Support de Cours Java

Mourad Oussalah

Professeur à l'Université de Nantes

MO

Présentation générale et historique

Historique (1)

- **Quoi : créer un langage qui s'affranchisse des disparités matérielles et logicielles**
  - **indépendant de la machine ("architecture neutral")**
  - **portable sur différents systèmes d'exploitation**
- **Qui : Equipe de James Gosling, Sun Microsystems**
- **Quand : A partir de 1991**

Historique (2)

- 1991 : Oak
  - développement d'un nouveau langage Oak pour pallier aux faiblesses constatées de C++
- 1993 : Essor d'internet grâce au web
- 1994 : Hotjava
  - Oak est utilisé pour distribuer des applications sur internet
  - Ecriture d'un navigateur web ("web browser") en Oak
  - Sun choisit de commercialiser Oak sous le nom de Java

Les différentes versions de Java

- Trois versions de Java depuis 1995
  - Java 1.0 en 1995
  - Java 1.1 en 1996
  - Java 1.2 en 1999
- Evolution très rapide et succès du langage
- Problèmes de compatibilité (en particulier avec les navigateurs)

Et demain ?

- ➜ **LE langage d'internet**
- ➜ **Très grande diffusion grâce aux navigateurs**
- ➜ **Problèmes liés au fait que les navigateurs ne sont pas écrits par Sun**

Caractéristiques du langage Java (1)

- ➜ **Simple**
  - ➢ **Apprentissage facile**
    - **faible nombre de mot clés**
    - **simplifications aux fonctionnalités essentielles**
  - ➢ **Développeurs opérationnels rapidement**
- ➜ **Orienté objet**
  - ➢ **Java ne permet que d'utiliser les objets**
  - ➢ **Les grandes idées reprises sont : encapsulation, dualité classe /instance, attribut, méthode / message, visibilité, dualité interface/implémentation, héritage simple, redéfinition de méthodes, polymorphisme**
- ➜ **Familier**
  - ➢ **Syntaxe proche de celle de C++**

Caractéristiques du langage Java (2)

- **Sûr**
  - **Indispensable sur les réseaux (protection contre les virus, modification des fichiers, lecture de données confidentielles, etc.)**
- **Fiable**
  - **Gestion automatique de la mémoire (ramasse-miette ou "garbage collector")**
  - **Gestion des exceptions**
  - **Sources d'erreurs limitées**
    - **typage fort,**
    - **pas d'héritage multiple,**
    - **pas de manipulations de pointeurs, etc.**
  - **Vérifications faites par le compilateur facilitant une plus grande rigueur du code**

Java : un langage de programmation

- Applications Java : programmes autonomes, "stand-alone"
- Applets (mini-programmes) : Programmes exécutables uniquement par l'intermédiaire d'une autre application
  - navigateur web : Netscape, Internet explorer, Hotjava
  - application spécifique : Appletviewer

- Java est souvent considéré comme étant uniquement un langage pour écrire des applets alors que c'est aussi un vrai langage de programmation

S'affranchir de la plateforme (1)

- Java est un langage interprété
  - La compilation d'un programme Java crée du pseudo-code portable : le "byte-code"
  - Sur n'importe quelle plateforme, une machine virtuelle Java peut interpréter le pseudo-code afin qu'il soit exécuté
- Les machines virtuelles Java peuvent être
  - des interpréteurs de byte code indépendants (pour exécuter les programmes Java)
  - contenues au sein d'un navigateur (pour exécuter des applets Java)

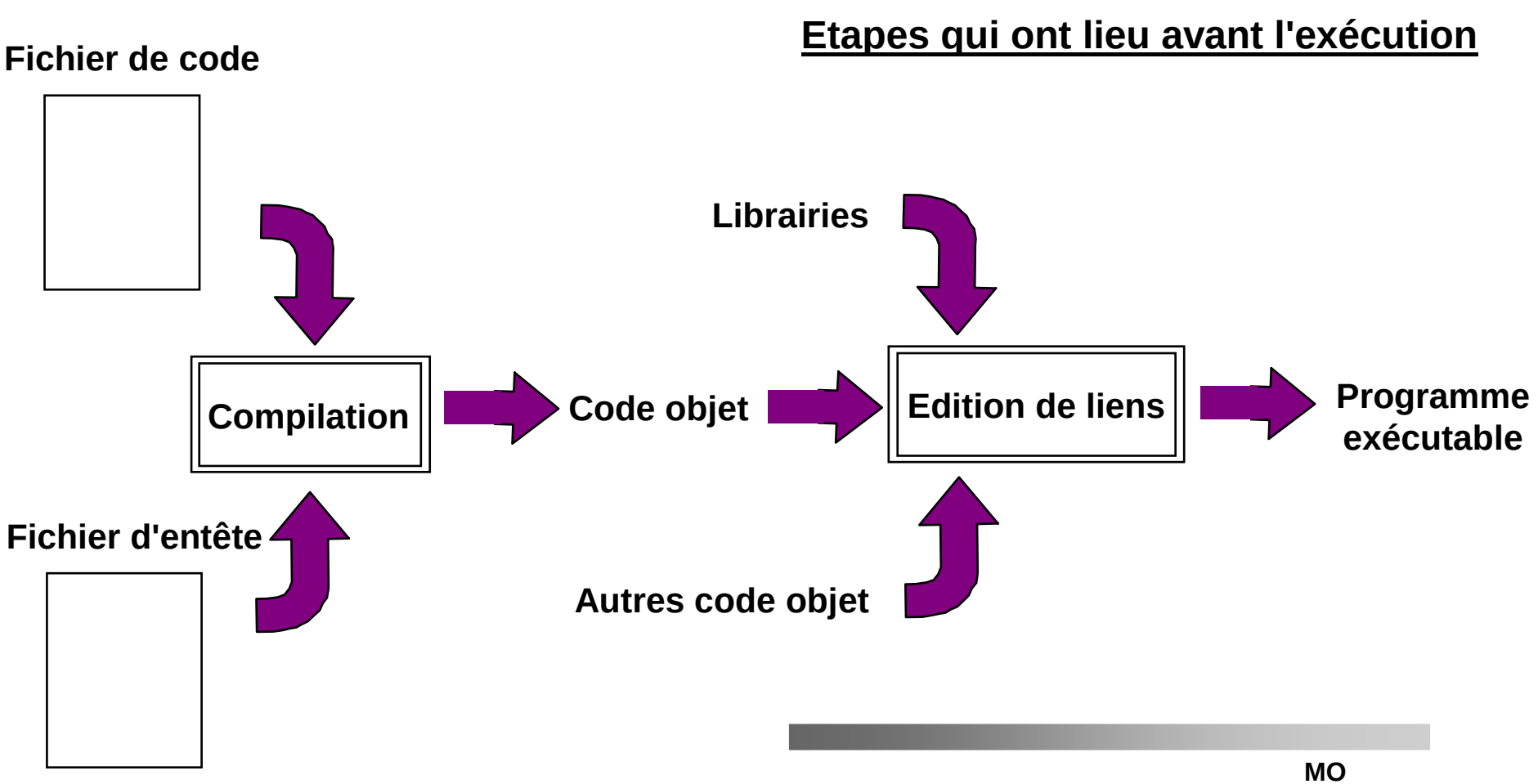
S'affranchir de la plateforme (2)

- **Avantages :**
  - **Portabilité**
    - **Des machines virtuelles Java existent pour de nombreuses plateformes dont : Solaris, Windows, MacOS**
  - **Développement plus rapide**
    - **courte étape de compilation pour obtenir le byte code,**
    - **pas d'édition de liens,**
    - **déboggage plus aisé,**
  - **Le byte-code est plus compact que les exécutables**
    - **pour voyager sur les réseaux.**

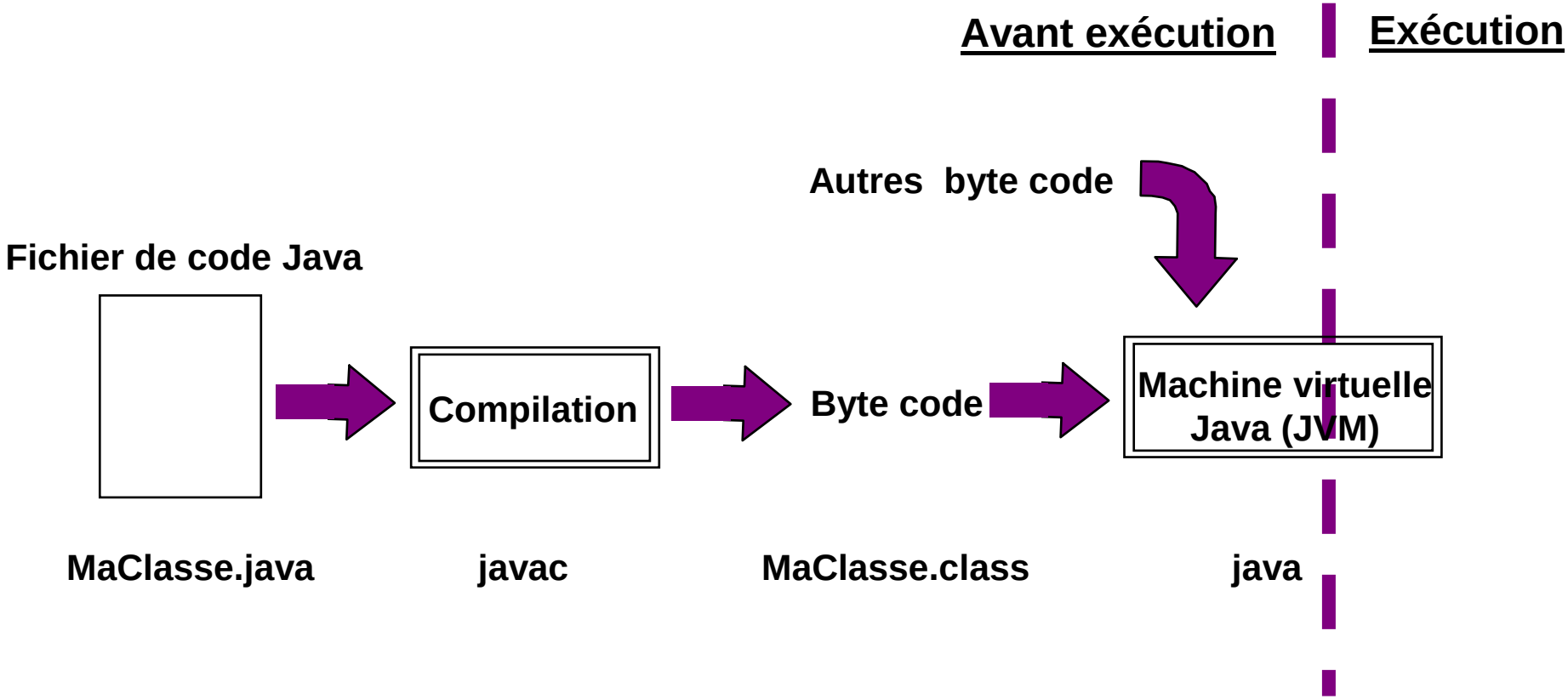
S'affranchir de la plateforme (3)

- **Inconvénients :**
  - **L'interprétation du code ralentit l'exécution**
    - **de l'ordre de quelques dixaines de fois plus lent que C++**
  - **Les applications ne bénéficient que du dénomminateur commun des différentes plateformes**
    - **limitation, par exemple, des interfaces graphiques**

Comparaison : langage compilé / langage interprété (1)



Comparaison : langage compilé / langage interprété (2)



Java ... compilé ?

- Il existe déjà au moins un vrai compilateur Java
  - Java devient bien plus rapide ...
  - ... mais perd quelques qualités (portabilité, etc.)
- Autre technologie (celle préconisée par Sun):
  - La compilation à la volée : "just in time compilers"
  - Le code est compilé au fur et à mesure de sa première exécution et stocké dans un cache pour être ensuite réutilisé tel quel

Java c'est aussi ... une API

- Java fournit de nombreuses librairies de classes remplissant des fonctionnalités très diverses : c'est l'API Java (Application Programming Interface).
- Ces classes sont regroupées, par catégories, en paquetages (ou "packages").

Principaux paquetages Java

- java.lang : chaînes de caractères, interaction avec l'OS, threads
- java.util : structures de données classiques
- java.io : entrées / sorties
- java.net : sockets, URL
- java.applet
- java.awt : fenètres, boutons, événements souris
- java.beans
- java.rmi : Remote Method Invocation pour lancer une méthode d'un objet Java distant

Java c'est aussi ... un langage multi-thread

- Un thread (mini-processus) est une entité d'exécution qui peut se dérouler en parallèle d'autres threads, c'est-à-dire de manière concourante, au sein d'une même application.
- Java permet de lancer plusieurs threads en même temps, sans bloquer les autres.
- Exemple de threads
  - le ramasse-miettes est un thread "système"
  - la gestion de l'interface graphique qui peut être parrallélisée avec l'accès à l'imprimante et l'envoi de données sur le réseau.

La documentation Java

- Elle est standard, que ce soit pour les classes de l'API ou pour les classes utilisateur.
- Elle est au format HTML.
  - intérêt de l'hypertexte pour naviguer dans la documentation
- Pour chaque classe, il y a une page HTML contenant :
  - la hiérarchie d'héritage de la classe,
  - une description de la classe et son but général,
  - la liste des attributs de la classe (locaux et hérités),
  - la liste des constructeurs de la classe (locaux et hérités),
  - la liste des méthodes de la classe (locaux et hérités),
  - puis, chacune de ces trois dernières listes, avec la description détaillée de chaque élément.

L'environnement de développement fourni par Sun

- Il s'appelle le JDK (pour Java Developpment Kit).
- Il contient :
  - les classes de base de l'API java (plusieurs centaines),
  - la documentation au format HTML
  - le compilateur : javac
  - la JVM : java
  - le visualiseur d'applets : appletviewer
  - le générateur de documentation : javadoc
  - etc.

Quoi de neuf dans Java ?

- Java n'est pas un langage novateur : il a puisé ses concepts dans d'autres langages existants et a même imité la syntaxe du C++.
- Cette philosophie permet à Java
  - De ne pas dérouter ses utilisateurs en faisant "presque comme ... mais pas tout à fait"
  - D'utiliser des idées, concepts et techniques qui ont fait leurs preuves et que les programmeurs savent utiliser
- En fait, Java a su faire une synthèse efficace de bonnes idées issues de sources d'inspiration variées
  - Smalltalk, C++, Ada, etc.

Syntaxe du langage Java

MO

Commentaires

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

Les commentaires d'explication placés juste avant une déclaration (d'attribut ou de méthode) indiquent qu'ils doivent être inclus dans la documentation éventuelle générée par l'utilitaire javadoc.

Instructions, blocs, blancs

- Les instructions Java se terminent par un ;
- Les blocs sont délimités par :
  - { pour le début de bloc
  - } pour la fin du bloc
- Les espaces, tabulations, sauts de ligne sont autorisés. Cela permet de présenter un code plus lisible.

Identificateurs

- Les identificateurs commencent par une lettre, _ ou $
  - Attention : Java distingue les majuscules des minuscules
- Conventions de nommage :
  - Si plusieurs mots sont accolés, mettre une majuscule à chacuns des mots sauf le premier
  - La première lettre est majuscule pour les classes et les interfaces
    - exemples : `MaClasse`, `UneJolieFenetre`
  - La première lettre est minuscule pour les méthodes, les attributs, les variables
    - exemples : `setLongueur`, `i`, `uneFenetre`
  - Les constantes sont entièrement en majuscules
    - exemple : `LONGUEUR_MAX`

Mots réservés

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

Types de base

- En Java, tout est objet sauf les types de base.
- Il y a huit types de base :
  - `boolean`
  - `char` et `String`
  - `byte`, `short`, `int` et `long`
  - `float` et `double`
- La taille nécessaire au stockage de ces types est indépendante de la machine.
  - Avantage : portabiité
  - Inconvénient : "conversions" coûteuses

Exemple d'utilisation des types de base

```
int x = 0, y = 0;
float z = 3.1415F;
double w = 3.1415;
long t = 99L;
boolean test = true;
char c = 'a';
String str1 = "Bonjour ! ";
String str2 = null;
str2 = "Comment vas-tu ?";
String str3 = str1 + str2;  //Concatenation de chaînes
```

Remarque importante :
Java exige que toutes les variables soient définies et initialisées. Le compilateur sait déterminer si une variable est succeptible d'être utilisée avant initialisation et produit une erreur de compilation.

Opérateurs

Ce sont, à peu d'exceptions près, les mêmes que ceux de C ou C++.
Le tableau suivant en donne les règles d'associativité de gauche à droite ou de droite à gauche.

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

Structures de contrôle

- Les structures de contrôle classiques existent en Java :
  - `if`, `else`
  - `switch`, `case`, `default`, `break`
  - `for`
  - `while`
  - `do`, `while`
- A utiliser avec parcimonie
  - `<étiquette>` : suivie d'une boucle `for`, `while` ou `do`
  - `break <étiquette>`
  - `continue <étiquette>`

Tableaux (1)

- **Déclaration**

```
int tab [ ];
Point pts [ ];
```

- **Création d'un tableau**

```
tab = new int [20];  // tableau de 20 éléments
                     // de type int
pts = new Point [100]; // tableau de 100
                       // variables de type Point
```

- **Le nombre d'éléments du tableau est stocké. Java peut ainsi détecter à l'exécution le dépassement d'indice et générer une exception.**

Tableaux (2)

- Initialisation
  - ```
    tab[0]=1;
    tab[1]=2;
    ...
    ```
  - ```
    pts[0]=new Point(2,5);
    pts[1]=new Point(4,-1);
    ...
    ```
- Création et initialisation simultanées
  - ```
    String noms[]={"Boule","Bill"};
    Color maPalette[]={ new Color(0, 0, 100),
                        new Color(100, 0, 100)};
    ```
    ```

Tableaux multidimensionnels

- Il est possible de créer des tableaux "rectangulaires" et des tableaux "non rectangulaires".
- Exemples :
  - `int matrice[][] = new int[4][3];`
  - ```
    int tab[][] = new int[4][];
    tab[0] = new int[5];
    tab[1] = new int[8];
    tab[2] = new int[3];
    tab[3] = new int[10];
    ```
    ```

Java et les objets

Classes et objets

Une classe Java ...

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

... et des instances de cette classe

```
Personne jean, pierre;
jean = new Personne ();
pierre = new Personne ();
```

Accès aux attributs

```
jean.nom = "Dupond";
jean.age = 25;
jean.salaire = 10000;
```

Remarque :
Contrairement aux variables, les attributs d'une classe, s'ils ne sont pas initialisés, se voient affecter automatiquement une valeur par défaut.
Cette valeur vaut : `0` pour les variables numériques, `false` pour les booléens, et `null` pour les références.

Objets, tableaux, types de base et références

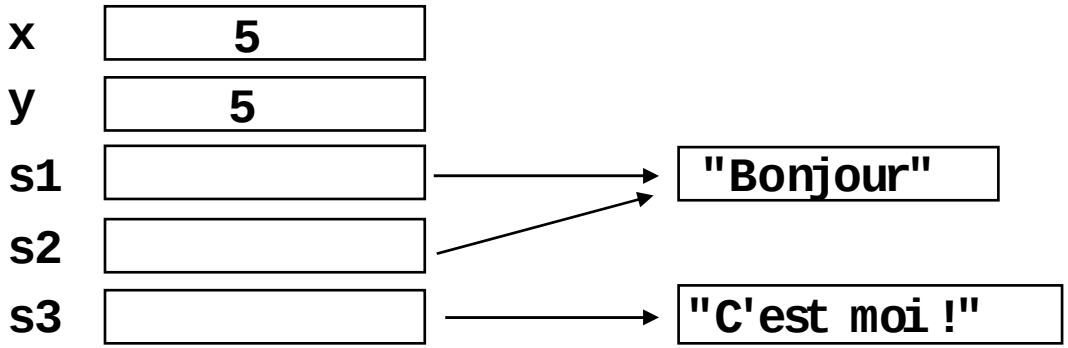
- Lorsqu'une variable est d'un type objet ou tableau, ce n'est pas l'objet ou le tableau lui-même qui est stocké dans la variable mais une référence vers cet objet ou ce tableau (adresse mémoire).
- Lorsqu'une variable est d'un type de base, la variable contient la valeur.

Références (1)

- La référence est, en quelque sorte, un pointeur pour lequel le langage assure une manipulation transparente, comme si c'était une valeur (pas de déréférencement).
- Par contre, du fait qu'une référence n'est pas une valeur, c'est au programmeur de prévoir l'allocation mémoire nécessaire pour stocker effectivement l'objet (utilisation du `new`).

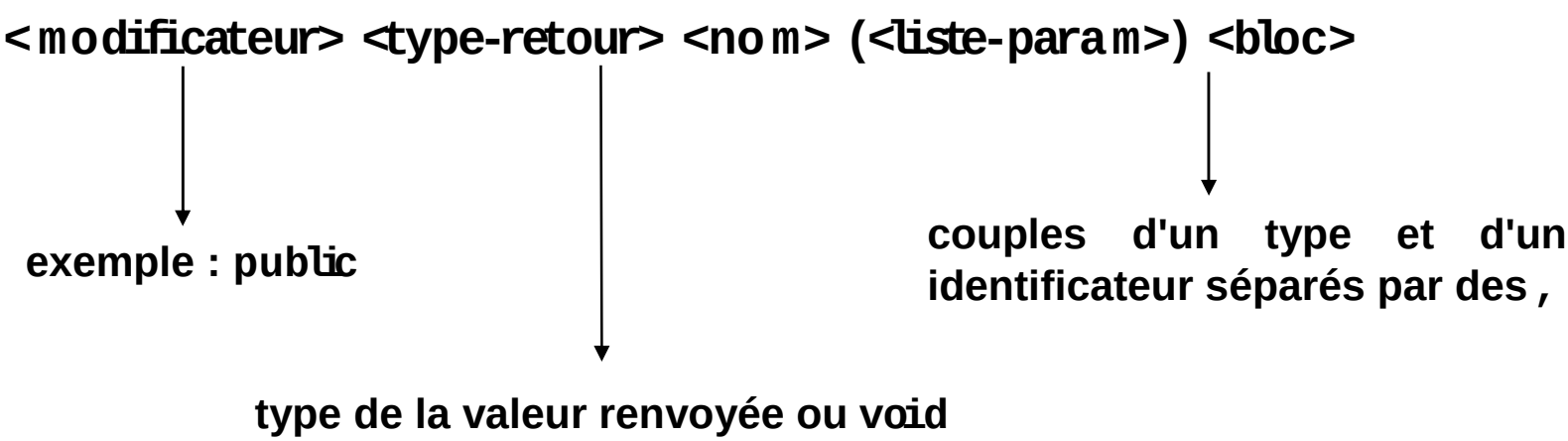
Références (2)

```
int x = 5;
int y = 5;
String s1 = "Bonjour";
String s2 = s1;
String s3 = "C'est moi !";
```



Méthodes

➜ **Déclaration et définition d'une méthode**



Contrôle d'accès (1)

- **Chaque attribut et chaque méthode d'une classe peut être :**
  - **visible depuis les instances de toutes les classes d'une application. En d'autres termes, son nom peut être utilisé dans l'écriture d'une méthode de ces classes. Il est alors public.**
  - **visible uniquement depuis les instances de sa classe. En d'autres termes, son nom peut être utilisé uniquement dans l'écriture d'une méthode de sa classe. Il est alors privé.**
- **Les mots réservés sont :**
  - **`public`**
  - **`private`**

Contrôle d'accès (2)

```
public class Parallelogramme
  {
    private int longueur = 0;  // déclaration + initialisation explicite
    private int largeur = 0;   // déclaration + initialisation explicite
    public int profondeur = 0; // déclaration + initialisation explicite
    public void affiche ()
      {
      System.out.println("Longueur= " + longueur +
                 "Largeur = " + largeur +
                 "Profondeur = " + profondeur);
      }
  }

public class ProgPpal
  {
    public static void main(String args[])
      {
      Parallelogramme p1 = new Parallelogramme();
      p1.longueur = 5;     //invalide car l'attribut est prive
      p1.profondeur = 4;   // OK
      p1.affiche ();       // OK
      }
  }
```

Comment utiliser le contrôle d'accès ?

- **En toute rigueur, il faudrait toujours que :**
  - **les attributs ne soient pas visibles,**
    - **Les attributs ne devraient pouvoir être lus ou modifiés que par l'intermédiaire de méthodes prenant en charge les vérifications et effets de bord éventuels.**
  - **les méthodes "utilitaires" ne soient pas visibles,**
  - **seules les fonctionnalités de l'objet, destinées à être utilisées par d'autres objets soient visibles.**

Constructeurs (1)

- L'appel de `new` pour créer un nouvel objet déclenche, dans l'ordre :
  - L'allocation mémoire nécessaire au stockage de ce nouvel objet et l'initialisation par défaut de ces attributs,
  - L'initialisation explicite des attributs, s'il y a lieu,
  - L'exécution d'un constructeur.
- Un constructeur est une méthode d'initialisation.

Constructeurs (2)

- Lorsque l'initialisation explicite n'est pas possible (par exemple lorsque la valeur initiale d'un attribut est demandée dynamiquement à l'utilisateur), il est possible de réaliser l'initialisation au travers d'un constructeur.
- Le constructeur est une méthode :
  - de même nom que la classe,
  - sans type de retour.
- Toute classe possède au moins un constructeur. Si le programmeur ne l'écrit pas, il en existe un par défaut, sans paramètres, de code vide.

Cas où il y a plusieurs constructeurs

- Pour une même classe, il peut y avoir plusieurs constructeurs, de signatures différentes.
- L'appel de ces constructeurs est réalisé avec le `new` auquel on fait passer les paramètres.
  - Exemple

    ```
    p1 = new Parallelepipede (5, 7, 8);
    ```
- Le déclenchement du "bon" constructeur, se fait en fonction des paramètres passés lors de l'appel (nombre et types). C'est le mécanisme de "lookup".
- Remarque : Si le programmeur crée un constructeur (même si c'est un constructeur avec paramètres), le constructeur par défaut n'est plus disponible. Attention aux erreurs de compilation !

Destruction d'objets (1)

- Java n'a pas repris à son compte la notion de destructeur.
- C'est le ramasse-miettes qui s'occupe de collecter les objets qui ne sont plus référencés.
- Le ramasse-miettes fonctionne en permanence dans un thread de faible priorité. Il est basé sur le principe du compteur de références.
- Ainsi, l'application n'a plus à détruire les objets, ce qui était une importante source d'erreurs ("memory leak").
- Le ramasse-miettes peut être "désactivé" en lançant l'interpréteur `java` avec l'option `-noasyncgc`.
- Inversement, le ramasse-miettes peut être lancé par une application avec l'appel `System.gc();`

Destruction d'objets (2)

- Il est possible au programmeur d'indiquer ce qu'il faut faire juste avant de détruire un objet.
- C'est le but de la méthode `finalize()` de l'objet.
- Cette méthode est utile, par exemple, pour :
  - fermer une base de données,
  - fermer un fichier,
  - couper une connection réseau,
  - etc.

L'héritage en Java

- **Java implémente le mécanisme d'héritage simple qui permet de "factoriser" de l'information dans le cas où deux classes sont reliées par une relation de généralisation / spécialisation.**
- **L'héritage multiple n'existe pas en Java.**
- **Pour le programmeur, il s'agit d'indiquer, dans la sous-classe, le nom de la superclasse.**
- **Mot réservé : extends**

Exemple

```
class Personne
  {
    private String nom;
    private Date date_naissance;
    // ...
  }
class Employe extends Personne
  {
    private float salaire;
    // ...
  }
class Etudiant extends Personne
  {
    private int numero_carte_etudiant;
    // ...
  }
```

Redéfinition de méthodes

- Une sous-classe peut redéfinir des méthodes existant dans une de ses superclasses (directe ou indirectes), à des fins de spécialisation.
- Le terme anglophone est "overriding". On parle aussi de masquage.
- La méthode redéfinie doit avoir la même signature.

```
class Employe extends Personne
  {
    private float salaire;
    public calculePrime()
      {
        // ...
      }
    // ...
  }
```

```
class Cadre extends Employe
  {
    public calculePrime()
      {
        // ...
      }
    // ...
  }
```

Recherche dynamique de la "bonne" méthode

- Le polymorphisme et le mécanisme de "lookup" dynamique permettent, lorsqu'on possède la référence d'un objet, de déclencher la méthode la plus spécifique, c'est-à-dire celle correspondant au type réel de l'objet, déterminé à l'exécution uniquement (et non le type de la référence, seul type connu à la compilation, qui peut être plus générique).
- Cette dynamicité permet d'écrire du code plus générique.

```
Employe jean = new Employe();
jean.calculePrime();
```

```
Employe jean = new Cadre();
jean.calculePrime();
```

Paquetages

- Un paquetage regroupe des classes et des interfaces apportant une même catégorie de fonctionnalités.
- Les classes d'un paquetage sont dans un même répertoire.
  - Exemple
    Les classes de `java.lang` sont dans `.../java/lang/`.
- Pour utiliser une classe située dans un paquetage, il y a deux possibilités :
  - Indiquer le nom du paquetage avant celui de la classe,
  - Utiliser le racourci lexical `import`.
    Précision : `import` ne réalise pas le chargement du paquetage.
- Si aucun paquetage n'est précisé pour une classe, celle-ci est intégré dans le paquetage anonyme (`unnamed`).

Exemples

```
import java.util.Vector; // en debut de fichier

// ...

Vector unVecteur;
```

équivaut à :

```
java.util.Vector unVecteur;
```

La ligne suivante permet d'utliser toutes les classes du paquetage.

```
import java.util.*;    // en debut de fichier
```

Insertion d'une classe dans un paquetage

- Pour indiquer qu'une classe `MaClasse` doit être placée dans un paquetage `mon_paquetage`, il faut ajouter en première ligne du fichier `MaClasse.java` la ligne :
  - `package mon_paquetage;`
- Il faut, en outre, indiquer le chemin au compilateur :
  - `javac -d $HOME/mon_paquetage MaClasse.java`

Classes abstraites (1)

- Il peut être nécessaire au programmeur de créer une classe déclarant une méthode sans la définir (c'est-à-dire sans en donner le code). La définition du code est dans ce cas laissée aux sous-classes.
- Une telle classe est appelée classe abstraite.
- Elle doit être marquée avec le mot réservé abstract.
- Toutes les méthodes de cette classe qui ne sont pas définies doivent elles-aussi être marquées par le mot réservé abstract.

Classes abstraites (2)

- Une classe abstraite ne peut pas être instanciée.
- Par contre, il est possible de déclarer et d'utiliser des variables du type de la classe abstraite.
- Si une sous-classe d'une classe abstraite ne définit pas toutes les méthodes abstraites de ses superclasses, elle est abstraite elle aussi.

```
public abstract class Polygone
  {
    private int nombreCotes = 3;
    public abstract void dessine (); // methode non définie
    public int getNombreCotes()
      {
        return(nombreCotes);
      }
   // ...
  }
```

Interfaces

- Si le programmeur veut s'assurer qu'une certaine catégorie de classes (pas forcément reliées par des relations de généralisation / spécialisation) implémente un ensemble de méthodes, il peut regrouper les déclarations de ces méthodes dans une interface.
- De telles classes pourront ainsi être manipulées de manière identique.
- Les classes désirant appartenir à la catégorie ainsi définie :
  - déclareront qu'elles implémentent cette interface,
  - fourniront le code des méthodes déclarées dans cette interface.
- Mots réservés : `interface`, `implements`.

Exemple (1)

```
interface Conduisible
  {
  void demarrerMoteur();
  void couperMoteur();
  void tourner(float angle);
  }

class Voiture implements Conduisible
  {
  // ...
  void demarrerMoteur() {...}
  void couperMoteur() {...}
  void tourner(float angle) {...}
  }

class TondeuseGazon implements Conduisible
  {
  // ...
  void demarrerMoteur() {...}
  void couperMoteur() {...}
  void tourner(float angle) {...}
  }
```

Exemple (2)

```
// ...

Voiture maVoiture = new Voiture();
TondeuseGazon maTondeuse = new TondeuseGazon();
Conduisible vehicule;
Boolean weekEnd;

// ...

if(weekEnd == true)
  {
    vehicule=maTondeuse;
  }
else
  {
    vehicule=maVoiture;
  }
vehicule.demarrerMoteur();
vehicule.tourner(90.0F);
vehicule.couperMoteur();

// ...
```

Interfaces (suite)

- Toutes les méthodes d'une interface sont abstraites.
- Le modificateur abstract est facultatif.

Classes internes et classes anonymes

- **Java permet de définir des classes imbriquées dans une autre classe. Ces classes, appelées classes internes, n'ont pas d'existence en dehors de la classe dans laquelle elles sont imbriquées.**
- **Il est aussi possible de définir une sous-classe d'une classe en réalisant, avec une seule expression la déclaration de la classe et la création d'une instance. On parle de classe anonyme.**
- **L'usage de ces notions est marginal.**

Surdéfinition de méthodes

- **Dans une même classe, plusieurs méthodes peuvent posséder le même nom, pouvu qu'elles diffèrent en nombre et/ou type de paramètres. On parle de surdéfinition (ou surcharge, en anglais "overloading").**
- **Le choix de la méthode à utiliser est fonction des paramètres passés à l'appel. Ce choix est réalisé de façon statique (c'est-à-dire à la compilation).**
- **Quelques précisions cependant :**
  - **le type de retour seul, ne suffit pas à distinguer deux méthodes de signatures identiques par ailleurs,**
  - **les types des paramètres doivent être "suffisamment" différents pour qu'il n'y ait pas d'ambiguïtés, en particulier avec les conversions de types automatiques (exemple : promotion d'un float en double).**

Mode de passage des paramètres

- **Java n'implémente qu'un seul mode de passage des paramètres à une méthode : le passage par valeur.**
- **Conséquences :**
  - **l'argument passé à une méthode ne peut être modifié,**
  - **si l'argument est une instance, c'est sa référence qui est passée par valeur. Ainsi, le contenu de l'objet peut être modifié, mais pas la référence elle-même.**

Variables de classe

- Il peut s'avérer nécessaire de définir un attribut dont la valeur soit partagée par toutes les instances d'une classe. On parle de variable de classe.
- Ces variables sont, de plus, stockées une seule fois, pour toutes les instances d'une classe.
- Mot réservé : static.
- Accès :
  - depuis une méthode de la classe comme pour tout autre attribut,
  - via une instance de la classe,
  - à l'aide du nom de la classe.

Exemple

```
public class UneClasse
  {
    public static int compteur = 0;
    public UneClasse ()
      {
        compteur++;
      }
  }

public class AutreClasse
  {
    public void uneMethode()
      {
        int i = UneClasse.compteur;
      }
  }
```

Méthodes de classe

- Il peut être nécessaire de disposer d'une méthode qui puisse être appelée sans instance de la classe. C'est une méthode de classe.
- On utilise là aussi le mot réservé static.
- Puisqu'une méthode de classe peut être appelée sans même qu'il n'existe d'instance, une méthode de classe ne peut pas accéder à des attributs non statiques. Elle ne peut accéder qu'à ses propres variables et à des variables de classe.

Exemples

```
public class UneClasse
  {
    public int unAttribut;
    public static void main(String args[])
      {
        unAttribut = 5;   // Erreur de compilation
      }
  }
```

Java utilise souvent les méthodes de classe.

Exemples :

```
int Integer.parseInt (String);
String String.valueOf (int);
```

Opérateur instanceof

➜ L'opérateur instanceof confère aux instances une capacité d'introspection : il permet de savoir si une instance est instance d'une classe donnée.

```
if (...) Personne jean = new Etudiant();
else     Personne jean = new Employe();
if (jean instanceof Employe) //discuter affaires
else                         // proposer un stage
```

Forçage de type (cast)

- **Lorsqu'une référence du type d'une classe contient une instance d'une sous-classe, il est nécessaire de forcer le type de la référence pour accéder aux attributs spécifiques à la sous-classe.**
- **Si ce n'est pas fait, le compilateur ne peut déterminer le type réel de l'instance, ce qui provoque une erreur de compilation.**

Exemple

```
class Personne
  {
    private String nom;
    private Date date_naissance;
    // ...
  }
class Employe extends Personne
  {
    public float salaire;
    // ...
  }


Personne jean = new Employe ();
float i = jean.salaire;              // Erreur de compilation
float j = ( (Employe) jean ).salaire;   // OK
```

Autoréférence this

- Le mot réservé `this`, utilisé dans une méthode, désigne la référence de l'instance sur laquelle a été déclenchée la méthode.
- Il est utilisé principalement :
  - lorsqu'une référence à l'instance courante doit être passée en paramètre à une méthode,
  - pour lever une ambiguïté,
  - dans un constructeur, pour appeler un autre constructeur de la même classe.

Exemples d'utilisation de this

```
class Personne
  {
    public String nom;
    Personne (String nom)
      {
        this.nom=nom;
      }
  }
```

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

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

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

Référence à la superclasse - Mot réservé super

➜ **Le mot réservé super permet de faire référence aux informations provenant de la superclasse directe.**

```
class Employe extends Personne
  {
    private float salaire;
    public float calculePrime()
      {
        return (salaire * 0,05);
      }
    // ...
  }

class Cadre extends Employe
  {
    public calculePrime()
      {
        return (super.calculePrime() / 2);
      }
    // ...
  }
```

Autre exemple

➜ **super peut permettre d'appeler un constructeur de la superclasse directe.**

```
class Employe extends Personne
  {
    private float salaire;
    public Employe (String unNom, float unSalaire)
      {
        super(unNom);
        salaire = unSalaire;
      }
    // ...
  }
```

➜ **Remarques :**

- ➢ **Cette instruction doit être la première instruction du constructeur.**
- ➢ **Si la première ligne d'un constructeur n'est pas cette instruction, le compilateur exécute par défaut super(); c'est-à-dire déclenche le constructeur sans paramètre de la superclasse.**

Mot réservé final

- Une classe est déclarée finale lorsqu'on ne souhaite pas qu'elle puisse être sous-classée.
  - Exemple
    ```
    public final class Integer extends Number { ... }
    ```
- Une méthode est déclarée finale lorsqu'on ne souhaite pas qu'elle puisse être redéfinie.
  - Exemple
    ```
    public final String getWarningString() { ... }
    ```
- Une variable déclarée finale est une constante.
  - Exemple
    ```
    public static final Color white=new Color(255, 255, 255);
    ```

Contrôle d'accès évolué (1)

- Outre `private` et `public`, Java définit deux niveaux de contrôle d'accès supplémentaires.
- S'il n'y a pas de modificateur explicite dans la déclaration d'un attribut ou d'une méthode, celui-ci est visible depuis toute méthode d'une classe quelconque du même paquetage.
  - On l'appelle accès "friendly".
  - Pas de mot réservé.
- Un attribut ou une méthode protégé est visible depuis :
  - Toute méthode d'une classe quelconque appartenant au même paquetage,
  - Toute méthode d'une sous-classe.
  - Mot réservé : `protected`.

Contrôle d'accès évolué (2)

- **Une classe ou une interface peut être publique (c'est-à-dire accessible partout où son paquetage l'est) ou friendly (c'est-à-dire accessible seulement dans son paquetage).**

Les exceptions

Exceptions : Terminologie

- Une exception est un signal qui indique que quelque chose d'exceptionnel est survenu en cours d'exécution.
- Deux solutions alors :
  - laisser le programme se terminer avec une erreur,
  - essayer, malgré l'exception, de continuer l'exécution normale.
- Lancer une exception consiste à signaler quelque chose d'exceptionnel.
- Capturer l'exception consiste à signaler qu'on va la traiter.

Quelques exceptions préexistant dans Java

- Division par zéro pour les entiers : `ArithmeticException`
- Déréférencement d'une référence nulle : `NullPointerException`
- Tentative de forçage de type illégale : `ClassCastException`
- Tentative de création d'un tableau de taille négative : `NegativeArraySizeException`
- Dépassement de limite d'un tableau : `ArrayIndexOutOfBoundsException`

Des exceptions pour écrire un code fiable

- Java exige qu'une méthode succeptible de lever une exception (hormis les `Error` et les `RuntimeException`) indique quelle doit être l'action à réaliser.
  - Sinon, il y a erreur de compilation.
- Le programmeur a le choix entre :
  - écrire un bloc `try / catch` pour traiter l'exception,
  - laisser remonter l'exception au bloc appelant grâce à un `throws`.
- C'est ce qu'on appelle : "Déclarer ou traiter".

try / catch / finally

```
try
  {
    ...
  }
catch (<une-exception>)
  {
    ...
  }
catch (<une_autre_exception>)
  {
    ...
  }

...

finally
  {
    ...
  }
```

➜ Autant de blocs catch que l'on veut.

➜ Bloc finally facultatif.

Traitement des exceptions

- Le bloc try est exécuté jusqu'à ce qu'il se termine avec succès ou bien qu'une exception soit levée.
- Dans ce dernier cas, les clauses catch sont examinées l'une après l'autre dans le but d'en trouver une qui traite cette classe d'exceptions (ou une superclasse).
- Les clauses catch doivent donc traiter les exceptions de la plus spécifique à la plus générale.
  - La présence d'une clause catch qui intercepte une classe d'exceptions avant une clause qui intercepte une sous-classe d'exceptions déclenche une erreur de compilation.
- Si une clause catch convenant à cette exception a été trouvée et le bloc exécuté, l'exécution du programme reprend son cours.

Traitement des exceptions (2)

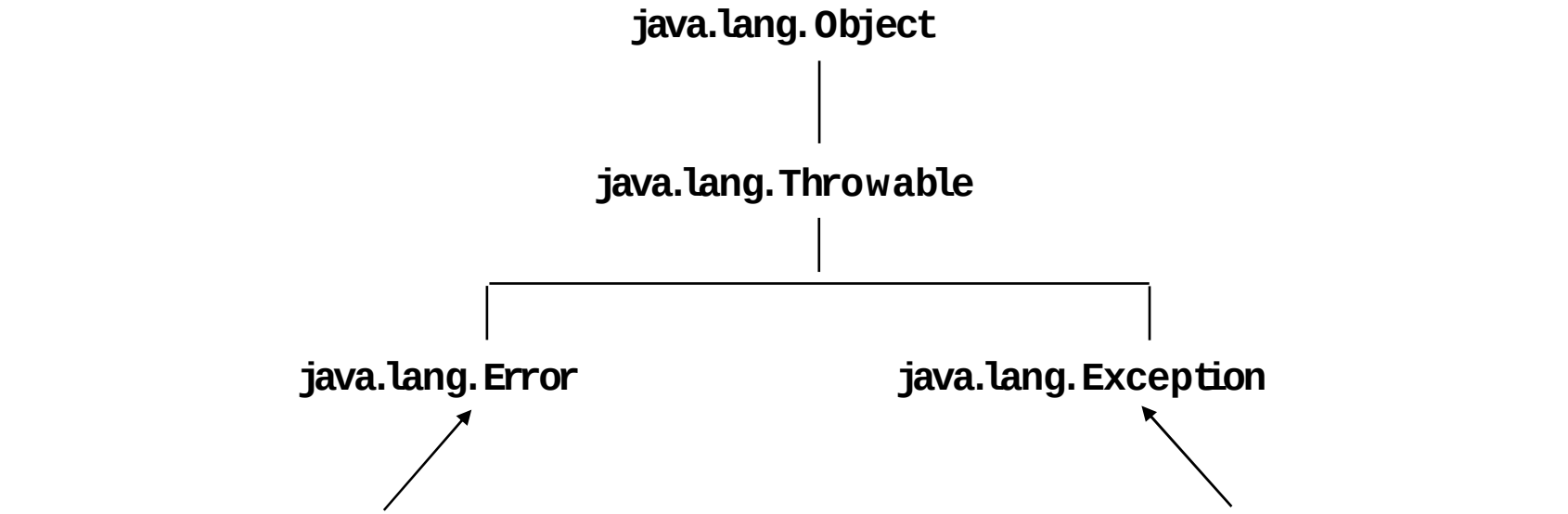
- Si elles ne sont pas immédiatement capturées par un bloc catch, les exceptions se propagent en remontant la pile d'appels des méthodes, jusqu'à être traitées.
- Si une exception n'est jamais capturée, elle se propage jusqu'à la méthode `main()`, ce qui pousse l'interpréteur Java à afficher un message d'erreur et à s'arrêter.
- L'interpréteur Java affiche un message identifiant :
  - l'exception,
  - la méthode qui l'a causée,
  - la ligne correspondante dans le fichier.

Bloc finally

- L'écriture d'un bloc `finally` permet au programmeur de définir un ensemble d'instructions qui est toujours exécuté, que l'exception soit levée ou non, capturée ou non.
- Le bloc `finally` s'exécute même si le bloc en cours d'exécution (`try` ou `catch` selon les cas) contient un `return`, un `break` ou un `continue`. Dans ce cas, le bloc `finally` est exécuté juste avant le branchement effectué par l'une de ces instructions.
- La seule instruction qui peut faire qu'un bloc `finally` ne soit pas exécuté est `System.exit()`.

Les objets Exception

➜ **Dans Java, il existe deux types d'exceptions, représentées par deux classes de l'API Java.**



Les objets Exception

- La classe `Throwable` définit un message de type `String` qui est herité par toutes les classes d'exception.
- Ce champ est utilisé pour stocker le message décrivant l'exception.
- Il est positionné en passant un argument au constructeur.
- Ce message peut être récupéré par la méthode `getMessage()`.

Exemple

```
public class MonException extends Exception
  {
  public MonException()
    {
      super();
    }
  public MonException(String s)
    {
      super(s);
    }
  }
```

Levée d'exceptions

- Le programmeur peut lever ses propres exceptions l'aide du mot réservé `throw`.
- `throw` prend en paramètre un objet instance de Throwable ou d'une de ses sous-classes.
- Les objets exception sont souvent alloués dans l'instruction même qui assure leur lancement.

```
throw new MonException("Mon exception s'est produite !!!");
```

throws (1)

➜ **Pour "laisser remonter" à la méthode appelante une exception qu'il ne veut pas traiter, le programmeur rajoute le mot réservé `throws` à la déclaration de la méthode dans laquelle l'exception est succeptible de se manifester.**

```
public void uneMethode() throws IOException
  {
    // ne traite pas l'exception IOException
    // mais est succeptible de la générer
  }
```

throws (2)

- Les programmeurs qui utilisent une méthode connaissent ainsi les exceptions qu'elle peut lever.
- La classe de l'exception indiquée peut tout à fait être une super-classe de l'exception de l'exception effectivement générée.
- Une même méthode peut tout à fait "laisser remonter" plusieurs types d'exceptions (séparés par des ,).
- Une méthode doit traiter ou "laisser remonter" toutes les exceptions qui peuvent être générées dans les méthodes qu'elle appelle (et ceci récursivement).

Applets Java

MO

Qu'est-ce qu'une applet ?

- ➜ **Une applet Java est une mini-application qui s'exécute dans un environnement navigateur.**
- ➜ **L'exécution des applets se fait sur la machine client.**
  - ➢ **Avantages :**
    - **plus rapide pour le client,**
    - **moins coûteux pour le serveur.**
- ➜ **Une applet n'est pas lancée directement par l'intermédiaire d'une commande, comme le serait un programme "usuel".**
- ➜ **Une applet est lancée par l'intermédiaire d'un fichier HTML indiquant au navigateur l'applet à exécuter.**
  - ➢ **Lors de la lecture de la page HTML par le navigateur, en local ou grâce à son URL, l'applet est lancée.**

Intégration d'une applet dans une page HTML

- HTML dispose d'un "tag" <applet> </applet> destiné à intégrer des applets dans une page web.
- Exemple

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

- Si Java n'est pas supporté, le tag <applet> est ignoré. Sinon, c'est le code HTML "régulier" qui est entre <applet> et </applet> qui est ignoré.

Syntaxe complète du tag <APPLET>

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

Syntaxe complète

- **Description :**
  - **code : nom du fichier contenant l'applet compilée.**
  - **width et heigth : largeur et la hauteur initiales de la zone d'affichage de l'applet.**
  - **codebase :URL de base de l'applet . Par défgaut, c'est l'URL du document HTML lu.**
  - **alt : texte à afficher si le navigateur comprend le paramètre applet mais ne réussit pas à exécuter les applets Java.**
  - **name : nom pour l'instance de l'applet.**
  - **align : alignement de l'applet (left, top, middle, etc.)**
  - **vspace et hspace : espace au-dessus et en dessous et de chaque côté de l'applet (hspace).**

Ecriture d'une applet

- Une applet est une classe Java :
  - sous-classe de la classe `java.applet.Applet`,
  - publique,
  - stockée dans un fichier de même nom que la classe suivi de l'extension `.java`
- Exemple :

```
import java.applet.*;
public class HelloWorld extends Applet
  {
    public void paint (Graphics gc)
      {
        gc.drawString ("Hello World !", 125,95);
      }
  }
```

Méthodes d'une applet

- Dans une application Java, l'exécution du programme commence par l'exécution de la méthode `main()`.
- Dans le cas d'une applet, après le déclenchement de son constructeur par défaut, des méthodes prédéfinies se déclenchent :
  - `init()`
  - `start()`
  - `paint()`
  - `stop()`
  - `destroy()`

Squelette d'une applet

```
public class MonApplet extends Applet
  {
    public void init ()
      {
            ...
        }
      public void start ()
    {
            ...
        }
      public void stop ()
    {
            ...
        }
      public void destroy ()
    {
              ...
        }
   }
```

init()

- Cette méthode est appelée une seule fois après l'instanciation de l'applet pour effectuer :
  - les initialisations,
  - l'analyse des paramètres,
  - la construction de l'interface utilisateur,
  - le chargement des ressources.
- C'est habituellement le rôle du constructeur.
  - Pour une applet, le constructeur est appelé avant que l'applet ait accès à certaines ressources, comme les informations relatives à l'environnement.
  - Une applet ne fournit donc généralement pas de constructeur autre que celui par défaut de la classe Applet.

start()

- Cette méthode est appelée à chaque fois que l'applet devient visible.
  - lorsque l'applet est affichée pour la première fois,
  - lorsque l'applet apparaît lors du défilement d'une fenêtre,
  - lorsque le navigateur est restauré après avoir été iconisé
  - lorsque le navigateur revient dans la page contenant l'applet après être passé dans une autre URL.
- La méthode start() indique à l'applet qu'elle est active.
  - L'applet peut donc utiliser cette méthode pour démarrer une animation ou jouer des sons.

stop()

- Cette méthode est appelée lorsque l'applet devient invisible. Cette situation se produit :
  - lorsque le navigateur est icônisé,
  - lorsque l'applet est déplacée à l'extérieur de l'écran,
  - lorsque le navigateur change de page.
- La méthode stop() est utilisée pour indiquer à l'applet qu'elle doit se mettre en veille.
  - L'applet peut utiliser cette méthode pour arrêter une animation ou un son.
- Les méthodes start() et stop() forment une paire.
  - start() peut servir à déclencher un comportement et stop() à arréter ce comportement.

destroy()

- Cette méthode est appelée juste avant que l'applet ne soit détruite.
- En ce sens, c'est un peu la méthode `finalize()` des applets.

Affichage d'une applet

- Les applets sont essentiellement graphiques par nature.
- Il est possible de dessiner sur l'affichage d'une applet grâce à la méthode `paint ()`.
- La méthode `paint ()` est appelée par le navigateur chaque fois que l'affichage a besoin d'être régénéré, par exemple, lorsque la fenêtre du navigateur est affichée après avoir été iconisée.
- L'affichage est contrôlé par l'environnement et non par le programme. Il s'effectue donc de manière asynchrone.

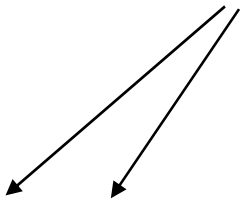
Méthode paint() et graphiques

- La méthode paint() prend en argument une instance java.awt.Graphics : le contexte graphique de l'applet.
  - Cela correspond à la zone de l'écran que l'applet peut utiliser pour dessiner ou écrire du texte.
- Exemple

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

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

Coordonnées de la base du texte



Méthode paint() et graphiques

- La classe `Applet` fait partie du paquetage `awt`.
  - En particulier, `Applet` est une sous-classe de `Component`.
- La mise à jour de l'affichage d'une applet est effectuée de façon asynchrone par un thread qui peut être appelé pour gérer deux situations liées à la mise à jour de l'affichage :
  - L'exposition : une partie de l'affichage a été endommagée et doit être remplacée.
  - L'affichage d'un nouveau contenu avec suppression de l'ancienne image dans un premier temps.
- Le traitement de l'exposition est automatique et déclenche la méthode `paint()`.
  - Une fonction de la classe `Graphics` appelée `clipRect()` sert à limiter l'affichage à la zone endommagée.

Paramétrer une applet

- Il peut être utile d'écrire une applet qui prenne des paramêtres.
  - nom des sons à jouer, nom des images à afficher, etc.
- Ainsi, la même applet pourra être réutilisée d'un contexte à un autre.
- Les paramêtres de lancement d'une applet paramétrée sont fixés par le fichier HTML qui lance l'applet.

Exemple d'applet paramétrée

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

public class MonAppletParam extends Applet
  {
    String leMessage;
      public void init()
        {
        leMessage = getParameter ("message");
           }
         public void paint (Graphics gc)
           {
        gc.drawString (leMessage, 125, 95);
           }
   }
```

Lancement d'une applet avec paramêtres

➜ **Dans le fichier HTML :**

```
<applet code=MonAppletParam.class width=300 heigth=200>
<param name="message" value="Hello World !">
</applet>
```

nom du paramêtre, repris comme argument du `getParameter()`

valeur du paramêtre

Accès, depuis une applet, aux ressources de l'environnement

- La classe `Applet` définit les méthodes suivantes :
  - `public URL getDocumentBase ()` : retourne l'URL de base du document où se trouve l'applet.
  - `public URL getCodeBase ()` : retourne l'URL du fichier de la classe applet.
  - Ces méthodes peuvent être utilisées par l'applet pour construire des URL relatives à partir desquelles elle peut charger d'autres ressources comme des données, des images ou des sons.
    - Remarque : Pour des raisons de sécurité, on ne peut que descendre dans les arborescences de fichiers (jamais remonter "plus haut" que l'adresse de la page servie).
- La méthode `getImage(...)` permet de récupérer une image, `getAudioClip(...)` permet de récupérer un son, etc...

Exemple d'applet : Test d'image simple

```
// Extension de HelloWorld pour afficher une image
// Suppose l'existence de "graphics/joe.surf.yellow.small.gif"

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

public class AfficheImage extends Applet
  {
    Image duke;
    public void init ()
      {
        duke = getImage (getDocumentBase (),
                "graphics/joe.surf.yellow.smallgif");
      }

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

Test d'image simple

- Les arguments utilisés avec la méthode `drawImage ()` sont :
  - L'objet `Image` à dessiner,
  - L'abscisse,
  - L'ordonnée,
  - L'observateur d'image.
    - Un observateur d'image correspond à la classe qui doit être référencée.
- La méthode `getImage ()` ne fait que créer une référence sur l'image à charger. L'image est chargée par `drawImage()`.
- Le chemin de l'image est indiqué relativement à la valeur retournée par `getDocumentBase()`.

Ecouter un clip audio

- La façon la plus simple d'écouter un clip audio consiste à utiliser la méthode d'applet play() :

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

- ou plus simplement :

```
play (URL soundURL);
```

- Si le fichier son est situé dans le même répertoire que le fichier HTML, on peut utiliser getDocumentBase () :

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

Exemple : Test audio simple

```
// Extension de HelloWorld pour restituer un son audio
// Suppose l'existence du fichier "sounds/cuckoo.au"
//

import java.awt.Graphics;
import java.applet.Applet ;

public class TestAudio extends Applet
  {
      public void paint (Graphics g)
      {
          g.drawString ("Audio Test", 25, 25);
          play (getDocumentBase (), "sounds/cuckoo.au");
        }
  }
```

Autres opérations sur les clips audio

- **Chargement d'un clip audio**

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

- **Exécution d'un clip audio**
  - Pour jouer le clip audio chargé :
    `sound.play ();`
  - Pour démarrer l'écoute du clip et l'écouter en boucle :
    `sound.loop ();`

- **Arrêt d'un clip audio**
  - Pour arrêter un clip audio en cours d'écoute :
    `sound.stop ();`

Exemple de boucle

```
// Extension de HelloWorld pour générer une boucle d'une piste
// audio - Suppose l'existence de "sounds/cuckoo.au"

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

public class SonEnBoucle extends Applet
{
  AudioClip sound;
    public void init ()
    {
    sound =getAudioClip(getDocumentBase(),"sounds/cuckoo.au);
      }
  public void paint (Graphics g)
    {
      g.drawString ("Audio Test", 25, 25);
    }
```

Test simple de boucle (suite)

```
public void start ()
  {
    sound.loop ();
  }

public void stop ()
  {
    sound.stop ();
  }
}
```

L'interface graphique awt

(abstract windowing toolkit)

Conteneurs et composants

- Les interafaces graphiques Java sont construites grâce aux notions de conteneur et de composant.
- Un composant est une partie "visible" de l'interface utilisateur Java.
  - Exemple : les fenêtres, les zones de dessin, les boutons, etc.
  - Ce sont des sous-classes de la classe abstraite `java.awt.Component`.
- Un conteneur est un espace dans lequel on peut positionner plusieurs composants.
  - Un conteneur est lui même un composant, ce qui implque qu'on peut imbriquer des conteneurs.
  - La méthode add() de la classe `Container` permet d'ajouter un composant à un conteneur.

Gestionnaire de présentation

- Un gestionnaire de présentation contrôle le placement et la taille des composants à l'intérieur de la zone d'affichage d'un conteneur.
- Tout conteneur possède un gestionnaire de présentation par défaut.
  - Tout instance de `Container` référence une instance de `LayoutManager`.
  - Il est possible d'en changer grâce à `setLayout()`.
- Le ré-agencement des composants dans un conteneur a lieu lors de :
  - la modification de sa taille,
  - le changement de la taille ou le déplacement d'un des composants.
  - l'ajout, l'affichage, la suppression ou le masquage d'un composant.

Un premier exemple

```
import java.awt.*;

public class ExempleIHM extends Frame
  {
      private Button b1;
      private Button b2;
    public static void main (String args [])
     {
            ExempleIHM that = new ExempleIHM ();
            that.pack (); // change taille du Frame pour englober boutons
       that.setVisible (true) ;
           }
      public ExempleIHM()
     {
            super("Notre exemple d'IHM"); // lance le constr. de Frame
            setLayout (new FlowLayout ()); //nveau gestionnaire pres.
            b1 = new Button ("Appuyer");
            b2 = new Button ("Ne pas appuyer");
            add (b1);
            add (b2);
          }
    }
```

Variante du premier exemple

```
import java.awt.*;

public class ExempleIHM
  {
    public Frame f;
       private Button b1;
       private Button b2;
    public static void main (String args [])
      {
             ExempleIHM that = new ExempleIHM ();
             that.f.pack (); //change taille du Frame pour englober boutons
        that.f.setVisible (true) ;
            }
      public ExempleIHM()
     {
             f=new Frame("Notre exemple d'IHM");
             f.setLayout (new FlowLayout ()); // nveau gestionnaire pres.
             b1 = new Button ("Appuyer");
             b2 = new Button ("Ne pas appuyer");
             f.add (b1);
             f.add (b2);
           }
    }
```

Plusieurs gestionnaires de présentation

- **Java fournit plusieurs gestionnaires de présentation, c'est-à-dire plusieurs politiques de positionnement et de redimensionnement des composants dans un conteneur :**
  - **`FlowLayout`,**
  - **`BorderLayout`,**
  - **`GridLayout`,**
  - **`GridBagLayout`,**
  - **`CardLayout`.**

Remarque :

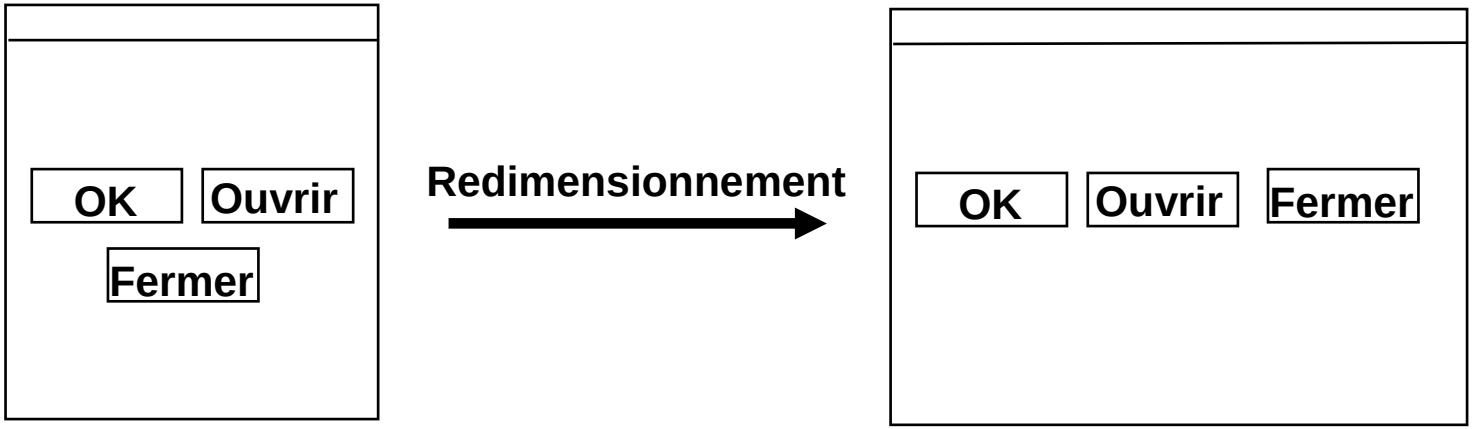
Une fois installé, un gestionnaire fonctionne "tout seul" en interagissant avec le conteneur. Il est donc généralement inutile de garder une référence sur un gestionnaire de présentation.

Exception : `CardLayout` et `GridBagLayout`.

FlowLayout (1)

- La présentation FlowLayout positionne les composants ligne par ligne.
  - Chaque fois qu'une ligne est remplie, une nouvelle ligne est commencée.
- Le gestionnaire FlowLayout n'impose pas la taille des composants mais leur permet d'avoir la taille qu'ils préfèrent.
- Un FlowLayout peut spécifier :
  - une justification à gauche, à droite ou centrée,
  - un espacement horizontal ou vertical entre deux composants.
  - Par défaut, les composants sont centrés à l'intérieur de la zone qui leur est allouée.

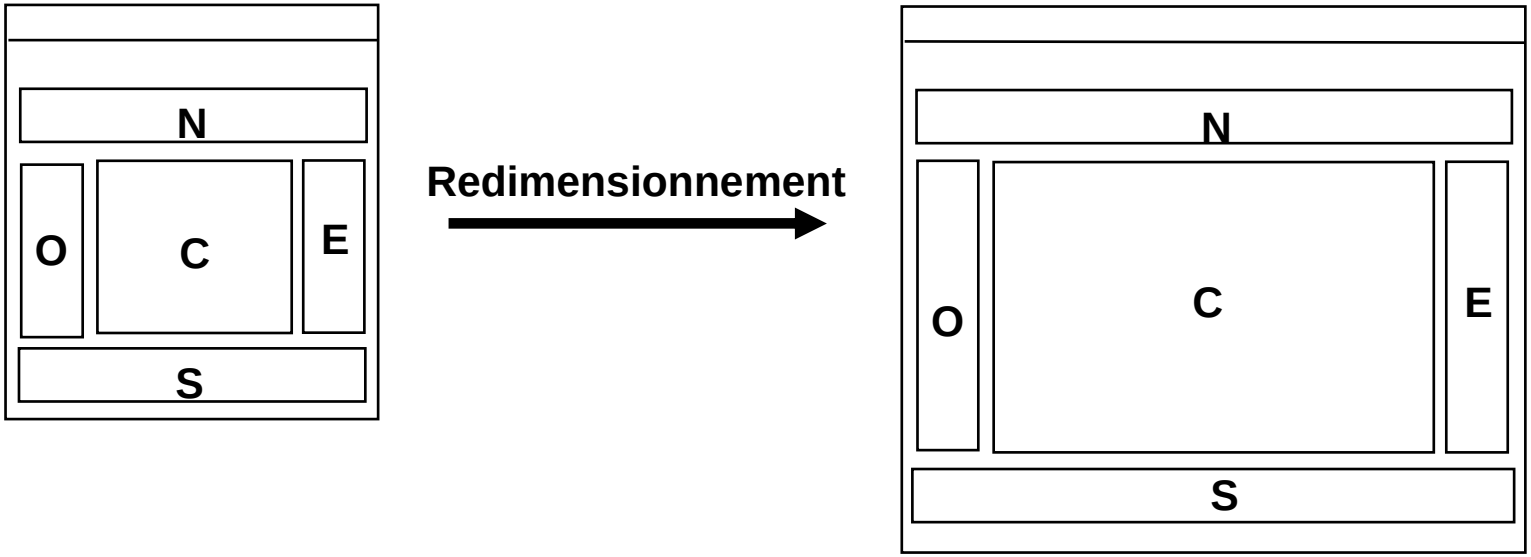
FlowLayout (2)



BorderLayout (1)

- BorderLayout divise son espace de travail en cinq zones géographiques : North, South, East, West et Center.
- Les composants sont ajoutés par nom à ces zones (un seul composant par zone).
  - Exemple
    `add("North", new Button("Le bouton nord !"));`
  - Si une des zones de bordure ne contient rien, sa taille est 0.
- Lors du redimensionnement, le composant est lui-même redimensionné en fonction de la taille de la zone, c-à-d :
  - les zones nord et sud sont éventuellement élargies mais pas alongées.
  - les zones est et ouest sont éventuellement alongées mais pas élargies,
  - la sone centrale est étirée dans les deux sens.

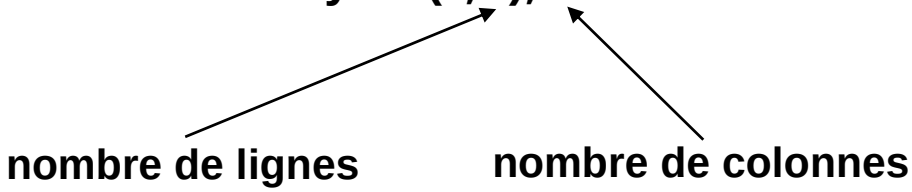
BorderLayout (2)



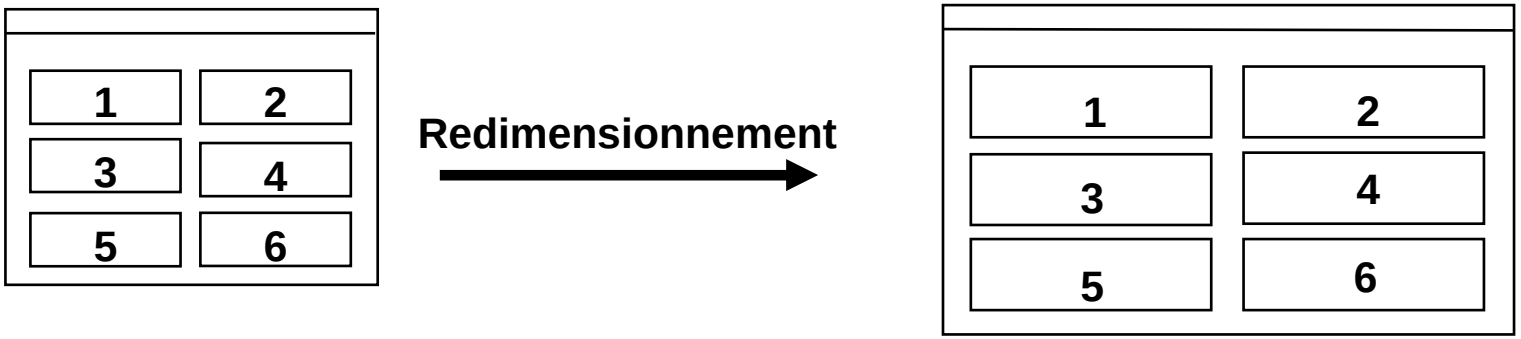
GridLayout(1)

- Le gestionnaire `GridLayout` découpe la zone d'affichage en lignes et en colonnes qui définissent des cellules de dimensions égales.
  - Lorsqu'on ajoute des composants, les cellules sont remplies de gauche à droite et de haut en bas.
  - Lors du redimensionnement les composants changent tous de taille mais leurs positions relatives ne changent pas.

Construction d'un `GridLayout` : `new GridLayout(3,2);`



GridLayout(2)



Autres gestionnaires de présentation

- Le gestionnaire `GridBagLayout` fournit des fonctions de présentation complexes
  - basées sur une grille,
  - permettant à des composants simples de prendre leur taille préférée au sein d'une cellule, au lieu de remplir toute la cellule.
  - Il permet aussi l'extension d'un même composant sur plusieurs cellules.
- La présentation `CardLayout` permet à plusieurs présentations de partager le même espace d'affichage de telle sorte que seule l'une d'entre-elles soit visible à la fois.

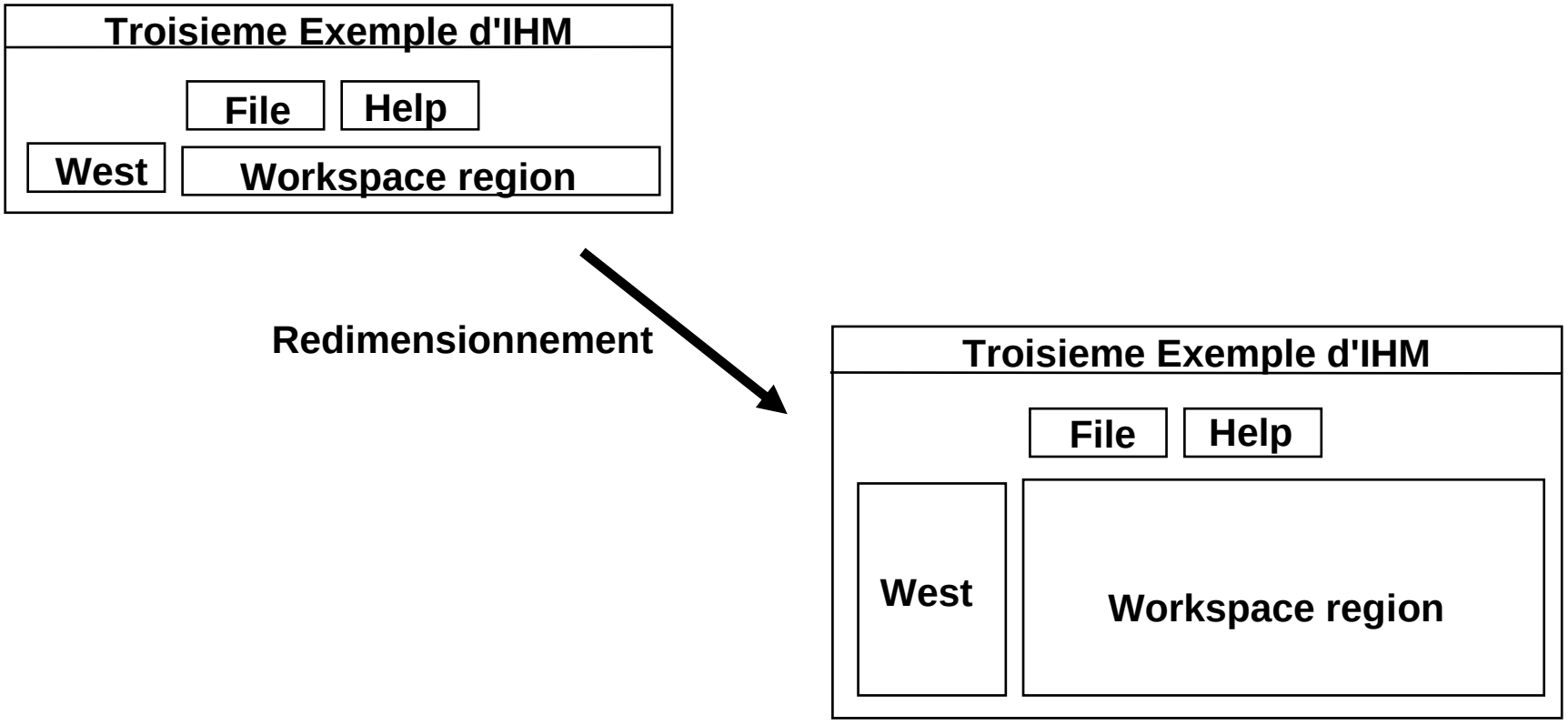
Conteneurs

- Les deux conteneurs les plus courants sont `Frame` et `Panel`.
- Un `Frame` présente une fenêtre de haut niveau avec un titre, une bordure et des angles de redimensionnement.
  - Un `Frame` est doté d'un `BorderLayout` par défaut.
  - La plupart des applications utilisent au moins un `Frame` comme point de départ de leur interface graphique.
- Un `Panel` n'a pas une apparence propre et ne peut pas être utilisé comme fenêtre autonome.
  - Les `Panel` sont créés et ajoutés aux autres conteneurs de la même façon que les composants tels que les boutons.
    - Les `Panel` peuvent ensuite redéfinir une présentation qui leur soit propre pour contenir eux-mêmes d'autres composants.
  - Un `Panel` est doté d'un `FlowLayout` par défaut.

Exemple d'interface intégrant un Panel (1)

```
import java.awt.*;
public class ExempleIHM3
  {
    private Frame f;
       private Panel p;
       private Button bw, bc;
       private Button bfile, bhelp;
       public static void  main (String [] args)
      {
           ExempleIHM3 that = new ExempleIHM3();
              that.go ();
            }
    public void go ()
      {
              f = new Frame ("Troisieme Exemple d'IHM");
              bw = new Button ("West");
              bc = new Button ("Work space region");
              f.add ("West", bw);
              f.add ("Center", bc);
              p = new Panel ();
              f.add ("North", p);
              bfile = new Button ("File");
              bhelp = new Button ("Help");
              p.add (bfile);
              p.add (bhelp);
              f.pack ();
              f.setVisible (true);
           }
  }
```

Exemple d'interface intégrant un `Panel` (2)



Evénements graphiques

Evénement graphique

- **Lorsque l'utilisateur effectue une action au niveau de l'interface utilisateur (click souris, sélection d'un item, etc), un événement graphique est émis.**
- **Lorsqu'un événement se produit, ce dernier est reçu par le composant avec lequel l'utilisateur interagit (par exemple un bouton, un curseur, un textField, etc.).**
- **Un traitement d'événement est une méthode qui reçoit un objet Evénement de façon à ce que le programme puisse traiter l'interaction de l'utilisateur.**

Remarque : La gestion des événements graphiques a considérablement évolué entre la version 1.0 et la 1.1. Nous ne présentons ici que la dernière version.

Le modèle d'événements par délégation

- **Java 1.1 implémente un modèle d'événements par délégation.**
- **Les événéments sont envoyés au composant, mais c'est à chaque composant d'enregistrer une routine de traitement d'événement (appelé écouteur) pour recevoir l'événement.**
- **De cette façon, le traitement d'événement peut figurer dans une classe distincte du composant. Le traitement de l'événement est ainsi délégué à une classe séparée.**

Exemple (1)

```
import java.awt.*;
import ButtonHandler;
public class TestButton
  {
      public static void main (String [] args)
      {
            Frame f = new Frame ("Test");
            Button b = new Button ("Press Me");
            b.addActionListener (new ButtonHandler ());
            f.add ("Center", b) ;
            f.pack () :
            f.setVisible (true) ;
         }
   }
```

Exemple (2)

```
import java.awt.event.*;
public class ButtonHandler implements ActionListener
  {
        public void actionPerformed (ActionEvent e)
     {
           System.out.println ("Action occured") ;
       }
  }
```

Lorsqu'on clique sur le bouton, un ActionEvent est envoyé, par l'intermédiaire de la méthode actionPerformed(...), à chaque ActionListener enregistré (ici seulement le ButtonHandler).

Catégories d'événements graphiques (1)

- Plusieurs types d'événements sont définis dans le package `java.awt.event`.
- Pour chaque catégorie d'événements, il existe une interface qui doit être définie par toute classe souhaitant recevoir cette catégorie événements.
  - Cette interface exige aussi qu'une ou plusieurs méthodes soient définies.
  - Ces méthodes sont appelées lorsque des événements particuliers surviennent.

Catégories d'événements graphiques (2)

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

Catégories d'événements graphiques (3)

Adjustment	AdjustmentListener	adjustmentValueChanged (AdjustmentEvent)
Component	ComponentListener	componentMoved (ComponentEvent)componentHiddent (ComponentEvent)componentResize (ComponentEvent)componentShown (ComponentEvent)
Window	WindowListener	windowClosing (WindowEvent) windowOpened (WindowEvent) windowIconified (WindowEvent windowDeiconified (WindowEvent) windowClosed (WindowEvent) windowActivated (WindowEvent) windowDeactivated (WindowEvent)
Container	ContainerListener	componentAdded (ContainerEvent) componentRemoved(ContainerEvent)
Text	TextListener	textValueChanged (TextEvent)

Pour aller plus loin ...

- Une même classe écouteur peut implémenter plusieurs interfaces.
  - Par exemple `MouseMotionListener` (pour les déplacements souris) et `MouseListener` (pour les clics souris) pour un écouteur associé à un `Frame`.
- Chaque composant awt est conçu pour écouter un ou plusieurs types d'événements.
  - Cela se voit grâce à la présence dans la classe de composant d'une méthode nommée `add...Listener()`.
- L'événement peut contenir des paramètres intéressants pour l'application.
  - Exemple : `getX()` et `getY()` sur un `MouseEvent` retournent les coordonnées de la position du pointeur de la souris.

Librairie de composants awt

Button

- C'est un composant d'interface utilisateur de base de type "appuyer pour activer".
- Il peut être construit avec une étiquette de texte précisant son rôle à l'utilisateur.

```
Button b = new Button ("Sample");
add (b);
b.addActionListener (...);
```

- L'interface `ActionListener` doit pouvoir traiter un clic de souris sur le bouton.

Checkbox

- La case à cocher fournit un dispositif d'entrée "actif / inactif" accompagné d'une étiquette de texte.

```
Checkbox one = new Checkbox("One", false);
add(one);
one.addItemListener(...);
```

- La sélection ou la déselection est notifiée à l'implémentation d'une interface `ItemListener`.
  - Utiliser la méthode `getStateChange()` sur un `ItemEvent`. Elle retourne une constante : `ItemEvent.DESELECTED` ou `ItemEvent.SELECTED`.
  - le méthode `getItem()` renvoie la `String` contenant l'étiquette de la case à cocher considérée.

CheckboxGroup

- On peut regrouper des cases à cocher dans un `CheckboxGroup` pour obtenir un comportement de type boutons radio.

```
CheckboxGroup cbg = new CheckboxGroup();
Checkbox one = new Checkbox("One", cbg, false);
...
add(one);
...
```

Choice

➜ **Ce composant fournit une entrée simple de type sélectionner un élément dans cette liste.**

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

Canvas

- Un `Canvas` fournit un espace vide qui peut être utilisé pour dessiner.
  - Sa taille par défaut est zéro par zéro. Ne pas oublier de la modifier avec un `setSize(...)`.
- Un Canvas peut être associé à de nombreux écouteurs : `KeyListener`, `MouseMotionListener`, `MouseListener`.

Label

➜ Un `Label` affiche une seule ligne de texte (étiquette).

```
Label l = new Label("Bonjour !");
add(l);
```

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

TextArea

- La zone de texte est un dispositif d'entrée de texte multi-lignes, multi-colonnes avec barres de défilement. Il peut être ou non éditable.

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

TextField

➜ **Le champ de texte est un dispositif d'entrée de texte sur une seule ligne. Il peut être éditable ou non.**

```
TextField f = new TextField ("Une ligne seulement ...", 30);
add(f);
```

List

➜ **Une liste permet de présenter à l'utilisateur plusieurs options de texte parmi lesquelles il peut sélectionner un ou plusieurs éléments.**

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

nombre d'items visibles

sélections multiples ?

Dialog

- Un `Dialog` ressemble à un `Frame` mais ne sert qu'à afficher des messages devant être lus par l'utilisateur.
  - Il n'a pas de bouton du gestionnaire des fenêtres permettant de le fermer ou de l'iconiser.
  - On y associe habituellement un bouton de validation.
  - Il est réutilisable pour afficher tous les messages au cours de l'exécution d'un programme.
- Un `Dialog` dépend d'un frame (qui est passé comme premier argument au constructeur).
- Un `Dialog` n'est pas visible lors de sa création. Utiliser `setVisible(true);`

FileDialog

➜ C'est un dispositif de sélection de fichier.

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

➜ Un FileDialog ne gère généralement pas d'événements.

ScrollPane

- C'est un conteneur général, ne pouvant pas être utilisé de façon autonome, qui fournit des barres de défilement pour manipuler une zone large.
- Un ScrollPane ne peut contenir qu'un seul composant.

Menu / MenuBar / MenuItem / etc.

- `Menu` : menu déroulant de base, qui peut être ajoutée à une barre de menus (`MenuBar`) ou à un autre menu.
- Des `MenuItem` peuvent être rajoutés dans un menu.
  - En règle générale un `MenuItem` est associé à un `ActionListener`.
- Des éléments de menus à cocher (`CheckBoxMenuItem`) permettent de proposer des sélections "activé / désactivé " dans un menu.
- Des `PopupMenu` sont des menus autonomes pouvant s'afficher instantanément sur un autre composant.
  - Ces menus doivent être ajoutés à un composant parent (par exemple un `Frame`), grâce à la méthode add(...).
  - Pour afficher un `PopupMenu`, utiliser la méthode `show(...)`.

Contrôle des couleurs d'un composant

- Deux méthodes permettent de définir les couleurs d'un composant :

  `setForeground ()`
  `setBackground ()`

- Ces deux méthodes utilisent un argument instance de la classe `java.awt.Color`.
  - La gamme complète de couleurs prédéfinies est listée dans la page de documentation relative à la classe `Color`.
- Il est aussi possible de créer une couleur spécifique (RGB) :

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

Contrôle des polices de caractères

- La police utilisée pour afficher du texte dans un composant peut être définie avec setFont(...) avec comme argument une instance de java.awt.Font. Exemple

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

- Les constantes de style de police sont en réalité des valeurs entières, parmi celles citées ci-après :
  - Font.BOLD
  - Font.ITALIC
  - Font.PLAIN
  - Font.BOLD + Font.ITALIC
- Les tailles en points doivent être définies avec une valeur entière.