JavaBeans

Les spécifications sur les JavaBeans recommandent certains principes lors de développement :

- Le respect de l'encapsulation, une des règles des langages orientés objets.
  L'accès des champs internes des objets ne peut pas se faire directement vers les champs eux-mêmes (déclarés en *private*), mais par le biais de méthodes (*public*).

  Les *Accessor* (méthodes *get, set, is* utilisant le nom du champ) permettent les lectures/écritures des valeurs au sein des objets :

```
public class simpleObject
{
 //déclaration des champs privés

 private String      description;
 private int         values[];
 private boolean     flagOK;

 //constructeurs

 public simpleObject() {}
 public simpleObject(String description)
 {
  setDescription(description);
 }

 //Accessor de champ simple

 public String getDescription()
 {
  return description;
 }
 public void setDescription(String newDesc)
 {
  description = newDesc;
 }

 //Accessor de champ booléen

 public boolean isFlagOk()
 {
  return flagOK;
 }
 public void setFlagOK(boolean newFlagOK)
 {
  flagOK = newFlagOK;
 }

 //Accessor de champs indexés

 public int[] getValues()
 {
  return values;
 }
 public void setValues(int[] newValues)
 {
  values = newValues;
 }

 public int getValues(int index)
 {
  return values[index];
 }
 public void setValues(int newValue, int index)
 {
  values[index] = newValue;
 }
}
```


CC 1999-2012
Droits réservés

http://creativecommons.org/licenses/by-nc-nd/2.0/fr/

- L'application de cette norme permet d'utiliser des mécanismes d'*introspection*.
  Ces processus puissants donnent un accès direct aux membres public des classes :
  Définition, constructeurs, méthodes, champs et droits.

  Exemples d'utilisations usuelles de l'API de réflexivité (package *java.lang.reflect*) :

```
try
{
  //Accès aux ressources de classes :

  Class objectClass        = Class.forName(strClassName);
  // Class objectClass     = myObject.getClass(); //accès possible depuis n'importe quel objet
  // Class objectClass     = java.lang.String.class; //accès static standard

  Field objectField         = objectClass.getField(strFieldName); //ce champ doit être déclaré public

  Class tabParameterTypes[] = {};  //la méthode doit être déclarée public, ici une méthode getX()
  Method getMethod        = objectClass.getMethod(strMethodName, tabParameterTypes);

  //recherche d'un constructeur basique, sans argument.
  Constructor classConstructor = objectClass.getDeclaredConstructor(tabClassParameterType);


  //Utilisation des ressources:

  //lecture d'un champ, nécessite un objet cible dans lequel lire le champ
  Object fieldValue         = objectField.get(objectTarget);

  //affectation d'un champ
  objectField.set(objectTarget, value);

  //invocation d'une méthode get, la méthode ne prend pas de paramètre en entrée
  Object tabObject[]        = {};
  Object objectResult      = getMethod.invoke(objectTarget, tabObject);

  //invocation du constructeur pour instancier un objet à la volée.
  Object objectCreated     = classConstructor.newInstance(tabObject);
}
catch (Exception e) {System.out.println(e.toString());}
```

- Les spécifications de JavaBeans mettent aussi en exergue la *persistance* des objets.

Elle permet d'utiliser par des médias physiques des objets, par exemple stocker sur le File System des objets, ou bien les envoyer à travers le réseau par des Sockets, ou bien selon des protocoles gérant la partie transmission (CORBA, RMI, par exemple).

La persistance passe par le biais de l'interface *java.io.Serializable*, qui permet à la machine virtuelle d'exploiter l'objet en tant que données brutes. On peut, par exemple, sérialiser vers des fichiers, des données de Session d'un client Web, puis les recharger sous leur forme objet en les relisant lors de la reprise de Session de ce client.

CC 1999-2012
Droits réservés

http://creativecommons.org/licenses/by-nc-nd/2.0/fr/

- Les JavaBeans fournissent une classe permettant le traitement d'événements génériques selon le modèle bien connu « producteur-consommateur ». Ce concept permet d'automatiser la communication entre objets au sein d'une application.

```
import java.util.EventListener;

public interface UpdateListener extends EventListener
{
  public void updateInfo(UpdateEvent e)
}
```

```
public class UpdateEvent
{
  private String newDescription;

  public updateEvent(String newDescription) {setNewDescription(newDescription);}

  public String getNewDescription()  {return newDescription;}
  public void setDescription(String newDescription) {this.newDescription = newDescription;}
}
```

```
public class myBean implements UpdateListener
{
  private String      description;

  public String getDescription()   {return description;}
  public void setDescription(String newDesc) {description = newDesc;}

  public void updateInfo(UpdateEvent e)
  {
    setDescription(e.getNewDescription());
  }
}
```

```
import java.util.Vector;

public beanRegister
{
  private Vector vectorListeners = new Vector();

  public void addUpdateListener(UpdateListener ul)   {vectorListeners.add(ul);}
  public void removeUpdateListener(UpdateListener ul)   {vectorListeners.remove(ul);}

  public void overallUpdate(myBean sourceBean)
  {
    UpdateEvent e = new UpdateEvent(sourceBean.getDescription);

    for (int i = 0; i != vectorListeners.size(); i++)
      ((UpdateListener) vectorListeners.elementAt(i)).updateInfo(e);
  }
}
```


CC 1999-2012
Droits réservés

http://creativecommons.org/licenses/by-nc-nd/2.0/fr/

- Les JavaBeans définissent un format de fichier de déploiement, les *.jar* (Java Archives), qui permettent d'empaqueter les classes et ressources d'applications (fichiers multimédia, descripteur de déploiement XML, descripteur du jar avec possibilité de signer à l'aide de certificats X509 le contenu pour en sécuriser la trans9ission sur Internet.

CC 1999-2012
Droits réservés

http://creativecommons.org/licenses/by-nc-nd/2.0/fr/