3.1 Introduction

L'un des types de composants qu'on peut concevoir avec la technologie Java grâce à son standard de développement Java EE est le JavaBean. Dans ce chapitre, nous allons étudier les concepts principaux qui nous amènent à bien comprendre ce type de composant.

3.2 Java EE

Java EE (Java Enterprise Edition) est une extension de la platform Java SE (Java Standard Edition) qui fournit des bibliothèques (APIs) et des spécifications qui assurent le développement des applications basée sur des composants centrées sur un serveur d'applications.

Java EE facilite aux développeurs web la création des applications qui sont robustes et évolutives et qui peuvent être distribuées et exécutées facilement sur un serveur d'applications.

Des services, au travers d'API, extensions Java permettant d'offrir en standard un certain nombre de fonctionnalités.

Sous forme d'APIs, Java EE offre une multitude de fonctionnalités à travers plusieurs composants. Ses derniers peuvent se répartir en deux grandes catégories.

I. **Les composants web :** Servlet, Portlet (extension d'une Servlet), Java Server Pages (JSP), Java Standard Tag Library (JSTL), Java Server Faces (JSF), etc.
II. **Les composants métiers :** Les EJBs (Entreprise Java Beans) qui ont des connexions avec d'autres APIs comme JDBC, JNDI, etc.

Dans le standard Java EE, les développeurs trouvent les outils (les composants web et d'autres APIs) qui leur permettent de se concentrer sur l'implémentation de des fonctions métier (business logic) de leurs l'applications.

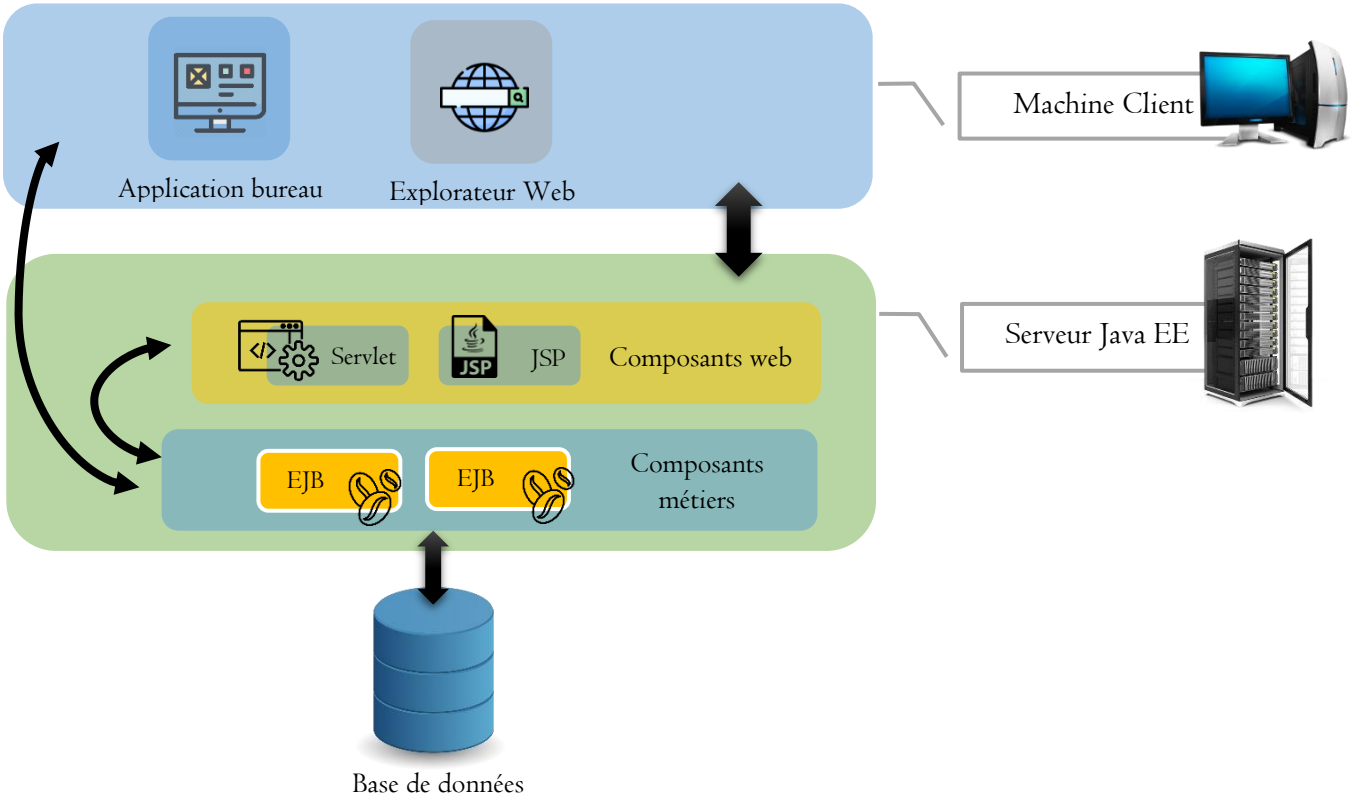


Fig.I : Architecture de Java EE

3.3 Le JavaBean

On désigne par un JavaBean ou simplement Bean un composant qu'on écrit en java pour qu'il soit réutilisable. Simplement, le Bean est un objet java conçu pour représenter une entité réutilisable.

Selon la spécification d'Oracle[1], un JavaBean (composant de type Java) est "un composant logiciel réutilisable manipulable visuellement dans un outil de conception".

Comme tout composant logiciel, un Bean est créé pour interagir avec d'autres Beans au sein d'une application qui a été créé à travers l'assemblage de ces Beans pour assurer un besoin métier. En d'autres termes, un Bean doit être créé de telle sorte qu'il soit facilement assemblé avec d'autres Beans afin de créer une application.

[1] https://download.oracle.com/otndocs/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/

Heureusement, la connaissance des bases de Java nous suffit largement pour créer les Beans sans souci.

Mais, néanmoins, la création des Beans nous requiert de respecter un ensemble de conventions à savoir :

a. **La réutilisation :** c'est l'objectif principale derrière la création d'un Bean.
b. **La sérialisation :** un Bean est créé pour qu'il soit Sérialisable. La sérialisation est le mécanisme qui permet au Bean de sauvegarder et restaurer ultérieurement son état. Ce processus permet ce qu'on appelle la persistance du Bean.
   Pour assurer cette fonctionnalité, la classe qui représente le Bean doit implémenter l'interface qui se nomme **Serializable**.

```
4   import java.io.Serializable;
5
6   public class Beans extends JPanel implements Serializable{
```

Fig2 : Le Bean implémente l'interface Serializable

c. **L'introspection :** C'est le mécanisme qui donne la possibilité au Bean d'être paramétré dynamiquement. Grace à cette fonctionnalité, les éléments du Bean (attributs, méthodes et événements) peuvent être connus de façon dynamique sans avoir à posséder le code source du Bean.
d. **Les propriétés :** elles constituent l'ensemble des champs non public qui reçoivent les données pour assurent le paramétrage du Bean. Chaque propriété (une variable d'instance par exemple) dispose d'une méthode déclarée public pour lire et une autre pour modifier sa valeur.

La Structure d'un Bean

Un Bean est une classe Java qui se conforme à un certain nombre de règles :

- Il doit être public.
- Il doit disposer d'un constructeur public sans paramètres (par défaut).

- Aucun champ public ne doit exister.
- Il peut contenir des propriétés (des champs non publics) qu'on peut y accéder (lecture/écriture) à travers des accesseurs (getter ou setter) publics.
  Suivant des règles de nommage, la méthode de lecture (getter) doit commencer par le préfixe "get" attaché au nom de la propriété qui commence avec une majuscule.

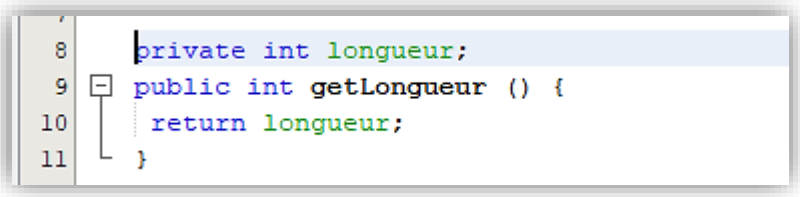


Fig3 : Le Bean implémente l'interface Serializable

Pour une propriété de type booléenne, elle commence par "is" au lieu de "get".
De la même façon, la méthode d'écriture (setter) doit commencer par le préfixe "set" attaché au nom de la propriété qui commence avec une majuscule.

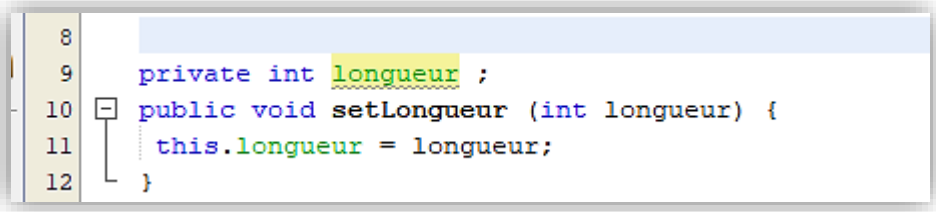


Fig4 : Le Bean implémente l'interface Serializable

La propriété qui n'a pas de getter n'est utilisable qu'en écriture seule, et vis versa.

- Il peut être personnalisé en modifiant ses Propriétés.

Selon les règles précédentes, on trouve que certains packages dédiés pour la réalisation d'applications graphiques ou GUI (Graphic User Interface) tel que Swing et AWT rassemblent des classes qui sont tous des Beans.

3.5 TP 01

Dans cet exemple, on illustre comment créer un simple Bean qui affiche "Hello JavaBean" sur un JLabel.

Beans.java

Source | History

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import java.io.Serializable;
5  /*
6   * @author Loucif Hemza
7   */
8  public class Beans extends JPanel implements Serializable{
9
10 private JLabel l = new JLabel("Hello JavaBean");
11 private int width=200, height=100;
12 public Beans() { setSize(width, height);
13 setBackground(Color.green);
14
15  l.setFont(new Font("TimesNewRoman", Font.BOLD, 20));
16  l.setForeground(Color.red);
17  l.setBorder(BorderFactory.createRaisedBevelBorder());
18  add(l);
19 }
20 public static void main(String args[])
21 {
22     Beans ex = new Beans();
23     JFrame jf = new JFrame("Testing JLabel ... ");
24     jf.getContentPane().add(ex, BorderLayout.CENTER);
25     jf.addWindowListener(
26             new WindowAdapter()
27             {
28                 public void windowClosing(WindowEvent e) {
29                     System.exit(0);
30                 }
31             });
32     jf.setSize(ex.getPreferredSize().width + 20,
33             ex.getPreferredSize().height + 40);
34     jf.setVisible(true);
35 }
36 }
37
```

Fig5 : Un Bean sous forme d'un JLabel

3.6 L'exportation en jar

Afin d'exporter un Bean sous forme d'un fichier jar, il est nécessaire de déclarer un fichier qui se nomme manifest. Le fichier manifest indique au compilateur si une classe particulière ou un ensemble de classe sont prévus d'être un Bean qui sera exporté sous l'extension .jar.

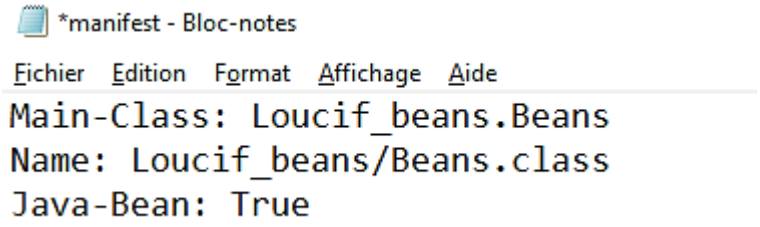


Fig6 : Un fichier manifest

De surcroît, la syntaxe du fichier manifest doit être correct selon les règles suivantes qui doivent être respectés :

- Chaque classe listée dans le fichier manifest doit être séparée d'une autre classe par une ligne vide.
- Si la classe (ou collection de classes) est un Bean, le fichier manifest doit contenir au minimum trois attributs, à savoir :
  - **Main-Class** qui prend le nom de la classe où elle se trouve la fonction **Main ()**.
  - **Name** qui prend le nom du Bean.
  - **Java-Bean** prend la valeur **True** si le résultat de la compilation sera un Bean.
  - La séparation entre le nom du package et le nom de l'une de ses classes doit se faire par un pont (.) dans l'attribut **Main-Class**, et par un slash (/) dans l'attribut **Name**.
- On utilise la commande suivante pour générer un Jar :
  jar cfm output_name.jar manifest_file.tmp

Note : l'extension du fichier manifest est .tmp
Dans la commande cfm le :

- **c** pour créer.
- **f** pour jar file, dont son nom est le premier argument.
- **m** pour manifest, dont son nom est le deuxième argument.

Une autre variante de la commande précédente se représente comme suit :

jar cfm output_name.jar manifest_file.tmp Class_Folder*.*
Le troisième argument **Class_Folder*.*** indique que tous les fichier (classes, images, etc.) qui existent dans le répertoire seront incluent dans le fichier jar résultant.

TP 02

Dans cet exemple de TP, on va illustrer étape par étape comment utiliser NetBeans pour la création des Bean Java.

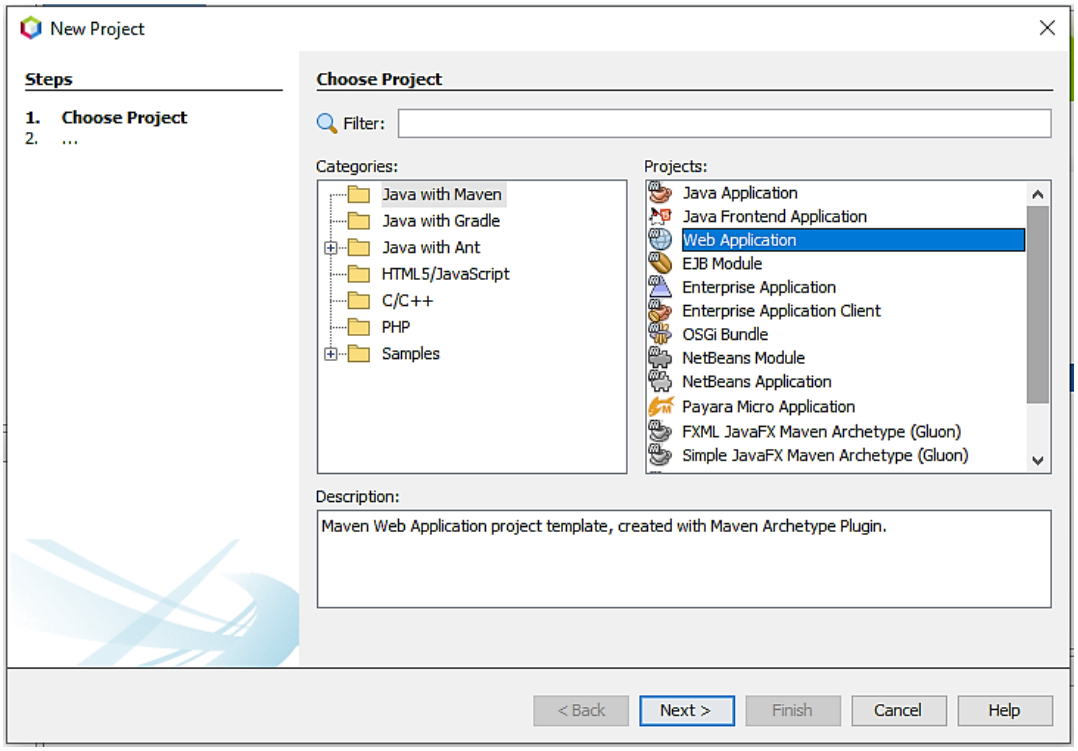


Fig7 : Choisir un projet de type Web Application

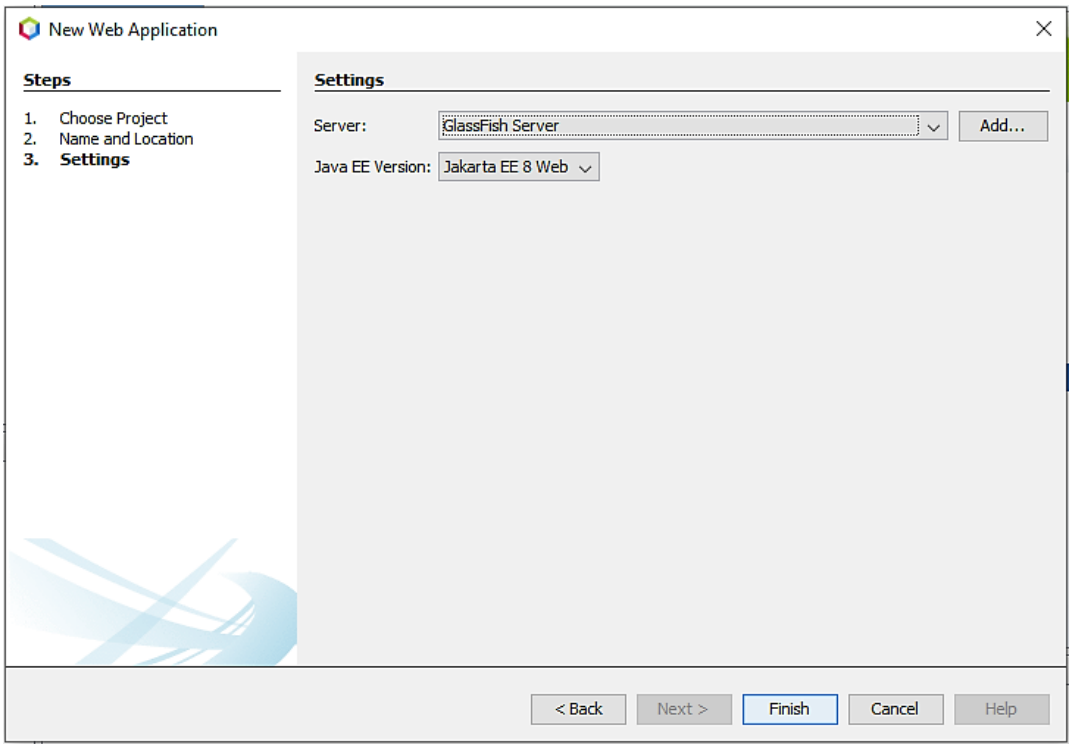


Fig8 : Sélectionner un Serveur d'application.

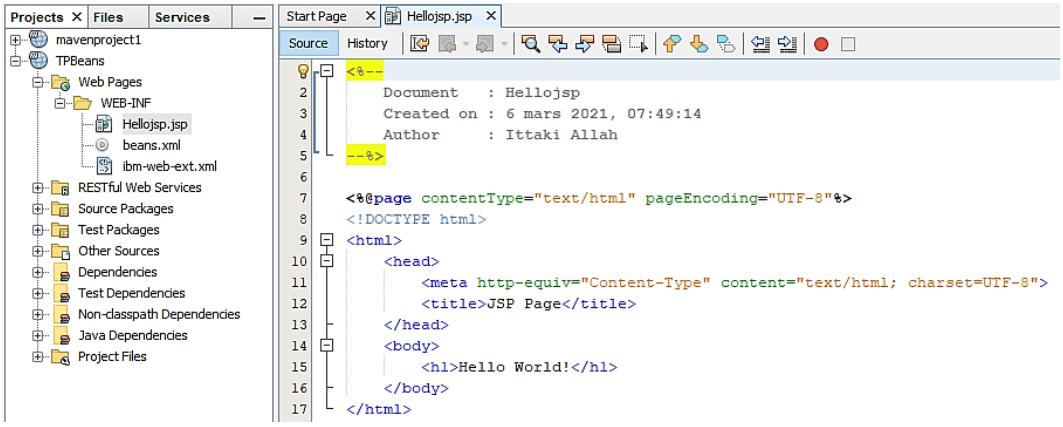
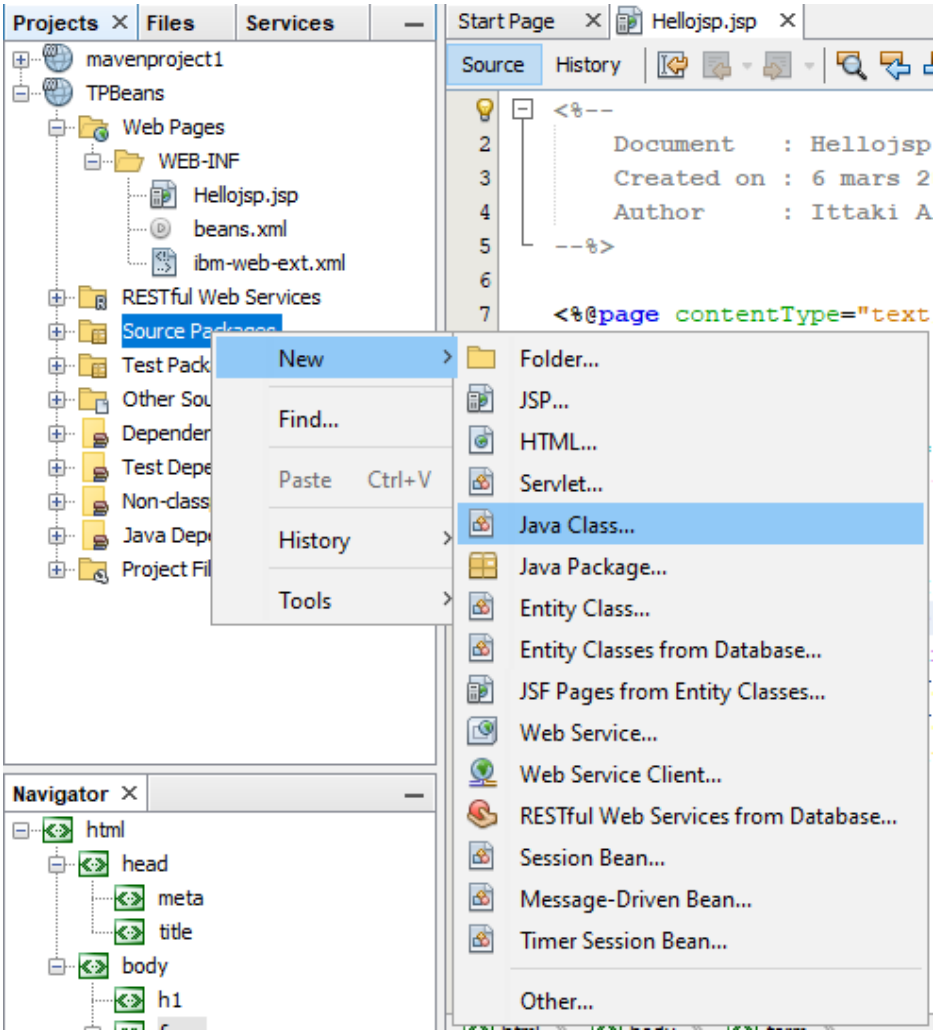


Fig9 : Créer une page JSP d'accueil.



FigI0 : Créer la classe du Bean sous le nom sudentBean.

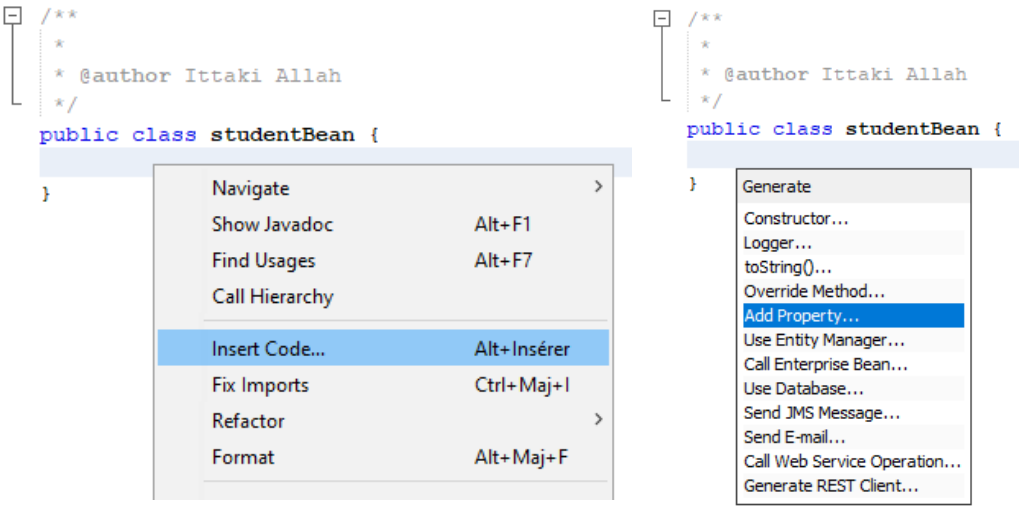


Fig11 : On définit l'ensemble des attributs/méthodes du Bean.

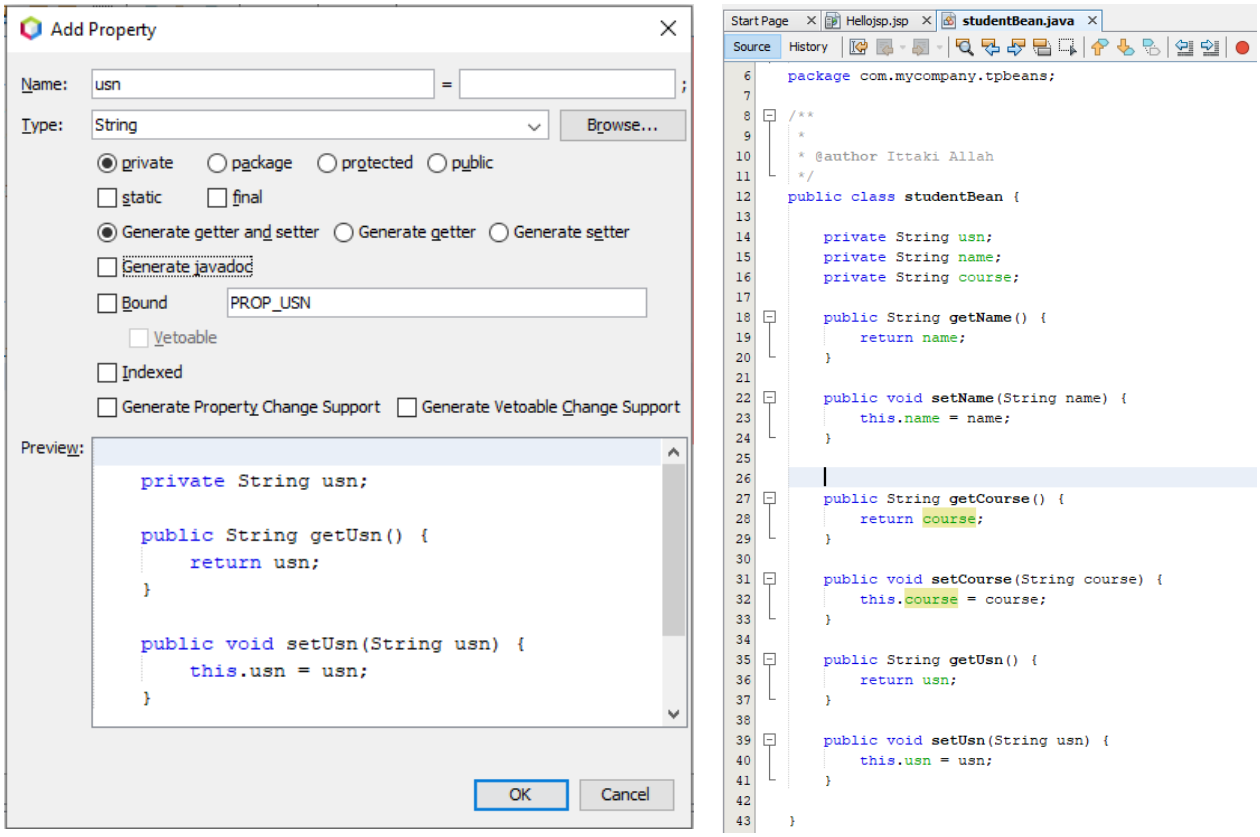


Fig12 : L'assistant qui génère automatiquement les getters/setters du Bean.

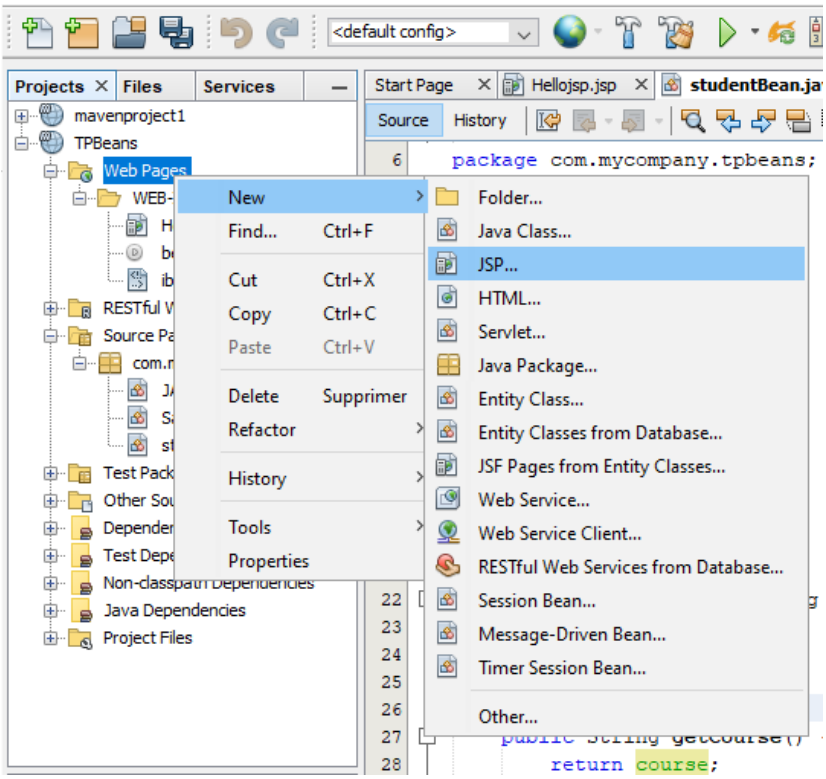


Fig13 : On crée une page JSP dans laquelle on intègre le Bean et récupère les valeurs de ses attributs.

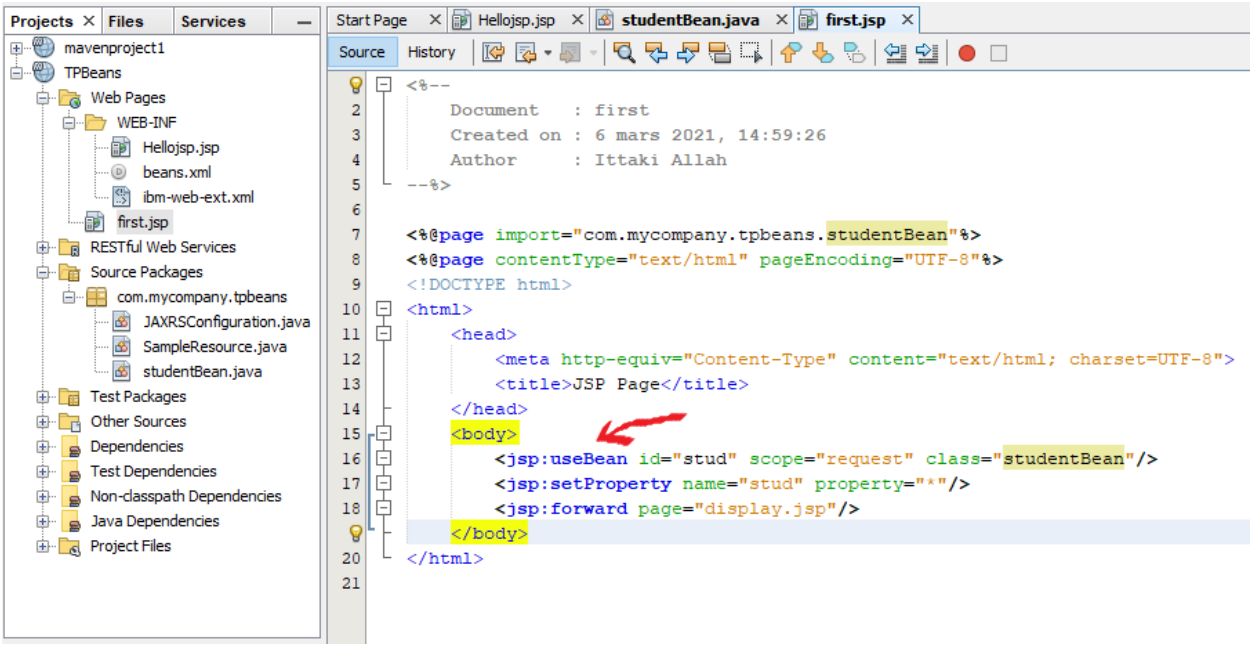


Fig14 : Le code de la page JSP dans laquelle on mit à jours les valeurs des attributs du Bean.

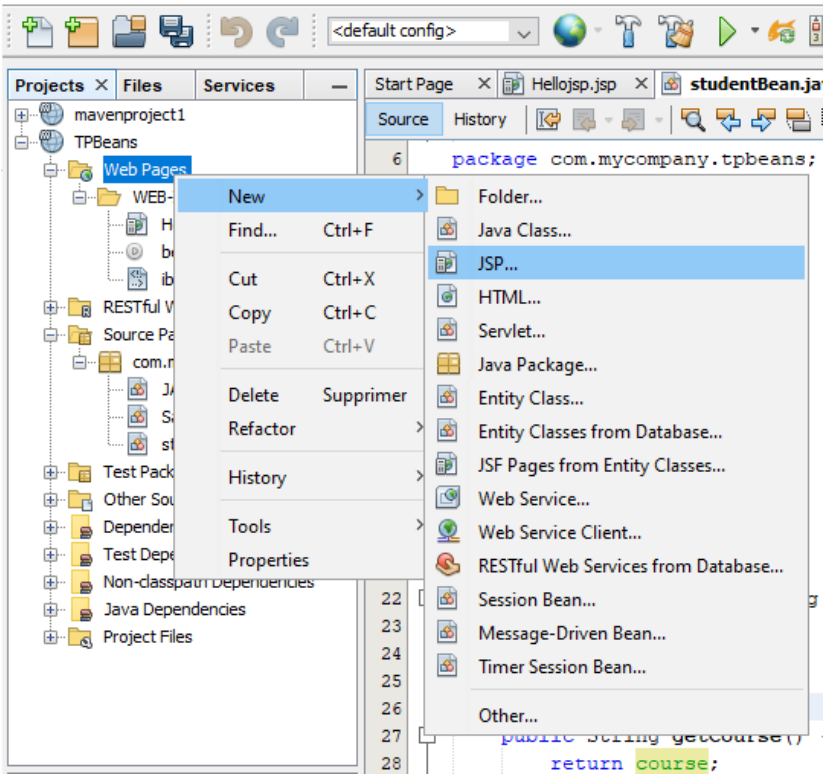


Fig15 : On crée une autre page JSP dans laquelle on intègre le Bean et modifie les valeurs de ses attributs.

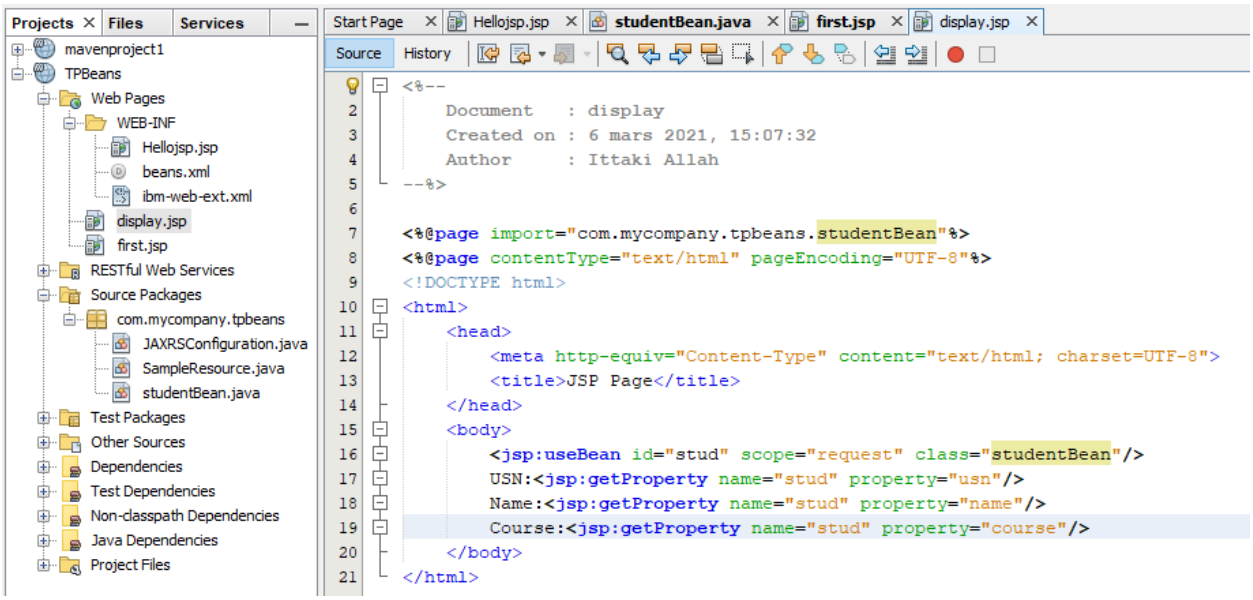
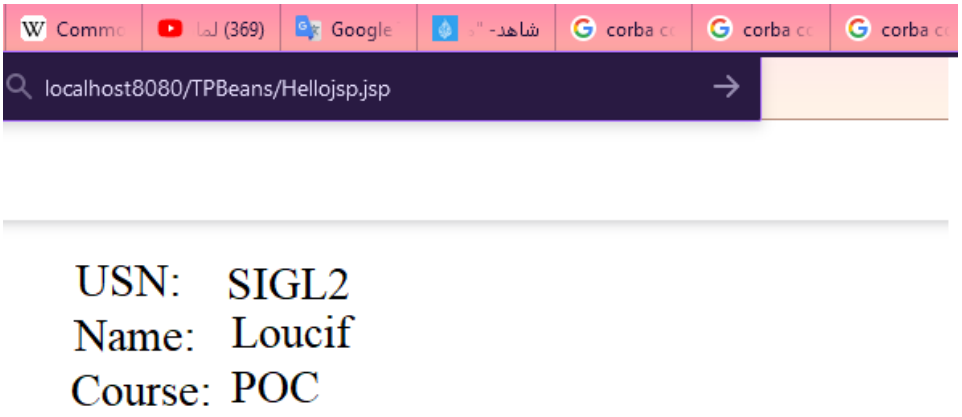


Fig16 : Le code de la page JSP dans laquelle on récupère les valeurs des attributs du Bean.



FigI7 : Après le deployment de l'application.