

# Java pour les ZZ

F. Chadebecq Version étudiante

B. Chapel

A. Harb

R. Martin

L. Yon

Éditions  
**ISIMA**



<http://www.isima.fr/loic>

Tout ce qu'il faudrait savoir !!!



## AVERTISSEMENT



### THESE SLIDES ARE RATED ZZ

- Le support original est présenté dans un cours
  - Certains transparents comportent volontairement des erreurs. Certaines peuvent aussi être involontaires
  - Il est fortement recommandé d'essayer les exemples proposés.
- 
- La page de garde est honteusement copiée des ouvrages des éditions First.
  - Ce cours est libre et réutilisable pour tout à chacun sous la licence EMAILCOUCOUWARE ;-)

## Cadre

- Le Java vu par ~~SUN~~ et uniquement **ORACLE**
- Préparer les premières certifications
- Exception : utiliser l'EDI Eclipse

## Évaluation

- Présence
- Examen final



- Île ?
- Javascript (ECMA) ?

- Langage
- Machine virtuelle
- Plateforme



## Plan



- Introduction
- Premier programme
- Notions de base & syntaxe
- Concepts objets en Java
- Les exceptions
- IHM : AWT & Swing
- Concurrence (Threads)
- Fichiers & flux, sérialisation  
+ Compléments

**ISIMA**



### 1. Introduction



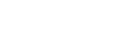
## Introduction

- Simple
- Sécurisé (réseaux, Internet)
- Portable
- Performant



- Île ?
- Javascript (ECMA) ?

- Langage
- Machine virtuelle
- Plateforme



## Motivations

- Langage proposé par (1995)
- Tout objet
  - Constantes ?
  - Variables globales ?
  - Fonctions globales ?



→ Tout est encapsulé dans des classes



## Objectifs

- Écrire un programme JAVA
- Utiliser les concepts objets avec JAVA
- Concevoir des IHM

## Pré-requis

- Syntaxe C/C++
- Concepts objets





## Paquetage / package (1)

- Un ensemble de classes/fichiers rassemblés pour une finalité  besoin fonctionnel
- [C++] espace de nommage
- Différents types :
  - Par défaut (java.lang)
  - Standard (gestion E/S, graphisme)
  - Personnel / Tiers

19



## Paquetage / package (2)

- Nom spécifique suivant le type :
  - `java.lang (sys)`, `java.awt (std)`
  - `javax.swing (std)`, `javax.xml (std)`
  - `org.w3c.dom (tiers/std)`
  - `loic.classeperso (perso)`
- Nom en chemin d'un package
  - Mécanisme arborescent comme les répertoires
  - Séparateur : le point
- Retrouver les chemins : `classpath`
  - Variable système,
  - Paramètres en ligne de commande (-cp ou -classpath)

20



## Plateforme (1)



- Packages « standards »
- Relativement à une version de java
  - 1.0 (1.1) - *applet, jni, awt* [1995]
    - 236 classes pour 1.0.2
  - 1.2 - *swing* (version 2) [1998]
    - 1524 classes
  - 1.3 - débogage

```
java -showversion
javac -version
// version > 1.3, options : -source et -target
```



22

## Java en 2011 ?

- Rachat de Sun par **ORACLE** en 2010
- Concurrents :
  - Plateforme .net, C#
  - Frameworks comme RubyOnRails
- Java FX
  - Concurrents : Adobe Air, Microsoft Silverlight
- Vers Java  8 open source GPL

25



## Plateforme (2)



- Relativement à une version de java
  - 1.4 – performances - *nio*
  - 1.5 - patrons / *templates* [2004]
    - 3279 classes
  - 1.6 – sécurité, scripts, performance [2006]
    - 3795 classes

23

24

## Plateforme (3)

- ⌚ Plus grosse difficulté du java
  - ➡ connaître ces classes standards
  - classes *deprecated*
- 😊 Documentation bien faite (javadoc)



- 2 versions distribuées
  - Exécution seule (JRE)
  - Développement (JDK < 2, SDK v ≥ 2)
- Autres dénominations
  - J2SE, J2EE, J2ME - standard, entreprise, micro
  - Java SE, Java EE, Java ME (v ≥ 5)

26

## Clause import

- Spécification complète d'une classe d'un package qui n'est pas chargé par défaut
  - Ex : `javax.swing.SwingUtilities`
- Facilité : clause import

```
import java.io.StreamTokenizer;
import javax.swing.*;
import javax.swing.event.*;
import nom_ccomplet_lasse;
```

- Liste des classes utilisées
  - Énumération à l'unité
  - Import automatique (\*) non récursif ;-(

7

## Java 7

- Sorti en juillet 2011
- Attendu depuis 4 ans
- Simplification de la syntaxe (Coin)
  - switch, catch multiple, utilisation des <>
- NIO2
- Gestion des fichiers / répertoires (Path)
- Framework de parallélisation

24

ISIMA



3. Notions de base  
Syntaxe

26

## { Accolades }

```
public class Exemple
{
    public static void main(String[] argv)
    {
        // afficher un commentaire monoligne
        /* commentaire
         * sur plusieurs lignes */
        /** commentaire javadoc */
    }
}
• Classe
• Méthode
• Bloc : ensemble séquentiel d'instructions
```

28

## Attribut / Variable ?

- Objet
  - Prédéfini ou utilisateur
  - Chaîne de caractères : String "Essai"
  - Manipulation par « références » (pointeurs ?)
- Scalaire / Primitive
  - entier / réel / booléen
  - caractère 'A'
  - pour l'efficacité
  - doublé par un type objet

29

## Types de données scalaires (2)

- Normalisés (portabilité)
- Doublés par des types objets :
  - Double
  - Integer
- Méthodes
  - `toString()`
  - `Double.parseDouble()`

31

## Déclaration de variables

```
public static void main(String[] argv) {
    int    i = 0;
    char   c = 'A';
    Classe objet;
    String  s;
    double[] tableau;
}
```

- N'importe où dans le bloc
- Initialisation d'une variable **pas automatique**
  - Erreur : "might not be initialized"

32

## Types de données scalaires (1)

- char
  - type caractère
  - ≠ String
  - unicode '\u0000'
- boolean
  - true ou false.
  - non homomorphe aux entiers
- types entiers
  - byte (8 bits)
  - short (16 bits)
  - int (32 bits)
  - long (64 bits)
- types réels
  - float (32 bits)
  - double (64 bits)

30

```
{
    int i = 0 ;
    {
        int j= 3 ;
        // i est utilisable dans ce bloc
    }
    // j n'est plus disponible ici
}
```

```
{
    int    i = 0 ;
    boolean b = true;
    {
        double i= 3 ;
        boolean b = false;
    }
}
```

33

## Manipulation de variables

```
public static void main(String[] a) {
    int i = 0;
    i = i + 1;
    i += 1 ;
    i *= 2 ;

    System.out.println(i) ;
    System.out.println(++i) ;
    System.out.println(i) ;
    System.out.println(i++) ;
    System.out.println(i) ;

    i = (int) 10.6;
}
```

34

## Condition (1)

```
if (test) {
    ...
}
Test
(i==5)
(i!=5)
```

Opérateur ternaire

```
if (test) {
    ...
} else {
    ...
}
boolean b1 =(i==5);
boolean b2 = !b1;
```

Un test est un booléen :  
true ou false

35

## Condition (2)

```
if (test) instruction1;
else instruction2;

if (b1) ...
if (!b1)...          Opérateur NON

if (b1 || b2) ... Opérateur OU
if (b1 && b2) ... Opérateur ET ALORS
```

Une séquence de test n'est pas complètement évaluée si ce n'est pas nécessaire.

36

## Condition (3)

```
switch (variable) {  
    case valeur1 :  
        instructions;  
        break;  
    case valeur2 :  
    case valeur3 :  
        instructions;  
        break;  
    default:  
        instructions;  
        [break;]  
}
```

- Variable de type simple (`String` possible dans 1.7)
- Oubli du `break`? ( $\neq$  C#)
- `default` facultatif

37

## Boucles conditionnelles

```
for (initialisation;test;incrémentation) {  
    ...  
}  
  
for (int i=0;i<10;++i)  
    System.out.println(i);  
  
while (test) {  
    ...  
} Accolades facultatives  
s'il n'y a qu'une instruction  
  
do {  
    ...  
} while (test);
```

38

Variable de boucle et visibilité...

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

39

## Tableaux (1)

```
// création d'un tableau vide de 10 entiers  
int[] t1 = new int[10];  
// déclaration d'un tableau sans éléments  
int[] t2;  
  
for(int i=0; i<t1.length; ++i)  
    System.out.println(t1[i]);
```

- Taille fixe, donnée par le champ `length`
- Premier indice du tableau : 0
- Vérification de la validité des indices
  - Exception : `OutOfBoundsException`

40

## Tableaux (2)

- Initialisation du tableau
  - Par des valeurs scalaires
  - Par des références nulles
- "Libérer" un tableau

```
t1 = null;
```

```
t1[i] = ?;
```

- Tableau multidimensionnel

```
String[][] chaines = new String[10][5];
```

41

## Chaîne de caractères (1)

- `String`  $\neq$  `char[]`
- **statique** `String`
- **dynamique** `StringBuffer`  
`StringBuilder`
- UTF-16
- Bibliothèque fournie
  - Comparaison de chaînes : `equals()`, `compareTo()`
  - Recherche : `indexOf()`
  - Extraction : `substring()`,  `StringTokenizer`, `split`, `regexp`
  - Transformation aisée de type scalaire vers `String`

42

## Chaîne de caractères (2)

- Création de chaîne(s)

```
String s1 = "hello";  
String s2 = new String("hello");  
String s3 = null;
```

- Que se passe-t'il ?

```
String s3 = s1 + " " + s1;
```

43

```
String s1 = "loic" ;  
String s2 = "loic";  
String s3 = new String("loic");  
String s4 = new String("loic");  
String s5 = s3;  
String s6 = null;  
  
System.out.println(s1==s2);  
System.out.println(s1==s3);  
System.out.println(s3==s4);  
System.out.println(s5==s3);  
  
System.out.println(s1.equals(s3));  
System.out.println(s1.equals(s6));  
System.out.println(s6.equals(s1));
```



44

## Manipulation de chaînes

```
// Un peu vieux, utiliser plutôt split  
// Scanner ou java.util.regex  
StringTokenizer st =  
    new StringTokenizer(" Quelle boucherie !");  
  
while (st.hasMoreTokens())  
    System.out.println(st.nextToken());  
  
String[] result =  
"et ça découpe toujours".split("\\s");  
  
for (int i=0; i<result.length; i++)  
    System.out.println(result[i]);
```

45

## « Fonction » main

```
public static void main(String[] argv);
```

- Obligatoire en mode *standalone*
- Point d'entrée unique du programme



```
java Exemple param1 "param 2" param3
```

- argv :
  - tableau de chaînes de caractères
  - Paramètres de la ligne de commande

46



## Déclaration d'une classe

- Moule / Modèle / Fabrique à objets
- Caractéristiques / Attributs
- Messages / Méthodes
- Nom unique (package)
- Relation entre les objets

49

## Exemple

```
public class ExempleSimple {  
    ...  
}  
  
class Voiture {  
    ...  
}  
  
class Camion {  
    ...  
}
```

50

## Contenu de la classe

- Attributs
  - D'instance
  - De Classe
- Méthodes
  - D'instance
  - De classe
- Constructeur(s)
- Instructions au chargement de la classe

51

## Attributs

- Listés n'importe où (début de classe préférable)
- Valeurs par défaut ( ≠ variables locales)
- D'instance

```
String chaine;  
int    entier;
```

- De classe
  - Accessibles sans créer d'objet
  - Initialisés à la déclaration ou bloc spécial

```
static int compteur = 0;
```

52

## Méthodes

### • D'instance

```
String getNom()          { return "NOM"; }  
int   max(int a, int b) { return a<b?b:a; }  
void  traitement()       { }
```

### • De classe

- Accessibles sans créer d'objet
- Ne peut PAS accéder aux attributs d'instance

```
static int getCompteur() { return compteur; }
```

- Pas de valeur par défaut des paramètres [C++]

## Modificateurs de méthode

Méthode publique, visible/appelable par tout le monde

Méthode de classe

Méthode non redéfinissable

Type de retour

Paramètre(s)

```
public static final int methode(double d);
```

53

54



## Constructeur

- Initialiser les attributs d'un nouvel objet
- Syntaxe différente d'une méthode
  - Porte le même nom que la classe
  - Pas de type de retour
- Constructeur sans argument
  - fourni automatiquement si pas d'autres constructeurs
- Surcharge de constructeur
  - Appel de constructeurs ≠ avec paramètres

① PAS d'héritage de constructeur

① PAS de constructeur de copie (clone())

```
public class Cours {
    int nb_etudiants;
    boolean passionnant;
    public Cours() {
        this(0);
    }
    public Cours(int n) {
        setNbEtudiants(n);
    }
    public void setNbEtudiants(int n) {
        nb_etudiants = n;
    }
    public int getNbEtudiants() {
        return nb_etudiants;
    }
    public boolean isPassionnant()
    { return passionnant; }
}
```

55

## 1 - Créer les classes

- Pas de relation entre les classes
- Répartition ?
  - même fichier
  - fichiers différents (même package ?)
- Où mettre le *main()* ?
- Quelles classes compiler ?
- Quelle classe exécuter ?

Voiture	
-immat : Chaîne	
-couleur : entier	
-places : entier	
+ afficher()	
+ avancer()	

Camion	
-immat : Chaîne	
-capacité : réel	
+ afficher()	
+ avancer()	

58

```
public class Gestion1 {
    // classe pour le programme
    public static void main(String[] a) {
    }

    class Voiture {
        String immat;
        Voiture() {}
        void avancer() {}
    }

    class Camion {
        int capacite;
        Camion() {}
        void avancer() {}
    }
}
```

59

```
public class Gestion2 {
    // classe pour le programme
    public static void main(String[] a) {
        Voiture v;
        Camion c;
    }
}

public class Voiture {
    String immat;
    Voiture() {}
    void avancer() {}
}

public class Camion {
    int capacite;
    Camion() {}
    void avancer() {}
}
```

60

## Dans des répertoires différents...

- Visibilité au niveau package
  - 1 package ≡ 1 répertoire
- Si la classe n'est pas dans le répertoire courant

```
javac -cp chemin Classe.java
javac paquetage.Classe.java
java -cp chemin Classe
```

- Fichiers jar

```
java -jar fichier.jar
```

61

## C'est pas déjà fait ?

Voiture
-immat : Chaîne
-couleur : entier
-places : entier
+ afficher()
+ avancer()

Camion
-immat : Chaîne
-capacité : réel
+ afficher()
+ avancer()



Afficher sur la console :  

- Je suis une Voiture/Camion et l'immatriculation
- J'avance

62

## Objectif : Gérer un parc de véhicules



- Écrire une application permettant de gérer un parc de véhicules d'une société possédant
  - des voitures
  - des camions
- Créer des classes simples sans relation
- Instancier des classes
- Étudier l'héritage et le polymorphisme
- Étudier les autres relations



63

## Créer une instance

- Demander la mémoire à la JVM
  - Opérateur *new*
- Appeler le constructeur

```
Classe instance = new Classe(paramètres);
```

- Manipulation de pointeurs références ?
- Valeur spéciale **null**
  - si création impossible
  - ou pas encore affectée

```
(instance == null)
```



## Détruire une instance

- Pas de destruction manuelle
- Destruction automatique par la JVM
  - Ramasse-miettes (*Garbage Collector*)
  - Le développeur peut demander un nettoyage, enfin ...
- Plus de fuites de mémoire ?
  - Tables de hachage complexe
  - Boucle infinie
  - Aider la JVM en mettant à `null`
- Méthode `finalize()`
  - Ressemble au destructeur C++
  - Peut ne pas être appelée (si gc non exécuté)



64

```
// constructeur proposé par défaut
public Voiture() { immat = null; }
```

```
Voiture v = new Voiture();
```

```
public Voiture() {
  immat = "0000 AA 00"; }
```

```
public Voiture(String im) {
  immat = im; }
```

```
Voiture v = new Voiture("300 ISI 63");
```

```
public String getImmat() {...}
public void setImmat(String im) {...}
```



67

## Accéder aux membres

- Opérateur point à l'extérieur de la classe

```
instance.methode();
instance.attribut;
```

- Référence valide ?
  - Null Pointer exception
- Membre visible
  - Niveaux d'accès (public/privé/package)
  - Interface de classe
  - Encapsulation



66

## 2- Créer des objets

- Créer quelques voitures avec des attributs différents
- Créer quelques camions avec des attributs différents
- Appeler les méthodes des objets ainsi créés
- **Constructeurs ? Accesseurs ?**

```
double d = 1.0;
float f = 2.0f;
```

```
public class B {
  static void methode1(A a) {
    a = new A(2); }
```

```
  static void methode2(A c) {
    c.valeur = 3; }
```

```
  static A methode3(A b) {
    b = new A(4);
    return b; }
```

```
  public static void main(String[] param) {
    A a = new A(1);
    methode1(a);
    methode2(a);
    a = methode3(a); }
```

Afficher `a.valeur` et `a`. Que se passe-t'il ?



Digression

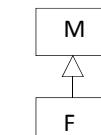
70

## Au chargement de la classe...

- Instructions spécifiques exécutées au **chargement** de la classe dans la JVM
  - Pas à l'instanciation d'objet
  - Plus général que l'initialisation des attributs statiques

```
static int[] tab = new int[100];
class Exemple {
  static int[] tab;
  static {
    // exécuté au chargement de la classe
    tab = new int[20];
    for(int i=0; i<20; ++i) tab[i] = 2*i;
  }
}
```

71



## Héritage (1)

- F hérite de M ?
- Conditions
  - M doit être visible (publique même package)
  - M n'est pas finale
- F hérite de **tous** les membres **protégés** et **publics** de M **sauf** les constructeurs
  - Les membres **privés** ne sont jamais transmis
- F n'hérite que d'une **SEULE** classe directe
- Toute classe hérite de `java.lang.Object`

72

## Héritage (2)

```
public class F extends M {
    public F() {
        super(); // appel du constructeur de M
        // initialisations spécifiques
    }
}
```

- `super` : ce qui vient de la classe mère

```
super(champ1, champ2); // appel de constructeur
super.methode();
super.attribut;
super.super.attribut // illégal
```

- `this` : concerne l'objet courant

73

## Référence `this`

```
public class C {
    String chaine1, chaine2;
    public C() {
        chaine1 = "CHAINE1";
        chaine2 = "CHAINE2" ;
    }
    void methode1(String chaine1, String c) {
        this.chaine1 = chaine1;
        chaine2 = c;
    }
    void methode2() {
        methode1("e","f");
        this.methode1("", "");
    }
}
```

74



## Noms qualifiés

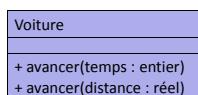
```
class M {
    protected int a;
}
public class F extends M {
    protected double a;
    public void toto() {
        a
        this.a
        super.a
        ((M)this).a
        ((F)this).a
    }
}
```

75

## Polymorphisme

### • Forme faible

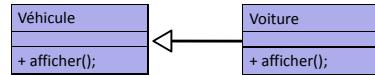
- Surcharge de méthode – *overloading*
- **Statique** (compilation)
- Méthodes de signatures différentes



76

### • Forme forte

- Redéfinition – *overriding*
- "Surcharge" **dynamique** (abus de langage)
- Actions différentes pour des classes d'une même hiérarchie

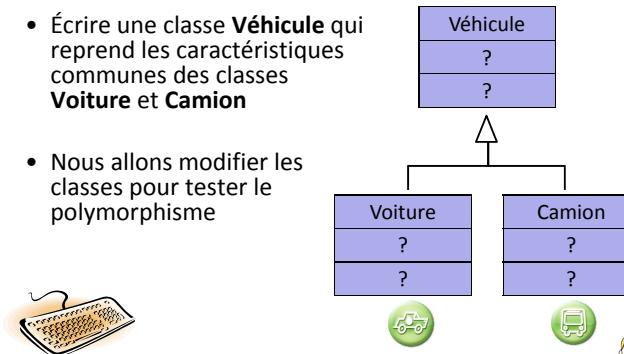


77

## 3- Hériter ...

- Écrire une classe **Véhicule** qui reprend les caractéristiques communes des classes **Voiture** et **Camion**

- Nous allons modifier les classes pour tester le polymorphisme



77

```
public class Vehicule {
    public void afficher() {
        System.out.println("Vehicule");
    }
    public static void main(String[] param) {
        Vehicule v = new Vehicule();
        Voiture w = new Voiture();
        Camion c = new Camion();
        Vehicule z = new Voiture();
        Voiture i = new Vehicule();
    }
}
```

Appeler les méthodes afficher() des objets

```
class Voiture extends Vehicule {
    public void afficher() {
        System.out.println("Voiture");
    }
}
class Camion extends Vehicule {
```



79

```
public class Vehicule {
    public void afficher() {
        System.out.println("Vehicule");
    }
    public static void main(String[] param) {
        Vehicule v = new Voiture();
        v.afficher();
        v.special();
        ((Voiture)v).special();
        ((Camion)v).afficher(); // défini précédemment
    }
}
```

Que se passe-t'il ?

```
class Voiture extends Vehicule {
    public void afficher() {
        System.out.println("Voiture");
    }
    public void special() {
        System.out.println("special");
    }
}
```

78

80

## Méthodes virtuelles ou finales ?

### • Méthodes virtuelles

- Par défaut
- Construction d'une table de méthodes pour une hiérarchie
- Recherche dans cette table (lenteur ?)

### • Méthodes finales

- Non redéfinissables dans les classes filles
- Plus rapides que les méthodes virtuelles
- Conseil : accesseurs en final

- Le dernier mot : la JVM *hotspot* ...

81

## java.lang.Object

- clone()
  - finalize()
  - toString()
  - getClass().getName()
  - equals()
  - hashCode()
- doivent être cohérentes
- Deux objets égaux ont le même hashcode
  - Ne doit pas changer pour une exécution
  - Deux objets distincts peuvent avoir le même



## Clonage

- Copier un objet pour ne pas le modifier
  - Pas de constructeur de copie
- Implémenter Cloneable
  - Sert seulement à prévenir le compilateur
- Appeler la méthode `clone()` de la classe mère en public
- S'assurer que la méthode `clone()` d'Object est également appelée en haut de l'échelle
- Traiter les exceptions dans `clone()`

82

83

84

```
class Trooper implements Cloneable {
    public Object clone() {
        Trooper object = null;
        try {
            object = (Trooper) super.clone();
        } catch(CloneNotSupportedException cnse) {
            cnse.printStackTrace(System.err);
        }
        // s'occuper des attributs "compliqués"
        // pour éviter la copie de surface
        // (shallow copy) si object != null
        return object;
    }
}
```



Copie des types primitifs  
Copie des références  
Objets non mutables (String)

## Niveaux d'accès

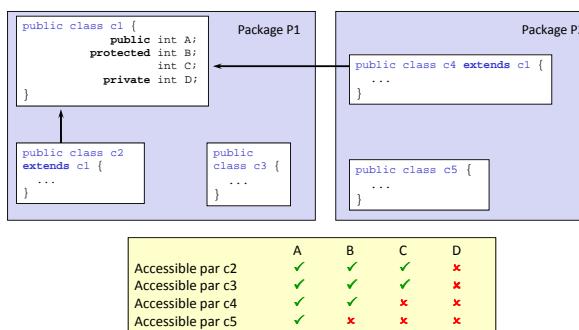
- Privé | private** : même classe
- Protégé | protected** : même package ou sous-classe d'un package différent
  - Moins restrictif que le C++ !
  - Different en UML également
- Package | - (par défaut)** : package
  - Sorte de friend du C++
  - DANGEREUX
- Public** : tout le monde



85

86

87



Tiré de « Éléments de programmation JAVA », Olivier Dedieu, INRIA

```
public abstract class Vehicule1 {
    public void afficher() {
        System.out.println("Vehicule");
    }
}
```

```
public abstract class Vehicule2 {
    abstract public void afficher();
}
```

```
class Voiture2a extends Vehicule2 {
}
```

```
abstract class Voiture2a extends Vehicule2 {
}
```

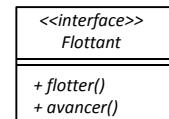
```
class Voiture2b extends Vehicule2 {
    public void afficher() {}
}
```

88

## Interface (1)



- "Classe virtuelle pure"
  - Pas de code
  - Pas de variable/attribut (UML)
  - "Constantes" autorisées (public| package `static final`)
- Toutes les méthodes sont `abstract` par défaut
  - [Vocabulaire] **IMPLEMENTER** une interface

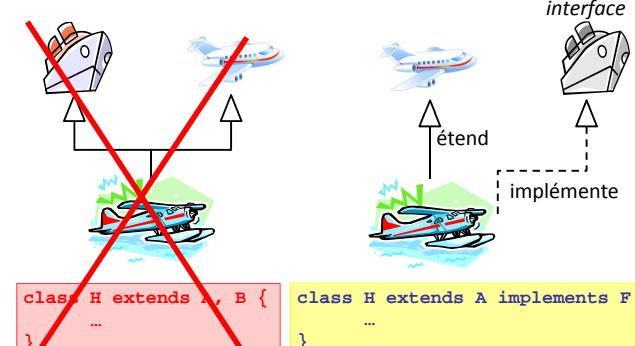


```
interface Flottant {
    public static final int CONSTANTE = 30;
    public void flotter();
    public void avancer();
}
```

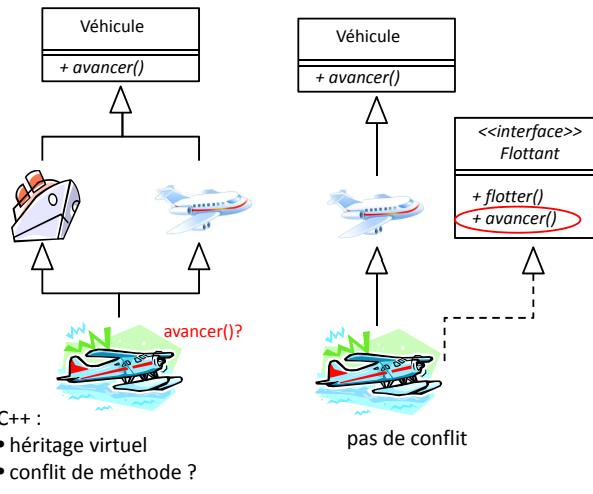
89

## Héritage multiple ?

Relation non symétrique  
= raison fonctionnelle



90



## Interface (2)

- Réponse à l'héritage multiple

```
public class B extends A implements IC, ID
    // ...
}
```

- Instancier une classe

Implémenter **TOUTES** les méthodes  
des interfaces qu'elle utilise

- Polymorphisme

la classe est du **type** de l'interface  
qu'elle implémente

- héritage virtuel
- conflit de méthode ?

pas de conflit

91

## Relations entre objets

- Relation
  - Agrégation
  - Composition
  - Référence ou tableau de références
  - Utiliser un conteneur spécifique
    - Collections
    - Ex : `java.util.ArrayList`

A cartoon illustration of a blue mouse with large, expressive black eyes and a small pink ear. It has a simple, rounded body and a short, thin tail.

## Tableau

- De scalaires
    - `int, double, char, ...`
    - Une case = un scalaire utilisable directement
  - D'objets
    - Un tableau de références sur des objets de la classe
    - Références initialisées à `null`
    - Pas de création d'objets par défaut comme en C++
    - Initialiser chaque élément du tableau pour l'utiliser

```
public class Zoo {  
    static final int NB_ANI = 50;  
    Animal[] animaux;  
    public Zoo() {  
        // pas de création d'objet, sinon le constructeur  
        // par défaut serait obligatoire  
        animaux = new Animal[NB_ANI];  
    }  
    public void placerAnimal(int i, Animal a) {  
        // if ((i>=0) && (i<NB_ANI))  
        animaux[i] = a;  
    }  
    public static void main(String[] chaines) {  
        Zoo zoo = new Zoo();  
        zoo.placerAnimal(0, new Animal("lion"));  
    }  
    class Animal {  
        String nom;  
        // public Animal() { nom="INCONNU"; }  
        public Animal(String nom) {  
            this.nom = nom;  
        }  
    }  
}
```



```
// ajouter dans la classe Zoo
public Animal quelAnimal(int i) {
    return animaux[i];
}

// ajouter dans la class Animal
public void afficher() {
    System.out.println(nom);
}

// ajouter dans la methode main()
zoo.quelAnimal(0).afficher(); // c'est bon
// c'est la meme chose que d'écrire
// zoo.animaux[0].nom
// si animaux et nom sont publics
zoo.quelAnimal(1).afficher(); // NullPointerException
zoo.quelAnimal(60).afficher(); // ArrayOutOfBoundsException
```

96

## Résumé

```
[Modificateur]* class identifiant
  [extends      classe_de_base ]
  [implements  interface {, interface}*  ]
}
```

- Héritage simple **seulement**
  - Implémentation multiple d'interfaces
  - *Toutes* les méthodes sont virtuelles
  - Une classe **finale** n'est pas dérivable
  - Tous les classes dérivent de `java.lang.Object`

97

```
public class Vehicule {  
    public void afficher() {  
        System.out.println("Vehicule");  
    }  
    public static void main(String[] param) {  
        Vehicule v = new Voiture();  
        ((Voiture)v).embrayer();  
        ((Manuel)v).embrayer();  
        System.out.println(Manuel.VITESSES);  
    }  
}  
interface Manuel {  
    public static final int VITESSES = 5;  
    public void embrayer();  
}  
class Voiture extends Vehicule implements Manuel {  
    public void afficher() {  
        System.out.println("Voiture");  
    }  
    public void embrayer() {  
        System.out.println("Boite manuelle");  
    }  
}
```

93

ISIMA



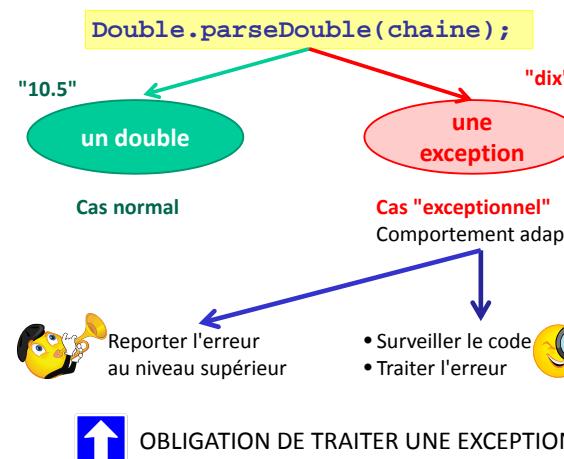
## 5. Exceptions

1



## Exceptions

- Manière élégante et efficace de gérer les erreurs potentielles d'exécution
- Fonctionnement similaire au C++
- Une erreur potentielle ≡ une exception
- Hiérarchie des exceptions
- Une erreur = une instance d'exception
- Partie intégrante de la signature d'une méthode
- Obligation de lever les exceptions



### Diviser par 0

- Faire une division par zéro avec deux variables entières **PUIS** afficher le résultat de 22-11
- Que se passe-t'il ?
- Mettre en place le mécanisme d'exception pour attraper l'exception arithmétique.

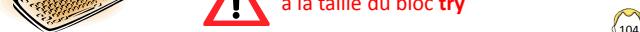


int n = 10;  
int d = 0;  
System.out.println(n/d);

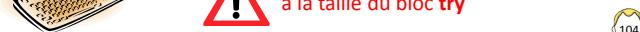
à la taille du bloc try



101



104



105



### Ordre des blocs catch

```
try {
    // code à tester
} catch (Exception e) {
    e.printStackTrace();
} catch (IOException e) {
    // traitement adapté
}
```

```
try {
    // code à tester
} catch (IOException e) {
    // traitement adapté
} catch (Exception e) {
    e.printStackTrace();
}
```

### Nouvelle syntaxe

7

```
try {
    // bloc à surveiller
} catch (NumberFormatException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Même traitement

```
try {
    // bloc à surveiller
} catch (NumberFormatException | IOException e) {
    e.printStackTrace();
}
```

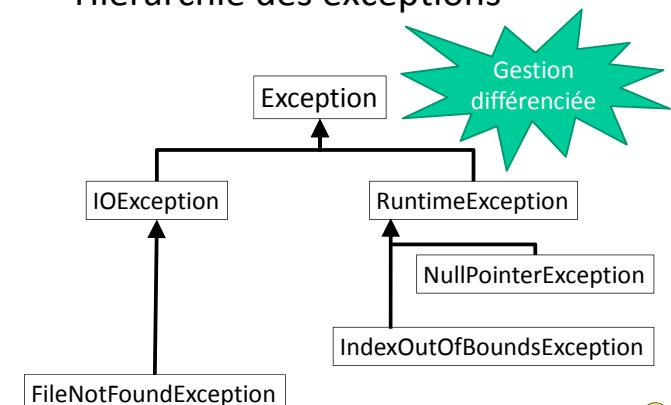
107

## Attraper une exception

```
public void methode() {
    try {
        // bloc à surveiller
        res = Double.parseDouble(chaine);
        total += res;
    } catch (NumberFormatException e) {
        System.out.println(e.getMessage());
        // ou e.printStackTrace();
    } finally {
        // Clause TOUJOURS exécutée
    }
}
```

102

## Hiérarchie des exceptions



105

## Bloc finally

- Optionnel
- TOUJOURS exécuté
  - Même si aucune exception n'a été levée
  - Même si une instruction *continue*, *break* ou *return* se trouve dans le bloc *try*
- Utilité avec un langage doté d'un ramasse-miettes ????
  - Rendre les ressources
  - Fermer des fichiers, par exemple
- final CATCH ...

7

108

## Exception personnalisée

- Exception dérive de Throwable
  - Error
  - Exception
- Dériver d'Exception
  - Surcharger Constructeur (String message)
  - OU Redéfinir getMessage()
- Lancer une exception

```
throw new MonException();
```

```
class AutorisationException extends Exception {  
    public String getMessage() {  
        return "Op impossible : découvert trop grand";  
    }  
  
    public class CompteBancaire {  
        double solde = 0.0;  
        double decouvert = -700.; // ... autorise  
  
        public void retrait(double montant)  
            throws ArithmeticException, AutorisationException {  
            double nouveau = solde - montant;  
            if (montant<0.0)  
                throw new ArithmeticException("Mauvais montant");  
            if (nouveau<decouvert)  
                throw new AutorisationException();  
            solde = nouveau;  
        }  
    }  
}
```

109

## Conclusion exceptionnelle



Pour toute exception déclenchée, le compilateur **impose** un traitement

1. Bloc try/catch qui gère cette exception
2. Passage de l'exception au niveau supérieur (appelant).
  - L'exception apparaît alors dans la **signature** de la méthode

110

111

```
// exemple d'utilisation convertir("90");  
public double convertir(String n)  
    throws NumberFormatException  
{  
    double res = Math.PI /180;  
    try {  
        res *= Double.parseDouble(n);  
    } catch(NumberFormatException e) {  
        res = .0;  
    }  
    return res;  
}
```

112

# ISIMA



6. Bibliothèques graphiques  
AWT & Swing

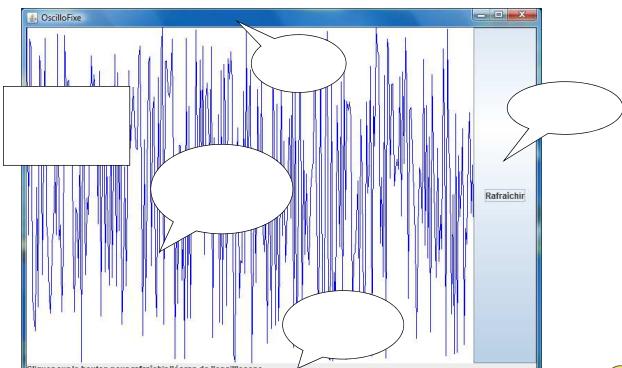
113

## AWT vs Swing

- Abstract Window Toolkit (java.awt.\*)
  - "Figé" depuis 1.1
  - Présent dans les navigateurs sans plug-in
  - Gestion des événements obsolète
  - Composants lourds
- Swing (javax.swing.\*)
  - Toujours en évolution
  - Plus complexe qu'AWT
  - Composants légers
  - Surcouche d'AWT (Jcomposant)

114

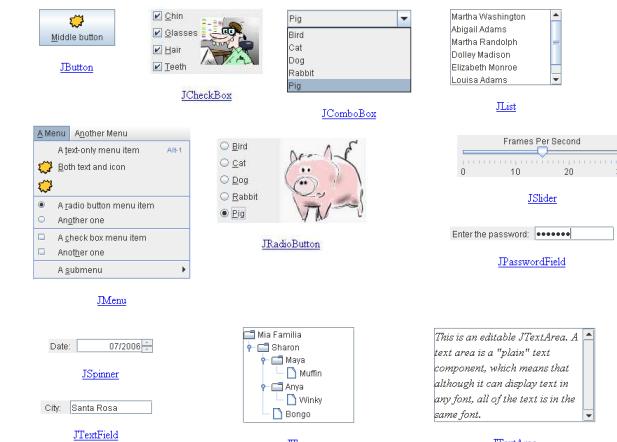
## Exemple d'application Swing



115

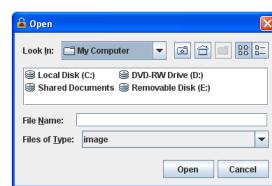
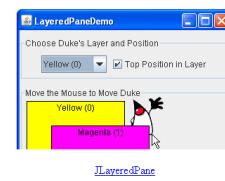
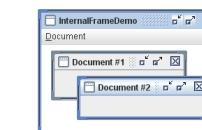
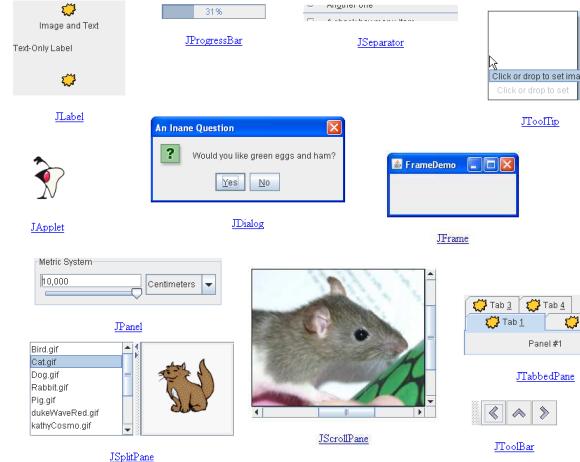
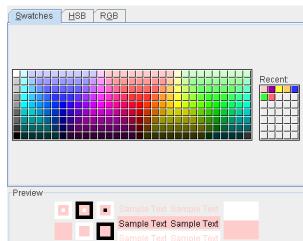
## Swing

- Composants de haut-niveau (lourds)
  - JFrame, JDialog, JApplet
- Conteneurs
  - JPanel, JScrollPane, JToolBar
- Composants basiques
  - JMenu, JButton, JLabel
- Dessiner
  - Canvas (AWT), JPanel (Swing)
- Index graphique des composants Swing  
<http://download.oracle.com/javase/tutorial/ui/features/components.html>
- Apparence / système (Look & Feel) **NON ABORDÉ**



116

117



118

119

Exemple du tutoriel officiel Sun/Oracle sur Swing :

```
import javax.swing.*;
public class HelloWorldSwing {
    private static void createAndShowGUI() {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame("HelloWorldSwing");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JLabel label = new JLabel("Hello World");
        frame.getContentPane().add(label);

        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

Cas 1

121

```
invokeLater(new Runnable() {
    public void run() {
        // faire qq chose
    }
});
```



- Classe **imbriquée** dite **anonyme**
- Spécialise la classe donnée ou **implémente** l'interface donnée
- Déclarée à la volée / utilisée qu'une seule fois
- Compilée avec un nom arbitraire nomclasseenglobante\$nombre.class

122

## Manipuler une fenêtre ? (1)

- Instancier un objet **JFrame**
- 1. Déclaré à la volée dans un méthode
  - Ne peut être réutilisé ailleurs
- 2. Membre de classe (agrégation)
- 3. Spécialisation de classe

123

```
public class MonFrame extends JFrame{
    public MonFrame() {
        super("Mon application");
    }

    static public void main(String[] args) {
        MonFrame mf = new MonFrame();
        mf.afficher();
    }
}

public class MonApplication {
    JFrame frame;
    public MonApplication() {
        frame = new JFrame("Mon Application");
    }

    static public void main(String[] args) {
        MonApplication ma = new MonApplication();
        ma.afficher();
    }
}
```

Cas 3

Cas 2

124

## Manipuler une fenêtre ? (2)

Cas le plus général et recommandé !  
L'application gère une fenêtre spécialisée !

125

## Boîte de dialogue simple

- Classe **JOptionPane**
- Méthodes statiques **showXXXDialog**
  - Message, Confirm, Option, Input



```
JOptionPane.showConfirmDialog(frame,
    "Alors, ça vous plaît ?");
```

126

## Ouvrir une fenêtre

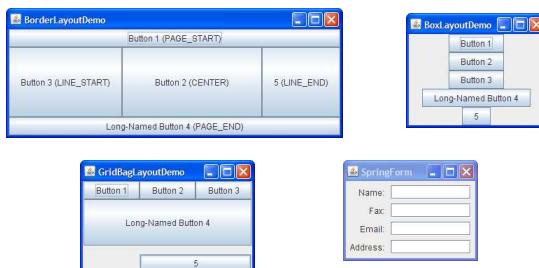
- Afficher une application graphique simple agrémentant un *JFrame*
- Ajouter une boîte de dialogue **modale**



127



## Layout Manager (1)



<http://download.oracle.com/javase/tutorial/uiswing/layout/visual.html>

128



## Layout Manager (2)

- Flow
- Box
- Border
- Grid
- Card
- GridBag
- Spring
- Aucun

Managers par défaut :

- BorderLayout pour le contentPane du JFrame
- FlowLayout pour un JPanel

Package : `java.awt`

```
frame.getContentPane().setLayout(new FlowLayout());  
frame.getContentPane().add(objet [, paramètres]);
```

129



## Positionnement

- Placer des composants
  - Taille (fixe, minimale, maximale et préférée)
  - Position relative (portabilité)
  - Position absolue (extension JBuilder)
- Politique de placement d'un conteneur : Layout
  - JFrame, JDialog, JPanel, ...
- Complexé à manipuler
  - Conteneurs et Layouts spécialisés en cascade
  - GUI Builder
    - Matisse / Netbeans avec GroupLayout
    - WindowBuilder / Eclipse
    - IntelliJ



128

129



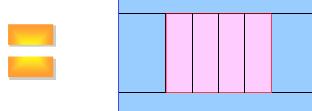
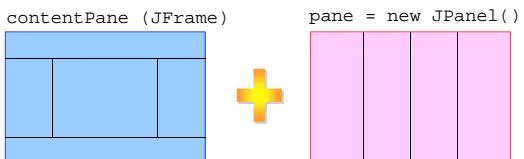
## On teste ...

- Tester le BorderLayout ...
  - avec un bouton dans chaque zone
  - Nord, Sud, Est, Ouest, Centre
- Tester le FlowLayout
  - ... en ajoutant des boutons

```
frame.getContentPane().setLayout(new FlowLayout());  
for (int i=0; i < 5; ++i)  
    frame.getContentPane().add(new JButton(new String(i)));
```



130

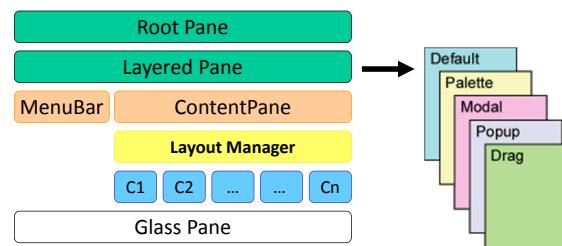


(Layouts par défaut des composants)

131



## Organisation par couches



132

- Communication entre les composants d'une application et avec l'utilisateur
  - Envoi de messages
- Gestion des événements incompatible entre AWT et Swing
- Réaction au moindre événement
  - Action
  - Souris ? (déplacement, clic, action)
  - Clavier ? (appui, focus)
  - Fenêtre ? (déplacement, visibilité)



## Événements

133

134

135

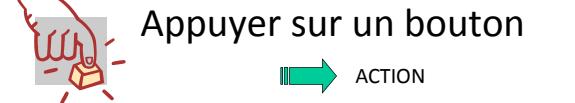
## Gestionnaire d'événements

- Surveiller (écouter) les messages/événements
- Délégation des actions à réaliser
- Dépendant des objets manipulés

- Écrire le gestionnaire
  - Implémentation de Listener (interface) **OU**
  - Spécialisation d'Adapter (classe)

- Enregistrer le gestionnaire

136



### Appuyer sur un bouton

**ACTION**

```
// Ecrire le gestionnaire
class Gestionnaire implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    // ce qu'il y a à faire
  }
}
```

**OPTION 1**

```
class Gestionnaire extends AbstractAction {
  public void actionPerformed(ActionEvent e) {
    // ce qu'il y a à faire
  }
}
```

**OPTION 2**

```
// enregistrement du gestionnaire
JButton bouton = new JButton("Libellé");
bouton.addActionListener(new Gestionnaire());
```

137



## Essayer

- Choisir un des deux gestionnaires
- Afficher une boîte de dialogue de type Message à l'appui d'un bouton



138

## Event / Listener / Adapter

- Event **MouseEvent**
- Listener
  - Spécifique à l'événement
  - Liste des méthodes / messages possibles `composant.addEventListener()` **MouseListener**
- Adapter
  - Patron de conception / *design pattern*
  - Implémentation "vide" de convenance **MouseAdapter**
- Toujours possible ?**

**ActionAdapter n'existe pas !**

**AbstractAction existe**

139

### A bas les boutons ... (1)



- Afficher quelques boutons
- incrémenter la valeur d'un bouton à chaque fois que l'on clique dessus
- Ajouter une boîte de confirmation pour la sortie du programme

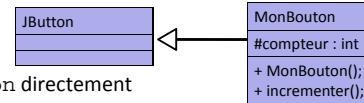


140

### A bas les boutons ... (2)

- Bouton ?

- Spécialiser JButton pour compter



- Utiliser JButton directement

```
JButton : setText()
          setLabel()
          Integer : parseInt()
```

- Gestionnaire ?

- Un pour tous les boutons
- Un par bouton

**event.getSource()**

141

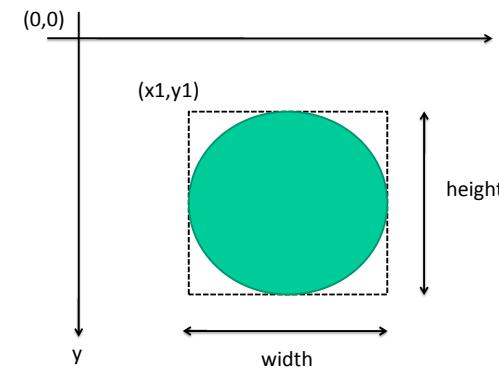
## Contexte graphique

- Boîte à dessin
- Associé à une fenêtre ou une image
- Classe `java.awt.Graphics`
  - Couleur
  - Mode de dessin (XOR ...)
  - Police de caractères
  - Actions basiques ( `drawXXX`, `fillXXX` )
- Classe `java.awt.Graphics2D`
  - Plus de fonctionnalités en dessin
  - Changement de repère, transformations



142

### Orientation par défaut



`graphics.fillOval(x1, y1, width, height);`

143



## Dessiner

- Canvas [AWT]

- Composant "lourd"
- `void paint(Graphics g)`

- JPanel [Swing]

- Composant léger
- Opacité

- `void paintComponent(Graphics g)`
- `paint()` existe mais ...



144

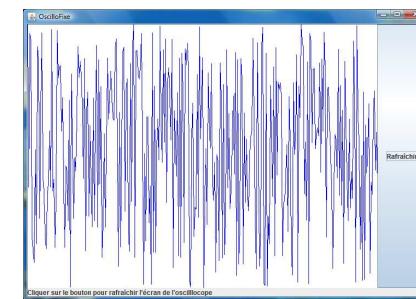
- Contexte graphique "prêté" par le système
- Méthode `repaint()` pour mettre à jour



## JCanvas

- Étendre la classe JPanel
  - Opaque ? double buffering ?
  - Layout inutile !
- Fixer la taille du composant
  - PreferredSize (utilisée par pack())
  - MinimumSize, MaximumSize
- Redéfinir paintComponent()
  - À ne jamais appeler directement
  - Appeler super.paintComponent(g) si besoin

145

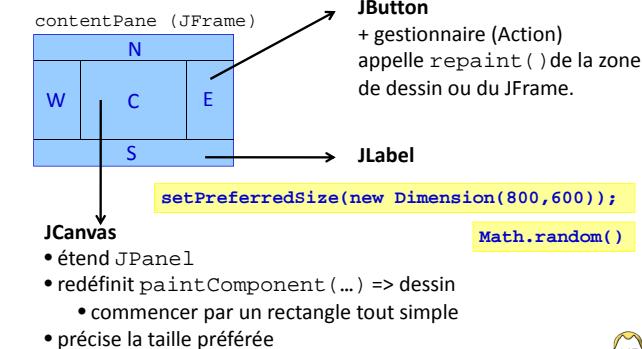


1. Simuler un oscillateur "fixe"
2. Gérer l'appui du bouton : générer un nouvel affichage (aléatoire ou sinusoïdal)

146

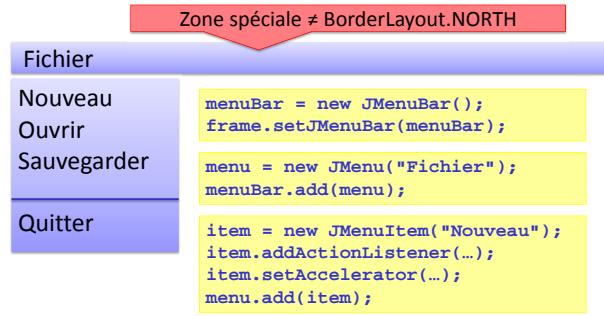


## Solution ?



147

## Barre de menus



148



## NetBeans vs Eclipse

- Sun / Oracle
- Supporte toujours les dernières technos Java
- Conception de GUI native (matisse)
- IBM
- Réputation plus pro ?
- Plugin : délai ? Incompatibilité ?
- Plugin : VEP

Choix arbitraire mais obligatoire pour les TPs et mini-projets

149

## ISIMA



Java

7. Concurrence (Threads)

150



## Threads (1)

- Processus "léger"
  - Exécution simultanée / concurrence
  - Partage de données
- 3 méthodes
  - Implémenter run() de Runnable
  - Déléguer à un Executor (java.util.concurrent)
  - Redéfinir la méthode run() de Thread

5

151

## Threads (2)

```

// Exemple avec implémentation d'interface
// Modélisation : créer une nouvelle tâche
class Tache implements Runnable {
    // implémenter run()
}
  
```

CONSEILLÉ



```

// AILLEURS :
Tache tache = new Tache();
(new Thread(tache)).start();

// Exemple avec spécialisation de classe
// Modélisation : créer un nouveau type de Thread
class Special extends Thread {
}

Special special = new Special();
special.start();
  
```

152

## Threads (3)

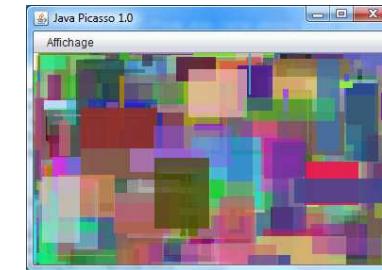
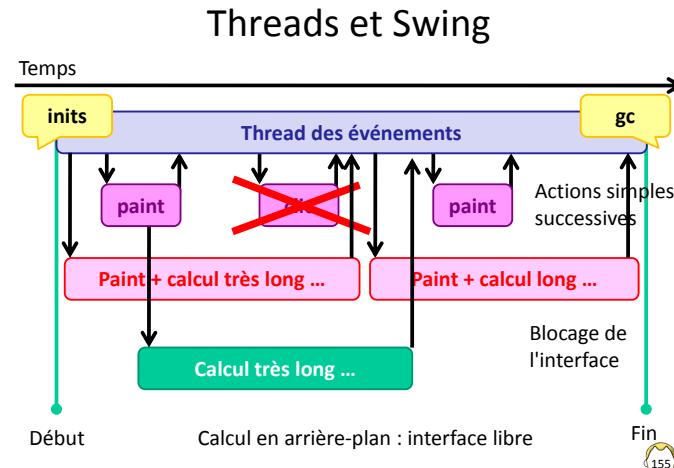
- Sécurité d'accès aux méthodes et données
  - Méthode/statement qualifié **synchronized**
  - Variable qualifiée **volatile**
  - Variable qualifiée **final (Immutable)**
- Méthode **start()** pour démarrer
- Méthodes **interrupt()** ou **sleep()**
- L'arrêt doit être NATUREL
  - la méthode **run()** doit se terminer normalement
  - Ex : test d'une variable d'arrêt
  - **stop()** est obsolète



153

## Threads et Swing (1)

- Deux fils d'exécution
  - Fil du programme
  - Fil des événements
- Fil des événements **monothread**
  - Patron MVC : concurrence difficile
  - Interface peut être figée facilement
  - Swing Worker (antérieur au patron Executor)



Coder avec un EDI !

Dessiner avec la composante alpha  
=> bloquer l'interface

- "Rafraîchir" par menu
- "Quitter?" par le menu et la croix
- Dessiner dans une image (BufferedImage)
- Dessiner par thread (Runnable ou SwingWorker)
  - Rafraîchir ne doit plus faire un nouveau dessin
  - Menu pour faire une nouvelle toile ;-)
- Gérer le redimensionnement

156

## Éléments de réponse (1)

- Dessiner directement dans paintComponent()
- Dessiner dans une image
  - Créer une image
  - Dessiner dans l'image
  - Afficher l'image
- Mise en place du thread
  - Exporter le dessin de l'image dans le run() d'une tâche
  - Dans paintComponent() : Image à dessiner ?
    - Non : lancer le calcul (lancement d'un thread)
    - Oui : afficher l'image

154

## Éléments de réponse (2)

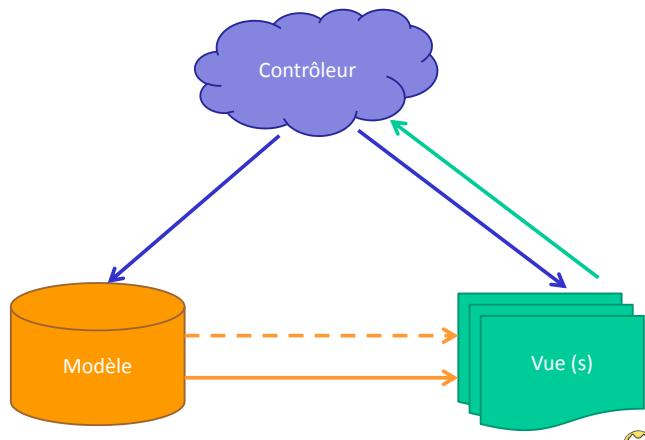
- Une seule méthode quitter() appelée
  - JOptionPane.show...
  - System.exit()
  - Par l'item du menu - ActionListener
  - Par le clic sur la souris - WindowListener
    - JFrame.DO NOTHING ON CLOSE
- BufferedImage
  - TYPE\_INT\_ARGB
  - graphics.drawImage(img, 0, 0, null);

158

## Modèle MVC

- Modèle / Vue / Contrôleur
- Logique applicative vs présentation
- Design(s) pattern(s) / Patron de conception
  - Communément adopté
  - Pas trivial du tout
  - A éviter sur les trucs simples
  - Adopté par Swing (Table <-> TableModel)

159



160

## Gestion des entrées/sorties (1)

- Flux E/S de données binaires
- Flux E/S de caractères
- Flux E/S d'objets
- Communication avec des fichiers
- Communication avec des ressources Internet
- Sérialisation (=> réseau)

162

## Gestion des entrées/sorties (2)

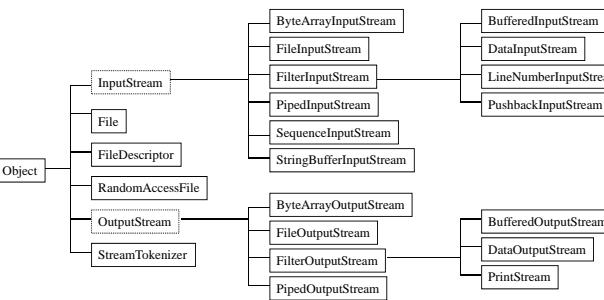
- java.io
  - Flux de données (fichier, pipe/thread, ...)
  - Sérialisation
  - Système de fichiers
- java.nio (java 4)
  - Mémoire tampon Buffer
  - Canaux + Sélecteurs : hautes performances
  - Traduction des jeux de caractères
- java.nio2
  - Simplification
  - Path



7

163

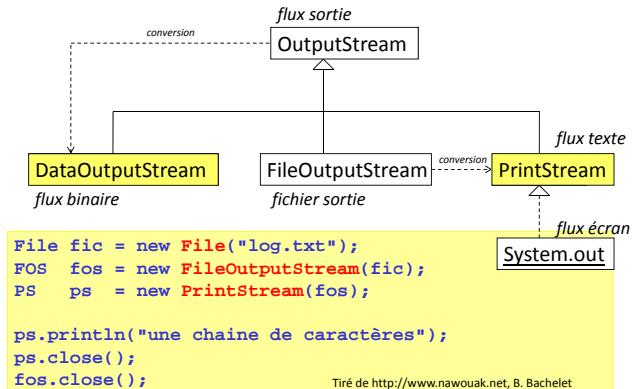
## java.io.\*



Tiré de « Elements de programmation JAVA », Olivier Dedieu, INRIA

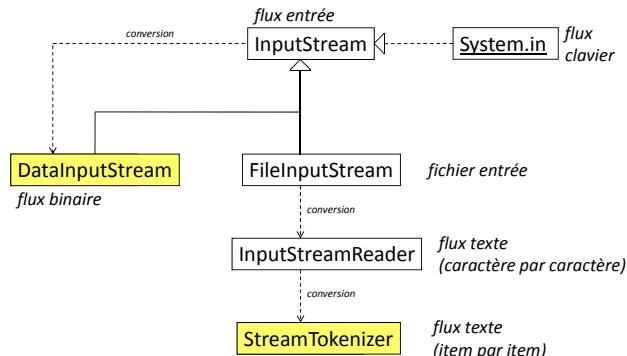
164

## Flux de sortie



165

## Flux d'entrée



166

## Saisie clavier

// Version 1 standard  
 // Attention aux exceptions soulevées  
 InputStreamReader isr =  
 new InputStreamReader(System.in);  
 BufferedReader br = new BufferedReader(isr);  
 br.readLine();

// Version 1.6  
 // les exceptions sont gérées par la Console  
 System.console().readLine();

6

167

## Sérialisation (2)

Méthodes à redéfinir pour un comportement particulier

```

void writeObject(ObjectOutputStream out)
    throws IOException;
void readObject(ObjectInputStream in)
    throws IOException, ClassNotFoundException;
void readObjectNoData()
    throws ObjectStreamException;
  
```

Flux (streams) à utiliser ...

FileOutputStream/FileInputStream  
 ObjectOutputStream/ObjectInputStream

168

FileOutputStream fos = null;
ObjectOutputStream oos = null;
try {
 fos = new FileOutputStream("fichier.dat");
 oos = new ObjectOutputStream(fos);
 oos.writeObject(objects);
 oos.flush();
} catch (IOException e) {
 e.printStackTrace();
} finally {
 if (FileInputStream fis = null;
 if (ObjectInputStream ois = null;
 try {
 fis = new FileInputStream("fichier.dat");
 ois = new ObjectInputStream(fis);
 objects = (Composite) ois.readObject();
 } catch (Exception e) {
 e.printStackTrace();
 } finally {
 if (ois!=null) ois.close();
 if (fis!=null) fis.close();
 }
}

**Écrire**

Classe de l'objet

**Lire**

169

170

## Sérialisation (1)

- Transformer un **objet** présent en mémoire en bits
  - Sur un disque (Stockage - Persistance)
  - Sur le réseau (Communication - RMI)
  - Les infos de classe ne sont pas transmises
- Implémenter l'interface **Serializable**
  - Ne fait rien, prévient le compilateur
- Proposer une version de sérialisation
  - static final long serialVersionUID = 110L;
- Attention à la protection des données
  - Données **transient** : données non copiées
  - Ou implémenter **Externalizable**

171

## Sérialisation en XML

- XMLEncoder pour les objets respectant les conventions NetBeans
- XStream pour les autres ;)
  - Une bibliothèque tiers

172

```

FileOutputStream fos = null;
XStream xstream = null;
try {
    fos = new FileOutputStream(name);
    xstream = new XStream();
    xstream.toXML(objects, fos);
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (fos!=null) fos.close();
}

```

Écrire

```

FileInputStream fis = null;
XStream xstream = null;
try {
    fis = new FileInputStream(name);
    xstream = new XStream();
    objects = (Composite)xstream.fromXML(fis));
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (fis!=null) fis.close();
}

```

Lire

172

## Compléments JAVA

Ça aussi  
c'est  
important



Éditions

ISIMA

175

### Terminologie : classe imbriquée

- Classe imbriquée / *nested class*
  - Classe définie à l'intérieur d'une autre classe
- 4 types de *nested class*
  - Classe membre statique
  - Classe membre non statique, *inner class*
    - Pas de membres statiques
    - N'existe qu'avec une instance de la classe
  - Classe locale (définie dans une méthode)
  - Classe anonyme (locale sans nom)

178

### TP "Agenda Simple"

Nom	Prénom	Téléphone	Courriel
Yon	Loïc	0473405042	loic AT isima DOT fr
Mahey	Philippe	0473405000	mahey AT isima DO...
Gouinaud	Christophe	0473405041	gouinaud AT isima ...
Closset	Martine	0473405000	martine POINT closs...
Peyraud	Corinne	0473405035	peyraud AT isima D...
Toledo	Françoise	0473405000	toledo AT isima DOT fr



- Manipulation de classes et d'interfaces
- Fichiers textes
- Javadoc
- JTable de Swing

173



```

// tableau statique
static int MAX = 50;
Personne[] p1 = new Personne[MAX];
p1[0] = new Personne();

```

```

// tableau dynamique (Legacy ou thread-safe)
Vector<Personne> p2 = new Vector<Personne>();
p2.addElement(new Personne());

```

```

// Liste : ArrayList, LinkedList, ...
ArrayList<Personne> p3 =
    new ArrayList<Personne>();
p3.add(new Personne());

```

```

new StringBuffer("nom"+";"+prenom);

```

174

### Plan



- Plus d'exemples avec JAVA
  - Enumération, nouveautés 5 et 6
- Généricité & collections
- Applet
- Outils
- Sur un complément : JNI, JDBC

176

ISIMA



Langage

177

### Conventions (1)

- Documentation officielle
- Tutoriaux SUN/Oracle
- Respect à l'écriture, facilité de lecture
- Production rapide
- Intégrée dans les EDI classiques
  - Formatage automatique dans Eclipse (CTRL+i)

179

### Conventions (2)

- Nom de classe ou interface
  - Première lettre majuscule
  - Reste en minuscules
  - Majuscules aux mots composés
- Attribut écrit en minuscule
  - Pas de tiret
- Méthode
  - Verbe pour action
  - Premier mot en minuscule
  - Majuscules à la première lettre des mots suivants

class  
CoursGenial

int attribut;

void ronfler();

180

## Conventions (3)

- Accesseur / Accessor
  - get + nom de l'attribut
  - is pour un booléen
- Mutateur/Mutator
  - set + nom de l'attribut
- "Constante"
  - Tout en majuscules
- Package
  - Tout en minuscules

`getAttribut()`  
`isAttribut()`

`setAttribut()`

`CONSTANTE`

`fr.isima.paquetage`

181

## Énumération ? (<1.5)

- Pas de vérification de type
- Affichage de la valeur sans intérêt

```
public static final int LUNDI      = 0;
public static final int MARDI      = 1;
public static final int MERCREDI   = 2;
public static final int JEUDI      = 3;
```

182

## Énumération

```
enum Semaine { LUNDI, MARDI, MERCREDI,
  JEUDI, VENDREDI, SAMEDI, DIMANCHE}
```

```
for (Semaine jour : Semaine.values())
  System.out.println(jour);
```

```
for (Semaine j : EnumSet.range(
  Semaine.LUNDI, Semaine.VENDREDI))
  System.out.println(j);
```

Dangereux mais utile : le static import

<http://download.oracle.com/javase/1.5.0/docs/guide/language/enums.html>

183

## Enum améliorés

```
enum Nom { VAL1(1), VAL2 ( 2);
  private int valeur;
  Nom(int i) { this.valeur = i };
}
```

```
enum Nom {
  VAL1 { retour methode(params) {...}},
  VAL2 { retour methode(params) {...}};

  abstract retour methode(params);
}
```

5

184

## Annotation

- "Extension" de celles du javadoc trop restrictives
- Méta données
  - Développeur
  - Compilateur
  - Machine virtuelle
  - Génération code, documentation, configuration
- Plus simple et plus léger que le XML
- Utilisation intensive
  - JUnit 4+
  - Java EE : JPA, ...

5

185

## Annotations standards

- `@Override`
- `@SuppressWarnings`
  - `@SuppressWarnings(value="deprecation")`
  - `@SuppressWarnings("deprecation")`
  - `@SuppressWarnings({"unchecked", "deprecation"})`
- `@Deprecated`
- `@Documented`
  - Documentation
- `@Retention(RetentionPolicy.RUNTIME)`
  - Exécution

186

```
class Mere {
  public void methode() {
    System.out.println("Methode de Mere");
  }
}
```

```
class Fille extends Mere {
  @Override
  public void Methode() {
    System.out.println("Methode de Fille");
  }
}
```

```
Fille f = new Fille();
f.methode();
```

The method Methode() of Fille must override or implement a supertype method

187

## Annotations personnalisées

```
@interface ClassPreamble {
  String author();
  String date();
  int currentRevision() default 1;
  String lastModified() default "N/A";
  String lastModifiedBy() default "N/A";
  String[] reviewers();
  // utilisation possible des tableaux
}
```

Aller plus loin :  
<http://download.oracle.com/javase/tutorial/java/javaOO/annotations.html>  
<http://www.jmdoudoux.fr/java/dej/chap010.htm#annotations>

## Méthode à nombre d'arguments variable

```
public void somme(double ... nombres) {
  double s = 0.0;

  // méthode classique
  for(int i=0; i < nombres.length; ++i)
    s += nombres[i];

  // nouvelle forme de for
  for (double nb: nombres) s+= nb;
}
```

188

189



## Auto [un] Boxing

5

- Conversions automatiques
- AutoBoxing
  - attention à la création implicite d'objets

```
Integer nombre = 10; // conversion automatique
Integer nombre = new Integer(10);
```

- Auto unboxing

```
int n = nombre; // converti automatiquement
int n = nombre.intValue();
```

Avertissements / Erreurs possibles sous Eclipse



5

**ISIMA**



Généricité &  
Collections

## Généricité

- Classes paramétrées
- Méthodes paramétrées
- Wildcards (`<? extends classe>`)
  - En lecture (pas en création)
- Définition générique
  - Compilée une fois pour toutes ( $\neq$  C++), efficacité ?
  - Partagée par toutes ses invocations
- Pas de `typedef`
  - paramètre : une lettre en majuscule (convention)
  - Éviter de dériver une classe pour donner un nom

Plus de renseignements :  
<http://www.oracle.com/technetwork/java/javase/generics-tutorial-159168.pdf>



## Généricité : exemple

5

```
public interface List<E> {
    void add(E e);
    Iterator<E> iterator();
}

public interface Iterator<E> {
    E next();
    boolean hasNext();
}

public void dessiner(List<E extends ObjetGraphique> l) {
    static<T> void ajouter(T[] tab, Collection<T> c) {
        for(T o : tab)
            c.add(o);
    }
}
```



193

## Collections (1)

- Gérer des collections dynamiques d'objets
  - Conteneurs non générique d'objets de classe `Object`
  - Conteneurs génériques (Java 1.5+)
  - Incompatibilité de plateforme
- Algorithmes
  - Tri
  - Recherche
  - Manipulation
- Paquetage `java.util`

Options -source et -target

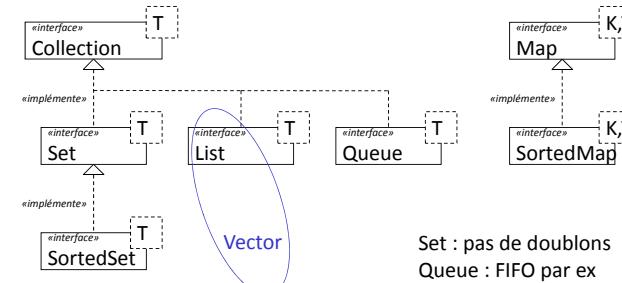


<http://download.oracle.com/javase/tutorial/collections/index.html>



194

## Collection (2)



Set : pas de doublons  
 Queue : FIFO par ex  
 Map : table de hachage

195

## Collections (3)

5

- Avant Java 1.5
  - Conteneurs d'Object
  - Downcast obligatoire
  - Vector [Vecteur dynamique]
  - Enumeration [itérateur]
  - Dictionary (Hashtable) [Map]

- Après 1.5
  - Mêmes conteneurs templatisés
  - Utiliser `ArrayList` plutôt que `Vector` sauf multithread
  - Utiliser la `Map` plutôt que `Dictionary`



196

## Exemple (non générique)

```
Vector v = new Vector();
// on en peut stocker que des objets, int est exclu

for (int i=0; i<10; ++i)
    v.addElement(new Integer(i));

// transtypage obligatoire
int somme = 0;
for (int i=0; i<10; ++i)
    somme += ((Integer)v.elementAt(i)).intValue();

// avec une énumération
Enumeration e = v.elements();
while (e.hasMoreElements())
    somme += ((Integer)e.nextElement()).intValue();
```



197

## Exemple (générique)

```
Vector<Integer> v = new Vector<Integer>();
// on ne peut toujours pas stocker de int

for (int i=0; i<10; ++i)
    v.addElement(new Integer(i));
// plus de transtypage

int somme = 0;
for (int i=0; i<10; ++i)
    somme += v.elementAt(i).intValue();

// avec une énumération, elle aussi paramétrée
Enumeration<Integer> e = v.elements();
while (e.hasMoreElements())
    somme += e.nextElement().intValue();
```

198

## Nouveau for

```
ArrayList<Integer> al = new ...; // Collection
int s = 0; // somme

for(int i = 0; i < al.size(); ++i) {
    s += al.get(i).intValue();
}

for(Integer i : al) {
    s += i.intValue();
}
```

**ISIMA**



Applet



## Applet / Appliquette

- Embarquée dans une page Ouaib
  - Butineur
  - Outil *appletviewer*
  - Java plug-in*
- Different d'une application autonome
  - Sécurité renforcée (Gestionnaire de sécurité) : Fichiers inaccessibles sauf JNLP (Java 6u10)
  - Gestion du son
- Classes : Applet ou JApplet (Swing)

## Code HTML

- Contient une balise décrivant l'applet à exécuter
  - Dans un fichier .class
  - Dans un fichier .jar

```
<applet code="Exemple.class"
        codebase=""
        archive="ex01.jar, ex02.jar"
        width="600" height="95">
    <param name="param1" value="1000">
    <param name="nom" value="loic">
</applet>
```

199

## Code JAVA

- ```
public String getParameter(String)
```
- Permet de récupérer un paramètre du fichier HTML
  - Méthodes liées à la vie de l'applet
    - init()
    - destroy()
  - Méthodes liées à l'exécution de l'applet
    - start()
    - stop()

202

**ISIMA**



Outils

203



## Outils

- javadoc
- appletviewer
- jar
- Netbeans



## javadoc

Outil externe pour générer une documentation au format HTML (similaire à la documentation officielle Java)

- Intégré au langage, commande en standard
- Format "universel" et simple
  - Commentaires dans le code
  - Autre outil similaire : Doxygen
- Lisibilité du code
- Le code doit compiler
- Ne fait pas tout !

```
/** */
```

205

206

## javadoc ?

- Commentaires spéciaux
- Personnalisation avec code HTML ou CSS
- Balises
  - @param
  - @return
  - @throws @exception
  - @see
  - @since
  - @deprecated
  - @author

```
/** */
```

Eclipse : (M) Project > Generate Javadoc

207

## Exemple

```
/**  
 * Description de la classe  
 *  
 */  
class ExempleJavaDoc {  
  
    /**  
     * description courte  
     *  
     * description longue .....  
     * @param param1 description  
     * @param param2 description  
     * @return le resultat de la methode  
     */  
    public int methode(int param1, double param2) {}  
}
```



## Appletviewer

Exécuter une applet sans butineur

`appletviewer page.html`

Attention au gestionnaire de sécurité !



## jar

- Java ARchive
  - Exécutable ( ≈ tar)
  - Format de fichier compressé
  - Contrôle de version par exemple
- Permet de distribuer/déployer
  - une application (autonome ou applet, JEE)
  - un package
- Contenu
  - Fichiers .class
  - Fichiers ressources (images)
  - Fichier *Manifest*

210

## Déployer une application JAVA

- Utilisation intensive de JAR
- Exécution protégée (sandbox)
  - Sécurité (Java Network Launching Protocol)
- Java Web Start (>1.4)
  - Pour les applications *standalone*
  - Simple clic dans un page Web
- Java Plug-in
  - Pour les Applets

<http://download.oracle.com/javase/tutorial/deployment/index.html>

**ISIMA**



## Bibliographie (1)

- Documentation JAVA de Sun/Oracle
- Tutoriaux Web de Sun/Oracle  
<http://download.oracle.com/javase/tutorial/>
- Cours JAVA d'Olivier Dedieu, INRA  
<http://www-sor.inria.fr/~dedieu/java/cours/>
- Intro JAVA, Bruno Bachelet  
<http://www.nawouak.net/?doc=java+lang=fr>

213



## Bibliographie (2)

- Thinking in Java, 2<sup>nd</sup> ed, Bruce Eckel
- Head First Java, 2<sup>nd</sup> ed, Kathy Sierra, Bert Bates , O'Reilly, 2005

## Aller plus loin...



- Java standard
  - Introspection
  - Génériques
  - Réseau, RMI
  - Threads avancés
  - JNLP
- Outils
  - Ant, Maven
  - Netbeans, IntelliJ IDE
- Java Entreprise
  - Glassfish, tomcat
  - Servlets, jsp, beans
  - Persistance (JPA, Hibernate)
  - Facelets (JSF)
  - Frameworks : Struts, Spring, ...



215