrôle

instruction `for`

Syntaxe:

```
for(initialisation; condition; incrémentation)
    { bloc d'instructions; }
```

Exemple:

```
for(i=1;i<=3;i++)
    system.out.println(i+"", "");
```

Résultat: 1, 2, 3,

Instructions de contrôle

instruction `while`

Syntaxe:

```
while (condition)
    { bloc d'instructions; }
```

Exemple

```
i=1;
while(i<=3)
{ system.out.println(i+" ", "");
  i++; }
```

Résultat: 1, 2, 3,

Instructions de contrôle

instruction `do ... while`

Syntaxe:

`do { bloc d' instructions; } while (condition);`

Exemple

```
i=1;  
do  
{system.out.println(i+"", "");  
i++;  
} while(i<=3);
```

Résultat: 1, 2, 3,

Exercice

Calcul de $N!$ avec N lu au clavier

1- utiliser for

2- utiliser while

3- utiliser do ... while

Instruction de branchement inconditionnel

Dans une boucle (while, do...while, for) `break` sert à interrompre l'exécution de cette boucle.

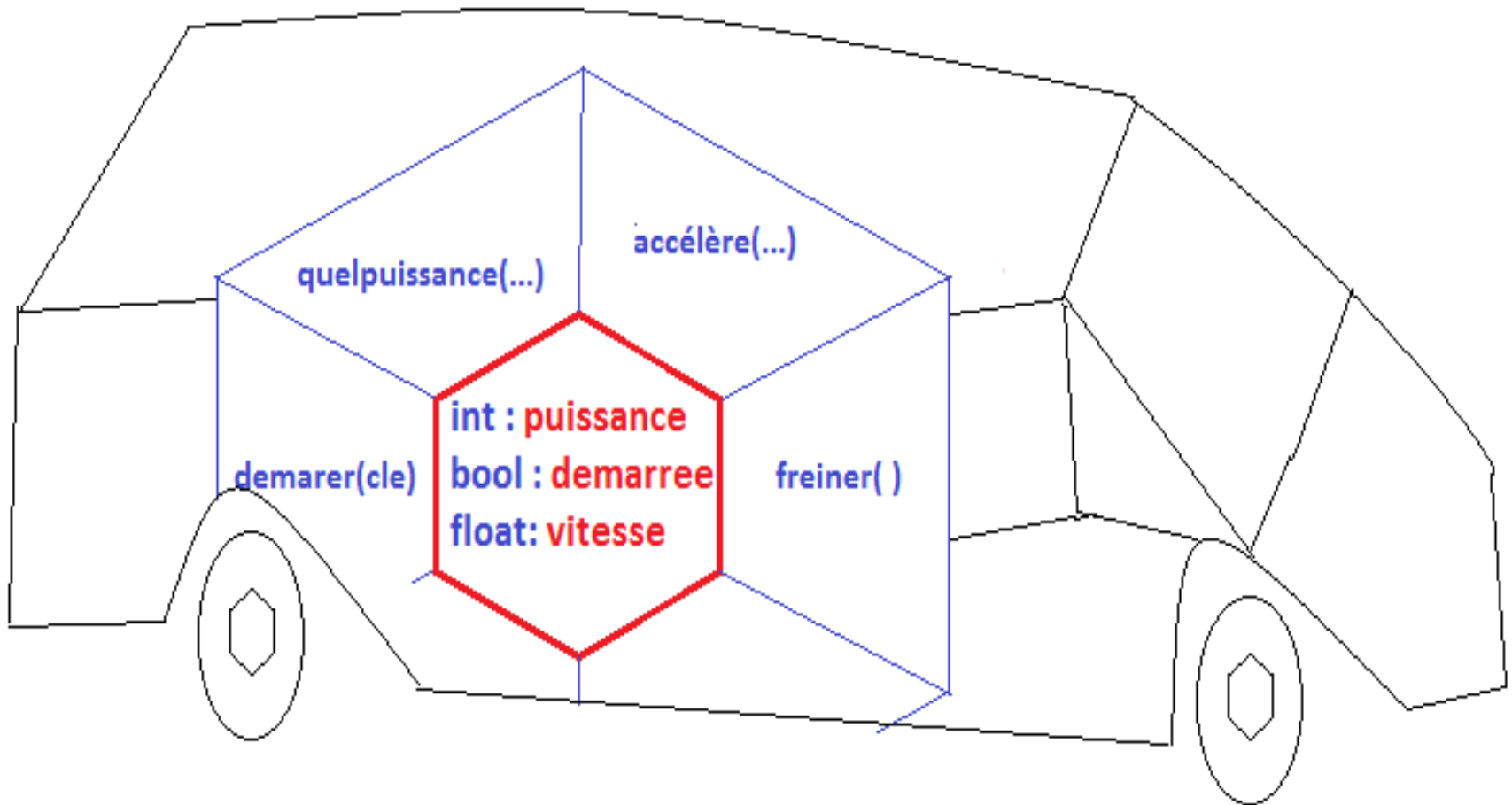
```
import java.util.Scanner;
public class TP9 {
    public static void main(String args[]) {
        Scanner alire = new Scanner(System.in);
        int i;
        for(i=-3;i<=3;i++)
            {if(i==0) { break; }
            System.out.println("1/" + i + " = " + (1/(double)i)); }
    }
}
```

Instruction de branchement inconditionnel

L'instruction `continue` permet de sauter des instructions en changeant directement le compte de la boucle.

```
public class TP10 {  
    public static void main(String args[]) {  
        int i;  
        for(i=-3;i<=3;i++)  
            {if(i==0) { continue; }  
            System.out.println("1/" + i + " = " + (1/(double)i)); }  
        }  
    }
```

Notion de Classe & d'Objet



classe voiture

```
public class Voiture
```

```
{
```

```
private int puissance;  
private boolean demarree;  
private float vitesse;
```

attributs

constructeur

```
public Voiture(int arg1, boolean arg2, float arg3)  
{ puissance=arg1; demarree=arg2; vitesse=arg3; }
```

```
public boolean demarrer(char cle)  
{ if(cle=='o')  
  { demarree=true;  
    return true; }  
else  
  { demarree=false;  
    return false; }  
}
```

méthode damarrer

```
public float quelpuissance(int preOudeuOutr) // 1, 2,  
{ float vv=(float)preOudeuOutr;  
  if(this.demarree==true)  
    return (vitesse*vv);  
  else  
    return 0.0f;  
}
```

méthode quelpuissance

```
public static void main(String args[])  
{ Voiture BM = new Voiture(1, false, 22.0f) ;  
  System.out.println("demarree= "+BM.demarrer('o'));  
  System.out.println("puissance = "+BM.quelpuissance(2));  
  System.out.println("demarree= "+BM.demarrer('n'));  
  System.out.println("puissance = "+BM.quelpuissance(2));  
}
```

méthode main

```
}
```

Attributs → Variable global de la classe

→ Accessible depuis toutes les méthodes de la classe

Méthode → Les méthodes ne peuvent être créées que comme des composantes d'une classe.

→ Une méthode ne peut être appelée que pour un objet.

→ L'appel d'une méthode pour un objet se réalise en nommant l'objet suivi d'un point suivi du nom de la méthode et de sa liste d'arguments:

nomObjet.nomMethode(arg1, arg2,)

Objet → il n'existe pas de variable de type objet

→ Voiture BM=new Voiture(2, true, 160.0f)

→ Toute chose est un objet.

→ Un objet est une variable améliorée

Classe → Des objets semblables, leur état durant l'exécution du programme mis à part, sont regroupés ensemble dans des « classes d'objets ».

→ Le mot clef **class** est utilisé dans la plupart des langages orienté objet.

Classe → On utilise les classes exactement de la même manière que les types de données prédéfinis.

→ On peut créer des variables d'une classe (appelés objets ou instances) et les manipuler comme des variables d'un type prédéfini.

→ Les objets d'une même classe partagent des caractéristiques communes, mais chaque objet a son propre état.

Constructeurs

- On peut garantir l'initialisation d'un objet grâce à une méthode spéciale appelée constructeur qui permet d'automatiser le mécanisme d'initialisation d'un objet.
- Un constructeur est une méthode appelée au moment de la création de l'objet.
- La création d'un objet passe par l'application de l'opérateur `new` sur les constructeurs de la classe.

Exemple:

```
Voiture BM=new Voiture(2, false,44.50f);
```

```
Voiture Ford;
```

```
Ford = new Voiture(3, true, 55.88f);
```

Exemple : Constructeurs

```
public class Voiture1
{
    private int puissance;    private boolean demarree;
    private float vitesse;
    public Voiture1(int arg1, boolean arg2, float arg3)
    { puissance=arg1; demarree=arg2; vitesse=arg3; }
    public Voiture1() { puissance=0; demarree=false; vitesse=0.0f; }
    public Voiture1(Voiture1 v)
    { puissance=v.puissance; demarree=v.demarree; vitesse=v.vitesse; }
    public boolean demarrer(char cle)
    { if(cle=='o') { demarree=true;    return true; }
      else          { demarree=false;  return false; }
    }
    public float quelpuissance(int preOudeuOutr) // 1, 2, ...
    { float vv=(float)preOudeuOutr;
      if(this.demarree==true) return (vitesse*vv);
      else                    return 0.0f;
    }
}
```

Constructeur de
copie

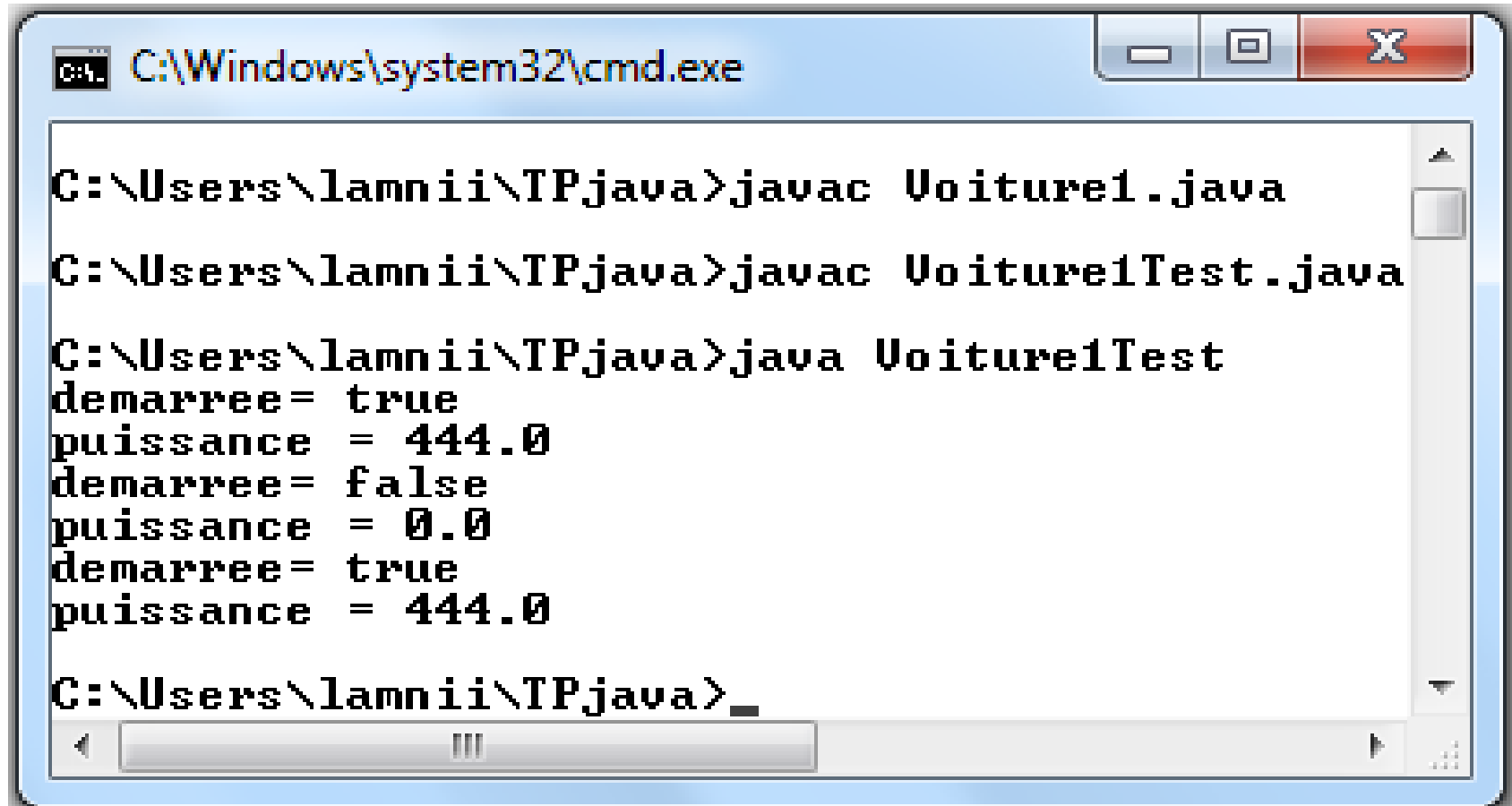
Constructeur par
defaut

Constructeur de
recopie

Exemple : Constructeurs

```
public class Voiture1Test
{
    public static void main(String args[])
    { Voiture1 BM = new Voiture1(1, false, 222.0f) ;
      Voiture1 Dacia= new Voiture1() ;
      System.out.println("demarree= "+BM.demarrer('o')) ;
      System.out.println("puissance = "+BM.quelpuissace(2)) ;
      System.out.println("demarree= "+Dacia.demarrer('n')) ;
      System.out.println("puissance = "+Dacia.quelpuissace(2)) ;
      Dacia=BM;
      System.out.println("demarree= "+Dacia.demarrer('o')) ;
      System.out.println("puissance = "+Dacia.quelpuissace(2)) ;
    }
}
```

Exemple : Constructeurs



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window shows the following commands and output:

```
C:\Users\lannii\TPjava>javac Voiture1.java
C:\Users\lannii\TPjava>javac Voiture1Test.java
C:\Users\lannii\TPjava>java Voiture1Test
demarree= true
puissance = 444.0
demarree= false
puissance = 0.0
demarree= true
puissance = 444.0
C:\Users\lannii\TPjava>_
```

Le destructeur

Un destructeur (et un seul) peut être défini, pour être utilisé lors de la destruction de l'objet. Celui-ci doit forcément se nommer **finalize**, il ne prend aucun paramètre et ne renvoie aucun type (**void**).

Exemple :

```
public class Voiture2
{
    private int puissance;    private boolean demarree;

    void finalize() {
        System.out.println("Objet Voiture détruit");
    }
}
```


Les modificateurs

Les modificateurs sont des mots-clé spéciaux du langage qui modifient la définition et le comportement d'une classe, d'une méthode ou d'une variable. On peut les classer en quatre grandes catégories :

- les modificateurs permettant de contrôler l'accès à une classe, une méthode ou une variable : **public**, **private** et **protected** ;
- le modificateur **static** utilisé pour les variables et les méthodes de classe ;
- le modificateur **final** permettant de « figer » l'implémentation d'une classe, d'une méthode ou d'une variable ;
- le modificateur **abstract** pour la création de classes ou de méthodes abstraites.

Les modificateurs

Il existe d'autres modificateurs que nous allons voir plus tard tels que : **synchronized** et **volatile** (utilisés pour les threads) et **native** (utilisé pour la création de méthodes natives).

public → On utilise le mot clé **public** pour signifier qu'une classe, une méthode ou une variable sont accessibles n'importe où dans « l'univers des classes Java » (même dans d'autres packages).

→ Le mot-clé **public** est le mode par défaut.

Les modificateurs

private → personne ne peut accéder à ces définitions à part les méthodes interne (Il limite la visibilité des méthodes et des variables d'instance à la classe dans laquelle elles sont définies)

→ les sous-classes ne peuvent hériter ni de variables privées, ni de méthodes privées.

Protected → forme de protection entre classe et sous-classes.

→ les méthodes et les variables d'une classe donnée restent accessibles à toute classe du même package (différence avec **C++**), mais ne sont accessibles, en dehors du package, qu'aux sous-classes de la classe à laquelle elles appartiennent.

Les modificateurs

static → est utilisé pour définir des variables ou des méthodes de classe. Le mot-clé **static** indique que l'élément est stocké dans la classe.

final → s'applique aux classes, aux méthodes et aux variables :

- empêche le « sous-classement » de la classe ;
- final signifie que la méthode ne peut pas être redéfinie par les sous-classes ;
- final indique que la « variable » est constante.

modificateur **class** NomClass [extends ClassMere] [implements Interfaces]
{ // champs et les méthodes }

Imbrication de classe

Dans java une classe est dite interne lorsque sa définition est située à l'intérieur de la définition d'une autre classe.

Exemple:

```
public class ImbricTest {  
    public static void main(String args[]) {  
        Imbric test = new Imbric(12.5f, 16.75f);  
        test.affiche(); }  
}
```

```

public class Imbriq
{ private float moy;
  private float note1, note2;
  public Imbriq(float a, float b)
  {moy=0.0f; note1=a; note2=b;}
  public float calcMoy(float arg1, float arg2)
  { Intern notes=new Intern(arg1, arg2);
    return (notes.math+notes.info)/2;
  }
  public void affiche()
  { moy=calcMoy(note1, note2);
    System.out.println(" Moyenne de : " +note1+" et "+note2+" = " +moy);
  }
  class Intern{
    private float math;
    private float info;
    public Intern(float a, float b)
    {math=a; info=b;}
  }
}

```

Auto-référence : le mot clé this

- L'objet « courant » est désigné par le mot clé **this**.
- permet de désigner l'objet dans lequel on se trouve.
- désigne une référence particulière qui est un membre caché utilité de l'objet « courant ».
- Rendre explicite l'accès aux propres attributs et méthodes définies dans la classe.

Auto-référence : le mot clé this → Exemple

```
public class Thistest {
    private int x;
    private int y;
    public Thistest(int a, int b)
    {x=a; y=b;}
    void affiche()
    { int x=2;
      System.out.println("x de affiche = " + x);
      System.out.println("x de l'objet courant = " + this.x);
      System.out.println("y de l'objet courant = " + this.y);
    }
    public static void main(String[] args) {
        Thistest T= new Thistest(4,6);
        T.affiche();
    }
}
```


Tableaux

- Les tableaux sont considérés comme des objets
- Les éléments d'un tableau peuvent être
 - ➔ Des variables d'un type primitif (int, char, float, double,...)
 - ➔ Des références sur des objets

Création

- Déclaration
- Dimensionnement
- Initialisation

Attention: En java il n'est pas possible de définir des tableaux de taille statique à la déclaration: `float[7] Tab;` //erreur de compilation

Tableaux : Exemple 1

```
public class Tab11 {  
    public static void main(String[] args) {  
        int i, n, j=0;  
        char []Tab1={'A', 'Z', 'E', 'R', 'T', 'Y'};  
        for(i=0; i<Tab1.length; i++)  
            {System.out.print(Tab1[i]+ ", "); }  
        System.out.println();  
        System.out.print(" Saisir un entier >0 : ");  
        n= Clavier.lireInt();  
        int Tab2[] = new int[n] ;  
        while(j<n)  
            {Tab2[j]=j*j*n; j++;}  
        j=0;  
        while(j<n)  
            {System.out.println(Tab2[j]); j++;}  
    }  
}
```

Tableaux : Exemple 2

```
public class Tab12 {  
    public static void main(String[] args) {  
        int i;  
        float A, B;  
        Notes T[];  
        T=new Notes[4];  
        for(i=0; i<4; i++)  
            {System.out.println("etudiant "+ (i+1)+ ": M1 et M2 :");  
             A=Clavier.lireFloat();  
             B=Clavier.lireFloat();  
             T[i]=new Notes(A, B);  
             T[i].afficheMoyenne(); }  
    }  
}  
  
class Notes{  
    private float M1;  
    private float M2;  
    public Notes(float a, float b){M1=a; M2=b;}  
    public Notes(){M1=0.0f; M2=0.0f;}  
    public void afficheMoyenne(){  
        System.out.println("(" +M1+" "+M2+") / 2 = " + ((M1+M2) / 2)); }  
}
```

Tableau d'objet

Tableaux à plusieurs indices

- Tableaux dont les éléments sont eux-mêmes des tableaux
- Deux cas:
 - Tableau rectangulaire
 - Tableau non rectangulaire

Exemple :

Tableaux à plusieurs indices

```
public class Tabl3 {  
    public static void main(String[] args) {  
        int i, j;  
        char [][]Tab1={{'A', 'Z'}, {'E', 'R', 'R'}, {'T', 'Y', 'T', 'G'}};  
        for(i=0; i<3; i++)  
            {for(j=0; j<Tab1[i].length; j++)  
                {System.out.print( " "+Tab1[i][j]+ " "); }  
                System.out.println();}  
    }  
}
```

tableau non rectangulaire

Méthodes

main → la première méthode exécutée par une application Java lors de son lancement.

syntaxe → `public static void main (String args[]) { ... }`

Une application Java peut avoir des arguments, introduits après le nom de l'application.

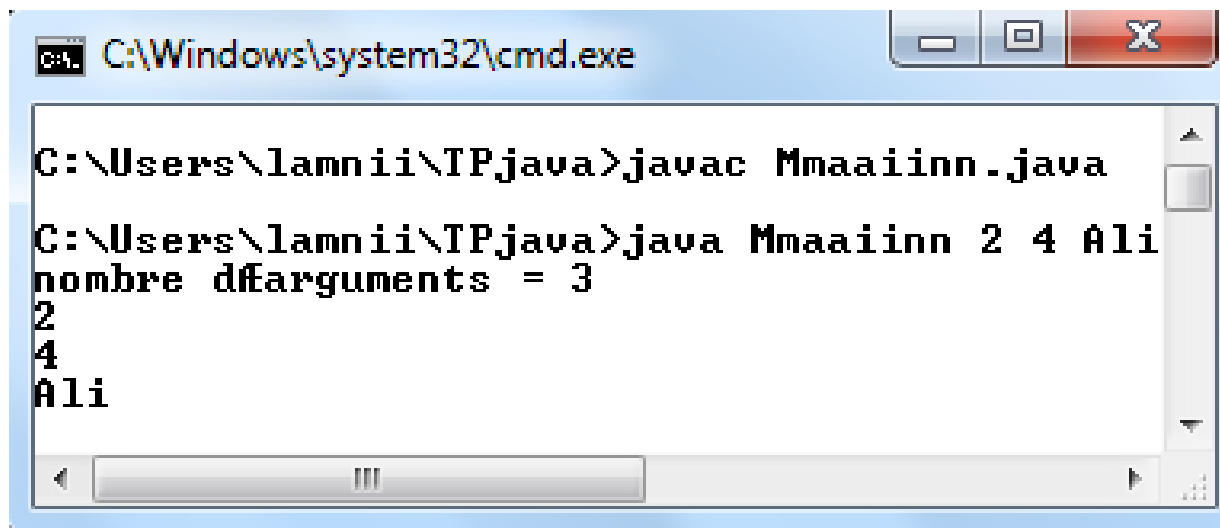
La méthode main doit avoir comme paramètres la chaîne

"String [] args", nécessaire pour stocker les arguments introduits.

"**args.length**" permet de déterminer le nombre d'arguments introduits après le nom de l'application.

Méthodes : main

```
public class Mmaaiinn {  
    public static void main (String args[ ])  
    {  
        int Nbarg; Nbarg=args.length;  
        System.out.println ("nombre d'arguments = " + Nbarg);  
        for (int i=0; i<Nbarg; i++)  
            System.out.println (args[i]);  
    }  
}
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The prompt is at "C:\Users\lamnii\TPjava>". The user has entered the command "javac Mmaaiinn.java" to compile the program. Then, they entered "java Mmaaiinn 2 4 Ali" to run the program with three arguments. The output of the program is displayed on the next lines: "nombre d'arguments = 3", "2", "4", and "Ali".

```
C:\Windows\system32\cmd.exe  
  
C:\Users\lamnii\TPjava>javac Mmaaiinn.java  
C:\Users\lamnii\TPjava>java Mmaaiinn 2 4 Ali  
nombre d'arguments = 3  
2  
4  
Ali
```

Méthodes

- Les méthodes implémentent les traitements de la classe
- N'importe quelle exécution fait intervenir des méthodes
- Une méthode a deux parties : déclaration & corps
- Le type retourné peut être élémentaire ou un objet. Si la méthode ne retourne rien on utilise void.

syntaxe :

```
modificateurs  type_retourné  nom_méthode (arg1, ... )  
{  // définition des variables locales et du bloc d'instructions }
```


Méthodes : Exemple

```
public class ObjetRetour {  
    public static void main(String[] args) {  
        Complex_t z1=new Complex_t(2.2f, -4.5f);  
        Complex_t z2=new Complex_t(5.5f, 14.5f);  
        Complex_t z3=new Complex_t();  
        z3=z3.somme(z1,z2);  
        z1.affiche("z1");  
        z2.affiche("z2");  
        z3.affiche("z3");    }  
}
```

prog. principal

classe Complex_t

```
class Complex_t{  
    private float R, I;
```

Constructeurs

```
    public Complex_t(){R=0.0f; I=0.0f;}  
    public Complex_t(float R, float I){this.R=R; this.I=I;}  
    public Complex_t(Complex_t Z){this.R=Z.R; this.I=Z.I;}
```

Méthode
somme

```
    public Complex_t somme(Complex_t A, Complex_t B){  
        Complex_t Z=new Complex_t();    Z.R=A.R+B.R; Z.I=A.I+B.I;  
        return Z; }
```

Méthode
affiche

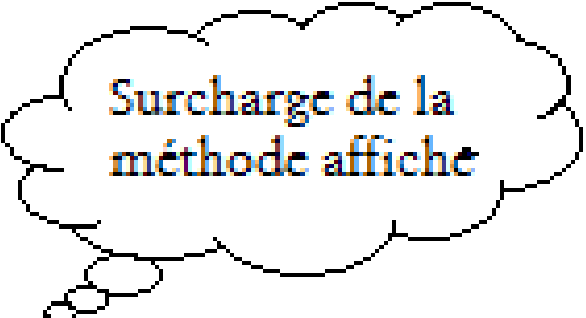
```
    public void affiche (String var) {  
        if(I>=0) System.out.println(var + " = " + R + " + " + I + " i");  
        else     System.out.println(var + " = " + R + " - " + (-I) + " i");    }  
}
```

Méthodes : surcharge

- La surcharge d'une méthode permet de définir plusieurs fois une même méthode avec des arguments différents.
- Le compilateur choisit la méthode qui doit être appelée en fonction du nombre et du type des arguments.
- La signature d'une méthode comprend le nom de la classe, le nom de la méthode, le nombre et les types des paramètres.
- Il est possible de donner le même nom à deux méthodes différentes à condition que les signatures de ces deux méthodes soient différentes.

Méthodes : surcharge

```
public class Animal {  
    private String A;  
    public Animal() {A=" ";}  
    public Animal(String A) {this.A=A;}  
}
```



Surcharge de la
méthode affiche

```
    public void affiche (int D)  {  
        System.out.println(A + " est un animal domestique"); }  
    public void affiche ()  {  
        System.out.println(A + " est un animal sauvage"); }  
}
```

```
public static void main(String[] args) {  
    Animal D=new Animal("Chat");  
    D.affiche(0);    //D.affiche();  
    Animal S=new Animal("Tigre");  
    S.affiche(); }  
}
```

Les paquetages

- Les classes Java sont regroupées en paquetages (packages en anglais) comparables aux «bibliothèques» des autres langages comme le langage C++, ...
- L'instruction : **import MesClasses ;** permet l'utilisation de toutes les classes dont le paquetage MesClasses est engendré.

Exemple:

```
import nomClasse;    // Importe une classe
```

```
import nomPack.nomClasse; // Importe une classe de nomPack
```

```
import nomPack.*;    // Importe toutes les classes de nomPack
```

- Le package par défaut : c'est le package qui n'a pas de nom, et auquel appartiennent toutes les classes d'un même répertoire.

Les paquets

Quelques paquets du SDK (Software Development Kit)

- **java.lang** : classes de base de Java
- **java.util** : utilitaires
- **java.io** : entrées-sorties
- **java.awt** : interface graphique
- **javax.swing** : interface graphique avancée
- **java.applet** : applets
- **java.net** : réseau
- **java.rmi** : distribution des objets

Les paquetages : créer un package

- Pour créer un **package**, il suffit de commencer le fichier source contenant les classes à regrouper par l'instruction **package** suivi du nom que l'on désire donner au package.
- Un package portant le nom **PaquetageA** doit être stocké dans un répertoire de nom **PaquetageA**.
- Par défaut le compilateur (ainsi que la machine virtuelle) recherchent les classes dans le répertoire courant et le répertoire des classes standards.

Chat.java

```
package Animale;  
public class Chat{  
    public void afficheChat() { System.out.println(" domestique "); }  
}
```

Tigre.java

```
package Animale;  
public class Tigre{  
    public void afficheTigre() { System.out.println(" sauvage "); }  
}
```

```
import Animale.*;  
public class TestPackage {  
    public static void main(String[] args) {  
        Chat c=new Chat();  
        c.afficheChat();  
        Tigre t=new Tigre();  
        t.afficheTigre();  
    }  
}
```

HÉRITAGE

- La notion d'héritage constitue l'un des fondements de la programmation OO.
- L'héritage est un mécanisme qui permet à une classe d'hériter l'ensemble du comportement et des attributs d'une autre classe.
- Permet de définir une nouvelle classe, dite classe dérivée, à partir d'une classe existante dite classe de base (super classe).
- La classe dérivée
 - se comporte comme la classe de base mais avec quelques différences.
 - on a seulement besoin du code compilé de la classe de base.

HÉRITAGE

On peut par exemple:

- ajouter de nouvelles méthodes
- adapter (modifier) certaines méthodes

Syntaxe:

```
class <ClasseDerivee> extends <ClasseDeBase>
```

Interprétation:

→ Permet de définir un lien d'héritage entre deux classes:

La classe <ClasseDeBase> est le nom de la classe de base. Elle s'appelle une classe mère, une classe parente ou une super-classe.

La classe <ClasseDerivee> est le nom de la classe dérivée

→ Par défaut il n'y a pas de **extends** dans la définition d'une classe: toutes les classes héritent de la classe Object.

```
class Animal
{ public void NombreDePattes ()
  { System.out.print("Nombre de pattes = ") ;   }
  public void deplace ()
  { System.out.print("Bouge en ") ;   }
}
```

classe de base

Chat derivée
de Animal

```
class Chat extends Animal
{ public void NombreDePattes ()
  { super.NombreDePattes(); System.out.println(" 4 ") ; }
  public void deplace ()
  {super.deplace (); System.out.println("marchant ") ; }
}
```

Oiseau derivée
de Animal

```
class Oiseau extends Animal
{ public void NombreDePattes ()
  { super.NombreDePattes(); System.out.println(" 2 ") ; }
  public void deplace ()
  { super.deplace(); System.out.println("Volant ") ;   }
}
```

classe utilisant
Chat et Oiseau

```
public class TestHeritage
{ public static void main (String args[])
  { Chat C = new Chat(); C.NombreDePattes(); C.deplace();
    Oiseau O = new Oiseau(); O.NombreDePattes(); O.deplace(); }
}
```

HÉRITAGE : CONSTRUCTEURS

- Possibilité comme les méthodes de réutiliser le code des constructeurs de la super-classe.
- L'appel au constructeur de la super-classe doit se faire absolument en première instruction.
- La première instruction d'un constructeur peut être un appel à un constructeur de la classe mère : **super(...)** ou à un autre constructeur de la classe : **this(...)**
- Interdit de placer **this()** ou **super()** ailleurs qu'en première instruction d'un constructeur.

HÉRITAGE : CONSTRUCTEURS

```
class Vehicule
{ protected int  Model;
  protected String Matricule;
  public Vehicule(String Matricule, int Model)
  {this.Matricule=Matricule; this.Model=Model;}
  public void NombreDeChevaux()
  { System.out.print("Nombre de chevaux = ") ;    }
}

class Voiture extends Vehicule
{ private String Marque;
  public Voiture(String A, int B, String C)
  {super(A,B) ;
   this.Marque=C;}
  public void NombreDeChevaux()
  { System.out.println(" Voiture : ") ;
   super.NombreDeChevaux(); System.out.println(" 12 ") ; }
  public void Information ()
  {System.out.println("Matricule = " + Matricule + ", Model = " + Model ); }
  public void Type ()
  {System.out.println("Marque : " + Marque); }
}
```

HÉRITAGE : CONSTRUCTEURS

```
class Camion extends Vehicule
{
    private int Poids;
    public Camion(String A, int B, int C)
    {super(A,B) ;
     this.Poids=C;}
    public void NombreDeChevaux()
    { System.out.println(" Camion : ") ;
      super.NombreDeChevaux() ; System.out.println(" 15 ") ; }
    public void Type ()
    {System.out.println("Poids : " + Poids); }
    public void Tonage ()
    {System.out.println("Tonage: 50 tonne "); }
}

public class TestVehicule
{
    public static void main (String args[])
    {
        Voiture V = new Voiture("ww11212M",2002,"BM") ;
        V.NombreDeChevaux() ; V.Information() ; V.Type() ;
        Camion C = new Camion("ww11212M",2002,13) ;
        C.NombreDeChevaux() ; C.Type() ; C.Tonage() ;
    }
}
```

HÉRITAGE

- L'héritage successif de classes permet de définir une hiérarchie de classe qui se compose de super classes et de sous classes.
- Une classe ne peut avoir qu'une seule classe mère : il n'y a pas d'héritage multiple en java.
- **Object** est la classe parente de toutes les classes en java
- Toutes les variables et les méthodes contenues dans **Object** sont accessibles à partir de n'importe quelle classe car par héritage successif toutes les classes héritent de la classe **Object**.
- Java ne permet pas l'héritage multiple: chaque classe a une et une seule classe mère dont elle hérite les variables et les méthodes.

Polymorphisme

Soit B une classe qui hérite d'une autre classe A.

f() une méthode de A qui redéfinie dans B.

g() une méthode de A.

h() une méthode de B.

Polymorphisme

```
class A { public void f( ){ System.out.println("Méthode f de A"); }
        public void g( ){ System.out.println("Méthode g de A"); }
        }

class B extends A{ public void f( ){ System.out.println("Méthode f de B"); }
                   public void h( ){ System.out.println("Méthode h de B"); }
                   }

public class TestPoly
{ public static void main (String args[])
  { A a=new A(); B b = new B(); A ab = new B();
    // B ba = new A(); // erreur de compilation
    a.f(); // appelle la méthode définie dans A
    //b=a; // erreur de compilation
    a=b;    // A a = new(B);
    a.f(); // appelle la méthode définie dans B
    a.g(); // appelle la méthode définie dans A
    b.g(); // appelle la méthode définie dans A
    //a.h(); // erreur de compilation
    ab.f(); // appelle la méthode définie dans B
    ab.g(); // appelle la méthode définie dans A
  }
}
```


Polymorphisme

- L'appel se fonde sur le type effectif de l'objet référencé par **ab** au moment de l'appel et non sur le type de la variable **ab**.
- Il faut noter que la même écriture `a.f()`; peut correspondre à des appels différents de la méthode `f()`. Ceci est réalisé grâce au polymorphisme.
- Le Polymorphisme veut dire que le même service peut avoir un comportement différent suivant la classe dans laquelle il est utilisé. C'est un concept fondamental de la programmation objet, indispensable pour une utilisation efficace de l'héritage.
- Pour l'objet **ab** si `f()` est redéfinie dans `B`, alors c'est cette méthode qui sera exécutée sinon, la recherche de `f()` se poursuit dans la classe mère de `B`, puis dans la classe mère de cette classe mère, et ainsi de suite, jusqu'à trouver `f()` qui sera alors exécutée.

Polymorphisme

Le polymorphisme permet d'éviter les codes qui comportent de nombreux embranchements et tests.

Exemple :

Considérons une classe **TerreVéhicule**. Supposons que les classes **Voiture** et **Camion** héritent de la super classe **TerreVéhicule**. Dans un tableau hétérogène, on range des objets de type **Voiture** , **Camion** . Ensuite on affiche le contenu du tableau :

```
class TerreVehicule {  
    .....  
    public void appelvoiture() {}  
    public void appelcamoin() {}  
}  
class Voiture extends TerreVehicule {  
    public void appelvoiture() {System.out.println("Voiture");}  
}  
class Camion extends TerreVehicule {  
    public void appelcamoin() {System.out.println("Camion");}  
}
```

```
public class TestTerreVehicule  
{ public static void main(String[] args) {  
    TerreVehicule [] VC = new TerreVehicule[4];  
    VC[0]=new Voiture(); VC[1]=new Voiture();  
    VC[2]=new Camion(); VC[3]=new Voiture();  
    for (int i=0; i < VC.length; i++) {  
        if (VC[i] instanceof Voiture) VC[i].appelvoiture();  
        else VC[i].appelcamoin(); }  
    }  
}
```

Polymorphisme

instanceof: `Voiture instanceof Véhicule` retourne **true**

Inconvénients :

- Si on veut rajouter un **Fourgon**, on est obligé de:
changer dans le code source aux niveaux définition de la classe **Véhicule** (rajouter la méthode `public void appellefourgon() {}`)
- En exploitant le polymorphisme, on peut :
 - Éviter les tests
 - Rendre le code extensible

Exemple:

```
public class TestTerreVehicule2
{
    public static void main(String[] args) {
        TerreVehicule [] VC = new TerreVehicule[4];
        VC[0]=new Voiture(); VC[1]=new Fourgon();
        VC[2]=new Camion(); VC[3]=new Voiture();
        for (int i=0; i < VC.length; i++) VC[i].appelVehicule();
    }
}

class TerreVehicule {
    .....
    public void appelVehicule() {}
}

class Voiture extends TerreVehicule {
    public void appelVehicule(){System.out.println("Voiture");}
}

class Camion extends TerreVehicule {
    public void appelVehicule(){System.out.println("Camion");}
}

class Fourgon extends TerreVehicule {
    public void appelVehicule(){System.out.println("Fourgon");}
}
```

Cast (transtypage)

- Le « cast » est le fait de forcer le compilateur à considérer un objet comme étant d'un type qui n'est pas le type déclaré ou réel de l'objet.
 - Les seuls casts autorisés entre classes sont les casts entre classe mère et classes filles.
 - UpCast : classe fille → classe mère → implicite
 - DownCast : classe mère → classe fille → explicite
- Toujours accepté par le compilateur, mais peut provoquer une erreur à l'exécution ;

Cast (transtypage)

```
public class TestCast
{ public static void main(String[] args) {
    TerreVehicule A=new TerreVehicule();
    Voiture V =new Voiture();    A= (TerreVehicule)V;
    // W= (Voiture)B; accepté par le compilateur --> erreur à l'exécution
    A.appelVehicule();    V.appelVehicule();    V.fctDEmere();    }
}

class TerreVehicule {
    ...    public void appelVehicule() {System.out.println("Vehicule");}
}

class Voiture extends TerreVehicule {
    public void appelVehicule() {System.out.println("Voiture");}
    public void fctDEmere() {System.out.println("fct de mere");}
}
```

Classe et méthodes abstraites

- Une méthode est dite abstraite si on la déclare sans donner son implémentation: on ne donne que le type de la valeur de retour et la signature (l'entête de la méthode).
- Les méthodes abstraites sont déclarées avec le mot clé **abstract**.
- La méthode abstraite sera implémentée par les classes filles.
- Une classe doit être déclarée abstraite si elle contient une méthode abstraite : **public abstract class NomClasse{...**
- Il est interdit de créer une instance d'une classe abstraite.

Classe abstraites : Intérêt

On peut placer dans une classe abstraite toutes les fonctionnalités dont on souhaite disposer pour toute ses descendantes, sous forme :

- ➔ d'implémentation complète de méthodes (non abstraites) et de champs lorsqu'ils sont communs à toutes ses descendantes.
- ➔ d'interface de méthodes abstraites dont on est alors sûr qu'elles existeront dans toute classe dérivée instanciable.

Classe abstraites : Exemple

Soient **Etudiant**, **Mater**, **Licence** et **Stocker** telles que:

Stocker des noms d'Etudiant de deux niveaux (Mater, Licence) dans une liste puis les afficher.

Etudiant

```
float note1  
float note2  
String nom  
Etudiant( float a, float b , String s)  
float moyenne()  
void afficher()
```

Licence

```
int age  
Licence(float a, float b, String s, int e)  
float moyenne()  
void afficher()
```

Stocker

```
Etudiant[ ] liste;  
int leng;  
int i;  
void ajouterEtudiant()  
void afficherEtudiant()
```

Mater

```
int age  
Licence(float a, float b, String s, int e)  
float moyenne()  
void afficher()
```

```
public abstract class Etudiant {  
    protected float note1, note2;  
    protected String nom;  
    public Etudiant(float a, float b, String s) {  
        ..... note1 = a; note2 = b; nom = s; }  
    public float moyenne() { return 0.0f; }  
    public void afficher() { }  
}
```

```
public class Licence extends Etudiant {  
    private int age ;  
    public Licence(float a, float b, String s, int e) {  
        super(a,b,s);  
        this.age = e; }  
    public float moyenne() { return (note1+note2)/2; }  
    public void afficher() {  
        System.out.println("| Nom | Age | Niveau | Moyenne |");  
        System.out.println(" "+nom + " "+ age + " Licence "+" "+moyenne()) ;  
    }  
}
```

```
public class Master extends Etudiant {
private int age ;
public Master(float a, float b, String s, int e) {
super(a,b,s);
this.age = e; }
public float  moyenne(){ return((note1+note2)/2); }
public void  afficher() {
    System.out.println("|  Nom  | Age | Niveau | Moyenne |");
    System.out.println("    "+nom + "        "+ age + "        Master "+"    "+moyenne()) ;
}
}
```

```

public class Stoker {
    private Etudiant[ ] liste;
    private int leng, i;
    public Stoker(int leng) {
        this.leng = leng;
        liste = new Etudiant[this.leng];
        i = 0; }
    public void ajouterEtudiant(Etudiant E) {
        if (i < leng) { liste[i] = E; i++; } }
    public void afficherEtudiant() {
        for (int j = 0; j < i; j++) liste[j].afficher(); }
    public static void main (String[ ] argv) {
        Stoker base = new Stoker(6);
        base.ajouterEtudiant(new Licence(15.5f,18.0f,"Saad",20));
        base.ajouterEtudiant(new Master(12.0f,11.0f,"Ali",22));
        Master M1 = new Master(13.0f,14.5f,"Adil",23);
        Etudiant L1 = new Licence(15.0f,17.0f,"Ahmed",19);
        base.ajouterEtudiant(M1);
        base.ajouterEtudiant(L1);
        base.afficherEtudiant(); }
}

```

C:\Windows\system32\cmd.exe

C:\Users\lannii\TPjava>javac Etudiant.java

C:\Users\lannii\TPjava>javac Licence.java

C:\Users\lannii\TPjava>javac Master.java

C:\Users\lannii\TPjava>javac Stoker.java

C:\Users\lannii\TPjava>java Stoker

| | | | | | | | | |
|--|-------|--|-----|--|---------|--|---------|--|
| | Nom | | Age | | Niveau | | Moyenne | |
| | Saad | | 20 | | Licence | | 16.75 | |
| | Nom | | Age | | Niveau | | Moyenne | |
| | Ali | | 22 | | Master | | 11.5 | |
| | Nom | | Age | | Niveau | | Moyenne | |
| | Adil | | 23 | | Master | | 13.75 | |
| | Nom | | Age | | Niveau | | Moyenne | |
| | Ahmed | | 19 | | Licence | | 16.0 | |

Interfaces

- Une interface est une "classe" purement abstraite dont toutes les méthodes sont abstraites et publiques.
- Une interface ne possède pas d'attribut mais elle peut posséder des constantes.
- Une interface ne peut contenir que des entêtes de méthodes.
- Une classe peut implémentée plusieurs interfaces.
- Les interfaces pourront se dériver.

Interfaces

Syntaxe et définition:

```
public interface Interface1 { ... }
```

```
public interface Interface2 { ... }
```

→ Dans la définition d'une classe, on peut préciser qu'elle implémente une ou plusieurs interfaces donnée(s) en utilisant une fois le mot clé **implements**

```
public class NomClasses implements Interface1, Interface2, ... {  
    ...  
}
```

→ Une classe hérite d'une autre classe peut également implémenter une ou plusieurs interfaces

```
public class NomClasses extends SuperClasse implements Interface1,  
    Interface1 ... { ... }
```


Interfaces : Exemple 1

```
interface Perimetre{public float per();}
```

```
interface Surface{public float sur();}
```

```
interface Resultat{public void result();}
```

Rectangle implémente
les interfaces :
Perimetre, Surface
et Resultat

```
class Rectangle implements Perimetre, Surface, Resultat
{ private float A, B;
  public Rectangle (float a, float b){A=a; B=b;}
  public float per() {return (2*(A+B)); }
  public float sur() {return (A*B); }
  public void result(){
    System.out.println("Perimetre = " + per());
    System.out.println("Surface    = " + sur());
  }
}
```

```
public class TestInterface
{ public static void main (String[] args)
  { Rectangle R = new Rectangle(2.5f, 4.0f);
    R.result(); }
}
```

Interfaces : Exemple 2

```
interface Perimetre{public float per();}
interface Surface{public float sur();}
interface Resultat{public void result();}

class Rectangle implements Perimetre, Surface
{ protected float A, B;
  public Rectangle (float a, float b){A=a; B=b;}
  public float per() {return (2*(A+B)); }
  public float sur() {return (A*B); }
}

class Square extends Rectangle implements Resultat
{ public Square(float a){super(a,a); }
  public void result(){
    System.out.println("Perimetre = " + per());
    System.out.println("Surface    = " + sur());}
}

public class Test2interface
{ public static void main (String[] args)
  { Square S = new Square(5.0f);
    S.result(); }
}
```

Square hérite
de Rectangle

Les chaînes de caractères

```
public class Chainel
{ public static void main (String[] args)
  { String ch= "Bonjour ", rech="on";
    String ch1= new String();
    String ch2= new String ("Tous le monde");
    String ch3= new String();
    int longo;      longo = ch2.length( );
    ch3=ch+ch2; // ch3=ch.concat(ch2);
    System.out.println("ch = " + ch + "ch1 = " + ch1 +
      "ch2 = " + ch2 + "ch3 = " + ch3 );
    System.out.println("ch.charAt(0) = " + ch.charAt(0));
    System.out.println("ch.charAt(2) = " + ch.charAt(2));
    int pos= ch.indexOf(rech);
    System.out.println("pos = " + pos + ", longo = " + longo);
  }
}
```

Les chaînes de caractères

→ Comparaisons de chaînes :

→ == et !=

→ equals()

→ equalsIgnoreCase() // sans distinguer les majuscules des minuscules

→ compareTo() // comparaisons lexicographiques

Les chaînes de caractères

```
public class Chaîne2
{ public static void main (String[] args)
  { String ch= "Bonjour ", ch1="BONJOUR", ch2="Bonsoir";
    System.out.println("ch1 == ch2 --> " + (ch1 == ch2));
    System.out.println("ch1 != ch2 --> " + (ch1 != ch2));
    System.out.println("ch1.equals(ch2) --> " + ch1.equals(ch2));
    System.out.println("ch.equals(ch1) --> " + ch.equals(ch1));
    System.out.println("ch1.equalsIgnoreCase(ch2) --> " + ch1.equalsIgnoreCase(ch2));
    System.out.println("ch.equalsIgnoreCase(ch1) --> " + ch.equalsIgnoreCase(ch1));
    System.out.println("ch1.compareTo(ch2) --> " + ch1.compareTo(ch2));
    System.out.println("ch.compareTo(ch1) --> " + ch.compareTo(ch1));
  }
}
```

Les chaînes de caractères

→ Modifications de chaînes

→ `toLowerCase()`

→ `toUpperCase()`

→ `replace()` //remplace toutes les occurrences d'un caractère donné par un autre.

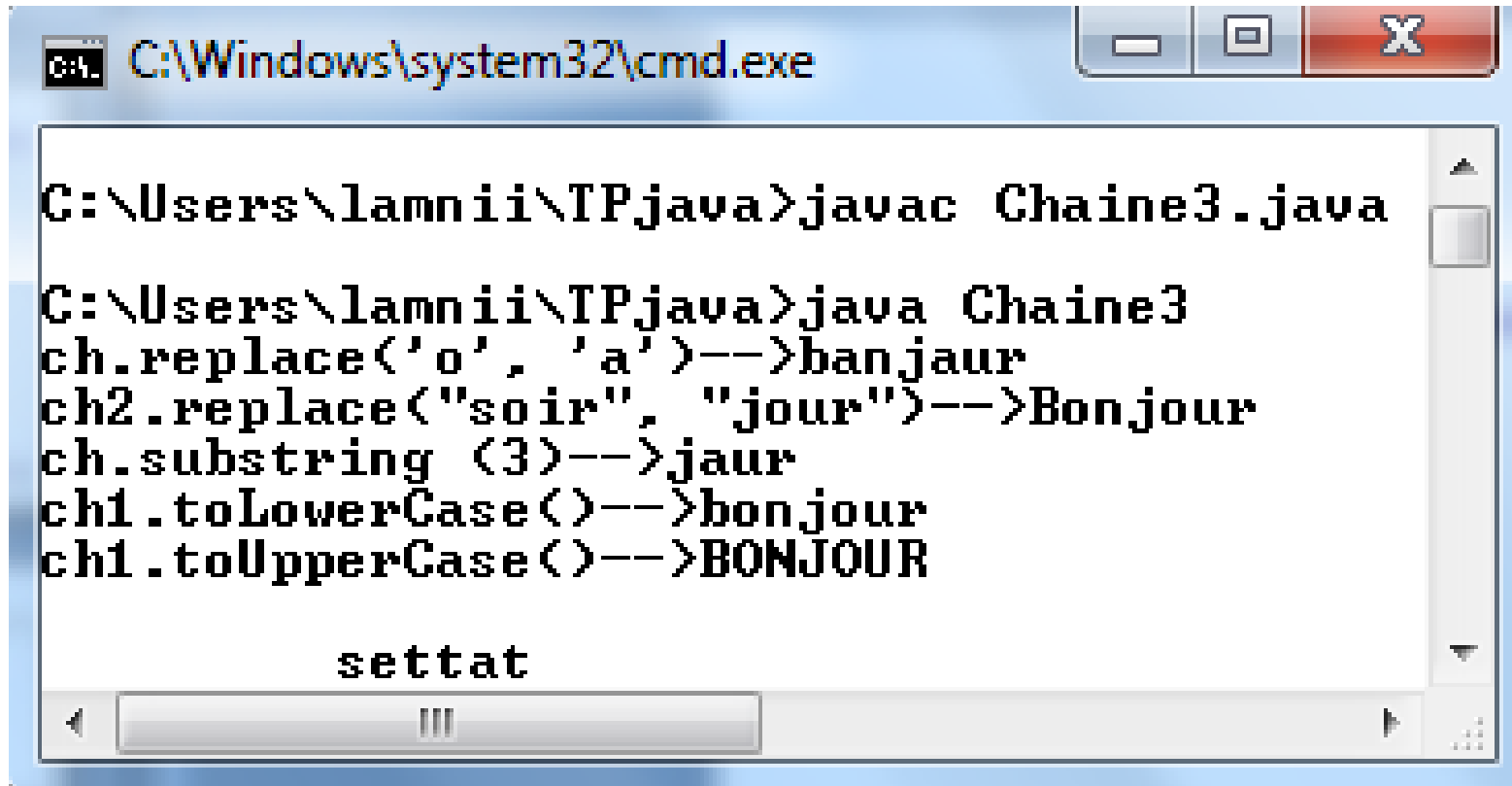
→ `substring()` //créer une nouvelle chaîne en extrayant de la chaîne courante

→ `trim()` //créer une nouvelle chaîne en supprimant les éventuels séparateurs

Les chaînes de caractères

```
public class Chaîne3
{ public static void main (String[] args)
  { String ch= "bonjour ", ch1="BONJOUR", ch2="Bonsoir", ch3="\n\t settat \b";
    ch = ch.replace('o', 'a') ;
    System.out.println("ch.replace('o', 'a')-->" + ch);
    ch2 = ch2.replace("soir", "jour") ;
    System.out.println("ch2.replace(\"soir\", \"jour\")-->" + ch2);
    ch = ch.substring (3) ;
    System.out.println("ch.substring (3)-->" + ch);
    ch1 = ch1.toLowerCase() ;
    System.out.println("ch1.toLowerCase()-->" + ch1);
    ch1 = ch1.toUpperCase() ;
    System.out.println("ch1.toUpperCase()-->" + ch1);
    System.out.println(ch3);
    ch3 = ch3.trim() ;
    System.out.println("ch3.trim()-->" + ch3);
  }
}
```

Les chaînes de caractères



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window shows the following commands and their outputs:

```
C:\Users\lannii\TPjava>javac Chaine3.java  
C:\Users\lannii\TPjava>java Chaine3  
ch.replace('o', 'a')-->banjaur  
ch2.replace("soir", "jour")-->Bonjour  
ch.substring (3)-->jaur  
ch1.toLowerCase()-->bonjour  
ch1.toUpperCase()-->BONJOUR  
  
settat
```


Les chaînes de caractères

- Les String Java ne peuvent pas être considérées comme des tableaux de caractères
- Pour convertir une chaîne en un tableau de caractères, on doit utiliser la méthode `toCharArray`

```
String ch= "settat" ; char [] tab1DEcarc; char [] tab2DEcarc;  
tab1DEcarc = ch.toCharArray( ) ;  
tab2DEcarc = "settat".toCharArray( );
```

- Conversion de type:

| | |
|-----------------------------------|-------------------------------------|
| → <code>String.valueOf()</code> | → <code>Long.parseLong()</code> |
| → <code>Integer.parseInt()</code> | → <code>Float.parseFloat()</code> |
| → <code>Byte.parseByte()</code> | → <code>Double.parseDouble()</code> |
| → <code>Short.parseShort()</code> | |

Les chaînes de caractères

```
public class Chaîne4
{ public static void main (String[] args)
  { int a = 31 ;
    String ch = String.valueOf(a) ; // fournit une chaîne obtenue par formatage
    String ch1 = "2011" ;
    int n = Integer.parseInt(ch) ;
    float x = Float.parseFloat(ch1) ;
    System.out.println(a +" --> String.valueOf(a) -->" + (ch+3));
    System.out.println(ch1 +" --> Integer.parseInt(ch) -->" + (n+2));
    System.out.println(ch1 +" --> Float.parseFloat(ch1) -->" + (x+2));
  }
}
```

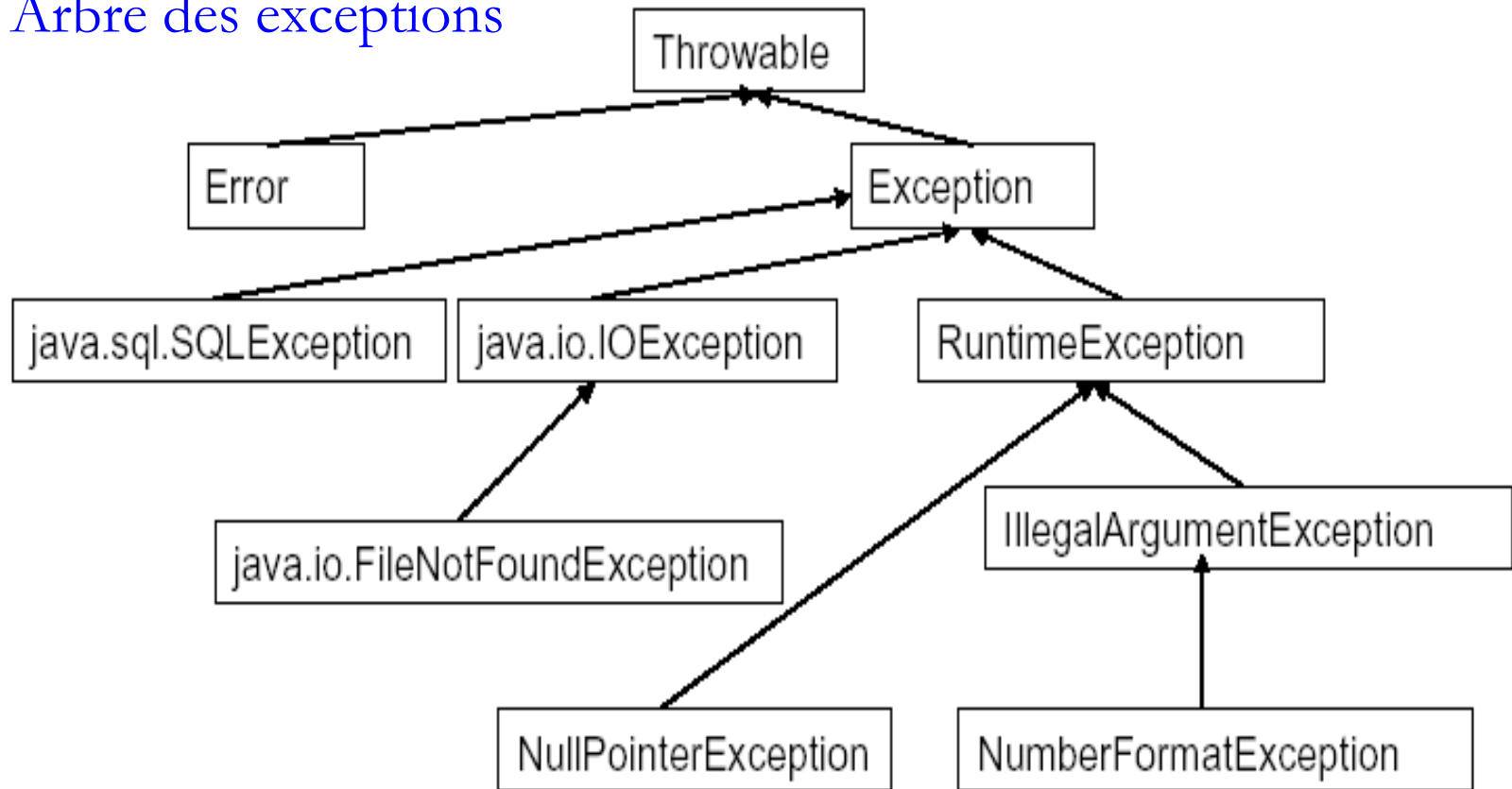
Gestion des exceptions

- Une exception est un signal indiquant que quelque chose d'exceptionnelle (comme une erreur, disque plein, division par zéro, ...) s'est produit.
- Elle interrompt le flot d'exécution normal du programme
- Plutôt que de compliquer le code du traitement normal, on traite les conditions anormales à part.
- Le traitement « normal » apparaît ainsi plus simple et plus lisible.
- Résoudre les problèmes d'exception soit :
 - En envoyant un message
 - Mettre fin au programme
 - Revenir en arrière
 - ...

Gestion des exceptions

→ Toute exception en Java est un objet instancier d'une sous classe de la classe `Exception`

→ Arbre des exceptions



Gestion des exceptions

→ Quelques exceptions prédéfinies :

→ Division par zéro : `ArithmeticException`

→ Référence nulle : `NullPointerException`

→ Tentative de forçage de type illégale : `ClassCastException`

→ Tableau de taille gative : `NegativeArraySizeException`

☐☐☐→Dépassement de dimension: `ArrayIndexOutOfBoundsException`

→ ...

Gestion des exceptions : try et catch

Try {

// Instructions ;

}

L'instruction *try* indique une instruction (ou plus un bloc d'instructions) susceptible de lever des exceptions

catch (Exception excep)

{

// Instructions ;

}

L'instruction *catch* indique le traitement pour un type particulier d'exceptions

Gestion des exceptions

Exceptions de type RuntimeException

- Les exceptions de type RuntimeException correspondent à des erreurs qui peuvent survenir dans toutes les portions du codes:
 - `ArithmeticException` : division par zéro (entiers), etc
 - `IndexOutOfBoundsException` : dépassement d'indice tableau.
 - `NullPointerException` : référence null alors qu'on attendait une référence vers une instance.
 - ...

Exemple :

```

class Inv{
    private int n;
    public Inv(int n){this.n=n;}
    public void inverse(){
        if(n!=0) System.out.println ("1/"+n+" = " +(1/(float)n)) ;
        else      System.out.println ("1/"+n+" = " +(1/n)) ; }
}
public class TestExcept3
{ public static void main (String args[])
  { Inv invDEN= new Inv(0) ;
    Inv invDEN1= new Inv(3) ;
    try
    { invDEN1.inverse() ;
      .....
      invDEN.inverse() ;
    }
    catch (RuntimeException e)
    { System.out.println (" division par 0 ") ; }
  }
}

```


Gestion des exceptions : contrôlées

→ Pour définir une nouvelle exception, on crée une nouvelle classe sous la forme suivante :

```
class NomDeClassEception extends ExceptionDejaDefinie { }
```

→ Lorsque l'on veut lancer une exception, on utilise le mot clé `throw` suivi de l'exception à lancer : `throw new DivZero();`

→ La clause `try` s'applique à un bloc d'instructions correspondant au fonctionnement normal mais pouvant générer des erreurs.

→ La clause `catch` s'applique à un bloc d'instructions définissant le traitement d'un type d'erreur. Ce traitement sera lancé sur une instance de la classe d'exception passée en paramètre.

Gestion des exceptions : contrôlées

Le JDK définit de nombreuses exceptions :

- `IOException` : fin de fichier.
- `FileNotFoundException` : erreur dans l'ouverture d'un fichier.
- `ClassNotFoundException` : erreur dans le chargement d'une classe.
- ...

Toute exception contrôlée, du JDK, pouvant être émise dans une méthode doit être :

- soit levée dans cette méthode. Elle est alors lancée dans un bloc `try` auquel est associé un `catch` lui correspondant.
- soit être indiquées dans le prototype de la méthode à l'aide du mot clé `throws`.

Gestion des exceptions : Exemple 1

```
class Inverse
{ private int I ;
  public Inverse(int i) throws DivZero
  { if ( i==0) throw new DivZero();  I = i ;    }
  public void affiche()
  { System.out.println (" 1/" + I + " = " + (1/(float)I)) ;  }
}
class DivZero extends Exception { }
public class TestExcept1
{ public static void main (String args[])
  { try
    { Inverse [] Tab;
      Tab= new Inverse[11] ;
      for(int i=-5;i<=11;i++)
        {Tab[i+5]=new Inverse(i) ;
          Tab[i+5].affiche() ;}
    }
    catch (DivZero e)
    { System.out.println (" Division par zero ") ;    }
  }
}
```

Gestion des exceptions : Exemple 2

```
class Fact{
    private int n;
    public Fact(int n)throws PasDef {
        if(n<0) throw new PasDef(); this.n=n;}
    public int factorial(int n)
    { if(n==0) return 1;
      else return (n*factorial(n-1)); }
    public void affiche(){
        System.out.println (n+"! = "+factorial(n)) ; }
}
class PasDef extends Exception { }
public class TestExcept2
{ public static void main (String args[])
  {
    try
    { Fact factDen1= new Fact(5); factDen1.affiche();
      .....
      Fact factDen2= new Fact(-5); factDen2.affiche();
    }
    catch(PasDef e)
    { System.out.println (" pas definie pour les negatifs ") ; }
  }
}
```

Gestion des exceptions : personnalisées

→ Les exceptions personnalisées sont des sous-classe de la classe `Exception`.

Exemple:

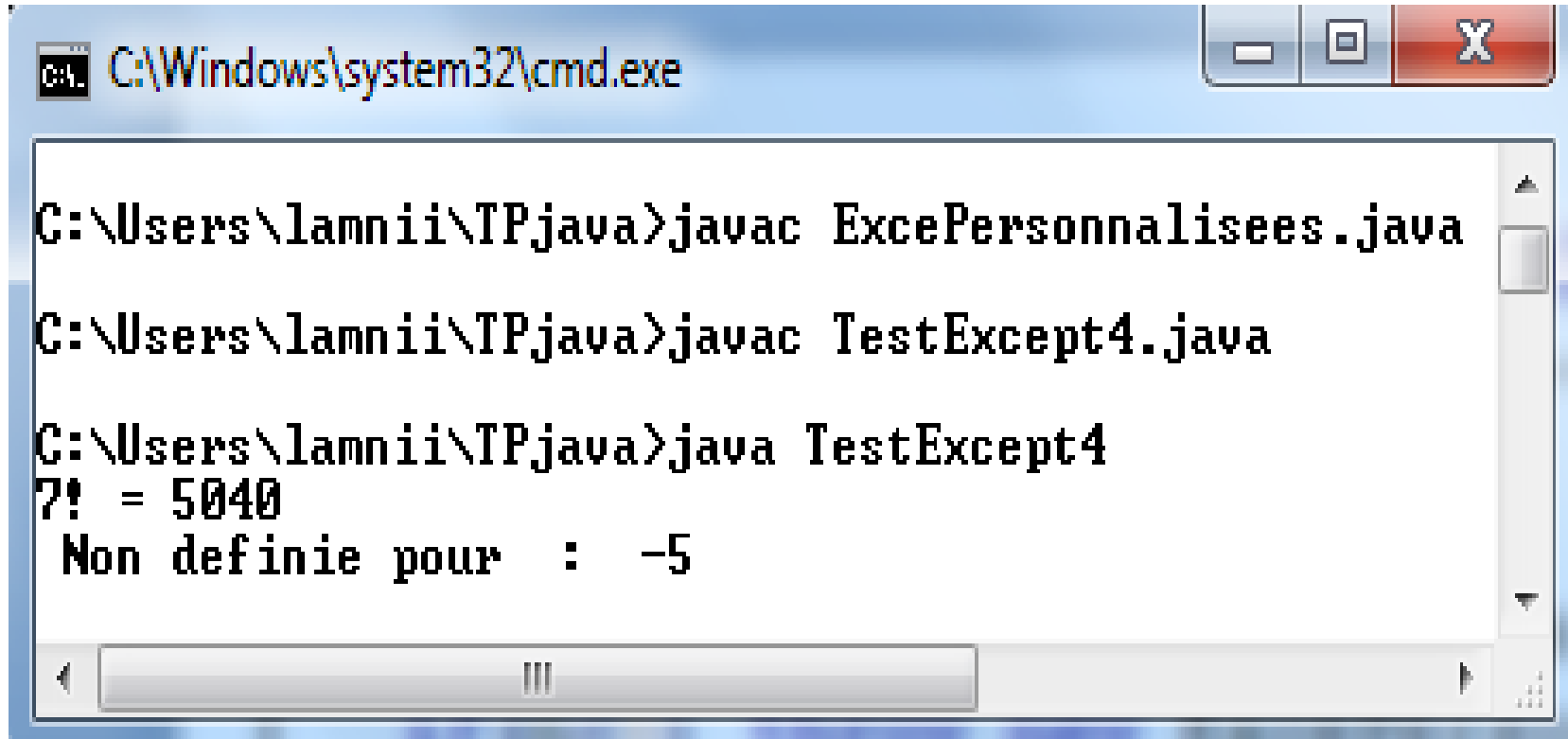
```
public class ExcePersonnalisees extends Exception {  
    private int n ;  
    public ExcePersonnalisees( int N ) {  
        n = N ; }  
    public String toString () {  
        return " Non definie pour : " + n ; }  
}
```

Gestion des exceptions : personnalisées

```
class Fact{
    private int n;
    public Fact(int n) throws ExcePersonnalisees {
        if(n<0) throw new ExcePersonnalisees(n); this.n=n;}
    public int factorial(int n)
    { if(n==0) return 1;
      else return (n*factorial(n-1)); }
    public void affiche(){
        System.out.println (n +"! = "+factorial(n)) ; }
}

public class TestExcept4
{ public static void main (String args[])
  {
    try
    { Fact factDen1= new Fact(7); factDen1.affiche();
      .....
      Fact factDen2= new Fact(-5); factDen2.affiche();
    }
    catch (ExcePersonnalisees e)
    { System.out.println (e) ; }
  }
}
```

Gestion des exceptions : personnalisées



```
C:\Windows\system32\cmd.exe

C:\Users\lamnii\TPjava>javac ExcePersonnalisees.java
C:\Users\lamnii\TPjava>javac TestExcept4.java
C:\Users\lamnii\TPjava>java TestExcept4
?! = 5040
Non definie pour : -5
```

The image shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window contains the following text:

```
C:\Users\lamnii\TPjava>javac ExcePersonnalisees.java
C:\Users\lamnii\TPjava>javac TestExcept4.java
C:\Users\lamnii\TPjava>java TestExcept4
?! = 5040
Non definie pour : -5
```

The output shows the compilation of two Java files and the execution of the resulting class, which produces the output "?! = 5040" and "Non definie pour : -5".

Gestion des exceptions : finally

- La clause `finally` définit un bloc d'instruction qui sera exécuté même si une exception est lancée dans le bloc d'essai. Elle permet de forcer la bonne terminaison d'un traitement en présence d'erreur.
- Un bloc `finally` est exécuté quelle que soit le résultat du bloc `try`.
- `finally` pour ce qui s'exécute dans tous les cas

Exemple :

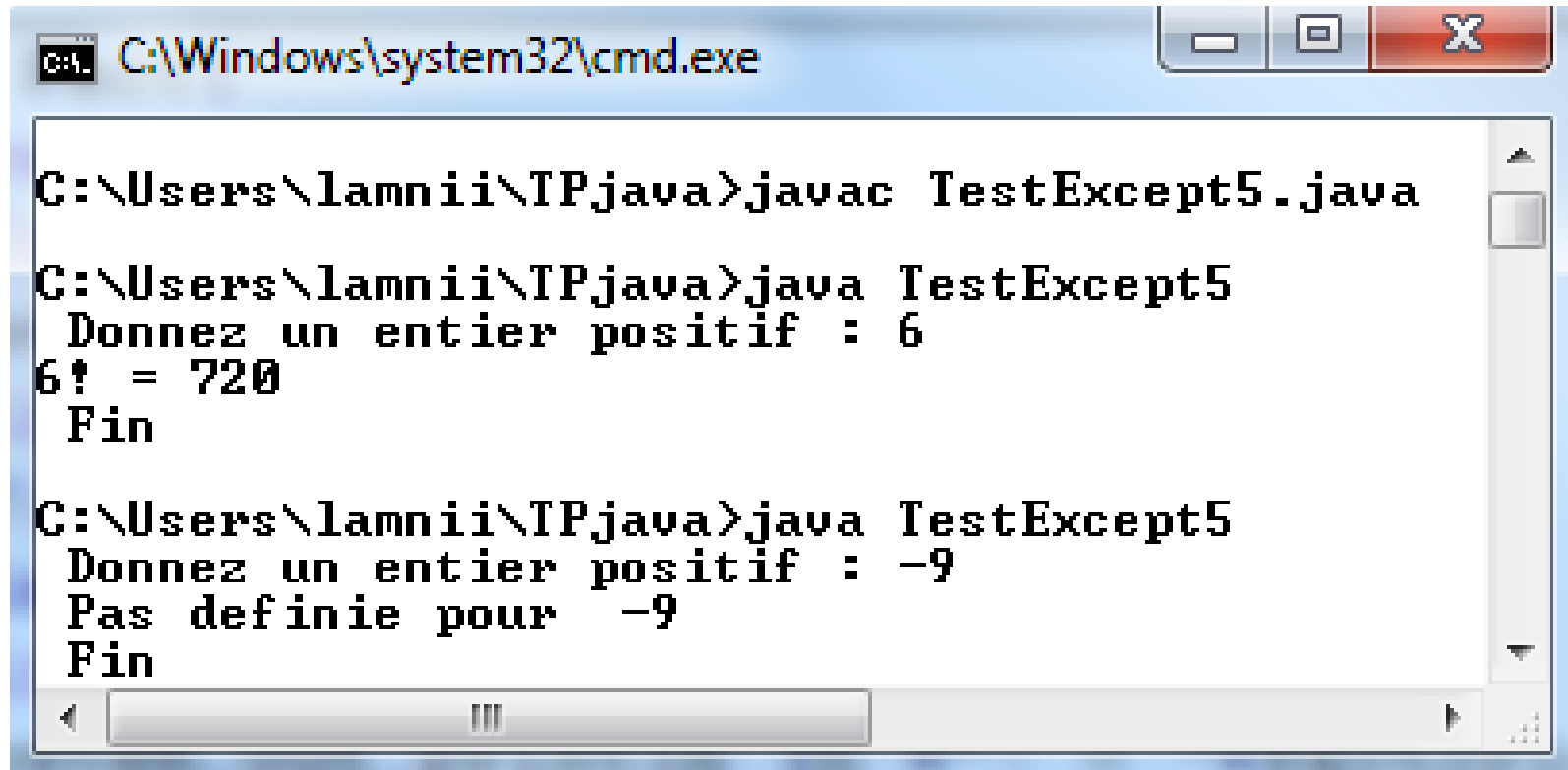
Gestion des exceptions : finally

```
class Fact{
    private int n;
    public Fact(int n)throws PasDef {
        if(n<0) throw new PasDef(); this.n=n;}
    public int factorial(int n)
    { if(n==0) return 1;
      else return (n*factorial(n-1)); }
    public void affiche(){
        System.out.println (n +"! = "+factorial(n)) ; }
}

class PasDef extends Exception { }

public class TestExcept5
{ public static void main (String args[])
  { int n; System.out.print(" Donnez un entier positif : ") ;
    n=Clavier.lireInt() ;
    try
    { Fact factDen1= new Fact(n); factDen1.affiche(); }
    catch(PasDef e)
    { System.out.println (" Pas definie pour " + n) ; }
    finally { System.out.println (" Fin ") ; }
  }
}
```

Gestion des exceptions : finally



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window shows the following commands and output:

```
C:\Users\lannii\TPjava>javac TestExcept5.java
C:\Users\lannii\TPjava>java TestExcept5
Donnez un entier positif : 6
6! = 720
Fin
C:\Users\lannii\TPjava>java TestExcept5
Donnez un entier positif : -9
Pas definie pour -9
Fin
```

The window includes standard Windows window controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

Interface Graphique: awt & swing

→ Java, permet de créer des interfaces graphiques homme machines (IHM).

→ Tous les outils nécessaires pour réaliser des interfaces graphiques existent dans les deux paquetages suivants : [java.awt](#) et [javax.swing](#).

→ Les interfaces (IHM) font intervenir de nos jours des éléments que l'on retrouve dans la majorité des systèmes d'exploitation : les fenêtres, les listes de choix, les menus déroulants, les boutons, les boutons radios, etc...

awt = 1^{ère} boîte à outil de java

awt: aspect change d'une plateforme à une autre.

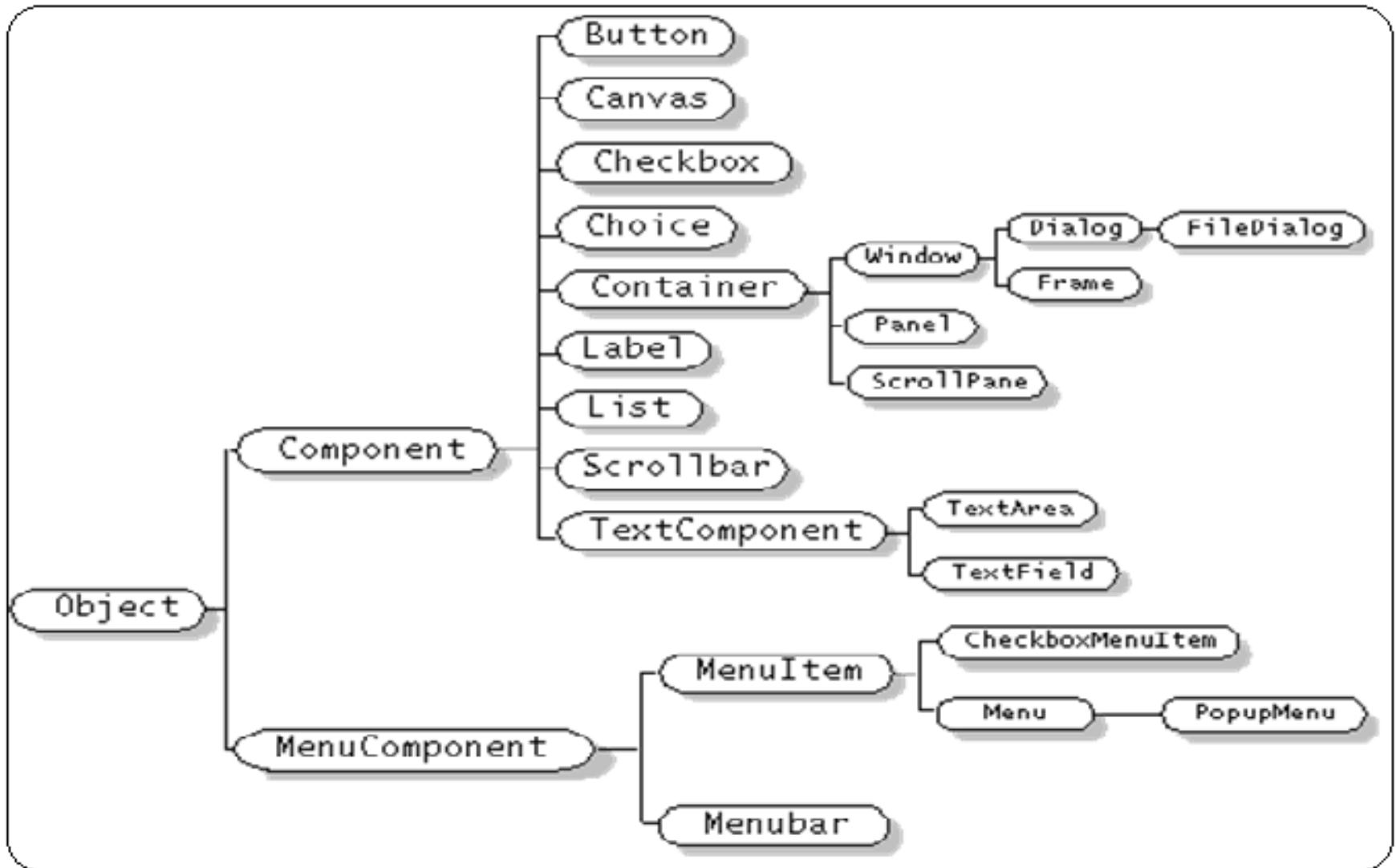
swing = extension

swing : faire que tout fonctionne de manière identique partout.

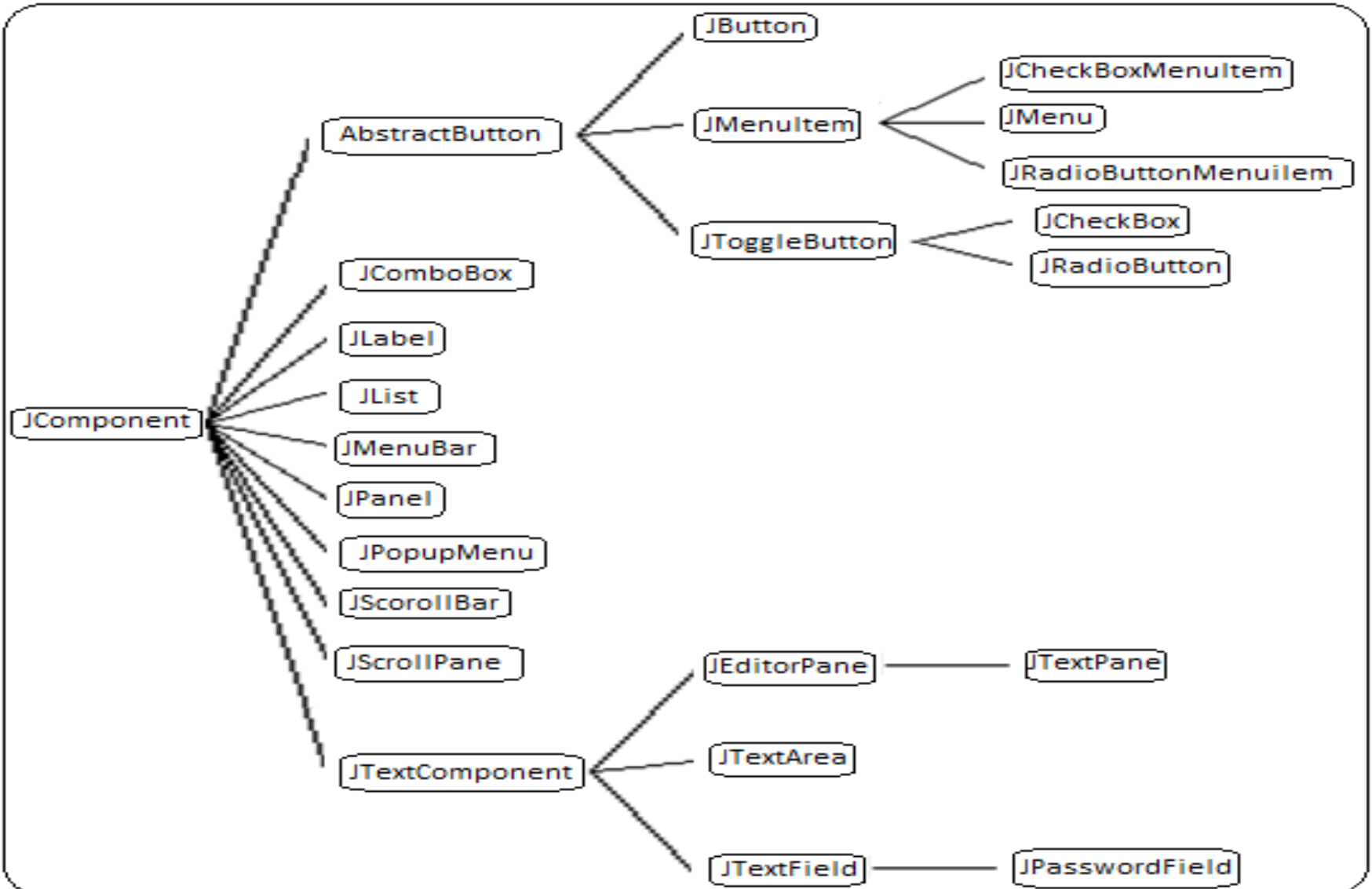
Interface Graphique: awt & swing

- awt (Abstract Window Toolkit) (java.awt):
 - C'est pour construire des IHM que le package awt est inclus dans toutes les versions de Java.
 - awt s'appuie sur les ressources systèmes
- Catégories de classes :
 - graphique (couleurs, fontes, formes, ...)
 - composants (Component) (fenêtres, boutons, menus, ...)
 - gestionnaires (position des composants)

awt

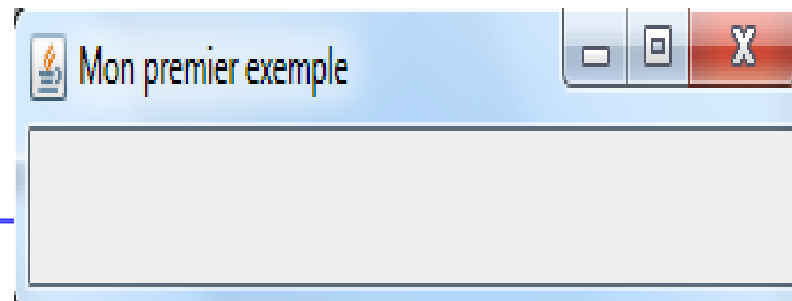


swing



Mon premier exemple

```
import javax.swing.*;
import java.awt.*;
class Fenetre1 extends JFrame {
    public Fenetre1 () {
        //super("Mon premier exemple") ;
        setTitle("Mon premier exemple") ;
        setBounds(85,55,350,400) ;
    }
}
public class Exemple1 {
    public static void main(String args[]) {
        JFrame F1 = new Fenetre1() ;
        F1.setVisible(true) ;
    }
}
```



Interface Graphique: Structure

- conteneurs: JApplet, JFrame, JPanel, JSplitPane, JTabbedPane,...
- composants « atomiques »: JButton, JList, JPopupMenu
- gestionnaire de disposition : LayoutManager
- interaction avec l'utilisateur : gestionnaire d'évènements
- Boite de dialogue

Interface Graphique: conteneurs

JFrame

- Dans une application Swing, oninstanciera un cadre **JFrame** qui permet d'avoir une fenêtre principale.
- Une JFrame est une fenêtre avec un titre et une bordure.
- La JFrame est un composant qui contient tous les autres.
- La plupart des composants graphiques ont une taille par défaut, qui peut d'ailleurs être nulle.

Interface Graphique: conteneurs

```
import javax.swing.*;
import java.awt.*;
public class Fenetre2 {
    public static void main(String[] arg) {
        JFrame cadre = new javax.swing.JFrame("Dixième exemple");
        JPanel panneau = new JPanel();
        panneau.setPreferredSize(new Dimension(324, 240));
        panneau.setBackground(Color.YELLOW);
        cadre.setContentPane(panneau);
        cadre.setLocation(500, 500);
        cadre.pack();
        JLabel label = new JLabel("Bonjour");
        cadre.setVisible(true);
        cadre.add(label);
        cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Interface Graphique: conteneurs

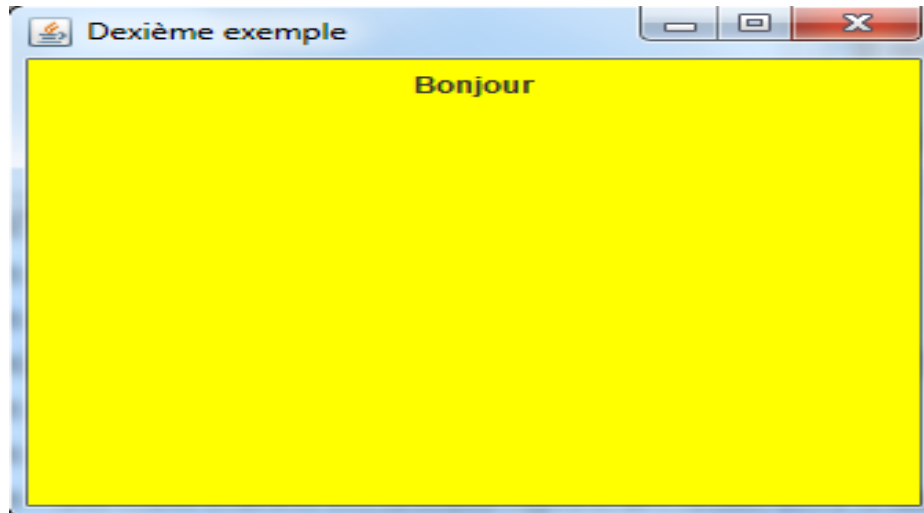
- `JFrame cadre = new javax.swing.JFrame(" Deuxième exemple ");`
on instancie ici la classe `JFrame` en passant au constructeur une chaîne de caractères pour le titre de la fenêtre.
- `panneau.setPreferredSize(new Dimension(324, 240));`
le panneau ait 324pixels de largeur et 240 pixels de hauteur.
- `panneau.setBackground(Color.YELLOW);`
le panneau ait la couleur jaune pour couleur de fond.
- `cadre.setContentPane(panneau);` le conteneur de la fenêtre soit un panneau.
- `cadre.setLocation(500, 500);` : on positionne le cadre.
- `cadre.pack();` : la méthode `pack`: calcule la dimension de la fenêtre en fonction de ce qui est mis à l'intérieur.
- `cadre.setVisible(true);` par défaut, une fenêtre est invisible ; on demande ici à ce que la fenêtre soit visible

Interface Graphique: conteneurs

→ `JLabel label = new JLabel("Bonjour");` on instancie ici un label en passant au constructeur une chaîne de caractères

→ `cadre.add(label);` l'ajout de label

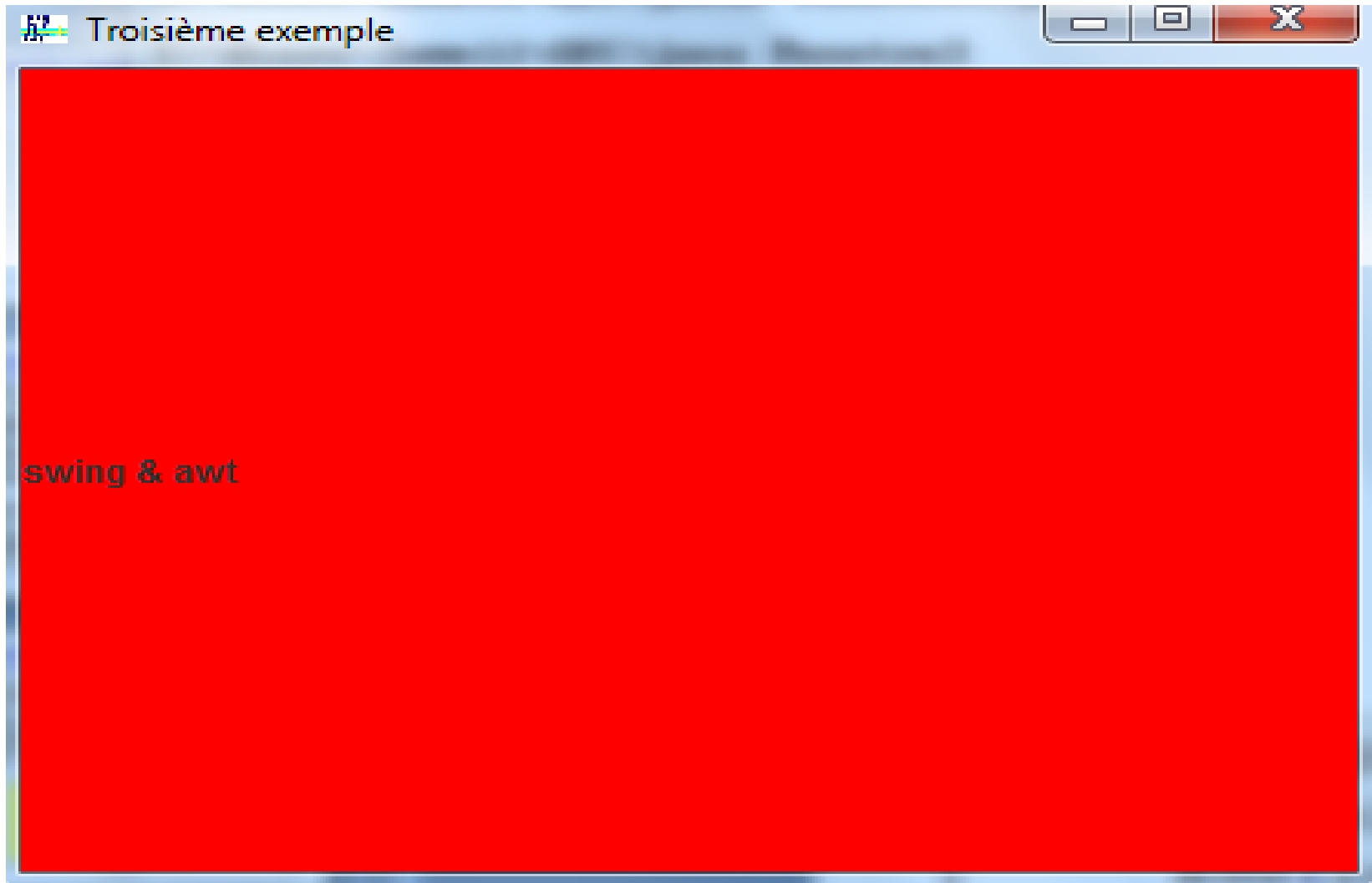
→ `cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`
par défaut, lorsqu'on ferme une fenêtre, cela ne termine pas l'application. On demande par cette instruction que l'application se termine quand on ferme la fenêtre.



Interface Graphique: conteneurs

```
import javax.swing.*;
import java.awt.*;
public class Fenetre3 extends JFrame {
    public Fenetre3() {
        setTitle("Troisième exemple");
        setBounds(120,150,450,350);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Toolkit kit= Toolkit.getDefaultToolkit();
        setIconImage(kit.getImage("fst.jpg"));
        getContentPane().setBackground(Color.RED);
        JLabel label = new JLabel("swing & awt");
        add(label);
        setResizable(true);
        setVisible(true);
    }
    public static void main(String[] arg) {
        new Fenetre3();
    }
}
```

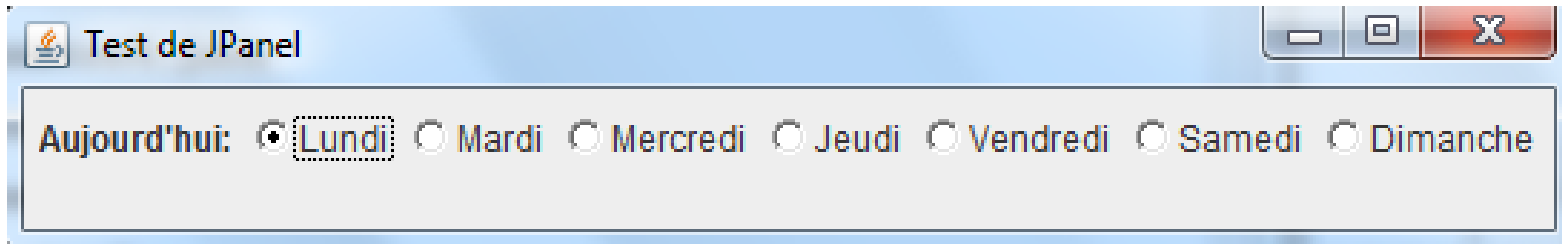
Interface Graphique: conteneurs



Interface Graphique: conteneurs

JPanel

- Un **JPanel** est un conteneur élémentaire destiné à contenir d'autres composants. Il est muni d'un gestionnaire de placement
- Il organise les éléments au fur et à mesure qu'ils sont intégrés à l'interface, de gauche à droite, à la manière des mots d'une phrase.



Interface Graphique: conteneurs

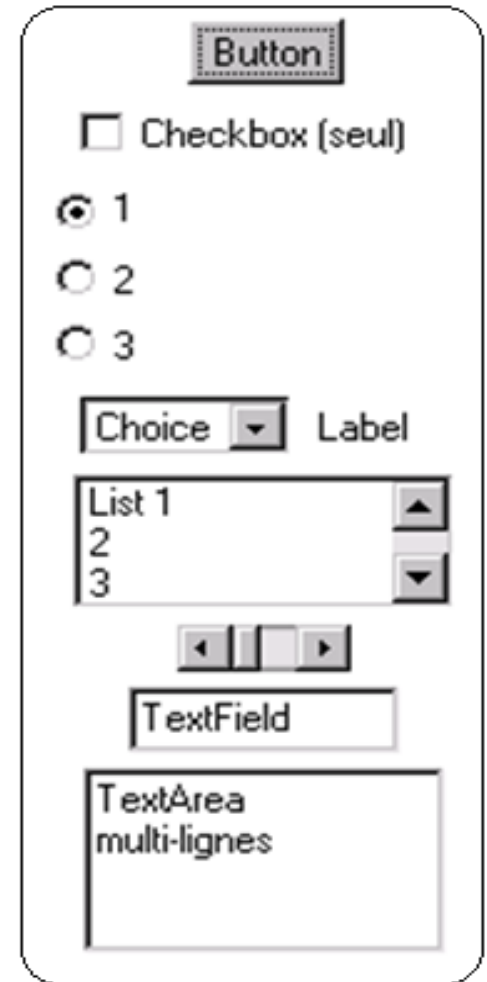
```
import javax.swing.*;
import java.awt.*;
public class Fenetre4 extends JFrame {
    public Fenetre4() {
        setTitle("Test de JPanel");
        setBounds(120,150,550,350);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel lab = new JLabel("Aujourd'hui: ");
        JPanel p = new JPanel();
        p.add(lab);
        CheckboxGroup gr = new CheckboxGroup();
        p.add( new Checkbox("Lundi", true, gr) );
        p.add( new Checkbox("Mardi", false, gr) );
        p.add( new Checkbox("Mercredi", false, gr) );
        p.add( new Checkbox("Jeudi", false, gr) );
        p.add( new Checkbox("Vendredi", false, gr) );
        p.add( new Checkbox("Samedi", false, gr) );
        p.add( new Checkbox("Dimanche", false, gr) );
        setContentPane(p);
        setVisible(true);
    }
    public static void main(String[] arg) {
        new Fenetre4();
    }
}
```


Interface Graphique: LES COMPOSANTS GRAPHIQUES

Les panneaux contiennent

- des étiquettes (JLabel)
- un champ de texte (JTextField)
- une liste (JList) et une combobox (JComboBox)
- des boutons à cocher (JCheckBox)
- des bordures (TitledBorder)
- des boutons radio (JRadioButton)
- des boutons ordinaires (JButton)
- zone de saisie multilignes (JTextArea)
- JScrollBar

...



Interface Graphique: LES COMPOSANTS GRAPHIQUES

Jlabel :

Un Label affiche une seule ligne de texte (étiquette) non modifiable.

En général, les étiquettes ne traitent pas d'événements.

→ Pour afficher un texte, on peut utiliser la méthode `setText` ou une version surchargée du constructeur.

→ Pour afficher une image, on doit utiliser un composant intermédiaire, l'icone (`ImageIcon`).

→ Le constructeur `ImageIcon` est capable de charger des images jpeg, gif et png.

→ Le constructeur de `JLabel` est surchargé pour accepter un objet `ImageIcon` en paramètre.

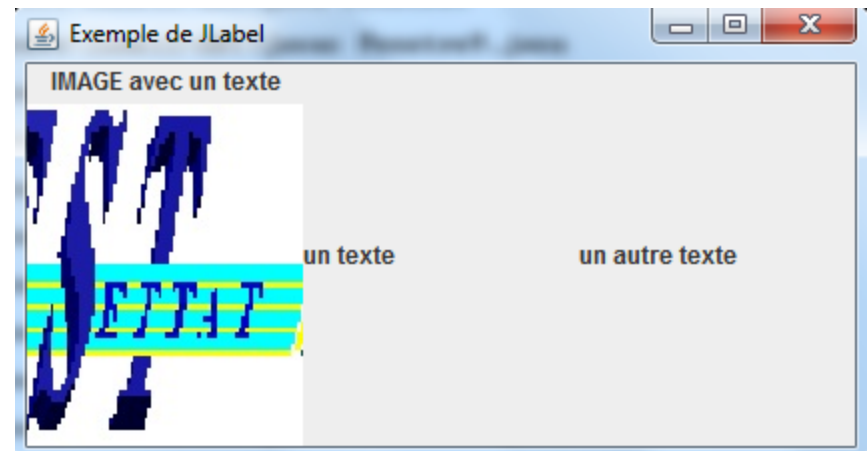
Interface Graphique: LES COMPOSANTS GRAPHIQUES

Jlabel :

Emplacement :

1. JLabel.BOTTOM
2. JLabel.CENTER
3. JLabel.LEFT
4. JLabel.RIGHT
5. JLabel.TOP

Exmple:



Interface Graphique: LES COMPOSANTS GRAPHIQUES

```
import java.awt.*;
import javax.swing.*;
public class Fenetre9 extends JPanel {
    JLabel jlbLabel1, jlbLabel2, jlbLabel3;
    public Fenetre9() {
        ImageIcon icon = new ImageIcon("fst.JPG", "FST de SETTAT");
        setLayout(new GridLayout(1,3));
        jlbLabel1 = new JLabel("IMAGE avec un texte", icon, JLabel.CENTER);
        jlbLabel1.setVerticalTextPosition(JLabel.TOP);
        jlbLabel1.setHorizontalTextPosition(JLabel.CENTER);
        jlbLabel2 = new JLabel("un texte");
        jlbLabel3 = new JLabel("un autre texte");
        add(jlbLabel1); add(jlbLabel2); add(jlbLabel3);
    }
    public static void main(String[] args) {
        JFrame frame = new JFrame("Exemple de JLabel");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setContentPane(new Fenetre9());
        frame.pack();    frame.setVisible(true);    }
}
```

Interface Graphique: LES COMPOSANTS GRAPHIQUES

JTextField :

- Le champ de texte est un dispositif d'entrée de texte sur une seule ligne.
- Il est source d'ActionEvent
- On peut définir un champ de texte comme étant éditable ou non.

Méthodes :

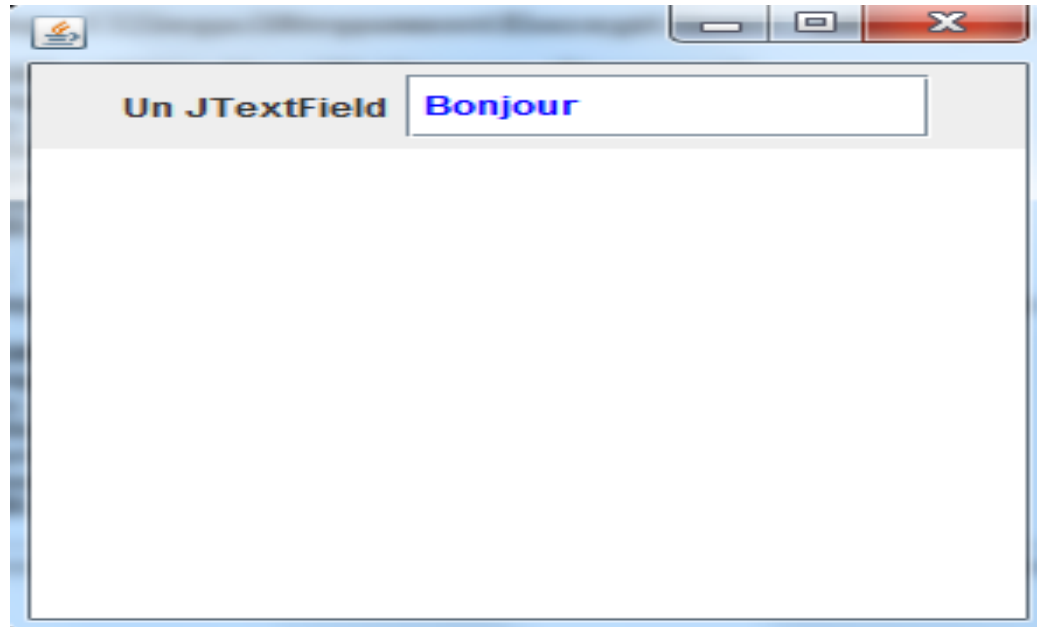
`void setText(String text)`

`String getText()` pour mettre ou retirer du texte dans le TextField

Exemple:

```
import java.awt.*;
import javax.swing.*;
public class Fenetre10 extends JFrame {
    private JPanel container = new JPanel();
    private JTextField jtf = new JTextField("Valeur par défaut");
    private JLabel label = new JLabel("Un JTextField");
    public Fenetre10(){ this.setSize(300, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocationRelativeTo(null);
        container.setBackground(Color.white);
        container.setLayout(new BorderLayout());
        JPanel top = new JPanel(); Font police = new Font("Arial", Font.BOLD, 12);
        jtf.setFont(police); jtf.setForeground(Color.BLUE);
        jtf.setPreferredSize(new Dimension(150, 30));
        top.add(label); top.add(jtf); container.add(top, BorderLayout.NORTH);
        this.setContentPane(container); this.setVisible(true);    }
    public static void main(String[] args) {
        JFrame frame = new JFrame("Exemple de JLabel");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setContentPane(new Fenetre10());
        frame.pack();    frame.setVisible(true);    }
}
```

Interface Graphique: LES COMPOSANTS GRAPHIQUES



Interface Graphique: LES COMPOSANTS GRAPHIQUES

JList : Constructeurs :

JList() // modèle vide

JList(ListModel dataModel) // cas général

JList(Object[] listData) // tableau

JList(Vector listData) // vecteur

Création d'une liste à partir d'un tableau (liste non modifiable) :

```
String[] data = {"A", "B", "C", "D"};
```

```
JList liste = new JList(data);
```

Accès au modèle, donc au contenu:

```
for(int i = 0; i < list.getModel().getSize(); i++) {  
    System.out.println(list.getModel().getElementAt(i));  
}
```


Interface Graphique: LES COMPOSANTS GRAPHIQUES

La classe `JComboBox` permet la sélection d'une entrée parmi une séquence.

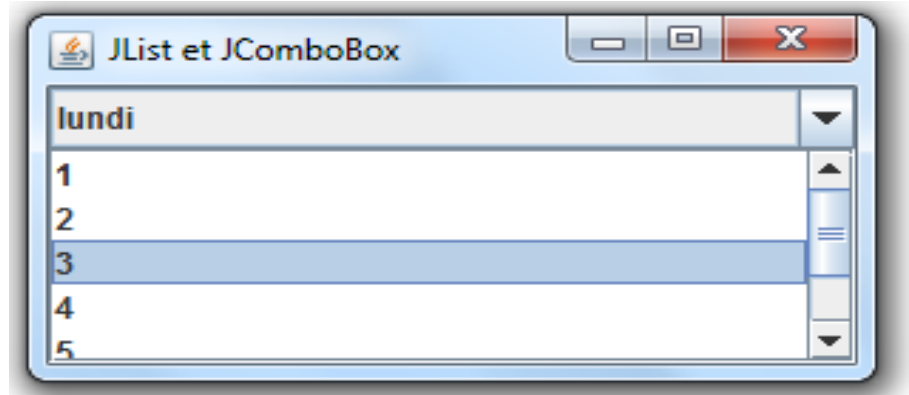
→ Le composant est constitué de deux parties :

- un éditeur qui affiche la sélection courante et permet ou non d'entrée des valeurs et
- une liste déroulante qui affiche les choix possibles.

→ Deux modes de fonctionnement différents :

- Non éditable : `ComboBoxModel`
- Editable : `MutableComboBoxModel`

Exemple:



```

import java.awt.*;
import javax.swing.*;
public class Fenetre11 extends JFrame {
    public static void main(String args[]) {
        String labels1[] = { "1", "2", "3", "4", "5", "6", "7" };
        String labels2[] = { "lundi", "Mardi", "Mercredi", "Jeudi",
            "Vendredi", "Samedi", "Dimanche" };
        JFrame f = new JFrame("JList et JComboBox");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JList list = new JList(labels1);
        JScrollPane scrollPane = new JScrollPane(list);
        JComboBox comboBox = new JComboBox(labels2);
        comboBox.setMaximumRowCount(4);
        Container contentPane = f.getContentPane();
        contentPane.add(comboBox, BorderLayout.NORTH);
        contentPane.add(scrollPane, BorderLayout.CENTER);
        f.setSize(300, 150); f.setVisible(true);
    }
}

```

Interface Graphique: LES COMPOSANTS GRAPHIQUES

JCheckBox :

- boîte à cocher qui peut être sélectionnée ou non.
- Il est source d'ActionEvent
- On peut définir un champ de texte comme étant éditable ou non.

Méthodes :

La sélection ou la désélection est notifiée par un ItemEvent à un écouteur implémentant l'interface ItemListener.

la méthode getStateChange() de ItemEvent retourne une constante : ItemEvent.DESELECTED ou ItemEvent.SELECTED.

la méthode getItem() de ItemEvent renvoie la chaîne contenant l'étiquette de la case à cocher considérée

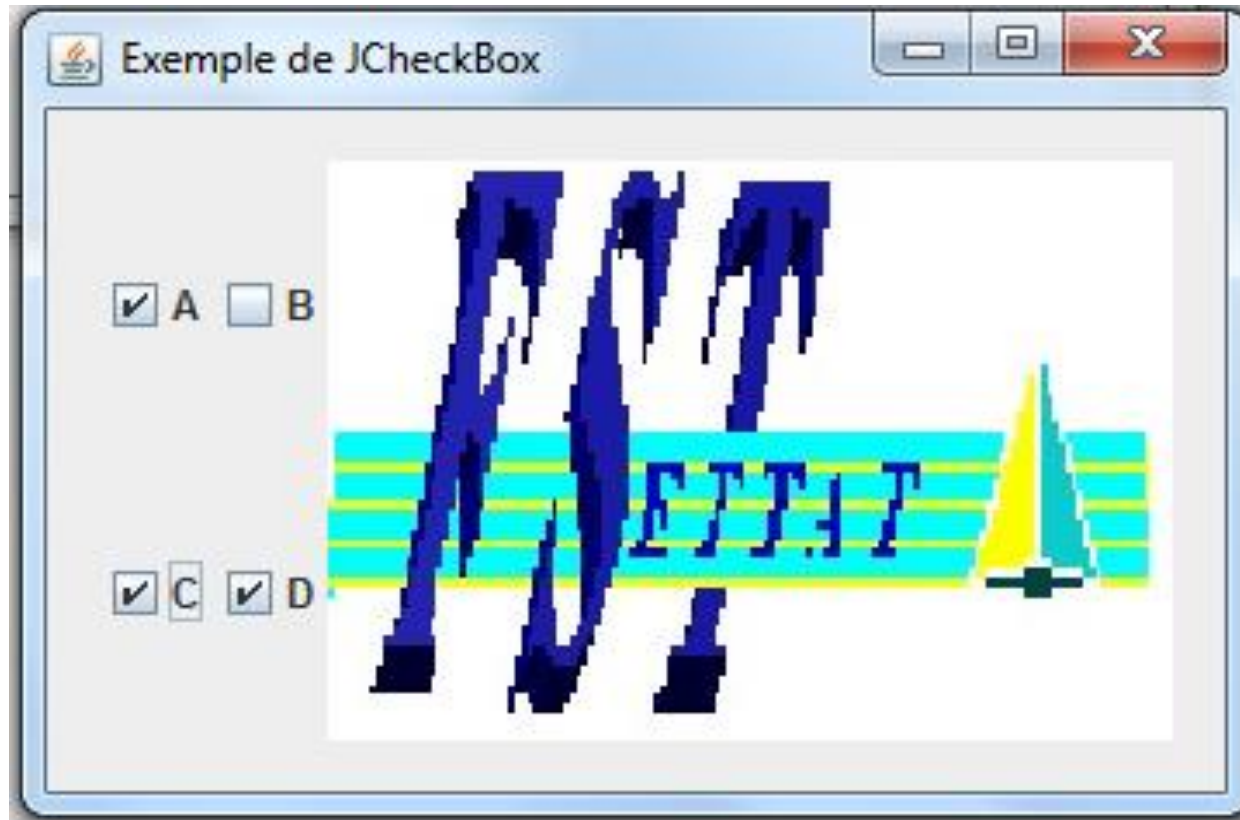
Exemple:

```

import java.awt.*;
import javax.swing.*;
public class Fenetre12 extends JPanel {
    JCheckBox jA, jB, jC, jD;          JLabel image;
    public Fenetre12() {
        jA = new JCheckBox("A"); jB = new JCheckBox("B");
        jC = new JCheckBox("C"); jD = new JCheckBox("D");
        image = new JLabel(new ImageIcon("fst.jpg"));
        JPanel jplCheckBox = new JPanel();
        jplCheckBox.setLayout(new GridLayout(0, 2)); // 0 rows, 2 Column
        jplCheckBox.add(jA); jplCheckBox.add(jB);
        jplCheckBox.add(jC); jplCheckBox.add(jD);
        setLayout(new BorderLayout());
        add(jplCheckBox, BorderLayout.WEST);
        add(image, BorderLayout.CENTER);
        setBorder(BorderFactory.createEmptyBorder(15,15,15,15));    }
    public static void main(String s[]) {
        JFrame frame = new JFrame("Exemple de JCheckBox");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setContentPane(new Fenetre12());
        frame.pack();
        frame.setVisible(true);    }
}

```

Interface Graphique: LES COMPOSANTS GRAPHIQUES

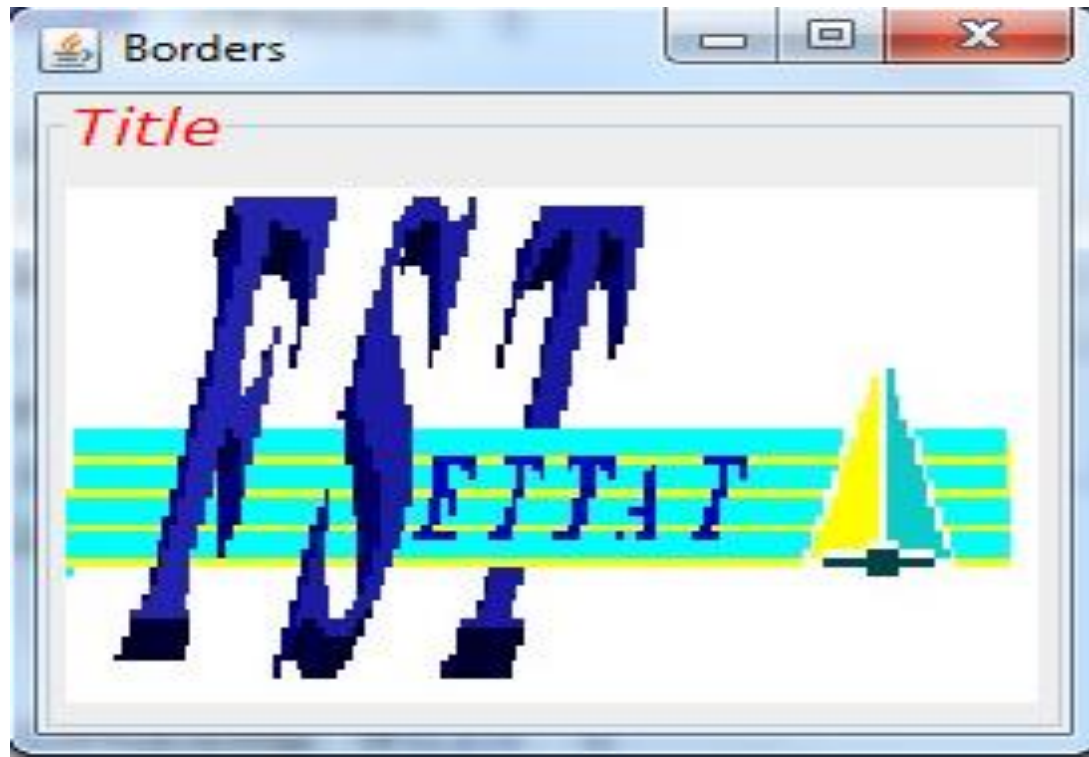


Interface Graphique: LES COMPOSANTS GRAPHIQUES

TitledBorder :

→ Une bordure permettant l'inclusion d'une chaîne de caractères

Exemple:



```

import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
public class Fenetre13 extends JPanel {
    public Fenetre13() {
        this.setLayout(new GridLayout(1, 1, 5, 5));
        JLabel label = new JLabel(new ImageIcon("fst.jpg"));
        label.setHorizontalAlignment(JLabel.CENTER);
        TitledBorder titled = new TitledBorder("Title");
        titled.setTitleColor(Color.red);
        titled.setTitleFont(new Font("Verdana",Font.ITALIC,16));
        label.setBorder(titled);
        add(label);
    }
    public static void main(String s[]) {
        JFrame frame = new JFrame("Borders");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(250, 250);
        frame.setContentPane(new Fenetre13());
        frame.setVisible(true);
    }
}

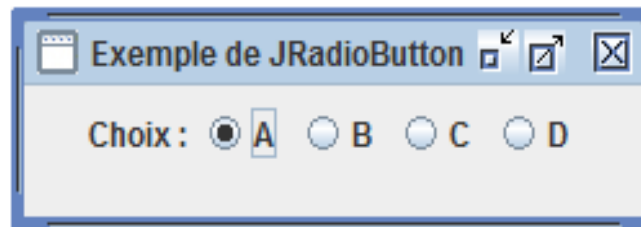
```

Interface Graphique: LES COMPOSANTS GRAPHIQUES

JRadioButton :

- possède des méthodes constructeurs dont les arguments et les fonctionnalités sont les mêmes.
- Pour les organiser en groupe, créer une classe d'objet : [ButtonGroup](#)

Exemple:




```

import java.awt.FlowLayout; import javax.swing.ButtonGroup;
import javax.swing.JFrame; import javax.swing.JLabel;
import javax.swing.JRadioButton;
public class Fenetre14 {
    public static void main(String[] args) {
        JFrame.setDefaultLookAndFeelDecorated(true); // change style
        JFrame frame = new JFrame("Exemple de JRadioButton");
        frame.setLayout(new FlowLayout());
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JRadioButton button1 = new JRadioButton("A");
        JRadioButton button2 = new JRadioButton("B");
        JRadioButton button3 = new JRadioButton("C");
        JRadioButton button4 = new JRadioButton("D");
        ButtonGroup colorButtonGroup = new ButtonGroup();
        colorButtonGroup.add(button1);
        colorButtonGroup.add(button2);
        colorButtonGroup.add(button3);
        button1.setSelected(true);
        frame.add(new JLabel("Choix :"));
        frame.add(button1);      frame.add(button2);
        frame.add(button3);      frame.add(button4);
        frame.pack();   frame.setSize(250, 80);
        frame.setVisible(true);
    }
}

```

Interface Graphique: LES COMPOSANTS GRAPHIQUES

JButton : Constructeurs :

```
JButton (String text);  
JButton (ImageIcon picture);  
JButton (String text, ImageIcon picture);
```

Méthodes :

```
void addActionListener (ActionListener object);  
void setText (String text);  
void setActionCommand (String cmd);  
void setIcon (ImageIcon icon);  
void requestFocus();
```

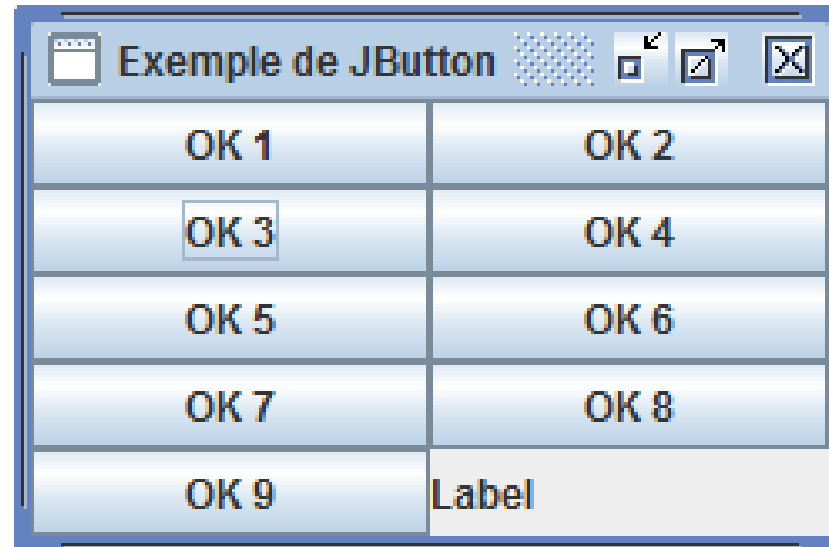
Exemple:

```

import java.awt.*;
import javax.swing.*;
public class Fenetre15 extends JPanel {
    public Fenetre15() {
        setSize(250, 100);
        this.setLayout(new GridLayout(0, 2));
        for (int i=0; i<9; i++)
            add(new JButton("OK " + (i+1)));
        add(new JLabel("Label"));
    }
    public static void main(String s[]) {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame("Exemple de JButton");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setContentPane(new Fenetre15());
        frame.pack();    frame.setVisible(true);    }
}

```

Interface Graphique: LES COMPOSANTS GRAPHIQUES



Interface Graphique: LES COMPOSANTS GRAPHIQUES

JTextArea : zone de texte plusieurs lignes

Constructeurs.

JTextArea() Construit une nouvelle instance de JTextArea. JTextArea(int l, int

c) Construit une nouvelle instance de JTextArea, lignes l et c colonnes.

JTextArea(String texte) Construit un JTextArea avec un texte initial.

JTextArea(String texte, int l, int c)

Méthodes:

String getText()

String getSelectedText()

void append(String t)

void replaceRange(String t,int d,int f)

void replaceSlection(String t)

String getText(int d, int l)

void setText(String t)

void insert(String t, int pos)

int getColumns()

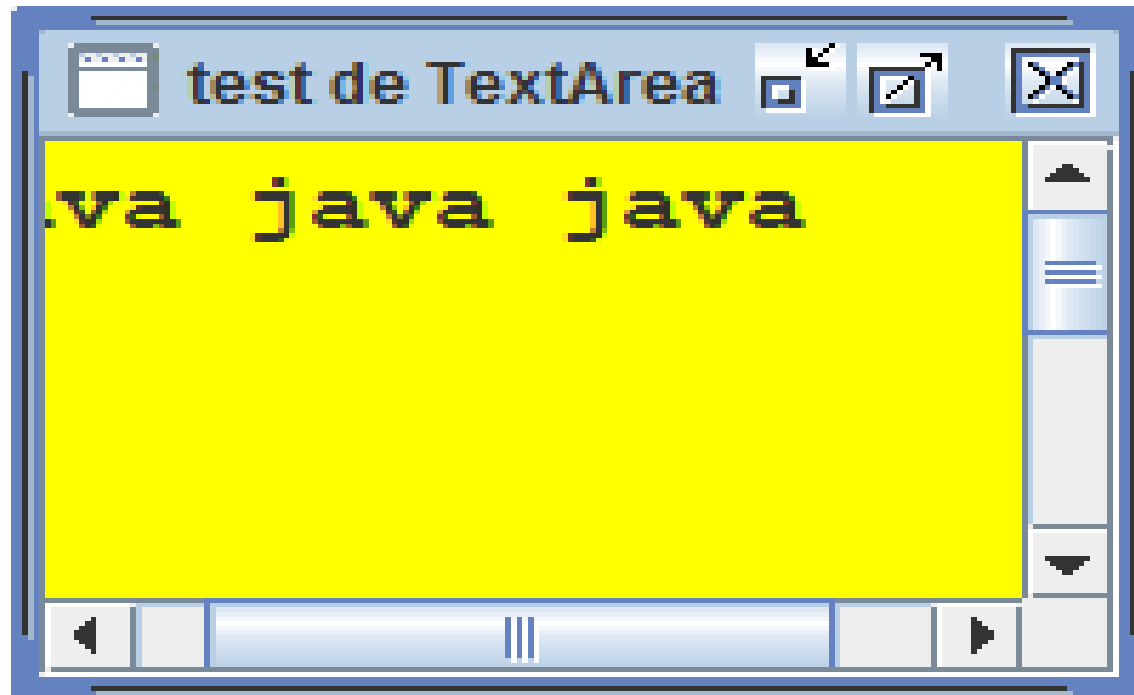
void setEditable(boolean)

Exemple:

```

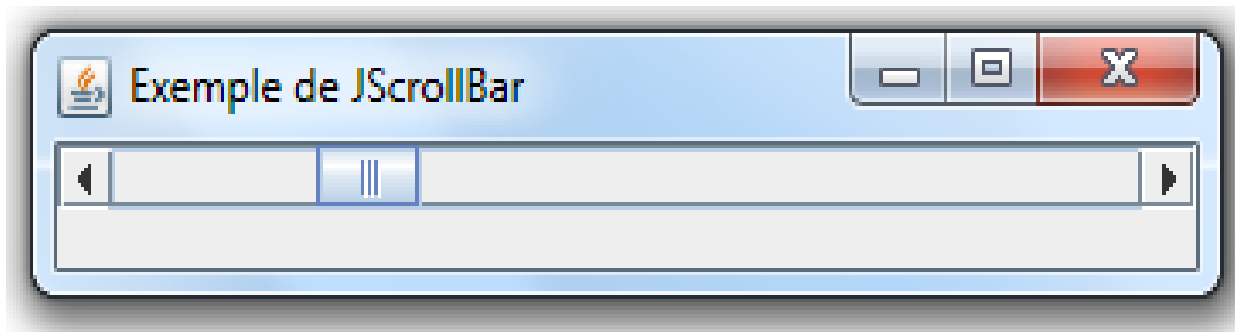
import java.awt.*; import javax.swing.*;
public class Fenetre16 extends JFrame {
    JTextArea jtexeArea = new JTextArea(10, 20);
    public Fenetre16() {
        jtexeArea.setText("java java java");
        JScrollPane scrollingArea = new JScrollPane(jtexeArea);
        JPanel P = new JPanel();
        P.setLayout(new BorderLayout());
        P.add(scrollingArea, BorderLayout.CENTER);
        jtexeArea.setBackground (Color.yellow);
        Font font = new Font("Courier", Font.BOLD,18);
        jtexeArea.setFont(font);  this.setContentPane(P);
        this.setTitle("test de TextArea"); this.pack();
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  }
    public static void main(String[] args) {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new Fenetre16();
        frame.setSize(200, 150);  frame.setVisible(true);  }
}

```



Interface Graphique: LES COMPOSANTS GRAPHIQUES

JScrollBar : barre de défilement




```

import java.awt.*; import javax.swing.*;
import java.awt.event AdjustmentEvent;
import java.awt.event AdjustmentListener;

public class Fenetre17 {
    public static void main(String args[]) {
        AdjustmentListener A = new AdjustmentListener() {
            public void adjustmentValueChanged(AdjustmentEvent A) {
                System.out.println("Valeur : " + A.getValue());    }    };
        JScrollBar oneJScrollBar = new JScrollBar(JScrollBar.HORIZONTAL);
        oneJScrollBar.addAdjustmentListener(A);
        JFrame frame = new JFrame("Exemple de JScrollBar");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(oneJScrollBar, BorderLayout.NORTH);
        frame.setSize(350, 70);
        frame.setVisible(true);
    }
}

```

Interface Graphique: Menu

Menu :

Pour créer un menu déroulant, il y a essentiellement 3 opérations :

→ création de la barre menu :

```
JMenuBar menuBar=new JMenuBar();
```

→ création des menus :

```
JMenu couleur=new JMenu("couleur"); menuBar.add(couleur);
```

```
JMenu taille=new JMenu("taille"); menuBar.add(taille);
```

→ création des options :

```
rouge=new JMenuItem("rouge");
```

```
rouge.addActionListener(this);couleur.add(rouge);
```

```
bleu=new JMenuItem("bleu");
```

```
bleu.addActionListener(this);couleur.add(bleu);
```

Naturellement, il faudra ajouter la barre des menus à la fenêtre, et adjoindre aux différentes options comme indiqué ci-dessus.

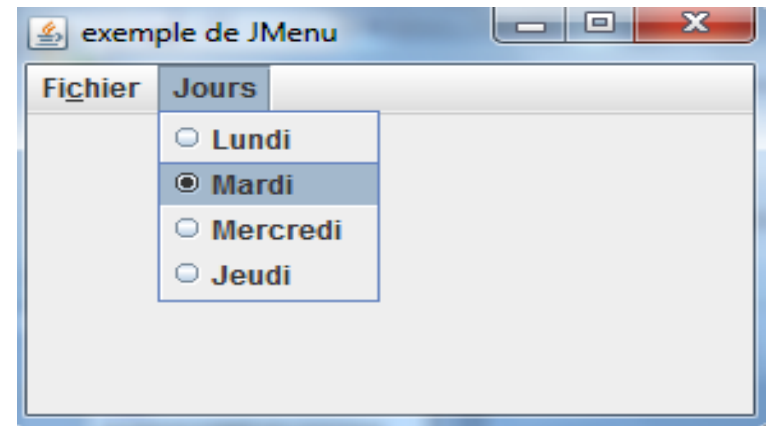
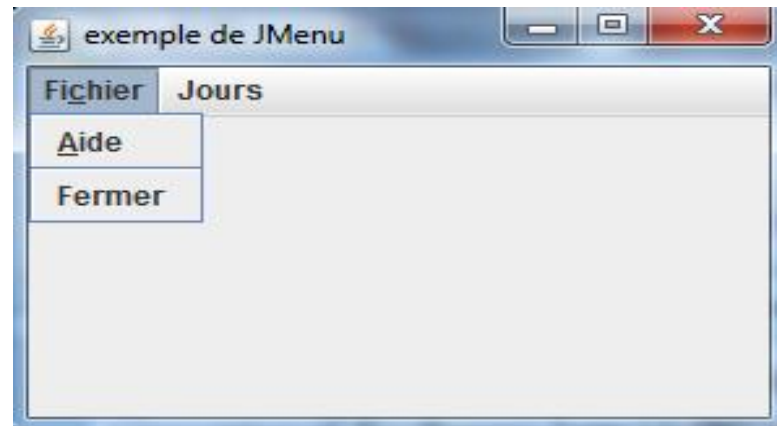
Exemple 1:

```

import java.awt.*; import java.awt.event.*; import javax.swing.*;
public class Fenetre19A extends JFrame {
    private JMenu fichier;    private JMenu jours;
    private JMenu formatMenu;    private JRadioButtonMenuItem joursItems[];
    private String joursT[]    = {"Lundi","Mardi","Mercredi","Jeudi"};
    private ButtonGroup joursButtonGroup;
    public Fenetre19A() {
        super("exemple de JMenu");
        Container container = getContentPane();
        createFileMenu(); createFormatMenu();    JMenuBar bar = new JMenuBar();
        setJMenuBar(bar); bar.add(fichier);    bar.add(jours);    }
    private void createFileMenu() {
        fichier = new JMenu("Fichier");    fichier.setMnemonic('F');
        JMenuItem aboutItem = new JMenuItem("Aide");
        aboutItem.setMnemonic('A');    aboutItem.setEnabled(true);
        JMenuItem exitItem = new JMenuItem("Fermer");
        exitItem.setMnemonic('x');    exitItem.setEnabled(true);
        fichier.add(aboutItem);    fichier.addSeparator();
        fichier.add(exitItem);    }
    private void createJoursMenu() {
        jours = new JMenu("Jours"); fichier.setMnemonic('C');
        joursItems = new JRadioButtonMenuItem[joursT.length];
        joursButtonGroup = new ButtonGroup();
        for (int index=0;index < joursT.length;index++) {
            joursItems[index] = new JRadioButtonMenuItem(joursT[index]);
            jours.add(joursItems[index]); joursButtonGroup.add(joursItems[index]);
        }
        joursItems[0].setSelected(true);    }
    private void createFormatMenu() {    formatMenu = new JMenu("Format");
        formatMenu.setMnemonic('r'); createJoursMenu(); formatMenu.add(jours);    }
}

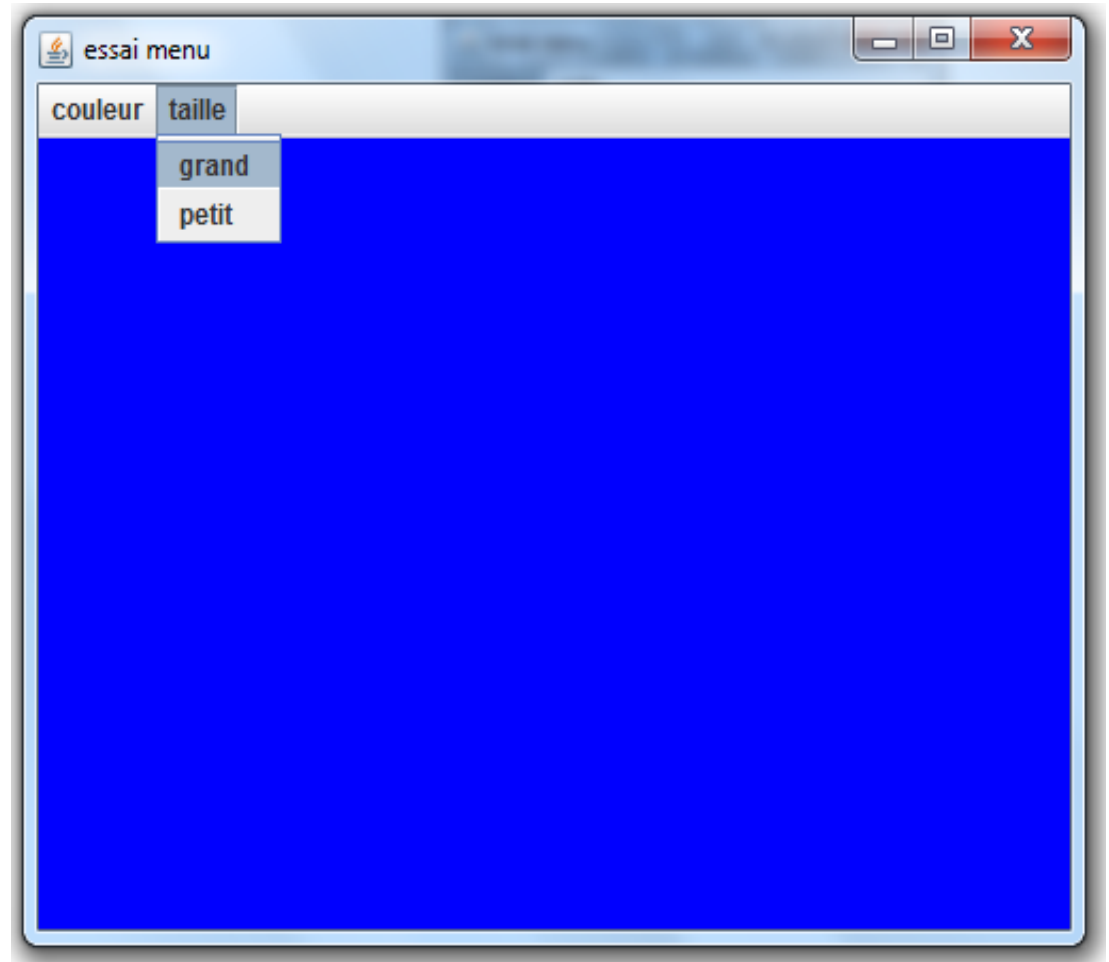
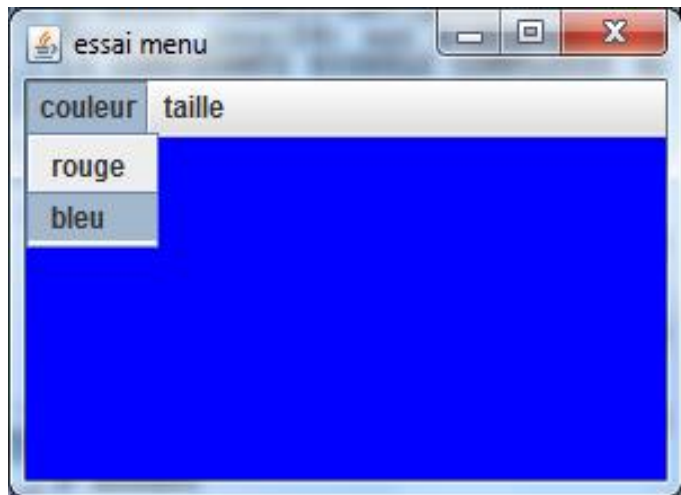
```

```
import javax.swing.JFrame;  
import java.awt.*;  
public class Fenetre19B {  
    public static void main(String Args[]) {  
        Fenetre19A menu = new Fenetre19A();  
        menu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        menu.setSize(300,200);  
        menu.setBackground(Color.blue);  
        menu.setVisible(true);  
    }  
}
```



Interface Graphique: Menu → Exemple 2

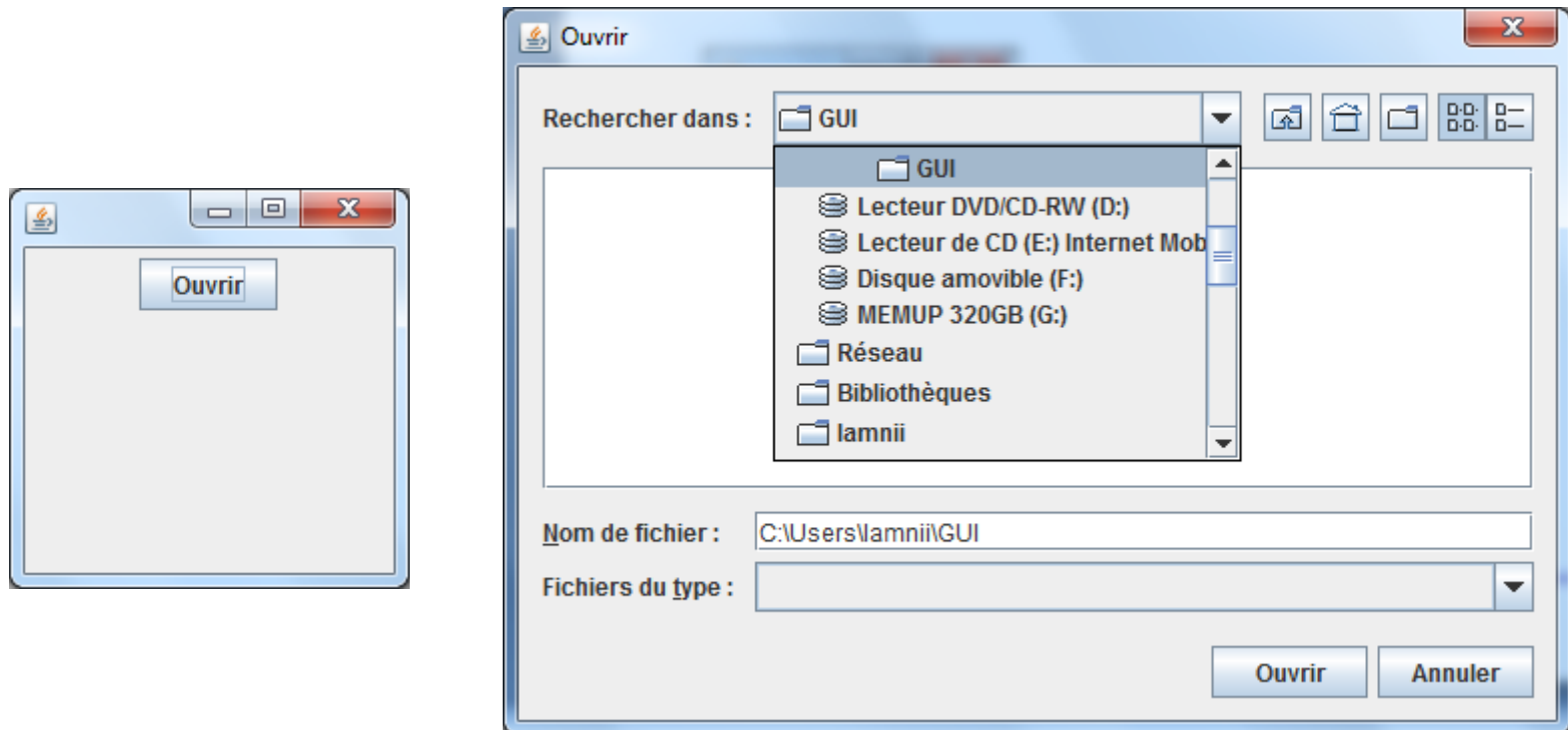
```
import javax.swing.* ; import java.awt.event.*; import java.awt.*;
public class Fenetre extends JFrame implements ActionListener{
    JMenuItem rouge,bleu,petit,grand ;    JPanel p0;
    public Fenetre() { super("essai menu"); setSize(300,200); setLocation(100,100);
    WindowListener l=new WindowAdapter(){
    public void windowClosing(WindowEvent e){ System.exit(0);}};
    addWindowListener(l); Container cont=getContentPane();
    cont.setLayout(new BorderLayout()); JMenuBar menuBar=new JMenuBar();
    JMenu couleur=new JMenu("couleur");menuBar.add(couleur);
    JMenu taille=new JMenu("taille");menuBar.add(taille);
    rouge=new JMenuItem("rouge"); rouge.addActionListener(this);couleur.add(rouge);
    bleu=new JMenuItem("bleu"); bleu.addActionListener(this);couleur.add(bleu);
    grand=new JMenuItem("grand"); grand.addActionListener(this);taille.add(grand);
    petit=new JMenuItem("petit"); petit.addActionListener(this);taille.add(petit);
    setJMenuBar(menuBar); p0=new JPanel(); cont.add(p0);  }
    public void actionPerformed(ActionEvent ev){
    Object source=ev.getSource();
    if (source==rouge) p0.setBackground(Color.red);
    if (source==bleu ) p0.setBackground(Color.blue);
    if (source==grand ) {setSize(500,400);validate();}
    if (source==petit ) {setSize(300,200);validate();}    }
    public static void main(String args[]){
    Fenetre f=new Fenetre();
    f.setVisible(true); }
}
```



Interface Graphique: JFileChooser

Un `JFileChooser` permet de sélectionner un fichier en parcourant l'arborescence du système de fichier.

Exemple :



```

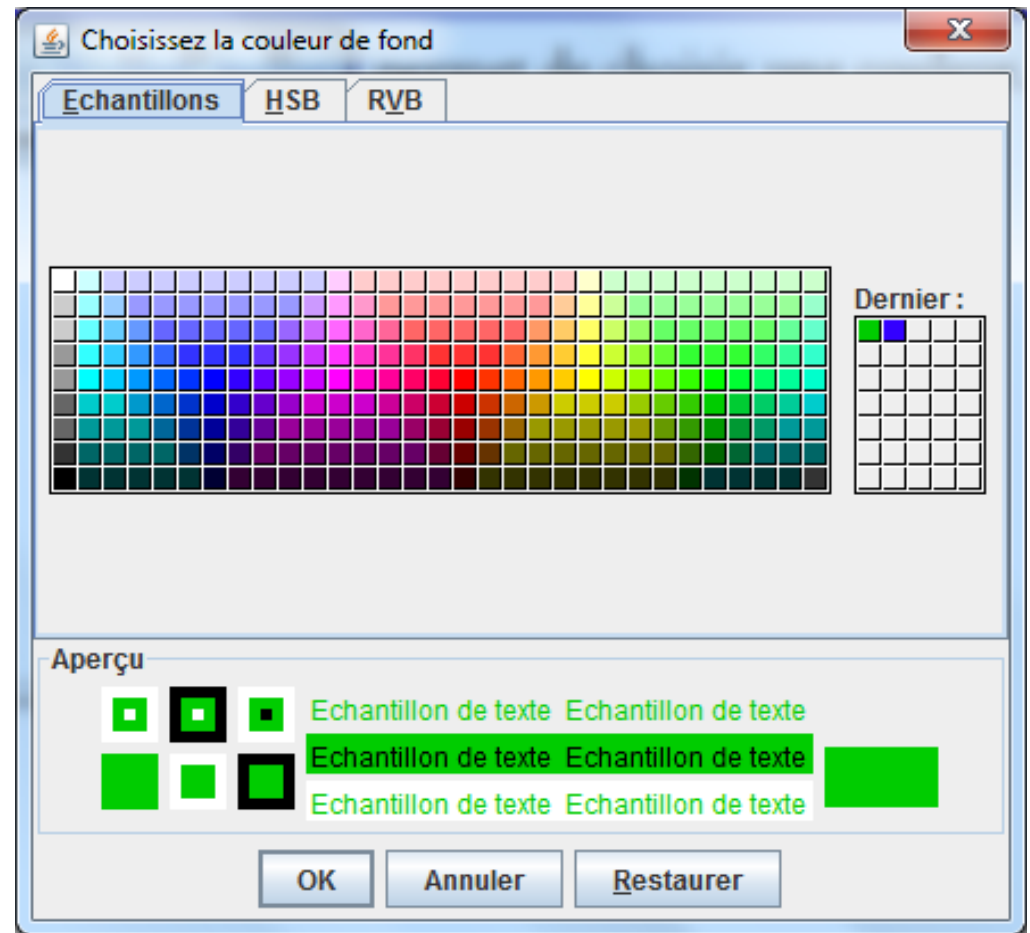
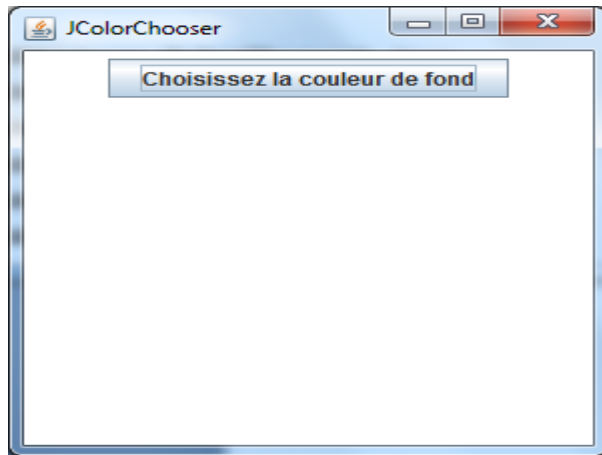
import javax.swing.* ; import java.awt.event.*; import java.awt.*;
public class DemoJFileChooser extends JPanel
    implements ActionListener {
    JButton B;      JFileChooser choix;      String titre;
    public DemoJFileChooser() {
        B = new JButton("Ouvrir");      B.addActionListener(this);      add(B);
    }
    public void actionPerformed(ActionEvent e) {
        choix = new JFileChooser();
        choix.setCurrentDirectory(new java.io.File("."));
        choix.setDialogTitle(titre);
        choix.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        choix.setAcceptAllFileFilterUsed(false);
        if (choix.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
            System.out.println("getCurrentDirectory(): " + choix.getCurrentDirectory());
            System.out.println("getSelectedFile() : " + choix.getSelectedFile());
        } else { System.out.println("pas de selection"); }
    }
    public Dimension getPreferredSize(){ return new Dimension(300, 300); }
    public static void main(String s[]) {
        JFrame frame = new JFrame("JFileChooser");
        DemoJFileChooser panel = new DemoJFileChooser();
        frame.addWindowListener( new WindowAdapter() {
            public void windowClosing(WindowEvent e) { System.exit(0); } } );
        frame.getContentPane().add(panel,"Center");
        frame.setSize(panel.getPreferredSize()); frame.setVisible(true);
    }
}

```


Interface Graphique: JColorChooser

Un JColorChooser permet de choisir une couleur.

Exemple :



```
import java.awt.*; import java.awt.event.*;
import javax.swing.*; import javax.swing.colorchooser.*;
import javax.swing.event.*;
public class ColorChooser1 extends JFrame implements ActionListener {
    public static void main(String[] args) { new ColorChooser1(); }
    public ColorChooser1() { super("JColorChooser");
        Container content = getContentPane();
        content.setBackground(Color.white);
        content.setLayout(new FlowLayout());
        JButton colorButton = new JButton("Choisissez la couleur de fond");
        colorButton.addActionListener(this); content.add(colorButton);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 300); setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        Color bgColor = JColorChooser.showDialog(this,
            "Choisissez la couleur de fond", getBackground());
        if (bgColor != null) getContentPane().setBackground(bgColor);
    }
}
```

Interface Graphique: gestionnaire de disposition

- **Layout** est une technique qui permet de placer des composants dans un container.
- **Layout** place les composants les uns par rapport aux autres.
- **Layout** organise les composants lorsque la taille du container varie.
- **Layout** gère le positionnement.

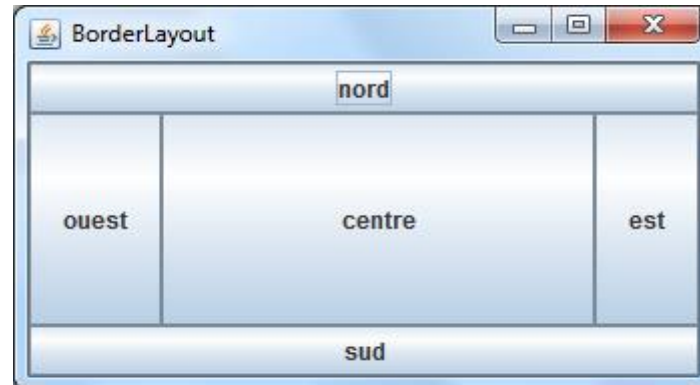
Gestionnaires les plus courants :

BorderLayout, BoxLayout, CardLayout, FlowLayout,
GridLayout, GridBagLayout.

Interface Graphique: BorderLayout

- Le **BorderLayout** sépare un container en cinq zones: nord, sud, ouest, est, et centre.
- "nord" et "sud" occupent toute la largeur,
- "ouest" et "est" occupent la hauteur qui reste,
- "centre" occupe la place restante.
- Lorsque l'on agrandit le container, le centre s'agrandit. Les autres zones prennent uniquement l'espace qui leur est nécessaire.

Exemple:



```

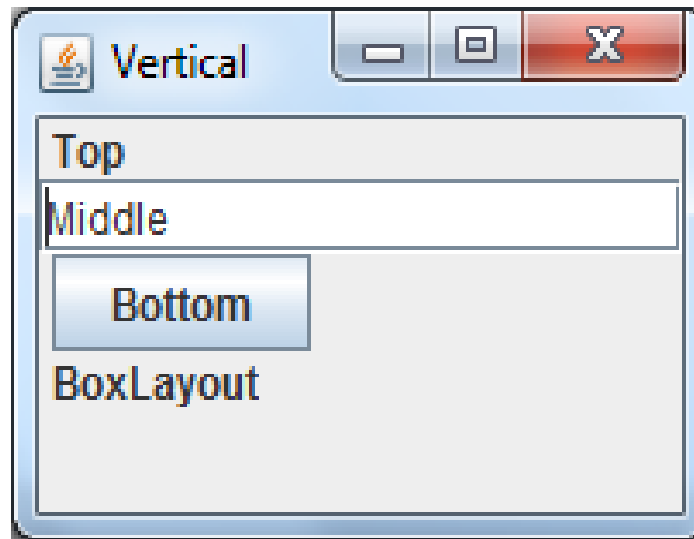
import java.awt.*;    import javax.swing.*;
class Exemple_BorderLayout extends JFrame {
    Exemple_BorderLayout() {
        JPanel content = new JPanel();
        content.setLayout(new BorderLayout());
        content.add(new JButton("nord")    , BorderLayout.NORTH);
        content.add(new JButton("est")     , BorderLayout.EAST);
        content.add(new JButton("sud")     , BorderLayout.SOUTH);
        content.add(new JButton("ouest")   , BorderLayout.WEST);
        content.add(new JButton("centre"), BorderLayout.CENTER);
        setContentPane(content);          setTitle("BorderLayout");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
    }
    public static void main(String[] args) {
        JFrame F = new Exemple_BorderLayout();
        F.setSize(350, 200);
        F.setVisible(true);
    }
}

```

Interface Graphique: BorderLayout

- Un **BoxLayout** permet d'empiler les composants du container (verticalement ou horizontalement)
- **BoxLayout** donne à chaque composant la place qu'il demande
- En utilisant ce layout on peut rajouter des blocs invisible.

Exemple:



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

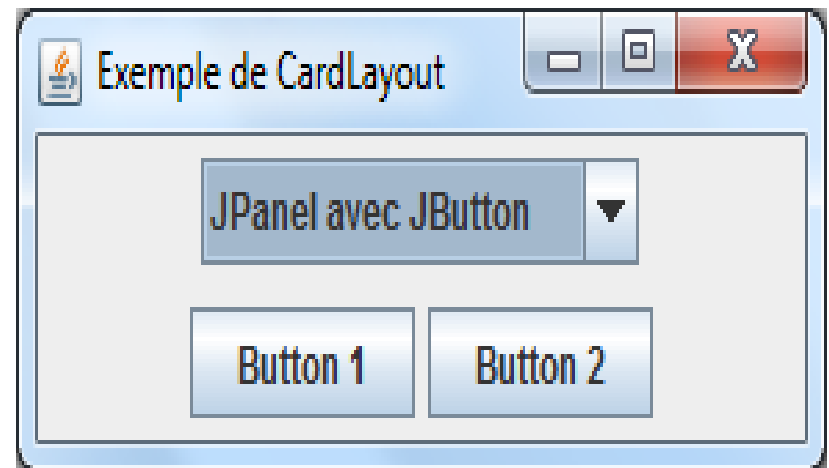
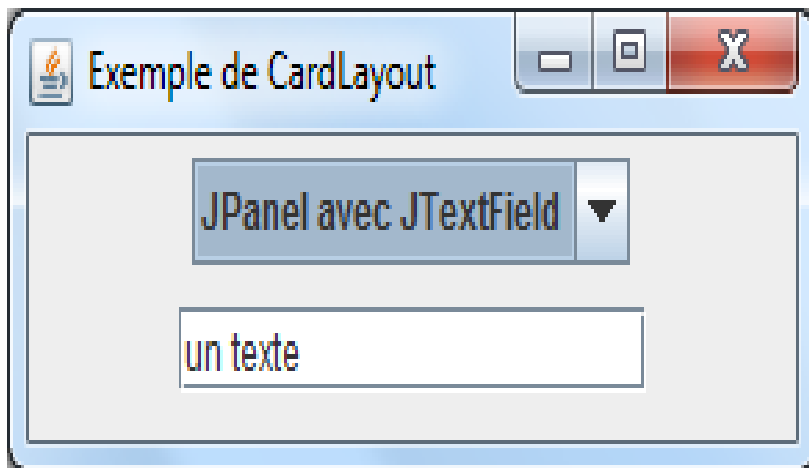
public class Exemple_BoxLayout {
    public static void main(String args[]) {
        JFrame verticalFrame = new JFrame("Vertical");
        verticalFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Box verticalBox = Box.createVerticalBox();
        verticalBox.add(new JLabel("Top"));
        verticalBox.add(new JTextField("Middle"));
        verticalBox.add(new JButton("Bottom"));
        verticalBox.add(new JLabel("BoxLayout"));
        verticalFrame.getContentPane().add(verticalBox, BorderLayout.NORTH);
        verticalFrame.setSize(250, 150);
        verticalFrame.setVisible(true);
    }
}
```

Interface Graphique: CardLayout

→ **CardLayout** aide à construire des boîtes de dialogue composées de plusieurs onglets.

→ Un onglet se compose généralement de plusieurs contrôles : on insère des panneaux dans la fenêtre utilisée par le **CardLayout**.

Exemple:




```

import java.awt.*; import java.awt.event.*; import javax.swing.*;
public class Exemple_CardLayout extends JFrame implements ItemListener{
    private JPanel cardsPanel;
    public Exemple_CardLayout() {
        super("Exemple de CardLayout");
        Container contentPane = this.getContentPane();
        contentPane.setLayout(new BorderLayout());
        JPanel comboBoxPanel = new JPanel();
        String comboBoxItems[] = {"JPanel avec JButton", "JPanel avec JTextField"};
        JComboBox comboBox = new JComboBox(comboBoxItems);
        comboBox.setEditable(false);    comboBox.addItemListener(this);
        comboBoxPanel.add(comboBox); contentPane.add(comboBoxPanel, BorderLayout.NORTH);
        cardsPanel = new JPanel();    cardsPanel.setLayout(new CardLayout());
        JPanel p1 = new JPanel();    JPanel p2 = new JPanel();
        p1.add(new JButton("Button 1"));    p1.add(new JButton("Button 2"));
        p2.add(new JTextField("un texte", 15));
        cardsPanel.add(p1, "JPanel avec JButton");
        cardsPanel.add(p2, "JPanel avec JTextField");
        contentPane.add(cardsPanel, BorderLayout.CENTER);
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) { System.exit(0); } });
    public void itemStateChanged(ItemEvent evt) {
        CardLayout cl = (CardLayout)(cardsPanel.getLayout());
        cl.show(cardsPanel, (String)evt.getItem());
    }
    public static void main(String[] args) {
        Exemple_CardLayout F = new Exemple_CardLayout();
        F.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        F.pack();    F.setVisible(true);    }
}

```

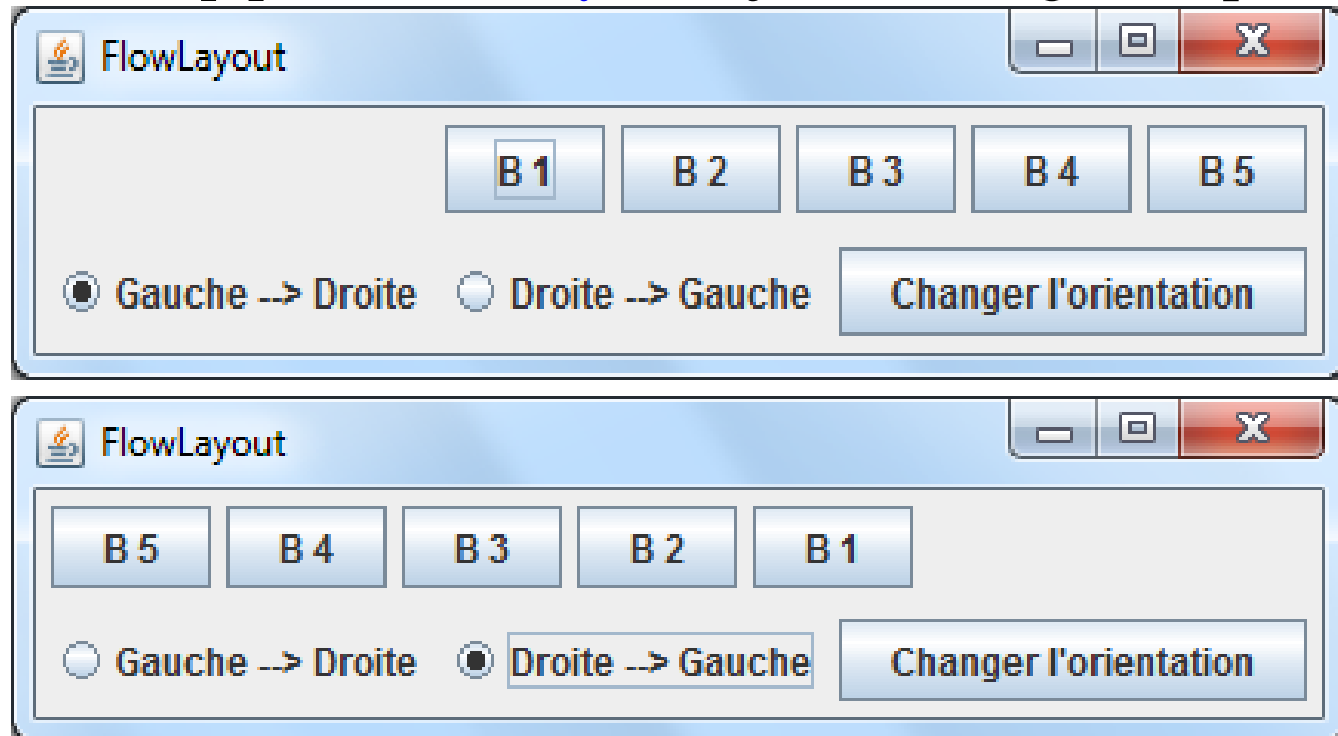
Interface Graphique: FlowLayout

→ Par défaut le gestionnaire de présentation d'un panel est de type **FlowLayout**.

→ Un **FlowLayout** permet de ranger les composants dans une ligne.

→ Si l'espace est trop petit **FlowLayout** ajoute une ligne de plus.

Exemple:



```

import java.awt.*; import java.awt.event.*;
import javax.swing.*; import java.util.*;
public class Exemple_FlowLayout extends JFrame{
    JRadioButton RtoLbutton;    JRadioButton LtoRbutton;
    FlowLayout experimentLayout = new FlowLayout();
    JButton applyButton = new JButton("Changer l'orientation");
    public Exemple_FlowLayout(String nom) { super(nom); }
    public void addComponentsToPane(final Container pane) {
        final JPanel aa = new JPanel();
        aa.setLayout(experimentLayout);
        experimentLayout.setAlignment(FlowLayout.TRAILING);
        JPanel controls = new JPanel();
        controls.setLayout(new FlowLayout());
        LtoRbutton = new JRadioButton("Gauche --> Droite");
        LtoRbutton.setActionCommand("Gauche --> Droite");
        LtoRbutton.setSelected(true);
        RtoLbutton = new JRadioButton("Droite --> Gauche");
        RtoLbutton.setActionCommand("Droite --> Gauche");
        aa.add(new JButton("B 1")); aa.add(new JButton("B 2"));
        aa.add(new JButton("B 3")); aa.add(new JButton("B 4"));
        aa.add(new JButton("B 5"));
        aa.setComponentOrientation(ComponentOrientation.LEFT_TO_RIGHT);
        final ButtonGroup group = new ButtonGroup();
        group.add(LtoRbutton);          group.add(RtoLbutton);
        controls.add(LtoRbutton);        controls.add(RtoLbutton);
        controls.add(applyButton);
    }
}

```

```

applyButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String command = group.getSelection().getActionCommand();
        if (command.equals("Left to right")) {
            aa.setComponentOrientation(
                ComponentOrientation.LEFT_TO_RIGHT);
        } else { aa.setComponentOrientation(
            ComponentOrientation.RIGHT_TO_LEFT); }
        aa.validate();aa.repaint();} });
pane.add(aa, BorderLayout.CENTER);
pane.add(controls, BorderLayout.SOUTH); ;    }

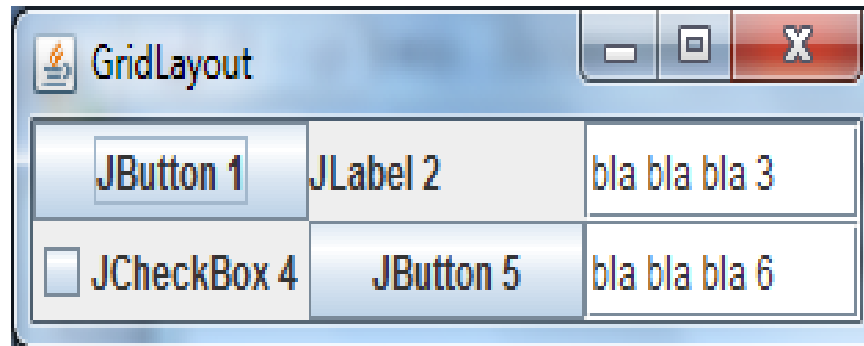
public static void main(String[] args) {
    Exemple_FlowLayout frame = new Exemple_FlowLayout("FlowLayout");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.addComponentsToPane(frame.getContentPane());
    frame.pack();          frame.setVisible(true);
}
}

```

Interface Graphique: GridLayout

- **GridLayout** établit un réseau de cellules identiques qui forment une sorte de quadrillage invisible : les composants sont organisés en lignes et en colonnes.
- Les éléments insérés dans la grille ont tous la même taille.
- Les cellules du quadrillage se remplissent de droite à gauche ou de haut en bas.

Exemple:



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

public class Exemple_GridLayout {

    public static void ajout(Container cc) {
        cc.setLayout(new GridLayout(2,2));
        cc.add(new JButton("JButton 1"));
        cc.add(new JLabel("JLabel 2"));
        cc.add(new JTextField("bla bla bla 3"));
        cc.add(new JCheckBox("JCheckBox 4"));
        cc.add(new JButton("JButton 5"));
        cc.add(new JTextField("bla bla bla 6"));
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("GridLayout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ajout(frame.getContentPane());  frame.pack();
        frame.setVisible(true);    }

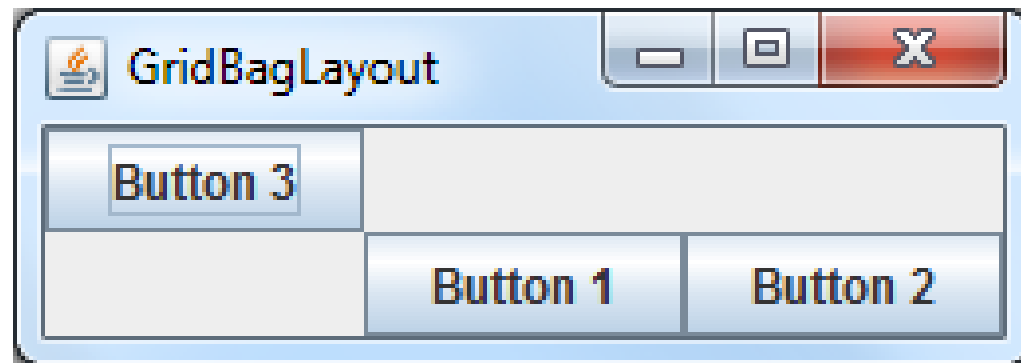
}
```

Interface Graphique: GridBagLayout

→ **GridBagLayout** (grille étendue) est le plus riche en fonctionnalités : le conteneur est divisé en cellules égales mais un composant peut occuper plusieurs cellules de la grille et il est possible de faire une distribution dans des cellules distinctes.

→ Un objet de la classe **GridBagConstraints** permet de donner les indications de positionnement et de dimension à l'objet **GridBagLayout**.

Exemple:



```

import java.awt.*; import javax.swing.*;
public class Exemple_GridBagWindow extends JFrame {
    public Exemple_GridBagWindow() {
        JButton b;      Container pp = getContentPane();
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        pp.setLayout(gridbag);    b = new JButton("Button 1");
        c.gridx = 10;    c.gridy = 10;
        gridbag.setConstraints(b, c);    pp.add(b);
        b = new JButton("2");    c.gridx = 10;    c.gridy = 10;
        gridbag.setConstraints(b, c);    pp.add(b);
        b = new JButton("Button 2"); c.gridx = 20; c.gridy = 10;
        gridbag.setConstraints(b, c);    pp.add(b);
        b = new JButton("Button 3");
        c.gridx = 1;    c.gridwidth = 2;    c.gridy = 2;
        gridbag.setConstraints(b, c);    pp.add(b);
    }
    public static void main(String args[]) {
        Exemple_GridBagWindow F = new Exemple_GridBagWindow();
        F.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        F.setTitle("GridBagLayout");F.pack();F.setVisible(true);
    }
}

```


Interface Graphique: Gestion des évènements

→ Il est possible d'affecter un évènement à un objet grâce à des classes spéciales appelées **listeners**. Ces classes nous permettent d'écouter les actions effectuées sur un objet swing et de lancer la fonction adéquat à l'action produite.

→ Les deux classes : **java.awt.event.MouseMotionAdapter** et **java.awt.event.ActionListener** nous permettent l'écoute des actions.

→ Lorsqu'un événement se produit :

→ il est reçu par le composant avec lequel l'utilisateur interagit (un bouton, un curseur, un champ de texte, ...).

→ Ce composant transmet cet événement à un autre objet, un écouteur qui possède une méthode pour traiter l'événement.

Interface Graphique: Gestion des événements

→ Les événements utilisateurs sont gérés par plusieurs interfaces `EventListener`.

→ Les interfaces `EventListener` permettent à un composant de générer des événements utilisateurs. Une classe doit contenir une interface auditeur pour chaque type de composant :

`ActionListener` : clic de souris ou enfoncement de la touche Enter

`ItemListener` : utilisation d'une liste ou d'une case à cocher

`MouseMotionListener` : événement de souris, ...

`WindowListener` : événement de fenêtre

`TextListener` : Changement de valeur dans une zone de texte

Interface Graphique: Gestion des événements

AdjustmentListener : Déplacement d'une échelle

ComponentListener : Savoir si un composant a été caché, affiché ...

ContainerListener : Ajout d'un composant dans un Container

FocusListener : Pour savoir si un élément a le "focus"

KeyListener : Pour la gestion des événements clavier

Interface Graphique: Gestion des évènements

Interface ActionListener

`void actionPerformed(ActionEvent e)`

Interface MouseListener

`void mousePressed(MouseEvent e)`

`void mouseReleased(MouseEvent e)`

`void mouseClicked(MouseEvent e)`

`void mouseEntered(MouseEvent e)`

`void mouseExited(MouseEvent e)`

`void actionPerformed(ActionEvent e)`

Interface Graphique: Gestion des évènements

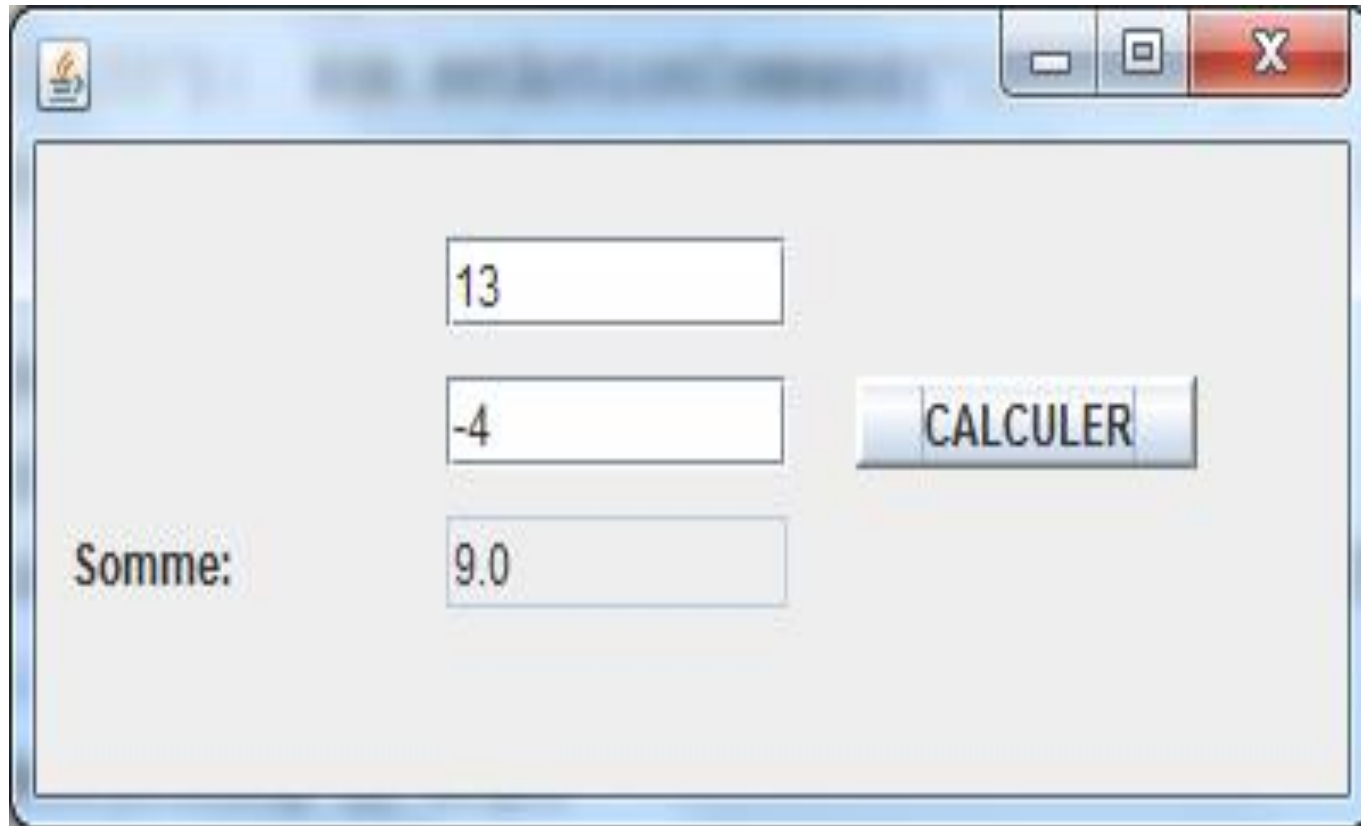
ActionListener : Exemple 1

```
import java.awt.*; import java.awt.event.*;
import javax.swing.*;
public class Ttest extends JFrame{
    public Ttest() {
        JButton monBouton = new JButton("Fermer");
        monBouton.addActionListener(new Classe_action());
        add(monBouton); }
    public static void main(String args[]) {
        JFrame f = new Ttest();
        f.pack(); f.setSize(300,100);
        f.setVisible(true);    }
}
class Classe_action implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);    }
}
```



Interface Graphique: Gestion des évènements

ActionListener : Exemple 2



```

import javax.swing.*; import java.awt.*;
import java.awt.event.*; import javax.swing.border.*;
public class Somme extends JFrame implements ActionListener
{
    private JLabel lab;           private JTextField A,B,S;
    private JButton som;          double x,y,s;
    Somme() {setBounds(100,40,400,300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container c;    c=getContentPane();
        BorderLayout bl=new BorderLayout(); c.setLayout(bl);
        JPanel p=new JPanel(); p.setLayout(null); lab= new JLabel("Somme:");
        lab.setBounds(10,80,100,20);    p.add(lab);
        A=new JTextField(15);  A.setBounds(120,20,100,20);  p.add(A);
        B=new JTextField(15);  B.setBounds(120,50,100,20);  p.add(B);
        S=new JTextField(15);  S.setBounds(120,80,100,20);  S.setEditable(false);
        p.add(S); Border b=BorderFactory.createRaisedBevelBorder();
        som=new JButton("CALCULER"); som.setActionCommand("CALCULER");
        som.addActionListener(this); som.setBorder(b); som.setBounds(240,50,100,20);
        p.add(som); c.add(p, BorderLayout.CENTER); setVisible(true);    }
    public void actionPerformed(ActionEvent e)  {
        if(e.getActionCommand().equals("CALCULER")) { double a,b;
            a=Double.parseDouble(A.getText()); b=Double.parseDouble(B.getText());
            S.setText(String.valueOf((a+b)));    }    }
    public static void main(String [] args)
    {
        Somme a=new Somme();    }
}

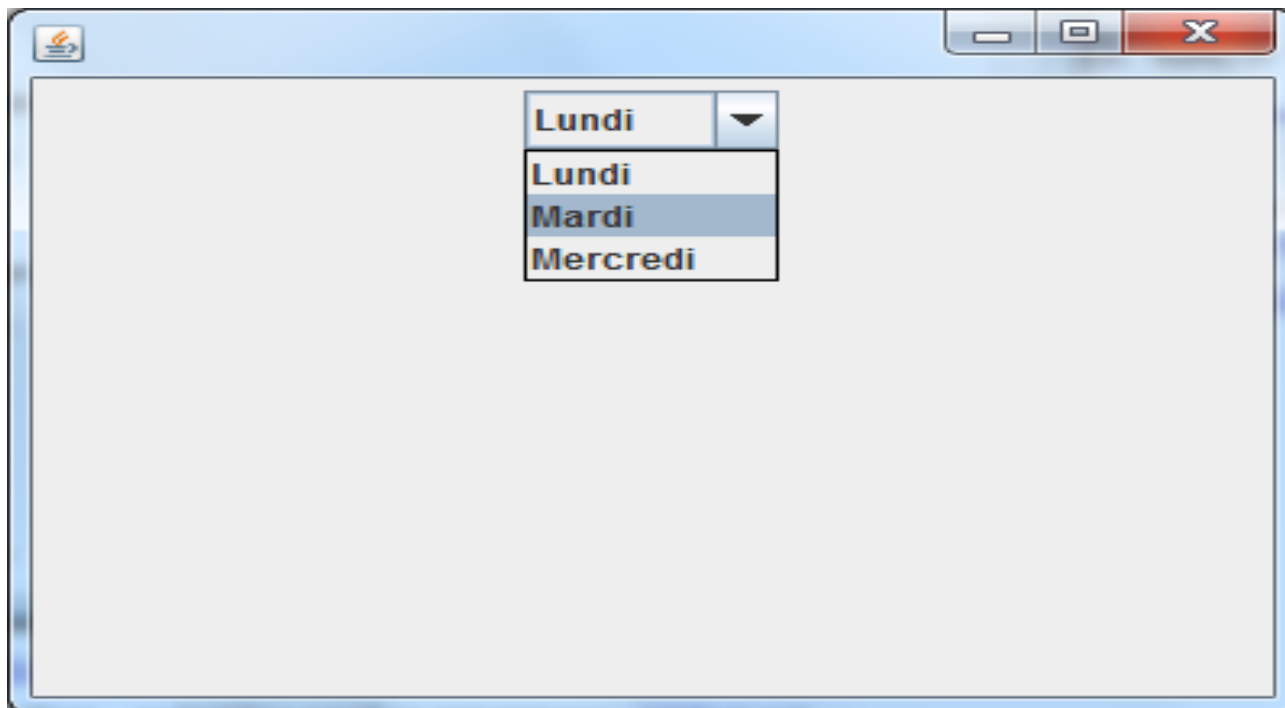
```

Interface Graphique: Gestion des évènements

ItemListener : Exemple

```
import java.awt.*; import java.awt.event.*; import javax.swing.*;
public class Test_ItemListener extends JFrame implements ItemListener{
    private JComboBox combo;
    public Test_ItemListener() {
        combo=new JComboBox();
        combo.addItem("Lundi");
        combo.addItem("Mardi");
        combo.addItem("Mercredi");
        this.getContentPane().add(combo);
        combo.addItemListener(this);
        this.getContentPane().setLayout(new FlowLayout());
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true); this.setBounds(100,40,400,300);    }

    public void itemStateChanged(ItemEvent e) {
        if(e.getSource()==combo) System.out.println("Changement "); }
    public static void main(String args[]) {
        Test_ItemListener a=new Test_ItemListener();    }
}
```

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The command prompt shows the following sequence of commands and output:

```
C:\Users\lamnii\GUI>javac Test_ItemListener.java
C:\Users\lamnii\GUI>java Test_ItemListener
Changement
Changement
C:\Users\lamnii\GUI>
```

The output shows two lines of 'Changement', indicating that the program was executed twice. The command prompt has a scrollbar on the right side.