1. Objectif

Cet atelier présente la mise en place des Java beans avec leurs différents types de propriétés. Il existe quatre types de propriétés :

- les propriétés simples
- les propriétés indexées (indexed properties)
- les propriétés liées (bound properties)
- les propriétés liées avec contraintes (Constrained properties)

2. Présentation des Java beans

Les Java beans sont des composants réutilisables et autonomes qui doivent pouvoir être facilement assemblés entres eux pour créer un programme.

La technologie JavaBeans propose de simplifier et faciliter la création et l'utilisation de composants. Les JavaBeans possèdent plusieurs caractéristiques :

- la persistance : elle permet grâce au mécanisme de sérialisation de sauvegarder l'état d'un bean pour le restaurer. Ainsi si on assemble plusieurs beans pour former une application, on peut la sauvegarder.
- la communication, grâce à des événements, qui utilise le modèle des écouteurs introduit par Java 1.1
- l'introspection : ce mécanisme permet de découvrir de façon dynamique l'ensemble des éléments qui composent le bean (attributs, méthodes et événements) sans avoir le code source.
- la possibilité de paramétrer le composant : les données du paramétrage sont conservées dans des propriétés.

Ainsi, les beans sont des classes Java qui doivent respecter un certain nombre de règles :

1. ils doivent posséder un constructeur sans paramètre. Celui-ci devra initialiser l'état du bean avec des valeurs par défauts.
2. ils peuvent définir des propriétés : celles-ci sont identifiées par des méthodes dont le nom et la signature sont normalisés
3. ils devraient implémenter l'interface serialisable : ceci est obligatoire pour les beans qui possèdent une partie graphique pour permettre la sauvegarde de leur état
4. ils définissent des méthodes utilisables par les composants extérieurs : elles doivent être public et prévoir une gestion des accès concurrents
5. ils peuvent émettre des événements en gérant une liste d'écouteurs qui s'y abonnent grâce à des méthodes dont les noms sont normalisés

3. Les propriétés simples

```
package demo.simple;

public class MyBean {

	private String strProperty;
	private boolean flg;

	public String getStrProperty() {
		return strProperty;
	}
	public void setStrProperty(String strProperty) {
		this.strProperty = strProperty;
	}
	public boolean isFlg() {
		return flg;
	}
	public void setFlg(boolean flg) {
		this.flg = flg;
	}

}
```

Exemple

```
package demo.simple;

import java.awt.*;
import java.io.Serializable;

public class SimpleBean extends Canvas
                        implements Serializable{

    private Color color = Color.green;

    //property getter method
    public Color getColor(){
        return color;
    }

    //property setter method. Sets new SimpleBean
    //color and repaints.
    public void setColor(Color newColor){
        color = newColor;
        repaint();
    }

    public void paint(Graphics g) {
        g.setColor(color);
        g.fillRect(20, 5, 20, 30);
    }

    //Constructor sets inherited properties
    public SimpleBean(){
        setSize(60,40);
        setBackground(Color.red);
    }
}
```

4. Les propriétés indexées

```
package demo.simple;

public class IndexedBean {

	private String[] elements;

	public String[] getElements() {
		return elements;
	}

	public void setElements(String[] elements) {
		this.elements = elements;
	}

	public String getElements(int index) {
		return elements[index];
	}

	public void setElements(int index, String elements) {
		this.elements[index] = elements;
	}

}
```

Exemple

```
package demo.simple;

public class GestionGrades {

	private String[] TestGrades;

	public String[] getTestGrades() {
		return TestGrades;
	}

	public void setTestGrades(String[] TestGrades) {
		this.TestGrades = TestGrades;
	}

	public String getTestGrades(int index) {
		return TestGrades[index];
	}

	public void setTestGrades(int index, String Grade) {
		this.TestGrades[index] = Grade;
	}

}
```

5. Les propriétés liées

```
package beans;

import java.io.Serializable;
import java.beans.*;

public class MonBean implements Serializable {
  protected int valeur;

  PropertyChangeSupport changeSupport;

  public MonBean(){
    valeur = 0;

    changeSupport = new PropertyChangeSupport(this);
  }

  public synchronized void setValeur(int val) {
    int oldValeur = valeur;
    valeur = val;

    changeSupport.firePropertyChange("valeur",oldValeur,valeur);
  }
  public synchronized int getValeur() {
    return valeur;
  }
  public synchronized void addPropertyChangeListener(PropertyChangeListener listener) {
    changeSupport.addPropertyChangeListener(listener);
  }

  public synchronized void removePropertyChangeListener(PropertyChangeListener listener) {
    changeSupport.removePropertyChangeListener(listener);
  }
}
```

```
package beans;

import java.beans.*;
import java.util.*;

public class TestMonBean {
  public static void main(String[] args) {
    new TestMonBean();
  }

  public TestMonBean() {
    MonBean monBean = new MonBean();

    monBean.addPropertyChangeListener( new PropertyChangeListener() {
      public void propertyChange(PropertyChangeEvent event) {
        System.out.println("propertyChange : valeur = "+ event.getNewValue());
      }
    } );

    System.out.println("valeur = " + monBean.getValeur());
    monBean.setValeur(10);
    System.out.println("valeur = " + monBean.getValeur());

  }
}
```

Sortie

```
<terminated> TestMonBean [Java Application] C:\Users\jarra\.p2\pool\plugins\org.
valeur = 0
propertyChange : valeur = 10
valeur = 10
```

6. Les propriétés liées avec contraintes

```
package beans;

import java.io.Serializable;
import java.beans.*;

public class MonBean implements Serializable {
          protected int oldValeur;
          protected int valeur;

          PropertyChangeSupport changeSupport;
          VetoableChangeSupport vetoableSupport;

          public MonBean(){
            valeur = 0;
            oldValeur = 0;

            changeSupport = new PropertyChangeSupport(this);
            vetoableSupport = new VetoableChangeSupport(this);
          }


          public synchronized void setValeur(int val) {
            oldValeur = valeur;
            valeur = val;

            try {
              vetoableSupport.fireVetoableChange("valeur",new Integer(oldValeur),
                      new Integer(valeur));
            } catch(PropertyVetoException e) {
              System.out.println("MonBean, un veto est emis :  "+e.getMessage());
              valeur = oldValeur;
            }
            if ( valeur != oldValeur ) {
              changeSupport.firePropertyChange("valeur",oldValeur,valeur);
            }

          }
          public synchronized int getValeur() {
            return valeur;
          }
          public synchronized void addPropertyChangeListener(PropertyChangeListener listener) {
            changeSupport.addPropertyChangeListener(listener);
          }

          public synchronized void removePropertyChangeListener(PropertyChangeListener listener) {
            changeSupport.removePropertyChangeListener(listener);
          }
```

```
        public synchronized void addVetoableChangeListener(VetoableChangeListener listener) {
          vetoableSupport.addVetoableChangeListener(listener);
        }
        public synchronized void removeVetoableChangeListener(VetoableChangeListener listener) {
          vetoableSupport.removeVetoableChangeListener(listener);
        }

      }
```

```
package beans;

import java.beans.*;
import java.util.*;

public class TestMonBean {
  public static void main(String[] args) {
    new TestMonBean();
  }

  public TestMonBean() {
    MonBean monBean = new MonBean();

    monBean.addPropertyChangeListener( new PropertyChangeListener() {
      public void propertyChange(PropertyChangeEvent event) {
        System.out.println("propertyChange : valeur = "+ event.getNewValue());
      }
    } );

    monBean.addVetoableChangeListener( new VetoableChangeListener() {

      public void vetoableChange(PropertyChangeEvent event) throws PropertyVetoException {
        System.out.println("vetoableChange : valeur = " + event.getNewValue());
        if( ((Integer)event.getNewValue()).intValue() > 100 )
          throw new PropertyVetoException("valeur superieure a 100",event);
      }
    } );

    System.out.println("valeur = " + monBean.getValeur());
    monBean.setValeur(10);
    System.out.println("valeur = " + monBean.getValeur());
    monBean.setValeur(200);
    System.out.println("valeur = " + monBean.getValeur());

  }
}
```

Sortie

```
<terminated> TestMonBean (1) [Java Application] C:\Users\jarra\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.7.v2023042
valeur = 0
vetoableChange : valeur = 10
propertyChange : valeur = 10
valeur = 10
vetoableChange : valeur = 200
MonBean, un veto est emis :  valeur superieure a 100
valeur = 10
```

7. Introspection d'un java bean

La classe BeanInfo contient des informations sur un bean et possède plusieurs méthodes pour les obtenir.

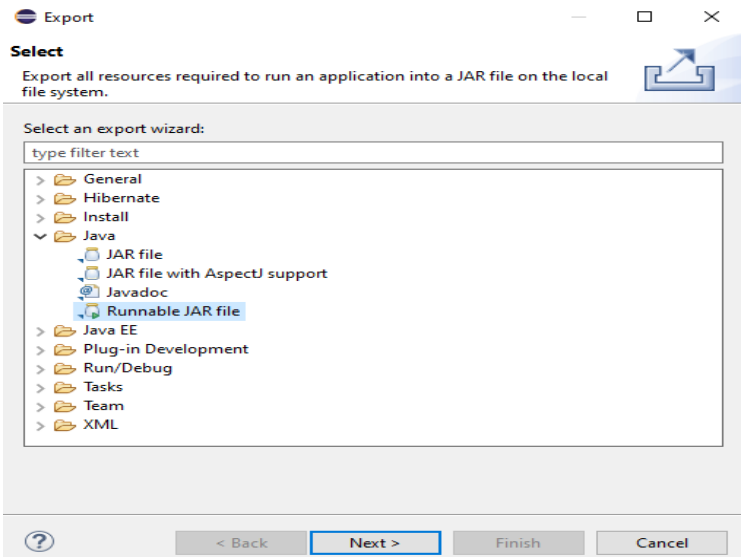
- La méthode **getBeanInfo()** prend en paramètre un objet de type Class qui représente la classe du bean et elle renvoie des informations sur la classe et toutes ses classes mères.
- La méthode **getBeanDescriptor()** permet d'obtenir des informations générales sur le bean en renvoyant un objet de type BeanDescriptor()
- La méthode **getPropertyDescriptors()** permet d'obtenir un tableau d'objets de type PropertyDescriptor qui contiennent les caractéristiques d'une propriété. Plusieurs méthodes permettent d'obtenir ces informations.
- La méthode **getMethodDescriptors()** permet d'obtenir un tableau d'objets de type MethodDescriptor. Cette classe fournit plusieurs méthodes pour extraire les informations des objets contenus dans le tableau.
- La méthode **getEventSetDescriptors()** permet d'obtenir un tableau d'objets de type EventSetDescriptor qui contient les caractéristiques d'un événement. Plusieurs méthodes permettent d'obtenir ces informations.

```
BeanInfo bi = Introspector.getBeanInfo(MonBean.class);
    BeanDescriptor unBeanDescriptor = bi.getBeanDescriptor();
    System.out.println("Nom du bean : "+unBeanDescriptor.getName());

    System.out.println("Classe du bean : "+unBeanDescriptor.getBeanClass());

    //*******************1 ère méthode************************
    System.out.println("1ere methode");
    PropertyDescriptor[] propertyDescriptor = bi.getPropertyDescriptors();
    for (int i=0; i<propertyDescriptor.length; i++) {
      System.out.println(" Nom propriete    : " +
                         propertyDescriptor[i].getName());
      System.out.println(" Type propriete   : "
                + propertyDescriptor[i].getPropertyType());
      System.out.println(" Getter propriete : "
                 + propertyDescriptor[i].getReadMethod());
      System.out.println(" Setter propriete : "
                + propertyDescriptor[i].getWriteMethod());
    }

  //*******************2 ème méthode************************
    System.out.println("2eme methode");
    MethodDescriptor[] methodDescriptor = bi.getMethodDescriptors();
    for (int i=0; i < methodDescriptor.length; i++) {
      System.out.println(" Methode : "+methodDescriptor[i].getName());
    }

   //*******************3 ème méthode************************
    System.out.println("3eme methode");
    EventSetDescriptor[] unEventSetDescriptor = bi.getEventSetDescriptors();
    for (int i = 0; i < unEventSetDescriptor.length; i++) {
```

```
        System.out.println(" Nom evt             : "
                           + unEventSetDescriptor[i].getName());
        System.out.println(" Methode add evt     : " +
                unEventSetDescriptor[i].getAddListenerMethod());
        System.out.println(" Methode remove evt  : " +
            unEventSetDescriptor[i].getRemoveListenerMethod());
        methodDescriptor = unEventSetDescriptor[i].getListenerMethodDescriptors();
        for (int j = 0; j < methodDescriptor.length; j++) {
            System.out.println(" Event Type: " + methodDescriptor[j].getName());
        }
    }
```

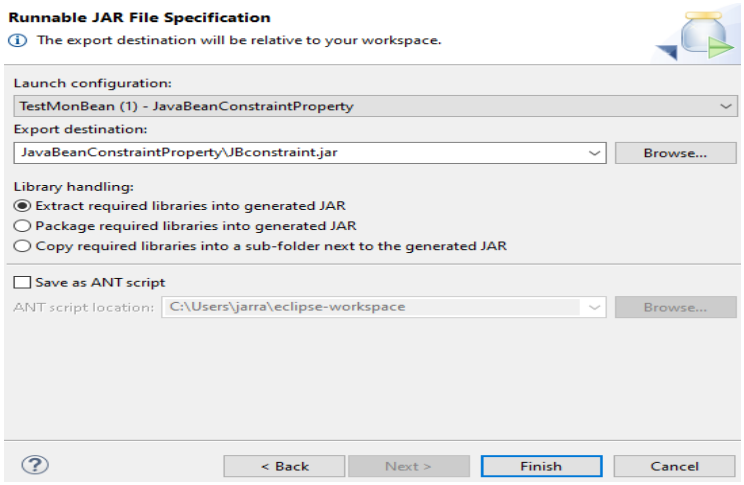
8. Diffusion d'un java bean

Pour générer le fichier jar correspondant à votre bean, cliquez droit sur votre projet -> Export -> Java->Runnable JAR file puis cliquez Next :

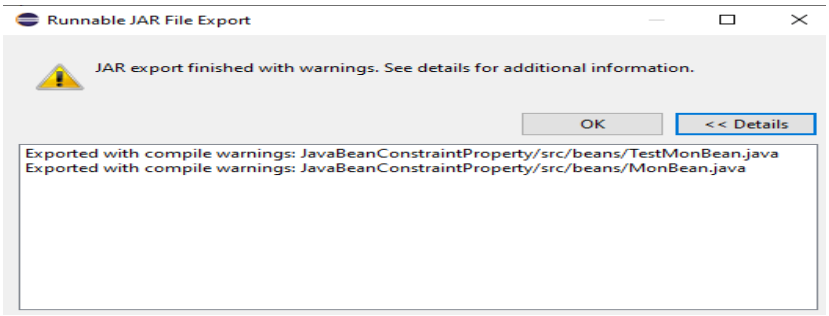


Remplissez les champs suivants :

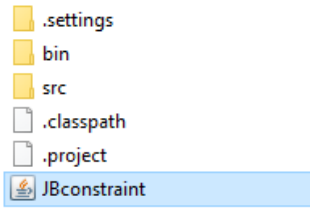
- Launch configuration : sélectionnez votre classe contenant la fonction main
- Export destination : choisissez l'emplacement et le nom pour le fichier jar à générer



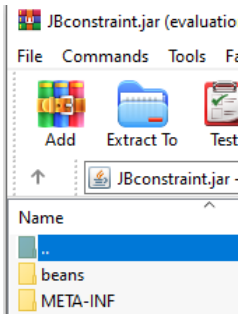
Cliquez sur Next. Vous obtenez le message suivant. Cliquez sur OK.



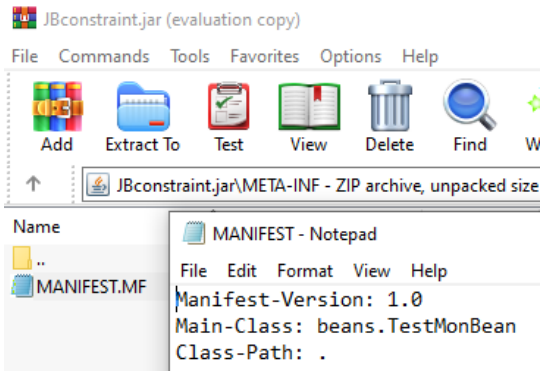
Le fichier jar « JBconstraint » est bien généré :



Pour explorer le fichier MANIFEST.MF, ouvrez le fichier jar avec un winrar comme suit :



Cliquez deux fois sur le dossier META-INF, vous trouverez le fichier MANIFEST.MF :



9. Exercice

1. Créer un « CompteurBean » possédant les propriétés suivantes :
   - int **valeur**
   - String **nom**

   La propriété **valeur** est liée à un afficheur. Ce dernier doit mettre à jour son affichage pour tout changement de cette propriété.

   Ce bean permet d'exécuter les deux opérations d'incrémentation et de décrémentation sur sa **valeur**.

   Ecrire la classe « CompteurBeanTest » permettant de tester le composant « CompteurBean ».
2. Rajouter des contraintes de sorte que l'incrémentation et la décrémentation ne doivent pas dépasser une certaine limite.
   Ecrire les composants « CompteurBean2 » et « CompteurBeanTest 2».
3. Proposer une méthode qui prend le nom du Bean et qui affiche à l'écran le nom de toutes les méthodes de ce bean.
4. Générer le .Jar exécutable de votre projet pour le diffuser.