

Architectures des SI: programmation par composants

Informatique, Statistique et IA 4^{ème} année

Olivier Caron¹

<http://ocaron.polytech-lille.net>

¹École d'ingénieurs Polytech Lille
Université de Lille

11 janvier 2026



© Robert Cringely

If the automobile had followed the same development cycle as the computer, a Rolls-Royce would today cost \$100, get a million miles per gallon, and explode once a year, killing everyone inside.

Le module Architectures des Systèmes d'Information (ASI)

- Pré-requis :

Le module Architectures des Systèmes d'Information (ASI)

- Pré-requis :
 - ▶ La programmation par objets (Java)

Le module Architectures des Systèmes d'Information (ASI)

- Pré-requis :
 - ▶ La programmation par objets (Java)
 - ▶ Les bases de données (SQL)

Le module Architectures des Systèmes d'Information (ASI)

- Pré-requis :

- ▶ La programmation par objets (Java)
- ▶ Les bases de données (SQL)
- ▶ Notion de réseau (les concepts), grammaires (JSON, XML)

Le module Architectures des Systèmes d'Information (ASI)

- Pré-requis :
 - ▶ La programmation par objets (Java)
 - ▶ Les bases de données (SQL)
 - ▶ Notion de réseau (les concepts), grammaires (JSON, XML)
- Structuration et évaluation :

Le module Architectures des Systèmes d'Information (ASI)

- Pré-requis :
 - ▶ La programmation par objets (Java)
 - ▶ Les bases de données (SQL)
 - ▶ Notion de réseau (les concepts), grammaires (JSON, XML)
- Structuration et évaluation :
 - ▶ 18h de cours, 16h de TP, 14h de projet

Le module Architectures des Systèmes d'Information (ASI)

- Pré-requis :
 - ▶ La programmation par objets (Java)
 - ▶ Les bases de données (SQL)
 - ▶ Notion de réseau (les concepts), grammaires (JSON, XML)
- Structuration et évaluation :
 - ▶ 18h de cours, 16h de TP, 14h de projet
 - ▶ 1h30 DS, interrogation(s) de cours (auto-évaluation), 2h contrôle TP individuel, projet en binôme

Le module Architectures des Systèmes d'Information (ASI)

- Pré-requis :

- ▶ La programmation par objets (Java)
- ▶ Les bases de données (SQL)
- ▶ Notion de réseau (les concepts), grammaires (JSON, XML)

- Structuration et évaluation :

- ▶ 18h de cours, 16h de TP, 14h de projet
- ▶ 1h30 DS, interrogation(s) de cours (auto-évaluation), 2h contrôle TP individuel, projet en binôme
- ▶ Supports de cours et illustrations Java sur <http://ocaron.polytech-lille.net>

Le module Architectures des Systèmes d'Information (ASI)

- Pré-requis :
 - ▶ La programmation par objets (Java)
 - ▶ Les bases de données (SQL)
 - ▶ Notion de réseau (les concepts), grammaires (JSON, XML)
- Structuration et évaluation :
 - ▶ 18h de cours, 16h de TP, 14h de projet
 - ▶ 1h30 DS, interrogation(s) de cours (auto-évaluation), 2h contrôle TP individuel, projet en binôme
 - ▶ Supports de cours et illustrations Java sur <http://ocaron.polytech-lille.net>
 - ▶ Projets exemples sur le serveur GitLab  de l'université

Motivation

- Découvrir intuitivement la notion d'Architectures Logicielles et de SI :
 Motivation

Plan du module ASI

- Des objets aux composants

Plan du module ASI

- Des objets aux composants
 - ▶ Le modèle de composants Java Beans

Plan du module ASI

- Des objets aux composants
 - ▶ Le modèle de composants Java Beans
- Des objets aux composants d'entreprise répartis

Plan du module ASI

- Des objets aux composants
 - ▶ Le modèle de composants Java Beans
- Des objets aux composants d'entreprise répartis
 - ▶ Les architectures clients/serveurs 3-tiers

Plan du module ASI

- Des objets aux composants
 - ▶ Le modèle de composants Java Beans
- Des objets aux composants d'entreprise répartis
 - ▶ Les architectures clients/serveurs 3-tiers
 - ★ L'architecture JEE (composants Web, composants EJB)
 - Concepts d'objets répartis
 - Le patron de conception MVC (Modèle-Vue-Contrôleur)

Plan du module ASI

- Des objets aux composants
 - ▶ Le modèle de composants Java Beans
- Des objets aux composants d'entreprise répartis
 - ▶ Les architectures clients/serveurs 3-tiers
 - ★ L'architecture JEE (composants Web, composants EJB)
 - Concepts d'objets répartis
 - Le patron de conception MVC (Modèle-Vue-Contrôleur)
- Introduction aux web services, web 2.0

Plan du module ASI

- Des objets aux composants
 - ▶ Le modèle de composants Java Beans
- Des objets aux composants d'entreprise répartis
 - ▶ Les architectures clients/serveurs 3-tiers
 - ★ L'architecture JEE (composants Web, composants EJB)
 - Concepts d'objets répartis
 - Le patron de conception MVC (Modèle-Vue-Contrôleur)
- Introduction aux web services, web 2.0
- Conception d'applications 3-tiers

Plan du module ASI

- Des objets aux composants
 - ▶ Le modèle de composants Java Beans
- Des objets aux composants d'entreprise répartis
 - ▶ Les architectures clients/serveurs 3-tiers
 - ★ L'architecture JEE (composants Web, composants EJB)
 - Concepts d'objets répartis
 - Le patron de conception MVC (Modèle-Vue-Contrôleur)
- Introduction aux web services, web 2.0
- Conception d'applications 3-tiers
- Les architectures de SI et le Cloud Computing

Problématique ?

- Les applications logicielles deviennent de plus en plus complexes :

Problématique ?

- Les applications logicielles deviennent de plus en plus complexes :
 - ▶ L'application ne fonctionne pas sur une machine mais sur plusieurs.
→ programmation réseau, efficacité, performances, sécurité, tolérance aux pannes, . . .

Problématique ?

- Les applications logicielles deviennent de plus en plus complexes :
 - ▶ L'application ne fonctionne pas sur une machine mais sur plusieurs.
→ programmation réseau, efficacité, performances, sécurité, tolérance aux pannes, . . .
 - ▶ Interopérabilité des langages et plates-formes.

Problématique ?

- Les applications logicielles deviennent de plus en plus complexes :
 - ▶ L'application ne fonctionne pas sur une machine mais sur plusieurs.
→ programmation réseau, efficacité, performances, sécurité, tolérance aux pannes, . . .
 - ▶ Interopérabilité des langages et plates-formes.
 - ▶ L'application utilise/ou réutilise des applications existantes.
→ applications patrimoines, bases de données

Des objectifs ambitieux

- Face à cette complexité, on veut, en outre :

Des objectifs ambitieux

- Face à cette complexité, on veut, en outre :
 - ▶ Améliorer la qualité et la production de code

Des objectifs ambitieux

- Face à cette complexité, on veut, en outre :
 - ▶ Améliorer la qualité et la production de code
 - ▶ Simplifier le travail du programmeur

Des objectifs ambitieux

- Face à cette complexité, on veut, en outre :
 - ▶ Améliorer la qualité et la production de code
 - ▶ Simplifier le travail du programmeur
 - ▶ Réutiliser au mieux le logiciel

L'industrie du logiciel

- Pour 10 projets de développement logiciel, __ sont des succès

L'industrie du logiciel

- Pour 10 projets de développement logiciel, — sont des succès
- Fiabilité des projets, spécification des règles de gestion, tests

L'industrie du logiciel

- Pour 10 projets de développement logiciel, __ sont des succès
- Fiabilité des projets, spécification des règles de gestion, tests
- Vol 501 d'Ariane 5, bug informatique : débordement de capacité



L'industrie du logiciel

- Pour 10 projets de développement logiciel, — sont des succès
- Fiabilité des projets, spécification des règles de gestion, tests
- Vol 501 d'Ariane 5, bug informatique : débordement de capacité



- Réalisées après la catastrophe, les simulations en laboratoire ont permis de vérifier que l'explosion était inéluctable.

- Définir en premier lieu l'architecture logicielle la mieux adaptée :
*A software architecture is an **abstract system specification** consisting primarily of functional **components** described in terms of their behaviors and interfaces and component-component interconnections". (Hayes-Roth, 1994)*

Les solutions

- Définir en premier lieu l'architecture logicielle la mieux adaptée :
*A software architecture is an **abstract system specification** consisting primarily of functional **components** described in terms of their behaviors and interfaces and component-component interconnections". (Hayes-Roth, 1994)*
- Disposer de briques logicielles réutilisables : les composants.

Les solutions

- Définir en premier lieu l'architecture logicielle la mieux adaptée :
*A software architecture is an **abstract system specification** consisting primarily of functional **components** described in terms of their behaviors and interfaces and component-component interconnections". (Hayes-Roth, 1994)*
- Disposer de briques logicielles réutilisables : les composants.
- Disposer d'infrastructures d'accueil de ces composants : les serveurs d'applications.

Les apports de la programmation par composants

- Ne pas partir de zéro, existence de composants **efficaces** et **robustes** :

Les apports de la programmation par composants

- Ne pas partir de zéro, existence de composants **efficaces** et **robustes** :
 - ▶ Navigateur HTML (aide en ligne), correcteur orthographique, chargement, restauration de fichier,...

Les apports de la programmation par composants

- Ne pas partir de zéro, existence de composants **efficaces** et **robustes** :
 - ▶ Navigateur HTML (aide en ligne), correcteur orthographique, chargement, restauration de fichier,...
- Plus facile à décrire une application complexe :

Les apports de la programmation par composants

- Ne pas partir de zéro, existence de composants **efficaces** et **robustes** :
 - ▶ Navigateur HTML (aide en ligne), correcteur orthographique, chargement, restauration de fichier,...
- Plus facile à décrire une application complexe :
 - ▶ Ensemble de composants

Les apports de la programmation par composants

- Ne pas partir de zéro, existence de composants **efficaces** et **robustes** :
 - ▶ Navigateur HTML (aide en ligne), correcteur orthographique, chargement, restauration de fichier,...
- Plus facile à décrire une application complexe :
 - ▶ Ensemble de composants
 - ▶ Connection entre ces composants

Qu'est-ce qu'un composant ?

- 4 exigences pour être considéré comme composant logiciel :

Qu'est-ce qu'un composant ?

- 4 exigences pour être considéré comme composant logiciel :
 - ① Module logiciel **autonome** pouvant être installé sur différentes plateformes

Qu'est-ce qu'un composant ?

- 4 exigences pour être considéré comme composant logiciel :
 - ① Module logiciel **autonome** pouvant être installé sur différentes plateformes
 - ② Mécanisme d'introspection (auto-descriptif)

Qu'est-ce qu'un composant ?

- 4 exigences pour être considéré comme composant logiciel :
 - ❶ Module logiciel **autonome** pouvant être installé sur différentes plateformes
 - ❷ Mécanisme d'introspection (auto-descriptif)
 - ❸ Conservation de son état

Qu'est-ce qu'un composant ?

- 4 exigences pour être considéré comme composant logiciel :
 - ① Module logiciel **autonome** pouvant être installé sur différentes plateformes
 - ② Mécanisme d'introspection (auto-descriptif)
 - ③ Conservation de son état
 - ④ Paramétrable ou configurable

Les composants en Java : les Java Beans

<http://docs.oracle.com/javase/tutorial/javabeans/>

- 4 exigences pour être considéré comme composant logiciel :

Les composants en Java : les Java Beans

<http://docs.oracle.com/javase/tutorial/javabeans/>

- 4 exigences pour être considéré comme composant logiciel :
 - 1 Module logiciel autonome pouvant être installé sur différentes plateformes → fichier jar

Les composants en Java : les Java Beans

<http://docs.oracle.com/javase/tutorial/javabeans/>

- 4 exigences pour être considéré comme composant logiciel :
 - 1 Module logiciel autonome pouvant être installé sur différentes plateformes → **fichier jar**
 - 2 Mécanisme d'introspection (auto-descriptif) → **reflection Java**

Les composants en Java : les Java Beans

<http://docs.oracle.com/javase/tutorial/javabeans/>

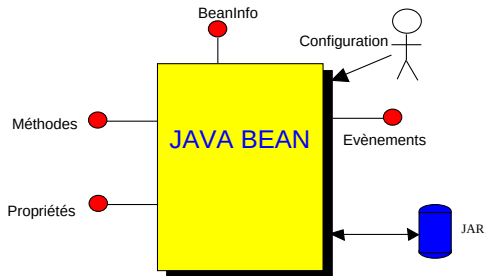
- 4 exigences pour être considéré comme composant logiciel :
 - ❶ Module logiciel autonome pouvant être installé sur différentes plateformes → **fichier jar**
 - ❷ Mécanisme d'introspection (auto-descriptif) → **reflection Java**
 - ❸ Conservation de son état → **sérialisation**

Les composants en Java : les Java Beans

<http://docs.oracle.com/javase/tutorial/javabeans/>

- 4 exigences pour être considéré comme composant logiciel :
 - ❶ Module logiciel autonome pouvant être installé sur différentes plateformes → **fichier jar**
 - ❷ Mécanisme d'introspection (auto-descriptif) → **reflection Java**
 - ❸ Conservation de son état → **sérialisation**
 - ❹ Paramétrable ou configurable → **sérialisation + réflexion + évènements**

Structure d'un Java Bean



Premier Bean : SmileyBean

```
package fr.polytechlille.is4 ;  
import ...  
  
public class SmileyBean extends javax.swing.JPanel {  
    private Color ourColor = Color.yellow;  
    private boolean smile = true;  
  
    public SmileyBean() { this.setSize(250,250); }  
  
    public synchronized void toggleSmile() {  
        smile = !smile; this.repaint();  
    }  
  
    public void paintComponent(Graphics g) { ...}  
}
```

Premières remarques :

- Ici, pas de constructions supplémentaires

Premières remarques :

- Ici, pas de constructions supplémentaires
- Deux *règles Bean* **obligatoires** :

Premières remarques :

- Ici, pas de constructions supplémentaires
- Deux *règles Bean* **obligatoires** :
 - ▶ Fonction constructeur sans paramètres

Premières remarques :

- Ici, pas de constructions supplémentaires
- Deux *règles Bean* **obligatoires** :
 - ▶ Fonction constructeur sans paramètres
 - ▶ Classe sérialisable

Premières remarques :

- Ici, pas de constructions supplémentaires
- Deux *règles Bean* **obligatoires** :
 - ▶ Fonction constructeur sans paramètres
 - ▶ Classe sérialisable
- Une classe simple **peut** correspondre à un Java Bean.

Première application

```
package fr.polytechlille.is4 ;
import java.beans.*;import java.awt.event.* ;
public class FirstSmileyApp extends javax.swing.JFrame
    implements WindowListener {
    private SmileyBean smiley = null;

    public FirstSmileyApp(String titre) {
        super(titre) ;
        try {
            smiley =(SmileyBean)Beans.instantiate( null ,
                "fr.polytechlille.is4.SmileyBean");
        } catch (Exception e) {
            System.err.println("Exception:"+e.getMessage());
            System.exit(1) ;
        }

        this.add("Center",smiley);  this.addWindowListener(this);
    }
    static public void main(String args[]) {...
```

La classe `java.beans.Beans`

- Fournit différents services

La classe java.beans.Beans

- Fournit différents services
- La création d'un Beans **peut** ne pas se faire par new

```
package java.beans ;  
public class Beans extends Object {  
    ...  
    public static Object instantiate(ClassLoader cls,  
        String beanName)  
        throws IOException, ClassNotFoundException ;  
    ...  
}
```

La classe java.beans.Beans

- Fournit différents services
- La création d'un Beans **peut** ne pas se faire par new

```
package java.beans ;  
public class Beans extends Object {  
    ...  
    public static Object instantiate(ClassLoader cls,  
        String beanName)  
        throws IOException, ClassNotFoundException ;  
    ...
```

- ▶ Chargement du class loader (null équivaut au system class loader)

La classe java.beans.Beans

- Fournit différents services

- La création d'un Beans **peut** ne pas se faire par new

```
package java.beans ;  
public class Beans extends Object {  
    ...  
    public static Object instantiate(ClassLoader cls,  
        String beanName)  
        throws IOException, ClassNotFoundException ;  
    ...
```

- ▶ Chargement du class loader (null équivaut au system class loader)
- ▶ Tentative de charger beanName.ser puis création du bean

Propriétés des composants - Conventions de nommage

- `public int getProp()`
`public void setProp(int valeur)`

Propriétés des composants - Conventions de nommage

- `public int getProp()`
`public void setProp(int valeur)`
 - ▶ Définit une propriété `entière` de nom `prop`

Propriétés des composants - Conventions de nommage

- `public int getProp()`
`public void setProp(int valeur)`
 - ▶ Définit une propriété `entière` de nom `prop`
- `public boolean isVisible()`
`public void setVisible(boolean valeur)`

Propriétés des composants - Conventions de nommage

- `public int getProp()`
`public void setProp(int valeur)`
 - ▶ Définit une propriété `entière` de nom `prop`
- `public boolean isVisible()`
`public void setVisible(boolean valeur)`
 - ▶ Définit une propriété `booléenne` de nom `visible`

Propriétés des composants - Conventions de nommage

- `public int getProp()`
`public void setProp(int valeur)`
 - ▶ Définit une propriété `entière` de nom `prop`
- `public boolean isVisible()`
`public void setVisible(boolean valeur)`
 - ▶ Définit une propriété `booléenne` de nom `visible`
- Possibilité de ne pas utiliser les conventions de nommage (PropertyDescriptor)

Propriétés des composants - Conventions de nommage

- `public int getProp()`
`public void setProp(int valeur)`
 - ▶ Définit une propriété **entière** de nom **prop**
- `public boolean isVisible()`
`public void setVisible(boolean valeur)`
 - ▶ Définit une propriété **booléenne** de nom **visible**
- Possibilité de ne pas utiliser les conventions de nommage (PropertyDescriptor)
- La réflexion Java permet d'analyser **dynamiquement** un composant et de le configurer dynamiquement !

Phases associées aux composants

- Récupération d'un composant, soit fichier `.class`, soit fichier `.jar`

Phases associées aux composants

- Récupération d'un composant, soit fichier `.class`, soit fichier `.jar`
- Découverte du composant : que peut-on faire avec ce composant ?
Quelles sont les méthodes de ce composant ?
→ **Introspection**

Phases associées aux composants

- Récupération d'un composant, soit fichier `.class`, soit fichier `.jar`
- Découverte du composant : que peut-on faire avec ce composant ?
Quelles sont les méthodes de ce composant ?
→ **Introspection**
- Utiliser ce composant : exécuter les méthodes découvertes
→ **Invocation dynamique**

Introspection et invocation dynamique (1/8)

- Soit le programme `Reflection.java`

Introspection et invocation dynamique (1/8)

- Soit le programme `Reflection.java`
- Ce programme accepte en argument n'importe quel nom de classe (accessible dans le CLASSPATH)

Introspection et invocation dynamique (1/8)

- Soit le programme `Reflection.java`
- Ce programme accepte en argument n'importe quel nom de classe (accessible dans le CLASSPATH)
- Objectif du programme :

Introspection et invocation dynamique (1/8)

- Soit le programme `Reflection.java`
- Ce programme accepte en argument n'importe quel nom de classe (accessible dans le CLASSPATH)
- Objectif du programme :
 - ❶ Découvrir si la classe en question dispose de propriétés booléennes, entières, réelles ou chaînes de caractères.

Introspection et invocation dynamique (1/8)

- Soit le programme `Reflection.java`
- Ce programme accepte en argument n'importe quel nom de classe (accessible dans le CLASSPATH)
- Objectif du programme :
 - ➊ Découvrir si la classe en question dispose de propriétés booléennes, entières, réelles ou chaînes de caractères.
 - ➋ Modifier les valeurs des propriétés découvertes.

Introspection et invocation dynamique (2/8)

java Reflection java.awt.Label

```
-- Partie Introspection de java.awt.Label --
int property found : alignment
java.lang.String property found : text
boolean property found : focusable
boolean property found : visible
boolean property found : enabled
java.lang.String property found : name
-- Partie Invocation dynamique --
invoke method setFocusable with boolean parameter
invoke method setAlignment with int parameter
invoke method setText with java.lang.String parameter
invoke method setEnabled with boolean parameter
invoke method setVisible with boolean parameter
invoke method setName with java.lang.String parameter
```

Introspection et invocation dynamique (3/8)

```
import java.lang.reflect.* ;
import java.beans.Beans ;
import java.util.HashMap;

public class Reflection {
    private HashMap<String , Class> recognizedTypes ;
    private HashMap<Method, String> detectedSetMethods ;

    public Reflection() {
        this.recognizedTypes=new HashMap<String , Class>() ;
        this.recognizedTypes.put("boolean",Boolean.TYPE) ;
        this.recognizedTypes.put("int",Integer.TYPE) ;
        this.recognizedTypes.put("double",Double.TYPE) ;
        this.recognizedTypes.put("java.lang.String",String.class) ;
    }
}
```

Introspection et invocation dynamique (4/8)

```
public boolean isRecognizedType (String typeName) {  
    return this.recognizedTypes.get(typeName) != null ;  
}  
  
public Class [] getParamForSetter(String typeName) {  
    return new Class [] { this.recognizedTypes.get(typeName) };  
}  
  
public static void main(String args[]) {  
    Reflection analyseComponent = new Reflection() ;  
    analyseComponent.run(args[0]) ;  
}
```

Introspection et invocation dynamique (5/8)

```
public void run(String className) {
    String setName, propertyName ; int ind ;
    this.detectedSetMethods = new HashMap<Method, String>();
    System.out.println("— Partie Introspection de "+className+" —");
    try {
        Object obj = Beans.instantiate(null, className) ;
        Class<?> laClasse = obj.getClass() ;
        for (Method m : laClasse.getMethods()) {
            String typeName=m.getReturnType().getName() ;
            if ((m.getName().startsWith("get") ||
                (m.getName().startsWith("is") && typeName.equals("boolean")))
                && m.getParameterTypes().length==0
                && isRecognizedType(typeName)) { // get method found
                if (typeName.equals("boolean")) ind=2 ; else ind=3 ;
                setName="set "+m.getName().substring(ind) ;
                propertyName = m.getName().substring(ind, ind+1).toLowerCase()
                    + m.getName().substring(ind+1) ;
            }
        }
    }
}
```


Introspection et invocation dynamique (6/8)

```
try {  
    Method methodeSet=  
        laClasse.getMethod(setName, getParamForSetter(typeName));  
    detectedSetMethods.put(methodeSet, typeName);  
    System.out.println(typeName+" property found: "  
                        +propertyName);  
} catch (NoSuchMethodException e1) {  
    System.err.println("No setter method found: "+  
                      setName);  
}  
}  
}
```

Introspection et invocation dynamique (7/8)

```
System.out.println("— Partie Invocation dynamique —") ;
for (Method m : detectedSetMethods.keySet()) {
    String typeName=detectedSetMethods.get(m) ;
    System.out.println("invoke method "+m.getName()
                      +" with "+typeName+" parameter") ;
    if (typeName.equals("boolean")) m.invoke(obj, true) ;
    else if (typeName.equals("int") ) m.invoke(obj,0) ;
    else if (typeName.equals("double") ) m.invoke(obj,0.0) ;
    else if (typeName.equals("String") )
        m.invoke(obj,"Hello World !") ;
}
```

Introspection et invocation dynamique (8/8)

```
} catch (ArrayIndexOutOfBoundsException e) {  
    System.err.println("Error: java Reflection className");  
}  
} catch (SecurityException e) {  
    System.err.println("exception raised ...") ;  
}  
} catch (IllegalAccessException e) {  
    System.err.println("Access ...") ;  
}  
} catch (InvocationTargetException e) {  
    System.err.println("Argument ...") ;  
}  
} catch (IllegalArgumentException e) {  
    System.err.println("Argument ...") ;  
}  
} catch (java.io.IOException e) {  
    System.err.println("IO ...") ;  
}  
} catch (ClassNotFoundException e) {  
    System.err.println("class ...") ; } } }
```

Les environnements à base de composants

- Possibilité de construire des outils paramétrables, des descripteurs de composants :

Les environnements à base de composants

- Possibilité de construire des outils paramétrables, des descripteurs de composants :
 - ▶ BeanBox, BeanBuilder, NetBeans (SUN)

Les environnements à base de composants

- Possibilité de construire des outils paramétrables, des descripteurs de composants :
 - ▶ BeanBox, BeanBuilder, NetBeans (SUN)
 - ▶ JBuilder (Inprise)

Les environnements à base de composants

- Possibilité de construire des outils paramétrables, des descripteurs de composants :
 - ▶ BeanBox, BeanBuilder, NetBeans (SUN)
 - ▶ JBuilder (Inprise)
 - ▶ Visual Cafe (Symantec),...

Les environnements à base de composants

- Possibilité de construire des outils paramétrables, des descripteurs de composants :
 - ▶ BeanBox, BeanBuilder, NetBeans (SUN)
 - ▶ JBuilder (Inprise)
 - ▶ Visual Cafe (Symantec),...
- d'autres outils : ex : BeanShell, groovy

Les évènements (1/3)

- Basé sur "Observer Design Pattern" :

Les évènements (1/3)

- Basé sur "Observer Design Pattern" :

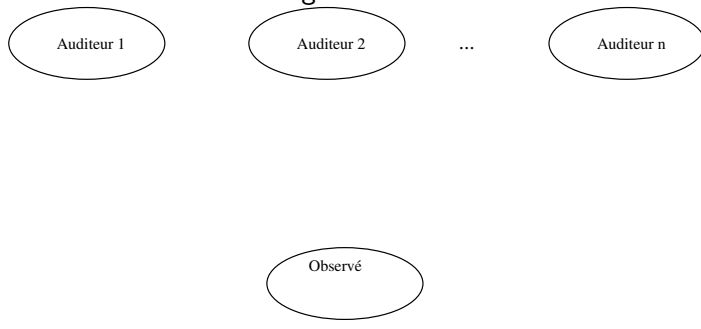
Les évènements (1/3)

- Basé sur "Observer Design Pattern" :



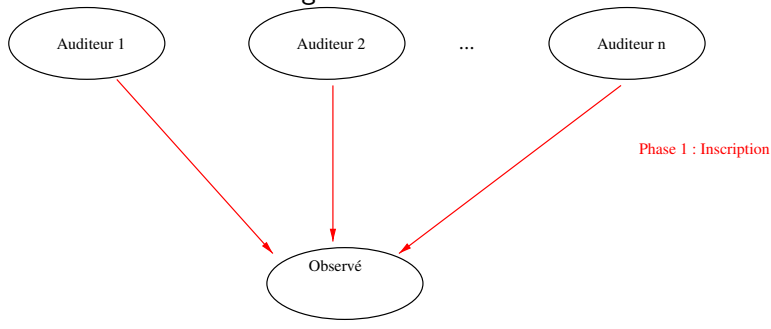
Les évènements (1/3)

- Basé sur "Observer Design Pattern" :



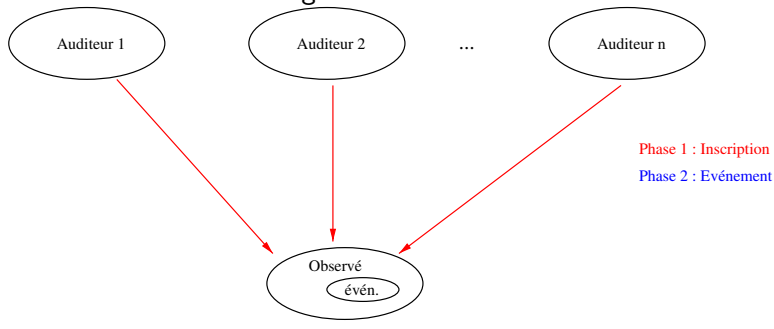
Les évènements (1/3)

- Basé sur "Observer Design Pattern" :



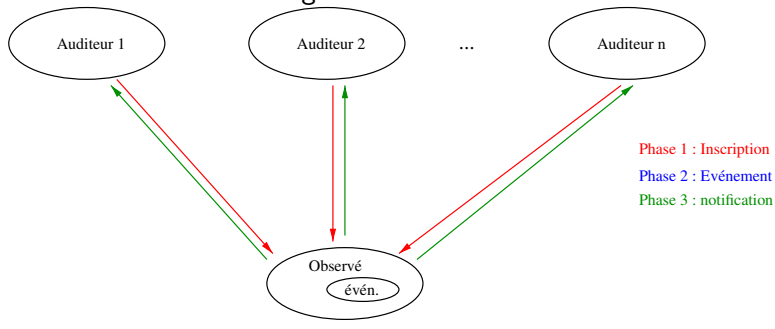
Les évènements (1/3)

- Basé sur "Observer Design Pattern" :



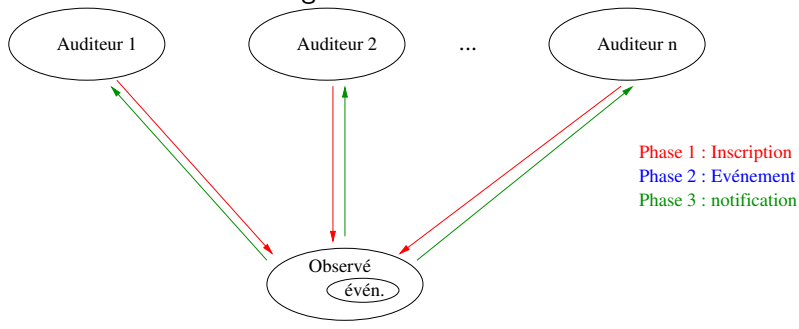
Les évènements (1/3)

- Basé sur "Observer Design Pattern" :



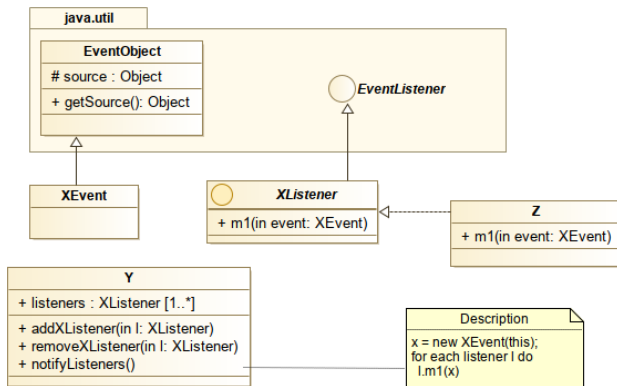
Les évènements (1/3)

- Basé sur "Observer Design Pattern" :



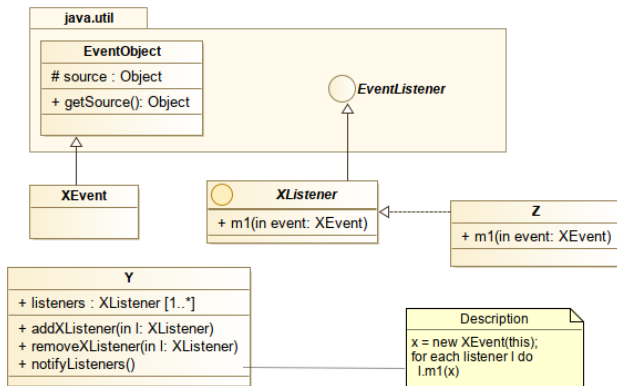
Les évènements (2/3)

- Illustration : le modèle des composants graphiques Java



Les évènements (2/3)

- Illustration : le modèle des composants graphiques Java



► $X \rightarrow \text{Action}$, $Y \rightarrow \text{JButton}$

Les évènements (3/3)

```
package java.util;  
public class EventObject implements java.io.Serializable {  
    protected transient Object source;  
    public EventObject(Object source) {...}  
    public Object getSource() { return source; }  
    public String toString() { ... }  
}
```

```
package java.util;  
public interface EventListener { }
```

Un nouvel évènement SourireEvent

- Objectif : on veut créer un nouvel évènement qui intervient quand le 'smiley' sourit.

```
import java.util.EventObject ;
```

```
public class SourireEvent extends EventObject {  
    public SourireEvent(SmileyBean src) { super(src); }  
}
```

- Plusieurs auditeurs peuvent être à l'écoute de cet évènement.

Interface pour les auditeurs

- Les auditeurs désirant capturer l'évènement doivent implémenter l'interface `SourireListener` :

```
import java.util.EventListener ;

public interface SourireListener extends EventListener {
    public void devientDrole(SourireEvent e) ;
}
```

La nouvelle classe SmileyBean (1/4)

- doit gérer les auditeurs (listeners) : ajout et retrait

La nouvelle classe SmileyBean (1/4)

- doit gérer les auditeurs (listeners) : ajout et retrait
- doit créer un évènement `SourireEvent` lorsque le 'smiley' sourit

La nouvelle classe SmileyBean (1/4)

- doit gérer les auditeurs (listeners) : ajout et retrait
- doit créer un évènement `SourireEvent` lorsque le 'smiley' sourit
- doit notifier à tous les auditeurs inscrits l'évènement

La nouvelle classe SmileyBean (1/4)

- doit gérer les auditeurs (listeners) : ajout et retrait
- doit créer un évènement `SourireEvent` lorsque le 'smiley' sourit
- doit notifier à tous les auditeurs inscrits l'évènement
- doit gérer les accès concurrents.

La nouvelle classe SmileyBean (2/4)

```
package fr.polytechlille.is4 ;
import javax.swing.JPanel ;
...
import java.util.ArrayList ;
public class SmileyBean extends JPanel {
    private boolean smile = true;
    private ArrayList<SourireListener> listeners ;

    public SmileyBean() {
        this.setSize(250,250);
        this.listeners = new ArrayList<SourireListener>() ;
    }
}
```

La nouvelle classe SmileyBean (3/4)

```
synchronized
public void addSourireListener(SourireListener l) {
    listeners.add(l) ;
}

synchronized
public void removeSourireListener(SourireListener l) {
    listeners.remove(l) ;
}

public synchronized void toggleSmile() {
    smile = !smile;
    if (smile) notifyListeners() ;
    this.repaint();
}
```

La nouvelle classe SmileyBean (4/4)

```
private void notifyListeners() {
    SourireEvent se=new SourireEvent(this) ;
    ArrayList<SourireListener> lv =null ;
    // realisation copie (acces concurrent)
    // cas ou un addListener en action
    synchronized(this) {
        lv= (ArrayList<SourireListener>) listeners.clone() ;
    }
    for (SourireListener l : lv) l.devientDrole(se) ;
}

public void paintComponent(Graphics g) { ... }
}
```

Conventions de nommage

- Evènement :

```
class NomEvent
```

Conventions de nommage

- Evènement :

```
class NomEvent
```

- Auditeur :

```
interface NomListener
```

Conventions de nommage

- Évènement :

```
class NomEvent
```

- Auditeur :

```
interface NomListener
```

- Classe source d'évènements :

```
public void addNomListener(NomListener l)
```

```
public void removeNomListener(NomListener l)
```

Connection de composants

- Le composant C1Bean génère l'événement Ev1, le composant C2Bean veut réagir pour cet événement.

Connection de composants

- Le composant C1Bean génère l'événement Ev1, le composant C2Bean veut réagir pour cet événement.
- Deux cas de figure sont possibles :

Connection de composants

- Le composant C1Bean génère l'événement Ev1, le composant C2Bean veut réagir pour cet événement.
- Deux cas de figure sont possibles :
 - ▶ Le composant C2Bean implémente l'auditeur Ev1Listener
⇒ `unC1.addEv1Listener(unC2)` ;

Connection de composants

- Le composant C1Bean génère l'événement Ev1, le composant C2Bean veut réagir pour cet événement.
- Deux cas de figure sont possibles :
 - ▶ Le composant C2Bean implémente l'auditeur Ev1Listener
⇒ `unC1.addEv1Listener(unC2)` ;
 - ▶ Le composant C2Bean n'implémente pas l'auditeur Ev1Listener

Connection de composants

- Le composant C1Bean génère l'événement Ev1, le composant C2Bean veut réagir pour cet événement.
- Deux cas de figure sont possibles :
 - ▶ Le composant C2Bean implémente l'auditeur Ev1Listener
⇒ `unC1.addEv1Listener(unC2)` ;
 - ▶ Le composant C2Bean n'implémente pas l'auditeur Ev1Listener
 - ★ Utilisation du design pattern *Adaptateur*

Application du patron de conception *Adaptateur* (solution 1)

- Définition de la classe Adaptateur :

```
public class Adaptateur implements Ev1Listener {  
    private C2Bean cible ;  
    public Adaptateur (C2Bean cible) { this.cible=cible ; }  
  
    /* les methodes de Ev1Listener */  
    public void m1(Ev1Event e) {  
        cible.methode() ;  
    }  
    ...  
}
```

- Utilisation et connexion des composants :

```
Adaptateur adapt=new Adaptateur(unC2) ;  
unC1.addEv1Listener(adapt) ;
```

Application du patron de conception *Adaptateur* (solution 2)

- Définition d'un adaptateur par classe **anonyme**
- Raccourci syntaxique :

```
unC1.addEv1Listener(  
    new Ev1Listener() {  
        /* les methodes de Ev1Listener */  
        public void m1(Ev1Event e) {  
            unC2.methode() ;  
        }  
    }  
);
```

Le composant SmileyLabelBean

```
package fr.polytechlille.is4 ;

import javax.swing.JLabel ;

public class SmileyLabelBean extends JLabel {
    private int compteur=0 ;
    public SmileyLabelBean() { super("compteur:0") ; }

    public void incr() {
        compteur++ ;
        this.setText("compteur:"+compteur) ;
    }
}
```

Exemple Adaptateur (solution 1)

- Définition de la classe Adaptateur :

```
public class Adaptateur implements SourireListener {  
    private SmileyLabelBean cible ;  
  
    public Adaptateur(SmileyLabelBean cible) {  
        this.cible=cible ;  
    }  
  
    public void devientDrole(SourireEvent e) {  
        this.cible.incr() ;  
    }  
}
```

- Utilisation et connexion des composants :

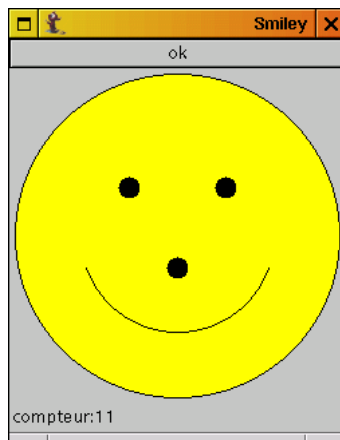
```
Adaptateur adapt=new Adaptateur(unC2) ;  
unC1.addSourireListener(adapt) ;
```


Exemple Adaptateur (solution 2)

- Définition et connexion d'un adaptateur par classe **anonyme**

```
unC1.addSourireListener(  
    new SourireListener() {  
        public void devientDrole(SourireEvent e) {  
            unC2.incr();  
        }  
    }  
);
```

Exemple application (1/3)



Exemple application (2/3)

```
package fr.polytechlille.is4 ;  
import javax.swing.* ;  
import java.awt.event.* ;  
  
public class SmileyApp extends javax.swing.JFrame  
implements WindowListener, ActionListener {  
  
    private SmileyBean smiley ;  
  
    public SmileyApp(String titre) {  
        super(titre) ;  
        SmileyLabelBean slb=new SmileyLabelBean();  
        smiley = new SmileyBean();  
        JButton ok=new JButton("ok") ;
```

Exemple application (3/3)

```
ok.addActionListener (this) ;
smiley.addSourireListener(new Adaptateur(slb)) ;
this.addWindowListener(this);

this.add("Center",smiley); this.add("South",slb) ;
this.add("North",ok) ;
}
public void actionPerformed(ActionEvent e) {
    smiley.toggleSmile() ;
}
...
```

Que fait au juste l'application ?

- Elle n'utilise que des composants (norme Java Beans) :
Button, SmileyBean, SmileyLabelBean

Que fait au juste l'application ?

- Elle n'utilise que des composants (norme Java Beans) :
Button, SmileyBean, SmileyLabelBean
- Elle connecte les composants entre eux
addActionListener, addSourireListener

Que fait au juste l'application ?

- Elle n'utilise que des composants (norme Java Beans) :
`Button`, `SmileyBean`, `SmileyLabelBean`
- Elle connecte les composants entre eux
`addActionListener`, `addSourireListener`
- Elle n'appelle que des méthodes existantes
`toggleSmile`

Que fait au juste l'application ?

- Elle n'utilise que des composants (norme Java Beans) :
Button, SmileyBean, SmileyLabelBean
- Elle connecte les composants entre eux
addActionListener, addSourireListener
- Elle n'appelle que des méthodes existantes
toggleSmile

Alors ...

Pourquoi programmer l'application ?

Les outils dédiés

- Beanbox, BeanBuilder, NetBeans, ...

Les outils dédiés

- Beanbox, BeanBuilder, NetBeans, ...
- Charge dynamiquement des composants Beans
fichiers jar normalisés

Les outils dédiés

- Beanbox, BeanBuilder, NetBeans, ...
- Charge dynamiquement des composants Beans
fichiers jar normalisés
- Outil visuel de programmation

Structure physique du composant

- Fichier archive (jar) contenant :

Structure physique du composant

- Fichier archive (jar) contenant :
 - ▶ Fichiers compilés Java (*.class)

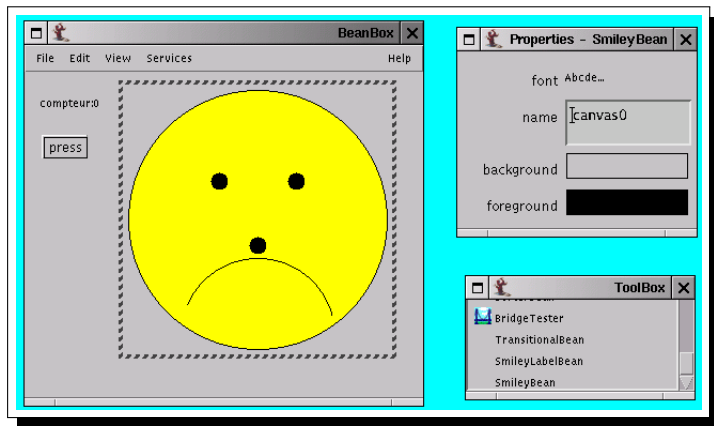
Structure physique du composant

- Fichier archive (jar) contenant :
 - ▶ Fichiers compilés Java (*.class)
 - ▶ Objets sérialisés (*.ser)

Structure physique du composant

- Fichier archive (jar) contenant :
 - ▶ Fichiers compilés Java (*.class)
 - ▶ Objets sérialisés (*.ser)
 - ▶ Fichier descriptif META-INF/MANIFEST.MF (facultatif)

La BeanBox en action



La sérialisation Java (1/2)

- Mécanisme permettant de rendre persistant des objets Java :
 - ▶ implémenter l'interface `java.io.Serializable`
 - ▶ ou bien hériter d'une classe *sérialisable*
- Sauvegarder un Java Bean :

```
FileOutputStream fic=new FileOutputStream("Smiley1.ser");  
ObjectOutputStream os=new ObjectOutputStream(fic);  
os.writeObject(unSmileyBean) ;
```

La sérialisation Java (2/2)

- Restaurer un Java Bean :

```
SmileyBean smiley ;  
try { // essai de récupérer le fichier .ser  
    smiley=(SmileyBean) Beans.instantiate(null ,"Smiley1" ) ;  
} catch(Exception e) {  
    smiley=new SmileyBean();  
}
```

- Clause Java transient permet d'ignorer des attributs (ex :
"dateDuJour")

Les propriétés multiples

- Les propriétés tableaux, conventions :

```
public Type[] getNomPropriété()  
public void setNomPropriété(Type[] valeur)
```

Les propriétés multiples

- Les propriétés tableaux, conventions :

```
public Type[] getNomPropriété()  
public void setNomPropriété(Type[] valeur)
```

- Les propriétés indicées, élément unique d'un tableau, conventions :

```
public Type getNomPropriété(int index)  
public void setNomPropriété(int index, Type valeur)
```

Les propriétés liées (1/3)

- Exploite le pattern Observateur
- C'est une propriété qui avertit tous les auditeurs des changements de valeur de la propriété.
- L'évènement est envoyé **après** la modification
- Un auditeur doit implémenter l'interface `PropertyChangeListener` :

```
package java.beans;  
    import java.util.EventListener ;  
  
    public interface PropertyChangeListener  
        extends EventListener {  
        void propertyChange(PropertyChangeEvent evt);  
    }  
}
```

Les propriétés liées (2/3)

- Le composant détenant des propriétés liées doit inclure des méthodes de recensement d'auditeurs :

- ▶ une seule propriété liée dans le composant :

```
public void addPropertyChangeListener(PropertyChangeListener l)
public void removePropertyChangeListener(PropertyChangeListener l)
```

- ▶ plusieurs propriétés liées dans le composant :

```
public void addPropertyChangeListener(
    String propertyName, PropertyChangeListener l)
public void removePropertyChangeListener(
    String propertyName, PropertyChangeListener l)
```

Les propriétés liées (3/3)

- L'évènement est du type :

```
package java.beans;
```

```
public class PropertyChangeEvent  
    extends java.util.EventObject {  
    public PropertyChangeEvent(  
        Object source, String propertyName,  
        Object oldValue, Object newValue) { ... }  
  
    public String getPropertyName() { return propertyName;}  
    public Object getNewValue() { return newValue;}  
    public Object getOldValue() { return oldValue; }  
}
```

Exemple de propriété liée (1/3)

```
import java.util.ArrayList ;
import java.beans.* ;

public class TemperatureBean extends javax.swing.JLabel {

    private double temperature=20.0 ;
    private ArrayList<PropertyChangeListener> listeners ;

    public TemperatureBean() {
        this.setText(""+temperature) ;
        this.listeners= new ArrayList<PropertyChangeListener>() ;
    }
}
```


Exemple de propriété liée (2/3)

```
synchronized public void addPropertyChangeListener  
                                (PropertyChangeListener l) {  
    listeners.add(l) ;  
}  
synchronized public void removePropertyChangeListener(  
                                PropertyChangeListener l) {  
    listeners.remove(l) ;  
}  
private void notifyListeners(PropertyChangeEvent event) {  
    ArrayList<PropertyChangeListener> lv = null ;  
    synchronized(this) {  
        lv= (ArrayList<PropertyChangeListener>) listeners.clone() ;  
    }  
    for (PropertyChangeListener pcl : lv)  
        pcl.propertyChange(event) ;  
}
```

Exemple de propriété liée (3/3)

```
public synchronized void setTemperature (double v) {  
    double old=this.temperature ;  
    this.temperature=v ;   PropertyChangedEventArgs event ;  
    event=new PropertyChangedEventArgs( this , "temperature",  
                                   new Double(old),new Double(temperature)) ;  
    this.notifyListeners(event) ;  
    this.setText(""+temperature) ;  
}  
public synchronized double getTemperature() {  
    return this.temperature ;  
}  
}
```

Les propriétés contraintes (1/2)

- Le changement de valeur peut être **bloqué** par un auditeur.
- L'auditeur implémente l'interface `VetoableChangeListener` :

```
package java.beans;  
import java.util.EventListener ;  
  
public interface VetoableChangeListener  
    extends EventListener{  
    void vetoableChange(PropertyChangeEvent evt)  
        throws PropertyVetoException;  
}
```

Les propriétés contraintes (2/2)

- L'auditeur peut déclencher l'exception `PropertyVetoException` pour refuser la modification de la propriété.

Les propriétés contraintes (2/2)

- L'auditeur peut déclencher l'exception `PropertyVetoException` pour refuser la modification de la propriété.
- Le composant possédant des propriétés contraintes doit gérer des auditeurs

Les propriétés contraintes (2/2)

- L'auditeur peut déclencher l'exception `PropertyVetoException` pour refuser la modification de la propriété.
- Le composant possédant des propriétés contraintes doit gérer des auditeurs
- Convention pour la définition de propriété contraintes :

```
public void setNomPropriété(Type valeur)  
throws java.beans.PropertyVetoException
```

Les composants Java Beans, c'est aussi...

- Des API pour des descripteurs de composants

Les composants Java Beans, c'est aussi...

- Des API pour des descripteurs de composants
- Des API pour des éditeurs de composants

Les composants Java Beans, c'est aussi...

- Des API pour des descripteurs de composants
- Des API pour des éditeurs de composants
- Des outils de *versioning* (utilitaire serialver)

Les composants Java Beans, c'est aussi...

- Des API pour des descripteurs de composants
- Des API pour des éditeurs de composants
- Des outils de *versioning* (utilitaire serialver)
- Des outils d'authentification (utilitaire javakey)

Les composants Java Beans, c'est aussi...

- Des API pour des descripteurs de composants
- Des API pour des éditeurs de composants
- Des outils de *versioning* (utilitaire serialver)
- Des outils d'authentification (utilitaire javakey)
- Des mécanismes de sécurité (JDK 2 : les permissions java)

Conclusion

- Facile ?

Conclusion

- Facile ?
- Puissant

Conclusion

- Facile ?
- Puissant
- Ce n'est pas un nouveau langage

Conclusion

- Facile ?
- Puissant
- Ce n'est pas un nouveau langage
- Permet une programmation très modulaire : chaque composant est parfaitement autonome.

Conclusion

- Facile ?
- Puissant
- Ce n'est pas un nouveau langage
- Permet une programmation très modulaire : chaque composant est parfaitement autonome.
- Portable

Conclusion

- Facile ?
- Puissant
- Ce n'est pas un nouveau langage
- Permet une programmation très modulaire : chaque composant est parfaitement autonome.
- Portable
- Interopérable avec composants ActiveX

Et pour finir...

© Ralph Johnson

Avant de vouloir qu'un logiciel soit réutilisable, il faudrait d'abord qu'il ait été utilisable.