Cours Architectures Logicielles

© Olivier Caron

Polytech'Lille

Problématique ?

$\surd$ 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.
  $\longrightarrow$ 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,. . .
- Intéropérabilité 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,. . .
- Intéropérabilité 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 :

- 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

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)

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.

Plan du cours (1/2)

- √ Des objets aux composants

Plan du cours (1/2)

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

Plan du cours (1/2)

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

Plan du cours (1/2)

- √ Des objets aux composants
  - ▶ Le modèle de composants Java Beans
- √ Des objets aux objets répartis
  - ▶ Le modèle Java RMI

Plan du cours (1/2)

- √ Des objets aux composants
  - Le modèle de composants Java Beans
- √ Des objets aux objets répartis
  - Le modèle Java RMI
  - *Le modèle CORBA*

Plan du cours (1/2)

- √ Des objets aux composants
  - Le modèle de composants Java Beans
- √ Des objets aux objets répartis
  - Le modèle Java RMI
  - *Le modèle CORBA*

Plan du cours (2/2)

- √ Des objets aux composants répartis
  - ▶ Les architectures clients/serveurs 3-tiers

Plan du cours (2/2)

- √ Des objets aux composants répartis
  - ▶ Les architectures clients/serveurs 3-tiers
    - L'architecture J2EE (composants Web, composants EJB)

Plan du cours (2/2)

- √ Des objets aux composants répartis
  - ▶ Les architectures clients/serveurs 3-tiers
    - L'architecture J2EE (composants Web, composants EJB)
    - Le modèle de composants CORBA

Plan du cours (2/2)

- √ Des objets aux composants répartis
  - ▶ Les architectures clients/serveurs 3-tiers
    - L'architecture J2EE (composants Web, composants EJB)
    - Le modèle de composants CORBA
- √ Conception d'applications 3-tiers

L'industrie du logiciel, aujourd'hui

√ Des produits chers !

L'industrie du logiciel, aujourd'hui

- √ Des produits chers !
  - Une bonne solution : le logiciel libre :-)

L'industrie du logiciel, aujourd'hui

√ Des produits chers !

- Une bonne solution : le logiciel libre :-)
- Une mauvaise solution : le piratage :-(

L'industrie du logiciel, aujourd'hui

- √ Des produits chers !
  - ▶ Une bonne solution : le logiciel libre :-)
  - ▶ Une mauvaise solution : le piratage :-(
- √ Existence de bugs !

L'industrie du logiciel, aujourd'hui

- √ Des produits chers !
  - ▶ Une bonne solution : le logiciel libre :-)
  - ▶ Une mauvaise solution : le piratage :-(
- √ Existence de bugs !
  - ▶ Succession des versions (à payer ?)

L'industrie du logiciel, aujourd'hui

- √ Des produits chers !
  - Une bonne solution : le logiciel libre :-)
  - Une mauvaise solution : le piratage :-(
- √ Existence de bugs !
  - Succession des versions (à payer ?)
  - Adéquation de la nouvelle version par rapport à la configuration matérielle

L'industrie du logiciel, aujourd'hui

- √ Des produits chers !
  - ▶ Une bonne solution : le logiciel libre :-)
  - ▶ Une mauvaise solution : le piratage :-(
- √ Existence de bugs !
  - ▶ Succession des versions (à payer ?)
  - ▶ Adéquation de la nouvelle version par rapport à la configuration matérielle
- √ Logiciel non adapté aux besoins :

L'industrie du logiciel, aujourd'hui

- √ Des produits chers !
  - ▶ Une bonne solution : le logiciel libre :-)
  - ▶ Une mauvaise solution : le piratage :-(
- √ Existence de bugs !
  - ▶ Succession des versions (à payer ?)
  - ▶ Adéquation de la nouvelle version par rapport à la configuration matérielle
- √ Logiciel non adapté aux besoins :
  - ▶ Trop de fonctionnalités (ex : fonctions EXCEL)

L'industrie du logiciel, aujourd'hui

- √ Des produits chers !
  - Une bonne solution : le logiciel libre :-)
  - Une mauvaise solution : le piratage :-(
- √ Existence de bugs !
  - Succession des versions (à payer ?)
  - Adéquation de la nouvelle version par rapport à la configuration matérielle
- √ Logiciel non adapté aux besoins :
  - Trop de fonctionnalités (ex : fonctions EXCEL)
  - Manque LA fonctionnalité !

L'industrie du logiciel de demain (1/2)

$\surd$ Des logiciels adaptés aux besoins :

L'industrie du logiciel de demain (1/2)

√ Des logiciels adaptés aux besoins :

- Ne se trouve que les fonctionnalités requises

L'industrie du logiciel de demain (1/2)

√ Des logiciels adaptés aux besoins :

- Ne se trouve que les fonctionnalités requises
- On dispose toujours de la dernière version

L'industrie du logiciel de demain (1/2)

√ Des logiciels adaptés aux besoins :

- ▶ Ne se trouve que les fonctionnalités requises
- ▶ On dispose toujours de la dernière version
  - • La dernière version est moins buggée que la précédente. . .

L'industrie du logiciel de demain (1/2)

√ Des logiciels adaptés aux besoins :
- ▶ Ne se trouve que les fonctionnalités requises
- ▶ On dispose toujours de la dernière version
  - • La dernière version est moins buggée que la précédente. . .
- ▶ On peut faire évoluer (reconfigurer) son logiciel

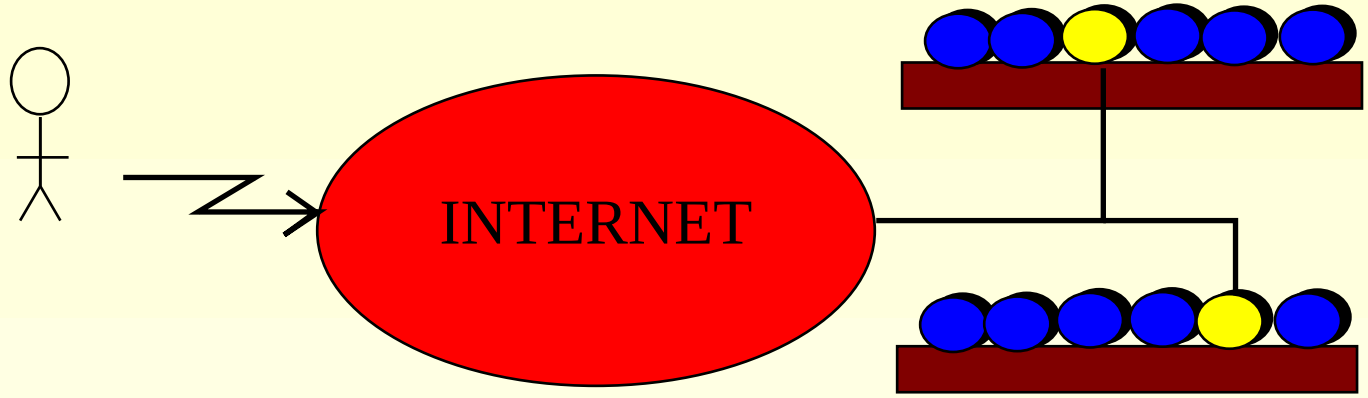
√ On ne paye que :
- ▶ Les fonctionnalités requises

L'industrie du logiciel de demain (1/2)

√ Des logiciels adaptés aux besoins :
- ▶ Ne se trouve que les fonctionnalités requises
- ▶ On dispose toujours de la dernière version
  - • La dernière version est moins buggée que la précédente. . .
- ▶ On peut faire évoluer (reconfigurer) son logiciel

√ On ne paye que :
- ▶ Les fonctionnalités requises
- ▶ La durée effective d'utilisation

L'industrie du logiciel de demain (2/2)



Via le **réseau** Internet, recherche des **composants** nécessaires, **association** de ceux-ci, puis exécution à distance.

Les Besoins technologiques

√ Vers des “composants sur l’étagère”

- Accéder à un composant (protocole réseau)

Les Besoins technologiques

√ Vers des “composants sur l’étagère”

- Accéder à un composant (protocole réseau)
- Localiser un composant (eq. DNS)

Les Besoins technologiques

√ Vers des “composants sur l’étagère”

- Accéder à un composant (protocole réseau)
- Localiser un composant (eq. DNS)
- Description d’un composant (service trader)

Les Besoins technologiques

√ Vers des “composants sur l’étagère”

- Accéder à un composant (protocole réseau)
- Localiser un composant (eq. DNS)
- Description d’un composant (service trader)
- Faire payer l’utilisation d’un composant

Les Besoins technologiques

√ Vers des “composants sur l’étagère”

- Accéder à un composant (protocole réseau)
- Localiser un composant (eq. DNS)
- Description d’un composant (service trader)
- Faire payer l’utilisation d’un composant
- Programmation : associer des composants

Des objets aux composants

√ Définition, principe et intérêt

Des objets aux composants

- √ Définition, principe et intérêt
- √ Modèle de composants Java Beans

Des objets aux composants

- √ Définition, principe et intérêt
- √ Modèle de composants Java Beans
- √ Notion d'introspection

Des objets aux composants

- √ Définition, principe et intérêt
- √ Modèle de composants Java Beans
- √ Notion d'introspection
- √ Programmation visuelle

Qu'est-ce qu'un composant ?

√ Module logiciel autonome pouvant être installé sur différentes plate-formes

Qu'est-ce qu'un composant ?

- √ Module logiciel autonome pouvant être installé sur différentes plate-formes
- √ Mécanisme d'introspection (auto-descriptif)

Qu'est-ce qu'un composant ?

- √ Module logiciel autonome pouvant être installé sur différentes plate-formes
- √ Mécanisme d'introspection (auto-descriptif)
- √ Conservation de son état

Qu'est-ce qu'un composant ?

- √ Module logiciel autonome pouvant être installé sur différentes plate-formes
- √ Mécanisme d'introspection (auto-descriptif)
- √ Conservation de son état
- √ Paramétrable ou configurable

Objectif des composants

$\surd$ A partir de plusieurs composants, construire une nouvelle application :

Objectif des composants

√ A partir de plusieurs composants, construire une nouvelle application :

- Associer/connecter des composants

Objectif des composants

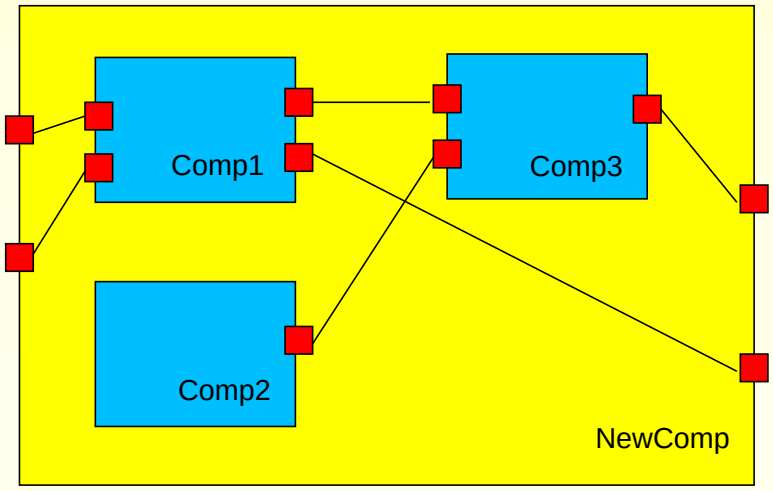
√ A partir de plusieurs composants, construire une nouvelle application :

- Associer/connecter des composants
- Proposer des outils d'aide à la composition de composants

Objectif des composants

√ A partir de plusieurs composants, construire une nouvelle application :

- Associer/connecter des composants
- Proposer des outils d'aide à la composition de composants



Les apports de la programmation par composants

√ Ne pas partir de zéro, existence de modules efficaces et robustes :

Les apports de la programmation par composants

√ Ne pas partir de zéro, existence de modules 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 modules 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

Les composants en Java : les Java Beans

`http ://java.sun.com/docs/books/tutorial/javabeans`

- √ Module logiciel autonome pouvant être installé sur différentes plate-formes (fichier jar)

Les composants en Java : les Java Beans

`http ://java.sun.com/docs/books/tutorial/javabeans`

- √ Module logiciel autonome pouvant être installé sur différentes plate-formes (fichier jar)
- √ Mécanisme d'introspection (auto-descriptif) (reflection Java)

Les composants en Java : les Java Beans

`http ://java.sun.com/docs/books/tutorial/javabeans`

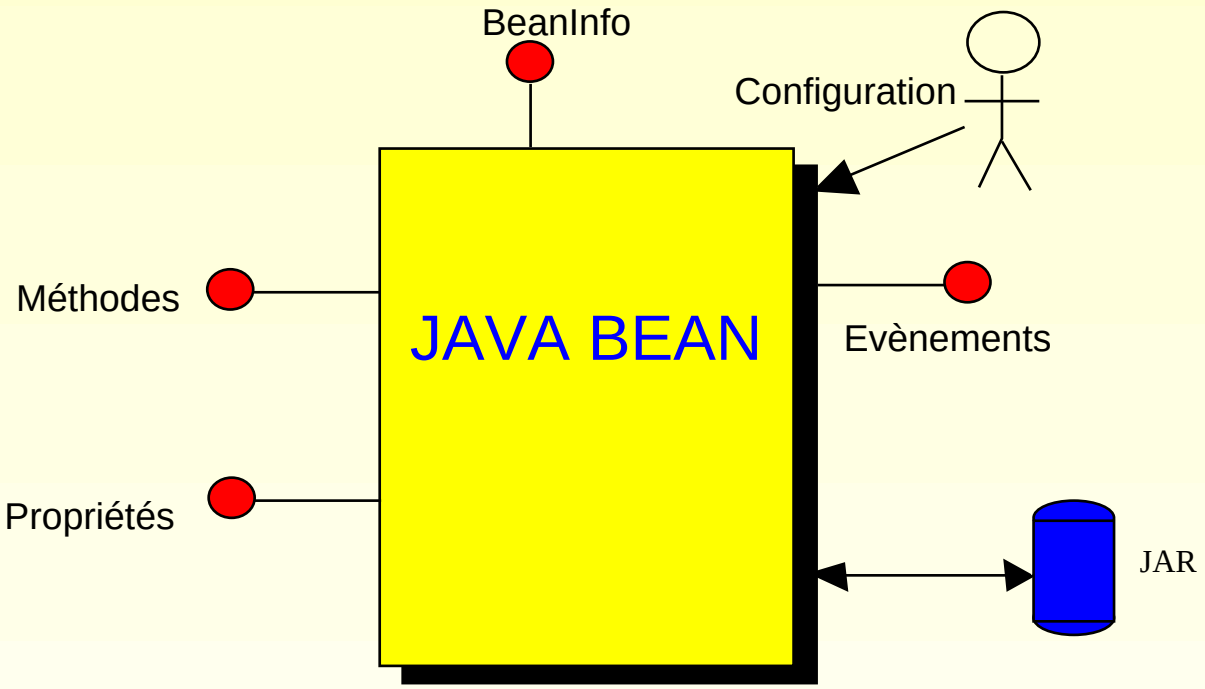
- √ Module logiciel autonome pouvant être installé sur différentes plate-formes (fichier jar)
- √ Mécanisme d'introspection (auto-descriptif) (reflection Java)
- √ Conservation de son état (sérialisation)

Les composants en Java : les Java Beans

`http ://java.sun.com/docs/books/tutorial/javabeans`

- √ Module logiciel autonome pouvant être installé sur différentes plate-formes (fichier jar)
- √ Mécanisme d'introspection (auto-descriptif) (reflection Java)
- √ Conservation de son état (sérialisation)
- √ Paramétrable ou configurable (sérialisation + réflection + évènements)

Structure d'un Java Bean



Premier Bean : SmileyBean

```
import java.awt.*;
public class SmileyBean extends Canvas {
  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 paint(Graphics g) { ...}
}
```

Premières remarques :

$\surd$ Ici, pas de constructions supplémentaires

Premières remarques :

- √ Ici, pas de constructions supplémentaires
- √ Une *règle Bean* respectée : fonction constructeur sans paramètres

Premières remarques :

- √ Ici, pas de constructions supplémentaires
- √ Une *règle Bean* respectée : fonction constructeur sans paramètres
- √ Une classe simple peut correspondre à un Java Bean.

Première application

```
import java.awt.*;
import java.awt.event.*;
public class SmileyPlace extends Frame implements WindowListener {
  public SmileyPlace(String titre) {
    SmileyBean smiley = null;
    try {
      smiley = (SmileyBean)
                java.beans.Beans.instantiate(null, "SmileyBean");
    } catch (Exception e) {
      System.err.println("Exception:"+e);
    }
    this.add(smiley); this.addWindowListener(this)
  }
 static public void main(String args[]) {...
```

La classe `java.beans.Beans`

- √ Fournit différents services
- √ La création d'un Beans ne se fait **jamais** 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 getX()
public void setX(int valeur)
```

Propriétés des composants - Conventions de nommage

√ `public int getX()`

`public void setX(int valeur)`

- Définit une propriété entière de nom x

Propriétés des composants - Conventions de nommage

√ `public int getX()`

`public void setX(int valeur)`

- Définit une propriété entière de nom x

√ `public boolean isX()`

`public void setX(boolean valeur)`

Propriétés des composants - Conventions de nommage

√ `public int getX()`

`public void setX(int valeur)`

- Définit une propriété entière de nom x

√ `public boolean isX()`

`public void setX(boolean valeur)`

- Définit une propriété booléenne de nom x

Propriétés des composants - Conventions de nommage

√ `public int getX()`
`public void setX(int valeur)`
▶ Définit une propriété entière de nom x

√ `public boolean isX()`
`public void setX(boolean valeur)`
▶ Définit une propriété booléenne de nom x

√ Possibilité de ne pas utiliser les conventions de nommage (`BeanDescriptor`)

Propriétés des composants - Conventions de nommage

√ `public int getX()`
`public void setX(int valeur)`
▶ Définit une propriété entière de nom x

√ `public boolean isX()`
`public void setX(boolean valeur)`
▶ Définit une propriété booléenne de nom x

√ Possibilité de ne pas utiliser les conventions de nommage (`BeanDescriptor`)

√ La reflexion Java permet d'analyser **dynamiquement** un composant et de le configurer dynamiquement !

Analyse dynamique et invocation dynamique (1/5)

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

public class Reflection {
  public static void main(String args[]) {
    Class laClasse ; Method lesMethodes[] ;
    Vector bools=new Vector() ;
    try {
        Object obj = Beans.instantiate(null,args[0]) ;
        laClasse = obj.getClass() ;
        lesMethodes = laClasse.getMethods() ;
        int i=0 ;
```

Analyse dynamique et invocation dynamique (2/5)

```
while (i<lesMethodes.length) {
  if (lesMethodes[i].getName().startsWith("is")&&
      lesMethodes[i].getParameterTypes().length==0 &&
      lesMethodes[i].getReturnType().toString().equals("boolean")) {
      String nomProp=lesMethodes[i].getName().substring(2) ;
      System.out.println(nomProp) ;
      bools.addElement(nomProp) ;
  }
  i++ ;
}
```

Analyse dynamique et invocation dynamique (3/5)

```
i=0 ; Object param[] =new Boolean[1] ;
param[0]=new Boolean(true) ;
while (i<lesMethodes.length) {
  if (lesMethodes[i].getName().startsWith("set") &&
    bools.contains(lesMethodes[i].getName().substring(2))) {
    System.out.println("invocation méthode") ;
    lesMethodes[i].invoke(obj,param) ;
  }
  i++ ;
}
```

Analyse dynamique et invocation dynamique (4/5)

```
} catch (ArrayIndexOutOfBoundsException e) {
    System.err.println("Erreur : java Reflection nomClasse") ;
} 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...") ; } } }
```

Analyse dynamique et invocation dynamique (5/5)

√ On ne connait pas la structure de la classe au départ !

Analyse dynamique et invocation dynamique (5/5)

- √ On ne connait pas la structure de la classe au départ !
- √ Possibilité de construire des outils paramètrables, des descripteurs de composants :
  - ▶ BeanBox (SUN)

Analyse dynamique et invocation dynamique (5/5)

√ On ne connait pas la structure de la classe au départ !

√ Possibilité de construire des outils paramètrables, des descripteurs de composants :

- BeanBox (SUN)
- JBuilder (Inprise)

Analyse dynamique et invocation dynamique (5/5)

- √ On ne connait pas la structure de la classe au départ !
- √ Possibilité de construire des outils paramètrables, des descripteurs de composants :
  - BeanBox (SUN)
  - JBuilder (Inprise)
  - Visual Cafe (Symantec),. . .
- √ Exécution de l'exemple :

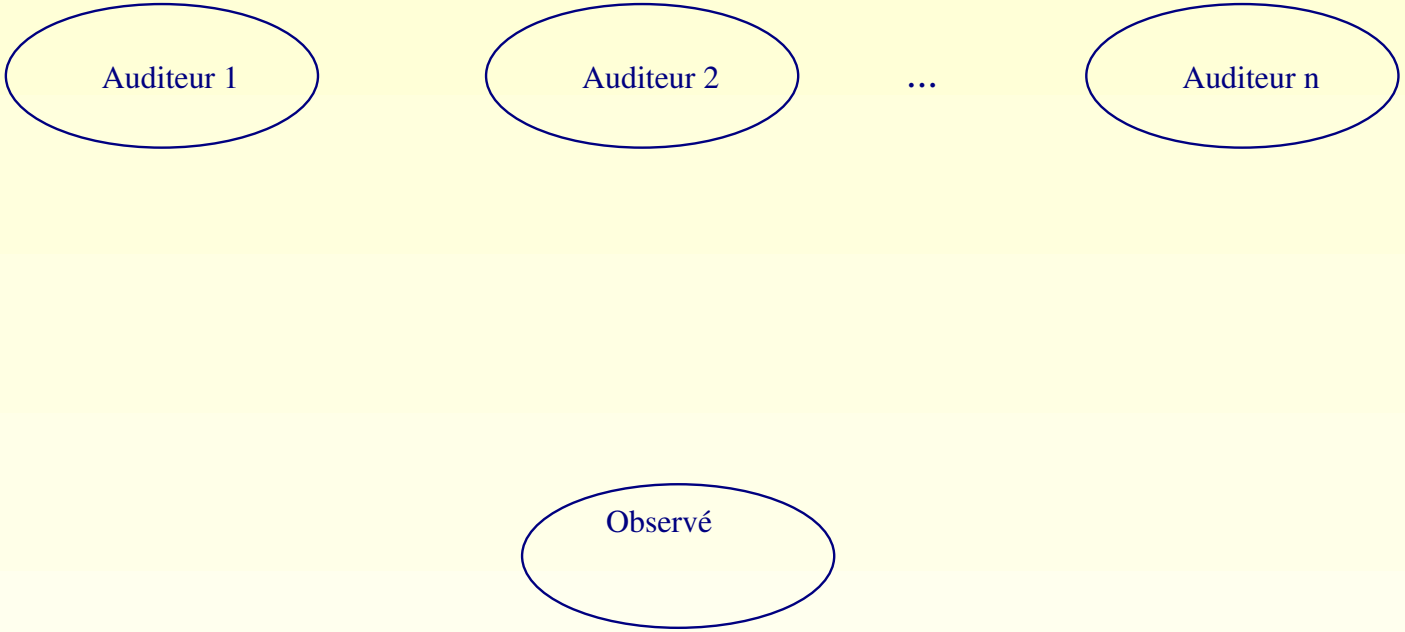
```
java Reflection java.awt.Button
Valid
Displayable
Visible ...
```

Les évènements (1/3)

√ Basé sur “Observer Design Pattern” :

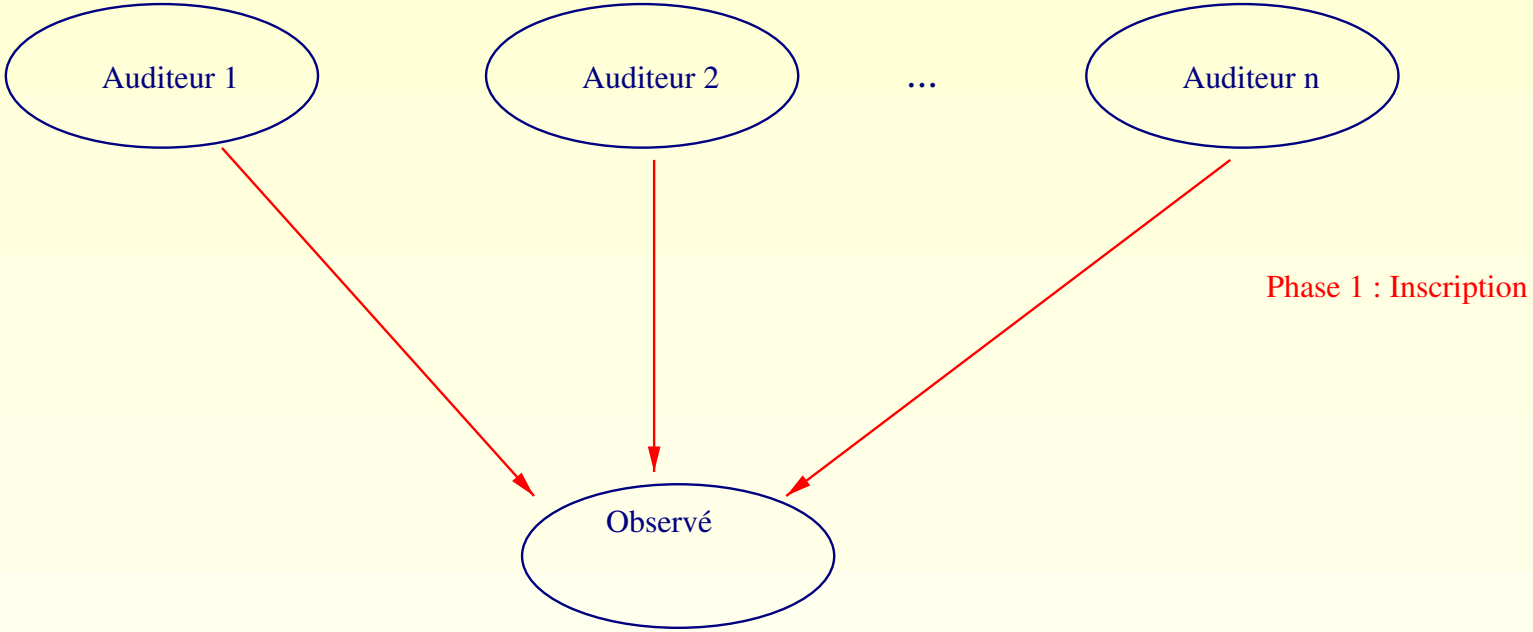
Les évènements (1/3)

√ Basé sur “Observer Design Pattern” :



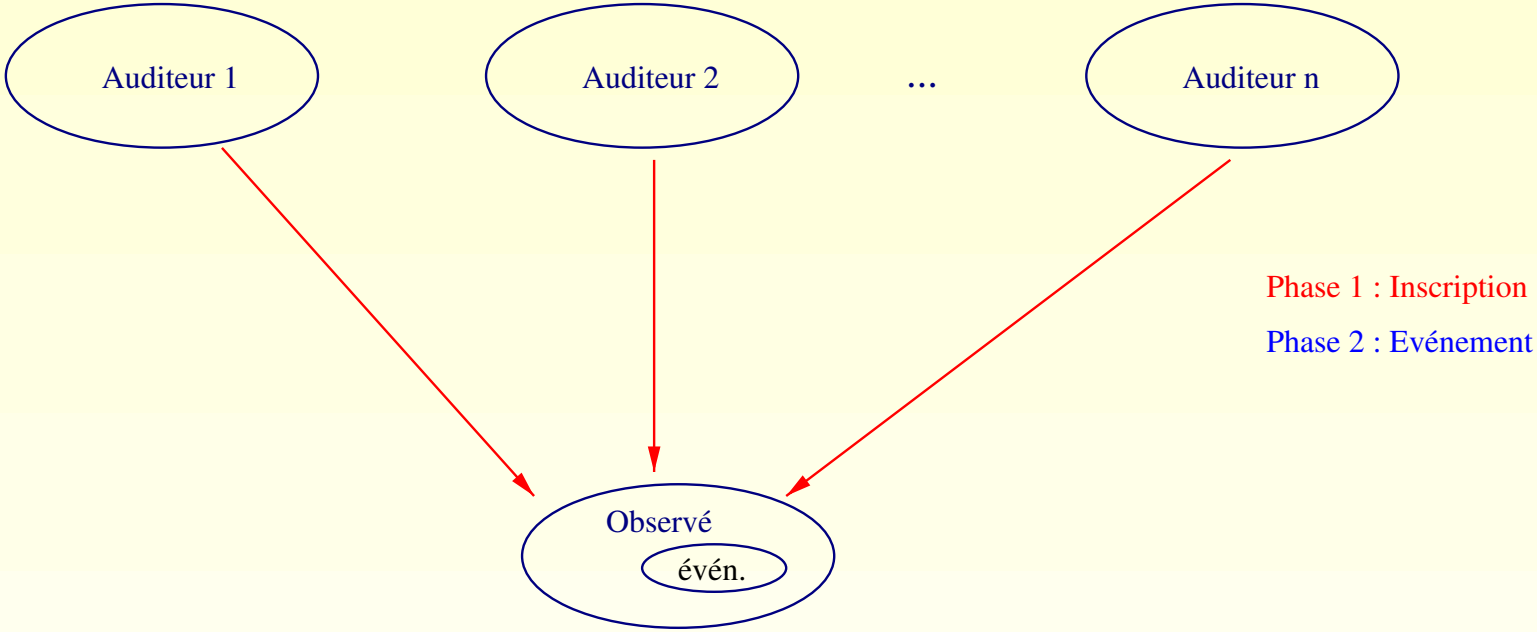
Les évènements (1/3)

√ Basé sur “Observer Design Pattern” :



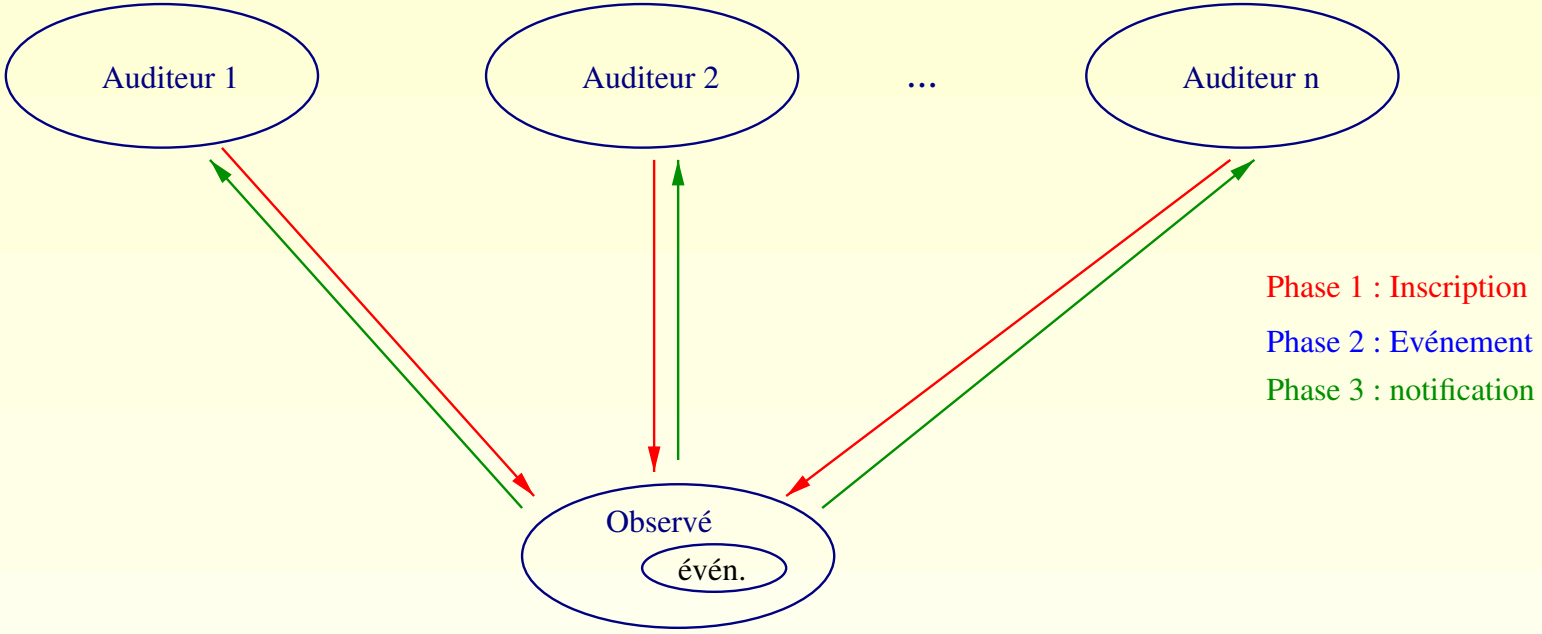
Les évènements (1/3)

√ Basé sur “Observer Design Pattern” :



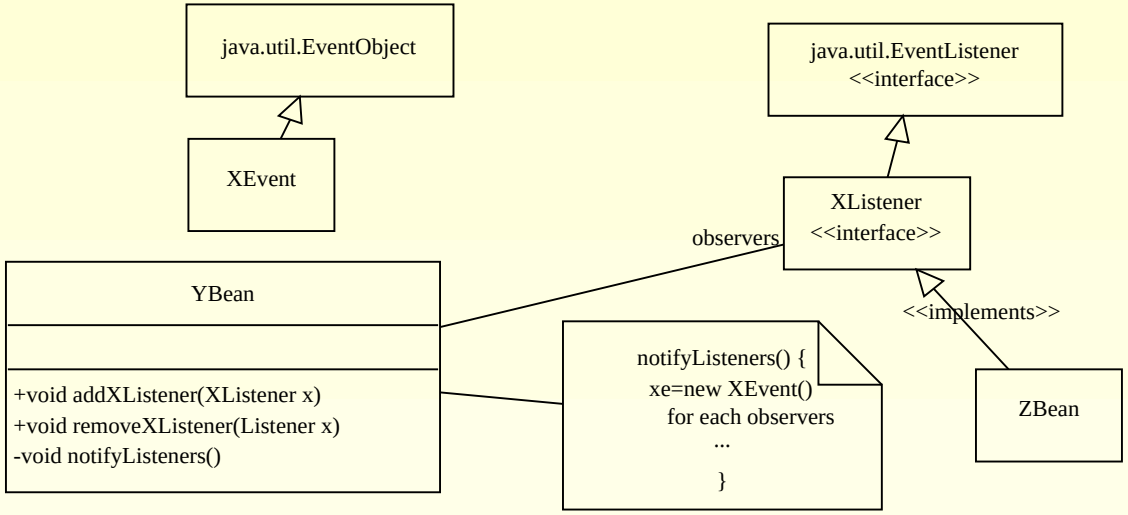
Les évènements (1/3)

√ Basé sur “Observer Design Pattern” :



Les évènements (2/3)

√ Illustration : le modèle de l'AWT



- X → Action, Y → Button

Les évènements (3/3)

√ EventObject

```
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() { ... }
}
```

√ EventListener

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

Un nouvel évènement `SourireEvent`

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

Un nouvel évènement `SourireEvent`

- √ Objectif : on veut créer un nouvel évènement qui intervient quand le 'smiley' sourit.
- √ Plusieurs auditeurs peuvent être à l'écoute de cet évènement.

Un nouvel évènement `SourireEvent`

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

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

```
import java.util.EventObject ;

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

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)

$\surd$ 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)

```
import java.awt.*;
import java.beans.*;
import java.util.ArrayList ;
public class SmileyBean extends Canvas {
  // Private data fields:
  private Color ourColor = Color.yellow;
  private boolean smile = true;
  private ArrayList  sourireListeners = new ArrayList() ;
    public SmileyBean() {
        this.setSize(250,250);
    }
```

La nouvelle classe SmileyBean (3/4)

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

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

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

La nouvelle classe SmileyBean (4/4)

```
    public void notifyListeners() {
       SourireEvent se=new SourireEvent(this) ;
       ArrayList lv =null ;
       // réalisation copie (accès concurrent)
       // ex: cas ou un addListener en action
       synchronized(this) {
         lv=(ArrayList)sourireListeners.clone() ;
       }
       for (int i=0;i<lv.size();i++)
           ((SourireListener) lv.get(i)).devientDrole(se) ;
    }
    public void paint(Graphics g){ ... }
}
```

Un composant auditeur

```
import java.awt.Label ;
public class SmileyLabelBean extends Label
  implements SourireListener {

    private int compteur=0 ;

    public SmileyLabelBean() { super("compteur:0") ; }

    public void devientDrole(SourireEvent e) {
        compteur++ ;
        this.setText("compteur:"+compteur) ;
    }
}
```

Conventions de nommage

√ Evènement :

```
class NomEvent
```

Conventions de nommage

- √ Evènement :

  `class NomEvent`

- √ Auditeur :

  `interface NomListener`

Conventions de nommage

- √ Evènement :

  `class NomEvent`

- √ Auditeur :

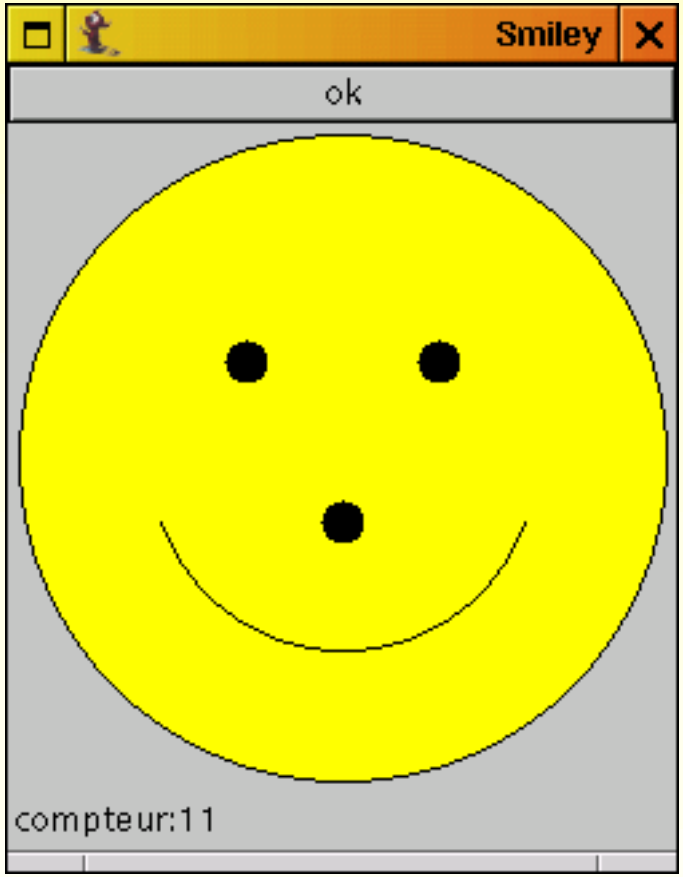
  `interface NomListener`

- √ Classe source d'évènements :

  `public void addNomListener(NomListener l)`

  `public void removeNomListener(NomListener l)`

Exemple application (1/3)



Exemple application (2/3)

```
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
public class SmileyPlace extends Frame
    implements WindowListener, ActionListener {
    private SmileyBean smiley = null;
  public SmileyPlace() {
    SmileyLabelBean label=null ;
    Button ok=new Button("ok") ;
    try {
      smiley=(SmileyBean)Beans.instantiate(null,"SmileyBean");
      label=(SmileyLabelBean)
              Beans.instantiate(null,"SmileyLabelBean");
    } catch (Exception e) { System.err.println("Exception:"+e); }
```

Exemple application (3/3)

```
  this.add("Center",smiley); this.add("South",label) ;
  this.add("North",ok) ;
  ok.addActionListener (this) ;
  smiley.addSourireListener(label) ;
  this.addWindowListener(this);
}

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`

Dans ce cas, pourquoi programmer l'application ?

La BeanBox

√ Le BDK (Beans Development Kit) (`http://www.javasoft.com`)

La BeanBox

- √ Le BDK (Beans Development Kit) (`http://www.javasoft.com`)
- √ Charge dynamiquement des composants Beans
  fichiers jar normalisés

La BeanBox

- √ Le BDK (Beans Development Kit) (`http://www.javasoft.com`)
- √ Charge dynamiquement des composants Beans
  fichiers jar normalisés
- √ Outil visuel de programmation

Structure physique du composant (1/2)

- √ fichier archive (`jar`) contenant :
  - classes Java (`*.class`)

Structure physique du composant (1/2)

√ fichier archive (`jar`) contenant :

- classes Java (`*.class`)
- objet sérialisé (`*.ser`)

Structure physique du composant (1/2)

√ fichier archive (`jar`) contenant :

- classes Java (`*.class`)
- objet sérialisé (`*.ser`)
- fichier descriptif (`Manifest.MF`), sa structure :

```
Name : <name>
<attribute> : <value>
<attribute> : <value>
...
Name : <name>
...
```

Structure physique du composant (2/2)

√ Fichier `smiley.mf` :

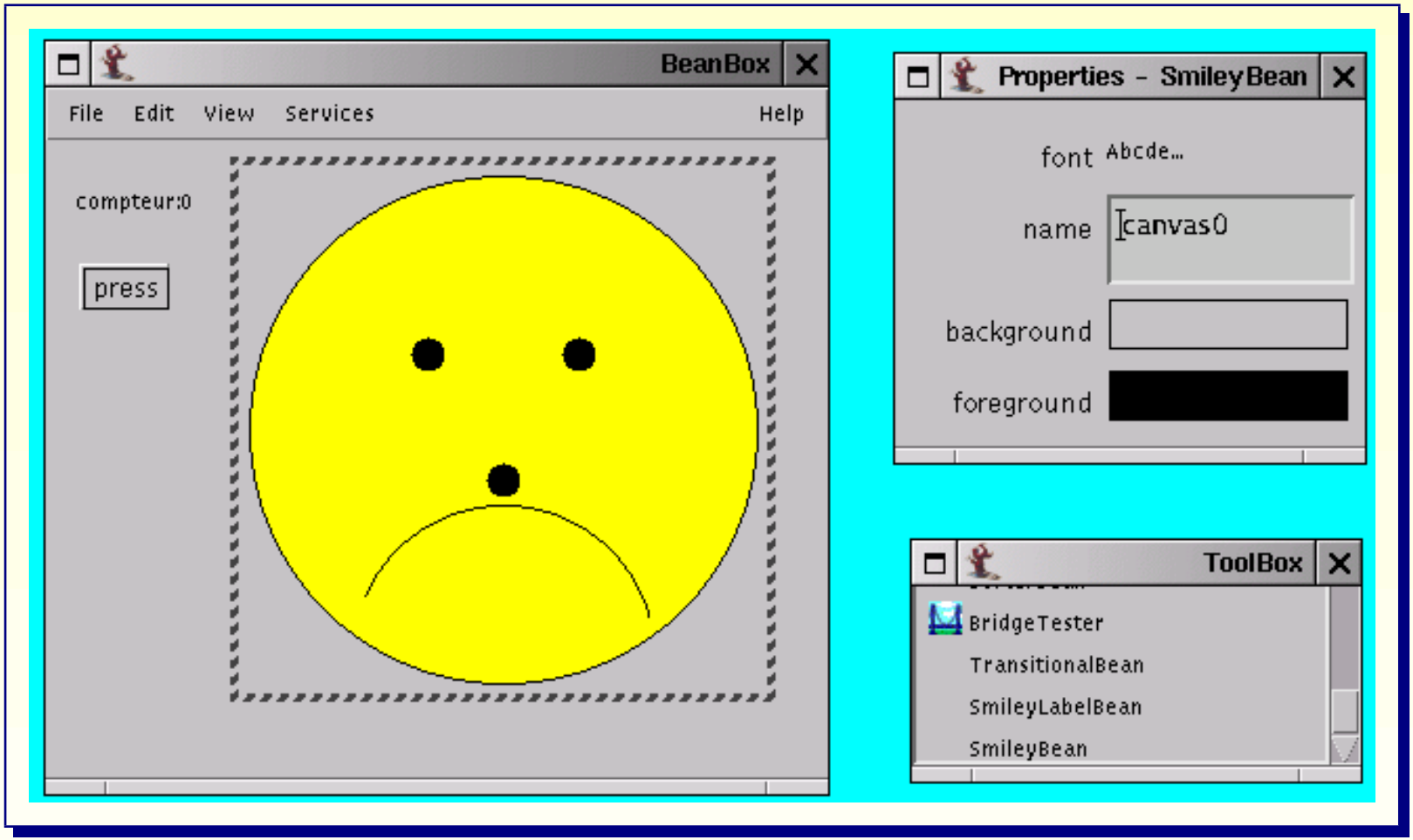
```
Manifest-Version: 1.0
Name: SmileyBean.class
Java-Bean: True

Name: SmileyLabelBean.class
Java-Bean: True

Name: SourireEvent.class
Java-Bean: False...
```

√ `jar cvfm smiley.jar smiley.mf *.class`

La BeanBox en action



La sérialisation Java (1/2)

√ Mécanisme permettant de rendre persistant des objets Java :

- implémenter l'interface `java.io.Serializable`

La sérialisation Java (1/2)

√ Mécanisme permettant de rendre persistant des objets Java :

- implémenter l'interface `java.io.Serializable`
- hériter d'une classe *sérialisable*

La sérialisation Java (1/2)

- √ Mécanisme permettant de rendre persistant des objets Java :
  - ▶ implémenter l'interface `java.io.Serializable`
  - ▶ 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 récuperer le fichier .ser
  smiley=(SmileyBean) Beans.instantiate(null,"Smiley1") ;
} catch(Exception e) {
  try {
    smiley=(SmileyBean) Beans.instantiate(null,"SmileyBean");
  } catch(...
```

√ Clause java `transient`

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

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.

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

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)
```

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; }
```

Premier exemple de propriété liée (1/3)

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

public class TemperatureBean extends java.awt.Label {

    private double temperature=20.0 ;
    private ArrayList  listeners = new ArrayList() ;

    public TemperatureBean() {
      this.setText(""+temperature) ;
    }
```

Premier exemple de propriété liée (2/3)

```
synchronized public void addPropertyChangeListener
                          (PropertyChangeListener l) {
  listeners.add(l) ;
}
synchronized public void removePropertyChangeListener(
                          PropertyChangeListener l) {
  listeners.remove(l) ;
}
public void notifyListeners(PropertyChangeEvent event) {
  ArrayList lv =null ;
  synchronized(this) { lv=(ArrayList) listeners.clone() ;}
  for (int i=0;i<lv.size();i++)
    ((PropertyChangeListener) lv.get(i)).propertyChange(event) ;
  }
```

Premier exemple de propriété liée (3/3)

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

Programmation propriété liée

√ Relativement lourd à programmer (notifyListener,. . .)

Programmation propriété liée

√ Relativement lourd à programmer (notifyListener,. . .)

√ Toujours le même code

Programmation propriété liée

- √ Relativement lourd à programmer (notifyListener,. . .)
- √ Toujours le même code
- √ Existence de classe utilitaire dans le JDK :
  `java.beans.PropertyChangeSupport`

Second exemple de propriété liée (1/2)

```
public class TemperatureBean2 extends java.awt.Label {
  private double temperature=20.0 ;
  private java.beans.PropertyChangeSupport pcs ;
  public TemperatureBean2() {
    this.setText(""+temperature) ;
    pcs=new java.beans.PropertyChangeSupport(this) ;
  }
  synchronized
  public void addPropertyChangeListener(PropertyChangeListener l) {
    pcs.addPropertyChangeListener(l) ;
  }
  synchronized
    public void removePropertyChangeListener(PropertyChangeListener l) {
    pcs.removePropertyChangeListener(l) ; }
```

Second exemple de propriété liée (2/2)

```
  public synchronized void setTemperature (double v)  {
    double old=this.temperature ;
    this.temperature=v ;
    PropertyChangeEvent event ;
    event=new PropertyChangeEvent(this, "temperature",
                        new Double(old),new Double(temperature)) ;
    pcs.firePropertyChange(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.

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 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
```

- √ Classe utilitaire : `java.beans.VetoableChangeSupport`

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