Java
les bases du langage

Sylvain LECOMTE
Vincent POIRRIEZ
Didier DONSEZ

Plan

- Introduction
- Les bases du langage Java

Citations

- Ce qui est très intéressant avec JAVA, c'est qu'il a été conçu pour l'Internet.
- Java est parvenu à s'installer avec une facilité tout à fait étonnante.
- Java a été conçu pour offrir une portabilité maximale et pour laisser le moins de choses possibles dépendantes de l'implémentation.

Citations(suite)

- Java:
  - un langage « simple »,
  - orienté objet,
  - réparti,
  - interprété,
  - robuste, sûr, indépendant de l'architecture, portable, efficace, intégrant les processus légers et dynamique...

Citations(suite)

- **« Simple »**: d'une syntaxe familière aux programmeurs C et C++
- **Orienté Objet:** Tout est objet => Il faut concevoir ainsi.
- **Réparti:** Développement d'applications réparties, fourni des classes et paquetages de communication.
- **Interprété**: Le code produit par le compilateur n'est pas du code machine. C'est du bytecode.

Citations (suite)

- **Robuste:** Objectif: logiciel fiable. Moyens: typage fort, pas de pointeurs, gestionnaire de cellules ou ramasse-miettes, vérifications de validité d'accès.
- **Sûr:** Eviter d'exécuter du code dommageable.
- **Indépendant de l'architecture**: Bytecode

Citations(suite)

- **Portable:** Bytecode + compilateur écrit en Java et en C.
- **Efficace:** Pas autant que du code compilé en code machine. Suffisamment pour une large classe d'applications.

Citations(suite)

- **Processus légers**: Augmente l'expressivité d'un langage par la possibilité de plusieurs points de contrôle de la pile de calcul.
- **Dynamique:** Accepte de lier dynamiquement des morceaux de codes à l'exécution.

Citations (fin)

- Avec Java nous gagnons au moins 30 à 40% en temps de développement par rapport à C++.
- Java ne résout pas tous les problèmes.
- ***Java partage ses fonctionnalités avec beaucoup de langages en activité***.

Historique

- 1991: Bill Joy, James Gosling, Arthur Van Hoff propose OAK (électroménager)
- 1995 Présentation de Java en lien avec Internet
- 1996: Animation de pages Web, JDK 1.0, Netscape 2, JVM d 'IBM, Explorer 3;Java Beans
- 1997: Entreprise Java Beans, JDK1.1, JavaCard,JavaOS
- 1998: Java 2 Platform

Documentation et langage

- Distribué gratuitement par Sun, version courante Java 2 Platform alias JDK1.2. Avec des versions pour de nombreux OS.
- La documentation est en ligne à
  - http://java.sun.com
  - http://www.javasoft.com/

Des Environnements de développement

JavaWorkShop	Sun	java.sun.com
JBuilder 2	Inprise	www.inprise.com
Visual Cafe	Symantec	www.symantec.com
Visual Age	IBM	www.ibm.com

Documentation sur la toile

- Faire une recherche avec le mot clef java!!!
- Quelques didacticiels francophones:
  - http://www.infres.enst.fr/~charon/coursJava/index.html
  - http://www.loria.fr/~molli/java/java.article.pdf
  - http://cedric.cnam.fr/~farinone/Java/index.html

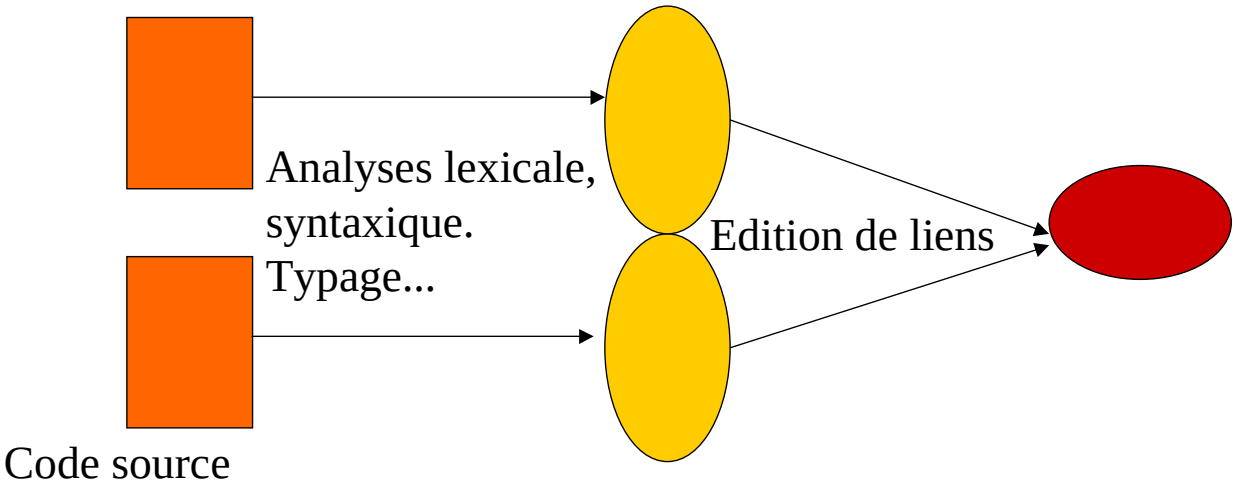
Les bases du langage Java

Rappels sur la compilation

Schéma général de production de code

- La compilation est effectuée en plusieurs étapes. Elle a plusieurs objectifs:
  - Vérifier la correction lexicale, syntaxique,
  - Vérifier « le plus possible » de sémantique: typage
  - Produire du code dans un langage plus simple
  - Produire « des briques » de code assemblables

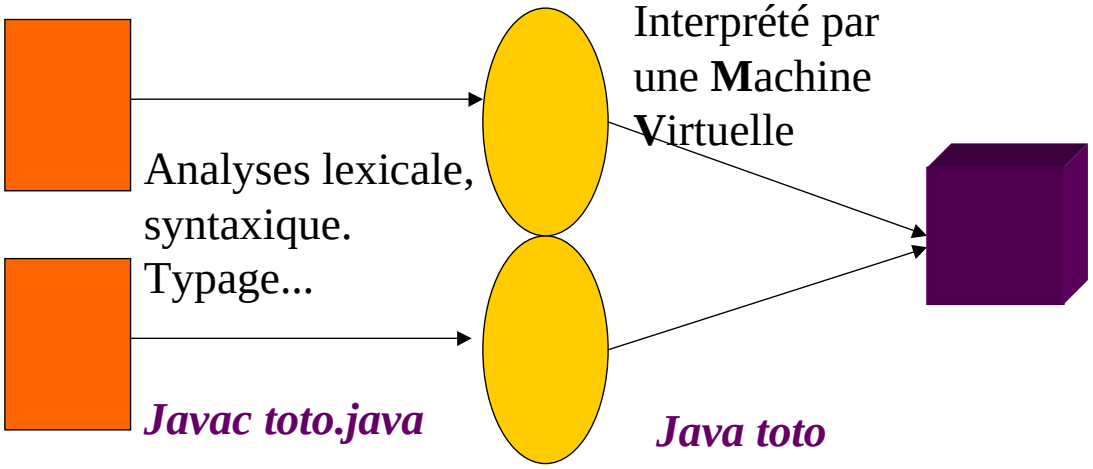
Génération de Code
a) Produire du code machine *natif*



Avantages/Inconvénients du code natif

- Rapidité d 'exécution
- *Obligation de recompiler si l 'on change d 'environnement (Architecture, OS)*
- *Choix entre fournir le source ou le binaire pour la machine du client.*

Génération de Code
b) Produire du code intermédiaire *bytecode*



Code source

***.java** /*.ml/...

Code objet

***.class**/*.cmo/...

Machine Virtuelle

JVM/

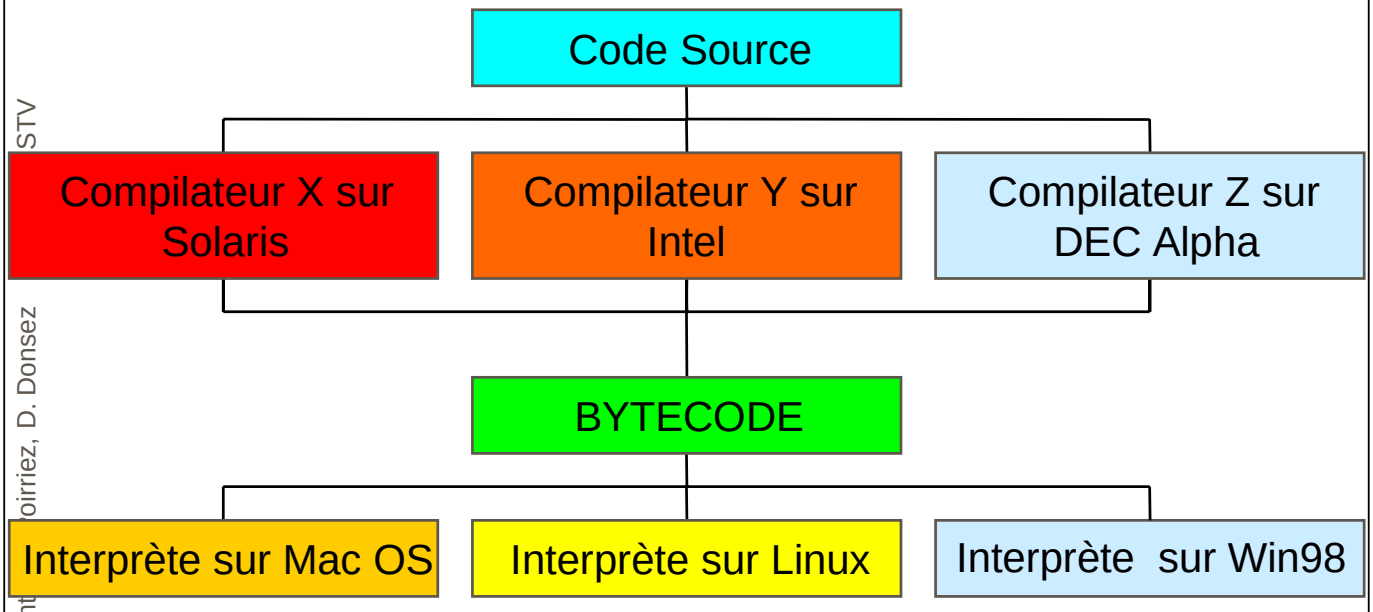
ocamlrun/...

Avantages/Inconvénients du Bytecode

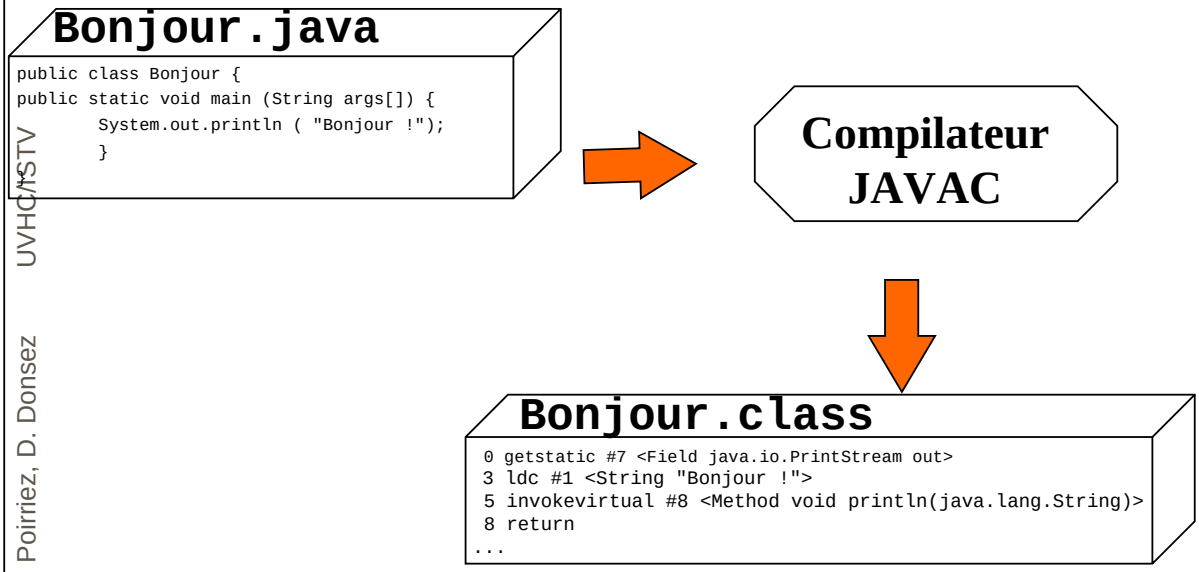
- Code portable
- Code Mobile

- Moins efficace que du code natif

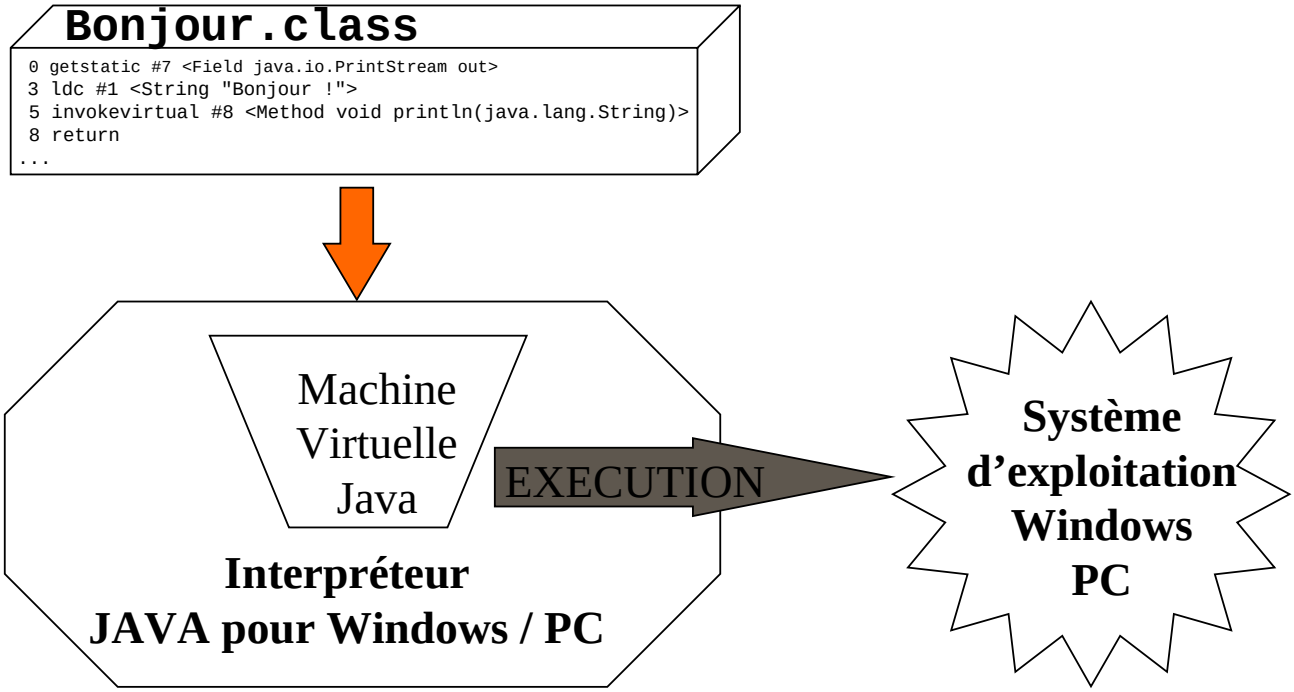
Principe d ’une machine virtuelle:



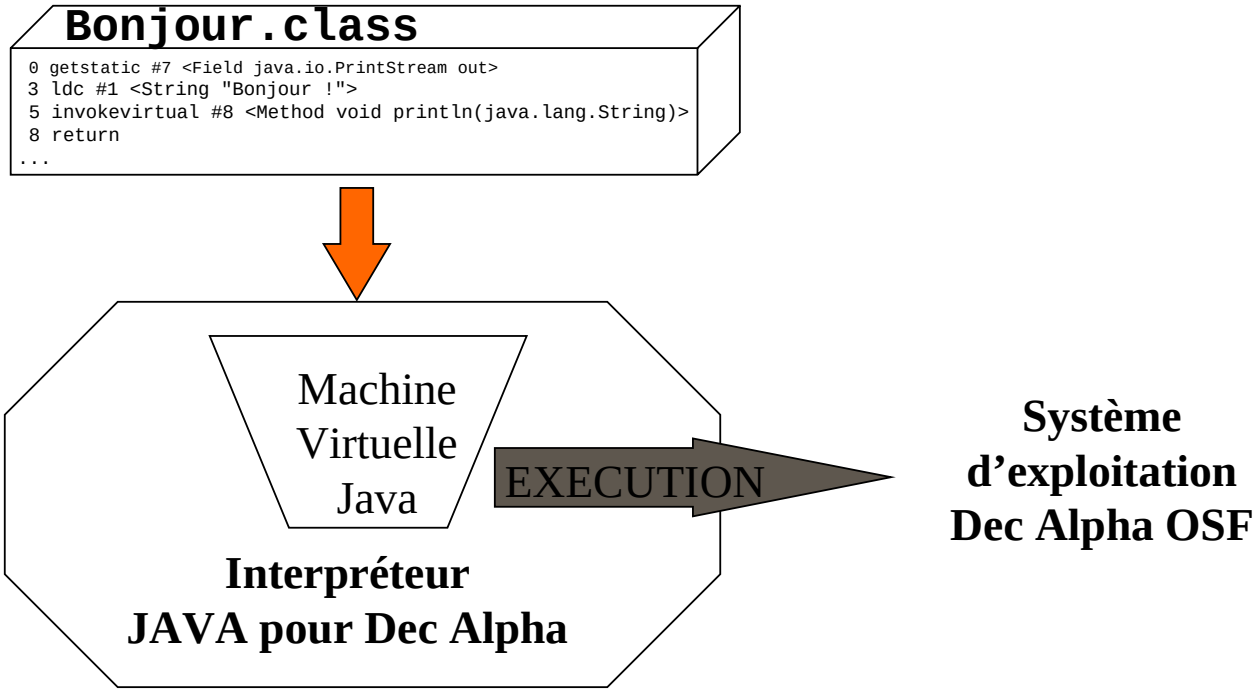
Compilation de Bonjour.java



Interprétation sur PC Dos



Interprétation sur Dec Alpha OSF



Données et bytecode

- Portable => format de donnée complètement spécifié, indépendant de l 'architecture.
- Les types primitifs ne sont pas directement les types machines. Ainsi, les entiers ont une représentation unique, indépendante du choix du constructeur.
- Travail de l'interprète: codage, décodage

Important :

- Bien positionner les variables d 'état :
  - CLASSPATH : indique le chemin où se trouve les classes
  - JAVA_HOME : répertoire de base du JDK
  - PATH : doit contenir le répertoire du compilateur et de la machine virtuelle
  - Être TRES ordonné !!!
- Sinon…
  - Erreur à la compilation (no class def found…)

Les bases du langage Java

Rappels sur la POO/
Vision java de la POO

Vocabulaire de la POO

- Classe (ou type d'objets)
  - décrit des "choses" ayant les mêmes propriétés.
- Objet
  - une instance de classe

Une classe est une représentation abstraite d'un objet.

Pour java, tout est décrit en terme de classe et d'objet. Pas d'autre notion.

Tout est objet *sauf:*

- Les types primitifs:
  - int, boolean, double …
- Attention, les chaînes sont des objets (classe String)
- Il existe pour chaque type primitif une vision en tant qu'objet.(classe Integer,…)
  - Intérêt: utilisable partout où peut être un objet.
  - Inconvénient: surcoût d'encodage décodage.

Vocabulaire de la POO

- Membre (d'une classe): Champs ; Méthode; *Classe interne*.
  - Champs (ou attribut)
    - l'ensemble des membres définissant l'état d'un objet
  - Méthode
    - une action (fonction, procédure) qui manipule l'état d'un objet

Principe à respecter: l'état d'un objet n'est modifié que par l'invocation de méthodes.

Membres de classe static

- Si un membre est déclaré static alors il est commun à toutes les instances de la classe.
- Une classe qui ne comporte que des membres static ne sert pas à créer des objets!!!
  - Si tous les attributs et méthodes sont de classe, il est abusif de dire que le programme est pensé en POO.

Membres de classe

- Invocation de l'extérieur d'un membre de classe:

nom_classe.nom_attribut_ou_méthode

- Un membre qui n'est pas *de classe* est dit *d'instance* (c'est le défaut)

Attribut 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.

Méthode static

- Ne peut utiliser directement aucun attribut ni aucune méthode non statiques de sa classe ; une erreur serait détectée à la compilation.
- Ne peut pas être redéfinie dans les classes dérivées.
- Optimisation possible.

Attribut d ’instance

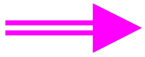
- Un attribut d'instance n'existe par rapport à une allocation mémoire que dans une instance de cette classe.
- Avant qu'une instanciation de la classe (new) ne soit effectuée, l'attribut n'a aucune existence physique.
- Un exemplaire d'un certain attribut d'instance de la classe par instance créée.

Méthode d 'instance

- Une méthode qui manipule, en lecture ou en écriture, des attributs d'instance de sa classe doit être d'instance .
- À l'envers, il est plus correct de déclarer en static une méthode n'utilisant aucun attribut ou méthode d 'instance (non static) de sa classe.
- Lorsqu'une méthode d'instance est invoquée, la référence (this) de l'instance traitée lui est passée implicitement.

Membres d 'instance

- Non static,
- Invoqués de l'extérieur comme membre d 'une instance et non pas comme membre d 'une classe.



nom_instance.nom_attribut_ou_méthode

Gestion de la mémoire

MaClasseBouton a, b;

a et b ne sont pas des objets mais des références sur de futurs objets.

a = new MaClasseBouton("A");

a réfère un bouton appelé A

b = new MaClasseBouton("B");

b réfère un bouton appelé B

a = b; // a et b référent le même bouton B.

Le bouton de nom "A" n'est plus référencé

Gestionnaire de cellule(gc)

Ou garbage collector ou encore éboueur ou bien même ramasse-miettes

- Chargé de détecter les objets devenus inaccessibles.
- Gain de fiabilité (pas de désallocation erronée)
- A un coût (programme concurrent)
- N'est pas une exclusivité de Java: Smalltalk,..., Lisp, Ocaml, ...

Vocabulaire de la POO: Encapsulation

- Deux sens
  - Regrouper des caractéristiques au sein d 'une classe. Principe de POO.
  - Cacher certains membres d 'une classe à certaines autres classes, avec un choix éventuel dans les niveaux de confidentialité. En java, correspond aux modifieurs de visibilité: protected, private, public

Ensembles de classes(en java)

- Il est possible de ranger les classes selon des ensembles appelés **paquetages (package)**.
- Les règles de visibilité entre les classes et entre les attributs et méthodes qu'elles contiennent dépendent de l'appartenance ou non à un même paquetage.

Vocabulaire de la POO
Héritage

- permet de décrire des classes de plus en plus précises, héritant des caractéristiques de classes plus générales.
- Classe dérivée ou sous-classe ou classe fille
- Surclasse ou classe mère

Les membres hérités ne peuvent pas avoir une visibilité plus faible dans les classes filles que dans la classe mère.

Un premier programme

```
/*Ce fichier s'appelle Premier.java
compiler avec javac Premier.java
exécuter avec java Premier
*/
class Premier{
  public static void main(String[] arg)
  {System.out.println("Bonjour");}
}
/*A l'exécution, on obtient : Bonjour*/
```

Une première « vraie » classe

```
class Ecriture {
 /*chaîne est un attribut de la classe Ecriture,
    qui doit contenir une référence vers une instance de la
   classe String*/
       String chaîne = "";
 //méthode de la classe Ecriture
       void écrire(String autreChaîne){
         System.out.print(chaîne);
         System.out.println(autreChaîne);}
     }
```


Un « vrai » programme

```
class PourSaluer {
       public static void main(String[] arg){
        Ecriture écrivain;
        écrivain = new Ecriture();
        écrivain.chaîne = " Bonjour ";
        écrivain.écrire("à tous");
        écrivain.chaîne = "et bon courage ";
        écrivain.écrire("pour java");}
     }
```

Remarques

- Un nom de classe commence par une majuscule.
- Les deux classes peuvent être dans un même fichier.
- Le fichier doit avoir le même nom que la classe qui contient une méthode public main (ici PourSaluer.java)
- Il peut y avoir au plus une classe avec une méthode main dans un fichier
- Il y aura autant de fichier .class produit par le compilateur qu'il y a de classe dans le fichier .java
- Eviter les accents dans les noms des classes => pb avec les systèmes de fichiers

Constructeur

- Toute classe comporte au moins un constructeur.
- Un constructeur est une méthode qui a le même nom que la classe.
- Si le programmeur n 'en spécifie pas, le compilateur en rajoute un qui ne comporte qu'une seule instruction: super() c'est à dire appel au constructeur de la classe mère.

Constructeur

- Initialiser les attributs de l 'instance créée.
- Il peut y avoir plusieurs constructeurs car plusieurs manière d 'initialiser.
- Tout constructeur d'une classe fille commence par faire appel à un constructeur de la mère
  - implicitement: super()
  - explicitement: super(args)

Exemples de Constructeurs

```
class Segment{
  int gauche;
  int largeur;
  int droite;
  Segment(int g, int d )
    {gauche = g; droite = d; largeur = d-g;}
  Segment(Segment orig, int dep )
    {this(orig.gauche+dep,orig.droite+dep);}
}
```

Exemple incorrect

```
class Segment{
  int gauche;
  int largeur;
  int droite;
  Segment(int g, int d )
    {gauche = g; droite = d; largeur = d-g;}
  Segment(int  larg, int g )
    {this(g,g+larg);}
}
```

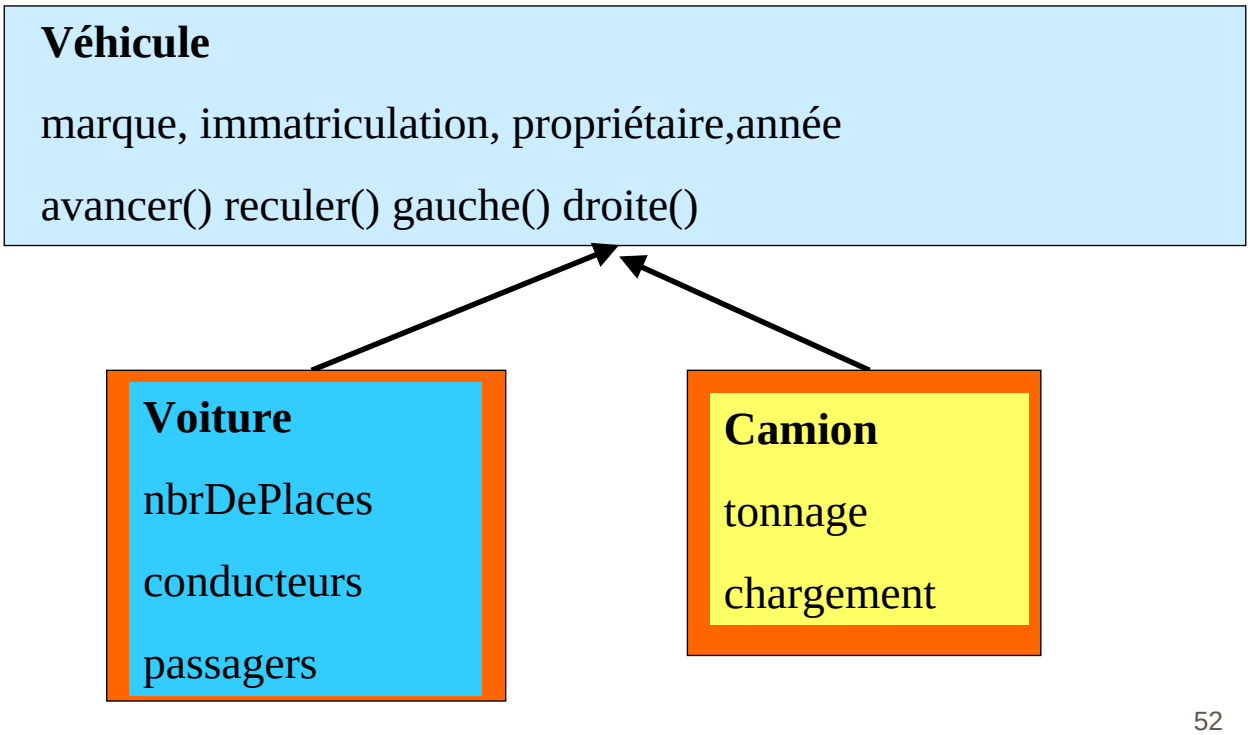

Vocabulaire de programmation: Surcharge

- Ecrire plusieurs méthodes de même nom dans une classe.
- Les méthodes doivent différer par le nombre des arguments, ou par le type des arguments.
- Avoir plusieurs constructeurs est un cas particulier de surcharge.

Surcharge des opérateurs

- Est interdite en Java
- Exception: l 'opérateur + est surchargé, c 'est l 'addition et la concaténation de chaînes de caractères

Exemple d 'héritage



Squelette de véhicule

```
class Vehicule {
        String marque;
        String immatriculation;
        String propriétaire;
        int année;
        int posx;
        int posy;

        Vehicule()
        Vehicule(int x, int y,String m, String i String
   p, int a){// initialisation par les arguments}
        ...}
```


Squelette de Voiture

```
class Voiture extends Vehicule {
 private int nbrDePlace;
 private String passagers[];
 protected LinkedList conducteurs;
 Voiture(int places, String c1, int x, int y, String m,
   String i, String p, int a){
   super(x,  y, m,  i, p, a);
   nbrDePlace=places;
   passagers = new String[places];
   conducteurs = new LinkedList();
   conducteurs.add(c1);}...
```

Redéfinition

- On peut redéfinir une méthode définie dans une classe à l'intérieur d'une classe dérivée.
- Il faut que la méthode redéfinie ait
  - même nom,
  - mêmes types et nombre d'arguments
  - même type de retour que la méthode d'origine.
- L'interprète cherche la définition d'une méthode invoquée à partir de la classe de l'instance concernée.
- La première implantation rencontrée en remontant la hiérarchie de classes est celle choisie.

Redéfinition et super

- Le mot réservé super permet de faire appel à la méthode définie dans la classe mère (super.p est un appel à la méthode p de la classe mère)

```
class That {
   protected String nm(){return "That";}}
class More extends That {
   protected String nm(){return "More";}
   protected void printNM(){
        That sref =  new More();
        System.out.println(this.nm());
        System.out.println(super.nm());} }
class TestSuper {
    public static void main(String args[]){
         More p = new More();
          p.printNM();}}
```

Le modifieur final

- Une méthode finale ne peut pas être redéfinie dans une classe dérivée.
- Une classe finale ne peut pas être dérivée
- Un attribut final ne peut pas être modifié

Méthode finale

- Déclarée avec final
- Le contraire d'une méthode finale est une méthode virtuelle ou différée. Par défaut, toute méthode est donc virtuelle.
- Intérêts:
  - l 'appel à une méthode déclarée finale peut être optimisé par le compilateur (« inlining »)
  - Sécurisation accrue.

Classe abstraite

- Méthode abstraite:
  - méthode déclarée et non définie, mot clef: abstract
- Classe abstraite:
  - classe qui contient au moins une méthode abstraite
  - Pas d 'instance d 'une classe abstraite, ce ne serait pas concret. Seule possibilité: dériver jusqu'à une non abstraite.

Figures

Liste de figures comportant des cercles et des rectangles, possibilité d'obtenir une liste des périmètres.

```
Import java.util.*
abstract class Figure {
        Point orig;
        Figure(Point o){orig=new Point(o);}
        abstract double périmètre();
       }
```

Cercles et rectangles

```
class Cercle extends Figure{
  private static final double pi = 3.141592;
  double rayon;
  Cercle(Point centre,double r){super(centre);rayon=r;}
  double périmètre(){return 2*pi*rayon;}}
class Rectangle extends Figure{
  double LX;double LY;
  Rectangle(Point ig,double lx, double ly){super(ig);LX=lx;LY=ly;}
   double périmètre(){return 2*(LX+LY);}}
```


Liste de figures et périmètres

```
...
lf = new LinkedList();
lp = new LinkedList();
for (i =0;i<7;i++) lf.add(new Cercle(//));
for (i =0;i<4;i++) lf.add(new Rectangle(//));
ListIterator e =lf.listIterator(0);
while(e.hasNext()){lp.add(e.next().périmètre());}
...
```

Généricité

- Une structure est générique si elle peut contenir n 'importe quel type de donnée
- Une méthode est générique si elle s 'applique à n 'importe quelle instance d 'une structure générique.
- Exemple: le tri, l 'itération, …
- Implanté en java par une racine unique: Object

Héritage multiple

- Java interdit qu'une classe hérite de plus d'une classe.
- Toute classe hérite d'une classe et d'une seule (exception: la classe Object= racine de la hiérarchie, n'a pas de mère).

Interfaces

- le point limite d'une classe abstraite:
  - Pas de champs,sauf des constantes
  - Toutes les méthodes sont abstraites
- syntaxe:interface xxx {...}

exemple:

```
import java.util.*;
interface Instrument {
 int diapason = 140; // static & final: Compile-time constant
 void play(); // Automatiquement public
 String what(); // Pas de corps de méthode
 void adjust();   }
```


Instruments suite

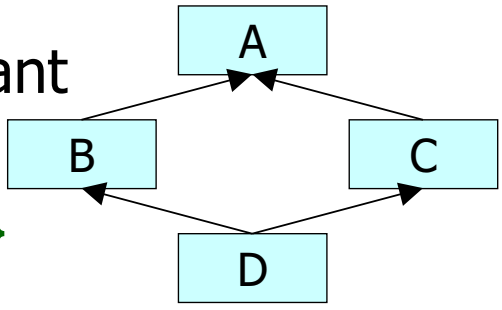
```
class Vents implements Instrument {
  public void play(){System.out.print("Vents.play()");}
  public String what() { return "Vents"; }
  public void adjust() {}
 }
class Percussion implements Instrument {
  public void play(){System.out.print("Percussion.play()");}
  public String what() {return "Percussion"; }
  public void adjust() {}
 }
class Cuivres extends Vents {
  public void play(){System.out.print("Cuivres.play()");}
  public void adjust(){System.out.print("Cuivres.adjust()"); }
 }
```

Héritage multiple et interface

- Hériter d 'une seule classe
  - étendre une seule classe
- Implanter une ou plusieurs interfaces

Héritage multiple et interface

- Cas de l 'héritage en Diamant




```
class A {int a;void ma(){};}
class B extends A {int b;void mb(){ };}
class C extends A {int c;void mc(){ };}
interface C_intf extends C { }
class D extends B implements C_intf {
    int d; C oc;
    D(){super(); oc=new C();}
    void mc() { oc.mc(); }
}
```

Les bases du langage Java

Instructions de contrôle

Les commentaires

- Sur une seule ligne :

```
// ceci est un commentaire
```

- Sur plusieurs lignes :

```
/* ceci est un commentaire plus
   long ... */
```

- pour JavaDoc (documentation automatique) :

```
/**  commentaire pour javaDoc
* blah blah...
*/
```

Les types élémentaires

Nom	Taille	Exemple
byte	8	1
short	16	2
int	32	-12
long	64	2L
float	32	3.14
double	64	0.5d
boolean	1	true ou false
char	16 (UNICODE)	' a ', ' \u0000 '

Variables...

- Déclarations : <type> <ident.> [= exp];
  - int x = 10;
  - float f;
- Le compilateur contrôle l'initialisation
  - => pas d 'initialisation automatique
  - ex : variable i may not havce been initialized
- Une variable peut être déclarée finale :
  - ex : public meth (final int x) {
    int y = x+2; **OK**
    x = x+3; }} **ERREUR**

Tableaux

- Tableaux = objets référencés
  - Déclaration :
    - int tableau1[];
    - int[] tableau2;
    - int matrive[][];
    - int[] x,y[]; équivalent à int x[], y[][];
  - utilisation :
    - exemple de méthode dispo: length
    - Allocation et initialisation :

```
int tab1[] = new int[10];
int tab2[] = {1, 2, 3, 4};
```


Tableaux (2)

- Utilisation correcte :

```
int tab1[] = new int[10];
tab1[0] =100; // OK
tab1[10] = 100 ; // ERREUR
```

- Les chaînes de caractères ne peuvent pas être manipulées comme un tableau de caractères.
- Le tableau n 'est pas un type élémentaire => une variable de type tableau est donc une référence sur un tableau (utilisation de new pour créer le tableau)

If...else

- if(expression booléenne) {
  instructions
  } else {
  instructions
  } // Le bloc else est optionnel
- Exemple :

```
if(i < 0) i=0;
if(error) {
        fin = true;
        i = 0;
} else i++;
if(i) i=0;  // NON, car i n 'est pas booléen
```


switch

- switch (expression) {
  case expr1:
  intructions;
  [break;]
  ...
  default:
  instructions;
  [break;]

Boucle for

- for(init; test; incrément) {
  instructions;
  }
- Exemple :

```
int i;
for(i=0; i < 10; i++) {
        System.out.println(i);
        }
```

- Il est possible de déclarer la variable dans la boucle for
  - for(int i = 0...

While et do...while

- while(expression_booléenne) {
  instructions;
  }
- do {
  instructions;
  } while (expression_booléenne);

Et maintenant...

Il faut beaucoup pratiquer...

Les bases du langage Java

Un petit programme pour commencer...

Le Jeu FizzBuzz…

- Code 1 : TestFizzBuzz.java

```
class FizzBuzz{
    int tour;
    void joue (){
        if ((tour%5)==0){
            if ((tour%7)==0) System.out.print(« Fizzbuzz »);
            else System.out.print(« Fizz »);}
    else if ((tour%7)==0) System.out.print(«buzz »);
        else System.out.print(tour);
    System.out.print«  »);
    if (tour%10==0) System.out.println();
};
}
```


Le point d ’entrée…

```
class TestFizzBuzz{
    static int Bornesup=50;
    static FizzBuzz fizzbuzz = new FizzBuzz();
    public static void main(String argv[]){
        Compteur cpt = new Compteur();
    while (! Cpt.equals(Bornsup)){
        cpt.incr();
        fizzbuzz.tour=cpt.valCompteur();
        fizzbuzz.joue();
}
};
}
```

Et le compteur ?

```
class Compteur{
   int contents;
   Compteur() {contents=0;}
   void incr(){contents++;}
   boolean equals(int j) {return(contents==j);}
int valCompteur(){return(contents);};
}
```


Les bases du langage

Un peu plus complexe…

La Classe Robot

- Un robot est un objet qui :
  - dispose de 4 attributs :
    - Son nom, sa position X et Y et son orientation
  - de 2 méthodes :
    - Se déplacer, et tourner à droite
- Tout les Robot ont en communs la représentation de l 'orientation :
  - Nord = 1, Est = 2, Sud = 3 et Ouest = 4
- Ecrire la classe Robot
- Ecrire une classe qui test Robot

La Classe Robot (suite)

- Définir 3 constructeurs différents pour instancier la classe Robot :
  - Un constructeur par défaut, qui place le Robot en (0,0) et orienté au Nord
  - Un constructeur qui permet de fournir coordonnées, nom et orientation en paramètre
  - un constructeur qui place le robot en fonction d 'un robot déjà existant (on fournira la référence du premier robot et le nom que l 'on veut donner au nouveau)

Les bases du langage Java

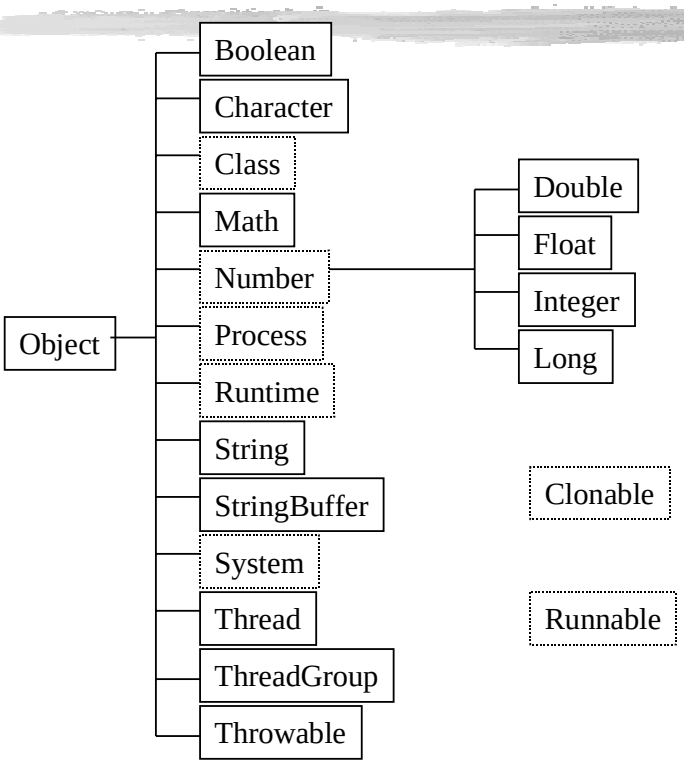
Notions de l ’API Java

L’aide ou documentation en ligne

- Toutes les API du JDK sont documentées. Au format html.

  jdk1.2.1/docs/api/index.html
- Ces API sont très puissantes, allons voir à quoi correspond une collection.
- Puis nous parcourrons rapidement les API.

Java.lang



java.lang.String

- La classe `String` représente une chaîne de `char` (encodage UNICODE sur 16 bits)
  - La notation litéral
    - String str = "abc";
  - est transformée par le compilateur en
    - char data[] = {'a', 'b', 'c'};
    - String str = new String(data);
- Exemples
    - System.out.println("abc");
    - String cde = "cde";
    - System.out.println("abc" + cde);
    - String c = "abc".substring(2,3);
    - String d = cde.substring(1, 2);

java.lang.StringBuffer

- La classe `StringBuffer` représente aussi une chaîne de `char` mais elle est implantée de façon à rendre performantes les opérations de chaînes (concaténation, insertion, …).
- Le compilateur l 'utilise pour transformer :
  - Float cote; String rep; ...
  - rep = "cote vaut " + cote + " $";
- en :
  - rep = new StringBuffer().append("cote vaut ")\
  - .append(cote).append(" $").toString()

java.lang.Object

- La classe `Object` est la classe racine de la hiérarchie des classes Java
- Les méthodes suivantes sont implantées par toute classe y compris les tableaux []
  - clone()
  - equals(Object)
  - finalize()
  - getClass()
  - hashCode()
  - toString()
  - wait(..), notify(..), notifyAll(..)

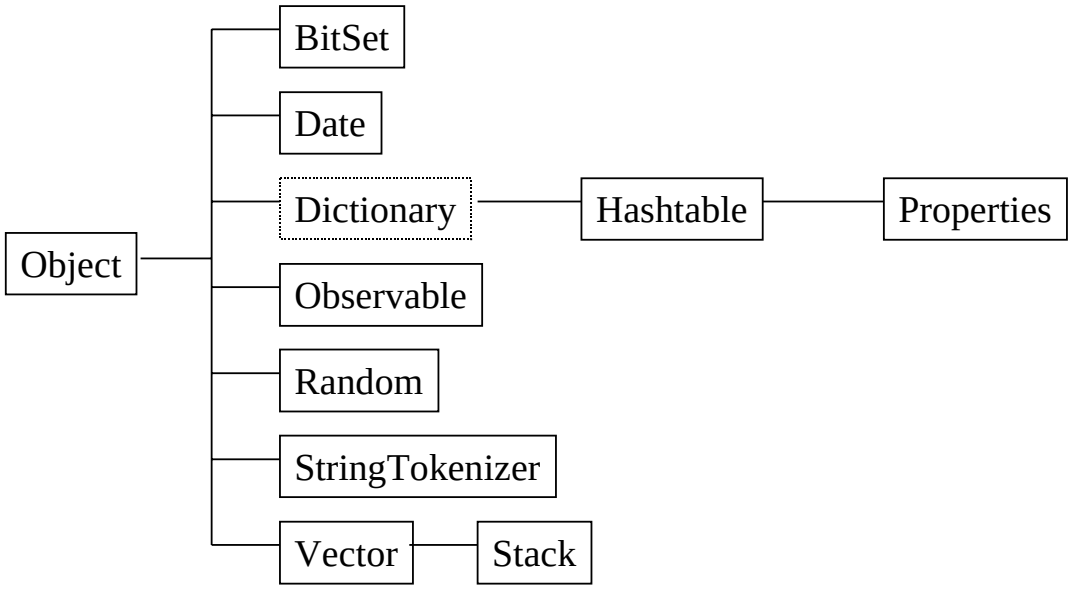
java.lang.Class

- La classe `Class` représente une classe java.
- Elle n'est pas instanciable
- Elle permet de créer dynamiquement des nouvelles instances (mais seul le constructeur par défaut est appelé)

```
Class classname = Class.forName("java.util.date");
Date d = (Date)classname.newInstance();

System.out.println("Date : " + d);
   // Sat Jan 18 12:10:36 GMT+0100 1997
Integer i = classname.getMethod("getMinutes", null).invoke(d,
   null);
```


Java.util



La classe Java.util

- Très souvent utilisée :
  - Les classes Vector et Stack pour stocker des objets dans des zones de taille variable
  - La classe StringTokenizer pour découper une chaîne de caractères en mots
  - Le mécanisme observer/Observable pour prévenir un objet de la modification d 'un autre objet
  - … consultez le Tutorial Java 1.2

La classe Vector : un tableau d 'objets

- Un Vector se comporte comme un tableau, mais
  - il peut grossir automatiquement
    - A la création, on peut indiquer une taille initiale et un incrément en cas de remplissage (sans quoi, la taille double à chaque fois)
  - il propose différentes méthodes pour travailler :
    - public final synchronized void addElement(Object newElement)
    - public final synchronized void InsertElement(Object newElement, int index) throws ArrayIndexoutofBoundsException
    - public final synchronized void SetElementAt(Object newElement, int index) throws ArrayIndexoutofBoundsException

Accès aux éléments

- On ne peut pas utiliser d 'indexation
- Méthodes spécialisées :
  - public final synchronized Object ElementAt(int index) throws ArrayIndexoutofBoundsException
  - public final synchronized Object FirstElement() throws NoSuchElementException
  - public final synchronized Object LastElement() throws NoSuchElementException
- Autres méthodes :
  - IsEmpty() : teste si le vecteur est vide
  - size() : donne la taille du vecteur

Recopie d 'un Vecteur

- Utilisation de CopyInto qui permet de dupliquer un objet :
  - Exemple
    Object tab = new object(myVector.size());
    myVector.copyInto(tab);

A consulter également dans java.util

- L 'interface Enumeration
  - permet d 'énumérer une liste objets qui peuvent être hétérogènes
  - Ex : la méthode elements() de la classe Vector retourne une énumération
  - 2 méthodes :
    - nextElement() : retourne l 'obj suivant dans la liste
    - hasMoreElements : retourne True si il reste des éléments
  - Ces méthodes retourent des « Object » il faut donc les caster...

Les bases du langage Java

Les exceptions

Les exceptions

- Une exception correspond à un événement anormal ou inattendu.
  - On ne peut éviter toute erreur (plus de mémoire, division par 0, ...)
  - Mais certaines sont prévisibles :
    - fichier inexistant
    - mauvaise saisie
- Les exceptions permettent de traiter ces erreurs

Les exceptions (2)

- 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)
  - java.lang.Exception (pour des événements inattendus, qui seront souvent traités de sorte qu'elle ne provoque pas l'arrêt du programme).

Capture d'une exception

```
try{
   // bloc try : les instructions ou les appels
   // de méthodes peuvent lever des exceptions }
catch(HorsBorne_X ex){
   //section à effectuer si une exception HorsBorne_X
   // est levée dans le bloc try
   }
finally{
  //section effectuée quelque soit la façon dont on
  //sort du bloc try effectuée le cas échéant après
  //le bloc
  // catch
}
```


exemple

- try{
  k = Divise(i, 0);
  }
  catch(exception e) {
  // traitement de l 'exception
  }
- Il peut y avoir plusieurs catch, mais on ne peut passer que dans un seul à la fois

Propagation d'une exception

- Lancée par une instruction I de uneMéthode.
- I se trouve dans un bloc de uneMéthode :
  - précédé de try, et suivi d'un bloc précédé du mot réservé catch(*arg*), *arg* est de la classe ou d'une super-classe de l'exception lancée.
  - Alors :
    - les instructions qui suivent I et précèdent catch sont ignorées
    - les instructions du bloc catch sont effectuées
    - le programme reprend normalement avec l'instruction qui suit le bloc catch.

Propagation d'une exception

- I n'est pas située comme indiqué ci-dessus.
- Alors,
  - uneMéthode se termine .
    - Si uneMéthode est la méthode main, le programme se termine et l'exception n'a pas été attrapée.
    - Sinon, on se retrouve dans la méthode qui a appelée uneMéthode, au niveau de l'instruction I' qui a fait appel à uneMéthode. L'instruction I' lance à son tour l'exception.

Propagation d'une exception

- Si une exception est lancée et pas attrapée, et donc qu'elle provoque la terminaison du programme, la pile des méthodes traversées par l'exception est indiquée à l'utilisateur.

Exception exemple

```
class ExceptionCatch{
 static int moyenne(String[] liste) {
   int somme = 0, entier, nbNotes = 0;
   for (int i = 0; i < liste.length;i++) {
     try {
       entier = Integer.parseInt(liste[i]);
       somme += entier;
       nbNotes++;}
       catch (NumberFormatException e){
         System.out.println("La "+(i+1)+" eme note n'est pas entière");}}
   return somme/nbNotes;
}
public static void main(String[] argv){
   System.out. println("La moyenne est " + moyenne(argv));}}
```

Trace de l'exemple

- Pour : java ExceptionCatch ha 14 12

  on obtient en sortie

  La 1 eme note n'est pas entière

  La moyenne est 13

- et pour : java ExceptionCatch ha 15.5

  La 1 eme note n'est pas entière

  La 2 eme note n'est pas entière

  java.lang.ArithmeticException: / by zero

  at ExceptionCatch.moyenne(ExceptionCatch.java:22)

  at ExceptionCatch.main(ExceptionCatch.java:27)

Définir son exception

```
class ExceptionRien extends Exception
{
  public String toString()
  {return "Aucune note n'est valide";}
}
```

Déclencher son exception

```
class ExceptionThrow{
  static int moyenne(String[] liste) throws ExceptionRien{
   //idem précédent
    if (nbNotes==0) throw new ExceptionRien();
    return somme/nbNotes;}
  public static void main(String[] argv){
    try {
       System.out.println("La moyenne est " +moyenne(argv));}
    catch (ExceptionRien e)
      { System.out.println(e);}  }
}
```


Les bases du langage Java

Les Entrées/Sorties

La classe Java.lang.System

- Interface avec l 'OS
- 3 fichiers standards :
  - System.in : un java.io.bufferedInputStream
    - public int read() : attend une entrée et retourne son code
    - public long skip(long) : se déplace d 'un nb d 'octets dans le flot
  - System.out et system.err : des java.io.PrintStream
    - public void print(tout type java)
    - public void println(tout type java)

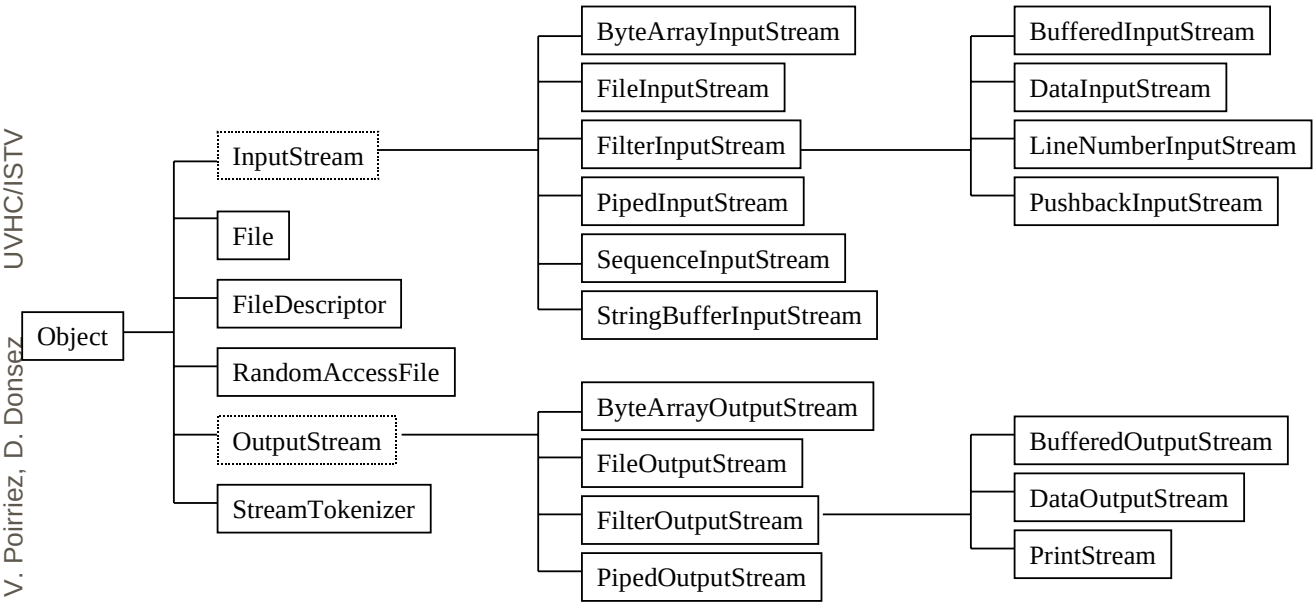
Exemple

```
import java.io.*;
class MonAppli {
   public static void main(String[] argv){
        int b, nb = 0;
        try {
                while((b = System.in.read()) != -1){
        nb++;
        system.out.println(c);
   }
} catch(IOException e ...
```

Le package Java.io

- Fournit des classes pour manipuler différentes ressources
  - Fichiers (classe Java.io.File) :
    - FileInputStream, FileOutputStream
  - Mémoire
    - BufferedInputStream, BufferedoutputStream
    - DataInputStream, DataOutputStream (lectures typées)
  - pipe :
    - PipedInputStream, PipedOutputStream (échange Threads)
  - Lecture filtrée

Un peu plus détaillé...



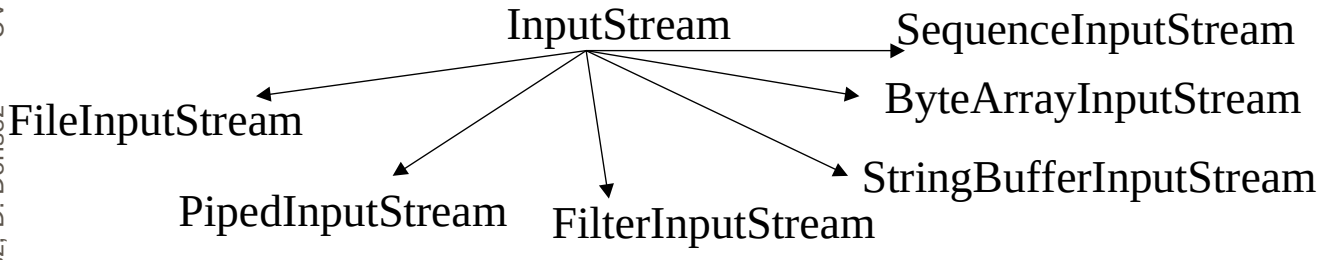
La classe Java.io.File

- Gestion de fichier :
  - Nommage, Droits d 'accès, informations diverses...
- Exemple :

```
File f = new File(« toto.txt »);
System.out.println(« toto.txt »+f.getPath());
if(f.exists()) {
        System.out.println(« droits » + f.canRead()
+ f;canWrite());
```


Java.io.InputStream

- Classe abstraite qui fournit un constructeur + un ensemble de méthode (dont read)



Java.io.FileInputStream

- Exemple :

```
FileinputStream fis;
byte[] b = new byte[1024];
try {
      fis = new FileInputStream(« toto.txt »);
} catch (FileNotFoundException e) {…}
try {
      int i =fis.read(b);
} catch (IOException e) {…}
String s = new String (b,0);
```

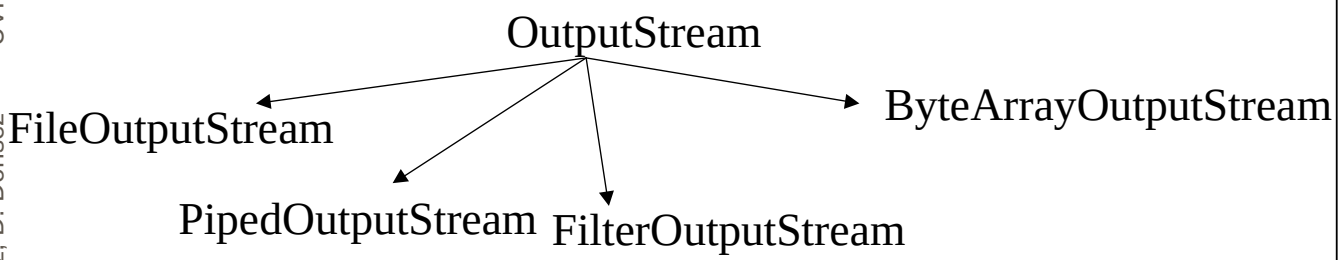

Java.io.DataInputStream

- Lecture de données typées
  - Un entier a la même taille partout…
- Méthode de lecture :
  - byte readByte(), short readShort(), …

```
FileinputStream fis;
fis = new FileInputStream(« toto.txt »);
DataInputStream dis =new DataInputStream(fis);
String ligne = new String();
while((ligne = dis;readLine()) != null) {
  System.out.println(ligne);
}
dis.close;
```

Java.io.OutputStream

- Classe abstraite de base de tout les flots de sortie



Java.io.FileOutputStream

- Exemple d ’écriture dans un fichier :

```
FileOutputStream fos = new FileOutputStream(« toto.txt »);
String s = new String (« Coucou »);
int longueur = s.length();
byte[] buffer = new byte[longueur];
s.getBytes(0, longueur-1, bufffer, 0);
fon(int i = 0; i < longueur; i++)
      fos.write(buffer[i]);
```

Java.io.BufferedOutputStream

- Ecriture sur un flot de sortie bufférisé
  - Spécialisation à partir de n 'importe quel flot de sortie
  - Un entier a la même taille partout…
- Méthodes
  - write(int), write(byte[], int offset, int length), flush(), close() …

```
FileOutputStream fos= new FileOutputStream(« toto.txt »);
BufferedOutputStream bos =new BufferedOutputStream (fos);
String s = new String (« Coucou »);
int longueur = s.length();
byte[] buffer = new byte[longueur];
s.getBytes(0, longueur-1, bufffer, 0);
fon(int i = 0; i < longueur; i++)
bos.write(buffer[i]); bos.write(« \n »);
bos.close;
```


Java.io.DataOutputStream

- Ecriture de données typées
  - Un entier a la même taille partout…
  - Conseil : spécialisation à partir d 'un BufferedOutputStream
- Méthode de lecture :
  - byte writeByte(), short writeShort(), …

```
FileinputStream fos = new FileInputStream(« toto.txt »);
BufferedOutputStream bos =new BufferedOutputStream (fos);
DataOutputStream dos =new DataOutputStream(bos);
dos.writeChars(« position »);
dos.writeDouble(10.0);
dos.writeDouble(12.0);
dos.writeDouble(7.0);
…
```

Nouveautés du JDK 1.1

- Notion de flot de caractères (characters Streams) :
  - Avant on travaillait que sur des bytes
  - Maintenant : caractères (16b)
  - 2 classes : Writer et Reader
    - FileReader <-> FileInputStream
    - FileWriter <-> FileOutputStream
  - Echange facile de données avec accents (unicode)

Récapitulation

Flot de caractères	Description
Reader	Classe Abstraite pour flot d 'entrée (InputStream)
BufferedReader	Bufférise les entrées ligne par ligne
LineNumberReader	garde trace du nb de lignes lues
CharArrayReader	Lit un tableau de caractères
inputStreamReader	transforme un flot de byte et car. Unicode
FileReader	Transforme des bytes lus depuis un fichier en car.
FilterReader	Classe abstraite pour filtrer les caractères d 'entrée
PipedReader	Lecture depuis PipeWriter
StringReader	Lit depuis une chaîne de caractères

Récapitulation

Flot de caractères	Description
Writer	Classe Abstraite pour flot de sortie (OutputStream)
BufferedWriter	Bufférise les sorties ligne par ligne
CharArrayWriter	Ecrit un tableau de caractères
inputStreamWriter	transforme un flot de car. Unicode en flot de bytes
FileWriter	Transforme un fichier de car. En flot de bytes
printWriter	Ecrit des valeurs et des objs dans un Writer
pipedWriter	Ecrit dans un PipeWriter
StringWriter	Ecrit dans une chaîne de caractères

La sérialisation

- Signifie persistance :
  - A la fin de l 'exécution, les données ne sont pas perdues
  - Vient du monde Unix et Windows (fichier = suite d 'octet en série)
  - Seules les données sont sauvegardées
  - Si un objet contient un autre objet, ce dernier est aussi sérialisé, et on obtient un arbre de sérialisation

Sérialisation (2)

- Les objets transients
  - Ce sont des objets qui ne peuvent pas être sérialisés
  - exemple : Thread ou FileXXXputStream
  - Si ils sont inclus dans un objet sérialisé, il faut indiquer qu 'ils sont transient
  - idem si on ne veut pas que certaines parties d 'un objet soient sérialisées

Syntaxe pour la sérialisation

- La classe doit implémenter l 'interface serializable (pas de méthode à implémenter, c 'est un marqueur). Les objets sont sauvegardés à l 'aide de writeObject(Object)
- Exemple :

```
import java.io.*;
class maclasse implements Serializable {
      public transient Thread mathread;
      private String nom;
      private int total;
      ...
```

Écriture d ’un objet persistant

```
import java.util.*;
import java.io.*
public class EcritObjPers {
    Date d = new Date();
    FileOutputStream f;
    ObjectOutputStream s;
    try {
        f = new FileOutputStream(« date.ser »);
        s = new ObjectOutputStream(f);
        s.writeObject(d);
        s.close();
    } catch (IOException e) {}
}
}
```


Lecture d ’un objet persistant

```
import java.util.*;
import java.io.*
public class LiitObjPers {
    Date d = null;
    FileInputStream f;
    ObjectInputStream s;
    try { f = new FileInputStream(« date.ser »);
        s = new ObjectInputStream(f);
        d=(date)s.readObject();
        s.close();
        System.out.println(« date : »+d);
    } catch (IOException e) {}
}}
```

Les bases du langage Java

Le clonage d 'objet

Cloner un objet

- Dans certains cas il est nécessaire de copier (on dit cloner) un objet
  - Par exemple, si une méthode souhaite manipuler les valeurs d'un objet qui lui a été passé en paramètre, mais ne souhaite pas modifier l'objet lui-même, elle doit commencer par cloner le paramètre

Copie de surface ou copie en profondeur

- Si l'état d'un objet contient des objets, un clonage de l'objet peut se contenter de copier l'adresse de ces objets (copie de surface) ou faire un clonage de ces objets (copie en profondeur)
- Le plus souvent, il est nécessaire de faire une copie en profondeur

Méthode clone() de la classe Object

- Permet le clonage de l'objet courant
- Effectue un clonage de surface :
  - elle crée un nouvel objet dont la classe et les variables d'instance sont les mêmes que l'objet courant
- Elle est protected, On ne peut donc l'utiliser sans créer une sous-classe qui redéfinit clone()
  - les nouvelles classes que l'on écrit ne sont pas dans le même paquetage que Object

Interface cloneable

- Un objet ne peut être cloné que si c'est une instance d'une classe qui implémente l'interface Cloneable
  - Si la classe de l'objet courant n'implémente pas Cloneable, la méthode clone() de Object renvoie une exception CloneNotSupportedException
- Cette interface est vide ; elle sert seulement de " drapeau " pour indiquer si un objet peut être cloné ou non

Comment permettre le clonage

- La classe doit
  - implémenter l'interface Cloneable
  - rendre public la méthode clone()
  - redéfinir la méthode clone() en effectuant une copie profonde
- Et si on le souhaite
  - lancer une exception CloneNotSupportedException au cas où la copie profonde rencontrerait un objet non clonable

Exemple de méthode clone() (copie complète)

```
public class Commande implements Cloneable {
  . . .
   public Object clone() throws CloneNotSupportedException {
        Commande nouvCommande =
   (Commande)super.clone();
        nouvCommande.lignesCommande =
   (Vector)lignesCommande.clone();
        return nouvCommande;
    }
    . . .
    }
```


Les bases du langage Java

Ecriture d 'Applets

Une Applet…

- N ’a pas de point d ’entrée main
- Ne peut être exécutée que depuis un Navigateur
  - En réalité, le Navigateur contient en interne
    - une machine virtuelle
    - sa propre méthode main

Le tag APPLET

- But: inclure dans un document HTML un espace pour l'exécution d'une petite application.

<APPLET
[CODEBASE = *localisation_programme*]
CODE=*nom_fichier_programme*
WIDTH=*largeur_fenêtre*
HEIGHT=*hauteur_fenêtre*
autres>
<PARAM NAME=*nom1* VALUE=*valeur*1>
<PARAM NAME=*nom2* VALUE=*valeur*2>
… </APPLET>

- Le tag PARAM permet d'envoyer des paramètres

Cycle de vie d'une applet

- Les applets sont sous le contrôle du navigateur WWW (Netscape, Hot Java, etc.)
- L'interface décide quand charger les applets d'une page HTML (=> état inactif)
- L'interface (re)démarre une applet quand sa fenêtre est visible sur l'écran (=> état actif)
- L'interface arrête l'applet qand elle disparaît de l'écran (=> état inactif)
- L'interface efface l'applet quand elle n'en a plus besoin

La classe Applet

```
Object
    +----- Component { paint(); resize(); … }
                +----- Container
                            +----- Panel
                                        +--- Applet
```

- Une applet est donc un *objet graphique*.
- Mais c'est aussi un *objet actif* créé et contrôlé par le navigateur Web

2 catégories de méthodes

- les méthodes d'interface graphique (héritées):

public void paint(Graphics g);
public boolean mouseDown(Event evt, int x, int y);
public boolean action(Event evt, Object what);
etc.
etc.

- les méthodes de contrôle d'exécution

public void init();
public void start();
public void stop();
public void destroy();
etc.

Ecrire une applet

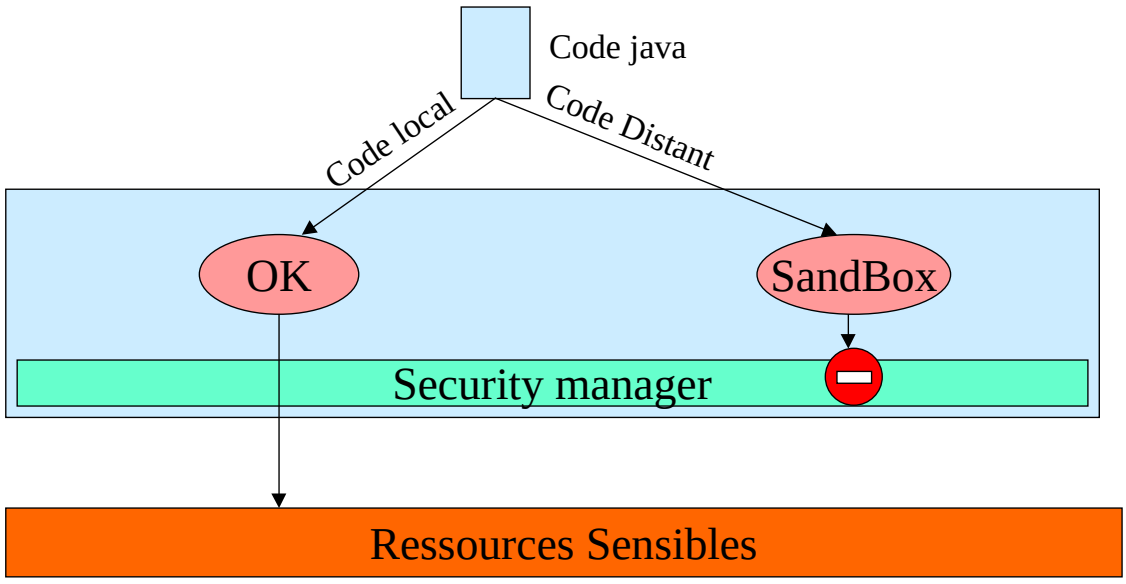
```
import java.applet.*; import java.awt.*;
public class <NomApplet> extends Applet {
public void init() {
    <Initialisations> <Démarrage de processus> }
public void start() {
    <Démarrer l'applet, la page Web est visitée> }
public void paint(Graphics g) {
    <Dessiner le contenu actuel de l'applet> }
public void stop() {
    <Arreter l'applet, la page Web est quittée> }
public void destroy() {
    <Relâcher les ressources>}
```

Exemple minimal

```
import java.applet.*;
import java.awt.*;
public class MonApplet extends Applet {
// Applet minimale : redéfinit que la méthode paint()
public void paint(Graphics g) {
    int px[] = {5, 10, 0, 20, 40, 20};
    int py[] = {5, 20, 40, 60, 40, 5};
    g.drawPolygon(px, py, 6);
    g.drawString("Une applet !", 50, 50);
}}
```

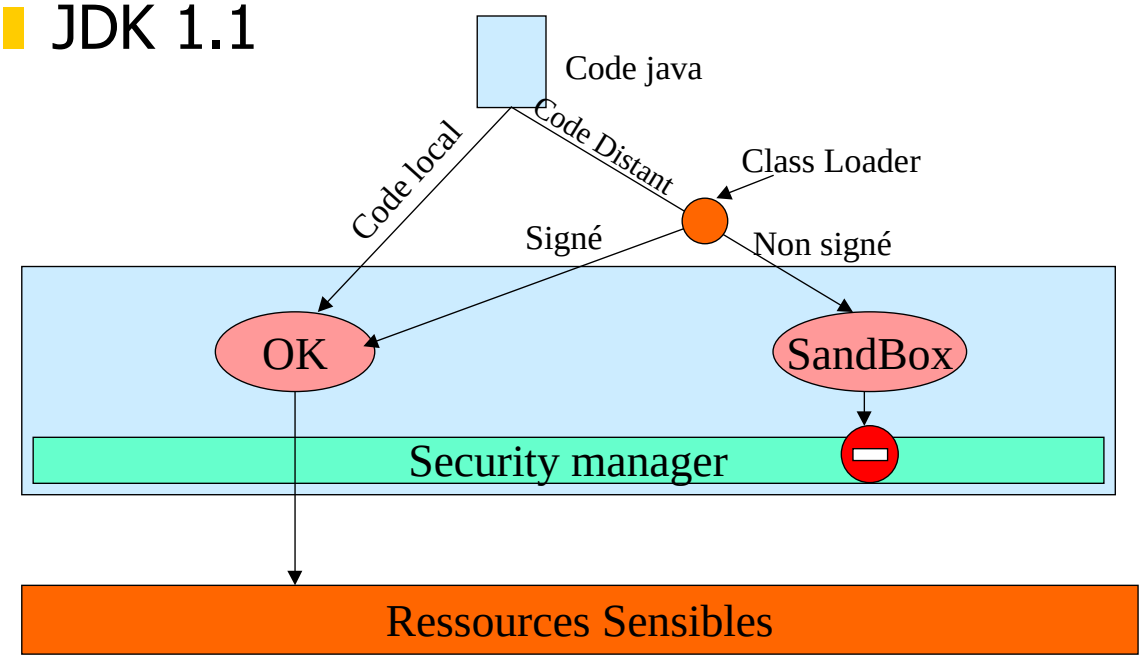

Sécurité et code distant

- JDK 1.0 : le sandbox



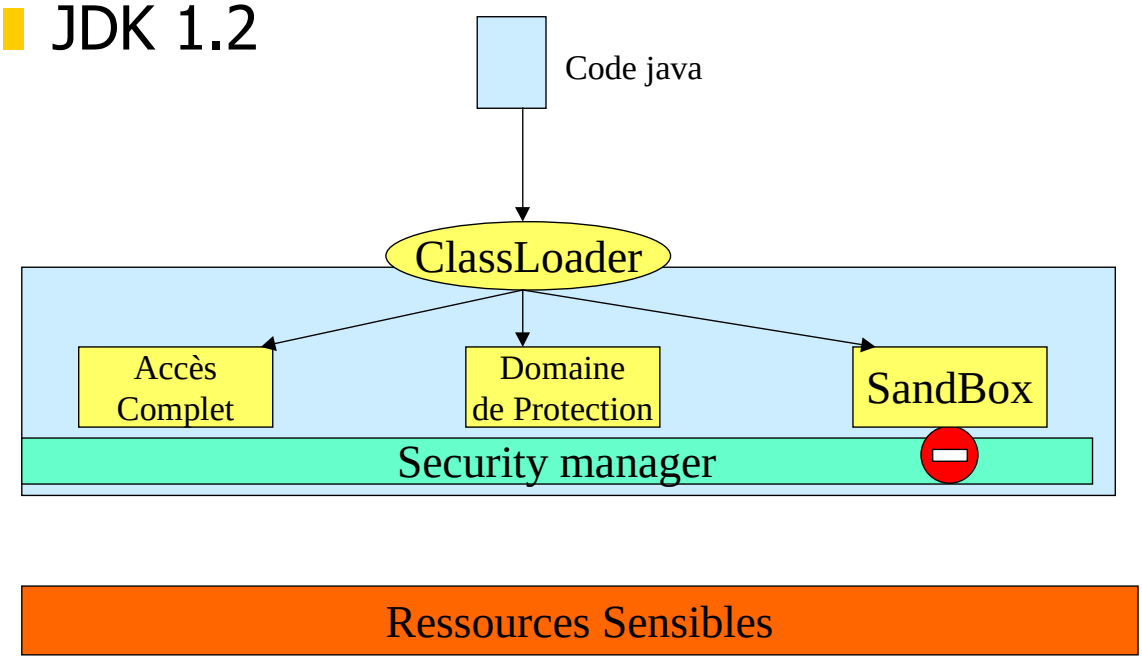
Evolution du modèle

- JDK 1.1



Encore Plus loin...

- JDK 1.2



- JDK 1.3 en cours de définition

Java : Aspects Avancés

Les Threads

Principe du MultiThreading

- Programmation Parallèle
  - Plusieurs activités simultanées
    - Simplification de Programmation Evénementielle
      - GUI, Serveur, ...
    - Meilleure utilisation du matériel
- Multi Processus
  - Espaces de Ressources séparés
  - Partage fortement contrôlé par le noyau
- Multi Thread
  - un Espace de Ressources commun
    - seules les piles d 'exécution sont privés
  - peu de contrôle système : efficacité de gestion

Multithreading (MT) dans Java

- Systèmes Multithreadés
  - POSIX : Solaris, OSF, NT, (Win95)
  - En général, le MT n 'est pas intégré à un langage
    - exception : ADA95, OCCAM
- Java Virtual Machine
  - les threads font partie du noyau de la JVM
- API Java
  - Paquetage `java.thread.*`
  - instructions `synchronized`
  - méthodes `Object.wait(), Object.notify()`

S. Lecomte, V. Poirriez, D. Donsez UVHC/ISTV

Réalisation d 'un Thread

- Définition d 'un classe dérivée de Thread
  - exemple : class maclasse extends Thread {...
- Définition d 'une classe comme implémentation de l 'interface Runnable
  - exemple : class maclass implements Runnable {...
  - C 'est la seule solution pour crée, par exemple, une applet « threadée » :
    class monapplet extends applet implements Runnable

S. Lecomte, V. Poirriez, D. Donsez UVHC/ISTV

Exemple de thread

```
class Crawdad implements Runnable {
  String name;
public Crawdad(String name) { this.name = name; }
public void run() {
   while (true) {
     try {Thread.currentThread().sleep(1000);}
     catch (InterruptedException e) {}
     System.out.println(name);
   }}}
```


Exemple de Thread

```
public class CrawdadRace {
  public static void main (String args[]) {
     new Thread(new Crawdad("Octavius")).start();
     new Thread(new Crawdad("Augustus")).start();
     System.out.println("And they're off!");}
}
```

Exécution:

```
And they're off!
Octavius
Augustus
Octavius
Augustus
...
```

Run et Start

- La méthode run()
  - Couramment dans une boucle infinie
- La méthode start()
  - Est utilisée pour lancer le Thread
  - Elle appelle la méthode run()

accès simultanés à une ressource :

- Gérer les concurrences d'accès à une méthode, déclarer la méthode synchronized
  - Lorsqu'un thread t1 exécute cette méthode sur un objet, un autre thread t2 ne peut pas l'exécuter pour le même objet. En revanche, t2 peut exécuter cette méthode pour une autre instance de la même classe.

```
public synchronized void maMéthode() { ... }
```

accès simultanés à une ressource

- Contrôler l'accés à un objet déclarer l'objet synchronized .

```
public void maMéthode() {
                    ...
                    synchronized(objet) {
                    objet.méthode();
                    }
                    ...
                    }
```


Synchronisation et efficacité

- Chaque objet possède un verrou *lock* ou *moniteur*. (Cela fait partie de la classe Object).
- Lorsqu'une méthode synchronisée d'un objet est appelée, le verrou est mis, aucune autre méthode synchronisée de cet objet peut être exécutée.
- Acquérir le verrou d'un objet est coûteux. Pour éviter une dégradation des performances il faut que les sections critiques soient courtes et utilisées à bon escient. Un appel à une méthode synchronisée coûte au moins 4 fois plus cher.

Les états d 'un Thread

- Nouveau : il est créé mais start() n 'est pas encore appelé
- Exécutable : il a été initialisé (il est exécuté si l 'ordonnanceur le décide)
- Mort : fin normale d 'un Thead (sortie de la méthode run()
- Bloqué : l 'ordonnanceur ignore le Tread

Raison d 'un blocage

- Le thread peut être :
  - endormi : sleep(milliseconds)
  - Suspendu : par suspend(). Il sera alors relancé par resume()
  - en attente : par wait(). Il pourra alors être réactivé par notify() ou notifyAll()
  - IO : en attente d 'une Entrée/Sortie
  - synch : en attente de la libération d 'une ressource synchronisée

Différence Wait/Suspend

- Suspend :
  - Tient le verrou de l 'objet
  - Bloque dons les méthode synchronisées
  - Source de DeadLock
- Wait
  - ne tient pas les verrous
  - ne doit pas être appelé d 'un bloc synchronisé, car sinon, risque d 'accès concurrent

Maintien en cohérence...

- Il faut éviter l 'utilisation de :
  - stop : cette méthode libère les verrous même si l 'objet est dans un état instable
    - Il faut le remplacer par un test dans la boucle du run()
    - lever l 'exception InterrupedException en utilisant la méthode interrupt()
  - suspend (et donc resume), car ils sont sources de nombreux deadlock

ne pas créer trop de threads

upper bound ~ 100

- Citations de *thinking in java*:
- If you see performance problems in a multithreaded program you now have a number of issues to examine:
  - 1.Do you have enough calls to sleep( ), yield( ), and/or wait( )?
  - 2.Are calls to sleep( ) long enough?
  - 3.Are you running too many threads?
  - 4.Have you tried different platforms and JVMs?
- Issues like this are one reason that multithreaded programming is often considered an art.

Inconvénients majeurs du MultiThreading

- 1.Slowdown while waiting for shared resources
- 2.Additional CPU overhead required to manage threads
- 3.Unrewarded complexity, such as the silly idea of having a separate thread to update each element of an array
- 4.Pathologies including starving, and deadlock

Java : Aspects Avancés

La programmation Réseau

Quelques Rappels...

- Définition d 'un socket :
  - point d 'entrée entre 2 applications du réseau
- Les types de sockets
  - TCP : Stream Socket
    - Communication en mode connecté
    - Applications prévenues de la déconnexion
  - UDP : Datagram Socket
    - Communication en mode non connecté
    - Plus rapide mais moins fiable que TCP
  - Raw Socket
    - Accès direct aux couches basses (debogage de protocole)
    - Non implémenté en Java

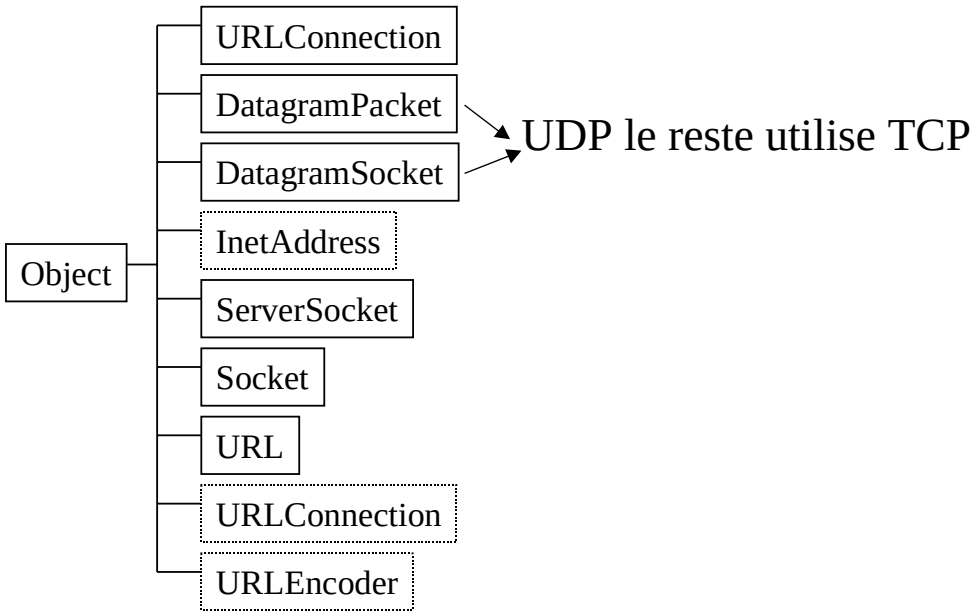
Domaines de socket

- Domaine Unix :
  - Rôle : communication inter-processus
  - TCP et UDP supportés, pas Raw socket
- Domaine Internet :
  - Rôle : communication à travers le réseau
  - Tous les modes sont supportés
  - Une fois la communication établie, les données sont lues et écrites comme dans un fichier

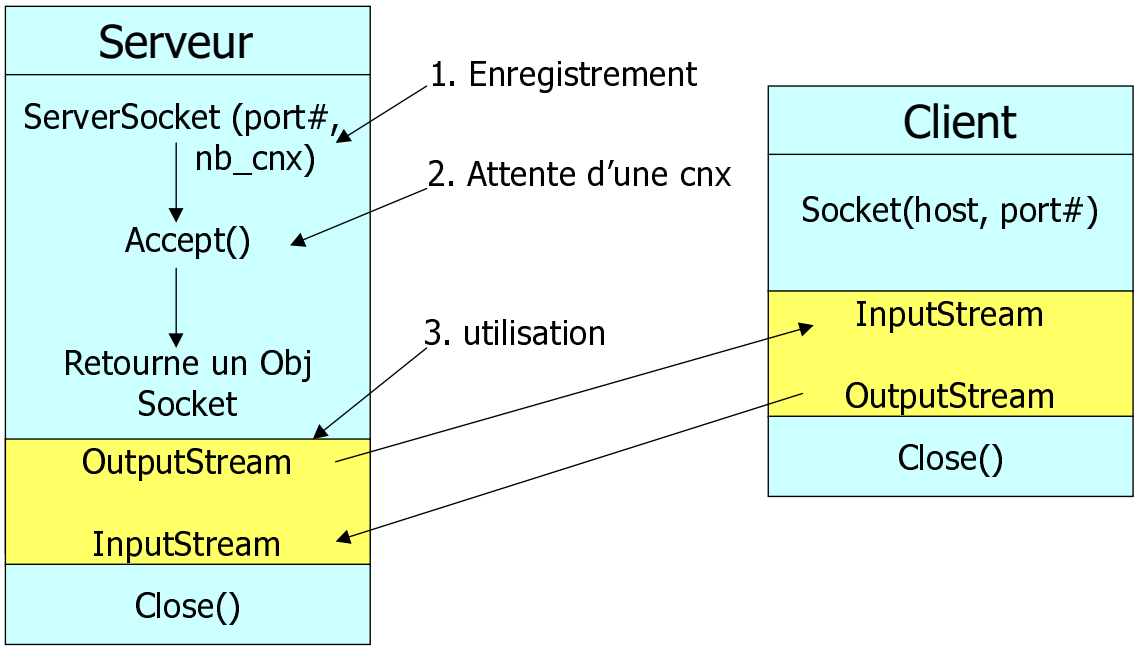
Le domaine Internet

- Applications TCP :
  - FTP, SMTP, Telnet, etc…
  - Ce sont les plus utilisées
- Application UDP :
  - Version datagram de FTPD pour le boot par réseau
- Raw socket : manipulations bas niveau
  - ping

Programmation Réseau : java.net



Fonctionnement d 'un Client/Serveur



java.net.ServerSocket

- Cette classe implémente une socket TCP coté serveur.

```
int port_d_ecoute = 1234;
ServerSocket serveur = new ServerSocket(port_d_ecoute, nbcnx);

while(true)
{
    Socket socket_de_travail = serveur.accept();
    new ClasseQuiFaitLeTraitement(socket_travail);
}
```


Utilisation des Datagrams

- Classes DatagramPacket et DatagramSocket
- initialisation différente pour envoyer ou recevoir
  - Envoie :
    - le DatagramPacket est créé en spécifiant les données, leur longueur, la machine réceptrion et le port.
    - Après, utilisation de la méthode sent(DatagramPacket) de la classe DatagramSocket
  - Réception :
    - On créé un DatagramSocket qui écoute sur le bon port du host destinataire et un DatagramPacket avec un buffer suffisamment grand
    - On utilise la méthode receive() de la classe DatagramPacket (bloquant)

java.net.DatagramSocket: Envoie

- Cette classe implémente une socket UDP

```
// Client
Byte[] data = "un message".getBytes();
InetAddress addr = InetAddress.getByName("titan.univ-
    valenciennes.fr");

DatagramPacket packet = new DatagramPacket(data,
    data.length, addr, 1234);

DatagramSocket ds = new DatagramSocket();

ds.send(packet);
ds.close();
```


S. Lecomte, V. Poirriez, D. Donsez UVHC/ISTV

java.net.DatagramSocket: Réception

```
// Serveur

DatagramSocket ds = new DatagramSocket(1234);

while(true)
{
    DatagramPacket packet =
                    new DatagramPacket(new byte[1024], 1024);
    s.receive(packet);
    System.out.println("Message: " + packet.getData());
}
```


S. Lecomte, V. Poirriez, D. Donsez UVHC/ISTV

java.net.URL

```
URL url = new URL("http://www.univ-
    valenciennes.fr/index.html");

DataInputStream dis = new DataInputStream(url.openStream());

String line;
while ((line = dis.readLine()) != null)
    System.out.println(line);
```


Java : Aspects Avancés

La programmation Graphique

Java Abstract Window Toolkit
java.awt

Ce cours est inspiré de :

Olivier Dedieu
BULL / INRIA

Généralités

- AWT comporte :
  - des éléments graphiques
  - des conteneurs
  - Un mécanisme de gestion d'événements, ...
- AWT a beaucoup evolué entre 1.0.2 et 1.1.1 :
  - nouveau modèle evenementiel
  - gestion de l'impression
  - gestion de l'echange de données
- AWT va encore évoluer avec *Java Foundation Class* et *java.2D*

Généralités

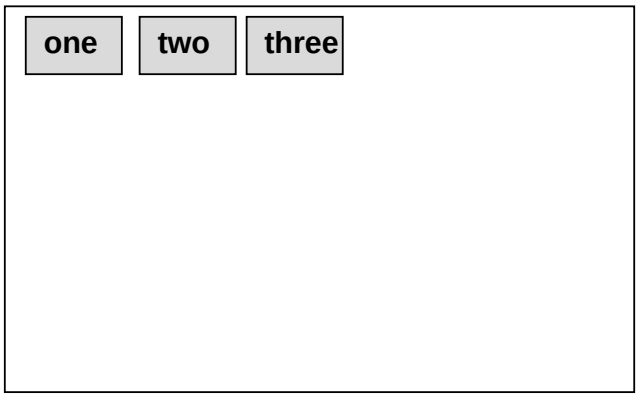
- Les éléments graphiques
  - java.awt.Button
  - java.awt.Canvas (zone de dessin)
  - java.awt.Checkbox
  - java.awt.Choice
  - java.awt.Label
  - java.awt.List
  - java.awt.MenuBar
  - java.awt.MenuItem
  - java.awt.TextArea
  - java.awt.TextField

Généralités

- Les conteneurs
  - Frame
  - Dialog
  - Window
  - Panel (conteneur de base)
- Les Gestionnaires de mise en page
  - Flowlayout, Gridlayout, etc...

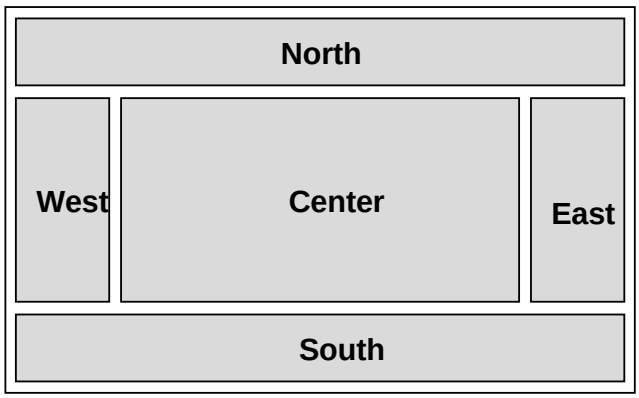
Généralités

- java.awt.FlowLayout :



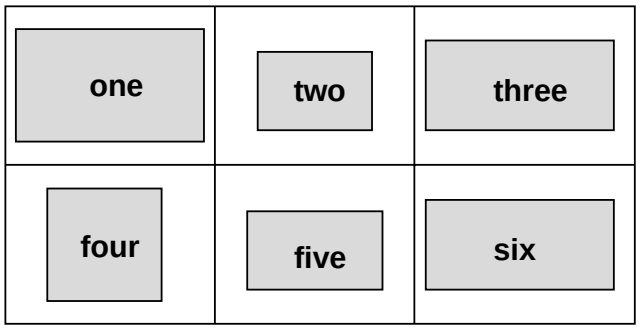
Généralités

- java.awt.BorderLayout



Généralités

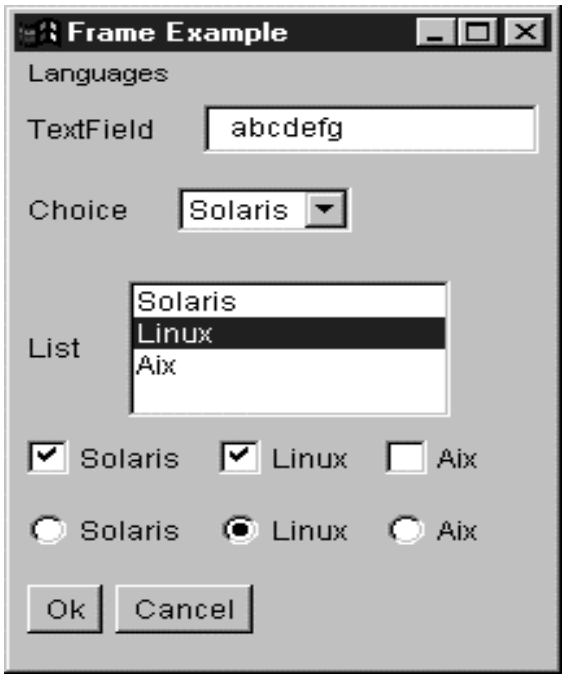
- java.awt.GridLayout



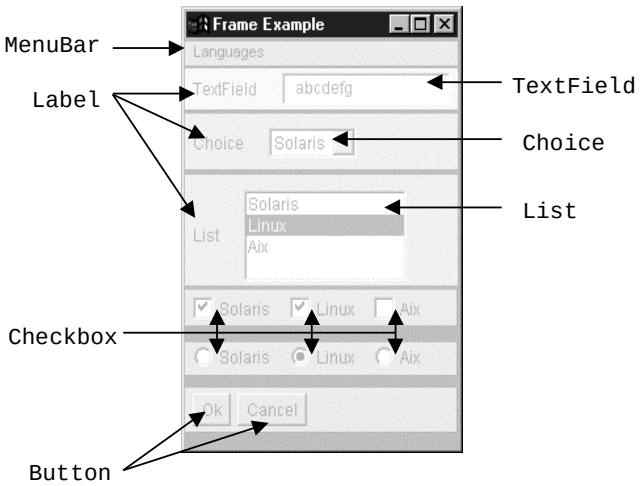
Buts

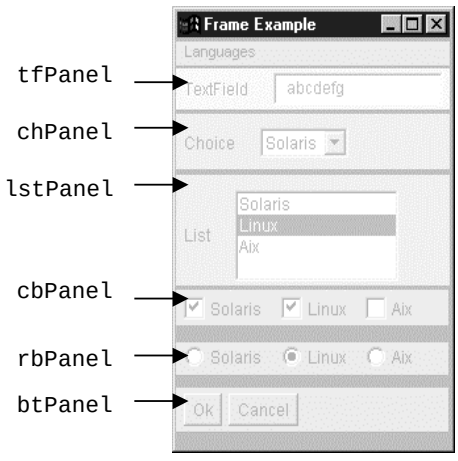
- S 'abstraire des contraintes du système
  - permet de créer une interface graphique d 'exécutant sous toutes les interfaces (Windows, Xwindows, ...)
- Simplifier la tâche du concepteur de l 'IHM
  - Grand nombre d 'objets prédéfinis

Exemple



Exemple





Exemple

```
// Text Field Panel
Panel tfPanel = new Panel(new FlowLayout(FlowLayout.LEFT));

label = new Label("TextField");
tfdPanel.add(label);

textField = new TextField(15);
tfPanel.add(textField);
```


Exemple

```
// Choice Panel
Panel chPanel = new Panel(new FlowLayout(FlowLayout.LEFT));

label = new Label("Choice");
chPanel.add(label);

choice = new Choice();
choice.addItem("Solaris");
choice.addItem("Linux");
choice.addItem("Aix");

chPanel.add(choice);
```

Exemple

```
// List Panel
Panel listPanel = new Panel(new FlowLayout(FlowLayout.LEFT));

label = new Label("List");
listPanel.add(label);

list = new List();
list.addItem("Solaris");
list.addItem("Linux");
list.addItem("Aix");

listPanel.add(list);
```


Exemple

```
// Checkbox Panel
Panel cbPanel = new Panel(new FlowLayout(FlowLayout.LEFT));

cbSolaris = new Checkbox("Solaris");
cbPanel.add(cbSolaris);

cbLinux = new Checkbox("Linux");
cbPanel.add(cbLinux);

cbAix = new Checkbox("Aix");
cbPanel.add(cbAix);
```

Exemple

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


Exemple

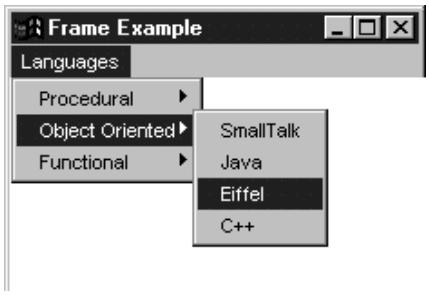
```
// Button Panel
Panel btPanel = new Panel(new FlowLayout(FlowLayout.LEFT));

okButton = new Button("Ok");
okButton.addActionListener(new OkButtonListener());
btPanel.add(okButton);

cancelButton = new Button("Cancel");
cancelButton.addActionListener(new CancelButtonListener());
btPanel.add(cancelButton);
```

Exemple

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



Exemple

```
public class FrameExample extends Frame {
   public FrameExample(String title) {
     super(title);
   // Widget, panel, menu bar creation
   ...
     setMenuBar(mb);
     setLayout(new GridLayout(0, 1, 2, 2));
     add(textFieldPanel);
     add(choicePanel);
     add(listPanel);
     add(checkboxPanel);
     add(radioButtonPanel);
     add(buttonPanel);
     pack();
     show();}
```

Exemple

- Gestion des événements : clic sur le bouton Ok

```
okButton.addActionListener(new OkButtonListener());

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


Exemple

- Gestion des événements : séléction d'items du menu

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

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

Exemple

- Gestion des événements : fermeture de la fenêtre

```
addWindowListener(new MyWindowListener());

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


Swing : Qu 'est ce que c 'est ?

- Une partie de JFC de Netscape
- Une librairie graphique
  - Complémentaire des classes AWT
  - Fournit de nombreux composants
  - livrée avec le JDK 1.2, mais peut fonctionner au dessus du JDK 1.1
    - téléchargeable sur http://java.sun.com/products/jfc

Nouvelles Fonctionnalités

- Nouveaux Composants
  - Bouton, Ascenseur, Table, Arbre,
  - Fenêtre interne, Barre de progression,
  - Menu déroulant, Barre d 'outil, ...
- Possibilité de masquer certains événements
- Look-and-Feel paramétrable et customisable
  - Windows, Motif, Metal, Mac
    - le paramétrage peut intervenir en cours d 'exécution
- Deboggage simplifié

Utilisation de Swing

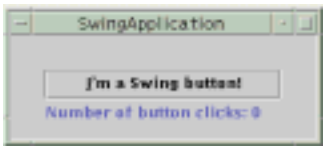
- 1er point : si on utilise le JDK 1.1.x, il faut inclure swingall.jar dans le CLASSPATH
- Dans l 'entête de la classe, on importe toutes les classes Swing
  - import javax.swing.*
- Dans le programme, on utilise les bojets des classes Swing :
  - Exemple JButton au lieu de Button

Un exemple...

```
import javax.swing.*; import java.awt.*;import java.awt.event.*;
public class SwingApplication {
    private static String labelPrefix = "Number of button clicks: ";
    private int numClicks = 0;
    public Component createComponents() {
        final JLabel label = new JLabel(labelPrefix + "0    ");
        JButton button = new JButton("I'm a Swing button!");
        button.setMnemonic(KeyEvent.VK_I);
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                numClicks++;
                label.setText(labelPrefix + numClicks);
            }});
        label.setLabelFor(button); }
```

Exemple Suite...

```
public static void main(String[] args) {
        try { UIManager.setLookAndFeel(
                UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) { }
        //Create the top-level container and add contents to it.
        JFrame frame = new JFrame("SwingApplication");
        SwingApplication app = new SwingApplication();
        Component contents = app.createComponents();
        frame.getContentPane().add(contents, BorderLayout.CENTER);
        //Finish setting up the frame, and show it.
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0); } });
        frame.pack(); frame.setVisible(true); }
```



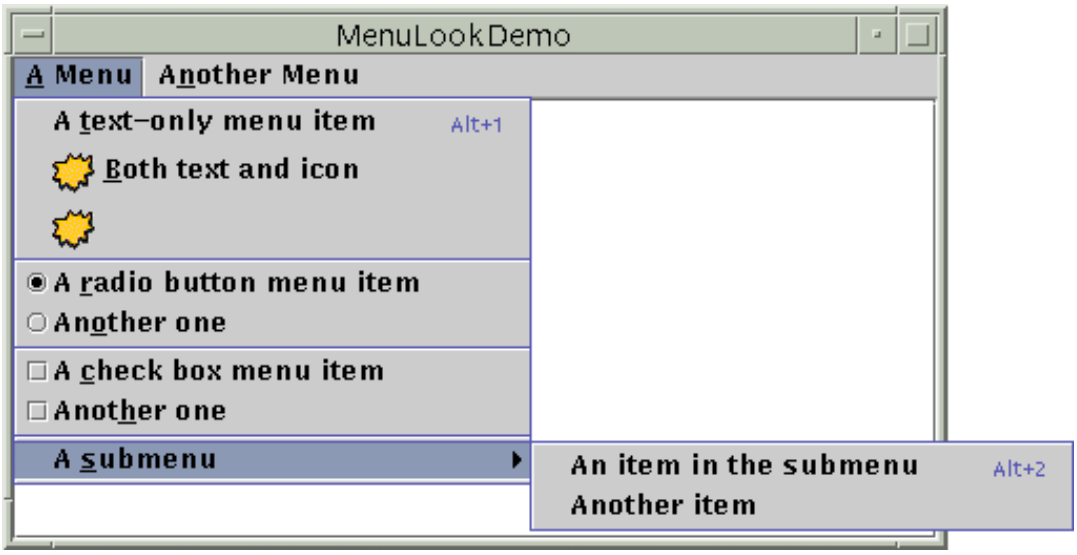
Explication d ’un exemple

- Cette application créé 4 composants Swing :
  - Une fenêtre (JFrame), un panel (JPanel), un bouton (JButton) , une étiquette (JLabel)
- La frame est un container de haut niveau. Les autres possibilités sont des boites de dialog (JDialog) ou des Applets (JApplet)

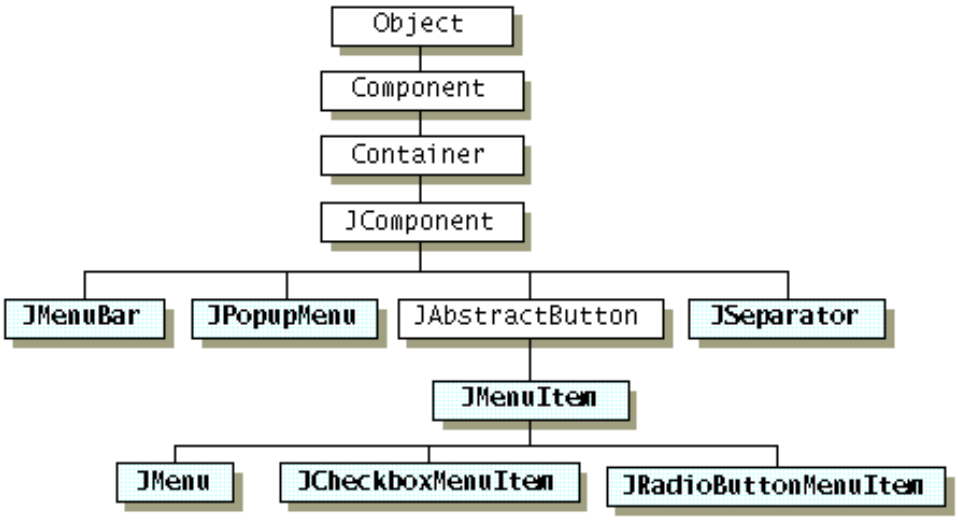
Swing dans une applet

```
import javax.swing.*;
import java.awt.*;
public class HelloSwingApplet extends JApplet {
    public void init() {
        JLabel label = new JLabel(
                        "You are successfully running a Swing applet!");
        label.setHorizontalAlignment(JLabel.CENTER);
        //Add border.
label.setBorder(BorderFactory.createMatteBorder(1,1,2,2,Color.black));
        getContentPane().add(label, BorderLayout.CENTER);
    }
}
```

Un autre exemple : un menu



Hierarchie ...



Les « menu item » sont en fait des boutons...

Conclusion sur Swing

- Swing offre surtout beaucoup plus de composants que AWT :
  - Le rendu est meilleur
  - Les options nombreuses
- Mais l 'API est difficile à connaître
- ne pas hésiter à utiliser le Tutorial :
  - jdk1.2.2/tutorial1.2/uiswing/components/components.html

Java Native Interface (JNI)

- Motivations
  - Un programme Java accède à des fonctions écrites en C (et compilées)
    - ☺ récupération de sources existants, performance
    - ☹ portabilité, mobilité, sécurité
  - Un programme en C (C++) lance une machine virtuelle et invoque la méthode main() d 'une classe Java

Java Native Interface (JNI)

- JNI définit
  - l 'édition de lien de la JVM avec la bibliothèque qui contient les fonctions
  - le passage des paramêtres et la récupération de la valeur de retour avec la correspondance des types

- Remarque : MicroSoft JVM supporte
  RNI (Raw NI) et Java/COM

JNI
Etapes de Développement

- **1) Définir les classes Java et leurs méthodes natives**

```
// file HelloWorld.java
class HelloWorld {
  public native void displayHelloWorld();
  static {
    System.loadLibrary("hello");
  }
}
// file Main.java
class Main {
  public static void main(String[] args) {
    new HelloWorld().displayHelloWorld();
  }
}
```

JNI
Etapes de Développement

2) Créer un fichier .h

```
C:\> javah HelloWorld
// file HelloWorld.h
#include <native.h>
typedef struct ClassHelloWorld { char PAD; } ClassHelloWorld;
extern void HelloWorld_displayHelloWorld(struct HHelloWorld *);
```

3) Créer un fichier stub

```
C:\> javah -stubs HelloWorld
#include <StubPreamble.h>
stack_item *Java_HelloWorld_displayHelloWorld_stub
            (stack_item *_P_,struct execenv *_EE_) {
        extern void HelloWorld_displayHelloWorld(void *);
        (void) HelloWorld_displayHelloWorld(_P_[0].p);
        return _P_;
}
```


JNI
Etapes de Développement

4) Ecrire le code des méthodes natives

```
#include <StubPreamble.h>
#include "HelloWorld.h"
#include <stdio.h>
void HelloWorld_displayHelloWorld(struct HHelloWorld *this) {
  printf("Hello World!\n");
  return;
}
```

5) Créer une bibliothèque (shared lib ou DLL)

JNI
Etapes de Développement

6) Exécuter le programme Java

- Une exception se produit

```
java.lang.NullPointerException
    at java.lang.Runtime.loadLibrary(Runtime.java)
    at java.lang.System.loadLibrary(System.java)
    at HelloWorld.(HelloWorld.java:5)
    at
        java.lang.UnsatisfiedLinkError displayHelloWorld
            at Main.main(Main.java:3)
```

- Configurer le chemin d'accès à libhello.so

```
% setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:mylibrarypath
```


JNI
Ecriture de Méthodes Natives

- Signature des Méthodes
- Passage d'arguments
- Valeur de Retour
- Manipulation des Chaînes
- Accès aux membres d'un objet Java
- Appel de méthodes d'un objet Java
- Levée d'une Exception
- Synchronisation des Threads

Correspondance de Types le codage UTF

- Types

Java Type	Native Type	Description
boolean	jboolean	unsigned 8 bits
byte	jbyte	signed 8 bits
char	jchar	unsigned 16 bits
short	jshort	signed 16 bits
int	jint	signed 32 bits
long	jlong	signed 64 bits
float	jfloat	32 bits
double	jdouble	64 bits
void	void	N/A

Correspondance de Types le codage UTF

- Codage UTF8 de chaîne
  - Rappel : le type char de Java occupe toujours 16 bits (Unicode)
  - UTF8 permet de coder les chaînes de caractères avec des symboles de taille variable (1 à 3 octets)