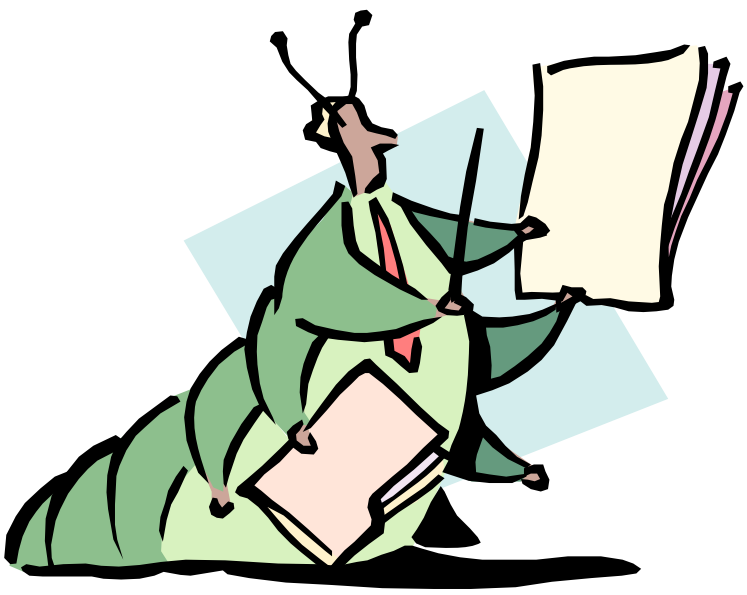
Pour commencer avec Java



Sommaire

Sommaire

Sommaire

Pour commencer avec Java

1. Rapide historique

OAK fut son nom avant qu'il ne s'appelle JAVA (mot argot pour "café) ceci apparemment à cause de la grande quantité de ce breuvage ingéré par les programmeurs.

C'est chez SUN MICROSYSTEM qu'il naquit pour pallier le manque d'adaptabilité du C++ dans le développement d'application de type "domotique".

A la même période, Internet fait son entrée avec la langage HTML. Très vite, le contenu des pages WEB demandera de l'interactivité (formulaires de saisie) et JAVA et ses caractéristiques devient le langage du WEB.

Du fait du peu d'encombrement des programmes écrits en JAVA, le temps de téléchargement par le WEB est réduit. Mais surtout, Java est présenté comme le langage universel de demain pouvant être utilisé sur n'importe quelle plate-forme technique (Linux, MacIntosh, Windows, IBM, …). Qu'y a-t-il derrière ce miracle : la JVM ➔ Java Virtual Machine. Chaque environnement technique a sa JVM : c'est elle qui rend le dialogue possible entre un programme écrit en JAVA et le reste de l'environnement technique.

Malheureusement, l'universalité de JAVA risque de rester une belle idée. Deux grands acteurs ont un rôle prépondérant dans la mise en place de JAVA :

- JAVASOFT filiale de SUN MICROSYSTEM
- MICROSOFT

Comme à son habitude, la firme de Bill GATES essaie d'imposer ses propres standards mais SUN et ses partenaires dont IBM ne laissent pas la dérive s'installer

Nous parlerons donc assez fréquemment du JDK (la version SUN) et de visual J++ (la version Microsoft) chacun ayant son propre environnement de développement (outils du programmeur allant autour de JAVA). Les bases du langage restent les mêmes et tout au long de ce support, nous tenterons de rester dans les éléments communs aux deux plates-formes.

Pour commencer avec Java

2. Les bases du langage

2.1. La syntaxe

Quel que soit le langage, y compris le langage humain, il se doit de respecter un certain nombre de règles.

Pour construire une phrase, il nous faut un verbe qui indique le type d'action à mener; puis un sujet et des compléments qui permettent de préciser l'environnement de l'action. En programmation, nous aurons des ***instructions*** qui indiqueront à la machine ce qu'elle doit faire

Une instruction JAVA se termine par un ;

Exemple :

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

Une instruction java peut être écrite sur plusieurs lignes de source

Il est impératif de bien commenter un programme quel que soit le langage utilisé. Ces commentaires ne sont pas pris en compte lors de l'exécution du programme, ils sont là pour que chacun se retrouve vite dans un programme. N'oubliez pas, vous mettrez quinze jours à réaliser un programme qui sera maintenu pendant des années.
Faites des commentaires brefs et bien aérés et dès que vous utilisez des trucs et astuces, faites l'effort de les expliquer en clair.

Un ***commentaire*** commence par // et se termine par le saut à la ligne suivante

```
//   Ceci est un commentaire occupant une ligne
```

```
Instruction JAVA        //   Ceci est un commentaire occupant une ligne à la suite d'une instruction
```

S'il occupe plusieurs lignes, il faut l'encadrer par /* */

```
/*   Ceci est un commentaire occupant plusieurs ligne :   ligne n° 1
                                                          ligne n° 2
                                                          ligne n° 3                    */
```

Pour commencer avec Java

Pour toutes les entités que vous allez utiliser, vous lui donnerez un nom aussi appelé ***identificateur***.

Cet identificateur doit respecter quelques règles de "nommage" :

Le premier caractère doit être une lettre,

La suite doit être composée de lettres, de chiffres

Ne pas y mettre d'espace, de tabulation, de retour chariot, de passage à la ligne

Ne pas être un mot réservé JAVA

Donnez un nom significatif qui permettra une lecture plus facile du code; la longueur autorisée pour un nom vous laisse de quoi vous exprimer.

Si vous utilisez les lettres minuscules et les lettres majuscules, le nom est considéré comme différent (langage "casse sensitive"):

Exemple : le nom Compteur et le nom compteur désigne deux entités différentes.

Selon ce que vous nommerez, il sera nécessaire de mettre la première lettre :

- En minuscule pour les variables et les méthodes
- En majuscule pour les classes ou les interfaces

Ces différents termes seront revus, en détail, plus loin.

Pour commencer avec Java

2.2. La définition des variables et des constantes.

Quand nous démarrons en programmation, il est important de comprendre que l'exécution d'un programme se fait dans la mémoire de l'ordinateur, que nous pourrions comparer avec notre cerveau. Il y a d'un coté les instructions à faire c'est à dire les actions à mener mais il faut également des éléments sur qui porteront les actions. Ces différents éléments sont stockés, mémorisés dans ce que j'appellerai pour l'instant des cases mémoires. Toujours pour faire le parallèle avec le cerveau humain, quand quelqu'un "déraille" on dit souvent qu'il a qu'une case de vide. Notre brave machine est pareil : il lui faut des cases qui auront un nom (identificateur) et un contenu.

Le nom correspond à **l'adresse** de la case mémoire : c'est l'endroit physique où le programme peut aller chercher l'information. Comme il serait compliqué de travailler avec une liste d'adresse et surtout très peu lisible, un nom est donné et la machine fait le lien entre le nom indiqué que vous comprenez et l'adresse que seule la machine connaît.

Il y a une différence selon la nature du contenu : parfois on souhaitera y stocker des caractères (pour mémoriser par exemple un prénom) d'autres fois on souhaitera y stocker des chiffres, des nombres (pour mémoriser un salaire).

Lors de l'exécution d'un programme, la machine a besoin de connaître la nature du contenu de la case qu'elle manipule : nous parlerons de ***type de données***.

Nous utiliserons les types de données suivants :

- ***Entiers***
- ***Réels***
- ***Booléens***
- ***Caractères***
- ***Chaînes de caractères***

Avec certains types de données, la taille de l'emplacement mémoire est connu d'office (cas des entiers); par contre, dans d'autres cas, il nous faudra indiquer la taille de l'emplacement à utiliser (cas des chaînes de caractères)

En résumé, une variable est définie par :

- ❒ Un nom ou identificateur
- ❒ Une nature de données (entière, chaîne de caractères, ...)
- ❒ Une taille ou longueur

Il existe des mots réservés pour le langage :
ces mots ne doivent pas être utilisés comme nom pour des variables, constantes ou objets

Pour commencer avec Java

2.2.1. Les entiers

Selon la valeur maximale que devra contenir la variable, nous la définirons en :

- ***byte*** (8 bits) pour des valeurs comprises entre -2^7 (-128) et 2^7 (127)
- ***short*** (16 bits) pour des valeurs comprises entre -2^{15}(-32768) et 2^{15}(+32767)
- ***int*** (32 bits) pour des valeurs comprises entre -2^{31} et 2^{31}
- ***long*** (64 bits) pour des valeurs comprises entre -2^{63} et 2^{63}

Par défaut, Java stocke les littéraux en int.

Exemples:

```
int   n1;                    //  réserve un emplacement pour une valeur entière nommée n1
```

```
short  s2  = 10;             //  réserve un emplacement pour une valeur entière courte nommée s2
```

Dans l'exemple ci dessus, en plus de la réservation de l'emplacement mémoire, nous lui affectons une valeur initiale qu'il sera possible de faire évoluer dans la suite du programme.

2.2.2. Les réels

Un réel est un nombre à virgule avec ou sans exposant.
Selon la valeur maximale que devra contenir la variable, nous la définirons en :

- ***float*** (32 bits)
- ***double*** (64 bits)

Par défaut , Java stocke les réels en double.

Exemple:

```
double nTva = 18.6;   //  réserve un emplacement de nom nTva pour une valeur réelle double
```

Pour commencer avec Java

2.2.3. Les booléens

Une valeur ***booléen***ne ne peut prendre que deux valeurs :

- ***vraie : mot clé true***
- ***fausse mot clé false***

car le courant passe ou ne passe pas

Une valeur booléenne ne peut pas être convertie dans un autre type.

2.2.4. Les caractères

Ils correspondent a une donnée contenant un seul caractère.
Ils sont stockés sur 16 bits.

Exemple :

```
char  Réponse = 'O';  //   réserve un emplacement d'un caractère et se nommant Réponse
```

Dans l'exemple ci-dessus, nous avons affecté la valeur O à la variable de type ***char*** nommée Réponse.

Notez que la valeur affectée est mise entre guillemets quand l'affectation concerne un type caractère ou chaîne de caractères. Ce n'est pas le cas pour l'affectation de variables de type numérique (entier ou réel)

Vous pouvez mettre un caractère qui représente un chiffre, vous n'aurez aucune alerte. Cependant, vous ne pourrez pas utiliser une telle variable pour faire des calculs.

Pour commencer avec Java

2.2.5. Les chaînes de caractères

Elles correspondent à une donnée contenant une suite de caractères.
Bien que nous évoquions les chaînes de caractères dans ce paragraphe, sachez que vous ne manipulez pas ici un type de données mais un objet d'une classe ***String.***

Exemple :

```
String str1 = "Ceci est une chaîne de caractères";
```

Mais qu'est ce qu'une classe me direz-vous ? Réponse dans quelques chapitres.

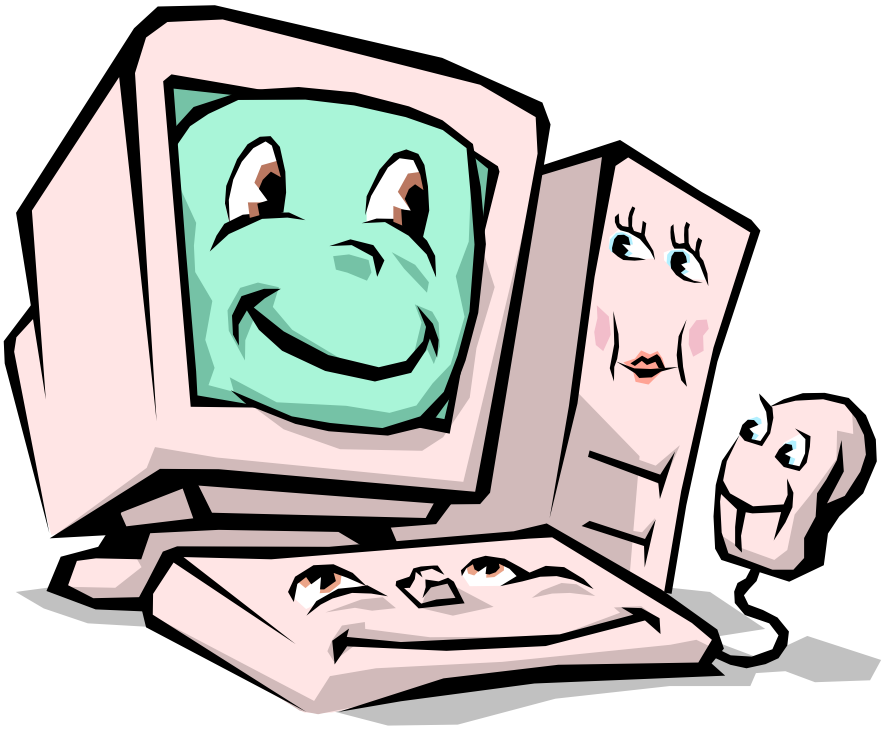
Pour commencer avec Java

Conventions de noms:

Pour une meilleure lisibilité, nous utiliserons des conventions que nous détaillerons tout au long de ce document et que nous vous demandons de respecter.

Les identificateurs de variables sont écrits en minuscule en commencent par :

b pour les booléens	(bOk)
n pour les entiers int	(nAge)
l pour les entiers longs	(lIndice)
d pour les réels	(dSalaire)
str pour les chaînes de caractères	(strNom)
i pour les indices	(iCpt)
frame pour les fenêtres	frameGesPers
panel pour les panneaux	panelSaisie
pane pour les pane	
lbl pour les labels ou intitulés	lblAdresse
cmd pour les boutons	cmdOk
txt pour les textarea	txtNom
fld pour les textfield	fldObservation
pop pour les popmenu	popPersonne



Pour commencer avec Java

La définition des tableaux.

Un ***tableau*** est une variable qui contient une liste d'éléments de même type de données ou de même classe.

Pour pouvoir utiliser un tableau, il faut passer par :

- La déclaration du tableau,
- La création du tableau car un tableau est un objet Java,
- L'accès aux éléments du tableau.

Comment déclarer un tableau ?

TypeDonnée nomElément [ ]

```
char lettre[ ];   // déclare un tableau nommé lettre dont chaque élément est de type caractère
```

Comment le créer ?

nomElément = new TypeDonnée[nbéléments];

```
lettre = new char[20];          // crée le tableau de 20 éléments de type caractère
```

Comment atteindre un élément précis à l'intérieur du tableau ?

```
nomElément [3] = "W";
```

permet de mettre la lettre W dans le quatrième poste du tableau car le premier poste a l'indice 0.

Pour commencer avec Java

2.3. Les opérateurs.

2.3.1. Les opérateurs binaires arithmétiques

Ce sont les classiques opérateurs qui permettent de faire des calculs et de stocker le résultat obtenu dans une variable.

Opérateur	Définition	Exemples
=	Affectation	nNum = 25
+	Addition	nTtc = nHt + nTva
-	Soustraction	nInd = nInd - 5
*	Multiplication	nInd = nInd -* 10
/	Division entière	nInd = nInd / 2
%	Reste de la division entière	nInd = nInd -* 10
+	Addition	nTtc = nHt + nTva
-	Soustraction	nInd = nInd - 5
*	Multiplication	nInd = nInd -* 10
/	Division	nInd = nInd / 2

L'***affectation*** permet d'envoyer la valeur se trouvant à droite du = dans le contenu de la zone dont le nom se trouve à gauche du =.

```
nTauxTva = 18.6;          //  envoi la valeur 18,60 dans la variable nTauxTva
```

Il est possible d'affecter une valeur directement indiquée derrière le = mais aussi d'affecter une valeur qui est contenue dans une autre variable de type compatible.

vous ne pourrez pas affecter n'importe quelle valeur dans n'importe quelle variable

une variable de type double ne peut être envoyée dans variable de type chaîne de caractère

```
nValAvant  = nValEnCours;   // envoie le contenu de nValEnCours dans le contenu de
                               nValAvant
```

L'addition, la soustraction, … permettent de faire le calcul demandé à partir des variables et/ou constantes numériques indiquées et de mettre le résultat dans la variable devant le signe d'affectation.

Pour commencer avec Java

```
nTtc = (nHorsTaxe  *  nTaux/Tva) / 100;   //   permet de faire * avant /
```

il y a un ordre de prise en compte des opérateurs : pour ne pas trop vous poser de questions, mettez des parenthèses ( ) pour définir l'ordre de calcul.

Pour commencer avec Java

2.3.2. Les conversions : transtypage

Lorsque vous avez appris à faire des additions à l'école primaire, vous avez dû entendre la phrase 'on n'ajoute pas des carottes et des navets'.

En java, c'est toujours vrai. Lorsque l'opération que vous voulez exécuter attend un entier vous ne pouvez pas lui donner un double; il faut donc convertir votre double en entier. Pour cela, indiquez le type souhaité entre parenthèses devant le nom de la variable de type double

```
Double dSalaire = 12345.70;
int nSalaire2 = (int)dSalaire;    //    attention ici on a perdu les chiffres après la virgule
```

Les conversions sont, si possible, à faire dans le sens type de données de longueur plus petite vers un type de données de longueur plus grande Lorsque c'est dans l'autre sens, attention aux troncatures opérées par la conversion.

Les conversions ne sont malheureusement pas toujours aussi simples.
Nous reprendrons cet aspect lors de la création de la classe Outils page 102

Pour commencer avec Java

2.3.3. Les opérateurs binaires de comparaison

Ils permettent d'effectuer des comparaisons logiques sur des opérandes de même type.

Opérateur	Définition
= =	Égal
!=	Différent
<	Inférieur
>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal

ne pas confondre = qui affecte et = = qui teste l'égalité

2.3.4. Quelques opérateurs binaires d'affectation

Opérateur	Définition
=	Affectation
+=	Addition et affectation
-=	Soustraction et affectation
*=	Multiplication et affectation
<<=	Décalage à gauche et affectation
>>=	Décalage à droite et affectation

2.3.5. Les opérateurs logiques

Opérateur	Définition
!	Non
&	Et
|	Ou
^	Ou exclusif
&&	Et logique
||	Ou logique

Différence entre & et && :

& évalue les expressions de part et d'autre du &
&& évalue l'expression de gauche : si elle est fausse, la partie de droite n'est pas évaluée et le résultat est faux

Pour commencer avec Java

2.3.6. Les opérateurs unaires

Les opérateurs unaires d'***incrémentation*** peuvent utiliser deux types de notations :

- La notation préfixée : i++ qui équivaut à évaluer la valeur de i puis à faire i + 1.
- La notation suffixée : ++i qui équivaut à faire i + 1 puis à évaluer la valeur de i.

Les opérateurs d'incrémentation sont :

Opérateur	Définition	Exemples
++	Post incrémentation	i++
	Pré-incrémentation	++i
--	Post décrémentation	i--
	Pré-décrémentation	--i
!	Opposé	

Pour commencer avec Java

2.4. La notion de classes

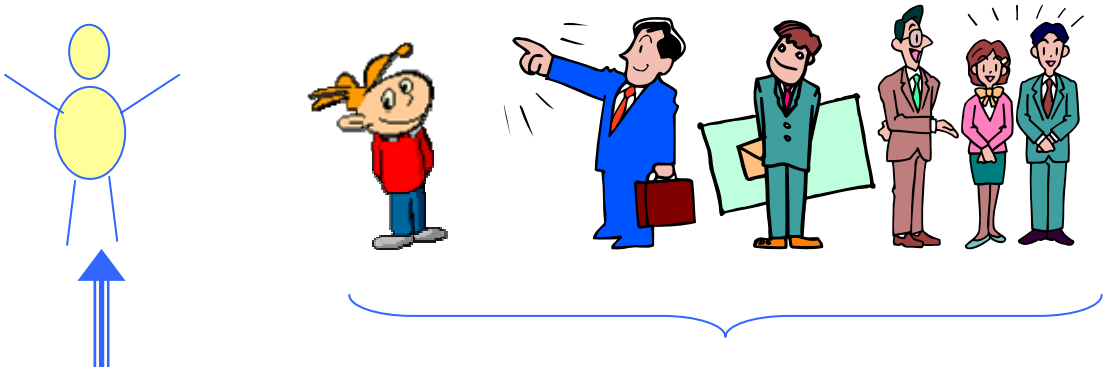
Une classe décrit une catégorie d'objets avec des propriétés et des méthodes (appelées aussi fonctions ou comportements) qui lui sont propres.

C'est un moule qui va permettre de façonner des individus appelés objet ou instance.

Prenons comme exemple une classe Salariés :

- nom de la classe : Salariés
- Attributs de cette classe :
  - Numéro matricule
  - Catégorie salariale
  - Service
  - Nom
  - Montant du salaire
- Méthodes ou comportements
  - Affichage du salaire

A partir de ce moule, nous pouvons créer des instances qui ont toutes les attributs et les comportements définis dans la classe. Ils différent par le contenu des attributs : l'instance n° 1 aura pour nom "Dupont", l'instance n° 2 aura pour nom "Durand"….



Classe

6 objets ou instances de la classe Salariés

Pour créer une instance, nous utiliserons new Salariés et nous donnerons à cette instance un nom qui lui est propre, dans notre exemple Dupont :

```
Salariés Dupont = new Salariés();
```

Un programme est aussi une classe avec :

- Un nom de classe
- Eventuellement des attributs
- Au moins une méthode (méthode main pour les applications)

Pour commencer avec Java

Le langage JAVA est un langage structuré car un programme écrit en JAVA est une suite de fonctions ou méthodes.

Nous trouvons plusieurs types de programmes écrits en Java :

- Les applications Java : elles fonctionnent en autonome avec un système d'exploitation et une "Java Virtual Machine".
- Les applets Java : elles fonctionnent par appel dans une page HTML; il faut donc en plus un navigateur (browser).
- Les servlets , application spéciale exécutée dans le cadre de la machine virtuelle du serveur Web.

Un programmeJava commence toujours par une ligne indiquant le nom de la classe et sa visibilité
La fonction **main** est le point d'entrée d'une application JAVA.Elle est donc obligatoire.

Une application java minimale nommée Class1 serait :

```
public class Class1
{
	public static void main (String[] args)
	{
		System.out.println("je suis une appli Java");
	}
}
```

Pour commencer avec Java

2.5. Quelques instructions pour écrire votre premier programme

2.5.1. Affichage d'un message à l'écran

```
System.out.println("ceci est un message");
```

Il est possible d'afficher une ligne contenant la concaténation (mise bout à bout) de variables et/ou constantes

```
System.out.println("ceci est un message pour "+prenom);
```

Petite difficulté : affichage des lettres accentuées
Vous devez utiliser le code UNICODE de la lettre concernée

Lettre	Code à utiliser
é	\u201a
è	\u0160
à	\u2026
â	\u0192

```
String prenom = "moi"
System.out.println("message envoy \u0082  \u0085 "+prenom);
```

Permet d'afficher "message envoyé à moi"

Pour commencer avec Java

2.5.2. Récupération d'une donnée saisie à l'écran

Il y a d'autres façons de pratiquer que celle décrite ci-dessous.

Nous allons utiliser la commande qui permet de récupérer un caractère

```
char c;
c=(char)System.in.read();
```

- Nous avons déclarer une variable c de type char qui nous permet de récupérer une position tapée sur le clavier par l'utilisation de la fonction **System.in.read**().
- Pour récupérer un prénom, il nous faut récupérer les x caractères jusqu'à ce que l'utilisateur fasse un retour chariot : le caractère "retour chariot" est "/n" çà ne s'invente pas, c'est comme çà!.
- Maintenant, il nous faut faire "récupération d'un caractère jusqu'à ce que ce caractère soit "\n"
  La commande WHILE permet ce FAIRE TANT QUE

```
while((c=(char)System.in.read()) != '\n')
 {
        S = S + C;
}
```

- Les lignes qui suivent la ligne WHILE sont encadrées par { et } et contiennent les instructions à réaliser si la condition mise derrière le While est vraie.
  Dans notre cas, nous rajoutons derrière la chaîne de caractères ayant S pour nom la valeur de C (zone mémoire de type caractère contenant le caractère tapé au clavier) ce qui nous donne une nouvelle valeur de S et ainsi de suite.
- (char)System.in.read() permet de transformer le caractère reçu depuis le clavier en une donnée de type char : on parle de **casting** , transtypage ou conversion.

Pour commencer avec Java

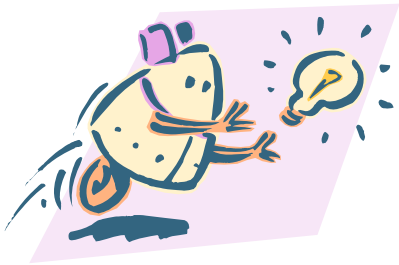
2.6. Création d'une première application

Activité n°1

Notre première application va consister à demander à la personne devant le clavier d'indiquer son nom, puis nous lui enverrons un message "Bonjour" suivi du prénom saisi. Codez cette application puis essayez-la.

Pour commencer avec Java

Correction : voici un exemple possible de code pour l'application



```
import java.lang.*;
/**
 * je démarre en JAVA
 */

public class Class1
{
	public static String Lit()
	{
		String S = "";
		char C;
		while((C=(char)System.in.read()) != '\r')
		 {
			S = S + C;
		 }
		return S;
	}
  /**
  * Point d'entrée principal de l'application.
  */
	public static void main (String[] args) throws java.io.IOException
	{
		String prenom;
		System.out.println("tapez votre prenom");
		prenom = Lit();
		System.out.println("Bonjour " + prenom);
	}
}
```

Pour commencer avec Java

Explications :

Dans ce programme, nous avons deux méthodes Lit et main.

Malgré qu'elle soit placée en fin, la méthode **main** est le début du chemin suivi par la machine pour exécuter le mode opératoire constitué par les instructions.

Puis nous définissons une chaîne de caractères nommée prenom (noter la non accentuation).

Le message "tapez votre prénom" est envoyé pour apparaître sur l'écran.

prenom = Lit() veut dire que dans prenom, nous récupérerons la valeur issue de l'exécution de Lit.

Puis le message "Bonjour Margo" sera affiché à l'écran.

prénom = Lit() débranche le mode opératoire à la ligne "public static String Lit()"

Que se-passe-t-il dans Lit ?
Nous définissons une chaîne de caractères nommée S avec rien dedans et une variable C de type caractère.
Les caractères tapés au clavier sont récupérés par l'instruction "System.in.read()" puis convertis en caractères avant d'être comparés au caractère \n.

Nous bouclons grâce au while jusqu'à ce que le caractère \n soit trouvé.
Pourquoi \n ? : parce qu'il correspond à ce qu'envoie le clavier dans le buffer accessible au programme quand on fait passage à la ligne suivante.

Pour commencer avec Java

Nous avons vu comment déclarer des variables certes, mais pour quoi faire ?

Le faire va se traduire par des instructions plus ou moins sophistiquées.
Les plus courantes constituent des structures de base:

- **L'alternative :** ces instructions permettent l'exécution d'une ou plusieurs instructions en fonction d'une condition.
- **La répétitive :** ces instructions permettent de répéter l'exécution d'une ou plusieurs autres instructions.

2.7. Les structures de base

2.7.1. l'alternative simple :

La forme la plus simple est la suivante :

```
if (<condition>) <instruction>;
```

condition est une expression écrite avec des variables, des constantes et des opérateurs de comparaison (vu plus haut). Instruction est exécutée si le résultat du test exprimé par condition est vrai.
Exemple : si le salaire est plus petit que 10000 alors il est augmenté de 5%.

```
if (dSalaire < 10000)   dSalaire = dSalaire * 1.05;
```

Une deuxième forme est possible :

```
if (<condition>)
        <instruction1>;
else    <instruction2>;
```

condition est une expression écrite avec des variables, des constantes et des opérateurs de comparaison.
Instruction1 est exécutée si le résultat du test exprimé par condition est vrai.
Instruction2 est exécutée si le résultat du test exprimé par condition est faux.

Exemple : si le salaire est plus petit que 10000 alors il est augmenté de 5% sinon il est augmenté de 3%.

```
if (dSalaire < 10000)
                dSalaire = dSalaire * 1.05;
        else    dSalaire = dSalaire * 1.03;
```

Pour commencer avec Java

Les deux formes précédentes ne permettent l'exécution que d'une instruction. Il est souvent nécessaire d'exécuter plusieurs instructions. On parle de **bloc** d'instructions; il est écrit entre { et } La syntaxe est alors :

```
if (<condition>)
   {
    <bloc d'instructions si condition vraie>;
   }
else
   {
    < bloc d'instructions si condition fausse >;
   }
```

Exemple : calcul d'un montant TTC avec TVA à 18,60 si le client est en France ou pas de TVA dans le cas contraire. Nous supposons connaître la quantité commandée et le prix unitaire de l'article commandé.

```
dMontantHt = dQuantitéCde * dPrixUnitaire;
if (strPays = = "France ")
   {
    dMontantTva = dMontantHt * 1.1860;
    dMontantTtc = dMontantHt + dMontantTva;
   }
else
   {
    dMontantTva = 0;
    dMontantTtc = dMontantHt;
   }
```

Pour commencer avec Java

2.7.2. L'alternative imbriquée :

Il est possible d'imbriquer les if quand on a une succession de tests à faire. Exemple : augmentation différente en fonction de la qualification professionnelle.

```
if (strQualif = = "O")
   dSalaire = dSalaire * 1.06;
else if (strQualif = = "E")
         dSalaire = dSalaire * 1.05;
      else if (strQualif = = "M")
               dSalaire = dSalaire * 1.04;
            else if (strQualif = = "A")
                     dSalaire = dSalaire * 1.03;
                  else dSalaire = dSalaire * 1.02;
```

Mais il est possible de rendre plus lisible l'écriture en utilisant l'instruction switch
(case et break sont des instructions allant avec switch).

Le case permet le test de la condition sur une variable de type élémentaire (mon cher Watson).

Break permet l'abandon anticipé de la boucle.

```
Char strQualif;

switch (strQualif)
   {
   case 'O' :
         dSalaire = dSalaire * 1.06;
         break;
   case 'E') :
        dSalaire = dSalaire * 1.05;
        break;
    case 'M' :
        dSalaire = dSalaire * 1.04;
        break;
    case 'A') :
        dSalaire = dSalaire * 1.03;
        break;
     default :  dSalaire = dSalaire * 1.02;
   }
System.out.println("augmentation terminée");
```

L'instruction break provoque un débranchement et l'instruction qui est exécutée à la suite est le System.out.println

Pour commencer avec Java

2.7.3. Itérative:

```
while (<condition>)
   {
    <bloc d'instructions tant que condition vraie>;
   }
```

Exemple : saisie d'un prénom et affichage de bonjour suivi du prénom jusqu'à ce que la saisie soit "Bye".

```
While (strPrenom != "Bye")
  {
    System.print.out ("Bonjour " + strPrenom);
  }
```

System.print.ln permet l'affichage d'une chaîne de caractères.

+ permet de concaténer (de mettre bout à bout) les deux chaînes de caractères "Bonjour" et strPrenom.

Pour commencer avec Java

2.7.4. Boucle :

Quand la condition dépend d'un indice qui évolue à chaque fois, il est possible d'écrire :

```
int nCpt = 1;
int nTotal;
while (nCpt < 15)
{
   nTotal = nTotal + nCpt;
   nCpt = nCpt++;
}
```

Mais il est plus lisible d'écrire :

```
For (nCpt  = 1 ; nCpt < 15; nCpt ++)
{
   nTotal = nTotal + nCpt;
}
```

L'instruction break permet de sortir de la boucle pour aller à l'instruction qui suit la fin du bloc.

L'instruction continue permet également de sortir de la boucle mais en se débranchant vers une étiquette dont le nom est indiqué derrière continue.

Une étiquette est composée d'un nom suivi par deux points suite :

Pour commencer avec Java

3. Quelques notions sur l'orientée objet

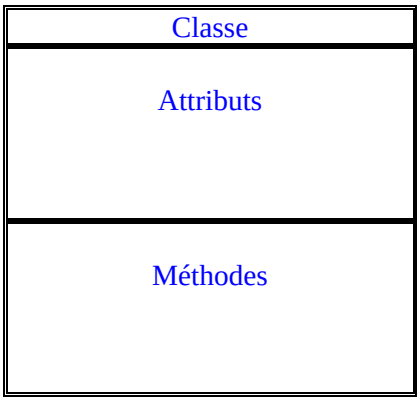
3.1. Les classes et les objets

Une classe est un modèle abstrait utilisé pour créer plusieurs objets présentant des caractéristiques communes et des comportements communs.

Une instance : c'est un objet concret.

Parmi la classe des humains, M. Dupont est un objet de la classe, une instance de la classe.

Il est nécessaire de créer la classe :



Caractéristiques de définition (apparence, état..)

Comportements

Puis on instance un objet par **new,** c'est à dire qu'on crée un objet précis.

Une variable de classe est un élément qui définit un attribut valable pour tout objet de la classe. La variable s'applique à la classe elle-même.

⇒ Une seule valeur est donc stockée quelque soit le nombre d'objets créés pour cette classe.

Une variable d'instance est une information qui définit un attribut d'un objet donné, d'une instance donnée. Dans la classe de l'objet, on trouve de quel type d'attribut il s'agit (String, int…). Les variables d'instance sont aussi appelées variables d'objet.

⇒ Une valeur est donc stockée pour chacun des objets créés pour cette classe.

Une bibliothèque de classes est un groupe de classes conçues pour être utilisées avec d'autres programmes que ceux pour lesquels ils ont été initialement créés. Elles permettent la **réutilisation**.

Pour commencer avec Java

Exemple :

Dans la classe Salarié, on définit l'attribut horaire travaillé : tous les salariés auront le même horaire de base mais c'est un attribut qui peut varier dans le temps : réduction de 39 à 35 heures : il s'agit d'une variable de classe. Le mot clé **static** est utilisé pour sa définition.

Le nom du salarié varie d'un salarié à l'autre : c'est une variable d'instance.

Le coefficient de conversion de francs – € est une donnée qui ne varie pas dans le temps et qui est vrai quelque soit le salarié : on parle de variable à valeur constante. Le mot clé **final** est utilisé pour sa définition

	Salarie	Comment définir
Une variable de classe ➔	- horaire de base	static double dHoraireBase;
Une variable d'instance ➔	- nom salarié	String strNomSalarie;
Une constante ➔	- Euro	final float fEuro = 6.55957;
	Méthodes	Comportements

Exemple d'écriture pour créer une classe :

```
Class Humains                    // permet de définir la classe
{
   String strCouleur;             |
   String strRegion;              |
   double dTaille;                ≻ …attributs
   double dPoids;                 |
   boolean bContent;              |
}
```

Pour commencer avec Java

3.2. Les méthodes

Les méthodes sont des groupes d'instructions reliées les unes aux autres au sein d'une classe d'objets qui agissent sur les objets eux-mêmes ou sur les objets d'autres classes.
Ces méthodes servent à accomplir des taches spécifiques, au même titre que les fonctions dans d'autres langages de programmation. Elles sont toujours situées à l'intérieur de la classe.

Une méthode d'instance, appelée méthode, s'applique à un objet de la classe. Elle apporte une modification à un objet donné.

Une méthode de classe s'applique à la classe entière c'est à dire à toutes les instances de la classe. Le mot clé **static** est utilisé pour sa définition.

Class Humains	Définition de la classe

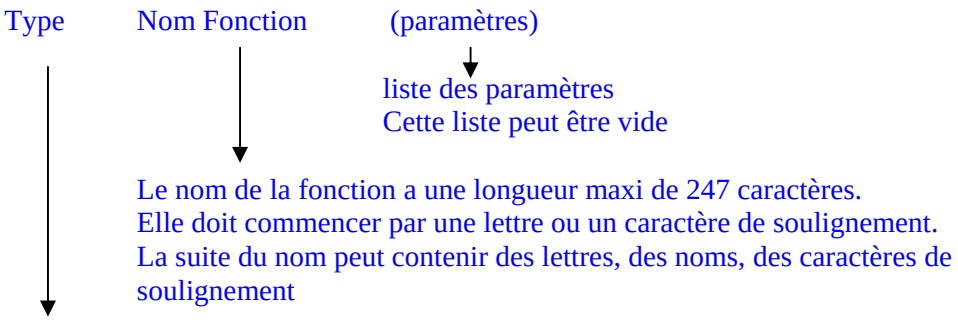
{
String strCouleur;
String strRegion;
double dTaille;
double dPoids;
boolean bContent;
}
	Attributs

Void chercherManger ()
{
if (région = = "Papouasie")
{
System.out.println("Moi prendre arc");
System.out.println("Mettre pagne et aller dans forêt");
System.out.println("Abattre grand caribou");
dPoids = dPoids + 2;
}
else if (région = = "Parisienne")
{
System.out.println("Ne pas oublier la carte bancaire");
System.out.println("Mon veston et je prends la voiture");
System.out.println("Choisir parmi les plats cuisinés");
dPoids = dPoids + 1;
}
}
	Définition d'une méthode d'instance
	
	ici, j'ai pris quelques liberté avec le gibier

static void quiEsTu ()
{
System.out.println("Je suis un humain");
}
}
	Définition d'une méthode de classe

Pour commencer avec Java

La syntaxe de la première ligne d'une fonction est



Puis vous avez un bloc commençant par { et finissant par } entre ces deux signes sont placés les instructions.

Rappel : La fonction **main** est le point d'entrée d'une application JAVA.

Une méthode se termine quand on arrive en fin de bloc ou quand on a une instruction return. Dans les deux cas, l'exécution est reprise en main par l'invocateur de la méthode.

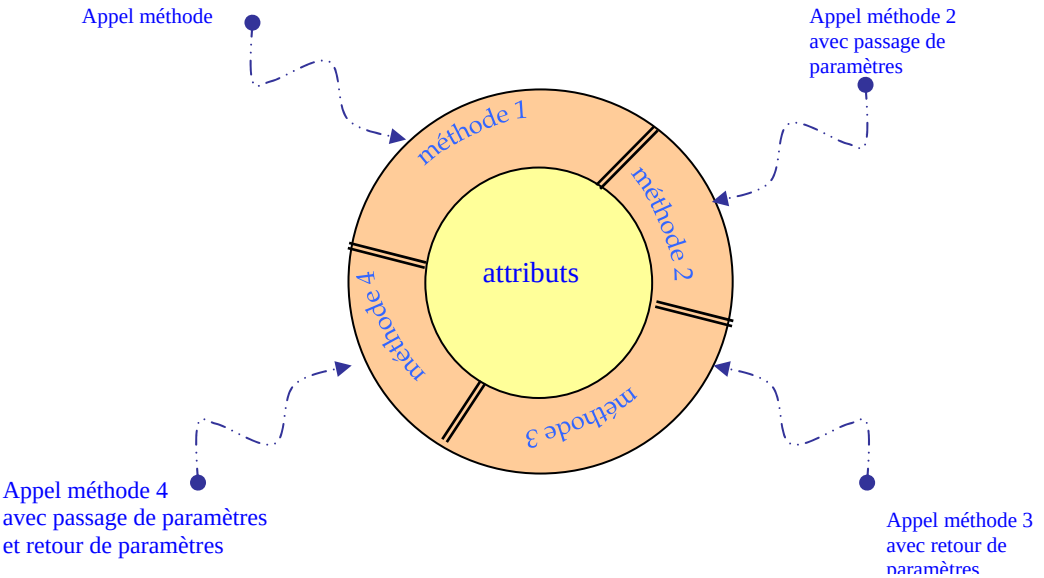
Il peut y avoir plusieurs méthodes écrites dans la même classe.

Comment écririez-vous la méthode indiquant :
"Miam Miam le caribou" pour un papou content
"Baaaah " pour un humain mécontent
"Succulent ce p'tit plat, j'en reprendrai" pour un parisien satisfait.

Pour commencer avec Java

3.3. L'encapsulation

L'encapsulation est un processus qui vise à empêcher les variables d'une classe d'être lues ou modifiées par d'autres classes. Le sel moyen d'utiliser ces variables consiste à appeler des méthodes de cette classe si elles sont accessibles c'est à dire définies avec le mot clé **public.**



Les méthodes peuvent :

- ne mener que des actions sur les données de la classe : seul un ordre est donné par une méthode appelante à une méthode appelée.

- mener une action en utilisant des informations venant de l'extérieur c'est à dire que la méthode appelante émet un ordre avec des informations passées à la méthode appelée qui utilise ces informations pour exécuter les instructions qu'elle contient.

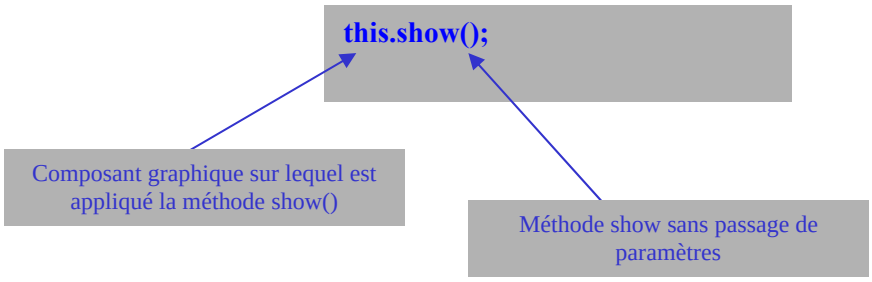
- mener une action puis retourner des informations vers la méthode appelante.

- mener une action en utilisant des informations transmises par la méthode appelante qui reçoit en retour des informations émises par la méthode appelée.

Pour commencer avec Java

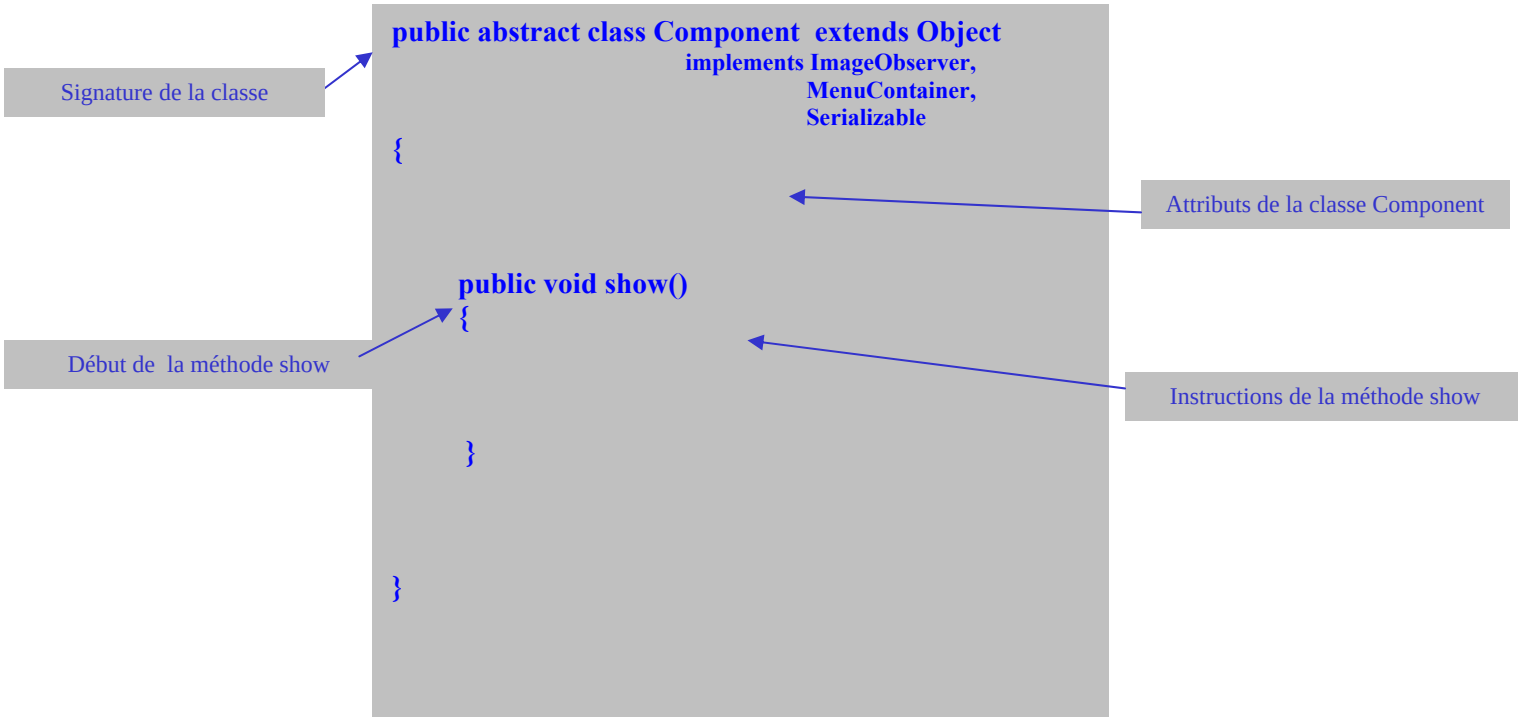
- sans envoi de paramètres :

- Appel :



- permet de montrer un composant graphique (de type JFrame par exemple) représenté par le mot this dans cet exemple.
- Aucune information ne se trouve entre les parenthèses.

- Méthode appelée :

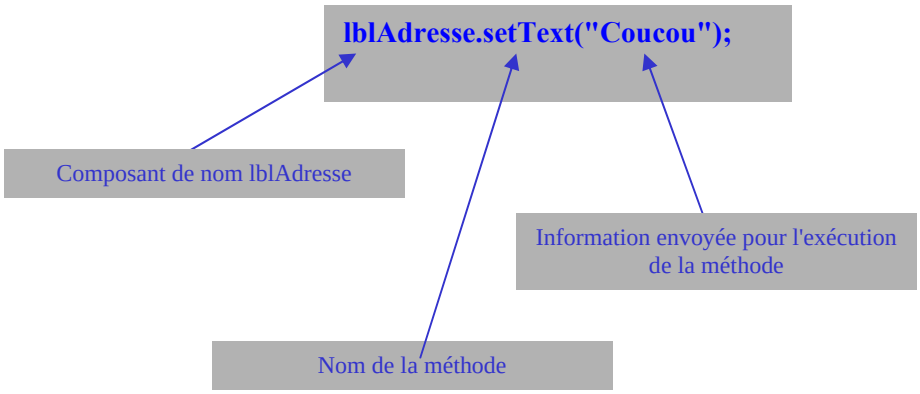


- Dans la ligne de début de la méthode void signifie pas de retour d'informations en fin de méthode, seule l'action demandée est menée.

Pour commencer avec Java

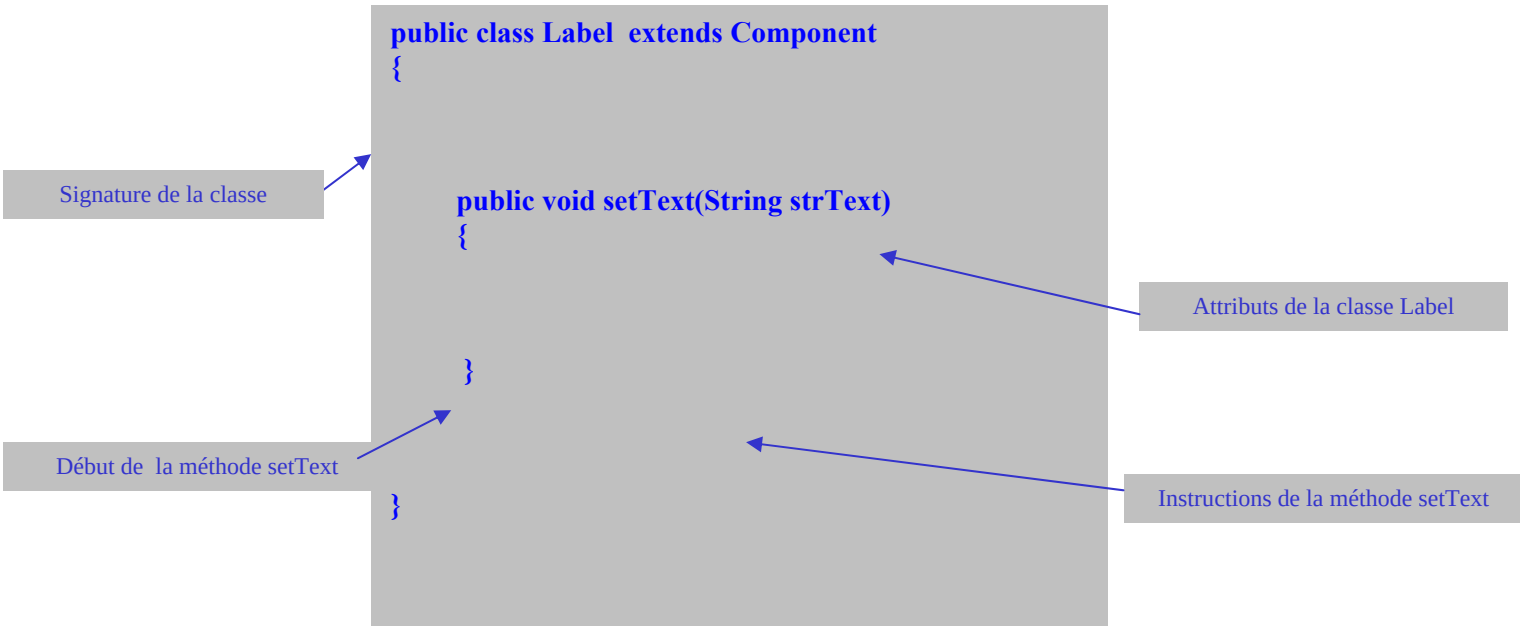
- avec envoi de paramètres :

- Appel :



- permet de mettre "Coucou"comme contenu du composant qui a pour nom lblAdresse.
- Les informations transmises sont indiquées entre les parenthèses qui suivent le nom de la méthode.

- Méthode appelée :



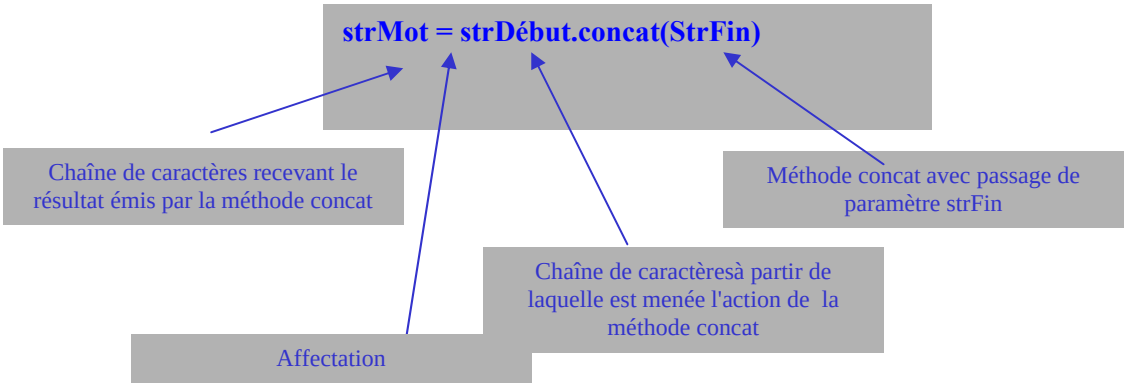
- Dans la ligne de début de la méthode void signifie pas de retour d'informations en fin de méthode, seule l'action demandée est menée.
- String strText correspond à "coucou" de l'appel.

Pour commencer avec Java

- avec passage et récupération de paramètres :

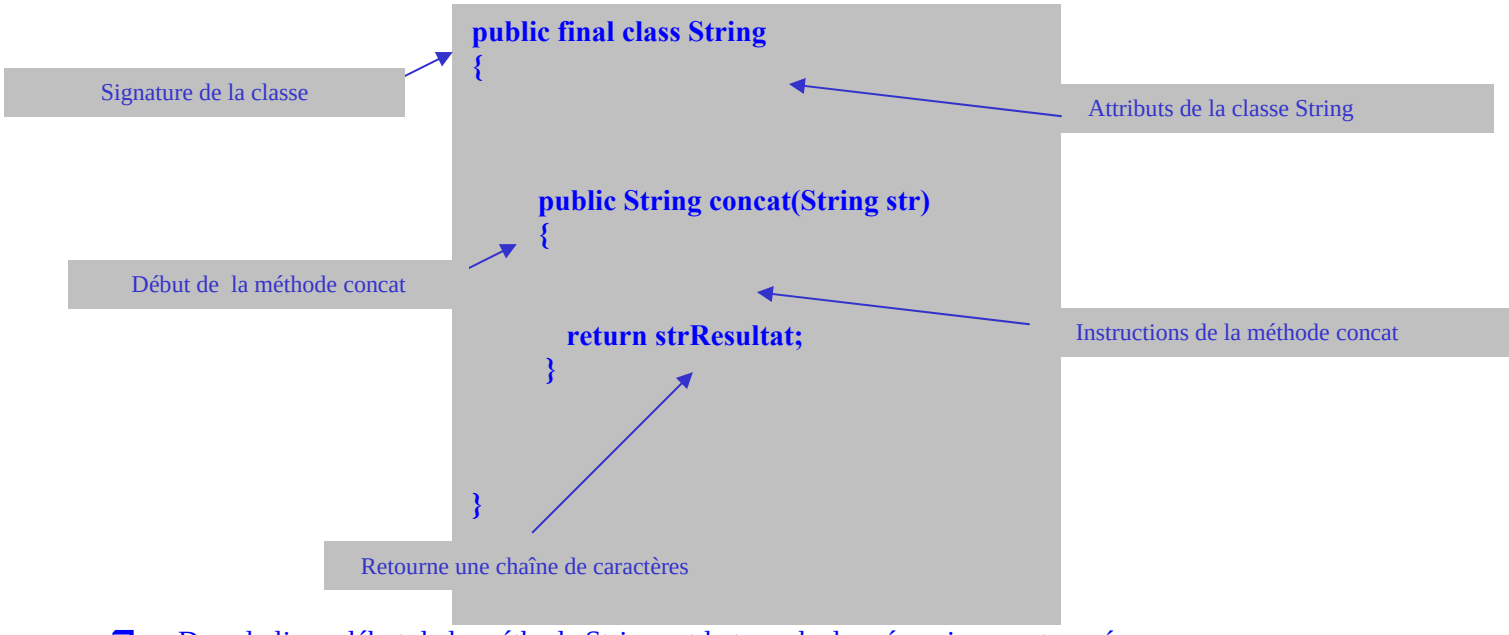
- Appel :

- la méthode concat renvoie une chaîne de caractères obtenue par la mise à la suite de deux autres chaînes de caractères.
- C'est une méthode de la classe String.



- strMot contientra strDébut suivie de strFin.

Méthode appelée :



- Dans la ligne début de la méthode String est le type de donnée qui sera retournée,
- return termine la méthode il est suivi du nom de la donnée qui contient l'information retournée. et qui correspond à la chaîne strMot de l'appel.
- String str est la chaîne de caractères reçue, elle correspond à strFin dans l'appel

Pour commencer avec Java

3.4. Les méthodes "constructeurs"

Considérons une partie du source de la classe Label.

```
public class Label extends Component
{
//  attributs de la classe
        public static final int LEFT;
        public static final int CENTER;
        public static final int RIGHT;
// methodes constructeurs
        public Label()
        {
                code lié à la méthode
        }
        public Label(String strText)
        {
                code lié à la méthode
        }
        public Label(String strText, int nalignment)
        {
                code lié à la méthode
        }

// autres méthodes
        public String getText()
        {
                code lié à la méthode
        }

}
```

Nous y trouvons des méthodes dites "constructeurs" qui ont la particularité d'avoir le même nom que la classe. De plus elles sont public donc accessibles de l'extérieur de la classe.

Ce sont les seules méthodes dont le nom commence par une majuscule.

Dans notre exemple de la classe Label, vous découvrez trois méthodes ayant le même nom; elles ont par contre une différence importante : les paramétres fournis.

Chacune de ces méthodes a un code, un bloc d'instructions qui lui est spécifique.

Dans les classes autres que Label, certaines méthodes peuvent avoir besoin de créer des objets, des instances de la classe Label : nous trouverons alors une instruction new.

- Label lblNom = new Label(); fera appel au code de la méthode dont la signature est **public Label()** et qui construit un label vide

Pour commencer avec Java

- Label lblNom = new Label("Adresse expédition");fera appel au code de la méthode dont la signature est **public Label(String strtext)** et qui construit un label avec le texte indiqué en contenu

- Label lblNom = new Label("Adresse expédition", Label.LEFT);fera appel au code de la méthode dont la signature est **public Label(String strText, int nalignment))** et qui construit un label avec le texte indiqué en contenu et aligné comme indiqué par l'int Label.LEFT

Nous voyons que le point sert à atteindre une méthode de la classe à laquelle l'objet appartient mais aussi à atteindre les attributs de la classe lorsque ceux ci ne sont pas "protected" ou "private" mais "public".

```
String strContenu = LblNom.getText();

Label lblNom = new Label("Adresse expédition", Label.LEFT );
```

Dans ces exemples, nous avons à la fois :

- La déclaration de l'objet String strContenu et Label lblNom,
- L'instanciation faite par la méthode new.

Ces deux opérations peuvent se faire séparant mais toujours dans cet ordre.

Pour commencer avec Java

3.5. L'héritage

L'héritage est un mécanisme qui permet à une classe d'hériter de l'ensemble des comportements et des attributs d'une autre classe.

Une classe qui hérite d'une autre est appelée **sous classe** et la classe qui offre son héritage à une autre est appelée **super classe**.

En java, chaque classe ne peut hériter que d'une super classe : on parle d'héritage simple.

Une classe peut avoir un nombre illimité de sous classes.

La super classe qui est le point de départ dans la hiérarchie java est la classe **Object.**

Notez qu'un nom de classe commence toujours par une majuscule. C'est une convention que nous vous demandons de respecter.

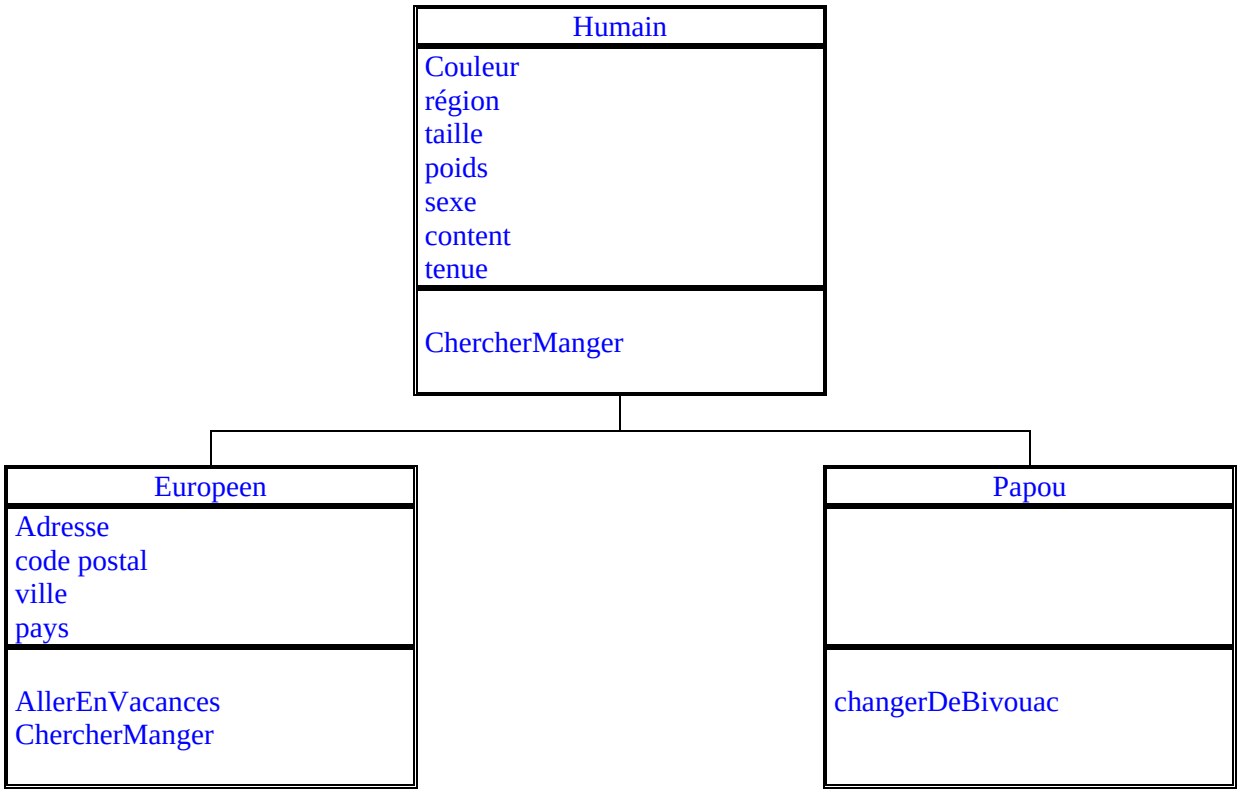
Le sous classement est une opération qui consiste à créer une nouvelle classe qui hérite d'une classe existante.Tous les comportements et les attributs de la classe existante sont repris et la nouvelle classe ne comportera que les attributs nouveaux et les comportements nouveaux.

Comment créer une sous classe?

public class Europeen **extends Humains** {	Définition de la sous classe public permet l'accès aux autres classes qui en ont besoin
	Attributs nouveaux
}	méthodes nouvelles

Quand dans une super classe, une méthode est publique, elle ne peut être redéfinie que par une méthode publique dans la sous classe.

Pour commencer avec Java



Dans l'exemple, quand nous accédons à une instance de la classe Européen, nous avons " Couleur, région, taille, poids, sexe, content, tenue " et aussi " adresse, code postal, ville, pays" et les méthodes utilisables sont " ChercherManger et AllerEnVacances".

Si nous accédons à une instance de la classe Papou, nous n'avons pas d'attributs supplémentaires par rapport à Humain mais nous disposons de deux méthodes " ChercherManger et changerDeBivouac ".

Si dans la classe Européen, on réécrit la méthode " ChercherManger", on dit que l'on redéfinit la méthode. C'est toujours la méthode définie au niveau le plus bas qui est pris en compte.

Quand nous n'avions encore défini que Humain, la méthode " ChercherManger" était prise dans la classe Humain.
Dans le deuxième cas, nous avons créé une sous classe Européen, et si on a instancié Européen, " ChercherManger" est prise dans la classe Européen.

Pour commencer avec Java

3.6. Portée d'une variable

La portée d'une variable est la partie de programme dans laquelle une variable existe et peut être utilisée.

Variable de classe ← portée : toute la classe.

Variable d'instance ← portée : toute la classe.

Variable locale ← portée : le bloc où elle est et les blocs de niveaux inférieurs contenus dans le bloc où elle est déclarée

```
public class Class1
{
String prenom;
public static String Lit()
        {
                String S = "";
                char C;
                while((C=(char)System.in.read()) != '\r')
                 {
                        S = S + C;
                 }
                return S;
        }
  /**
  * Point d'entrée principal de l'application.
  */
        public static void main (String[] args) throws java.io.IOException
        {
                System.out.println("tapez votre prenom");
                prenom = Lit();
                System.out.println("Bonjour " + prenom);
        }
}
```

Dans cet exemple, la chaîne de caractères "prenom" est connue dans toute la suite de la classe : on parle d'une variable globale car elle est déclarée avant toute méthode.

Au contraire la chaîne de caractères "S" est une variable qui ne peut être utilisée que dans la méthode LIT() : on parle alors de variable locale.

Un même nom de variable peut être déclaré dans plusieurs méthodes mais ceci est à éviter. Même s'il s'agit d'un même nom , la variable déclarée dans la méthode 1 a sa propre existence. Quelque soit son contenu, il ne peut être utilisé que dans la méthode 1. La variable de même nom dans la méthode 2 aura un contenu qui sera différent.

Pour commencer avec Java

3.7. Le polymorphisme

Nous en avons eu une représentation avec notre papou et notre parisien.
Quand on envoie le message "allerManger", le Papou et l'Européen, bien qu'objet de la même classe ont des réactions différentes puisque l'un prend son arc et va en forêt et l'autre prend sa voiture et va au supermarché.

Au même message, ils ont deux réactions différentes en fonction de leurs attributs.

Pour commencer avec Java

4. Méthodes graphiques

Elles se trouvent dans la classe graphique **java.awt.Graphics** qu'il faut importer en début de source.
La ligne suivante montre la syntaxe pour importer toutes les classes de java.awt

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

Dans la syntaxe qui suit **x, y** correspondent aux coordonnées du coin haut gauche d'un rectangle contenant la figure.

Les différentes méthodes sont :

- tracé d'une ligne : drawLine
  x y indique les coordonnées d'une extrémité da la droite et x1, y1 pour l'autre extrémité
  Tous les paramètres sont de type integer

```
drawLine(x, y, x1, y1)
```

- tracé d'un rectangle : drawRect
  Tous les paramètres sont de type integer

```
drawRect(x, y, longueur, hauteur)
```

- tracé d'un rectangle plein : fillRect
  Tous les paramètres sont de type integer

```
fillRect(x, y, longueur, hauteur)
```

- tracé d'un rectangle en relief : draw3DRect
  étatbouton est un booléen indiquant "bouton relevé" si vrai (1) ou "abaissé" si faux (0).

```
draw3DRect(x, y, longueur, hauteur, etatbouton)
```

- tracé d'un rectangle au coins arrondis : drawRoundRect
  départ est l'angle en degrés (0 = 3 heures sur montre)
  anglearc est l'angle d'ouverture par rapport à départ
Tous les paramètres sont de type integer

```
drawRoundRect(x, y, longueur, hauteur, départ, anglearc)
```

Pour commencer avec Java

- tracé d'un ovale
  Tous les paramètres sont de type integer

```
drawOval(x, y, longueur, hauteur)
```

- tracé d'un ovale plein

```
fillOval(x, y, longueur, hauteur)
```

- tracé d'un arc

```
drawArc(x, y, longueur, hauteur)
```

- tracé d'un arc plein

```
fillArc(x, y, longueur, hauteur)
```

- tracé d'un polygone
  tablx est un tableau de coordonnées x
  tably est un tableau de coordonnées y
  tablp est un tableau de nombre de points

```
drawPolygon(x, y, longueur, hauteur)
```

- tracé d'un polygone plein

```
fillPolygon (x, y, longueur, hauteur)
```

Pour commencer avec Java

5. L'applet

Une Applet java est un programme qui est exécuté dans un browser tel que Internet explorer ou Nescape. Une applet est intégrée dans une page HTML et est automatiquement téléchargée sur le poste client. Elle est ensuite exécutée par celui-ci.

5.1. Création d'applet

Pour intégrer du code Java dans une page HTML, il faut donc créer une **applet**

Prenons comme exemple, l'applet suivante qui affiche dans une page HTML le texte "Bonjour, comment allez-vous ?"

Nous avons besoin de deux éléments
- Un de type applet
- L'autre de type page HTML

Considérons tout d'abord l'applet

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

public class Applet1 extends Applet
{
        /**
         * Point d'entrée de l'applet.
         */
        Label bonjour = new Label ("Bonjour, comment allez-vous ?  ");

        public void init()
        {
                setBackground (Color.cyan);
                add (bonjour);
        }
}
```

Pour commencer avec Java

Le source de la page HTML est :

```
<HTML>
<HEAD>

<TITLE>Applet, vous avez dit Applet ??? </TITLE>

</HEAD>
<BODY >

<P>
<applet CODE="Applet1.class" WIDTH="600" HEIGHT="300"> </applet></P></BODY>
</HTML>
```

CODE indique le nom du fichier contenant la classe principale de l'applet.

WIDTH indique la largeur en pixels occupée par l'applet sur l'écran.

HEIGHT indique la hauteur en pixels occupée par l'applet sur l'écran au démarrage.

Mise en œuvre

Tout programme java est une classe. Une applet n'échappe pas à cette règle. Si on veut créé une applet, on doit étendre la classe java.applet.Applet.. Cette nouvelle classe contient des méthodes nécessaires à la gestion des applets, et à l'interaction de celle-ci avec le browser.

Les méthodes les plus importantes sont:

public void init()

Le browser fait appel à cette méthode quand l'applet est chargée ou rechargée.
Cette méthode charge les informations telles que images, sons et récupère les paramètres passés depuis la page HTML.

On ne doit pas trouver init plusieurs fois dans une instance d'applet.

Si cette méthode n'est pas indiquée dans l'applet, c'est la méthode originelle qui est utilisée.

public void start()

Après avoir été initialisée, l'applet est démarrée par cette méthode.
Elle est également redémarrée après avoir été stoppée, lorsqu'elle est nouveau visible.
On peut trouver plusieurs start dans une instance d'applet.

public void stop()

Cette méthode permet à l'applet de s'arrêter lorsqu'elle n'est plus visible suite à un changement de page HTML par exemple.

public void destroy()

L'applet est détruite lorsque le browser s'arrête ou avant que l'applet ne soit rechargée.
Cette méthode doit être remplacée si l'on veut stopper des threads créés par start de l'applet. La notion de thread sera revue plus loin.

Pour commencer avec Java

publ**ic void paint(Graphics g)**

Cette méthode est appelée chaque fois que l'on veut redessiner l'applet. Le paramètre est de type Graphics et représente la surface de dessin sur laquelle on doit travailler.

Autre exemple d'applet

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

public class Applet1 extends Applet
{
	Font font;

	public void init()
	{
		font = new Font("TimesRoman",Font.PLAIN.20);
	}

	public void paint(Graphics g)
	{
		g.setFont(font);
		g.setColor(Color.red);
		g.drawString("Y a pas eu de bug !),0, font.getSize());
	}
}
```

Pour commencer avec Java

Autre exemple d'applet cette fois animée par un changement de couleur :

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

public class Applet2 extends Applet implements Runnable
{
	thread tache;
	String strLabel;
	Font font;
	boolean bgris = false;

	public void init()
	{
		font = new java.awt.Font("TimesRoman",Font.PLAIN.24);
		strLabel = getParameter("MonNom");
	}
	public void paint(Graphics g)
	{
		g.setFont(font);
		if (bGris)
			g.setColor(Color.lightGray);
		else
			g.setColor(Color.black);
		g.drawString(strtLabel, 0, font.getSize());
	}
	public void start()
	{
		tache = new Thread(this);
		tache.start();
	}
	public void stop()
	{
		tache.stop();
	}
	public void run()
	{
		while (true)
		{	try
			{ Thread.sleep(1000);		// attente d'une seconde
			}
			catch (InterruptedException e {}
			bGris = !bGris;			// inversion pour changt couleur
			repaint();			// ré affichage
		}
	}
}
```

Pour commencer avec Java

5.2. Passage de paramètres depuis HTML vers l'applet

Le source de la page HTML est :

```
<HTML>
<HEAD>

<TITLE>Applet, vous avez dit Applet ??? </TITLE>

</HEAD>
<BODY >

<P>
<applet CODE="Applet1.class" WIDTH="600" HEIGHT="300">
<param name=MonNom value="Bibi">
</applet></P></BODY>
</HTML>
```

<param name=strNom value="Bibi"> a été intégré entre les balises <applet> et </applet>.

Dans la méthode init() de l'applet, pour récupérer le paramètre transmis :

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

public class MonApplet extends Applet
{
    String strNom;
    public void init()
    {
            strNom = getParameter("MonNom");        // strNom vaut Bibi
            (
             )
    }
}
```

Pour commencer avec Java

6. Utiliser AWT

AWT (Abstract Windowing Toolkit) est un ensemble de classes permettant de créer une interface utilisateur graphique (GUI) et de recevoir des informations de l'utilisateur par l'intermédiaire de la souris ou du clavier.

Pour pouvoir utiliser ces classes, il faut commencer par :

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

6.1. Les différents types d'éléments d'AWT

- Les conteneurs (container)
  Ils sont capables de contenir des canevas, des composants graphiques, ou d'autres conteneurs.

- Les composants graphiques (graphic componants)
  Ce sont les objets traditionnels présents sue une interface graphique (boutons, champ texte…).

- Les canevas (canvas)
  Un canevas est un objet graphique simple sur lequel on ne peut que dessiner et afficher des images. On parle aussi de toile de fond ou fond.

- Les composants de fenêtre (windowing componants)
  Ils permettent de créer des fenêtres (**frame**), des boites de dialogue (**dialog**), de gérer des menus et des barres de titre.

6.2. Fenêtres et panneaux

Un composant Window est un conteneur autonome de haut niveau sans bordure, barre de titre ni barre de menus. Bien qu'un composant Window puisse être utilisé pour implémenter une fenêtre surgissante, vous utiliserez normalement dans votre interface utilisateur une sous-classe de Window, comme l'une des suivantes, à la place de la classe Window réelle :

Pour commencer avec Java

6.2.1. Frame

C'est une fenêtre de haut niveau, avec une bordure et un titre.
Un Frame (cadre) possède les contrôles de fenêtre standard, tels un menu système, des boutons pour réduire ou agrandir la fenêtre et des contrôles pour la redimensionner. Il peut aussi contenir une barre de menus.

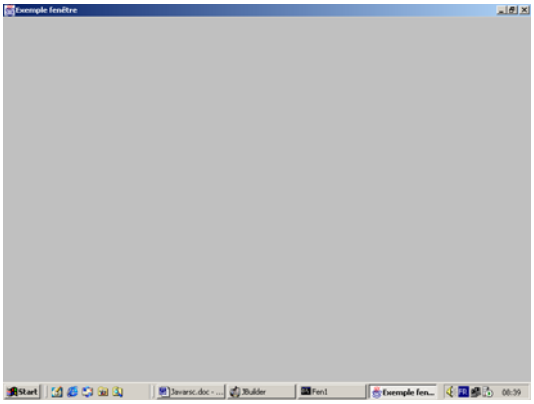
Habituellement, le conteneur principal de l'interface utilisateur d'une application Java (contrairement à une applet) sera une sous-classe personnalisée d'une classe Frame.

La personnalisation consiste à instancier et à positionner d'autres composants dans le cadre, définir des libellés, attacher des contrôles aux données, et ainsi de suite.

Exemple de création d'une fenêtre :

```
import java.awt.* ;
public class Fen1
{
	public static void main(String[] args)
	{
		Frame frameExemple = new Frame();

		// titre
		frameExemple.setTitle("Exemple fenêtre ») ;

		// taille de la fenêtre largeur , hauteur
		frameExemple.setSize(500,300) ;

		// Couleur de fond pour la fenêtre gris clair
		frameExemple.setBackground(Color.lightGray) ;

		// affichage de la fenêtre
		frameExemple.show() ;
	}
}
```

Le résultat est le suivant :



Pour commencer avec Java

La méthode show() permet d'afficher la fenêtre.
Parfois , il est nécessaire de cacher la fenêtre par la méthode hide()

Il est possible d'utiliser aussi la méthode setVisible :

- setVisible(true) pour afficher la fenêtre
- setVisible(false) pour la cacher

6.2.2. Dialog

Une fenêtre surgissante, semblable à un Frame, mais qui nécessite un parent et ne peut pas contenir de barre de menus.

Les boîtes de dialogue sont utilisées pour obtenir une saisie ou pour communiquer des avertissements. Elles sont généralement temporaires et de l'un des types suivants :

- Modale : Empêche la saisie dans n'importe quelle autre fenêtre de l'application tant que le dialogue n'est pas refermé.
- Non modale: Permet d'entrer des informations à la fois dans la boîte de dialogue et dans le reste de l'application.

C'est également un conteneur pour des panneaux, des boutons …

Exemple de création d'une boite de dialogue :

```
import java.awt.* ;
public class Dial1
{
	public static void main(String[] args)
	{
		Dialog dialogExemple = new Dialog (new Frame(), true);

		// titre
		dialogExemple.setTitle("Exemple boite de dialogue ") ;

		// taille de la fenêtre largeur , hauteur
		dialogExemple.setSize(200,100) ;

		// Couleur de fond pour la fenêtre gris clair
		dialogExemple.setBackground(Color.lightGray) ;

		// affichage de la fenêtre
		dialogExemple.show() ;
	}
}
```

La classe dialog permet d'utiliser un ensemble de méthodes spécifiques.

- **setModal** permet de définir si il y a possiblité de saisie(true) ou nom (false) dans la boite de dialogue
- **setResizable** permet d'indiquer si l'utilisateur peut, à l'aide de la souris, redimensionner la boite de dialogue (true) ou non (false).
- **Show** et **hide** comme pour les frame.

La classe FileDialog permet de créer une boite de dialogue de gestion de fichiers.

Pour commencer avec Java

6.2.3. Panel

Un composant Panel, panneau, est un conteneur d'interface utilisateur simple, sans bordure ni titre, utilisé pour regrouper d'autres composants, tels des boutons, des cases à cocher ou des champs de texte.
Les panneaux sont incorporés dans un autre conteneur d'interface utilisateur, par exemple dans un Frame, un Dialog. ou imbriqués dans d'autres panneaux.

6.2.4. Applet

Une sous-classe de la classe Panel utilisée pour construire un programme devant être incorporé dans une page HTML et exécuté dans un navigateur HTML ou dans un visualiseur d'applet.
Applet étant une sous-classe de Panel, elle peut contenir des composants, mais elle ne possède ni bordure, ni titre.

Pour commencer avec Java

6.2.5. Les mises en page

Pour définir l'apparence de l'interface graphique, on utilise un gestionnaire de mise en page : **layout manager**. Il existe plusieurs gestionnaires de mises en page permettant chacun une mise en forme différente.

FlowLayout : gestionnaire par défaut .
C'est une mise en page où les composants sont placés les uns à la suite des autres en fonction de l'ordre des instructions d'ajout de composants dans le conteneur.

```
FlowLayout miseEnPage = new FlowLayout(FlowLayout.CENTER, 10, 30);
```

Exemple :

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

public class TestFlow extends Applet
{
	public void init()
	{
		FlowLayout miseEnPage = new FlowLayout(FlowLayout.CENTER, 10, 10) ;
		setLayout(miseEnPage);

		for(int i=0 ; i < 6; i++)
		{
			Button cmdTest = new Button("commande " + I");
			add(cmdTest);
		}
	}
}
```

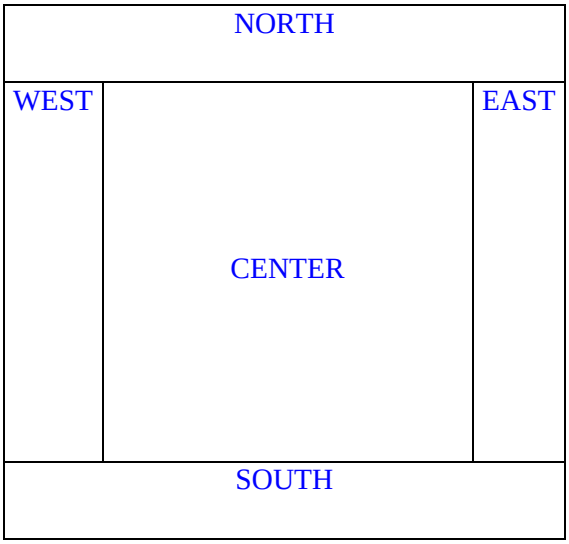
GridLayout :
C'est une mise en page en ligne-colonne. Les composants auront tous la même taille.

Exemple : création d'une mise en page de 5 colonnes sur 3 lignes avec un espacement horizontal de 10 et un espacement vertical de 15. Le code sera identique à l'exemple ci-dessus sauf la ligne d'instantiation de la mise en page.

```
GridLayout gridExemple = new GridLayout(3. 5.10, 15);
```

Pour commencer avec Java

BorderLayout :
c'est une mise en page avec la présentation suivante :



Exemple :

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

public class TestBorder extends Applet
{
        BorderLayout miseEnPage = new BorderLayout() ;
        Button cmdTest1 = new Button("Nord");
        Button cmdTest2 = new Button("Ouest");
        Button cmdTest3 = new Button("Centre");
        Button cmdTest4 = new Button("Est");
        Button cmdTest5 = new Button("Sud");

        public void init()
        {
                setLayout(miseEnPage);
                add("NORTH", cmdTest1);
                add("WEST", cmdTest2);
                add("CENTER", cmdTest3);
                add("EAST", cmdTest4);
                add("SOUTH", cmdTest5);
        }
}
```

Pour commencer avec Java

Si vous souhaitez gérer vous-mêmes les emplacements de vos composants dans un même conteneur (par exemple un Frame), incorporer à votre source la méthode suivante :
// x et y sont les coordonnées exprimées en cellules

```
// larg et haut sont le nombre de cellules en largeur et en hauteur
// largcol et hautlig  sont les proportions prises par les lignes & colonnes

        void placeElement(GridBagConstraints gbc, int x, int y,
                          int larg, int haut, int largCol, int hautLig)
        {
                gbc.gridx       = x;
                gbc.gridy       = y;
                gbc.gridwidth   = larg;
                gbc.gridheight  = haut;
                gbc.weightx     = largCol;
                gbc.weighty     = hautLig;
        }
```

Avant d'effectuer le add du composant définissez son emplacement

Dans l'exemple colonne 3 ligne 2 sur 1 case en largeur et 1 case en hauteur , représentant 40 % de la largeur du conteneur.

```
placeElement(gblEmplacement, 3, 2, 1, 1, 40, 0);
gblEmplacement.anchor      = GridBagConstraints.WEST;

//  Création de l'instance du composant par new
gblGrille.setConstraints(chkMademoiselle, gblEmplacement);
```

Dès que vous commencer à positionner un composant sur une ligne ou sur une colonne, vous devez indiquer le pourcentage tenu par cette ligne ou cette colonne.

Pour les composants suivants sur la même ligne ou la même colonne, le pourcentage est mis à 0 ce qui indique que l'on reprend celui fixé avec le premier composant.

Le total des pourcentages doit être exactement 100 %.

Pour commencer avec Java

6.2.6. Intitulés (Label)

Ils sont aussi appelés étiquettes.

Ils permettent de créer du texte sur une interface utilisateur.

Créer un intitulé :

```
Label strEtiq1 = new Label("Saisir un n° de client");
```

On peut indiquer l'emplacement de l'intitulé :

```
Label strEtiq1 = new Label("Choix : ", Label.CENTER);        // au centre
Label strEtiq1 = new Label("Choix : ", Label.RIGHT);         // à droite
```

Il est possible de créer un label vide que l'on remplira plus tard par setText :

```
Label strVide = new Label();
```

Mettre un texte dans un intitulé

```
strVide.setText("Bonjour");
```

Récupérer le texte d'un intitulé

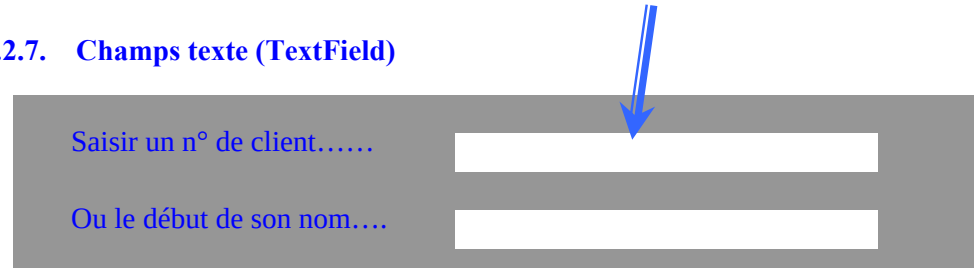
```
String strQuoi = strVide .getText();
```

Dans tous les cas, il faut utiliser la méthode init et y indiquer :

```
Add(strVide);
```

Pour commencer avec Java

6.2.7. Champs texte (TextField)



C'est un composant dans lequel l'utilisateur peut saisir du texte.

Créer un champ texte de 30 caractères sans texte :

```
TextField strDebut  = new TextField(30);
```

Créer un champ texte de 30 caractères avec du texte :

```
TextField strDebut  = new TextField("ceci est ma valeur par défaut",30);
```

Pour masquer la saisie faite dans ce champ au fur et à mesure (exemple d'un mot de passe)

```
TextField strMotPasse = new TextField(30);
strMotPasse.setEchoCharacter(" ");
```

Dans tous les cas, il faut utiliser la méthode init et y indiquer :

```
Add(strMotPasse);
```

Les méthodes utilisables :

Nom	Action
getText()	Retourne le texte contenu dans le champ
setText("texte par défaut")	Remplir le champ avec le texte indiqué
setEditable(false)	Indique que le champ ne peut pas être modifié
setEditable(true)	Indique que l'on peut saisir dans ce champ
isEditable()	Retourne une valeur true si champ modifiable ou false dans le cas de champ protégé en saisie

Pour commencer avec Java

6.2.8. Zones de texte

C'est un composant dans lequel l'utilisateur peut saisir plusieurs lignes de texte.

Créer une zone de texte de 2 lignes sur 30 caractères sans texte :

```
TextArea strAdresse  = new TextArea(30, 2);
```

Créer une zone de texte de taille indéterminée mais avec du texte :

```
TextArea strObservation  = new TextField("ceci est une observation");
```

Dans tous les cas, il faut utiliser la méthode init et y indiquer :

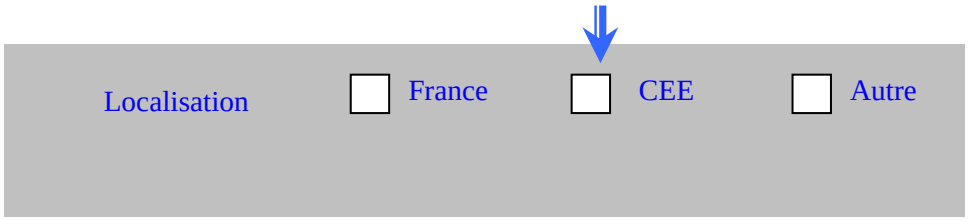
```
Add(strMotPasse);
```

Les méthodes utilisables :

Nom	Action
getText()	Retourne le texte contenu dans la zone
setText("texte par défaut")	Remplir la zone avec le texte indiqué
setEditable(false)	Indique que la zone ne peut pas être modifiée
setEditable(true)	Indique que l'on peut saisir dans cette zone
isEditable()	Retourne une valeur true si champ modifiable ou false dans le cas de zone protégée en saisie
insert(string, int)	Insère la chaîne de caractères à partir de la position indiquée par int
replace(string, int1, int1)	Remplace la chaîne de caractères à partir de la position indiquée par int1 jusqu'à la position int2

Pour commencer avec Java

6.2.9. Cases à cocher



Ce sont des cases représentées par des carrés que l'on peut "cliquer" ou décliquer".

Quand on peut avoir plusieurs cases à cocher, on utilise des CheckBox.

Quand on veut avoir des cases dont une seule peut être cliquée, on regroupe ces cases dans une CheckBoxGroup.

Créer une case à cocher sans texte :

```
CheckBox strChoix1  = new CheckBox ();
```

Créer une case à cocher avec texte :

```
CheckBox strChoix2  = new CheckBox ("OK");
```

Créer un groupe de cases à cocher :

```
CheckBoxGroup civilité = new CheckBoxGroup ();
CheckBox strMonsieur  = new CheckBox ("Monsieur", civilité,true);
CheckBox str Madame = new CheckBox ("Madame", civilité,false);
CheckBox strMlle       = new CheckBox ("Mademoiselle", civilité,false);
```

Ici on affiche trois cases avec la case Monsieur cochée

Dans tous les cas, il faut utiliser la méthode init et y indiquer :

```
Add(strMonsieur);
```

Pour commencer avec Java

Les méthodes utilisables :

Nom	Action
setState(true)	Permet de cocher la case
setState(false)	Permet de décocher la case
getCurrent()	Permet de déterminer la case cochée
setCurrent(nomchkbox)	Permet de cocher une case précise

Pour commencer avec Java

6.2.10. Boutons

Ce sont des zones "cliquables" qui permettent de déclencher des actions.

Créer un bouton sans texte :

```
Button btnVide  = new Button ();
```

Créer un bouton avec texte :

```
Button btnOk  = new Button ("OK");
```

Dans tous les cas, il faut utiliser la méthode init et y indiquer :

```
Add(btnOk);
```

Nous verrons lors de la gestion des évènements comment utiliser les boutons.

Pour commencer avec Java

6.2.11. Listes de choix

Comment créer une liste de choix :

```
Choice ColorChooser = new Choice();
ColorChooser.add("Green");
ColorChooser.add("Red");
ColorChooser.add("Blue");
```

Les méthodes utilisables :

Nom	Action
getItem(int)	Retourne le texte de l'élément n° int dans la liste
countItem()	Retourne le nombre d'éléments de la liste
getSelectedIndex()	Retourne le n° d'élément choisi dans la liste
getSelectedItem(i	Retourne le texte de l'élément choisi
select(int)	Sélectionne le poste n° int dans la liste
select(String)	Sélectionne le poste ayant le texte correspondant à la chaîne dans la liste

6.2.12. Listes à défilement

```
List lstPlanete = new List(4, false);   // crée une liste avec 4 éléments visibles
                                        //      mais 1 seul choix possible (false)
lstPlanete.addElement("Mercure");
lstPlanete.addElement ("Venus");
lstPlanete.addElement ("Terre");
lstPlanete.addElement ("Mars");
lstPlanete.addElement ("Jupiter");
lstPlanete.addElement ("Saturne");
lstPlanete.addElement ("Uranus");
lstPlanete.addElement ("Neptune");
lstPlanete.addElement ("Pluton");

cntContainer1.add(lstPlanete);
```

Les méthodes utilisables :

Nom	Action
getSelectedIndexed()	Retourne une matrice de n° d'éléments choisis dans la liste
getSelectedItem(i	Retourne une matrice de chaînes de caractères correspondant aux éléments choisis

Pour commencer avec Java

Pour commencer avec Java

7. Utiliser SWING

L'ensemble des composants Swing ont tous pour début de dénomination un "J".

Ils se trouvent dans javax.swing.*.

Nous disposons d'une fenêtre principale que nous allons remplir avec des composants graphiques.

On distinguera :

- Les conteneurs de haut niveau
  - JFrame
  - JWindow
  - Applet
  - JDialog
- Ainsi que des composants "légers"
  - JLabel 75
  - JTextField..... 75
  - JTextArea 76
  - JButton 76
  - JRadioButton..... 81
  - JComboBox..... 97
  - JCheckBox 86
  - Jlist 90
  - JScrollBar..... 130
  - JprogressBar..... 131
  - JtoolBar 130
  - JOptionPane 102
  - JScrollPane..... 90
  - JSplitPane..... 140
  - JTabbedPane 130
  - Panel 111
  - JSlider..... 123
  - JToolTip 96
  - JPasswordField..... 76
  - Jtable 131
  - JTree 140
  - JColorChooser..... 124
  - JMenu , JMenuBar, et entrées de menus. 119
  - JPopupMenu..... 130
  - JEditorPane
  - JTextPane
  - JFileChooser
  - JInternalFrame

Quand vous définissez un GUI (Interface Graphique Utilisateur), un certain nombre des composants sont déclencheurs d'évènements. Ces évènements seront à tester si nécessaire pour déclencher des actions de contrôle, de mises à jour, de changement d'interface, ….

Jusqu'à maintenant, nous faisions sans en avoir l'air de la programmation procédurale Nous allons ici entrer dans l'ère de la

programmation événementielle

Pour commencer avec Java

7.1. Exemple de structure de programme

Nous allons détailler, dans l'ordre, les différents morceaux de code à mettre bout à bout pour créer un programme, une classe permettant l'affichage et la gestion d'un GUI (Graphic User Interface) c'est à dire un ensemble de composants permettant à l'utilisateur de saisir et d'interroger ses informations.

Les imports : Dans les différentes méthodes qui constituent le programme, nous aurons à utiliser d'autres méthodes qui existent déjà soient parce qu'elles sont dans les standards Java soient parce qu'elles ont été créées et testées dans une autre classe par nos soins.

Ces méthodes appartiennent à des classe, elles mêmes contenues dans des packages.

Pour les appeler, il est possible d'écrire comme nous l'avons déjà fait
System.out.println
ce qui donne le chemin pour accéder à la méthode qui se trouve dans la classe nommée System.

Ecrire tous les noms de méthodes ainsi, alourdirait la frappe du code et surtout sa lisibilité.

Pour n'indiquer que le nom de la méthode non précédé par le chemin d'accès, nous indiquerons une fois pour toute les packages où se trouvent les classes nous intéressant en début de code avant même la déclaration de la classe.

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

Puis écrire la classe proprement dite: y inclure les déclarations d'objets qui auront une portée globale c'est à dire connus et utilisables par l'ensemble des méthodes de la classe. Nous déclarerons en particulier tous les composants constituant notre interface.

```
public class NomClasse extends JFrame
{
    // Déclaration des objets et variables globales
       Jbutton btnExit;
       int nMax = 255;
       …
   //=============================================================
```

extends JFrame permet l'héritage par notre classe de toutes les méthodes et tous les attributs constituant la classe JFrame.

Pour commencer avec Java

Nous sommes dans le cadre d'une application, donc description de la méthode main. Nous y instancions un objet de la classe par un new.

```
// entrée dans l'application
public static void main(String args[])
{
        new NomClasse ();
 }
```

La méthode main peut n'être décrite que dans une seule classe appelée "classe controler" qui contrôlerait la cinématique d'exécution des autres classes. Un exemple d'application, que nous verrons dans le chapitre Construire une application avec SWING page 101, fonctionne de cette manière.
Il peut être intéressant de mettre systématiquement la méthode main pour pouvoir tester seulement la classe sans avoir à passer par le lancement de la classe "controler".

Puis nous inclurons la méthode "constructeur de l'interface en public : elle est lancée par le main ou appelée par la classe "controler".

```
// méthode construction de NomClasse
public NomClasse ()
{
initGUI();
```

Tout de suite après la méthode initGui, tester la sortie c'est à dire la survenue d'un événement clic sur la croix en haut à droite de la fenêtre de notre application.

```
WindowListener  jecoute = new WindowAdapter()
    {
      public void windowClosing(WindowEvent e)
      {
            System.exit(0);
      }
    };
this.addWindowListener(jecoute);
}
```

Nous verrons par la suite, une autre façon d'écrire la sortie de l'application.

Pour commencer avec Java

Description de la méthode initGui

```
public void initGUI()
{
 this.getContentPane().setLayout(null);       // permet de gérer les emplacements sans
                                               //recourir à des formats de présentations standards
 this.setTitle("Mon titre");                  // on indique le titre de la fenêtre
 this.setSize(500, 400);                      // et la taille du container (ici JFrame)

 /*  application des méthodes de mise en forme des composants légers
     (boutons, labels,…)
       tels Font, couleurs, bordures, taille, alignement, raccourci clavier…


 // Ajouter des composants au container de façon visible

 this.setVisible(true);

 this.getContentPane().add( ?????);


 // ajouter les objet Listener concernés pour la gestion des évènements
 ???.addActionListener(new bEcoute());
}
```

Ne jamais omettre le **setBounds** des composants lorsqu'on a **setLayout(null)** sinon vos composants n'apparaissent pas sur votre interface.

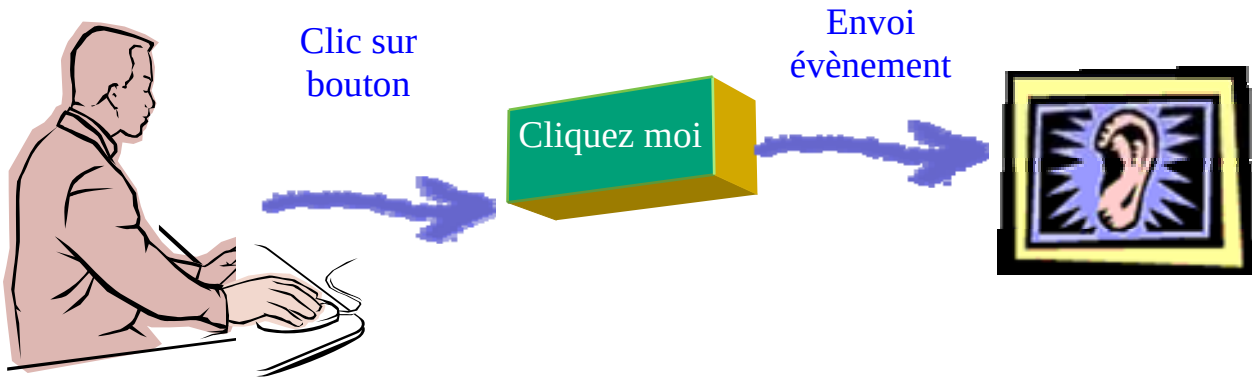
Pour commencer avec Java

Pour commencer avec Java

7.2. Les évènements et les "listeners"

Lorsqu'une application s'exécute, son comportement varie en fonction d'actions comme les clics de souris, l'appui sur les touches, l'ouverture et la fermeture des fenêtres, etc.

Exemple :



Un composant qui génère des événements est un composant source ; un composant qui répond par des actions à des occurrences d'événements est un écouteur ou auditeur ' (in english listener) : c'est le gestionnaire des évènements.

Le gestionnaire d'événement est un interface dont le nom qui se termine par "Listener".

Il attend des évènements de type précis dont le nom se termine par "Event".

Notre gestionnaire d'évènements possède des méthodes telles que actionPerformed, focusLost, …
S'il a plusieurs méthodes, il existe alors une classe Adapter qui nous permettra de ne traiter que la méthode souhaitée. Avant l'existence de cette classe, il fallait redéfinir toutes les méthodes du gestionnaire d'évènements.

Pour commencer avec Java

Les ensembles d'événements standard sont :

- Evénements d'actions
  Le gestionnaire de l'événement est ActionListener qui attend un ActionEvent
  Méthode de ActionListener
    - actionPerformed

- Evénements d'ajustement
  Le gestionnaire de l'événement est AdjustmentListener qui attend un AdjustmentEvent
  Méthode de AdjustmentListener :
    - adjustmentValueChanged

- Evénements de composants
  Le gestionnaire de l'événement est ComponentListener qui attend unComponentEvent
  Méthodes de ComponentListener
    - componentHidden
    - componentMoved
    - componentResized
    - componentShown

  Il existe un ComponentAdapter

- Evénements de conteneurs
  Le gestionnaire de l'événement est ContainerListener qui attend un ContainerEvent
  Méthodes de ContainerListener
    - componentAdded
    - componentRemoved

  Il existe un ContainerAdapter

- Evénements de focalisation
  Le gestionnaire de l'événement est FocusListener qui attend un FocusEvent
  Méthodes de FocusListener
    - focusGained
    - focusLost

  Il existe un FocusAdapter

- Evénements d'éléments
  Le gestionnaire de l'événement est ItemListener qui attend un ItemEvent
  Méthodes de ItemListener
    - ItemStateChanged

- Evénements de touches
  Le gestionnaire de l'événement est KeyListener qui attend un KeyEvent
  Méthodes de KeyListener
    - keyPressed
    - keyReleased
    - keyTyped

  Il existe un KeyAdapter

Pour commencer avec Java

- Evénements de la souris
  Le gestionnaire de l'événement est MouseListener qui attend un MouseEvent
  Méthodes de MouseListener
    - MouseClicked
    - mouseEntered
    - mouseExited
    - mousePressed
    - mouseReleased

  Il existe un MouseAdapter

- Evénements de mouvement de la souris
  Le gestionnaire de l'événement est MouseMotionListener qui attend un MouseEvent
  Méthodes de MouseMotionListener
    - mouseDragged
    - mouseMoved

  Il existe un MouseMotionAdapter

- Evénements de texte
  Le gestionnaire de l'événement est TextListener qui attend un TextEvent
  Méthodes de TextListener
    - textValueChanged

- Evénements de fenêtres
  Le gestionnaire de l'événement est WindowListener
  Méthodes de WindowListener
    - windowClosed
    - windowClosing
    - windowDeiconified
    - windowIconified
    - windowOpened
    - windowActivated()
    - windowDeactivated()

  Il existe un WindowAdaptateur

Pour commencer avec Java

Il y a généralement au moins trois éléments de code impliqués dans une gestion d'événements.

- Création du composant source qui engendre l'événement.
- Création d'un lien entre le composant source et le gestionnaire d'évènements.
  par exemple addActionListener qui indique quel type d'évènements sera traité.
  (removeActionListener permet de désunir le capteur et le composant).

```
btnOK.addActionListener(evtOk);
```

- Code de la classe gestionnaire d'évènements

Il existe deux façons de la programmer :

Adaptateur d'inner Classes anonymes

Les avantages des inner classes sont les suivants :
- Le code est généré en ligne, ce qui simplifie l'apparence du code.
- Aucune classe séparée n'est générée.
- L'inner classe peut accéder à toutes les variables relevant de la portée dans laquelle elle est déclarée, contrairement aux adaptateurs d'événements standard, qui disposent uniquement d'un accès de niveau public et paquet.

Exemple : utilisant les classes intégrées ou classes anonymes: lorsque l'utilisateur clique sur la croix pour fermer une fenêtre, il faut quitter la fenêtre.

```
public NomClassn()
{
initGUI();
WindowListener  jecoute = new WindowAdapter()    // constructeur
    {
      public void windowClosing(WindowEvent e)
      {
            System.exit(0);
      }
    };
this.addWindowListener(jecoute);                  // ajout de l'objet ainsi créé à l'interface
}
```

Le code permettant de sortir de l'application est écrit dans le corps de la classe permettant l'exécution.

Pour commencer avec Java

Adaptateurs d'événements standard

Le code permettant les traitements liés au déclenchement des évènements est "sorti" dans une classe ecite en fin de source..

Exemple : la classe principale

```
import java.util.*;

public class PanelGauche extends JPanel
{
  JButton btnExit   = new JButton("Sortie");
 JButton btnCreat   = new JButton("Ajout");
 JButton btnChang  = new JButton("Changement");
   //=============================================================

 public void initGauche()
 {
  //  format et ajout des composants du GUI
  //

   instanciation des composants

  this.add(btnExit);
  this.add(btnCreat);
  this.add(btnChang);
  //
  //    mise en place gestion des évènements
  //
  Ecoute evtExit  = new Ecoute(this, evtExit.nExit);
  Ecoute evtCreat = new Ecoute(this, evtCreat.nCreat);
  Ecoute evtChang = new Ecoute(this, evtChang.nChang);
  btnExit..addActionListener(evtExit);
  btnCreat.addActionListener(evtCreat);
  btnChang.addActionListener(evtChang);
 }
```

Pour commencer avec Java

La classe de gestion des évènements, dans notre cas, s'appelle Ecoute : elle a été invoquée dans la classe principale car nous l'avons instanciée. : Elle est décrite ci-dessous avec ses attributs et ses méthodes.

```
class Ecoute implements ActionListener
{
  PanelGauche panelEcoute;
  int nbtnClic;
  final int nCreat = 0;                  // constante de valeur 0
  final int nChang = 1;                  // constante de valeur 0
  final int nExit  = 2;                  // constante de valeur 0
// constructeur
  public Ecoute(PanelGauche panelOui, int nBouton)
  {
    this.panelEcoute = panelOui;
    this.nbtnClic    = nBouton;
  }
    public void actionPerformed(ActionEvent evt)
      {
        switch(nbtnClic)
          {
            case nExit:
                    System.exit(0);
                    break;
            case nCreat:
                      strAction = "creat";
                      strIdent = " ";
                      break;
            case nChang:
                    strAction = "chang";
                    strIdent = strParam.getText();
                    break;
                }
  }
}
}
```

Pour commencer avec Java

Comme pour AWT, Swing met à disposition un certain nombre de composants déjà créés.

7.3. JLabel

Créer un intitulé :

```
JLabel strEtiq1 = new JLabel("Saisir un n° de client");
```

On peut indiquer l'emplacement de l'intitulé :

```
JLabel strEtiq1 = new JLabel("Choix : ", Label.CENTER);        // au centre
JLabel strEtiq1 = new JLabel("Choix : ", Label.RIGHT);         // à droite
```

Il est possible de créer un label vide que l'on remplira plus tard par setText :

```
JLabel strVide = new Label();
```

Mettre un texte dans un intitulé

```
strVide.setText("Bonjour");
```

Récupérer le texte d'un intitulé

```
String strQuoi = strVide .getText();
```

7.4. JTextField

Créer un champ texte de 30 caractères sans texte :

```
JTextField strDebut  = new JTextField(30);
```

Créer un champ texte de 30 caractères avec du texte :

```
JTextField strDebut  = new JTextField("Les oiseaux chantent",30);
```

Créer un champ texte de 30 caractères avec du texte en utilisant un modèle:

```
JTextField strDebut  = new JTextField(Document D1,«ma valeur par defaut",30);
```

Pour commencer avec Java

7.5. JPasswordField

C'est un JTextField particulier car l'utilisateur ne voit pas les carctères qu'il entre au clavier.

7.6. JTextArea

Mêmes possibilités que TextArea vu sous AWT.

7.7. JButton

Créer un bouton sans texte :

```
JButton cmdVide  = new JButton ();
```

Créer un bouton avec texte :

```
JButton cmdOk  = new JButton ("OK");
```

Créer un bouton avec icône :

```
JButton cmdtop = new JButton ("images/coucou.gif");
```

Créer un bouton avec icône :et texte

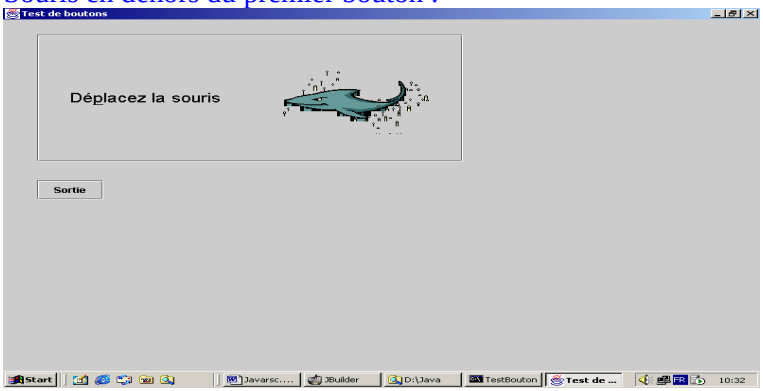
```
Button cmdOk  = new Button ("OK","images/coucou.gif");;
```

Pour commencer avec Java

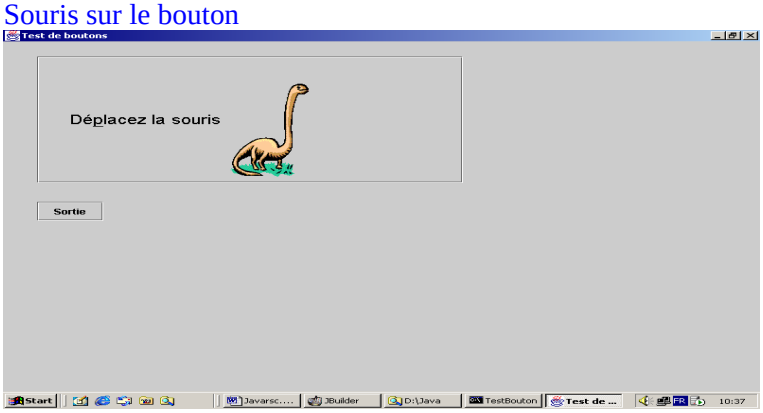
Exemple : Le programme qui suit affiche un JFrame avec deux boutons.

Le premier contient un texte «déplacer la souris» et une icône qui varie en fonction de la position de la souris sur l'écran.
Le deuxième est un bouton avec texte «sortie» qui lorsqu'on le clique termine le programme comme la X.en haut à droite du JFrame.

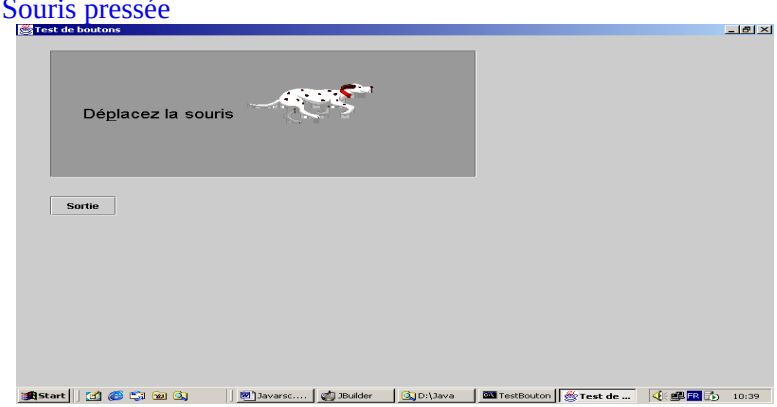
Souris en dehors du premier bouton :



Souris sur le bouton



Souris pressée



Pour commencer avec Java

Exemple commenté d'utilisation de JButton

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

public class TestBouton extends JFrame
{
     // Création d'un objet JButton avec icône
    JButton cmdTest        = new JButton();
    ImageIcon cmdDans   = new ImageIcon("images/Ag00043_.gif");
    ImageIcon cmdHors    = new ImageIcon("images/Ag00179_.gif");
    ImageIcon cmdPresse = new ImageIcon("images/Ag00185_.gif");


    // Création d'un bouton avec texte
    JButton cmdExit   = new JButton("Sortie");
    //============================================================
    // entrée dans l'application
    public static void main(String args[])
    {
            new TestBouton();
    }

   // méthode construction de TestBouton
   public TestBouton()
   {
   initGUI();
   WindowListener  jecoute = new WindowAdapter()
         {
           public void windowClosing(WindowEvent e)
           {
                   System.exit(0);
           }
         };
  this.addWindowListener(jecoute);
   }
```

Pour commencer avec Java

```
 public void initGUI()
{
  this.getContentPane().setLayout(null);  // permet de gérer les emplacements
  this.setTitle("Test de boutons");          // titre de la fenêtre
  this.setSize(500, 400);                    //  taille fenêtre largeur, hauteur

  // Création des fontes sur le bouton et du texte apparaissant sur le bouton
  cmdTest.setFont(new Font("Arial", 0, 20));
  cmdTest.setText("Déplacez la souris ");

  // Détermination de l'icône
      //  image apparaissant  quand la souris est en dehors du bouton
  cmdTest.setIcon(cmdHors);
       //  image apparaissant  quand la souris est sur le bouton
  cmdTest.setRolloverIcon(cmdDans);
       //  image apparaissant  quand on clique sur la souris
  cmdTest.setPressedIcon(cmdPresse);

  // Détermination des bordures du bouton
  cmdTest.setMargin(new Insets(2, 2, 2, 2));

  // Ne pas représenter l'état d'activation du bouton
  cmdTest.setFocusPainted(false);

  // Aligner le texte à gauche
  cmdTest.setHorizontalTextPosition(SwingConstants.LEFT);

  // Déterminer un raccourci clavier pour le bouton
  cmdTest.setMnemonic('p');

// Déterminer la taille du bouton
  cmdTest.setBounds(36, 23, 450, 200);      // marge gauche, marge haute, largeur, hauteur
  cmdExit.setBounds(36,253, 70, 30);

  // Ajouter le bouton au frame de façon visisble
  this.setVisible(true);
  // ajout du composant sur le ContentPane
  this.getContentPane().add(cmdTest);
  this.getContentPane().add(cmdExit);

  //  Ajout d'un écouteur d'événement sur le bouton de sortie
  cmdExit.addActionListener(new bEcoute());
}
```

Pour commencer avec Java

```
// ============================================================
//
//   G e s t i o n   d e s  é v è n e m e n t s  c o m p o s a n t s
//
  class bEcoute implements ActionListener
  {
        public void actionPerformed(ActionEvent evt)
        {
          //  récupération du composant qui a envoyé l'événement
          Object srcAction = evt.getSource();


          if (srcAction == cmdExit)
                {
                //  traitement du bouton de sortie
                   System.exit(0);
                }
        }
  }
}
```

Nous avons vu ici, une manière de traiter les évènements. Dans les exemples qui suivent, nous utiliserons une autre méthode.

A vous de choisir !

Par contre, nous avons vu apparaître la notion de ContentPane : what is it ?

Il faut se représenter le GUI comme un mille feuille.

Première couche : celle du dessous ➔ c'est le JFrame container de haut niveau.

Deuxième couche juste au-dessus ➔ le ContentPane

Couche suivante ➔ les composants.

Pourquoi ne pas directement poser les composants sur le JFrame : Raisonnons par analogie : peut-être avez-vous eu une grand-mère qui vous confectionnait des gâteaux pour vous goûter? Rappelez-vous son moule à tarte, il était en deux parties : le tour et le fond qui était amovible. Le JFrame est le tour, et le ContentPane est le fond. Si vous déposiez la pâte à gâteau sur le tour, elle se retrouvait par terre; De même, vous devez déposer les composants sur le ContentPane par un

```
this.getContentPane().add(cmdTest);
```

this représente dans notre exemple le JFrame.

Pour commencer avec Java

7.8. JRadioButton

Les boutons d'options ou boutons radio sont toujours plusieurs dans un même groupe mais leur sélection est exclusive : un seul bouton est coché.

Ils font partie d'un ButtonGroup non visible et non dimensionnable créé par .

```
ButtonGroup groupRad1 = new ButtonGroup();
```

Les boutons sont créés par .

```
JRadioButton radioCdi = new JRadioButton();
```

Puis ils sont attachés au groupe de boutons par :

```
groupRad1.add(radioCdi);
```

Les événements possibles :

C'est principalement le clic d'un des boutons. Mais cet événement peut ne pas être déclencheur d'une action immédiate.

Exemple : Sur une saisie de personne, il y a un groupe de boutons radio pour cadre employé ouvrier. C'est lorsque l'événement "cliquer sur le bouton OK" se produit que le bouton radio cliqué sera utilisé.

getSelectedObjects() permet de récupérer le fait qu'un bouton est cliqué.

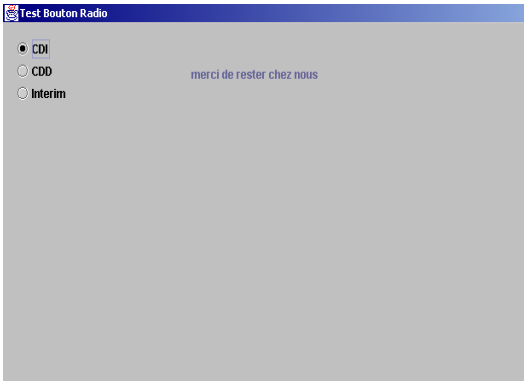
Pour commencer avec Java

L'exemple programme qui suit permet de changer le texte d'un label en fonction de l'action « cliquer sur » un bouton radio.

La sortie du programme se fait par X.

Affichage initiale

Clique sur Cdi



Clique sur Cdd

Clique intérim

Pour commencer avec Java

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

public class TestRadio extends JFrame
{

 JLabel lblResultat = new JLabel("Cochez-moi",SwingConstants.SOUTH_EAST);


 // création des boutons radio
 ButtonGroup groupRad1 = new ButtonGroup();
 JRadioButton radioCdi = new JRadioButton();
 JRadioButton radioCdd = new JRadioButton();
 JRadioButton radioInterim = new JRadioButton();
 Font font1 = new Font("Arial", 1, 20);


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


 public TestRadio() {
 initGUI();
 WindowListener  jEcoute = new WindowAdapter()
       {
         public void windowClosing(WindowEvent e)
         {
                System.exit(0);
         }
       };
  this.addWindowListener(jEcoute);
 }
```

Pour commencer avec Java

```
public void initGUI()
{
  this.getContentPane().setLayout(null);
  this.setTitle("Test Bouton Radio");                          // titre de la fenêtre
  this.setSize(500, 400);                                      // taille de la fenêtre
  this.getContentPane().setBackground(Color.lightGray);  // fond de la fenêtre

  this.getContentPane().add(lblResultat);                      // ajout du label
  lblResultat.setBounds(120,30,250,40);                       // taille du label


  // Initialisation des trois RadioButtons
  radioCdi.setText("CDI");
  radioCdi.setActionCommand("Indéterminé");
  radioCdi.setBounds(17, 14, 91, 22);
  radioCdi.setSelected(true);                                   //  coché par défaut


  radioCdi.addActionListener(                                  // ajout d'un écouteur
        new ActionListener()
        {
          public void actionPerformed(ActionEvent e)
          {
              contratCdi();                                   // action à mener si Cdi
          }
        }               );
  radioCdi.setOpaque(false);

  radioCdd.setText("CDD");
  radioCdd.setActionCommand("Déterminé");
  radioCdd.setBounds(17, 36, 91, 22);
  radioCdd.addActionListener(
        new ActionListener()
        {
          public void actionPerformed(ActionEvent e)
          {
              contratCdd();
          }
        }               );
  radioCdd.setOpaque(false);

  radioInterim.setText("Interim");
  radioInterim.setActionCommand("Interim");
  radioInterim.setBounds(17, 58, 91, 22);
  radioInterim.addActionListener(
        new ActionListener()
        {
          public void actionPerformed(ActionEvent e)
          {
              contratInt();
          }
        }               );
  radioInterim.setOpaque(false);
```

Pour commencer avec Java

```
    // Ajouter les RadioButtons au ButtonGroup
    groupRad1.add(radioCdi);
    groupRad1.add(radioCdd);
    groupRad1.add(radioInterim);

    // Ajouter les RadioButtons de l'objet JFrame
    this.getContentPane().add(radioCdi);
    this.getContentPane().add(radioCdd);
    this.getContentPane().add(radioInterim);

    this.setVisible(true);
  }


  public void contratCdi()
  {
     // envoi texte CDI
     lblResultat.setText("merci de rester chez nous");
     repaint();  // Nouvel affichage du frame
  }


  public void contratCdd()
  {
     // envoi texte CDD
      lblResultat.setText("un petit bout de chemin ensemble");
      repaint();  // Nouvel affichage du frame
  }
  public void contratInt()
  {
     // envoi texte Interim
     lblResultat.setText("merci de passer chez nous");
     repaint();  // Nouvel affichage du frame
  }
}
```

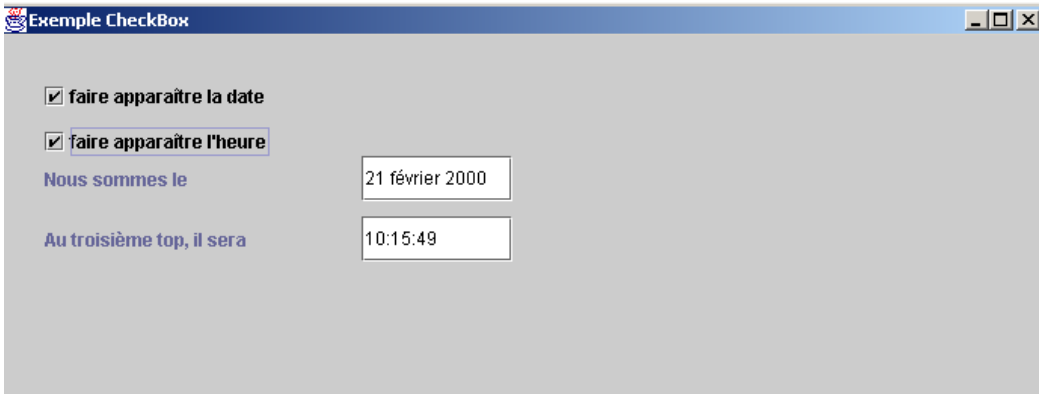
Pour commencer avec Java

7.9. JCheckBox

Une JCheckBox est un bouton qui connaît deux états (coché ou pas coché).

Dans l'exemple qui suit, le fait de cocher « faire apparaître la date » amène l'affichage d'un intitulé suivi d'un JTextfield avec la date en clair.

Même chose pour faire apparaître l'heure.



Pour commencer avec Java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import java.lang.*;
import java.text.*;
import java.util.Locale.*;
import java.sql.Time.*;
import java.sql.Time;


public class TestCheckBox extends JFrame
{
  JCheckBox chkDate = new JCheckBox();
  JCheckBox chkHeure = new JCheckBox();
  JLabel lblDate = new JLabel();
  JLabel lblHeure = new JLabel();
  JTextField txtDate = new  JTextField();
  JTextField txtHeure = new  JTextField();

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

  public TestCheckBox()
  {
    initGUI();
    WindowListener  jEcoute = new WindowAdapter()
          {
             public void windowClosing(WindowEvent e)
             {
                       System.exit(0);
             }
          };
   this.addWindowListener(jEcoute);
  }

  public void initGUI()
  {
    this.getContentPane().setLayout(null);
    this.setTitle("Exemple CheckBox");
    this.setBounds(new Rectangle(50,150,700,275));

    // Mise en forme ckkDate
    chkDate.setBounds(30,30,200,20);
    chkDate.setText("faire apparaître la date");
    chkDate.setActionCommand("date");
```

Pour commencer avec Java

```
chkDate.addActionListener
    (
       new ActionListener()
       {
         public void actionPerformed(ActionEvent e)
         {
            String cmdCheckBox = e.getActionCommand();
            if (cmdCheckBox.equals("date") == true)     // coché
            {
               if (lblDate.isVisible() == false)
               {
                 lblDate.setVisible(true);
                 txtDate.setVisible(true);
                 repaint();
               }
               else
               {
                 lblDate.setVisible(false);
                 txtDate.setVisible(false);
                 repaint();
               }
            }
         }
       }
    );
// Mise en forme ckkHeure
chkHeure.setBounds(30,60,200,20);
chkHeure.setText("faire apparaître l'heure");
chkHeure.setActionCommand("heure");
chkHeure.addActionListener
    (
       new ActionListener()
       {
         public void actionPerformed(ActionEvent e)
         {
            String cmdCheckBox = e.getActionCommand();
            if (cmdCheckBox.equals("heure") == true)     // coché
            {
               if (lblHeure.isVisible() == false)
               {
                 lblHeure.setVisible(true);
                 txtHeure.setVisible(true);
                 repaint();
               }
               else
               {
                 lblHeure.setVisible(false);
                 txtHeure.setVisible(false);
                 repaint();
               }
            }
          }
       }
    );
```

Pour commencer avec Java

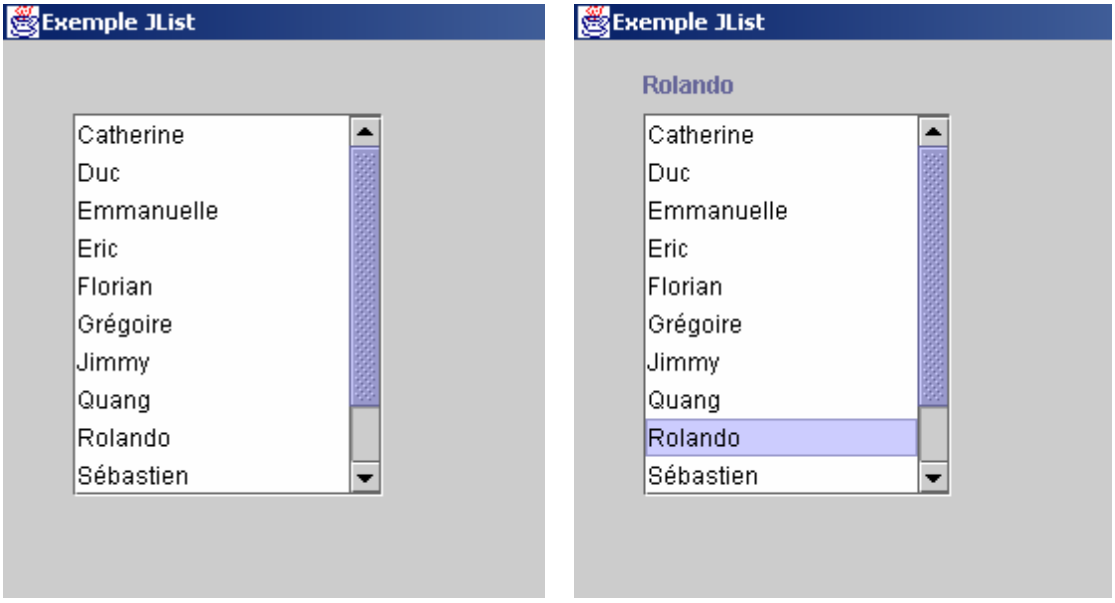
```
    // Initialisation des l'étiquettes
    lblDate.setBounds(30,80,200,30);
    lblDate.setText("Nous sommes le ");
    lblDate.setAlignmentY(RIGHT_ALIGNMENT);
    lblDate.setVisible(false);

    lblHeure.setBounds(30,120,200,30);
    lblHeure.setText("Au troisième top, il sera ");
    lblHeure.setAlignmentX(RIGHT_ALIGNMENT);
    lblHeure.setVisible(false);

    txtDate.setBounds(240,80,100,30);
    txtDate.setVisible(false);

    //  recupération de la date
    DateFormat df = DateFormat.getDateInstance(1);
    Date dateCourante = new Date();
    String strDate = df.format(dateCourante);
    txtDate.setText(strDate);
    txtHeure.setBounds(240,120,100,30);
    txtHeure.setVisible(false);
    // récupération de l'heure
    Time heureCourante = new Time(System.currentTimeMillis());
    // changeTime(HeureCourante);
    heureCourante = Time.valueOf(heureCourante.toString());
    String strHeure = heureCourante.toString();
    txtHeure.setText(strHeure);
    this.getContentPane().add(chkDate);
    this.getContentPane().add(chkHeure);
    this.getContentPane().add(lblDate);
    this.getContentPane().add(txtDate);
    this.getContentPane().add(lblHeure);
    this.getContentPane().add(txtHeure);
    this.setVisible(true);
    repaint();
  }

}
```

Pour commencer avec Java

7.10. JList

Une JList est un composant contenant une série de valeurs parmi lesquelles l'utilisateur peut choisir en double cliquant.

Exemple :
Affichage d'une liste de prénoms avec un Jlabel qui est alimenté par la sélection faite dans la liste




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


public class TestList extends JFrame
{
 JScrollPane scrollAscenseur = new JScrollPane();
 JList listeNom = new JList();
 JLabel lblListe = new JLabel();
  public static void main(String args[])
 {
   new TestList();
 }
 public TestList()
 {
   initGUI();
   WindowListener  jEcoute = new WindowAdapter()
          {
             public void windowClosing(WindowEvent e)
             {
                        System.exit(0);
             }
           };
  this.addWindowListener(jEcoute);
 }
```

Pour commencer avec Java

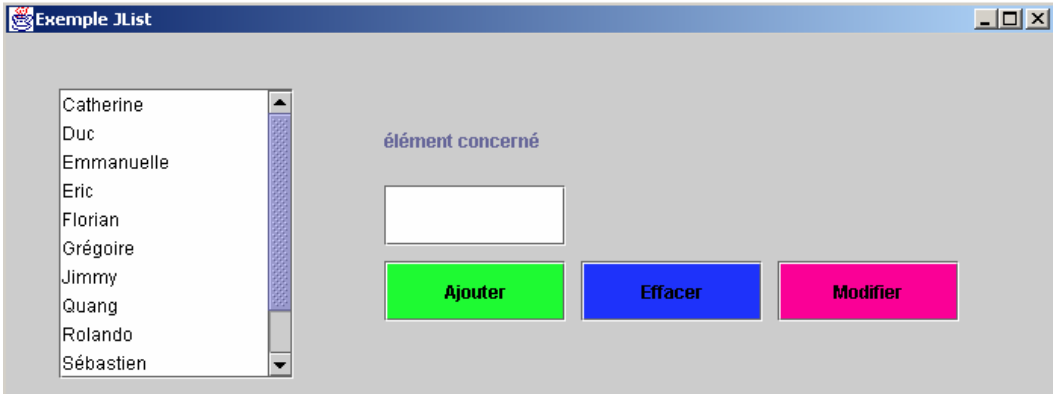
```
  public void initGUI()
  {
    this.getContentPane().setLayout(null);
    this.setTitle("Exemple JList");
    this.setBounds(new Rectangle(150,150,225,275));
    // Taille de la zone de défilement
    scrollAscenseur.setBounds(new Rectangle(35, 36, 155, 192));


    // Préparer les données
     String[] strContenu = {"Catherine","Duc","Emmanuelle","Eric", "Florian","Grégoire",
                    "Jimmy","Quang","Rolando","Sébastien","Stéphane","Yu"};
    // Alimenter la liste
              listeNom.setListData(strContenu);



   // Initialisation de l'étiquette
    lblListe.setBounds(35,14,155,15);
    lblListe.setBackground(new Color(230,230,230));


    // Classe de traitement d'évèmenent utilisée pour la liste
    listeNom.addMouseListener(new MouseAdapter()
    {
      public void mouseClicked(MouseEvent e)
      {
         if (e.getClickCount() == 2)
         {  // après un double-clic
            String sélection = (String)(listeNom.getSelectedValue());
            lblListe.setText(sélection);
         }
      }
    });



    scrollAscenseur.getViewport().add(listeNom);
    this.getContentPane().add(scrollAscenseur);
    this.getContentPane().add(lblListe);

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

Pour commencer avec Java

Dans l'exemple qui précède, il n'est pas possible de modifier le contenu de la liste.
Par contre, par l'ajout des ascenseurs, il est possible de faire défiler la liste si elle dépasse les possibilités d'affichage.

Prenons cette fois-ci, l'exemple d'une liste dite"dynamique", c'est à dire que l'utilisateur peut y ajouter des postes, en supprimer, voir modifier des postes existants. Nous ne nous soucierons pas de l'ordre de la liste.

- Taper le nouveau nom dans le champ texte puis cliquer sur entrée
- Double cliquer sur un nom de la liste, il apparaît dans le champ texte puis cliquer sur effacer, il disparaît de la liste
- Double cliquer sur un nom de la liste, il apparaît dans le champ texte puis modifier le texte et cliquer sur modifier, il disparaît de la liste sous son ancienne orthographe et apparaît dans cette même liste avec sa nouvelle orthographe.



Pour commencer avec Java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TestListDyn extends JFrame
{
  JScrollPane scrollAscenseur = new JScrollPane();
  JList listeNom = new JList();

  DefaultListModel listModel = new DefaultListModel();                    // *******
  JLabel lblListe = new JLabel();
  JLabel lblSaisieListe = new JLabel("élément concerné");
  JTextField txtNouveauPoste = new  JTextField();
  JTextField txtAvant = new  JTextField();          // pour stocker valeur avant modif
  JButton cmdAjout = new JButton();
  JButton cmdSuppr = new JButton();
  JButton cmdModif = new JButton();
   int index;

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

  public TestListDyn()
  {
    initGUI();
    WindowListener  jEcoute = new WindowAdapter()
          {
             public void windowClosing(WindowEvent e)
             {
                      System.exit(0);
             }
           };
   this.addWindowListener(jEcoute);
  }
  public void initGUI()
  {
    this.getContentPane().setLayout(null);
    this.setTitle("Exemple JList");
    this.setBounds(new Rectangle(50,150,700,275));
    // Taille de la zone de défilement
    scrollAscenseur.setBounds(new Rectangle(35, 36, 155, 192));
    //

    listeNom.setModel(listModel);                                          // *******
    // Préparer les données
     String[] strContenu = {"Catherine","Duc","Emmanuelle","Eric",
                            "Florian","Grégoire","Jimmy",
                             "Quang","Rolando","Sébastien","Stéphane","Yu"};
```

Pour commencer avec Java

```
// Alimenter la liste
for (int i=0;i< strContenu.length; i++)
     listModel.addElement(strContenu[i]);


// Initialisation de l'étiquette
lblListe.setBounds(35,14,155,15);
lblListe.setBackground(new Color(230,230,230));

// Classe de traitement d'évèmenent utilisée pour la liste
listeNom.addMouseListener(new MouseAdapter()
{
 public void mouseClicked(MouseEvent e)
 {
    if (e.getClickCount() == 2)
    {                                                          // après un double-clic
       String selection = (String)(listeNom.getSelectedValue());
       index = listeNom.locationToIndex(e.getPoint());
       txtNouveauPoste.setText(selection);

       txtAvant.setText(selection);                  // permet de conserver la valeur avant modif
    }
 }
});
lblSaisieListe.setBounds(250,50,120,40);
txtNouveauPoste.setBounds(250,100,120,40);

cmdAjout .setText("Ajouter");
cmdAjout.setBackground(new Color(30,250,50));
cmdAjout.setBounds(250, 150, 120,40);

cmdAjout.addActionListener(new ActionListener()
{
 public void actionPerformed(ActionEvent e)
 {
    String strMiseAJour = txtNouveauPoste.getText();
    if (strMiseAJour.length() > 0)
    {
      listModel.addElement(strMiseAJour); // ajout dans la liste
      txtNouveauPoste.setText(" ");
    }
 }
});
```

Pour commencer avec Java

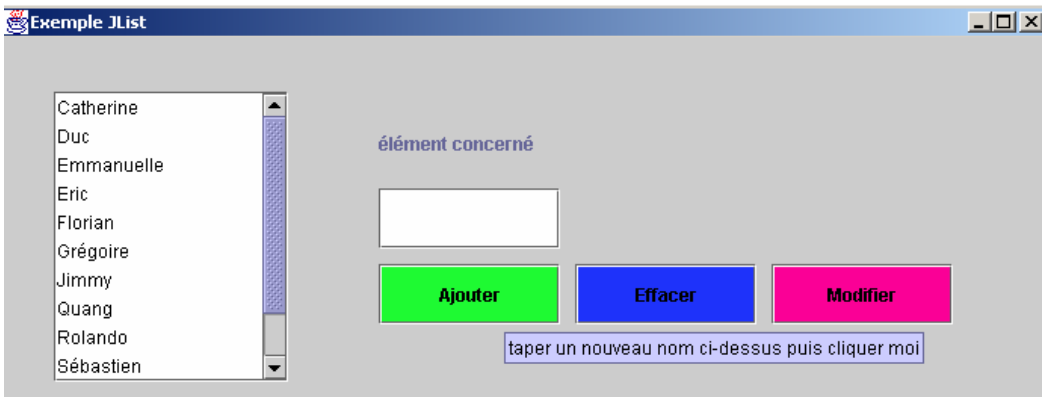
```
    cmdSuppr .setText("Effacer");
    cmdSuppr.setBounds(380, 150,120,40);
    cmdSuppr.setBackground(new Color(30,50,250));
    cmdSuppr.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
         String strMiseAJour = txtNouveauPoste.getText();
         if (strMiseAJour.length() > 0)
         {
            listModel.removeElement(strMiseAJour); // suppression dans la liste
            txtNouveauPoste.setText(" ");
         }
      }
   });

    cmdModif .setText("Modifier");
    cmdModif.setBounds(510, 150, 120,40);
    cmdModif.setBackground(new Color(250,0,150));
    cmdModif.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
         String strMiseAJour = txtNouveauPoste.getText();
         if (strMiseAJour.length() > 0)
         {
             listModel.addElement(strMiseAJour); // ajout dans la liste de l'élément saisi
             txtNouveauPoste.setText(" ");
         }

         strMiseAJour = txtAvant.getText();
         if (strMiseAJour.length() > 0)
         {
             listModel.removeElement(strMiseAJour); // suppression dans liste de l'élément avant modif
         }
      }
   });


    scrollAscenseur.getViewport().add(listeNom);
    this.getContentPane().add(scrollAscenseur);
    this.getContentPane().add(lblSaisieListe);
    this.getContentPane().add(txtNouveauPoste);
    this.getContentPane().add(cmdAjout);
    this.getContentPane().add(cmdSuppr);
    this.getContentPane().add(cmdModif);

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

Pour commencer avec Java

7.11. JToolTip

Ce sont des fenêtres d'aide qu'il est possible de faire apparaître quand la souris passe sur un composant "léger".

Exemple : aide sur le bouton "Ajouter"




```
cmdAjout .setText("Ajouter");
 String strAideAjout = "taper un nouveau nom ";
 strAideAjout = strAideAjout + "ci-dessus puis cliquer moi";
 cmdAjout.setToolTipText(strAideAjout);
```

Pour commencer avec Java

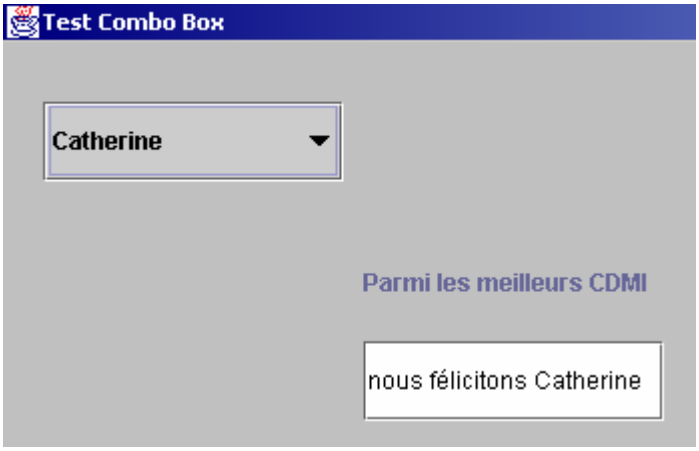
7.12. JComboBox

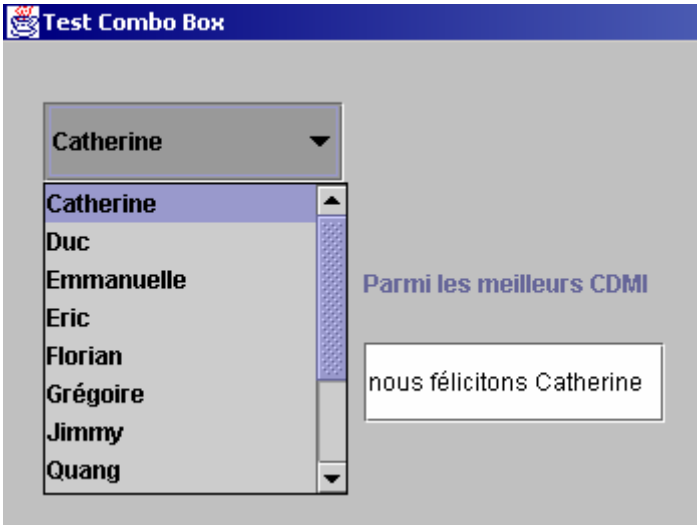
Une JComboBox est une combinaison d'une zone de texte et d'une liste déroulante.

Il est possible d'avoir une liste fermée ou une liste où les saisies sont permises.

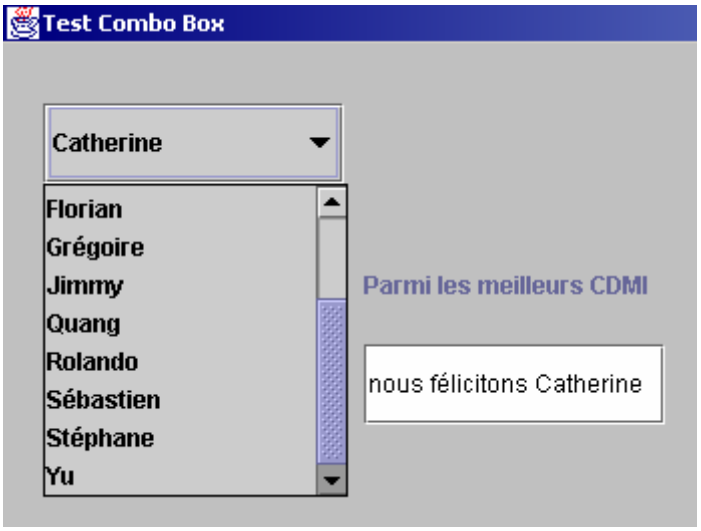
Exemple de programme avec JComboBox sans saisie possible:

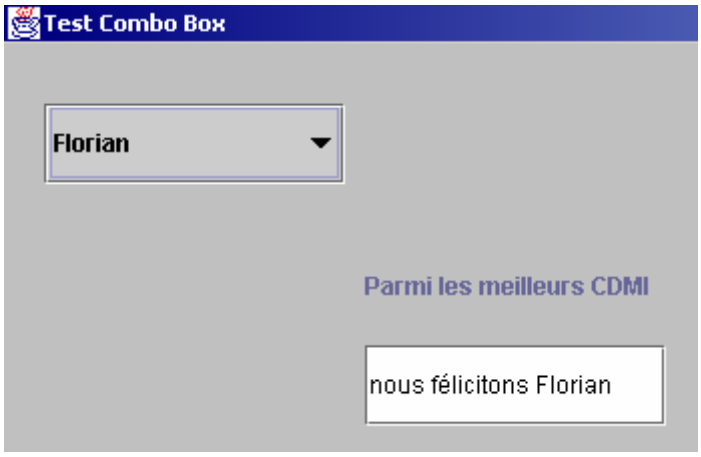
Affichage d'une JComboBox, d'un texte (JLabel) 'parmi les meilleurs CDMI » et d'un JTextField où est indiqué « nous félicitons « suivi du prénom sélectionné dans la JComboBox
En cliquant sur τ de la JComboBox, apparition d'une liste de prénoms
En cliquant sur τ du bas de la liste, défilement des prénoms suivants
En cliquant sur un prénom de la liste, le message du JTextField change.





Pour commencer avec Java





Pour commencer avec Java

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


public class TestCombo extends JFrame
{
 JLabel lblFelicitation       = new JLabel("Parmi les meilleurs CDMI");
 JTextField txtFelicitation = new JTextField();
 // création de la comboBox
 JComboBox comboChoix = new JComboBox();


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


 public TestCombo() {
 initGUI();
 WindowListener  jEcoute = new WindowAdapter()
        {
          public void windowClosing(WindowEvent e)
          {
                  System.exit(0);
          }
        };
  this.addWindowListener(jEcoute);
 }
 public void initGUI()
 {
  this.getContentPane().setLayout(null);      // paramètrage du Jframe
  this.setTitle("Test Combo Box");
  this.setSize(500, 400);
  this.getContentPane().setBackground(Color.lightGray);


  comboChoix.setBounds(20,30,150,40);       // taille des composants
  lblFelicitation.setBounds(180,100,150,40);
  txtFelicitation.setBounds(180,150,150,40);
```

Pour commencer avec Java

```
                                                    // écouteur d'évènements  sur combo
    comboChoix.addActionListener(
            new ActionListener()
            {
              public void actionPerformed(ActionEvent e)
              {
                  String strChoix = (String)comboChoix.getSelectedItem();
                  txtFelicitation.setText("nous félicitons " + strChoix);
                  repaint();
              }
            }                );
    initCombo();

    // Ajouter les composants à l'objet JFrame
    this.getContentPane().add(lblFelicitation);
    this.getContentPane().add(txtFelicitation);
    this.getContentPane().add(comboChoix);

    this.setVisible(true);
  }

  public void initCombo()
  {
     // Alimentation de la comboBox
     String[] strContenu ={"Catherine","Duc","Emmanuelle","Eric", "Florian",
                        "Grégoire","Jimmy","Quang","Rolando","Sébastien","Stéphane","Yu"};
                        for(int i=0; i < strContenu.length;i++)
                        {
                        comboChoix.insertItemAt(strContenu[i],i);
                        }
       // premier poste considéré comme sélectionné donc apparaissant dans texte de la Combo
       comboChoix.setSelectedIndex(0);
  }
}
```

Pour commencer avec Java

8. Construire une application avec SWING

Dans un premier temps, nous allons créer une classe qui ne contiendra que des constantes.
Nous l'appelons Constantes mais ce n'est qu'une convention.

Parmi les constantes, nous trouverons :

- Les informations liées aux drivers,
- Les informations liées à la charte graphique,
- Toute valeur constante nécessaire à l'application.

Exemple de classe Constantes

```
/*                  Constantes diverses
          ================================
          Auteur ..........: Nadine
          Date création ....: 10 mars 2000
          Date modification : 28 novembre 2000
                                        */
public interface Constantes
{
  final String strDriverStatic  = "sun.jdbc.odbc.JdbcOdbcDriver";
  final String strDriverDynamic = "com.inet.tds.TdsDriver";

  final String strTitreApplication  = "Application d'exemples";
  final int nHauteurBtn = 10;
  final int nLargeurBtn = 30;
}
```

Pourquoi créer une telle classe ?
Au lieu de faire un setBound d'un bouton avec des valeurs en dur

```
cmdTest.setBounds(36, 23, 450, 200);          // marge gauche, marge haute, largeur, hauteur
```

nous indiquerons dans la méthode initGui

```
cmdTest.setBounds(36, 23, nHauteurBtn,nLargeurBtn);
```

Avantage : si vous changez de norme graphique et que la taille de vos boutons passe de 10 sur 30 à 15 sur 50, vous n'avez pas à intervenir sur tous les setBound de bouton mais simplement à changer la valeur des constantes "taille de bouton" dans la classe Constantes et à regénérer l'application. Puissant, isn't it ?

Pour commencer avec Java

Autre préliminaire : la classe MesOutils. Comme pour la classe Contantes, le nom n'est qu'une convention vous pourriez la nommer autrement.

Nous y répertorierons les quelques méthodes qui nous aident lors du développement.

Exemple : création d'une méthode AffichMessage

```
import javax.swing.*;

public class MesOutils
{
   public static void AffichMessage(String strMessage)
   {
     JOptionPane.showConfirmDialog(null,strMessage,
                        "trace mise au point",
                        JOptionPane.OK_OPTION,
                        JOptionPane.INFORMATION_MESSAGE);
   }
}
```

Pourquoi cette méthode ?
Dans certains cas, nous n'aurez peut-être pas de déboggueur ou un déboggueur peu pratique d'utilisation. Vous pourrez appeler cette méthode pour afficher une boite de dialogue :

- Qui vous indique que vous passez bien par une séquence d'instructions données :

```
MesOutils.AfficheMessage("je crée l'évènement");
```

- Qui vous permet de sortir les valeurs de certaines variables :

```
MesOutils.AfficheMessage(panel2.strAction);
```

Autres méthodes utiles :

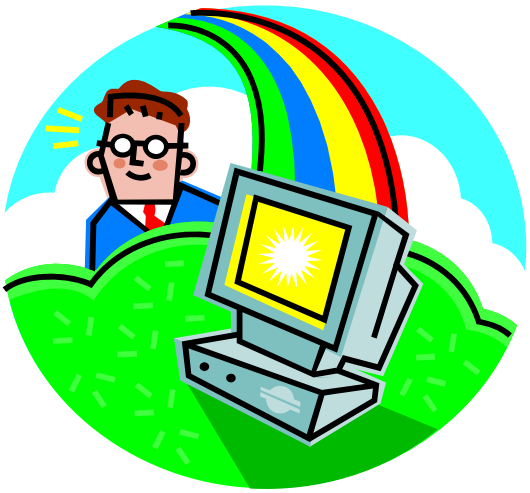
la conversion d'un int en String :

```
   public static String ConcertIntString(int nEntier)
   {
       Integer intConv = new Integer(nEntier),
       String strConvert = intConv.toString();
       return strConvert
   }
```

Pour commencer avec Java

la conversion d'un String en int:

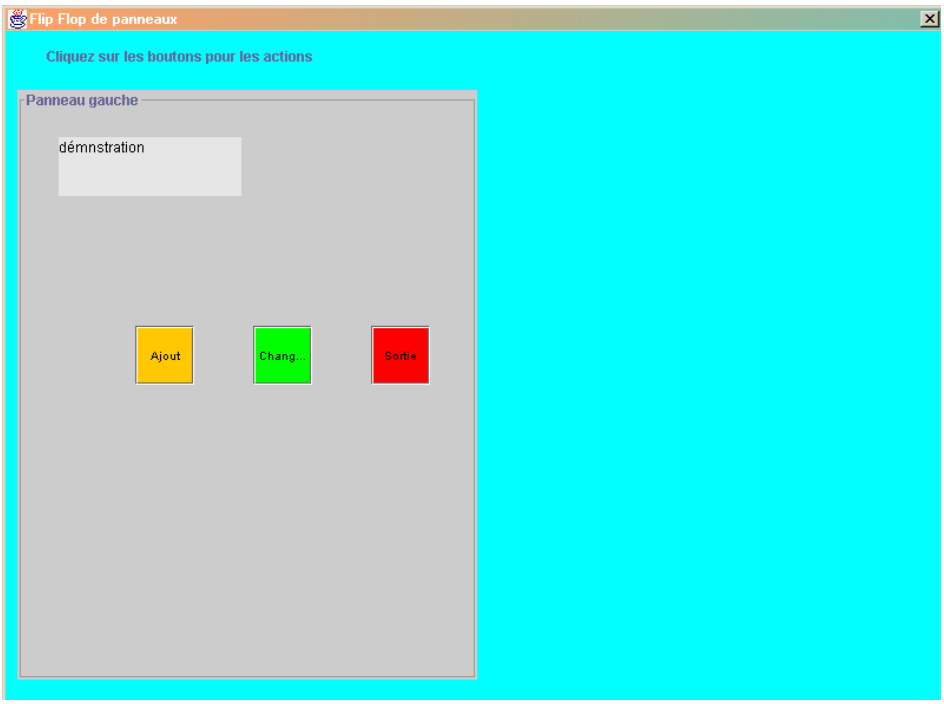
```
public static int concertStringInt(String strEntier)
{
    Integer intConv = new Integer(strEntier);
    Int nConv = intConv.intValue(),
    return nConv;
}
```



Pour commencer avec Java

Exemple d'application :

Lorsque nous lançons l'application, il y a affichage d'un JFrame avec un composant PanneauGauche.



Si nous cliquons sur le bouton Ajout, l'objet de la classe PanneauGauche émet un événement traité dans la classe Jframe qui provoquera l'affichage d'un objet PanneauDroit.

Pour commencer avec Java

8.1. JFrame

Voici le classe CtlGestion qui permet d'afficher et gérer la Jframe.

```
1.  import java.io.*;
2.  import java.awt.*;
3.  import java.awt.event.*;
4.  //import javax.swing.event.*;
5.  import javax.swing.*;
6.
7.  public class CtlGestion extends JFrame
8.  {
9.    JLabel lblParam  = new JLabel();
10.   PanelDroit  panel1;
11.   PanelGauche panel2;
12.
13.    //============================================================
14.    public static void main(String args[]) throws IOException
15.  {
16.     new CtlGestion();
17.  }
18.
19.  public CtlGestion()
20.  {
21.     initGUI();
22.     WindowListener  jecoute = new WindowAdapter()
23.             {
24.                public void windowClosing(WindowEvent e)
25.                {
26.                        System.exit(0);
27.                }
28.             };
29.    this.addWindowListener(jecoute);
30.  }
31.
```

Pour commencer avec Java

```
32.   public void initGUI()
33.   {
34.     this.getContentPane().setLayout(null);          // pas de mise enpage automatique
35.     this.setResizable(false);                         // pas de changement de taille
36.     this.setTitle("Flip Flop de panneaux");         // titre en haut de JFrame
37.     this.setBounds(new Rectangle(0,0,800,600)); // taille du JFrame
38.
39.     // Initialisation de l'étiquette
40.     lblParam.setBounds(35,14,250,15);
41.     lblParam.setText("Cliquez sur les boutons pour les actions");
42.     this.getContentPane().add(lblParam);
43.     this.getContentPane().setBackground(Color.cyan);
44.
45.     panel2              = new PanelGauche();
46.     WinEcoute panneauEcoute = new WinEcoute(this);
47.     panel2.addPanneauListener(panneauEcoute);
48.
49.     this.getContentPane().add(panel2);
50.     this.setVisible(true);
51.     }
52.
53.   public void affichePanel1(String strATraiter)
54.   {
55.             panel1    = new PanelDroit();
56.             this.getContentPane().add(panel1);
57.             WinEcoute panneauEcoute = new WinEcoute(this);
58.             panel1.addPanneauListener(panneauEcoute);
59.             this.setVisible(true);
60.             repaint();
61.   }
62.  public void effacePanel1()
63.   {
64.             this.getContentPane().remove(panel1);
65.             panel1.setVisible(false);
66.         //     this.setVisible(true);
67.             repaint();
68.   }
```

Pour commencer avec Java

```
69.  }
70.  // ============================================================
71.  //    G e s t i o n   d e s  é v è n e m e n t s  c o m p o s a n t s
72.  //                ===========================
73.  class WinEcoute  implements PanneauListener
74.  {
75.       CtlGestion  frmConteneur;
76.
77.       public WinEcoute(CtlGestion frmEnCours)
78.       {
79.           this.frmConteneur = frmEnCours;
80.                       MesOutils.AfficheMessage("constructeur ecoute ");
81.       }
82.           public void panneauAction(PanneauEvent evt)
83.             {
84.                       MesOutils.AfficheMessage("nom panneau" + evt.getNom());
85.         if (evt.getNom() == "panel2")
86.         {
87.            if (evt.getPanneauAction() == "creat")
88.           {
89.             frmConteneur.affichePanel1("  ");
90.           }
91.         }
92.         if (evt.getNom() == "panel1")
93.         {
94.                       MesOutils.AfficheMessage("début paneau 1");
95.            if (evt.getPanneauAction() == "retour")
96.           {
97.             frmConteneur.effacePanel1();
98.           }
99.         }
100.     }
101.}
```

Pour commencer avec Java

Quelques explications complémentaires s'imposent :

Les lignes 9 à 11 permettent de déclarer des objets de portée globale à l'application décrite.

En ligne 45, nous instancions un objet panel2 de la classe PanneauGauche.

En ligne 46 et 47, nous instancions un auditeur PanneauEcoute que nous attachons à l'objet panel2. Nous verrons juste après comment faire émettre des évènements par l'objet panel2.

De la ligne 53 à 61, nous trouvons une méthode d'affichage d'un objet de la classePanneauDroit. Cette méthode reçoit un paramètre strATraiter

De la ligne 62 à 67, c'est une méthode d'effacement du panneau droit affiché préalablement.

Regardons maintenant la classe WinEcoute :

Nous déclarons un objet frmConteneur de classe CtlGestion

Puis, la méthode constructeur avec en paramètre entrant le Jframe en cours.

La méthode panneauAction va traiter les évènements émis par les différents Jpanel. Pour l'instant nous n'avons intégré que le traitement suite à l'action clic sur le bouton création du panneau gauche et sur le bouton retour du panneau droit.

Pour commencer avec Java

Pour générer des évenements depuis les panneaux gauche et droit, il faut créer deux composants une classe et une interface. Voici tout d'abord la classe:

```
1.  import java.util.*;
2.  /*                                                définition d'un nouvel objet évènement
3.    */
4.  public class PanneauEvent extends EventObject
5.  {
6.     protected String     panneauSource;
7.     protected String     strAction;
8.     protected String     strIdent;
9.     //
10.    // méthode constructeur
11.    //
12.    PanneauEvent(String objComposant, String strA, String strId)
13.    {
14.       super(objComposant);
15.       panneauSource = objComposant;
16.       strAction = strA;
17.       strIdent  = strId;
18.    }
19.    //
20.    // Autres méthodes
21.    //
22.   public String getPanneauAction()
23.   {
24.      return strAction;
25.   }
26.
27.   public String getPanneauIdent()
28.   {
29.      return strIdent;
30.   }
31.     public String getNom()
32.   {
33.      return panneauSource;
34.   }
35. }
```

Pour commencer avec Java

Puis l'interface :

```
1.  import java.util.*;
2.  /**
3.    définition d'un nouvel interface surveillant le nouvel évènement
4.    */
5.  public interface PanneauListener extends EventListener
6.  {
7.     //
8.     // méthode évènementdu panneau
9.     //
10.    public void panneauAction(PanneauEvent evt);
11.
12.
13. }
```

La classe **PanneauEvent** permet de définir les évènements qui seront générés avec dans notre exemple trois attributs :

- Le panneau qui émet l'événement
- L'action à déclencher qui dépend du bouton sur lequel on clique
- L'identifiant sur lequel porte l'action

Les méthodes "get" sont définies pour chaque attribut.

Fonctionnellement, une interface ressemble à une classe abstraite dans laquelle il n'y a pas de code.

Une interface est une déclaration de classe spécialisée qui peut déclarer des constantes et des déclarations de méthodes sans code. L'interface PanneauListener en est une illustration.

Pour commencer avec Java

8.2. Jpanel

Nous allons voir maintenant les classes PanneauDroit et PanneauGauche instanciées dans CtlGestion vu précedamment.

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  import javax.swing.*;
4.  import javax.swing.border.*;
5.  import java.util.*;
6.
7.  public class PanelGauche extends JPanel
8.  {
9.    private String strAction;
10.   private String strIdent;
11.   private Vector listenerList = new Vector();
12.
13.
14.   JTextArea strParam = new JTextArea();
15.   // Création d'un bouton avec texte
16.   JButton btnExit   = new JButton("Sortie");
17.   JButton btnCreat   = new JButton("Ajout");
18.   JButton btnChang  = new JButton("Changement");
19.      //============================================================
20.  //  public static void main(String args[])
21.  // {
22.   //  new PanelGauche();
23.  // }
24.
25.   public PanelGauche()
26.   {
27.    initGauche();
28.    }
29.
30.   public void initGauche()
31.   {
32.    //  format et ajout des composants du GUI
33.    //
```

Pour commencer avec Java

```
34.     this.setLayout(null);
35.     //
36.     this.setBounds(new Rectangle(10,50,390,500));
37.     Border border1 = BorderFactory.createTitledBorder("Panneau gauche");
38.     this.setBorder(border1);
39.     //
40.     strParam.setBounds(35,40,155,50);
41.     strParam.setBackground(new Color(230,230,230));
42.     //
43.     btnExit.setFont(new Font("Arial", 0, 10));
44.     btnExit.setBackground(Color.red);
45.     btnExit.setMargin(new Insets(2, 2, 2, 2));
46.     btnExit.setBounds(300,200,50,50);
47.     //
48.     btnCreat.setFont(new Font("Arial", 0, 10));
49.     btnCreat.setBackground(Color.orange);
50.     btnCreat.setMargin(new Insets(2, 2, 2, 2));
51.     btnCreat.setBounds(100,200,50,50);
52.     //
53.     btnChang.setFont(new Font("Arial", 0, 10));
54.     btnChang.setBackground(Color.green);
55.     btnChang.setMargin(new Insets(2, 2, 2, 2));
56.     btnChang.setBounds(200,200,50,50);
57.
58.     this.add(strParam);
59.     this.add(btnExit);
60.     this.add(btnCreat);
61.     this.add(btnChang);
62.     //
63.     //    mise en place gestion des évènements
64.     //
65.     EcouteGauche evtExit  = new EcouteGauche(this, evtExit.nExit);
66.     EcouteGauche evtCreat = new EcouteGauche(this, evtCreat.nCreat);
67.     EcouteGauche evtChang = new EcouteGauche(this, evtChang.nChang);
68.     btnExit.addActionListener(evtExit);
69.     btnCreat.addActionListener(evtCreat);
70.     btnChang.addActionListener(evtChang);
```

Pour commencer avec Java

```
71.  }
72.
73.  public synchronized void addPanneauListener(PanneauListener panneauEcouteur)
74.  {
75.     listenerList.addElement(panneauEcouteur);
76.  }
77.
78.  public synchronized void removePanneauListener(PanneauListener panneauEcouteur)
79.  {
80.    listenerList.removeElement(panneauEcouteur);
81.  }
82.
83.  protected void processPanneauAction()
84.  {
85.     MesOutils.AfficheMessage("je crée l'évènement");
86.     PanneauEvent evtPanneau = new PanneauEvent("panel2", strAction, strIdent);
87.     for (int nEvt = 0; nEvt < listenerList.size(); nEvt++  )
88.        ((PanneauListener)listenerList.elementAt(nEvt)).panneauAction(evtPanneau);
89.  }
```

Pour commencer avec Java

```
90.  // ============================================================
91.  //   Gestion des évènements composants
92.  //
93.     class EcouteGauche implements ActionListener
94.     {
95.       PanelGauche panelEcoute;
96.       int nbtnClic;
97.       final int nCreat = 0;
98.       final int nChang = 1;
99.       final int nExit  = 2;
100.
101.      public EcouteGauche(PanelGauche panelOui, int nBouton)
102.      {
103.         this.panelEcoute = panelOui;
104.         this.nbtnClic    = nBouton;
105.      }
106.         public void actionPerformed(ActionEvent evt)
107.            {
108.               switch(nbtnClic)
109.         {
110.            case nExit:
111.                                    System.exit(0);
112.              break;
113.                  case nCreat:
114.              strAction = "creat";
115.              strIdent = " ";
116.              panelEcoute.processPanneauAction();
117.              break;
118.                        case nChang:
119.                    strAction = "chang";
120.              strIdent = strParam.getText();
121.              break;
122.                        }
123.     }
124.  }
125.}
```

Pour commencer avec Java

A partir de la ligne 65, nous instancions un objet de la classe EcouteGauche avec deux paramétres :

- Le composant en cours
- Des valeurs constantes qui sont définies dans la classe EcouteGauche. Nous avons fixé arbitrairement 0 pour création, 1 pour changement et 2 pour sortie.

Puis nous attchons à chaque bouton, l'écouteur (l'auditeur) qui lui correspondant.

En ligne 73, vous trouverez la méthode qui permet d'ajouter à la liste des écouteurs notre écouteur spécifique.
En ligne 78, la méthode qui permet l'action inverse.

En ligne 83, la méthode processePanneauAction permet de générer un événement et de l'ajouter à la liste des évènements. C'est cette méthode qui sera invoquée ligne 117 quand il a été détecté une action sur le bouton création du panneau gauche pour générer un événement qui sera récuépré par un écouteur (auditeur) de CtlGestion.

Pour commencer avec Java

Voici une partie du code de la classe gérant le panneau droit :

```
126.import java.awt.*;
127.import java.awt.event.*;
128.import javax.swing.border.*;
129.import javax.swing.*;
130.import java.util.*;
131.
132.public class PanelDroit extends JPanel
133.{
134.  // zones pour gestion d'évènements
135.  private String strAction;
136.  private String strIdent;
137.  private Vector listenerList = new Vector();
138.
139.  // Création d'un bouton avec texte
140.  JLabel lblRecu      = new JLabel();
141.  JButton btnAction   = new JButton("Retour");
142.
143.  public PanelDroit()
144.  {
145.    initDroit();
146.   }
147.
148.  public void initDroit()
149.  {
150.    Border border1 = BorderFactory.createTitledBorder("Panneau droit");
151.    this.setBorder(border1);
152.    this.setLayout(null);
153.    this.setBounds(new Rectangle(400,50,390,500));
154.
155.    this.setBackground(Color.lightGray);
156.    // Initialisation des composants posés sur le Jpanel
157.    lblRecu.setBounds(35,14,155,15);
158.    //
159.    btnAction.setFont(new Font("Arial", 0, 10));
160.    btnAction.setBackground(Color.pink);
161.    btnAction.setMargin(new Insets(2, 2, 2, 2));
162.    btnAction.setBounds(100,200,50,50);
```

Pour commencer avec Java

```
163.
164.    //
165.    this.add(btnAction);
166.    this.add(lblRecu);
167.
168.    EcouteDroit evtRetour  = new EcouteDroit(this);
169.    btnAction.addActionListener(evtRetour);
170.     this.setVisible(true);
171.   }
172.
173.     //  méthodes pour la gestion des évènements panneau
174.
175.  public synchronized void addPanneauListener(PanneauListener panneauEcouteur)
176.  {
177.      listenerList.addElement(panneauEcouteur);
178.  }
179.
180.  public synchronized void removePanneauListener(PanneauListener panneauEcouteur)
181.  {
182.     listenerList.removeElement(panneauEcouteur);
183.  }
184.
185.  protected void processPanneauAction()
186.  {
187.      MesOutils.AfficheMessage("je crée l'évènement 1");
188.      PanneauEvent evtPanneau = new PanneauEvent("panel1", strAction, strIdent);
189.      for (int nEvt = 0; nEvt < listenerList.size(); nEvt++  )
190.          ((PanneauListener)listenerList.elementAt(nEvt)).panneauAction(evtPanneau);
191.  }
192.
193.
194.// ================================================================
195.//    G e s t i o n   d e s   é v è n e m e n t s   c o m p o s a n t s
196.//                ==========================
197.//  class EcouteDroit implements ActionListener
198.  {
199.      PanelDroit panelEcoute;
```

Pour commencer avec Java

```
200.
201.    public EcouteDroit(PanelDroit panelOui)
202.    {
203.       this.panelEcoute = panelOui;
204.    }
205.        public void actionPerformed(ActionEvent evt)
206.          {
207.         //  récupération du composant qui a envoyé l'évènement
208.            Object srcAction = evt.getSource();
209.
210.            if (srcAction == btnAction)
211.                    {
212.                 //  traitement du bouton action
213.                          strAction = "retour";
214.             strIdent = " ";
215.             panelEcoute.processPanneauAction();
216.                    }
217.    }
218. }
219.}
```

A titre d'exercice, compléter le code des deux dernières classes pour gérer les boutons non encore traités.

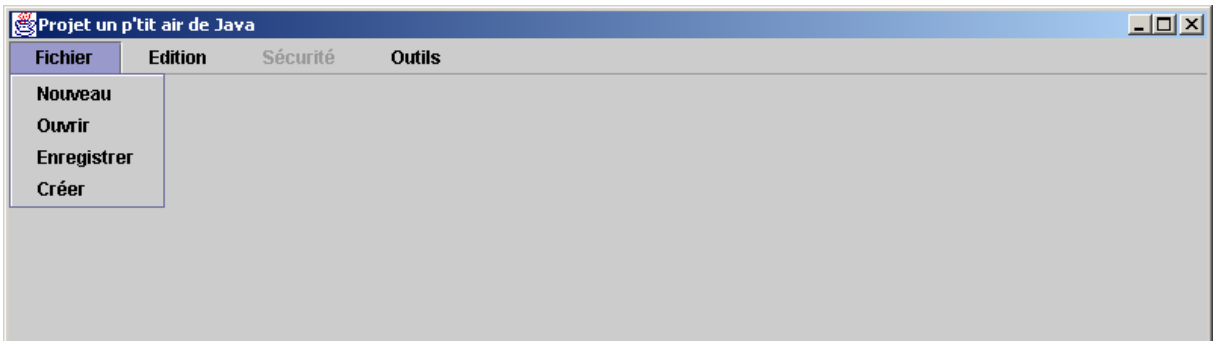
Pour commencer avec Java

8.3. JMenu , JMenuBar, et les entrées de menus

En fin, il nous faut créer des menus facilitant l'accès des utilisateurs aux différentes procédures mises à leur disposition.

Un exemple sera bien mieux qu'un grand discours.

Exemple : affichage d'une barre de menus avec quatre menus ayant chacun des options accessibles ou pas.



Le code est le suivant :

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
/*     Exemple de menu
                                        */
public class AirMenGen extends JFrame
{
  JMenuBar menuBarre    = new JMenuBar();

  JMenu menuFichier     = new JMenu("Fichier");
  JMenu menuEdition     = new JMenu("Edition");
  JMenu menuSecurite    = new JMenu("Sécurité");
  JMenu menuOutils      = new JMenu("Outils");
//  option du menu Fichier
  JMenuItem itemNouveau = new JMenuItem("Nouveau");
  JMenuItem itemOuvrir  = new JMenuItem("Ouvrir");
  JMenuItem itemEnreg   = new JMenuItem("Enregistrer");
  JMenuItem itemCreer   = new JMenuItem("Créer");
//  option du menu Edition
  JMenuItem itemCouper  = new JMenuItem("Couper");
  JMenuItem itemCopier  = new JMenuItem("Copier");
  JMenuItem itemColler  = new JMenuItem("Coller");
//  options du menu Sécurité
  JMenuItem itemSauve   = new JMenuItem("Sauvergarder");
  JMenuItem itemResto   = new JMenuItem("Restaurer");
//  options du menu Outils
  JMenuItem itemPurge   = new JMenuItem("Purge annuelle");
```

Pour commencer avec Java

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

  // constructeur de la fenêtre pincipale
  public AirMenGen()
  {
    initGUI();
    WindowListener  jEcoute = new WindowAdapter()
          {
            public void windowClosing(WindowEvent e)
            {
                    System.exit(0);
            }
          };
   this.addWindowListener(jEcoute);
  }

  public void initGUI()
  {
    this.getContentPane().setLayout(null);
    this.setTitle("Projet un p'tit air de Java");
    this.setBounds(new Rectangle(0,0,Normes.largeur,Normes.hauteur));
    this.setBackground(Normes.couleurFond);

//  options du menu Fichier
    menuFichier.add(itemNouveau);
    menuFichier.add(itemOuvrir);
    menuFichier.add(itemEnreg);
    menuFichier.add(itemCreer);
//  options du menu Edition
    menuEdition .add(itemCouper);
    menuEdition .add(itemCopier);
    menuEdition .add(itemColler);
//  options du menu Sécurité
    menuSecurite.add(itemSauve);
    menuSecurite.add(itemResto);
//  options du menu Outils
    menuSecurite.add(itemPurge);

// invalidation menu ou option
    menuSecurite.setEnabled(false);
    itemResto.setEnabled(false);


// ajout des écouteurs d'évènements
    itemNouveau.addActionListener(new MenuActionAdapter(this));
    itemOuvrir.addActionListener(new MenuActionAdapter(this));
    itemEnreg.addActionListener(new MenuActionAdapter(this));
    itemCreer.addActionListener(new MenuActionAdapter(this));
```

Pour commencer avec Java

```
    itemCouper.addActionListener(new MenuActionAdapter(this));
    itemCopier.addActionListener(new MenuActionAdapter(this));
    itemColler.addActionListener(new MenuActionAdapter(this));
    itemSauve.addActionListener(new MenuActionAdapter(this));
    itemResto.addActionListener(new MenuActionAdapter(this));
    itemPurge.addActionListener(new MenuActionAdapter(this));

    menuBarre.add(menuFichier);
    menuBarre.add(menuEdition);
    menuBarre.add(menuSecurite);
    menuBarre.add(menuOutils);
    this.setJMenuBar(menuBarre);
    this.setVisible(true);
  }
//   méthode traitement des actions sur les options de menus
  void menuTraitement(ActionEvent evtclic)
  {
     String strDemande = evtclic.getActionCommand();
     if(strDemande.equals("Nouveau"))
       System.out.println("Coucou Nouveau");
     else
     if(strDemande.equals("Ouvrir"))
       System.out.println("Coucou Ouvrir");
     else
     if(strDemande.equals("Enregistrer"))
       System.out.println("Coucou Enregistrer");
        else
     if(strDemande.equals("Créer"))
       System.out.println("Coucou Créer");
          else
     if(strDemande.equals("Couper"))
       System.out.println("Coucou Couper");
          else
     if(strDemande.equals("Copier"))
       System.out.println("Coucou Copier");
          else
     if(strDemande.equals("Coller"))
       System.out.println("Coucou Coller");
     //  et la suite
  }
}
// Classe pour mise en écoute sur évènements
class MenuActionAdapter implements ActionListener
{
     AirMenGen obj;                //  donnée de MenuActionAdapter
     MenuActionAdapter(AirMenGen obj)
     {
       this.obj = obj;
     }
     public void actionPerformed(ActionEvent e)
     {
        System.out.println("coucou");
        obj.menuTraitement(e);
     }
}
```

Pour commencer avec Java

8.4. Jwindow

C'est un container c'est à dire un composant de haut niveau. Pour la partie traitement, il est très voisin de Jframe sauf pour l'apparence qui n'est qu'une surface rectangulaire.
Pour sortir d'un Jwindow, il est impératif de prévoir un bouton qui une fois cliqué permettra le System.exit(0)

8.5. Japplet

Mêmes possibilités qu'avec la classe applet vu en début de brochure.

Pour commencer avec Java

9. Utiliser SWING (suite)

9.1. JSlider

Un JSlider est un composant avec graduations sur lequel se trouve un crseur qu'il est possible de déplacer. Les graduations sont de deux types :

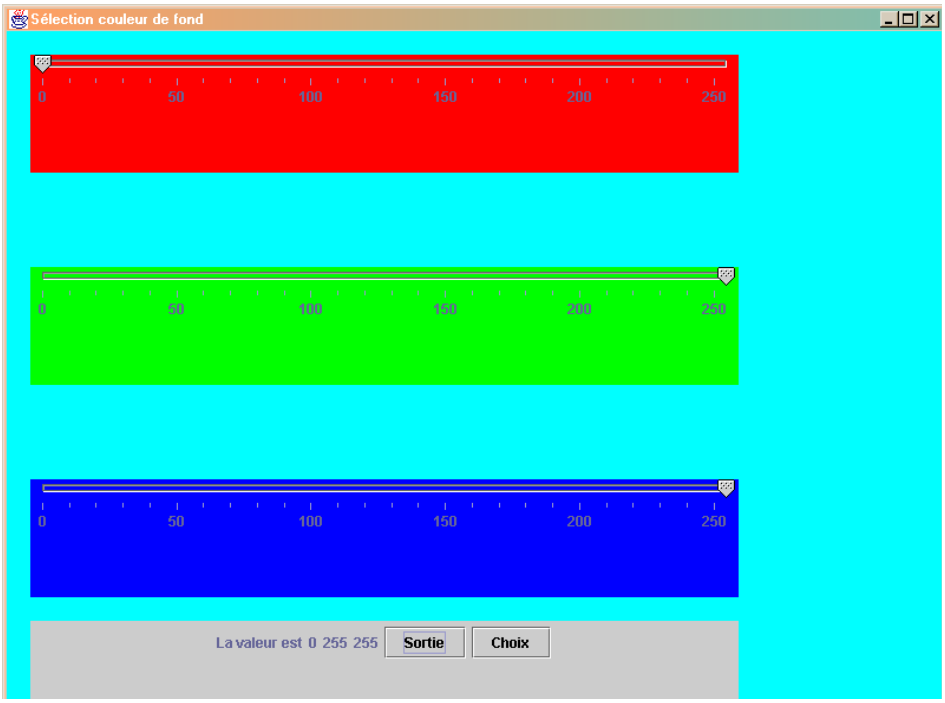
- des valeurs en clair
- des tirets entre deux valeurs

Comme exemple nous vous proposons ce JFrame avec 3 sliders (un pour la couleur rouge, le deuxième pour la couleur verte, le dernier pour la couleur bleue). Pour chacun, la graduation va de 0 à 255.

En faisant varier la position des différents curseurs, nous obtenons un code RGB (red, green blue) permettant de déterminer la couleur du fond du Jframe. Cette valeur est affichée au fur et à mesure des changements de positions des curseurs.

Un bouton Sortie permet de quitter l'application : une boite de dialogue est afficher pour confirmation (voir code page 128).

Un bouton Choix permet 'afficher un objet standard JColorChooser permettant le choix de couleur sur une grille de couleur.




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

Pour commencer avec Java

```
public class JeuCouleur extends JFrame{
  JSlider rougeSlider = new JSlider();
  JSlider vertSlider  = new JSlider();
  JSlider bleuSlider  = new JSlider();
  JPanel panelRouge   = new JPanel();
  JPanel panelVert    = new JPanel();
  JPanel panelBleu    = new JPanel();
  JPanel panelBouton  = new JPanel();

  JColorChooser colorChooser;
  Color couleur1;

  JLabel lblRouge = new JLabel("0");
  JLabel lblVert = new JLabel("255");
  JLabel lblBleu = new JLabel("255");
  JLabel lblRgb = new JLabel("La valeur est");
  JButton btnExit = new JButton("Sortie");
  JButton btnChoix = new JButton("Choix");
// ================================================Point d'entrée
  public static void main(String args[])
  {
   new JeuCouleur();
  }

  public JeuCouleur()
  {
   initGUI();
   WindowListener Jecoute = new WindowAdapter()
   {
    public void windowClosing(WindowEvent evt)
    {
     sortieAppli();
    }
   };
   this.addWindowListener(Jecoute);
  }

  public void initGUI() {
   this.setTitle("Sélection couleur de fond");
   this.setSize(800, 600);
   this.getContentPane().setBackground(new Color(0,255,255));
   panelRouge.setBounds(20, 20, 600, 100);
   panelRouge.setLayout(new BorderLayout());
   panelVert.setBounds(20, 200, 600, 100);
   panelVert.setLayout(new BorderLayout());
   panelBleu.setBounds(20, 380, 600, 100);
   panelBleu.setLayout(new BorderLayout());
   panelBouton.setBounds(20, 500, 600, 100);
   panelBleu.setLayout(new GridLayout());

   JLabel lblRgb = new JLabel("La valeur est");

   rougeSlider.setOrientation(JSlider.HORIZONTAL);
   vertSlider.setOrientation(JSlider.HORIZONTAL);
```

Pour commencer avec Java

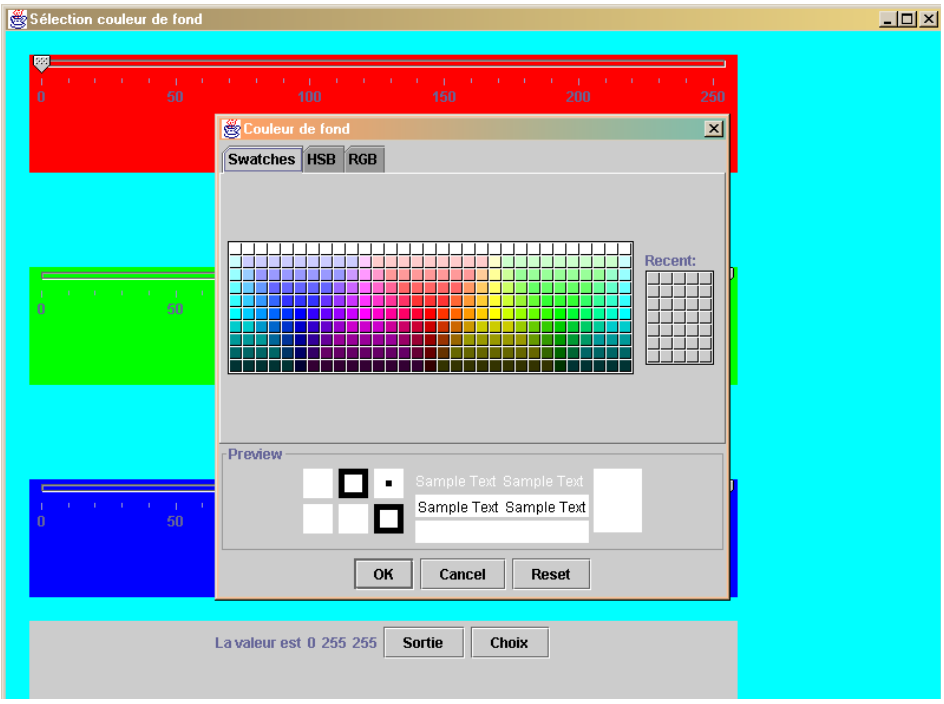
```
bleuSlider.setOrientation(JSlider.HORIZONTAL);

rougeSlider.setMaximum(255);

vertSlider.setMaximum(255);
bleuSlider.setMaximum(255);
                    // valeur de départ sur les différents sliders
rougeSlider.setValue(0);
vertSlider.setValue(255);
bleuSlider.setValue(255);
                     // 50 est la valeur séparant les chiffres
rougeSlider.setMajorTickSpacing(50);
vertSlider.setMajorTickSpacing(50);
bleuSlider.setMajorTickSpacing(50);
                     // 10 est la valeur séparant les tirets
rougeSlider.setMinorTickSpacing(10);
vertSlider.setMinorTickSpacing(10);
bleuSlider.setMinorTickSpacing(10);
                     // définir l'étiquette et les tirets
rougeSlider.setPaintTicks(true);
vertSlider.setPaintTicks(true);
bleuSlider.setPaintTicks(true);

rougeSlider.setPaintLabels(true);
vertSlider.setPaintLabels(true);
bleuSlider.setPaintLabels(true);
                    // liaison entre tirets et la position du slider
rougeSlider.setSnapToTicks(true);
vertSlider.setSnapToTicks(true);
bleuSlider.setSnapToTicks(true);

rougeSlider.setPreferredSize(new Dimension(500,70));
bleuSlider.setPreferredSize(new Dimension(500,70));
vertSlider.setPreferredSize(new Dimension(500,70));

rougeSlider.setBackground(Color.red);
vertSlider.setBackground(Color.green);
bleuSlider.setBackground(Color.blue);

rougeSlider.addChangeListener(new SliderEcoute(this));
vertSlider.addChangeListener(new SliderEcoute(this));
bleuSlider.addChangeListener(new SliderEcoute(this));
lblRgb.setBounds(10,20,30,10);
lblRouge.setBounds(30,20,30,10);
lblVert.setBounds(50,20,30,10);
lblBleu.setBounds(70,20,30,10);
btnExit.setBounds(200,20,20,50);
btnChoix.setBounds(200,20,20,50);
btnExit.addActionListener(new BoutonEcoute(this,1));
btnChoix.addActionListener(new BoutonEcoute(this,2));

this.getContentPane().setLayout(null);
panelRouge.add(rougeSlider);
panelVert.add(vertSlider);
panelBleu.add(bleuSlider);
panelBouton.add(lblRgb);
panelBouton.add(lblRouge);
```

Pour commencer avec Java

```
    panelBouton.add(lblVert);
    panelBouton.add(lblBleu);
    panelBouton.add(btnExit);
    panelBouton.add(btnChoix);

    this.getContentPane().add(panelRouge);
    this.getContentPane().add(panelVert);
    this.getContentPane().add(panelBleu);
    this.getContentPane().add(panelBouton);

    this.setVisible(true);
  }
          // si choix via slider
  public void majCouleur()
  {
    int nRouge = rougeSlider.getValue();
    int nVert  = vertSlider.getValue();
    int nBleu  = bleuSlider.getValue();
                          // conversion int en String
    Integer int1 = new Integer(nRouge);
    Integer int2 = new Integer(nVert);
    Integer int3 = new Integer(nBleu);
    lblRouge.setText(int1.toString());
    lblVert.setText(int2.toString());
    lblBleu.setText(int3.toString());
    this.getContentPane().setBackground(new Color(nRouge,nVert,nBleu));
  }
           // si choix pour colorChooser
  public void nouvelleCouleur()
  {
    couleur1 = JColorChooser.showDialog(this, "Couleur de fond",couleur1);
    this.getContentPane().setBackground(couleur1);
  }
  public void sortieAppli()
  {
     DialogConfirm dial1 = new DialogConfirm(this,"Kenavo", false);
     dial1.setVisible(true);
  }

  public void fermeture()
  {
     System.exit(0);
  }
}

//                        auditeur d'évènements sur les sliders
class SliderEcoute implements ChangeListener
{
  JeuCouleur obj;

  public SliderEcoute(JeuCouleur obj)
  {
    this.obj = obj;
  }

  public void stateChanged(ChangeEvent e)
  {
```

Pour commencer avec Java

```
    obj.majCouleur();
  }
}
//                          auditeur d'événements sur les boutons
class BoutonEcoute implements ActionListener
{
 JeuCouleur JC1;
 int nBouton;

 public BoutonEcoute(JeuCouleur obj, int nBtn)
 {
     //  récupération du conteneur et du bouton qui a envoyé l'événement
  this.JC1 = obj;
  this.nBouton = nBtn;
 }

 public void actionPerformed(ActionEvent evt)
 {
  if (nBouton == 1)
                    {
                      JC1.sortieAppli();      //  traitement du bouton de sortie
                    }
   if (nBouton == 2)
                    {
                      JC1.nouvelleCouleur();   //  traitement du bouton de choix
                    }
 }
}
```



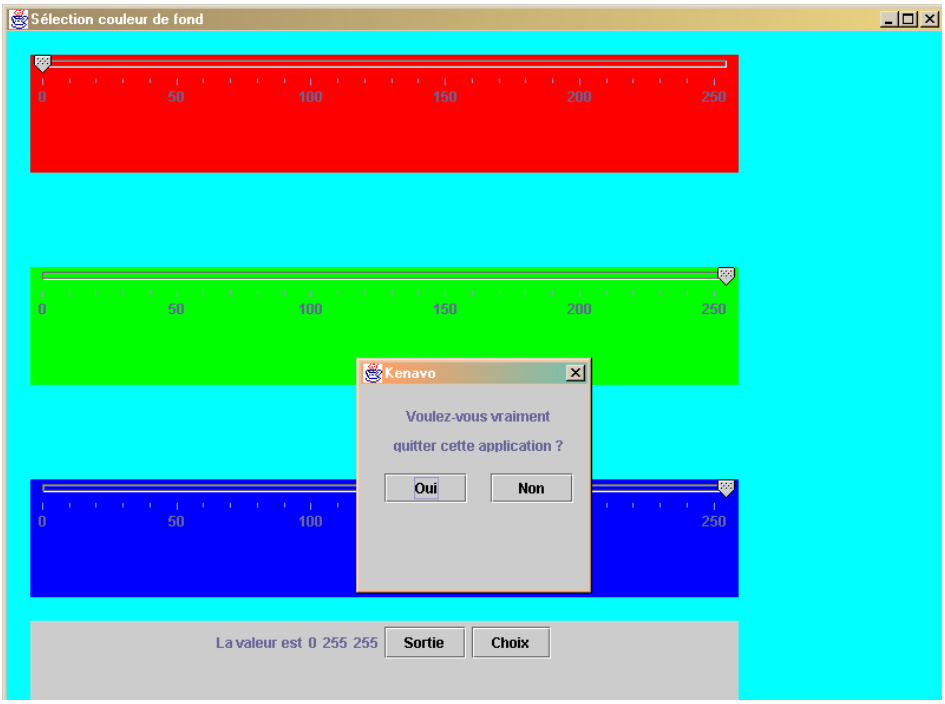
Exemple avec JColorChooser

Pour commencer avec Java

9.2. Jdialog

Un JDialog permet d'afficher une fenêtre avec question – réponse sur la fenêtre d'application (Jframe).

Exemple : boite de dialogue affichée lors de la sortie de l'application précédante.




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

public class DialogConfirm extends JDialog {
 JeuCouleur JeuCouleur;
 JLabel lblDébutPhrase = new JLabel("Voulez-vous vraiment", SwingConstants.CENTER);
 JLabel lblFinPhrase= new JLabel("quitter cette application ?", SwingConstants.CENTER);
 JButton btnOui = new JButton("Oui");
 JButton btnNon = new JButton("Non");

 public DialogConfirm(JeuCouleur JeuCouleur, String titre, boolean modal)
  {
   super(JeuCouleur, titre, modal);
   this.JeuCouleur = JeuCouleur;
   initGUI();
   }
```

Pour commencer avec Java

```
  void initGUI() {
    setBounds(300,300,200,200);
    getContentPane().setLayout(null);
    lblDébutPhrase.setBounds(10,20,180,12);
    lblFinPhrase.setBounds(10,45,180,12);
    btnOui.setBounds(20,75,70,25);
    btnOui.addActionListener(new DialogAction(this));
    btnNon.setBounds(110,75,70,25);
    btnNon.addActionListener(new DialogAction(this));

    getContentPane().add(lblDébutPhrase);
    getContentPane().add(lblFinPhrase);
    getContentPane().add(btnOui);
    getContentPane().add(btnNon);
  }
  // Traitement d'évènements pour les deux boutons
  public void ClicBouton(ActionEvent e) {
    String cmd = e.getActionCommand();
    if("Oui".equals(cmd))
      JeuCouleur.fermeture();     // Appeler la méthode du frame
    else
      dispose();                 // Ne fermer que la boîte de dialogue
  }
}
// Classe d'initialisation pour les deux
class DialogAction implements ActionListener {
  DialogConfirm obj;

  public DialogAction(DialogConfirm obj) {
    this.obj = obj;
  }
  public void actionPerformed(ActionEvent e) {
    obj.ClicBouton(e);
  }
}
```

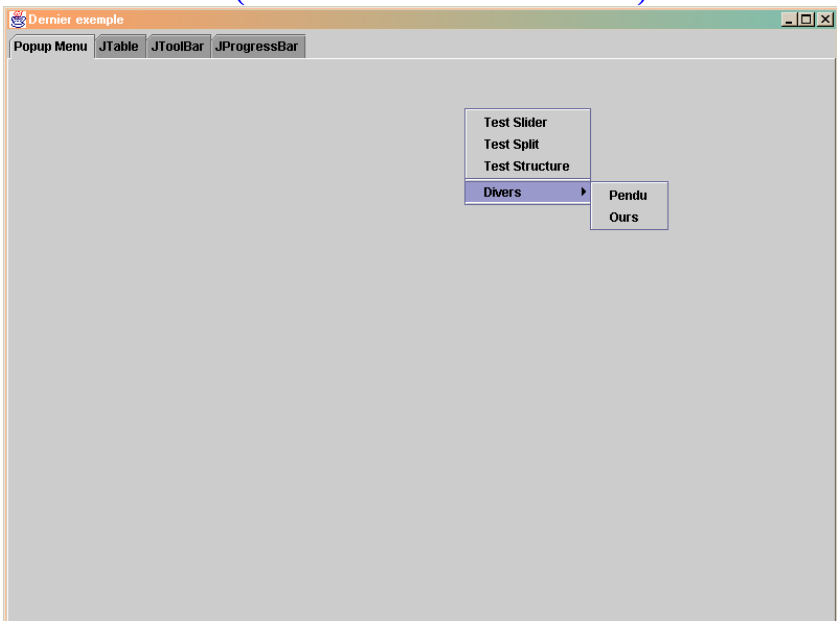
Pour commencer avec Java

9.3. JTabbedPane

Ce composant permet d'afficher plusieurs panneaux qu'il est possible d'atteindre par les "onglets" situés en haut juste sous la barre du JFrame. L'exemple source qui suit utilise ce composant avec 4 "JPanel"

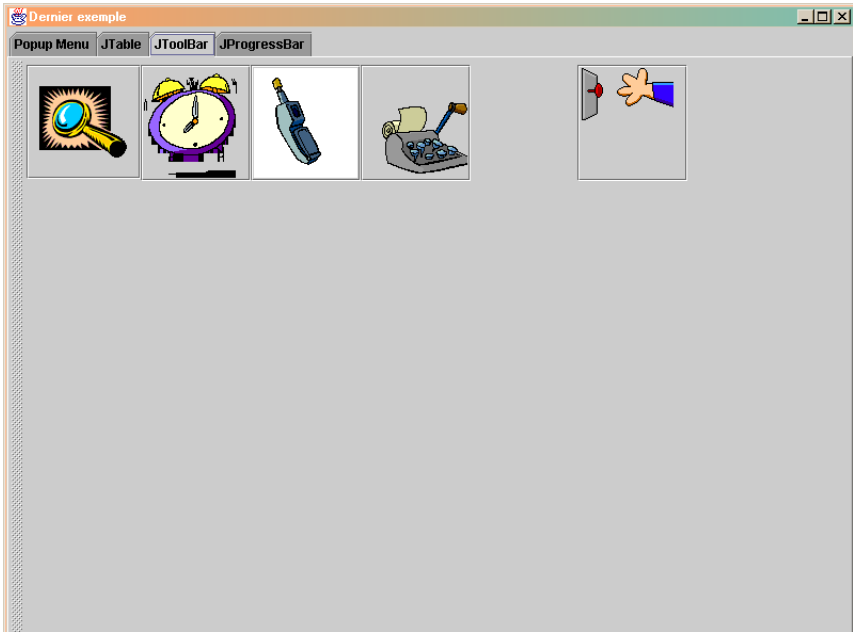
9.4. JPopupMenu

Sur le premier "JPanel", un composant "JPopupMenu" apparaît lorsque nous agissons sur la souris par clic droit. Cet événement est traité dans le source qui suit. Il permet l'apparition (show) d'une fenêtre menu à l'endroit où le clic a été fait (voir commentaires dans le source).



9.5. JToolBar

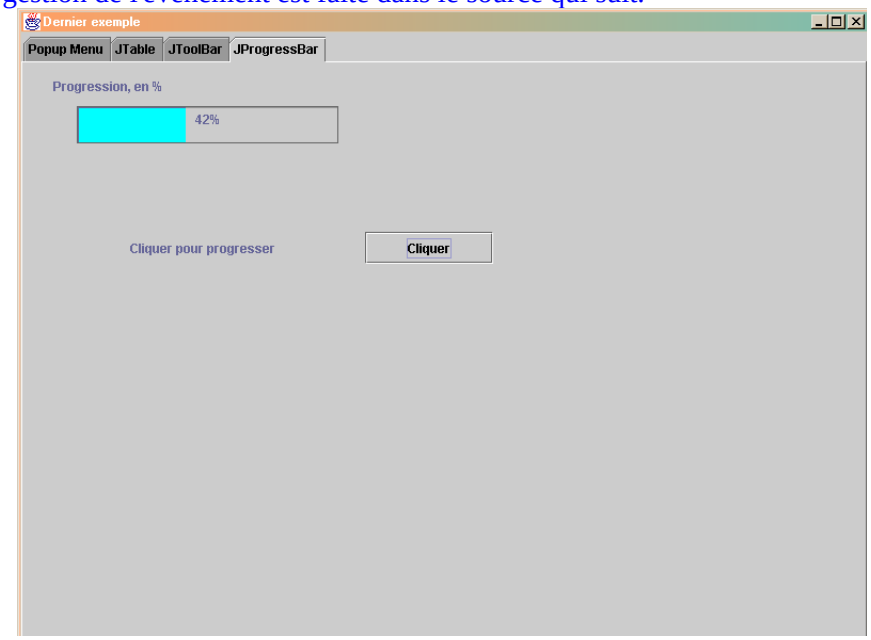
Le deuxiéme "JPanel" affiche une barre d'outils



Pour commencer avec Java

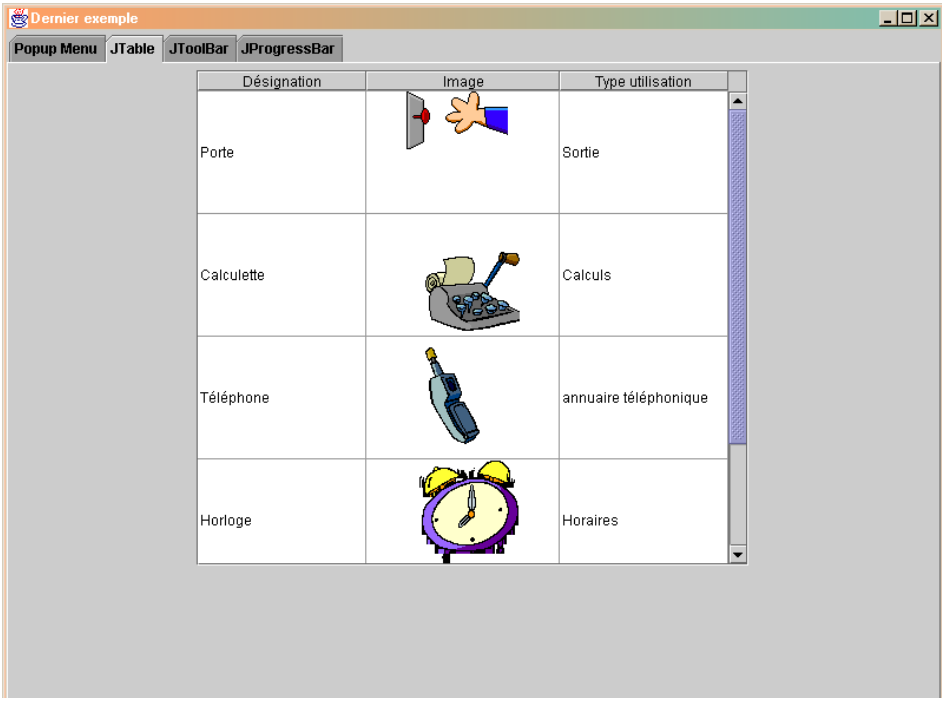
9.6. JProgressBar

Le troisième "JPanel" afiiche une barre de progression à partir du nombre de clic faits sur le bouton "Cliquer". La gestion de l'événement est faite dans le source qui suit.



9.7. JTable

Le dernier Jpanel affiche des renseignements sous forme d'un tableau à deux dimensions. Les éléments à mettre en œuvre sont illustrés dans le source qui suit. Un objet JScrollPane est également mis en œuvre.



Pour commencer avec Java

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

public class Onglet extends JFrame
{
  JTabbedPane paneAonglet = new JTabbedPane();
  JPanel panel1 = new JPanel();
  JPanel panel2 = new JPanel();
  JPanel panel3 = new JPanel();
  JPanel panel4 = new JPanel();
  ImageIcon imgExit = new ImageIcon("d:/Java/images/AG00427.gif", "imgExit");
  ImageIcon imgCalcul = new ImageIcon("d:/Java/images/AG00283.gif", "imgCalcul");
  ImageIcon imgTel = new ImageIcon("d:/Java/images/AG00429.GIF", "imgTel");
  ImageIcon imgHeure = new ImageIcon("d:/Java/images/Ag00215_.gif","imgHeure");
  ImageIcon imgLoupe = new ImageIcon("d:/Java/images/BS00047A.GIF","imgLoupe");

//  composants du panel1
  JPopupMenu menuVolant = new JPopupMenu("Oeuvres diverses");
  JMenuItem item1       = new JMenuItem("Test Slider");
  JMenuItem item2       = new JMenuItem("Test Split");
  JMenuItem item3       = new JMenuItem("Test Structure");
  JMenu     item4       = new JMenu("Divers");
  JMenuItem item41      = new JMenuItem("Pendu");
  JMenuItem item42      = new JMenuItem("Ours");
```

Pour commencer avec Java

```
// composants du panel2
  JTable tblIllust;
  final Object[] enteteCol = {"Désignation", "Image", "Type utilisation"};
  final  Object[][] ligneTable =
          {
          {"Porte", imgExit, "Sortie"},
          {"Calculette", imgCalcul, "Calculs"},
          {"Téléphone", imgTel, "annuaire téléphonique"},
          {"Horloge", imgHeure, "Horaires"},
          {"Loupe", imgLoupe, "Agrandissement"},
          };

//  composants et variable du panel3
  JToolBar barOutils = new JToolBar();
  JButton btnExit    = new JButton();
  JButton btnCalcul  = new JButton();
  JButton btnTel     = new JButton();
  JButton btnHeure   = new JButton();
  JButton btnLoupe   = new JButton();

//  composants et variable du panel4
  JLabel lblPanel4 = new JLabel();
  JButton btnPanel4 = new JButton();
  int nFois = 0;
  JProgressBar progressBar = new JProgressBar();
  JLabel progressLabel = new JLabel();

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

  public Onglet() {
    initGUI();
  }
```

Pour commencer avec Java

```
public void initGUI() {
  this.setTitle("Dernier exemple");
  this.setSize(800,600);

  this.addMouseListener(new Mecoute(this));
   //  initialisation PopupMenu   sur panel1
   panel1.setLayout(null);
   menuVolant.add(item1);
   menuVolant.add(item2);
   menuVolant.add(item3);
   menuVolant.addSeparator();
   menuVolant.add(item4);
   item4.add(item41);
   item4.add(item42);
   item1.addActionListener(new Mecoute(this));
   item2.addActionListener(new Mecoute(this));
   item3.addActionListener(new Mecoute(this));
   item41.addActionListener(new Mecoute(this));
   item42.addActionListener(new Mecoute(this));
   panel1.add(menuVolant,null);
   panel1.addMouseListener(new Mecoute(this));
```

Pour commencer avec Java

```
    // initialisation du panel2 avec une table
TableModel dataModel = new AbstractTableModel()
 {       public int getColumnCount()
         {
            return enteteCol.length;
         }
         public int getRowCount()
         {
            return ligneTable.length;
         }
         public Object getValueAt(int nLigne, int nColonne)
         {  return ligneTable[nLigne][nColonne];
         }
         public String getColumnName(int colonne)
         {  return (String)enteteCol[colonne];
         }
         public Class getColumnClass(int cl)
         {  return getValueAt(0, cl).getClass();
         }
  };
 tblIllust = new JTable(dataModel);
 JScrollPane ascenseur = new JScrollPane(tblIllust);
 // Création d'un objet CellRenderer
  DefaultTableCellRenderer misenFormeCellule = new DefaultTableCellRenderer();
 // Orientation du texte pour l'objet Render
 misenFormeCellule.setHorizontalAlignment(SwingConstants.CENTER);
 int nLargColonne = imgExit.getIconWidth();
 TableColumn colonne = tblIllust.getColumn("Image"); //TableColumn(2,nLargColonne);
 colonne.setPreferredWidth(nLargColonne);              // détermine largeur cellule
 int nHautLigne = imgExit.getIconHeight();    // détermine hauteur cellule
 tblIllust.setRowHeight(nHautLigne);
   //
 panel2.add(ascenseur);
```

Pour commencer avec Java

```
// initialisation du panel3 avec sa ToolBar
panel3.setLayout(new BorderLayout());
JToolBar barOutils = new JToolBar();
btnExit.setIcon(imgExit);
btnCalcul.setIcon(imgCalcul);
btnTel.setIcon(imgTel);
btnHeure.setIcon(imgHeure);
btnLoupe.setIcon(imgLoupe);
btnExit.setToolTipText("c'est l'heure de la sortie");
btnCalcul.setToolTipText("les bons comptes font les bons amis");
btnHeure.setToolTipText("avant l'heure, c'est pas l'heure");
btnTel.setToolTipText("Téléphone");
btnLoupe.setToolTipText("que vois-je ?");
barOutils.add(btnLoupe);          // ajout des boutons à la barre
barOutils.add(btnHeure);          // dans l'ordre d'apparition
barOutils.add(btnTel);
barOutils.add(btnCalcul);
barOutils.addSeparator(new Dimension(100,btnLoupe.HEIGHT));
barOutils.add(btnExit);
panel3.add(barOutils, null);
```

Pour commencer avec Java

```
    // initialisation du panel4 avec sa progressBar
    panel4.setLayout(null);
    progressBar.setMaximum(12);              // valeur maxi de la barre
    progressBar.setForeground(Color.cyan);
    progressBar.setBounds(50, 40, 245, 35);
    progressBar.setStringPainted(true);      // pour afficher le %
    progressLabel.setText("Progression, en %");
    progressLabel.setBounds(new Rectangle(28, 14, 123, 15));
    lblPanel4.setText("Cliquer pour progresser");
    lblPanel4.setBounds(100, 158, 200, 30);
    btnPanel4.setText("Cliquer");
    btnPanel4.setBounds(320, 158, 120, 30);
    btnPanel4.addActionListener(new Mecoute(this));
    panel4.add(progressBar, null);
    panel4.add(progressLabel, null);
    panel4.add(lblPanel4, null);
    panel4.add(btnPanel4, null);
    // Ajouter les objets JTabbedPane aux objets JPanel
    paneAonglet.addTab("Popup Menu", panel1);
    paneAonglet.addTab("JTable", panel2);
    paneAonglet.addTab("JToolBar", panel3);
    paneAonglet.addTab("JProgressBar", panel4);
    this.getContentPane().add(paneAonglet, BorderLayout.CENTER);
    this.setVisible(true);
  }
//  Fermeture du programme
  protected void processWindowEvent(WindowEvent e)
  {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      System.exit(0);
    }
  }
```

Pour commencer avec Java

```
// action souris
  void actionMulot(MouseEvent e)
  {
    if(e.getModifiers() == MouseEvent.BUTTON3_MASK)
      // affichage du menu à l'endroit de l'évènement e
      menuVolant.show(e.getComponent(),e.getX(), e.getY());
  }
// action bouton
  void actionPanel4(Object objSource)
  {
      if (objSource == btnPanel4)
      {
        nFois = nFois + 1;
        if (nFois < 12)
        {
          progressBar.setValue(nFois);
        }
        else
        {
          lblPanel4.setText("You win");
          nFois = 0;
         }
      }
  }
}
```

Pour commencer avec Java

```
class Mecoute extends MouseAdapter implements ActionListener{
  Onglet obj;
  Mecoute(Onglet obj)
  {
    this.obj = obj;
  }
  public void mouseClicked(MouseEvent e)
  {
    obj.actionMulot(e);
  }
  public void actionPerformed(ActionEvent e)
  {
    Object sourceAction =e.getSource();
        obj.actionPanel4(sourceAction);

  }
}
```

Pour commencer avec Java

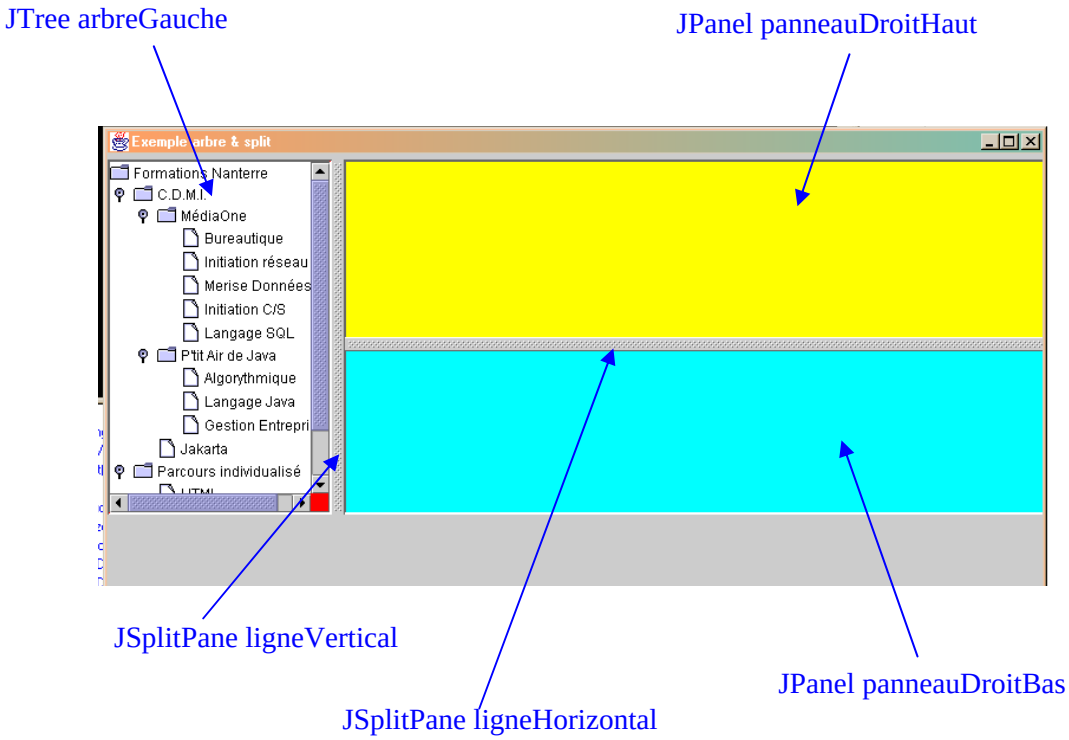
9.8. JTree

Ce type de composant est illustré par les arborescences obtenues lorsque vous utiliser l'explorateur Windows. Vous avez une racine et son premier niveau de décomposition; En cliquant sur un nœud, il est possible de e déplier c'est à dire d'en connaître son contenu et ainsi de suite.

9.9. JSplitPane

Un SplitPane permet de séparer un container en deux parties, la frontière entre ces deux parties pouvant être changée par l'utilisateur à l'aide de la souris en sélectionnant la barre de séparation et en la faisant glisser.

Dans l'exemple qui suit, nous avons mis en œuvre les deux types d'objets.



Pour commencer avec Java

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

public class CtlPane extends JFrame
{
 JTree arbre = new JTree();
 JScrollPane arbreGauche      = new JScrollPane();
 JPanel      panneauDroitHaut = new JPanel();
 JPanel      panneauDroitBas  = new JPanel();
 JSplitPane  ligneHorizontal  = new JSplitPane();
 JSplitPane  ligneVertical    = new JSplitPane();
 JLabel      txtMessage       = new JLabel();
 JLabel      txtBig           = new JLabel();

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

 public CtlPane()
 {
  initGUI();
 }

 private void initGUI() {
  this.getContentPane().setLayout(null);
  this.setTitle("Exemple arbre & split");
  this.setBackground(Color.magenta);
  arbreGauche.setBackground(Color.red);
  panneauDroitHaut.setBackground(Color.yellow);
  panneauDroitBas.setBackground(Color.cyan);
  panneauDroitHaut.setBounds(200,5,600,300);
  panneauDroitBas.setBounds(200,155,600,300);
  txtMessage.setFont(new Font("CASTELLAR",0,20));
```

Pour commencer avec Java

```
    txtMessage.setForeground(Color.orange);
    txtBig.setFont(new Font("CASTELLAR",0,20));
    txtBig.setForeground(Color.black);
    panneauDroitHaut.add(txtMessage);
    panneauDroitBas.add(txtBig);

    // Positionner le Jframe au centre de l'écran quelque soit sa taille
    int nLargeur = 800;
    int nHauteur = 400;
    Dimension TailleEcran = Toolkit.getDefaultToolkit().getScreenSize();
    int nLargEcran = TailleEcran.width;;
    int nHautEcran = TailleEcran.height;
    int nX = (int)(0.5 * (nLargEcran - nLargeur));
    int nY = (int)(0.5 * (nHautEcran - nHauteur));
    this.setSize(nLargeur, nHauteur);
    this.setLocation(nX,nY);

    //  mise en place de l'arbre sur le panneau gauche
    arbreGauche.setBounds(new Rectangle(8, 15, 200, 200));

//  mise en place de la racine
    DefaultMutableTreeNode formations = new DefaultMutableTreeNode("Formations Nanterre");
    DefaultTreeModel arbreModele = new DefaultTreeModel(formations);

    DefaultMutableTreeNode cdmi = new DefaultMutableTreeNode("C.D.M.I.");
    arbreModele.insertNodeInto(cdmi, formations, 0);
    DefaultMutableTreeNode projet1 = new DefaultMutableTreeNode("MédiaOne");
    DefaultMutableTreeNode projet2 = new DefaultMutableTreeNode("P'tit Air de Java");
    DefaultMutableTreeNode projet3 = new DefaultMutableTreeNode("Jakarta");
    cdmi.add(projet1);
    cdmi.add(projet2);
    cdmi.add(projet3);
    DefaultMutableTreeNode projet1a = new DefaultMutableTreeNode("Bureautique");
    DefaultMutableTreeNode projet1b = new DefaultMutableTreeNode("Initiation réseau");
    DefaultMutableTreeNode projet1c = new DefaultMutableTreeNode("Merise Données");
    DefaultMutableTreeNode projet1d = new DefaultMutableTreeNode("Initiation C/S");
    DefaultMutableTreeNode projet1e = new DefaultMutableTreeNode("Langage SQL");
```

Pour commencer avec Java

```
    projet1.add(projet1a);
    projet1.add(projet1b);
    projet1.add(projet1c);
    projet1.add(projet1d);
    projet1.add(projet1e);
    DefaultMutableTreeNode projet2a = new DefaultMutableTreeNode("Algorythmique");
    DefaultMutableTreeNode projet2b = new DefaultMutableTreeNode("Langage Java");
    DefaultMutableTreeNode projet2c = new DefaultMutableTreeNode("Gestion Entreprise");
    projet2.add(projet2a);
    projet2.add(projet2b);
    projet2.add(projet2c);

// ajout deuxième branche
    DefaultMutableTreeNode indiv  = new DefaultMutableTreeNode("Parcours individualisé");
    formations.add(indiv);
// ajout des noeuds de niveau inférieur sur deuxiéme branche
    DefaultMutableTreeNode indiv1 = new DefaultMutableTreeNode("HTML");
    DefaultMutableTreeNode indiv2 = new DefaultMutableTreeNode("JavaScript");
    DefaultMutableTreeNode indiv3 = new DefaultMutableTreeNode("ASP");
    indiv.add(indiv1);
    indiv.add(indiv2);
    indiv.add(indiv3);
// déterminer le modèle à utiliser pour le composant JTree
    arbre.setModel(arbreModele);
// Création et ajout de SelectionListener
    arbre.addTreeSelectionListener(new TEcoute(this));
    arbreGauche.getViewport().add(arbre, null);

//  mise en place des barres de fractionnement sur panneaux droits
     ligneVertical.setBounds(0,5,800,300);
     ligneVertical.setBackground(Color.orange);
     ligneVertical.setOrientation(JSplitPane.HORIZONTAL_SPLIT);
     ligneVertical.setLeftComponent(arbreGauche);
     ligneVertical.setRightComponent(ligneHorizontal);
     ligneVertical.setDividerSize(10);
     ligneVertical.setDividerLocation(200);
```

Pour commencer avec Java

```
    ligneHorizontal.setOrientation(JSplitPane.VERTICAL_SPLIT);
    ligneHorizontal.setTopComponent(panneauDroitHaut);
    ligneHorizontal.setBottomComponent(panneauDroitBas);
    ligneHorizontal.setDividerSize(10);
    ligneHorizontal.setDividerLocation(150);
    this.getContentPane().add(ligneVertical);

    this.setVisible(true);
  }
  // Permettre la fermeture du programme
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      System.exit(0);
    }
  }
// Gestion des évènements pour les composants de l'arbre
  public void treeEvent(TreeSelectionEvent e) {
    DefaultMutableTreeNode node1 = (DefaultMutableTreeNode)e.getPath().getLastPathComponent();
    String strChoix = node1.toString();
    if(node1.isLeaf()) {      // Pour les noeuds terminaux

      if("Bureautique".equals(strChoix))
      txtMessage.setText("Dur Dur la bureautique");
      else if("Initiation réseau".equals(strChoix))
      txtMessage.setText("Dur Dur les collisions");
      else if("Merise Données".equals(strChoix))
      txtMessage.setText("EX CD par MCD");
      else if("Initiation C/S".equals(strChoix))
      txtMessage.setText(" et pour Monsieur ce sera");
        else if("Langage SQL".equals(strChoix))
      txtMessage.setText("SSSSS  QQQQQQQ  LLLLLL");
      else
      txtMessage.setText("Coulé");
    }
    else
      txtMessage.setText("");
```

Pour commencer avec Java

```
    // Redessiner à chaque fois
    txtBig.setText(strChoix);
    txtMessage.repaint();
    txtBig.repaint();
  }
}

// Classe d'initialisation des composants JTree
class TEcoute implements TreeSelectionListener {
  CtlPane obj;

  public TEcoute(CtlPane obj) {
    this.obj = obj;
  }

  public void valueChanged(TreeSelectionEvent e) {
    obj.treeEvent(e);
  }
}
```

Pour commencer avec Java

10.Traitement des exceptions

10.1. Généralités

Nous sommes tous confrontés, en utilisant un micro, à des erreurs liées au matériel ou aux logiciels, qui provoquent une sortie intempestive du logiciel, voir nous obligent à "rebooter" après nous avoir lancer des injures souvent dures à interpréter.

Pour éviter ses débordements, nous disposons de plusieurs atouts :

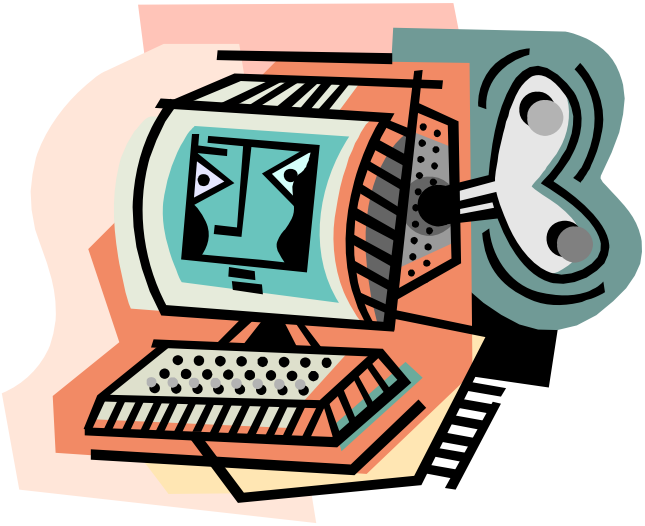
Effectuer tous les tests souhaitables lors d'entrées d'informations dans l'application via saisie écran ou transfert d'informations. Si nous détectons une anomalie, il est de bon ton d'envoyer un message clair à l'émetteur et de lui donner la possibilité de corriger.

Parfois, l'erreur n'est pas liée à une erreur d'entrée d'informations mais à un résultat non conforme à notre attente

- Accès à un objet inexistant
- Division par zéro
- Indice en dehors de ses limites…

En Java , de telles erreurs sont appelées exceptions.

Nous avons donc un certain nombre de classes d'exception. Lorsqu'un incident survient, une erreur est générée ce qui revient à dire qu'une instance de la classe Throwable est créée. Deux sous classes héritent de la classe Throwable : la classe Error et la classe Exception. Si l'erreur provoque la génération d'une instance de la classe Error, cela veut dire que la machine virtuelle Java qui est en cause. Cela est rare mais aucune intervention n'est possible de notre part. Par contre, si l'erreur provoque la génération d'une instance de la classe Exception, c'est le programme qui est en cause et c'est donc à nous de jouer pour "attraper" l'erreur au bond à l'aide de **try** et de mener l'action de correction (message d'erreur clair ou traitement de sécurisation adéquat) à l'aide de **catch**. Ou d'utiliser throws en début de méthode.



Pour commencer avec Java

10.2. Try et catch

```
Instructions non surveillées

        try
        {
                séquences d'instructions à surveiller
                }
        catch (type d'exception traitée)
                {
                séquences d'instructions traitement de l'erreur
}
```

Dans la séquence d'instructions à surveiller, nous avons souvent plusieurs instructions, certaines ne générant jamais d'erreur, d'autres en générant.

Il faut donc prendre instruction par instruction et définir les types d'exception liés à chaque instructions. Il existe une hiérarchie dans les classes d'exception dont voici un aperçu non exhaustif:

```
Exception
    |----------®    ClassNotFoundException
    |----------®    IllegalAcessException
    |----------®    InstantiationException
    |----------®    InterruptedException
    |----------®    NoSuchFieldException
    |----------®    NoSuchMethodException
    |----------®    RunTimeException
    |                   |---------®    ArithmeticException
    |                   |---------®    ArrayStoreException
    |                   |---------®    ClassCastException
    |                   |---------®    IllegalArgumentException
    |                   |---------®    IllegalThreadException
    |                   └--------®     NumerFormatException
    |----------®    IllegalMonitorStateException
    |----------®    IllegalStateException
    |                   |---------®    IndexOutBoundsException
    |                   |                 |----®    ArrayIndexOutOfBoundsException
    |                   |                 └---®     ArrayIndexOutOfBoundsException
    |                   |---------®    NegativeArraySizeException
    |                   |---------®    NullPointerException
    |                   |---------®    SécurityException
    |                   └--------®     UnsupportedOperationException
    └---------
®                   SQLException
```

Pour commencer avec Java

Il est possible de tester les différentes exceptions : il faut simplement partir du niveau le plus fin pour remonter au niveau le plus haut.

```
Instructions non surveillées

try
{
        séquences d'instructions à surveiller
}
catch (FileNotFoundException e)
{
        séquences d'instructions traitement de l'erreur
        fichier non trouvé
}
catch (IOException e)
{
        séquences d'instructions traitement de l'erreur
        sur entrées sorties sauf l'erreur fichier non trouvé
        se trouvant avant donc déjà analysée et traitée
}
```

Pour commencer avec Java

10.3. throws

Autre façon de traiter l'erreur : throws à la place de try et catch.

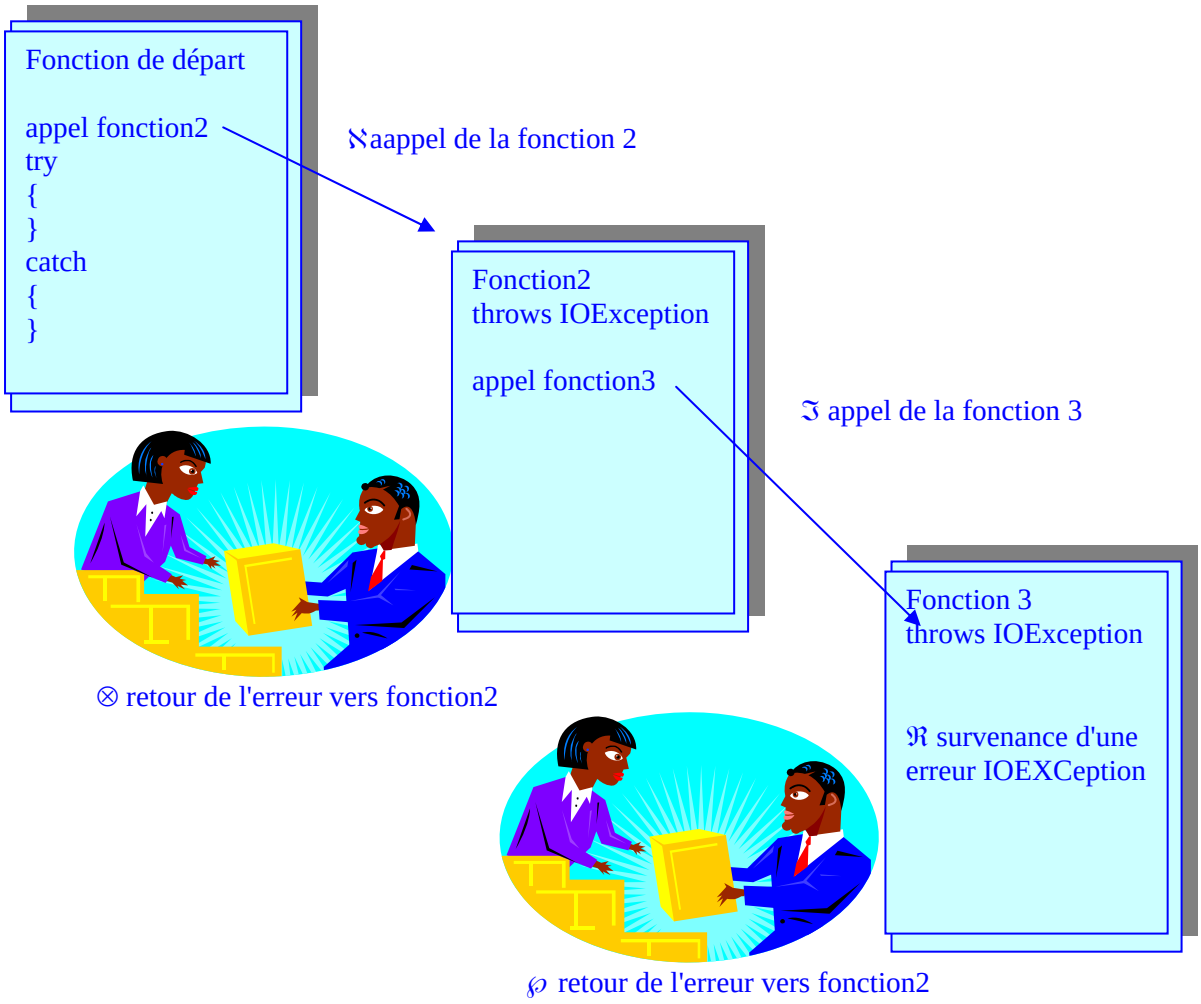
il s'agit de throws avec un s à la fin.

Cette indication se fait sur la déclaration de la méthode.

```
Public static   int   getInteger()   throws IOException, NumberFormatException
```

Il est possible d'indiquer plusieurs exceptions en les séparant par une virgule.

Que ce passe-t-il quand on utilise throws ? les exceptions sont détectées mais ne provoquent pas de message système, elles sont automatiquement retournées à la fonction appelante qui devra soit les traiter par try et catch ou les retourner à la fonction appelante du niveau d'au dessus.



⊕ Interception et traitement de l'erreur par try et catch de fonction de départ.

Si throws est employé dès la fonction de départ, C'est la JVM qui la traitera. Ceci est à éviter.

Pour commencer avec Java

10.4. throw

Cette fois ci sans s à la fin.

"throw" permet de générer ses propres exception dans un programme. J'en entends des qui disent "ils sont fous ces gaulois" et bien non : vous verrez que cela peut être très pratique.

Il effectue un branchement inconditionnel au bloc catch associé à la classe d'exception générée.

```
Exception excMonErreur = new Exception("Mon exception");

throw excMonErreur;
```

Dans le cas d'une génération explicite d'exception vous sortez de la méthode sans valeur de retour.

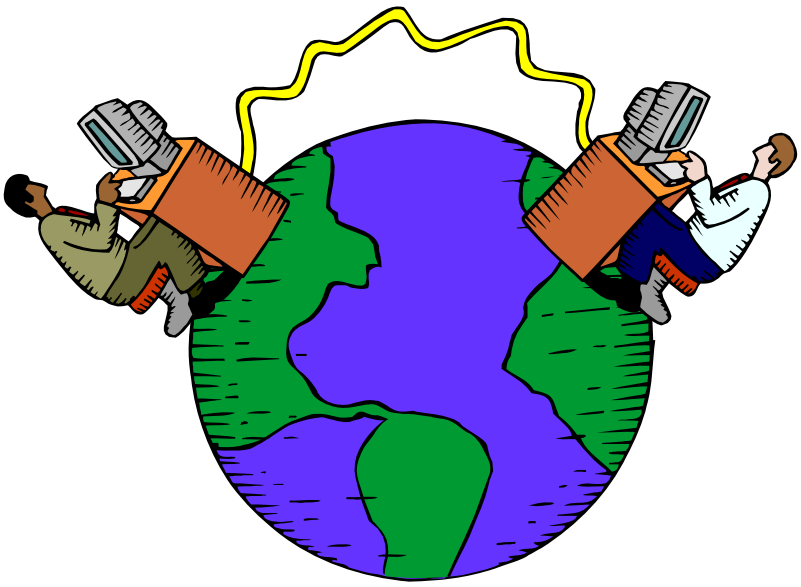
Pour commencer avec Java

11.Accès aux bases de données

11.1. Généralités

Pour pouvoir accéder aux bases de données, nous utiliserons l'API (application Program Interface) nommée JDBC (Java DataBase Connectivity) à rapprocher de ODBC (Open DataBase Connectivity) de Microsoft.

Cette API est constituée de différentes classes et d'interfaces Java, d'un gestionnaire de pilotes et de pilotes. Elle permet d'écrire des applications Java avec tout type d'accès aux bases de données (Access, SQLServer, Oracle, Sybase, …).



Pour commencer avec Java

11.2. Qu'est–ce que JDBC.

11.2.1. Introduction.

JDBC est l'API java pour se connecter à des bases de données hétérogènes. JDBC est conçue pour que le développeur puisse se concentrer au maximum sur son application, et perde le moins d'énergie possible pour traiter des problèmes techniques de liens avec la base de données.

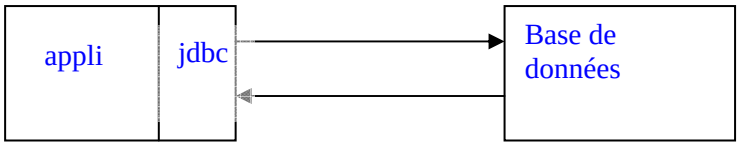
JDBC à trois rôles :

- Se connecter à une base de données avec l'interface adéquat (driver).
- Envoyer des requêtes SQL.
- Exploiter les résultats des requêtes.

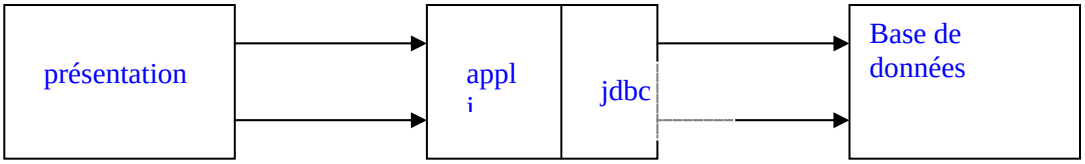
ODBC ne pourrait pas être employé dans java car son interface est en langage C, ODBC n'est pas objet, et son emploi n'est pas très simple. De plus ODBC nécessite d'installer un driver ODBC sur la machine cliente.
Avec JDBC nous avons une solution objet, simple, en pur Java, qui ne nécessite rien de particulier sur la machine cliente (à condition de prendre le driver JDBC qui n'est pas encore adopté par Bill).

11.2.2. Technologies.

Technologie 2/3 ou client-serveur.



Technologie 3/3 ou serveur d'application et serveur de base de données

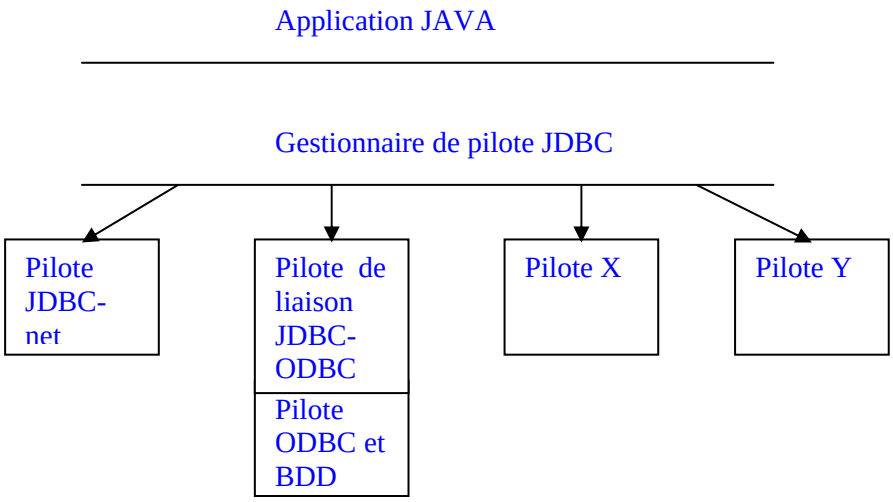


Pour commencer avec Java

L'avantage de la technologie 3/3 est de permettre de sérier les problèmes. Cela permet encore de décharger le client et de le banaliser (explorateur internet), et d'améliorer les performances globales du système (client plus réduit, moins d'informations sur les lignes, …).

JDBC est conforme au standard SQL.

Voici un schéma du fonctionnement de java :



Le pilote JDBC ne nécessite pas d'installation sur la machine cliente, et il est de plus en plus intégré aux bases de données.

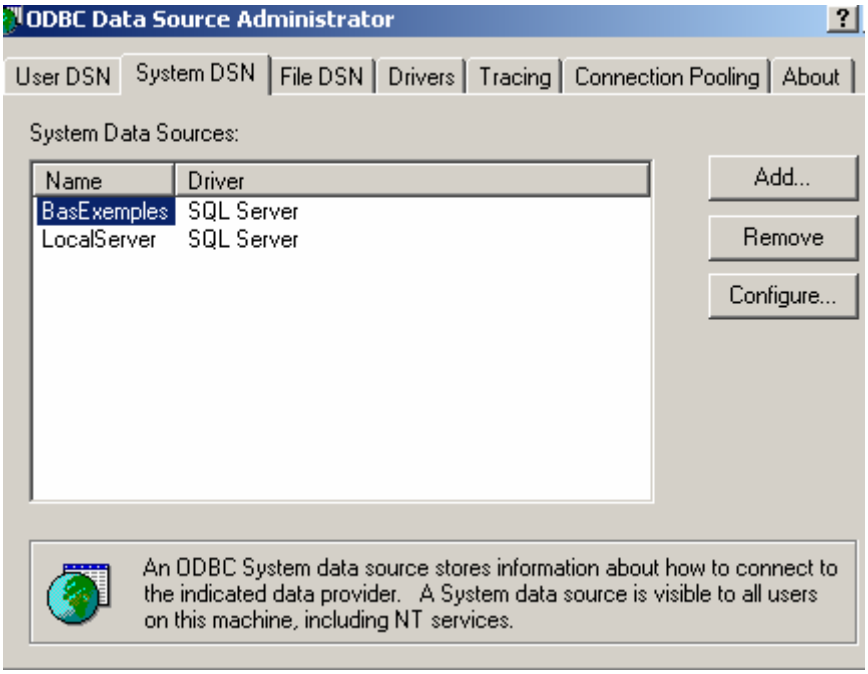
Le pilote ODBC lui doit être installé sur les machines clientes.

Pour commencer avec Java

11.3. La connexion aux bases de données.

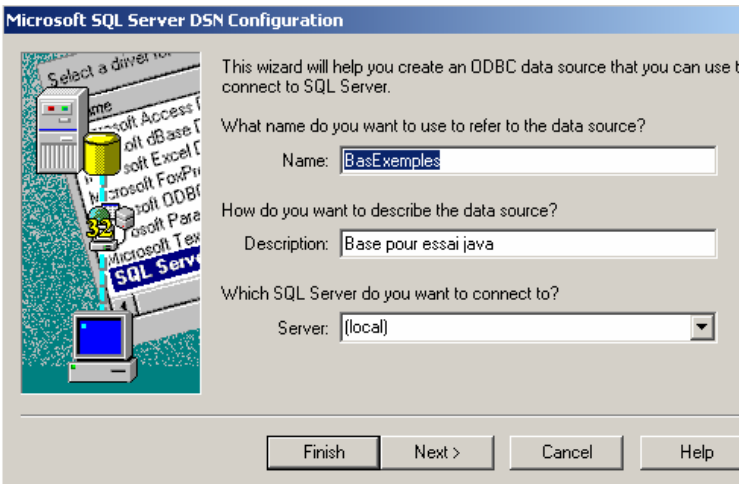
11.3.1. Au niveau base de données

Dans le panneau de configuration de Windows, utilisez l'icône "outils d'administrations" puis "data Source ODBC"



Ajoutez votre base en indiquant son nom et son driver: ici , nous utiliserons la base de données BasExemples en SQL Server.

Puis passez par configuration en indiquant le nom de la base de données et le nom du serveur.



Par Finish, vous obtiendrez un écran avec les options Choisies ainsi qu'une possibilité de test avec leurs résultats.

Pour commencer avec Java

11.3.2. Au niveau programme

La classe DriverManager est le gestionnaire des drivers permettant à l'utilisateur l'accès aux bases de données.

11.3.3. chargement de la classe du Driver désiré.

```
try
{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver") ;       // pour charger le driver ODBC
}
catch ( Exception e)
{
        …
}
```

La classe statique désirée est chargée ; à l'établissement de la connexion le driver sera automatiquement chargé par la fonction getConnection.

11.3.4. Etablissement de la connexion

Par l'administrateur de source de données ODBC vous avez défini un nom logique pour représenter votre base. A ce nom sont associés un pilote, un nom de serveur, … Nous accéderons à la base de données par ce nom logique.

Pour se connecter à la base de données il suffit de demander une connexion à la base au DriverManager en donnant le nom logique de la base.

```
String url = "jdbc:odbc: sebo"; // sebo est le nom logique de la base


Connection cox = null;
try{
        cox = DriverManager.getConnection(url,"user","pswd");
                // demande d'une connexion à la base avec le nom d'user et son mot de passe
}
catch(Exception e)
{
        …
}
```

Pour commencer avec Java

exemple de connexion à une base de données:

```
// connexion à une base de donnée


import java.sql.*; // importation des classes jdbc

public class Class1
{
	public static void main (String[] args)
	{
		try
		{
			Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
			// chargement de la classe du pilote
		}
		catch(Exception e)
		{
			e.printStackTrace ();
		}
		try
		{

			String url ="jdbc:odbc:banque";
			Connection con= DriverManager.getConnection (url, "arrault", "");
				// chargement du pilote et connexion à la base
			System.out.println ("ça marche ");
			System.in.read ();
		}
		catch(Exception e)
		{
			System.out.println ("la connexion a échoué");
			e.printStackTrace ();
		}
	}
```

Pour commencer avec Java

11.4. Passage d'une requête et exploitation des résultats

Ici nous allons passer une requête à la base de données et exploiter les éventuels résultats. Dès que nous avons récupérer une connexion, nous allons pouvoir passer des requêtes à la base. Nous allons pouvoir travailler de trois manières différentes.

- Les requêtes directes à la base (Statement)
- Les requêtes préparées (PreparedStatement). Ce sont des requêtes, souvent paramétrées, qui sont pré compilées par la base, ce qui accélère leurs traitement. Nous utiliserons des requêtes préparées chaque fois qu'une requête doit être passée sur un grand ensemble de données.
- L'appel de procédures stockées. Ces procédures peuvent être paramétrées, avoir un résultat en retour, et nous retourner un ensemble solution (les lignes résultant d'un SELECT par exemple).

Nous verrons également comment exploiter les résultats des requêtes.

11.4.1. Requêtes directes.

Voici un exemple de requête, une fois que nous avons eu une connexion (cox).

```
Statement st = cox.createStatement(); // creation d'un objet requête directe
Resulset Resultat; // création d'une variable qui référencera l'ensemble des résultats


Resultat = stm.executeQuery("SELECT x,y,z FROM table");
                        // exécution de la requête
                        // nous exploiterons plus loin les résultats
int ret = stm.executeUpdate("DELETE …");
```

Il existe trois manières d'exécuter des requêtes SQL.

- **ExecuteQuery** : c'est une interrogation qui produit un ensemble simple de lignes résultat (SELECT).
- **ExecuteUpdate** : c'est la modification de l'état de la base (INSERT, UPDATE, DELETE, CREATE TABLE, DROP TABLE). Le résultat est la modification de 1 ou plusieurs colonnes de 0 à plusieurs lignes de la table. Cela retourne le nombre de lignes modifiées (pour les requêtes CREATE et DROP le résultat est toujours 0).
- **Execute :** est utilisé pour exécuter des requêtes qui retournent plus d'un ensemble de résultat, ou plus d'une valeur de retour, ou les deux. Cette manière d'exécuter des requêtes SQL est assez peu utilisées par les développeurs, nous la passerons donc sous silence.

Pour commencer avec Java

11.4.2. Exploitation des résultats.

Un ResultSet contient en retour d'un executeQuery toutes les lignes qui satisfont les conditions. Reprenons notre code:

```
Resultat = stm.executeQuery("SELECT x,y,z FROM table");          // exécution de la requête
while (Resultat.next())
{
        int i = Resultat.getInt("x");
        String S = Resultat.getString("y");
        Float f = Resultat.getFloat("z");
        System.out.println("ligne = "+i+" "+S+" "+f);
}

stm.close();    // il est recommandé de fermer la Statement même si le garbage collector fait
//le travail quand même.
cox.close(); // idem pour la connexion
```

Le ResultSet à un curseur sur les lignes résultantes.

Initialement ce curseur est positionné avant la première ligne.
La méthode **next** le positionne sur la ligne suivante.

D'autres méthodes existent telles que :

- previous() pour se positionner sur la ligne précédente
- first() pour se positionner sur la première ligne du résultset
- absolute(n) pour se positionner sur la nième ligne du résultset
- …..

Consultez l'aide en ligne de Java pour l'ensemble des possibilités.
Pour lire la ligne nous utilisons les fonctions:

- **getXXX**("nom-de-colonne"); ou XXX est le type de la donnée lue (voir la classe ResultSet) .
- **getXXX**(numcolonne); ou XXX est le type de la donnée lue (voir la classe ResultSet) et numcolonne est le numéro de la colonne concernée (1 pour la colonne la plus à gauche).

Les noms de colonne ne peuvent être utilisés que si ils figurent dans la requête SQL. Si deux colonnes ont le même nom, l'accès par le nom de colonne ne donne que la première des deux. Il est plus efficace de passer par le numéro de colonne, mais il est plus lisible de passer par le nom des colonnes (sauf dans le cas ou le nom est dupliqué).

Ci joint une copie des correspondances entre les types Java et les types JDBC, et un exemple de traitement d'une requête SQL en direct sans résultat, et d'une requête SQL avec un ResultSet.

Pour commencer avec Java

```
import java.sql.*;                              // exemple de requête SQL sans résultat
public class Class1
{
        public static void main (String[] args)
        {
                try
                {
                        //Chargement de la classe du 'Driver' pour etablir la connection ODBC
                        Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
                }
                catch(Exception e)
                {
                        System.out.println ("le pilote n'a pu être chargé");
                }
                try
                {

//établissement de la connexion au lien ODBC 'banque'
//,où l'identificateur de connexion  est "arrault" , et de code ""
                Connection con= DriverManager.getConnection ("jdbc:odbc:banque", "arrault", "");


//création d'un objet de la classe Statement qui permet
//d'effectuer des requêtes liées à la connection 'con'
                Statement select=con.createStatement ();

//appel de la méthode executeUpdate de la classe  Statement qui permet d'écrire dans une base
                select.executeUpdate("INSERT INTO COMPTE(NumCompte, Nom, Prenom, Solde)
                                        VALUES(13, 'Grisolano', 'Philippe', 18000)");
                        select.close();         //il est recommandé de fermer l'objet Statement
                        con.close ();           //et de fermer la connection
                }
                catch(Exception e)
                {
                        e.printStackTrace ();   //ceci permet d'écrire l'exception interceptée
                }
        }
}
```

Pour commencer avec Java

```
import java .sql  .*;                    // exemple de requête SQL avec un résultat

public class Class1
{
        public static void main (String[] args)
        {
                try
                {
                        Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
                }
                catch(Exception e)
                {
                        System.out.println ("le pilote n'a pu être chargé");
                }
                try
                {
                    Connection con= DriverManager.getConnection("jdbc:odbc:banque","arrault","");
                    Statement select=con.createStatement ();

//executeQuery correspond à la demande d'exécution de la  requête. La variable result ( ResultSet )
// est l'ensemble des résultats renvoyés par la requête
                        ResultSet result =select.executeQuery ("SELECT * "+ "FROM COMPTE");

                        //next renvoie true lorsqu'il existe une rangée supplémentaire
                        while(result.next())
                        {
                                //conversion du résultat dans le bon type
                                int Num=result.getInt (1);
                                String Nom=result.getString (2);
                                String Prenom=result.getString (3);
                                String argent=result.getString (4);
                                System.out.println ("Num : "+Num+"    Nom : "+Nom+"
                                        Prenom : "+Prenom+ "    argent : "+argent);
                        }

                        select.close();         //il est recommandé de fermet l'objet Statement
                        con.close ();           //et de fermer la connection
                }
                catch(Exception e)
                {
                        //ceci permet d'écrire l'exception interceptée
                        e.printStackTrace ();

                }
        }
}
```

Pour commencer avec Java

11.4.3. Les requêtes précompilées (ou paramétrées).

Les requêtes précompilées sont des requêtes " à trous ", que le SGBD compile afin de préparer leur exécution . Cela permet d'accélérer leur traitement.
La classe PreparedStatement représente ce type de requête. Une PreparedStatement contient une requête précompilée. Elle a au moins un paramètre en entrée. Ce paramètre est représenté dans la requête par un point d'interrogation '?'.

Avant l'exécution d'une PreparedStatement il faut appeler la fonction setXXX pour chacun de ces paramètres (afin de remplir tous les trous).

Les PreparedStatements sont des requêtes exécutées un grand nombre de fois, qui sont précompilées afin d'en optimiser le traitement.

La classe PreparedStatement hérite de Statement, mais il ne faut pas utiliser les méthodes de la classe mère, mais toujours les méthodes de la classe fille.

Exemple d'utilisation:

```
        // préparation de la requête
PreparedStatement ps = cox.prepareStatement("UPDATE table SET m=? WHERE x=?");

        // garniture des trous avant l'exécution
ps.setString(1,"toto");
ps.setFloat(2,5.0);

        // exécution de la requête
ps.executeUpdate();
```

Pour l'appel suivant nous pouvons redéfinir un ou plusieurs des paramètres, les paramètres non modifiés étant conservés. La fonction clearParameters efface tous les paramètres. Le mode de fonctionnement par défaut est le mode autocommit.

setNull() met un paramètre à null.
setxxx () xxx représente le type Java.

Dans le cas d'un SELECT le traitement du ResultSet retourné est le même que pour une requête directe.
Il est à noter que les points d'interrogation de la PreparedStatement ne remplace que des valeurs de champs de la base. Il ne peuvent pas se substituer à des noms de colonne, ou de table; cela serait l'objet de la définition d'une nouvelle requête. N'oublions pas qu'ici nous traitons des requêtes paramétrées (pour plus de détail voir votre documentation SQL préférée).

Voici un exemple de traitement par une requête précompilée:

Pour commencer avec Java

:

```
                                                    // exemple de requêtes paramétrées
import java .sql  .*; // importation des classes SQL
public class Class1
{
        public static void main (String[] args)
        {
                try
                {
                        Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
                }
                catch(Exception e)
                {
                        System.out.println ("le pilote n'a pu être chargé");
                }
                try
                {
                    Connection con= DriverManager.getConnection ("jdbc:odbc:banque","arrault","");

//Créer une requête prédéfinie par la classe  PreparedStatement sur la connexion con
                        PreparedStatement instruction = con.prepareStatement (
                                "SELECT * "+
                                "FROM COMPTE "  +
                                "WHERE Nom=? ");

//donner la valeur du paramètre pour l'exécution de la requête
                        instruction.setString (1,"perrin" );

                        //exécution de la requête (lecture de la base)
                        ResultSet result =instruction.executeQuery ();

                        while(result.next())       //affichage du résultat
                        {
                                String Nom=result.getString  (2);
                                System.out.println ("Nom : " + Nom);
                        }
                        instruction.close();                // fermeture de la requête

//Créer une requête prédéfinie pour une mise a jour de la  base avec cette fois-ci deux paramètres
                        instruction = con.prepareStatement (
                                "UPDATE COMPTE "+
                                "SET Solde=? "  +
                                "WHERE Nom=? ");

//donner les valeurs manquant à la requête pour chacun des 2 paramètres de la requête paramètrée
                        instruction.setInt (1,122 );
                        instruction.setString (2,"perrin" );

                        //exécuter la requête
                        instruction.execute ();
```

Pour commencer avec Java

```
                    //fermeture de la requête
                    instruction.close ();

                    // fermeture de la connexion
                    con.close ();

                }
                catch(Exception e)
                {
                    //ceci permet d'écrire l'exception interceptée
                    e.printStackTrace ();
                }
        }
}
```

Pour commencer avec Java

11.4.4. Procédures stockées.

Une procédure stockée est une procédure enregistrée sur la base de donnée. La classe CallableStatement est la classe qui permet une requête via une procédure stockée. Elle hérite de PreparedStatement. Un objet CallableStatement contient un appel à une procédure stockée.

Nous trouverons deux formes d'appel d'une procédure stockée.

- Les procédures non paramétrées.
- Les procédures paramétrées.

a. Pour chacun de ces cas il peut y avoir ou pas un code de retour de la procédure stockée

Cela nous donne les quatre interfaces suivantes.
{ call nom_proc }
{ call nom_proc (?, ?, ?) }
{ ?=call nom_proc }
{ ?=call nom_proc (?, ?, ?) }

Voici comment créer une CallableStatement:

```
CallableStatement cs = cox.prepareCall( "{call test ( ?, ? ) }");
```

Avant d'exécuter la requête il faut garnir les paramètres.

- Les paramètres en entrée seront affectés en utilisant la méthode setXXX ou XXX est le type JDBC des données. C'est une conversion entre une donnée Java et une donnée JDBC.
- Les paramètres en sortie seront décrits avant l'appel, par l'appel de la méthode registerOutParameter().

b. setInt(1,7); // le premier paramètre est un entier en entrée. Il vaut 7

```
cs.registerOutParameter(2,java.sql.Types.TINYINT);
 // le paramètre 2 est un Byte en sortie.


cs.executeUpdate(); // ou ResultSet rs = cs.executeQuery();
```

il ne reste plus qu'à récupérer la valeur du paramètre en sortie.

```
byte x = cs.getByte(2);
```

Si la procédure stockée nous retourne un ResultSet, il sera alors traité comme précédemment.

Pour commencer avec Java

Attention!!!

Si nous traitons le code de retour de la procédure stockée ce sera le premier paramètre de l'appel. Il sera traité comme un paramètre en sortie. Il est important de noter que si nous avons un ResultSet et une valeur de retour, ou des paramètres de sortie, il faut d'abord vider le ResultSet avant de récupérer les valeurs de sortie, ou de retour.

Vous trouverez des exemples d'appels des procédures stockées dans les 7 cas suivants:

- Pas de paramètres.
- Un paramètre en entrée.
- Un paramètre en sortie.
- Un Paramètre en entrée/sortie.
- Un ResultSet en résultat de l'exécution de la requête, avec un paramètre en entrée.
- Une valeur de retour de la procédure stockée.
- Une valeur de retour de la procédure stockée avec un ResultSet.

Pour commencer avec Java

/* exemple 1 : procédure stockée non paramétrée codage de la procédure sql */

```
CREATE PROCEDURE phil AS
BEGIN
SELECT *
FROM COMPTE
END
```

Procédure stockée non paramétrée codage du programme java

```
import java.sql.*; // importation des classes jdbc
public class Class1
{
	public static void main (String[] args)
	{
		try
		{
			Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
		}
		catch(Exception e)
		{
			System.out.println ("le pilote n'a pu être chargé");
		}
		try
		{
		    Connection con= DriverManager.getConnection("jdbc:odbc:banque","arrault","");
//Appel d'une procédure stockée(préalablement créée sous  sql Server) sans paramètre
			CallableStatement cs= con.prepareCall ("{call phil}");
			ResultSet result=cs.executeQuery ();	//exécution de la procédure stockée
			while(result.next())		// lecture du résultat
			{
				int Num=result.getInt  (1);
				String Nom=result.getString (2);
				String Prenom=result.getString (3);
				String argent=result.getString(4);
			System.out.println ("Num : "+Num+"  Nom :" +Nom
+"  Prenom : "+Prenom+ " argent : "+argent);
			}
			cs.close ();		// fermeture de le requête et de la connexion
			con.close ();
		}
		catch(Exception e)
		{
			e.printStackTrace ();		//ceci permet d'écrire l'exception interceptée
		}
	}
}
```

Pour commencer avec Java

/* exemple 2 : procédure stockée avec un paramètre en entrée codage de la procédure sql */

```
CREATE PROCEDURE paramin ( @num int)  AS
BEGIN
SELECT *
FROM COMPTE
WHERE NumCompte = @num
END
```

Pour commencer avec Java

/* exemple 2 : procédure stockée avec un paramètre en entrée codage du programme java */

```
import java.sql.*; // importation des classes jdbc
public class Class1
{
	public static void main (String[] args)
	{
		try
		{
			Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
		}
		catch(Exception e)
		{
			System.out.println ("le pilote n'a pu être chargé");
		}
		try
		{
			Connection con= DriverManager.getConnection
					("jdbc:odbc:banque","arrault","");

//Appel d'une procédure stockée(préalablement créée sous sql Server) avec paramètre
			CallableStatement cs= con.prepareCall("{call paramin(?)}");

			// passage du premier paramètre
			cs.setInt(1,4);

			//exécution de la procédure stockée(avec un paramètre)
			ResultSet result=cs.executeQuery ();
		while(result.next())	//récupération des valeurs renvoyées par la procédure
			{
				int Num=result.getInt  (1);
				String Nom=result.getString (2);
				String Prenom=result.getString (3);
				String argent=result.getString(4);
			System.out.println ("Num : "+Num+" Nom : "+Nom+
					" Prenom : "+Prenom+ "  argent : "+argent);
			}
			cs.close ();		// fermeture de la requête et de la connexion
			con.close ();

		}
		catch(Exception e)
		{
			e.printStackTrace ();		//ceci permet d'écrire l'exception interceptée
		}
	}
}
```

Pour commencer avec Java

/* exemple 3 : procédure stockée avec un paramètre en sortie

codage de la procédure sql */

```
CREATE PROCEDURE paramout @solde int OUTPUT
AS
BEGIN
SELECT @solde = MAX(Solde)
FROM COMPTE
WHERE Nom = 'Dumousseau'
END
```

Pour commencer avec Java

```
/* exemple 3 : procédure stockée avec un paramètre en sortie    codage du programme java */
import java.sql.*;       // importation des classes JDBC

public class Class1
{
        public static void main (String[] args)
        {
                try
                {
                        Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
                }
                catch(Exception e)
                {
                        System.out.println ("le pilote n'a pu être chargé");
                }
                try
                {
                    Connection con= DriverManager.getConnection ("jdbc:odbc:banque","arrault","");

                  //Appel d'une procédure stockée avec un paramètre de sortie
                        CallableStatement cs= con.prepareCall ("{call paramout(?)}");

                        //Description du paramètre en sortie
                        cs.registerOutParameter (1,java.sql.Types.INTEGER  );

                        //exécution de la requête
                        cs.executeUpdate ();

                        //récupération du paramètre
                        int Num=cs.getInt (1)  ;

                        //affichage du résultat
                        System.out.println ("Num : "+Num);

                        // fermeture de la requête et de la connexion
                        cs.close ();
                        con.close ();

                }
                catch(Exception e)
                {
                        //ceci permet d'écrire l'exception interceptée
                        e.printStackTrace ();

                }
        }
}
```

Pour commencer avec Java

/* exemple 4 : procédure stockée avec un paramètre en entrée/sortie

codage de la procédure sql */

```
CREATE PROCEDURE paramresult @solde int OUTPUT
AS
BEGIN
SELECT Solde
FROM COMPTE
WHERE NumCompte = @solde
Select @solde = Solde
FROM COMPTE
WHERE Nom = 'Grisolano'
END
```

Pour commencer avec Java

```
/* exemple 4 : procédure stockée avec un paramètre en entrée/sortie  codage du programme java */

import java.sql.*;       // importation des classes jdbc

public class Class1
{
        public static void main (String[] args)
        {
                try
                {
                        Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
                }
                catch(Exception e)
                {
                        System.out.println ("le pilote n'a pu être chargé");
                }
                try
                {
                    Connection con= DriverManager.getConnection ("jdbc:odbc:banque","arrault","");

//instruction d'appel d'une procédure stockée avec un paramètre d'entrée/ sortie
                        CallableStatement cs= con.prepareCall ("{call paraminout(?)}");

                        //passage du paramètre en entrée
                        cs.setFloat  (1,12.0f);//si float

                        //Description du paramètre en sortie ( le même )
                        cs.registerOutParameter (1,java.sql.Types.FLOAT   );

                        //execution de la requete
                        cs.executeQuery();

                        //récupération du paramètre
                        float  Solde=cs.getFloat  (1)  ;

                        //affichage du résultat
                        System.out.println ("Nouveau solde : "+Solde);
                        // fermeture de la requête et de la connexion
                        cs.close ();
                        con.close ();

                }
                catch(Exception e)
                {
                        e.printStackTrace ();  //ceci permet d'écrire l'exception interceptée
                }
        }
}
```

Pour commencer avec Java

/* exemple 5 : procédure stockée avec un paramètre en entrée et un résultat

codage de la procédure sql */

```
CREATE PROCEDURE paramresult @solde varchar(30)
AS
BEGIN
SELECT *
FROM COMPTE
WHERE Nom = @solde
END
```

Pour commencer avec Java

```
/* exemple 5 : procédure stockée avec un paramètre en entrée et un résultat codage du progr. java */
import java.sql.*;        // importation des classes jdbc

public class Class1
{
        public static void main (String[] args)
        {
                int paye;
                try
                {
                        Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
                }
                catch(Exception e)
                {
                        System.out.println ("le pilote n'a pu être chargé");
                }
                try
                {
                    Connection con= DriverManager.getConnection ("jdbc:odbc:banque","arrault","");

//Appel d'une procédure stockée avec un paramètre en entrée et retournant un ResultSet
                        CallableStatement cs= con.prepareCall ("{call paramresult(?)}");

                        //passage du paramètre en entrée
                        cs.setString (1,"perrin");

                        //exécution de la requête et ,récupération de la table
                        ResultSet result=cs.executeQuery();

                        //affichage d'un champ de la réponse récupérée
                        while (result.next ())
                        {
                        // récupération du troisième champ de la ligne
                        String Num=result.getString (3)  ;

                        System.out.println ("les Prenoms de Perrin: "+Num);
                        }

                        // fermeture de la requête et de la connexion
                        cs.close ();
                        con.close ();
                }
                catch(Exception e)
                {
                        e.printStackTrace ();    //ceci permet d'écrire l'exception interceptée
                }
        }
}
```

Pour commencer avec Java

/* exemple 6 : procédure stockée avec des paramètres et un code de retour

codage de la procédure sql */

```
CREATE PROCEDURE param2in1retour @nom varchar(30),@combien int
AS
BEGIN
UPDATE COMPTE set Solde=@combien
WHERE Nom=@nom
RETURN 5
END
```

Pour commencer avec Java

```
/* exemple 6 : procédure stockée avec des paramètres et un code de retour
import java.sql.*; // importation des classes jdbc

public class Class1
{
	public static void main (String[] args)
	{
		int code; // valeur du code retour récupéré de l'appel SQL
		try
		{
			Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
		}
		catch(Exception e)
		{
			System.out.println ("le pilote n'a pu être chargé");
		}
		try
		{
		    Connection con= DriverManager.getConnection("jdbc:odbc:banque","arrault","");

			// préparation de l'appel de la procédure stockée
			// ?= call indique que l'on récupère la valeur de retour
			// le premier paramètre in est fourni à l'appel
			// le deuxième paramètre in est fixé pour cette requête
			CallableStatement cs= con.prepareCall ("{?=call param2in1retour(?,123)}");

// configuration du type de la valeur de retour ( c'est forcément un entier )
			cs.registerOutParameter (1,java.sql.Types.INTEGER );

			// la valeur du premier paramètre in est configurée
			cs.setString (2,"perrin");

			// exécution de la requête
			cs.executeUpdate ();

			// récupération du résultat de la valeur de retour
			code=cs.getInt(1);

			// fermeture de la requête et de la connexion
			cs.close ();
			con.close ();
			System.out.println ("code retour :"+code);
		}
		catch(Exception e)
		{
			e.printStackTrace ();	//ceci permet d'écrire l'exception interceptée
		}
	}
}
```

Pour commencer avec Java

/* exemple 7 : procédure stockée avec un code retour et un ResulSet

codage de la procédure sql */

```
CREATE PROCEDURE paramtotal @nom varchar(30),@max money OUTPUT,@combien int
AS
BEGIN
UPDATE COMPTE SET Solde=@combien
WHERE Nom=@nom
SELECT *
FROM COMPTE
WHERE Solde=@combien
SELECT @max=MAX(Solde)
FROM COMPTE
RETURN 5
END
```

Pour commencer avec Java

```
/* exemple 7 : procédure stockée avec un code retour et un ResulSet
import java.sql.*;  // importation des classes JDBC

public class Class1
{
	public static void main (String[] args)
	{
		int code;
		try
		{
			Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
		}
		catch(Exception e)
		{
			System.out.println ("le pilote n'a pu être chargé");
		}
		try
		{
		    Connection con= DriverManager.getConnection("jdbc:odbc:banque","arrault","");

			// constitution de la requête d'appel à la procédure avec
			// une valeur prédéfinie pour un paramètre (4)
			CallableStatement cs= con.prepareCall("{?=call paramtotal(?,?,123)}");
			//                                    (1)                 (2)(3)(4)
			// (1) est la valeur retournée par la fonction
			// (2) est le premier paramètre. Il est en entrée
			// (3) est le deuxième paramètre. Il est en sortie
			// (4) est le troisième paramètre. Il est en entrée et fixé dans la requête

			// description de la valeur de retour de la procédure (1)
			cs.registerOutParameter (1,java.sql .Types .INTEGER );

			// description du paramètre en sortie (3)
			cs.registerOutParameter (3,java.sql .Types .DOUBLE  );

			// la valeur du premier paramètre in est configurée (2)
			cs.setString (2,"perrin");

			// exécution de la requête
			ResultSet rs=cs.executeQuery  ();

			// il faut commencer par exploiter le ResultSet
			// sinon ça ne marchera pas !!!

			// parcours du ResultSet jusqu'au bout
			while (rs.next ())
				{
				System.out .println (rs.getInt (1));
				}
```

Pour commencer avec Java

```
                        // récupération de la valeur de retour de la procédure
                        code=cs.getInt(1);

                        // récupération de la valeur du paramètre en sortie
                        System.out.println ("le solde max est:"+cs.getDouble(3));

                        // fermeture de la requête et de la connexion
                        cs.close ();
                        con.close ();

                        System.out.println ("code retour :"+code);
                }
                catch(Exception e)
                {
                        //ceci permet d'écrire l'exception interceptée
                        e.printStackTrace ();
                }
        }
}
```

Pour commencer avec Java

1) Le Contrôle d'intégrité de la base de donnée.

Il serait temps de s'intéresser à l'intégrité de la base de donnée. Toutes les requêtes effectuées jusqu'à maintenant ont été effectuées en autocommit. Le mode de fonctionnement est une caractéristique de la connection. Le mode par défaut est autocommit, c'est à dire que chaque requête SQL est considérée comme une transaction individuelle. Le commit s'effectue quand la requête se termine, ou quand la prochaine exécution démarre (le premier des deux). Dans le cas de requête retournant un ResultSet, le commit s'effectue quand la dernière ligne du ResultSet a été lue, ou que le ResultSet à été fermé.

Les méthodes suivantes sont utilisées (méthodes de la classe Connection):

- getAutoCommit() : retourne le mode vrai si autocommit.
- setAutoCommit(true) : positionne le mode autocommit à vrai.
- commit() : si le mode est non autocommit, cela enregistre les derniers changements depuis le dernier commit/rollback, et relâche tous les verrous posés par la connexion.
- Rollback(): si le mode est non autocommit, cela annule les derniers changements depuis le dernier commit/rollback, et relâche tous les verrous posés par la connexion.
-

Voici un exemple de sécurisation de transaction en utilisant les exceptions, pour ne valider une transaction que si tout c'est bien passé.

Pour commencer avec Java

```
// programme java de préservation de l'intégrité de la base de donnée
import java.sql.*;  // importation des classes jdbc

public class Class1
{
	public static void main (String[] args)
	{
		Connection con=null; // référence de la connexion
		try
		{
			Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
		}
		catch(Exception e)
		{
			System.out.println ("le pilote n'a pu être chargé");
		}
		try
		{
		con= DriverManager.getConnection ("jdbc:odbc:banque","arrault","");

			Statement select=con.createStatement ();

// spécifions que les transactions devront être validées manuellement
			con.setAutoCommit (false);

/*appel de la méthode executeUpdate de la classe Statement qui permet d'écrire dans une base nous voulons ici faire les deux mises à jour, ou aucune des deux pour des raisons ( supposées ) d'intégrité de la base. */
	select.executeUpdate("INSERT INTO COMPTE(NumCompte, Nom, Prenom, Solde)"+
					"VALUES(23, 'Grisolano', 'Philippe', 5550)");

	select.executeUpdate("INSERT INTO COMPTE(NumCompte,Nom,Prenom,Solde)"+
					"VALUES(22, 'Grisolano', 'Philippe', 3680)");
```

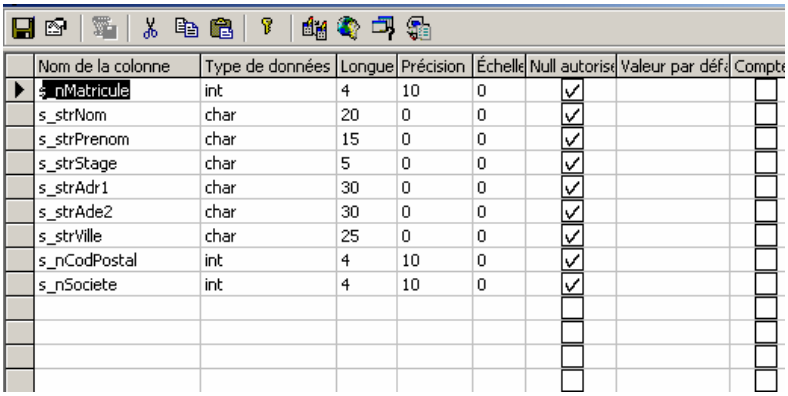
Pour commencer avec Java

```
//validation des deux requêtes si aucune des 2 n'a //générée une exception
                        con.commit ();
                        //fermeture de la requête et de la connexion
                        select.close();
                        con.close ();
                }
                catch(SQLException e)
                {

                        e.getSQLState  ();
                        try
                        {
//si la connection existe
                                if (con!= null)
                                {
//on annule la transaction
                                        con.rollback();
                                }
                        }
                        catch (Exception er) {}
                }
        }
}
```

Pour commencer avec Java

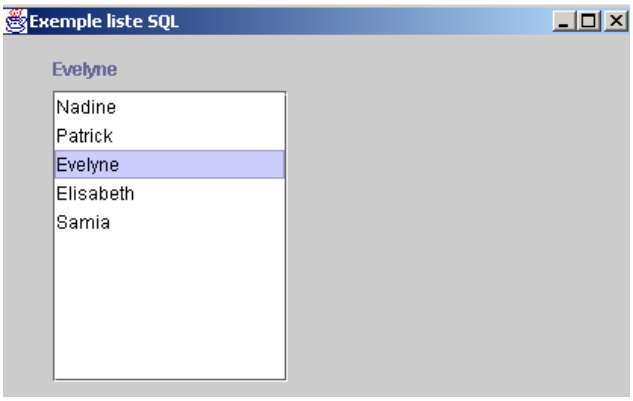
Exemple :Lister les prénoms de la table stagiaires de la base de données BasExemples créée à l'aide de SQLServer

La table a la structure suivante :



Nom de la colonne	Type de données	Longue	Précision	Échelle	Null autorise	Valeur par défa	Compte
s_nMatricule	int	4	10	0	☑		☐
s_strNom	char	20	0	0	☑		☐
s_strPrenom	char	15	0	0	☑		☐
s_strStage	char	5	0	0	☑		☐
s_strAdr1	char	30	0	0	☑		☐
s_strAde2	char	30	0	0	☑		☐
s_strVille	char	25	0	0	☑		☐
s_nCodPostal	int	4	10	0	☑		☐
s_nSociete	int	4	10	0	☑		☐
					☐		☐
					☐		☐
					☐		☐
					☐		☐

La JFrame aura l'aspect suivant



Un double clic avec la souris sur un nom permet de l'afficher au-dessus de la liste.

Pour commencer avec Java

Le code est :

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

public class TestListSQL extends JFrame
{
  JScrollPane scrollAscenseur = new JScrollPane();
  JList listeNom = new JList();
  JLabel lblListe = new JLabel();
  DefaultListModel listModel = new DefaultListModel();
  Connection contact;
  Statement Requete;            // contiendra la requête
  ResultSet rsExtrait;          // contiendra le résultat de la requête SQL

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

  public TestListSQL()
  {
    initGUI();
    WindowListener  jEcoute = new WindowAdapter()
          {
             public void windowClosing(WindowEvent e)
             {
                        System.exit(0);
             }
          };
   this.addWindowListener(jEcoute);
  }

  public void initGUI()
  {
    this.getContentPane().setLayout(null);
    this.setTitle("Exemple liste SQL");
    this.setBounds(new Rectangle(50,150,425,275));
    // Taille de la zone de défilement
    scrollAscenseur.setBounds(new Rectangle(35, 36, 155, 192));
    listeNom.setModel(listModel);
```

Pour commencer avec Java

```
  //  établir la connection  ================================================
  try
  {
     String strNomDriver = "sun.jdbc.odbc.JdbcOdbcDriver";
     // propriétés système : nom de ma connection, profil, mot passe
     Properties propriétésConnect = System.getProperties();
     propriétésConnect.put("jdbc.drivers", strNomDriver);
     System.setProperties(propriétésConnect);

     // Création des paramètres connection
     String dbURL = "jdbc:odbc:BasExemples";
     String strUtil = "sa";
     String strMotPasse = "";

     contact = DriverManager.getConnection(dbURL, strUtil, strMotPasse);
  }
  catch (Exception e)
  {
     System.out.println("connection non faite");
  }
    // Préparer les données  ================================================
  try
  {
    Requete = contact.createStatement();
    rsExtrait = Requete.executeQuery("SELECT s_strPrenom FROM Stagiaires");
  }
  catch(SQLException e)
  {
     System.out.println("requete non effectuée");
  }

  // Alimenter la liste
  try
  {
    while(rsExtrait.next() == true)
    {
       System.out.println("je suis passé par là");
       listModel.addElement(rsExtrait.getString("s_strPrenom"));
    }
  }
  catch (SQLException e)
  {
  }
   // Initialisation de l'étiquette
   lblListe.setBounds(35,14,155,15);
   lblListe.setBackground(new Color(230,230,230));
```

Pour commencer avec Java

```
    // Classe de traitement d'évèmenent utilisée pour la liste
    listeNom.addMouseListener(new MouseAdapter()
    {
      public void mouseClicked(MouseEvent e)
      {
         if (e.getClickCount() == 2)
         {  // après un double-clic
            String sélection = (String)(listeNom.getSelectedValue());
            lblListe.setText(sélection);
         }
      }
    });

    scrollAscenseur.getViewport().add(listeNom);
    this.getContentPane().add(scrollAscenseur);
    this.getContentPane().add(lblListe);

    this.setVisible(true);

   try
    {
      rsExtrait.close();
      Requete.close();
      contact.close();
    }
    catch(SQLException e)
    {
    System.out.println("erreur non exploitée");
    }
  }
}
```

Pour commencer avec Java

Tous les exemples précédents ont été écrits à partir d'un driver "sun.jdbc.odbc.JdbcOdbcDriver".

Ce driver ne permet pas tous les types de "Statement" tels que

```
strRequete = contact.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                      ResultSet.CONCUR_UPDATABLE);
```

Pour permettre ce type, nous avons du utiliser un autre driver trouvé à partir du site de la société SUN.

En fonction du driver utilisé, il y a plus ou moins de possibilités concernant les :

- bases de données supportées
- types d'action supportées

C'est principalement l'aspect connexion qui varie d'un driver à l'autre ; nous vous montrons ici une méthode pour une connexion au driver " com.inet.tds.TdsDriver ".

```
public boolean AirConnect()
{
 String dbURL = "jdbc:inetdae:localHost:1433?database=BasAirJava";
 boolean bConnectFail = false;
 contact = null;

 try
 {
    Class.forName("com.inet.tds.TdsDriver");
 }
 catch (ClassNotFoundException e)
 {
    MesOutils.AfficheMessage("Driver pas Ok "+ e);
 }
 // établir la connection  ========================================
 try
 {
  contact = DriverManager.getConnection(dbURL, "sa",  "");
 }
 catch (Exception e)
 {
    bConnectFail = true;
    MesOutils.AfficheMessage("JDBC pas Ok" + e);
 }
return bConnectFail;
}
```

Pour commencer avec Java

12.Lexique

Remarque : une partie des ces éléments provient de sites Internet

abstract

Il s'agit d'un modificateur utilisé lors de la déclaration d'une classe ou d'une méthode. On dit alors que la classe ou la méthode est abstraite.
Une méthode abstraite n'a que son prototype, c'est -à-dire son type de retour suivi, de son nom, suivi de la liste de ses paramètres entre des parenthèses, suivi d'un point-virgule. Une méthode abstraite ne peut pas être déclarée `static` ou `private` ou `final`.
Dès qu'une classe contient une méthode abstraite, elle doit elle aussi être déclarée abstraite. Une classe abstraite ne peut pas être instanciée. Il faudra l'étendre et définir toutes les méthodes abstraites qu'elle contient pour pouvoir l'utiliser.

API

Cela signifie Application Programming Interface.
Une API contient un ensemble de fonctions qui facilitent la programmation. L'API de java contient entre autres les paquetages java.applet, java.awt (awt pour Abstract Window Toolkit), java.awt.image, java.io, java.lang, java.net et java.util.

API de pilote JDBC

L'une des principales interfaces de JDBC. Elle est représentée sous forme de séries d'interfaces Java abstraites qui permettent à un programmeur de créer des pilotes écrits en Java qui se connectent à des bases de données SQL.

API JDBC

L'une des principales interfaces de JDBC. Elle est représentée sous forme de séries d'interfaces Java abstraites qui permettent à un programmeur d'applications d'ouvrir des connexions à desbases de données particulières, d'exécuter des instructions SQL et de traiter les résultats.

awt

Cela signifie Abstract Window Toolkit. C'est le nom du paquetage de java traitant de ce qui est nécessaire pour gérer des fenêtres, des interfaces graphiques, des opérations de dessin.

classe

Une classe est la description d'un ensemble d'attributs et de méthodes chargées de gérer ces attributs. Une classe comporte :

Pour commencer avec Java

- des attributs et des méthodes d'instance
- des attributs et des méthodes de classe

Une classe peut-être :

- publique, si elle a le modificateur de visibilité **public** elle est alors visible que de son propre paquetage.
- avoir la visibilité par défaut : elle est alors visible de partout.

Une classe peut posséder le modificateur **`final`** : elle ne peut alors pas être étendue.
Une classe peut posséder le modificateur **`abstract`** : elle est alors abstraite. Dès qu'une classe contient une méthode abstraite, elle doit elle aussi être déclarée abstraite.

constante

Toute variable déclarée *final* en Java est une constante. Sa valeur doit être spécifiée avec un initialiseur quand elle est déclarée et sa valeur ne doit jamais être modifiée. Par exemple :
public static final double pi=3.14159;

constructeur

Toute classe contient une méthode appelée constructeur. Cette méthode porte le même nom que la classe qui la contient et ne doit pas indiquer de valeur de retour ; l'objet retourné est implicitement une instance de la classe.

Le constructeur sert à initialiser les attributs de la classe. La première ligne d'un constructeur est un appel au constructeur de la super-classe. Lorsque le programmeur n'a pas défini de constructeur, le compilateur en ajoute automatiquement un qui ne comporte qu'une seule instruction :

```
super();
```

attributs et méthodes de classe

Un attribut ou une méthode d'une classe sont dits de classe lorsque le modificateur **`static`** est appliqué lors de leur déclaration. On dira aussi qu'il s'agit d'un attribut ou d'une méthode statiques.
Un attribut déclaré **`static`** existe dès que sa classe est évoquée, en dehors et indépendamment de toute instanciation. Quel que soit le nombre d'instanciations de la classe (0, 1, ou plus), un attribut de classe, i.e. statique, existe en un et un seul exemplaire. Un tel attribut sera utilisé un peu comme une variable globale d'un programme non objet.
Une méthode, pour être de classe ne doit pas manipuler, directement ou indirectement, des attributs non statiques de sa classe. En conséquence, une méthode de classe ne peut utiliser directement aucun attribut ni aucune méthode non statiques de sa classe ; une erreur serait détectée à la compilation.
De l'extérieur d'une classe ou d'une classe héritée, un attribut ou une méthode de classe pourront être utilisés précédés du nom de leur classe
(`nom_d'une_`**`classe`**`.nom_de_l_attribut_ou_méthode_de_classe`).

Pour commencer avec Java

attributs et méthodes d'instance

Un attribut ou une méthode d'une classe sont dits d'instance dès que le modificateur **`static`** n'est pas appliqué lors de leur déclaration.
Un attribut d'instance n'existe par rapport à une allocation mémoire que dans une instance de cette classe. Avant qu'une instanciation de la classe ne soit effectuée, l'attribut n'a aucune existence physique. Si plusieurs instances de la même classe coexistent, il y aura un exemplaire d'un certain attribut d'instance de la classe par instance créée.
Une méthode doit être d'instance lorsqu'elle manipule, en lecture ou en écriture, des attributs d'instance de sa classe. À l'envers, il sera plus correct de déclarer en **`static`** une méthode n'utilisant aucun attribut ou méthode non statiques de sa classe. Lorsqu'une méthode d'instance est invoquée, la référence (qui sera appelée **`this`** durant son déroulement) de l'instance traitée lui est passée implicitement.
Un attribut ou une méthode d'instance ne pourront être invoqués, à l'extérieur de la définition de leur classe ou d'une classe héritée, qu'à partir d'une instance de classe (`nom_d'une_`**`instance.`**`nom_de_l_attribut_ou_méthode_d'instance`).

attributs et méthodes privés

Un attribut ou une méthode sont dits privés si on leur applique le "modificateur de visibilite" **`private`**.

Un attribut ou une méthode privés ne sont pas visibles en-dehors de la définition de leur classe. Un attribut privé ne pourra donc pas être modifié par des instances de la classe qu'à travers des méthodes de la classe prévues à cet effet ; le programmeur de la classe peut ainsi contrôler le comportement des attributs privés (imaginer par exemple un attribut entier qui doit toujours être strictement positif pour qu'une instance de la classe se comporte correctement ; il serait dangereux qu'un tel attribut puisse être modifié par un utilisateur de la classe).

attributs et méthodes protégés

On dit qu'un attribut d'une classe est protégé si on lui a affecté le modificateur de visibilité **`protected`**. Un champ protégé d'une classe `A` est toujours visible à l'intérieur de son paquetage. A l'extérieur du paquetage de `A`, considérons une classe `B` qui hérite de `A` ; un champ protégé de `A` est hérité par `B` et visible directement dans `B` (au travers de la référence implicite **`this`**) et visibles également au travers d'instances de `B` ou de sous-classes de `B` définies dans `B` ; il n'est pas visible dans d'autres conditions.

CORBA

Common Object Request Broker Architecture. Environnement informatique hétérogène, multilangage et distribué possédant un modèle objet indépendant des langages. Les mécanismes de CORBA permettent aux objets d'émettre des requêtes et de recevoir des réponses de façon transparente.

Pour commencer avec Java

encapsulation

On parlera de l'encapsulation de données. Il s'agit d'un principe fondamental d'un langage objet: la possibilité de "cacher" des données à l'intérieur d'une classe en ne permettant de les manipuler qu'au travers des méthodes de la classe. L'intérêt essentiel d'une telle opération est d'interdire que les données soient modifiées de façon contraire à l'usage attendu dans sa classe, et ainsi de maintenir l'ensemble des données de la classe dans un état cohérent. Un exemple simple est le cas d'une donnée numérique qui devrait rester positive pour garder sa signification : il est indispensable qu'un utilisateur ne puisse pas, par mégarde ou ignorance, lui attribuer une valeur négative. Pour encapsuler une donnée, il suffit de la déclarer privée, c'est-à-dire de lui attribuier le modificateur de visibilité "**`private`**". Les modificateurs de visibilité "par défaut " ou "**`protected`**" correspondent aussi à une certaine encapsulation.

exception

Une exception correspond à un événement anormal ou inatendu. Les exceptions sont des instances de sous-classes des classes `java.lang.Error` (pour des erreurs graves, qui devront généralement conduire à l'arrêt du programme) et `java.lang.Exception` (pour des événements inattendus, qui seront souvent traités sans que cela provoque l'arrêt du programme). Un grand nombre d'exceptions sont définis dans l'API.
Un mécanisme, utilisant le mot réservé **`throw`**, permet de "lancer une exception" :

```
if (il_y_a_un_probleme) throw new MyException)();
```

où MyException serait ici une sous classe de `java.lang.Exception` définie par ailleurs Quand une exception est lancée, toutes les instructions suivantes sont ignorées et on remonte la pile des appels des méthodes : on dit que l'exception se propage. Si, soit l'instruction qui a lancé l'exception, soit une méthode de la pile des appels, est situé dans un "bloc **`try`**":

- suivi d'un "bloc-finally", les instructions du "bloc **`finally`**" sont exécutées et la propagation se poursuit ;
- suivi d'un "bloc **`catch`**" attrapant les exceptions de la classe de l'exception qui se propage, les instructions du "bloc **`catch`**" sont exécutées puis l'exécution reprend son cours normal avec l'instruction qui suit le "bloc **`catch`**".

Remarquons qu'une méthode qui peut lancer une exception sans l'attraper durant son exécution doit le signaler dans son en-tête en utilisant le mot réservé **`throws`** Par exemple :

```
void maFonction() throws MyException
```

signale qu'au cours de l'exécution de la méthode `maFonction()` une exception de type `MyException` peut être lancée sans être attrapée.

Pour commencer avec Java

On n'est néanmoins dispensé de signaler dans les en-têtes des méthodes le lancement éventuel des exceptions les plus classiques, comme une `ArrayIndexOutOfBoundsException` par exemple. Vous pouvez compter sur le compilateur pour vous dire si vous avez oublié de signaler qu'une méthode peut lancer une certaine exception.
Beaucoup de méthode de l'API lancent des exceptions que l'on peut tenter d'attraper.

extends

Ce mot réservé peut être utilisé suivi d'un nom de classe lors de la définition d'une nouvelle classe.
Exemple :

```
class MaNouvelleClasse extends ClasseGenerale
{
...
}
```

`MaNouvelleClasse` est alors dite sous-classe de `ClasseGenerale`. On dit aussi que `MaNouvelleClasse` étend `ClasseGenerale`. `MaNouvelleClasse` hérite de toutes les méthodes de `ClasseGenerale` qui sont visibles
Une classe étend au plus une autre classe.
Une interface peut aussi étendre (grâce à **`extends`**) une autre ou plusieurs autres interfaces.

final

Il s'agit d'un modificateur qui s'applique à une classe, une donnée ou une méthode.

- Pour une classe, **`final`** indique que la classe ne peut pas être étendue.
- Pour une méthode, **`final`** indique que la méthode ne peut pas être redéfinie. Une méthode qui est déclarée **`static`** ou **`private`** est automatiquement **`final`**.
- Pour une donnée, **`final`** indique qu'il s'agit d'une constante, d'instance s'il n'y a pas simultanément le modificateur **`static`**, et de classe si la donnée est **`final static`**. Une donnée **`final`** ne pourra être affectée qu'une seule fois.

finally

La clause précède un bloc d'instructions. La clause et le bloc d'insructions constituent ce qu'on appelle un "bloc **`finally`**".
Un "bloc **`finally`**"est en général utilisé pour effectuer des nettoyages (fermer des fichiers, libérer des ressources...).

Un bloc **`finally`** suit:

- soit un "bloc **`try`**"
- soit un "bloc **`try`**" suivi d'un "bloc **`catch`**"

Pour commencer avec Java

Dans les deux cas, quel que soit la façon dont on est sorti du "bloc **`try`**" (par une instruction **`break`**, ou **`continue`**, ou **`return`**, par une propagation d'exceptions, ou bien normalement), les instructions contenues dans le "bloc **`finally`**" sont exécutées. Si un "bloc **`catch`**" situé entre le "bloc **`try`**" et le "bloc **`finally`**" attrape une exception, les intructions du "bloc **`catch`**" sont faites avant celles du "bloc **`finally`**".

gestionnaire de répartition

Un gestionnaire de répartition est une classe qui permet de gérer la répartition des sous-composants graphiques dans un composant graphique. La méthode `setLayout` que l'on trouve dans la classe `java.lang .Container` dont hérite tout composant graphique qui peut en contenir d'autres, permet de choisir un gestionnaire de répartition. Vous pouvez utiliser :

- `java.awt.BorderLayout` : pour répartir les sous-composants au nord, au sud, à l'est, à l'ouest ou au centre du composant.
- `java.awt.CardLayout` : pour répartir un ensemble de sous-composants de telle sorte qu'un seul soit visible à la fois, comme s'il s'agissait d'un jeu de cartes.
- `java.awt.FlowLayout` : pour arranger les sous-composants en ligne, de la gauche vers la droite et du haut vers le bas au fur et à mesure de leurs insertions.
- `java.awt.GridLayout` : pour arranger les sous-composants dans une grille dont on peut éventuellement préciser le nombre de lignes et de colonnes.
- `java.awt.GridBagLayout` : pour arranger les sous-composants dans une grille, mais un sous-composant peut utiliser plusieurs lignes ou colonnes et on peut donner diverses informations sur la disposition des sous-composants.
- `javax.swing.BoxLayout` qui permet de répartir les composants soit les uns en-dessous des autres, soit les uns à côté des autres.

Si on n'utilise pas de gestionnaire de répartition, il faut préciser directement les positions et les tailles des différents sous-composants graphiques. Des méthodes sont prévues à cet effet dans la classe `java.awt.Component`.

héritage

C'est un principe fondamental de la programmation objet.
Les sous-classes d'une classe disposent de tous les attributs et méthodes de sa super-classe, moyennant néanmoins quelques nuances liées aux modificateurs de visibilité et au principe de la redefinition des méthodes.

En Java, une classe peut hériter d'une seule autre classe : Java ne permet pas l'héritage multiple.

`implements`

Pour commencer avec Java

Ce mot est employé dans l'en-tête de la déclaration d'une classe, suivi d'un nom d'interface ou de plusieurs noms d'interfaces séparés par des virgules. S'il y a une clause **`extends`**, la clause **`implements`** doit se trouver après la clause **`extends`**.

Exemple :

```
class MaClasse extends AutreClasse
   implements  UneInterface, UneAutreInterface
```

Si une classe non abstraite possède une clause **`implements`**, elle doit définir toutes les méthodes de l'interface indiquée.

`import :`

Le langage Java est fourni avec de nombreux paquets prédéfinis. Par exemple, le paquet java.applet contient des classes permettant de travailler avec les applets Java.:

```
public class Bonjour extends java.applet.Applet {
```

Dans ce code, nous avons réellement fait référence à la classe appelée Applet dans le paquet java.applet. Il n'est pas pratique d'avoir à répéter le nom complet de la classe java.applet.Applet chaque fois qu'il faut faire référence à cette classe. Java propose une autre solution. Vous pouvez importer un paquet fréquemment utilisé :

```
import java.applet.*;
```

Cela précise au compilateur "si tu rencontres un nom de classe inconnu, cherches dans le paquet java.applet". Maintenant, pour déclarer notre nouvelle classe, nous pouvons dire,

```
public class Bonjour extends Applet {
```

ce qui est plus court. Cependant, cela pose un problème si deux paquets importés contiennent deux classes qui portent le même nom. Dans ce cas, il faut utiliser le nom complet.

interface

Une interface comporte un en-tête (qui pourrait se voir attribuer le modificateur de visibilité **`public`**) suivi du mot réservé **`interface`** suivi du nom de l'interface, puis, entre accolades, une liste de constantes et de prototypes de méthodes.

Une interface `I` peut être implémentée par une classe `A`, ce qui signifie que :

- la classe `A` déclare implémenter `I`, avec le mot **`"implements"`**

Pour commencer avec Java

- toutes les méthodes prototypées dans la classe `I` doivent être définies dans la classe `A`.

Une interface sert essentiellement à

- regrouper des constantes
- permettre qu'un ensemble de classes, étendant éventuellement d'autres classes, puissent implémenter une même interface.

Les interfaces ne seraient pas indispensables si Java permettait l'héritage multiple (possibilité d'étendre plusieurs classes à la fois).
Un nom d'interface peut être utilisé comme un nom de classe. On pourra donc mettre dans une variable dont le type est une interface n'importe quel objet implémentant l'interface en question. Quelque soit l'objet référencé par la variable, on pourra lui appliquer une méthode déclarée dans l'interface.

instance

Une classe est une description abstraite d'un ensemble d'objets. Une instance de classe est un objet construit selon le modèle fourni par la classe.
Un objet et une instance de classe peuvent être considérés comme synonyme.

méthode

On appelle ainsi ce qu'on appellerait fonction ou procédure dans d'autres langages. Toute méthode fait partie d'une classe. Une méthode peut être soit d'instance, soit de classe.

Modale

Empêche la saisie dans n'importe quelle autre fenêtre de l'application tant que le dialogue n'est pas refermé.

modificateurs de visibilite

Il est possible d'indiquer pour une classe ou un champ (attribut ou méthode) d'une classe un certain degré d'accessibilité. Cela se fait avec les mots **`public`**, **`private`** ou **`protected`** situés au début de l'en-tête de la classe ou du champ en question. Ces mots sont les modificateurs de visibilite.

- Une classe possède deux degrés de visibilité :le degré par défaut et le degré public indiqué par le mot réservé **`public`**. Une classe est toujours visible de tout son paquetage, une classe n'est visible d'un autre paquetage que si elle est publique.
- Un attribut ou une méthode possèdent quatre degrés de visibilité : les modes indiqués par les mots **`public`**, **`private`** ou **`protected`** auquels s'ajoute le mode par défaut ; dans ce dernier cas, l'attribut ou la méthode en question sont visibles de partout à l'intérieur de son paquetage et de nulle part à l'extérieur de celui-ci.

Pour commencer avec Java

new

Il s'agit d'un opérateur unaire qui crée un nouvel objet ou un nouveau tableau.

objet

On appelle ainsi une instance d'une classe.

polymorphisme

c'est la capacité d'un objet à décider quelle méthode il doit appliquer sur lui-même selon sa hiérarchie dans les classes. Il se fonde sur le fait que deux objets peuvent émettre des réponses différentes alors qu'ils ont reçu le même message.

`private`

Ce mot réservé est un modificateur de visibilité qui s'applique à un champ (attribut ou méthode) d'une classe. On dit alors que le champ est privé. Un champ privé n'est visible que depuis sa propre classe. Elle n'est visible nulle part ailleurs et en particulier pas dans les sous-classes.

`protected`

Ce mot réservé est un modificateur de visibilité qui s'applique à un champ (attribut ou méthode) d'une classe. On dit alors que le champ est protégé. Un champ protégé d'une classe A est toujours visible à l'intérieur de son paquetage. A l'extérieur de son paquetage, un champ protégé est hérité par une sous-classe B mais non visible au travers d'une instance de A ou de B (sauf pour une instance de B invoquée dans B ou dans une sous-classe de B).

`public`

Ce mot réservé est un modificateur de visibilité qui s'applique à une classe, une interface ou à un champ (attribut ou méthode) d'une classe. On dit alors que la classe, l'interface ou le champ est publique.
Une classe ou une interface publique est visible de partout, y compris les autres paquetages. Si ce modificateur n'est pas appliqué à une classe ou une interface, celle-ci n'est visible que de l'intérieur de son paquetage.
Un champ public est visible de partout du moment que sa classe est visible.

redéfinir

On peut redéfinir une méthode définie dans une classe à l'intérieur d'une sous-classe de celle-ci. Pour cela, il faut que la méthode redéfinie ait même nom, mêmes types et nombre d'arguments et même type de retour que la méthode d'origine. L'interpréteur cherche la définition d'une methode invoquée à partir de la classe de l'instance concernée, en retenant la première définition rencontrée correspondant à la liste des paramètres d'appel, en remontant de classe en super-classe.

Pour commencer avec Java

static

Ce modificateur s'applique aux attributs et aux méthodes ; lorsque ce modificateur est utilisé, on dit qu'il s'agit d'un attribut ou d'une méthode de classe.
On rappelle qu'une méthode ou un attribut auxquels n'est pas appliqué le modificateur **static** sont dits d'instance.

- Un attribut déclaré **static** existe dès que sa classe est évoquée, en dehors et indépendamment de toute instanciation. Quel que soit le nombre d'instanciation de la classe (0, 1, ou plus) un attribut de classe, i.e. statique, existe en un et un seul exemplaire. Un tel attribut sera utilisé un peu comme une variable globale d'un programme non objet.
- Une méthode, pour être de classe (i.e. **static**) ne doit pas manipuler, directement ou indirectement, des attributs non statiques de sa classe. En conséquence, une méthode de classe ne peut utiliser directement dans son code aucun attribut ou aucune méthode non statique de sa classe ; une erreur serait détectée à la compilation. Autrement dit, une méthode qui utilise (en lecture ou en en écriture) des attributs d'instance ne peut être statique, et est donc nécessairement une méthode d'instance.

De l'extérieur d'une classe ou d'une classe héritée, un attribut ou une méthode de classe pourront être utilisés précédés du nom de leur classe :

```
nom_d'une_classe.nom_de_la_donnée_ou_méthode_de_classe
```

ou bien (mais cela est moins correct) du nom d'une instance de la classe.
Signalons enfin qu'une méthode **static** ne peut pas être redéfinie, ce qui signifie qu'elle est automatiquement **final**.

super

Ce mot réservé permet d'invoquer un champ (attribut ou méthode) de la super-classe de l'objet sur lequel on est entrain de travailler (objet dit courant), et qui serait sinon caché par un champ de même nom de la classe de l'objet courant (soit que le champ en question soit une méthode qui a été redéfinie, soit que le champ soit un attribut qui a été masqué). Lorsque, dans une méthode, on invoque un champ de l'objet courant, on ne précise en général pas le nom de l'objet, ce que l'on peut néanmoins faire avec le mot **this**. Lorsqu'on emploie **super**, ce mot vient remplacer **this** qui figure implicitement. Employer **super** revient à remplacer la classe de l'objet courant par la super-classe.
Remarquons qu'il est interdit de "remonter de plusieurs étages" en écrivant **super.super**.etc.
On utilise aussi **super**(paramètres) en première ligne d'un constructeur. On invoque ainsi le constructeur de la super-classe ayant les paramètres correspondants.

Pour commencer avec Java

super-classe

On appelle ainsi une classe qui est étendue par d'autres classes.
Considérons une classe C. La super-classe de C est la classe dont hérite directement C. Cette super-classe peut être indiquée par le mot réservé `extends` au moment de la définition de la classe. Si ce n'est pas le cas, la classe C hérite directement de la classe `java.lang.Object`, qui est donc sa super-classe.

surcharge

Une classe peut définir plusieurs méthodes ayant un même nom du moment que les suites des types de paramètres de ces méthodes diffèrent (globalement, ou par leurs ordres). Une différence uniquement sur le type de la valeur retournée est interdite.
Lorsqu'une méthode surchargée est invoquée, la différence sur les paramètres permet de déterminer la méthode à exécuter.

`synchronized`

Cela peut être un modificateur pour une méthode ou bien une instruction.

- Lorsqu'une méthode **`synchronized`** est invoquée, celle-ci ne pourra s'exécuter que lorsqu'elle aura obtenu un verrou sur l'instance à laquelle elle s'applique ; elle gardera alors le verrou jusqu'à ce qu'elle soit totalement exécutée.
- L'instruction **`synchronized`** est utilisée de la façon suivante :

synchronized(identificateur)
{
  instruction1;
  instruction2;
  ...
}

L'identificateur `identificateur` situé dans les parenthèses doit être un objet ou un tableau. Le bloc suivant l'instruction **`synchronized`** s'appele section critique. Pour exécuter la section critique, le bloc doit obtenir un verrou sur l'objet ou le tableau représenté par `identificateur`.

`this`

On peut utiliser **`this`** comme un objet ou bien **`this`**(paramètres) comme une méthode. Dans un constructeur ou une méthode d'instance, l'objet représenté par **`this`** est l'objet avec lequel on est entrain de travailler. **`this`** est généralement utilisée pour accéder à un attribut d'instance masqué par un argument de la méthode ou une donnée locale. On peut aussi utiliser **`this`** pour passer l'objet sur lequel on est entrain de travailler en paramètre à une méthode. Invoquer une méthode **`this`**(paramètres) ne peut se faire qu'en première ligne d'un constructeur. On invoque alors un autre constructeur dont la liste de paramètres corresponde aux classes des paramètres indiqués dans la parenthèse.

Pour commencer avec Java

throw

C'est ce mot réservé qui permet de "lancer" une exception lorsqu'un événement exceptionnel s'est produit. On écrira par exemple :

```
if (il_y_a_un_probleme) throw new MyException)();
```

où MyException serait ici une sous classe de `java.lang.Exception` définie par ailleurs. Remarquez bien que l'on lance une instance d'une exception (présence du mot **new**). Quand une exception est lancée, toutes les instructions suivantes sont ignorées et on remonte la pile des appels des méthodes : on dit que l'exception se propage. Pendant la propagation si une méthode de la pile d'appel a été invoquée à l'intérieur d'un "bloc **try**":

- suivi d'un "bloc **finally**", les instructions du "bloc **finally**" sont exécut&eacut;s et la propagation se poursuit ;
- suivi d'un "bloc **catch**" attrapant les exceptions de la classe de l'exception qui se propage, les instructions du "bloc **catch**" sont exécutés puis l'exécution reprend son cours normal avec l'instruction qui suit le "bloc **finally**".

throws

Lorsqu'une méthode contient une instruction (éventuellement à travers l'appel à une autre méthode) susceptible de lancer une certaine exception sans l'attraper (par un mécanisme **try-catch**), elle doit l'annoncer dans son en-tête. Par exemple :

```
void maMethode throws NumberFormatException, myException
{
  ...
  instruction pouvant lancer une erreur de la classe
     NumberFormatException ou myException;
  ...
}
```

types primitifs

Les types primitifs sont :

- le type booléen **boolean**, qui n'est pas un type entier, et qui peut prendre les valeurs **false** et **true**
- le type caractère **char**
- les types entiers **byte**, **short**, **int** et **long**
- les types nombres flottants **float** et **double**

try

Pour commencer avec Java

Ce mot doit être suivi d'un bloc d'instructions. On parlera alors d'un "bloc **`try`**". En soit, il ne fair rien d'autre que de permettre l'utilisation de blocs "**catch**" ou (et) "**finally**" qui peuvent éventuellement suivre le "bloc **`try`**"

transient

Il s'agit d'un modificateur de visibilité applicable aux variables attributs d'instance d'une classe. Il n'est actuellement pas utilisé mais il devrait désigner ultérieurement un attribut ne jouant pas de rôle dans la description de l'objet (on dira aussi qu'elle ne fait pas partie de l'état persistant de l'objet) et qu'il n'est donc pas nécessaire de sauver sur disque.

type référence

Toute variable qui n'est pas d'un type primitif est d'un type référence. Une référence est en fait une adresse en mémoire d'un objet.

variable

Emplacement en mémoire dans lequel les valeurs sont stockées. Une variable a un nom (avec lequel vous y faites référence), un type de données et une valeur. Une variable doit être déclarée afin de recevoir une valeur. Il y a trois types de variables : instance, classe et locale.

variable d'instance

Variable non-statique d'une classe qui est déclarée en dehors d'une définition de méthode. Une copie d'une variable d'instance existe dans chaque instance de la classe qui est créée.

variable locale

Variable définie à l'intérieur d'une définition de méthode.

Unicode

Un système de codage des caractères internationaux géré par le Consortium Unicode.
La page Web consacrée à Unicode à l'adresse http://www.unicode.com

La mémoire de l'ordinateur conserve toutes les données sous forme numérique. Il n'existe pas de méthode pour stocker directement les caractères. Chaque caractère possède donc son équivalent en code numérique: c'est le code ASCII (American Standard Code for Information Interchange - traduisez " Code Americain Standard pour l'Echange d'Informations"). Le code ASCII de base représentait les caractères sur 7 bits (c'est-à-dire 128 caractères possibles, de 0 à 127). Le code ASCII a été mis au point pour la langue anglaise, il ne contient donc pas de caractères accentués, ni de caractères spécifiques à une langue. Pour coder ce type de caractère il faut recourir à un autre code. Le code ASCII a donc été étendu à 8 bits (un octet) pour pouvoir coder plus de caractères (on parle d'ailleurs de code ASCII étendu...).

Ce code attribue les valeurs 0 à 255 (donc codées sur 8 bits, soit 1 octet) aux lettres majuscules et

Pour commencer avec Java

minuscules, aux chiffres, aux marques de ponctuation et aux autres symboles (caractères accentués dans le cas du code *iso-latin1*).

- Les codes 0 à 31 ne sont pas des caractères. On les appelle *caractères de contrôle* car ils permettent de faire des actions telles que:
  - retour à la ligne (CR)
  - Bip sonore (BEL)
- Les codes 65 à 90 représentent les majuscules
- Les codes 97 à 122 représentent les minuscules
  (il suffit de modifier le 5ème bit pour passer de majuscules à minuscules, c'est-à-dire ajouter 32 au code ASCII en base décimale)

12.1. Table des caractères ASCII

caractère	ASCII	Hexa									
NUL	0	00	%	37	25	K	75	4B
SOH	1	01	&	38	26	L	76	4C
STX	2	02	'	39	27	M	77	4D
ETX	3	03	(	40	28	N	78	4E
EOT	4	04	)	41	29	O	79	4F
ENQ	5	05	*	42	2A	P	80	50
ACK	6	06	+	43	2B	Q	81	51
BEL	7	07	,	44	2C	R	82	52
BS	8	08	-	45	2D	S	83	53
HT	9	09	.	46	2E	T	84	54
LF	10	0A	/	47	2F	U	85	55
VT	11	0B	0	48	30	V	86	56
NP	12	0C	1	49	31	W	87	57
CR	13	0D	2	50	32	X	88	58
SO	14	0E	3	51	33	Y	89	59
SI	15	0F	4	52	34	Z	90	5A
DLE	16	10	5	53	35	[	91	5B
DC1	17	11	6	54	36	\	92	5C
DC2	18	12	7	55	37	]	93	5D
DC3	19	13	8	56	38	^	94	5E
DC4	20	14	9	57	39	_	95	5F
NAK	21	15	:	58	3A	`	96	60
SYN	22	16	;	59	3B	a	97	61
ETB	23	17	<	60	3C	b	98	62
CAN	24	18	=	61	3D	c	99	63
EM	25	19	>	62	3E	d	100	64
SUB	26	1A	?	63	3F	e	101	65
ESC	27	1B	@	64	40	f	102	66
FS	28	1C	A	65	41	g	103	67
GS	29	1D	B	66	42	h	104	68
RS	30	1E	C	67	43	i	105	69
US	31	1F	D	68	44	j	106	6A
Espace	32	20	E	69	45	k	107	6B
!	33	21	F	70	46	l	108	6C
"	34	22	G	71	47	m	109	6D
#	35	23	H	72	48	n	110	6E
$	36	24	I	73	49	o	111	6F
			J	74	4A	p	112	70

Pour commencer avec Java

q	113	71
r	114	72
s	115	73
t	116	74
u	117	75
v	118	76
w	119	77
x	120	78
y	121	79
z	122	7A
{	123	7B
|	124	7C
}	125	7D
~	126	7E
suppr	127	7F
€	128	80
□	129	81
‚	130	82
ƒ	131	83
„	132	84
…	133	85
†	134	86
‡	135	87
ˆ	136	88
‰	137	89
Š	138	8A
‹	139	8B
Œ	140	8C
□	141	8D
Ž	142	8E
□	143	8F
□	144	90
‘	145	91
’	146	92
“	147	93
”	148	94
•	149	95
–	150	96
—	151	97
˜	152	98
™	153	99
š	154	9A
›	155	9B
œ	156	9C
□	157	9D
ž	158	9E
Ÿ	159	9F
espace	160	A0

¡	161	A1
¢	162	A2
£	163	A3
¤	164	A4
¥	165	A5
¦	166	A6
§	167	A7
¨	168	A8
©	169	A9
ª	170	AA
«	171	AB
¬	172	AC
	173	AD
®	174	AE
¯	175	AF
°	176	B0
±	177	B1
²	178	B2
³	179	B3
´	180	B4
µ	181	B5
¶	182	B6
·	183	B7
¸	184	B8
¹	185	B9
º	186	BA
»	187	BB
¼	188	BC
½	189	BD
¾	190	BE
¿	191	BF
À	192	C0
Á	193	C1
Â	194	C2
Ã	195	C3
Ä	196	C4
Å	197	C5
Æ	198	C6
Ç	199	C7
È	200	C8
É	201	C9
Ê	202	CA
Ë	203	CB
Ì	204	CC
Í	205	CD
Î	206	CE
Ï	207	CF
Ð	208	D0

Ñ	209	D1
Ò	210	D2
Ó	211	D3
Ô	212	D4
Õ	213	D5
Ö	214	D6
×	215	D7
Ø	216	D8
Ù	217	D9
Ú	218	DA
Û	219	DB
Ü	220	DC
Ý	221	DD
Þ	222	DE
ß	223	DF
à	224	E0
á	225	E1
â	226	E2
ã	227	E3
ä	228	E4
å	229	E5
æ	230	E6
ç	231	E7
è	232	E8
é	233	E9
ê	234	EA
ë	235	EB
ì	236	EC
í	237	ED
î	238	EE
ï	239	EF
ð	240	F0
ñ	241	F1
ò	242	F2
ó	243	F3
ô	244	F4
õ	245	F5
ö	246	F6
÷	247	F7
ø	248	F8
ù	249	F9
ú	250	FA
û	251	FB
ü	252	FC
ý	253	FD
þ	254	FE
ÿ	255	FF

Pour commencer avec Java