



L'IDE NetBeans

(version 7)

C. Bonnet
V. Deslandres
Département Informatique ©
IUT de Lyon 1



dernière M&J : 20/03/2017

NetBeans



- * Développé par **Sun**, après le rachat de Forte
- * NetBeans fonctionne sur les systèmes Windows, Linux, MacOs, Solaris.
- * NetBeans permet des développements Java, C++ mais aussi web (PHP et HTML5), web services, et du développement mobile (Android)
- * Son architecture repose entièrement sur **Java**
- * Diffusé gratuitement en open-source sur le site : www.netbeans.org ou fr.netbeans.org

HISTORIQUE

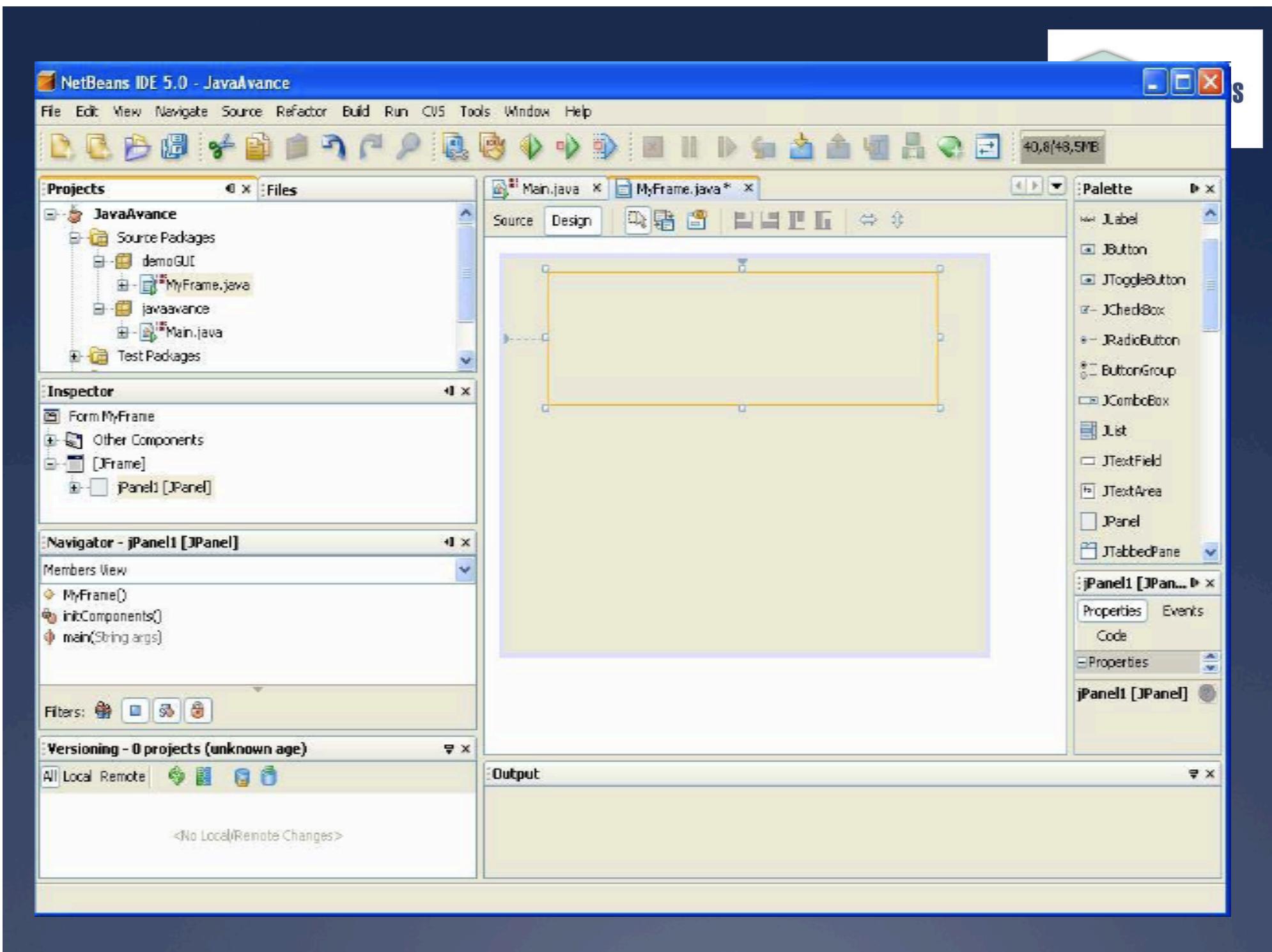


- * 1996 : NetBeans est inauguré par un Tchèque appelé Xelfi.
- * Le but était décrire un IDE de type Delphi, **pour Java** et écrit en Java.
- * Il y avait deux versions commercialisées, nommées "Developper" 2.0 et 2.1.Mai 1999 : la version 3.0 beta sort.
- * Octobre 1999 : Sun Microsystems rachète NetBeans.
- * L'IDE sort sous le nom de "Sun Forte For Java Community Edition". Celle-ci est gratuite. La version professionnelle est payante.
- * Juin 2000 : Sun sort NetBeans en open-source (qui reprend son nom original).
- * Décembre 2004 : la version 4 sort avec le support des nouveautés du langage Java 5, une meilleur gestion des projets, une interface graphique améliorée, une utilisation accrue de Ant, ...
- * Janvier 2012 : version 7.0

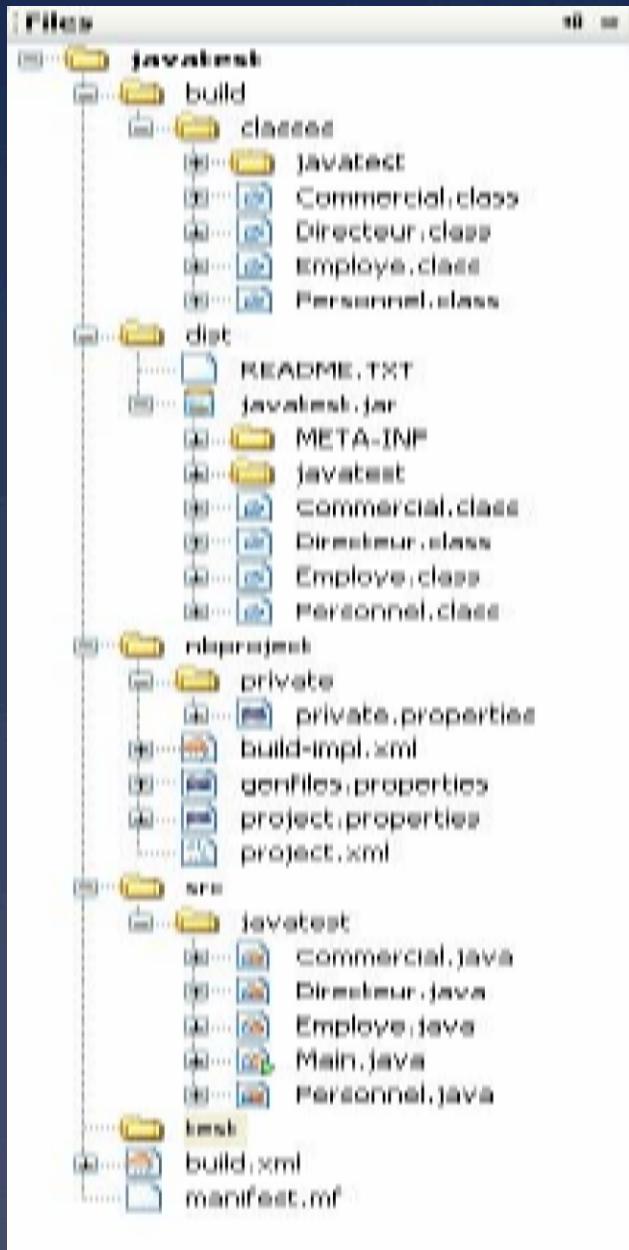
Richesse d'un IDE



- * Un éditeur visuel de GUI (**Graphical User Interface**)
- * Gestion de build et des dépendances avec **MAVEN**
- * Support pour des **VCS** (Versioning Control System)
- * Un **débogueur** très puissant
- * **Refactoring** de code
- * Développement d'applications Web en utilisant JSP, JSF et Struts
- * Développement des EJB (**Enterprise Java Beans**)
- * Développement de Web Services
- * Développement de modules ou d'applications RCP (Rich Client Platform)
 - * Avec **JavaFX** ou la **Swing**
- * Le serveur d'applications web **Tomcat** ou **GlassFich**
- * Des **exemples** d'applications
- * Le catalogue Java Blue Prints
 - * Bonnes pratiques de dévpt JEE 4



Organisation des fichiers



- * **build** : bytecodes
- * **dist** : répertoire de distribution
 - * Contient tout ce qui est nécessaire pour déployer l'application (jar, javadoc...)
 - * Dist/lib : les archives nécessaires
- * **nbproject**: fichiers de configuration du projet, utiles à ANT et NetBeans
- * **src** : fichiers sources
- * **test** : sources dédiées aux tests
 - + les fichiers ANT **buid.xml** et **manifest.mf** (équivalent du make) ou **pom.xml** de MAVEN

Rappel : manifest.mf



- * Dans le répertoire META-INF de l'archive jar du projet
- * Fichier de configuration permettant de rendre une application exécutable. Il spécifie :
 - * La classe principale
 - * Le Classpath des archives jar nécessaires
 - * Une éventuelle signature électronique, etc.
- * Le manifeste comporte une spécification par ligne.
- * Une ligne = “type de spécification : sa valeur”
 - * La ligne doit obligatoirement se terminer par un retour à la ligne
- * Exemple : manifest par défaut

```
Manifest-Version: 1.0
X-COMMENT: Main-Class will be added automatically by build
```

- * NOTA : le format « .jar » peut être utilisé pour compresser des fichiers au même titre que le format zip ou rar. C'est une **archive** compressée.
 - * Fonctionne sur des systèmes variés

Programme Ant (Apache)



- * **Ant** est un outil puissant qui permet d'automatiser certaines tâches du développeur averti :
 - * Compilation dans un dossier donné
 - * Archivage des sources, des binaires
 - * Exécution de tests unitaires
 - * Génération de la javadoc et des rapports de tests
 - * Déploiement sur un serveur distant
 - * Lancement de plusieurs navigateurs avec son appli web, etc ...
- * **Ant** est appelé automatiquement lors de la compilation d'un projet standard
→ génération d'un script "build.xml".

ANT : définitions



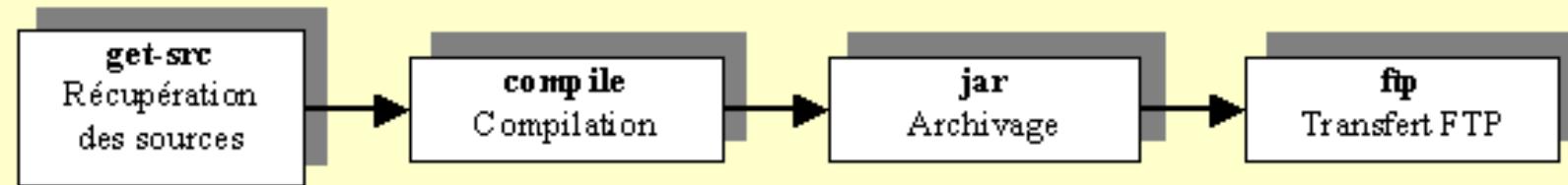
- * Le fichier de configuration de ANT, exprimé au format XML, décrit un **projet** ANT
 - * Voir **build.xml** et **build-impl.xml** (ou fichier .properties)
- * Un projet ANT est essentiellement constitué d'une liste de **cibles**, correspondant à des opérations unitaires plus ou moins complexes
 - * Ex. : récupération des sources, compilation Java, déploiement d'une application,...
- * Il est possible de définir **des relations de dépendance** entre les différentes cibles

ANT : définitions (2)



- * Une cible est elle-même composée de **tâches**. Une tâche correspond à un traitement unitaire, non décomposable : copie de fichier, compression JAR, etc.

Ex.: processus qui transfert via FTP un fichier JAR construit à partir des sources récupérées sur le référentiel de sources

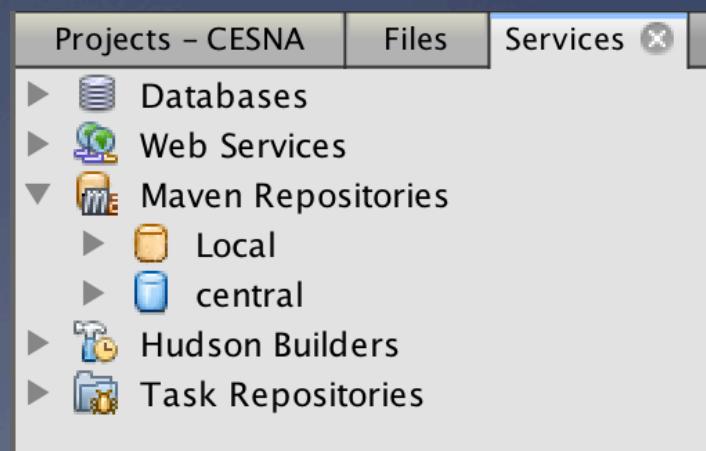


- * Projet ANT : configurable à l'aide de **propriétés**. Ces propriétés peuvent être définies dans le fichier de configuration (build.xml), dans un fichier de propriétés (.properties) ou en argument de l'exécution.
- * Netbeans repose entièrement sur ANT, de manière transparente :
 - * Ainsi pour la compilation par exemple, seules les classes ayant été modifiées depuis la dernière compilation seront compilées.

Ant → MAVEN



- * Depuis, il existe un outil de « production de programme » appelé MAVEN qui fait tout ce que permettait Ant, et plus encore, de manière plus aisée
 - * “outil de build”
- * Intégré à Netbeans
- * Basé sur un POM.xml (Project Object Model)
 - * Décrivant son contenu, son packaging (ce qu'on veut en faire, un jar, une DLL...), s'il a un parent, et toutes les dépendances : librairies, versions des compilateurs, du moteur d'exécution, etc.
- * Utilisé dans toutes les entreprises
 - * Car simplifie le développement de logiciels



Maven : un fichier POM



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org  
/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-  
v4_0_0.xsd">  
    <modelVersion>4.0.0</modelVersion>  
    <groupId>org.ccm.maven</groupId>  
    <artifactId>helloworld</artifactId>  
    <packaging>jar</packaging>  
    <version>1.0-SNAPSHOT</version>  
    <name>helloworld</name>  
    <url>http://maven.apache.org</url>  
    <dependencies>  
        <dependency>  
            <groupId>junit</groupId>  
            <artifactId>junit</artifactId>  
            <version>3.8.1</version>  
            <scope>test</scope>  
        </dependency>  
    </dependencies>  
</project>
```

Nom du projet (livrable : artifact)

Type de packaging (build un jar)

Type de livrable (en cours de dév)

Dépendances : ici JUnit (tests unitaires)

Liens utiles MAVEN



- * <http://maven-guide-fr.erwan-alliaume.com/maven-guide-fr/site/reference/lifecycle.html>
 - * Cycle des étapes dans la construction d'un logiciel MAVEN
- * <https://platform.netbeans.org/tutorials/nbm-maven-quickstart.html#02>
 - * Utilisation de MAVEN avec NETBEANS



Fonctionnement de l'IDE

- Auto-complétion
- Correction automatique
- Compilation automatisée
- Refactoring
- Debugage
- Développement d'IHM, etc.

Facilité d'écriture du code



* Auto complétion : **Ctrl Espace**



* Correction automatique : symbole Ampoule, puis choisir la correction



The screenshot shows the NetBeans IDE interface. On the left, there's a code editor window titled "Main.java" with some Java code. On the right, a detailed view of the "String" class is displayed. A large yellow arrow points from the "Auto completion" text above to this detailed view. Below the detailed view, another yellow arrow points from the "Correction automatique" text above to the code editor window.

```
Main.java * x
String*
1   String
2     String (com.sun.org.apache.xpath.internal.operations)
3     String (java.lang)
4       StringBuffer (java.lang)
5         StringBufferInputStream (java.io)
6         StringBufferPool (com.sun.org.apache.xml.internal.utils)
7         StringBuilder (java.lang)
8           StringCharacterIterator (com.sun.org.apache.regexp.internal)
9             StringCharacterIterator (java.text)
10            StringComparable (com.sun.org.apache.xml.internal.utils)
11              StringContent (javax.swing.text)
12                StringDV (com.sun.org.apache.xerces.internal.impl.dv.xs)
13                  StringDatatypeValidator (com.sun.org.apache.xerces.internal.impl.dv.dtd)
14                    StringHeadTail (com.sun.jndi.toolkit.ctx)
15                      StringHolder (org.omg.CORBA)
16                        StringIndexOutOfBoundsException (java.lang)
                           StringList (com.sun.org.apache.xerces.internal.xs)

String
18
19
20
21
22
23
24
25
26
17:1
Local V

java.lang.String
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
The String class represents character strings. All string literals in Java programs,
such as "abc", are implemented as instances of this class.
Strings are constant; their values cannot be changed after they are created. String
buffers support mutable strings. Because String objects are immutable they can be
shared. For example:
String str = "abc";
```

The screenshot shows the NetBeans code editor with Java code. A tooltip is open over the variable "locale", showing two options: "New instance ignored" and "Assign Return Value To New Variable". A second tooltip is open over the line of code "internationalisation2 = new Internationalisation2(locale);", also showing two options: "Assign Return Value To New Variable" and "Configure 'Result of new Object ignored' Hint". A yellow arrow points from the "Correction automatique" text above to this part of the interface.

```
120
121   public static void main(String args[]) {
122
123     Locale locale = Locale.getDefault();
124     New instance ignored
125     h == 3)
126     = new Locale(args[0], args[1], args[2]);
127
128     internationalisation2 internationalisation2 =
129       new Internationalisation2(locale);
130
131     Assign Return Value To New Variable >
132     Configure "Result of new Object ignored" Hint >
```

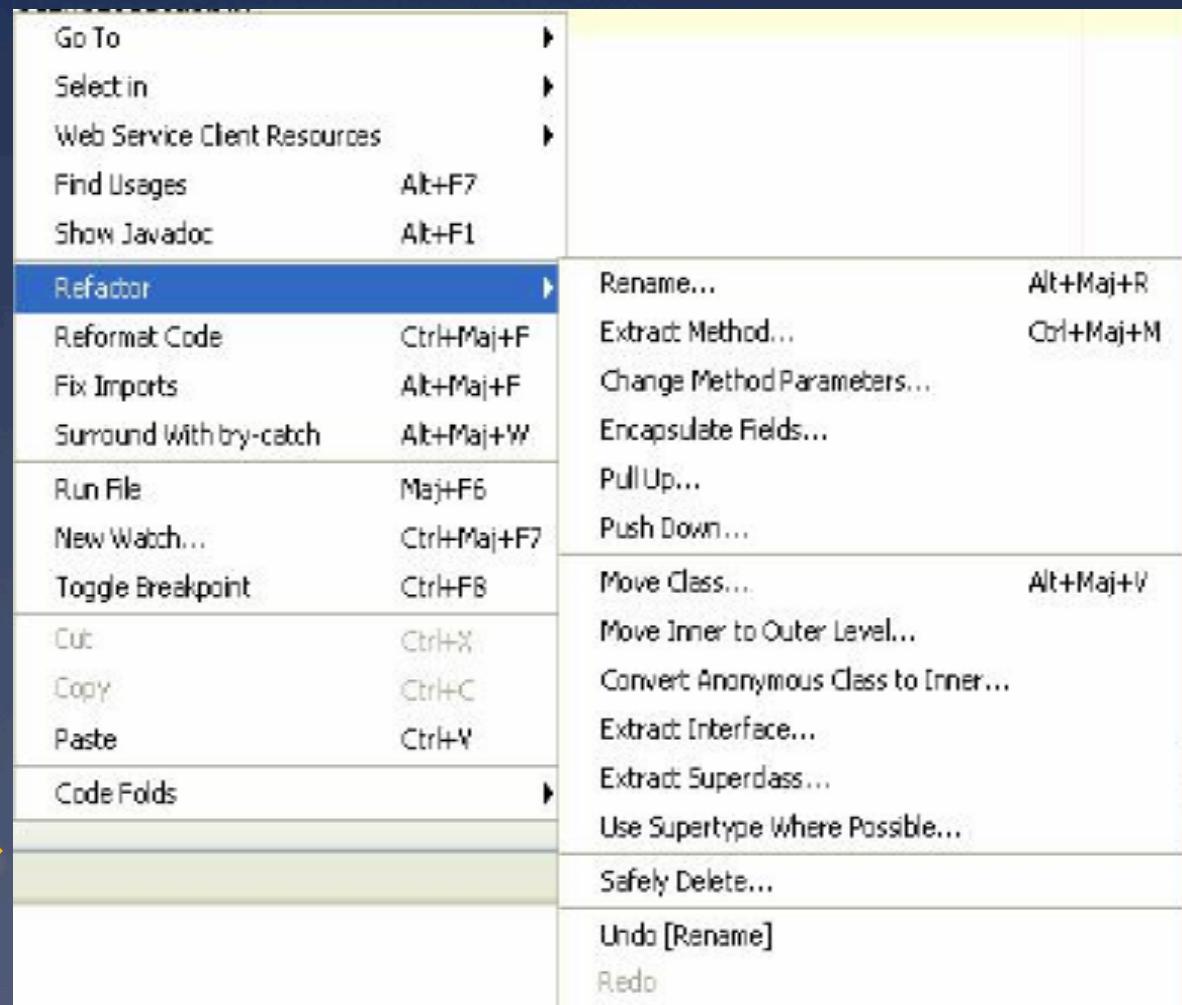
Facilité de codage (2)



* **Refactoring:** modification automatique du code Java

Possibilité de **déplacer**, **renommer** ou plus généralement modifier une partie du code source en contrôlant l'impact sur le reste du code.

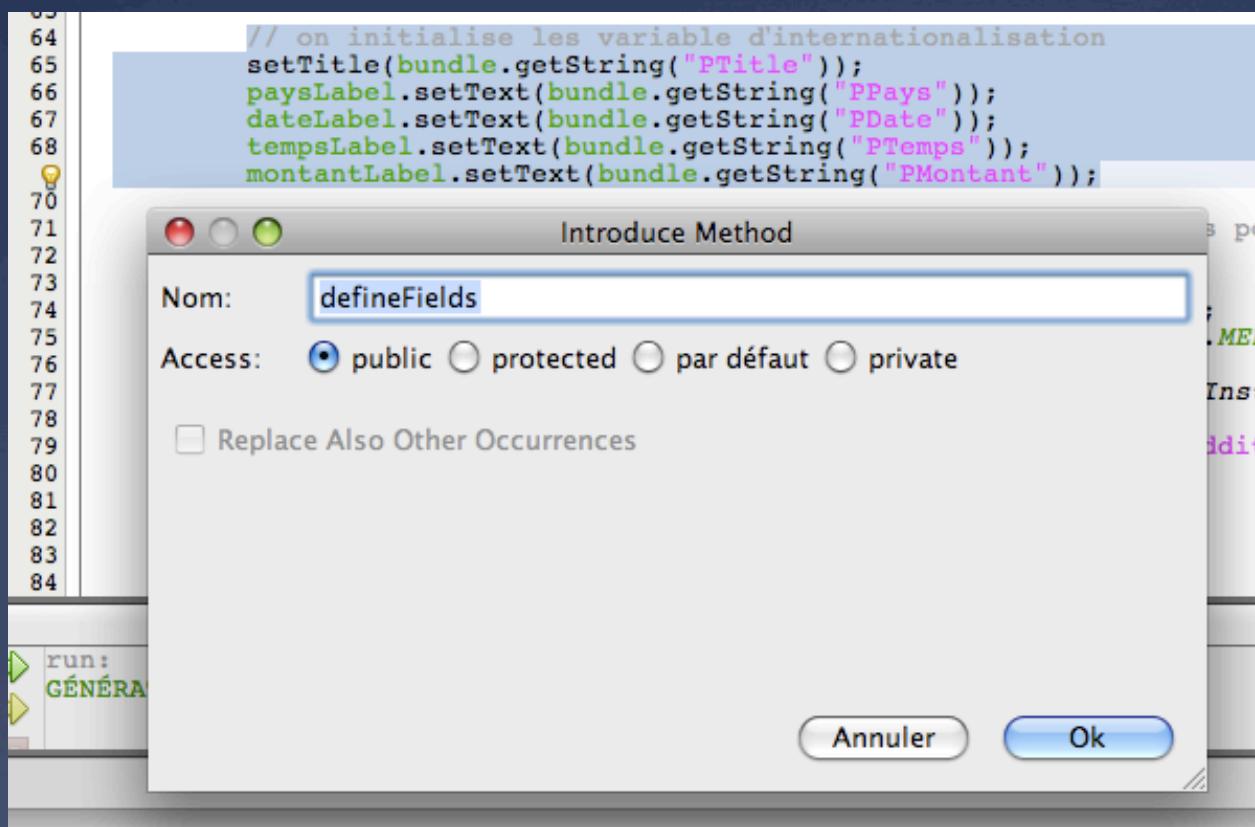
Fonctions de refactoring :



Ex. : Extraction de code



- * Création d'une nouvelle méthode à partir des lignes sélectionnées
 - ➔ remplacement de toutes les références à ces éléments (partout dans le code) par un appel à cette méthode
 - ➔ Sélectionner le code
 - ➔ Clic Droit : **Refactorer – Introduce Method**



Extraction (suite)



→ Modifie le code :



```
62 setSize(293, 123);  
63 defineFields();  
64
```

→ Crée la méthode :

```
81  
82     public void defineFields() {  
83         // on initialise les variables d'internationalisation  
84         setTitle(bundle.getString("PTitle"));  
85         paysLabel.setText(bundle.getString("PPays"));  
86         dateLabel.setText(bundle.getString("PDate"));  
87         tempsLabel.setText(bundle.getString("PTemps"));  
88         montantLabel.setText(bundle.getString("PMontant"));  
89     }  
90 }
```

→ Remplace les lignes identiques par l'appel partout où c'est nécessaire

Autre ex.: renommer



- * On peut renommer un identifiant de package, de classe, de méthode ou d'attribut
 - * Sélection – clic Droit – **Refactorer -> Rename**
 - * Soit vous êtes sûr de vous : clic sur **Refactor**
 - * Soit vous prévisualisez l'impact: **Aperçu** (ou Preview) dans la fenêtre de dialogue
 - * **Check** sur chaque modification, puis **Next**
 - * Puis **Do Refactoring** à la fin
 - * Toujours finir un **Refactoring** par un **Clean and Built** du projet

Configurer la compilation ou l'exécution



Sélectionner le Projet - Clic Droit : Propriétés

- * Choisir le point à configurer
- * **Compilation :**
 - * Choix de la faire à chaque sauvegarde
 - * Choix de construction du JAR à chaque compilation
 - * Insertion des bibliothèques dépendantes
 - * Options de génération de la javadoc
- * **Exécution :** on peut définir différents profils d'exécution
 - * Classe principale
 - * Arguments à passer
 - * **Run – Set Project Configuration - Customize**
 - * Répertoire de travail
 - * Options de la machine virtuelle

Débuguer



- * Menu Débogage
 - * Placer des points d'arrêt dans le code :
 - * **clic droit sur une ligne → Toggle Breakpoint**
 - * **clic Breakpoint**
 - * Démarrer le débogage
 - * Pas à pas (F5)
 - * Instruction suivante (F7)
- * Les fenêtres de **Debugging**
 - * Sortie, variables, ...
 - * Modification des valeurs en dynamique, les appliquer, etc.

[http://docs.oracle.com/cd/E50453_01/doc.80/e50452/
run_debug_japps.htm#NBDAG798](http://docs.oracle.com/cd/E50453_01/doc.80/e50452/run_debug_japps.htm#NBDAG798)

Fenêtre Débogage



Sous menu Fenêtre :

- Projets ⌘1
- Fichiers ⌘2
- Favoris ⌘3
- Services ⌘5
- Tâches ⌘6
- Palette ⌘8
- Propriétés ⌘7
- Sortie
- Navigation
- Débogage
- Profiling
- Contrôle de versions
- Autre
- Éditeur ⌘0
- Processus
- Fermer la fenêtre ⌘W
- Agrandir la fenêtre ⌘O
- Déancrer la fenêtre ⌘D

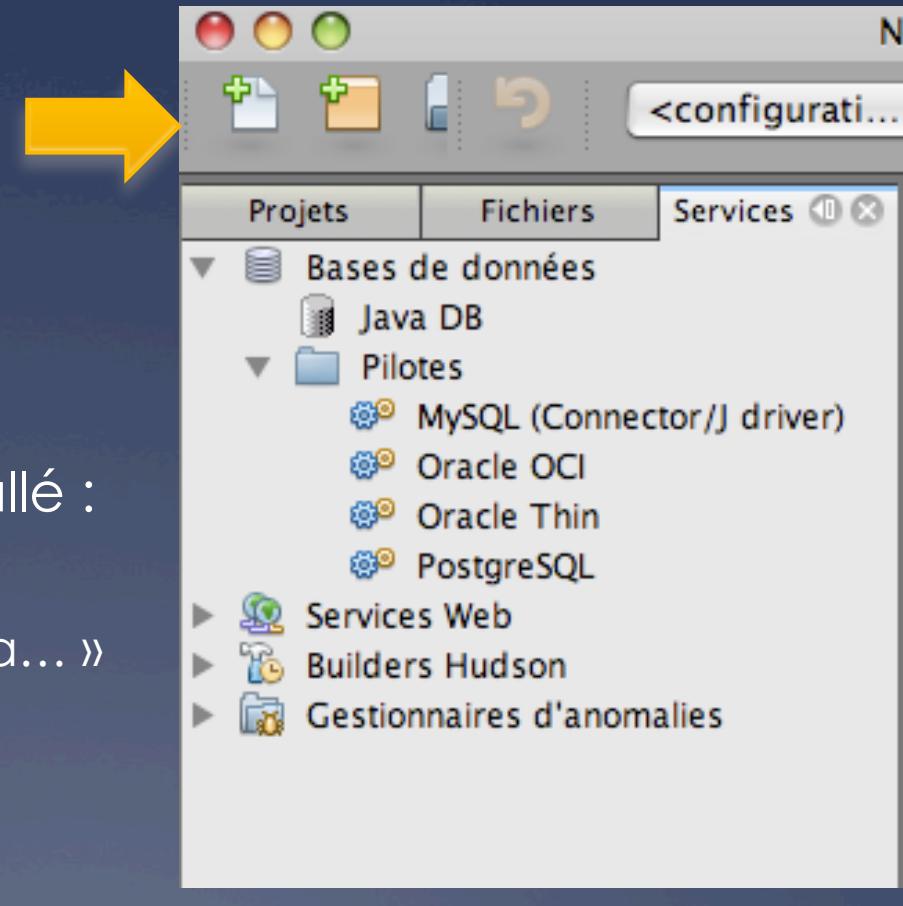
Le menu déroulant "Débogage" est actif et affiche les options suivantes :

- Variables ↑1
- Témoins ↑2
- Pile d'appel ↑3
- Classes Chargées ↑4
- Points d'arrêt ↑5
- Sessions ↑6
- Threads ↑7
- Sources ↑8
- Débogage ↑9

Connexion à une BD



- * Il faut éventuellement **ajouter un pilote** JDBC à la BD souhaitées
 - * Item Drivers / Pilotes – nouveau pilote
- * Voir les pilotes installés :
 - * Fenêtre Services, item BD
- * **Connecter une BD**
 - * Soit clic Droit sur BD :
 - * « Nouvelle connexion... »
 - * Soit sur un pilote déjà installé :
 - * Clic Droit
 - * « Etablir une connexion via... »



Analyse de performances



- * Profiler Java : Menu **Profile**
- * Analyse de CPU, de la génération de charge, de l'utilisation mémoire, ...

Profiling Task	Results
Monitor Application	Choose this to obtain high-level information about properties of the target JVM, including thread activity and memory allocations.
Analyze CPU Performance	Choose this to obtain detailed data on application performance, including the time to execute methods and the number of times the method is invoked.
Analyze Memory Usage	Choose this to obtain detailed data on object allocation and garbage collection.

https://blogs.oracle.com/nbprofiler/entry/getting_started_with_netbeans_profiler

Exemple Profilage utile



- * Profilage de méthode
- * On observe qu'une méthode fréquemment appelée prend **beaucoup trop de temps à s'exécuter**, et constitue ainsi un sérieux goulot d'étranglement.
- * **Analyse du code** : il s'avère qu'on vérifiait si un élément était présent dans une collection avec la méthode `contains()`. La collection était une `LinkedList`.
- * La raison ? Les listes chaînées ont une complexité en temps de **O(n)** pour des fonctions de recherche comme `contains()`.
- * Correction ? Remplacer la liste chaînée par un **HashSet**, qui effectue `contains()` beaucoup plus rapidement, en temps **O(1)**.

Autres...



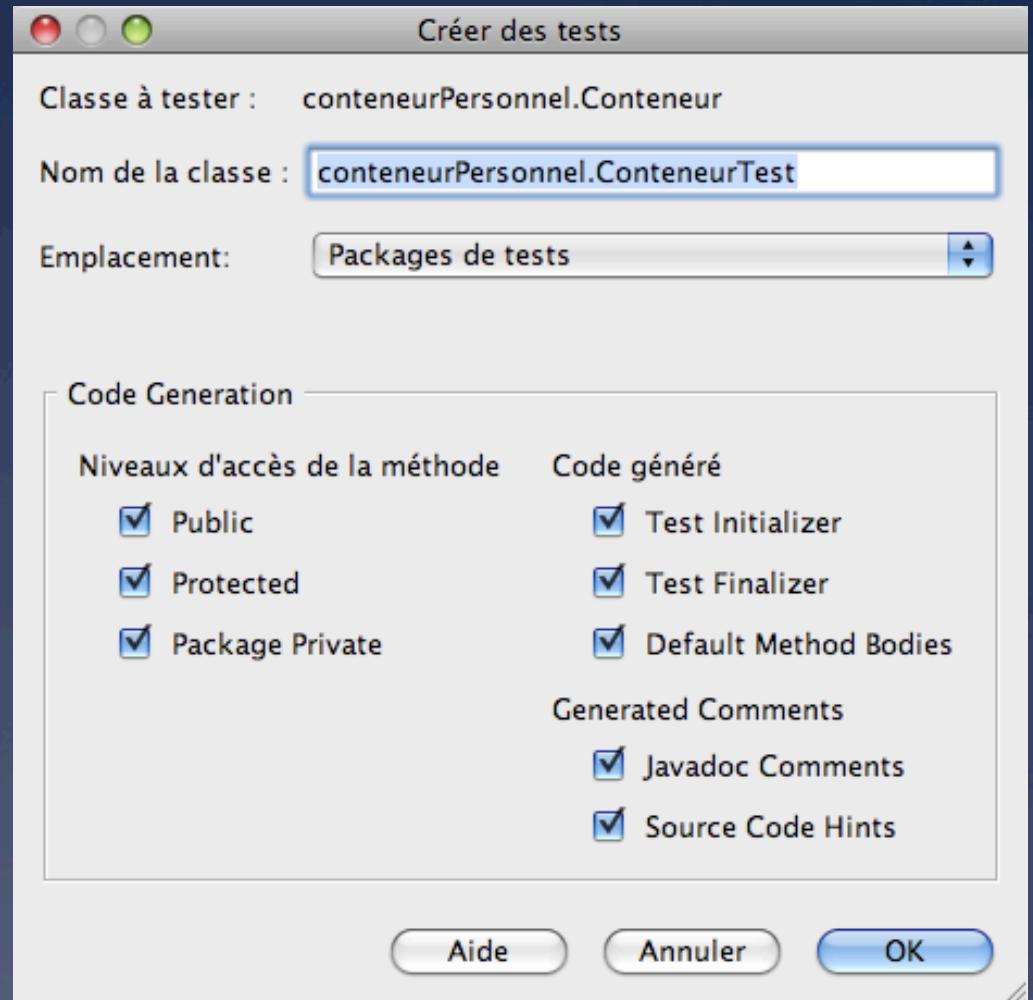
- * Génération de la **Javadoc**
 - * Build-> Generate JavaDoc
 - * La doc est placée dans le répertoire **dist** (onglet Files)
- * **Internationalisation** (menu Outil)
 - * Assistant permettant de mettre en œuvre les mécanismes d'internationalisation de Java
- * Classes :
 - * **java.util.Locale**
 - * **java.util.ResourceBundle**
 - + Classes de mise en forme :
DateFormat, MessageFormat, NumberFormat

Exemple d'internationalisation	
Pays	FRANCE
Date	21/02/3012
Temp	4°
Montant	25,5€

JUnit



- * Framework openSource de Tests unitaires intégré dans NB
 - * Rédaction et exécution de tests unitaires
 - * Création d'une classe de tests par classe à tester
- * Pour **créer** un test :
 - * Ajouter la librairies Add Livrairies (Junit et Harmcrest)
 - * sélectionner une classe dans la fenêtre Project
 - * Clic Droit : Outils – Créer un test unitaire



JUnit (suite)



- * Crée alors le squelette d'une classe de Tests pour la classe choisie, dans le *Package de Tests*, de toutes les méthodes
 - * Ex.: pour une méthode
 - * Méthode de test générée :

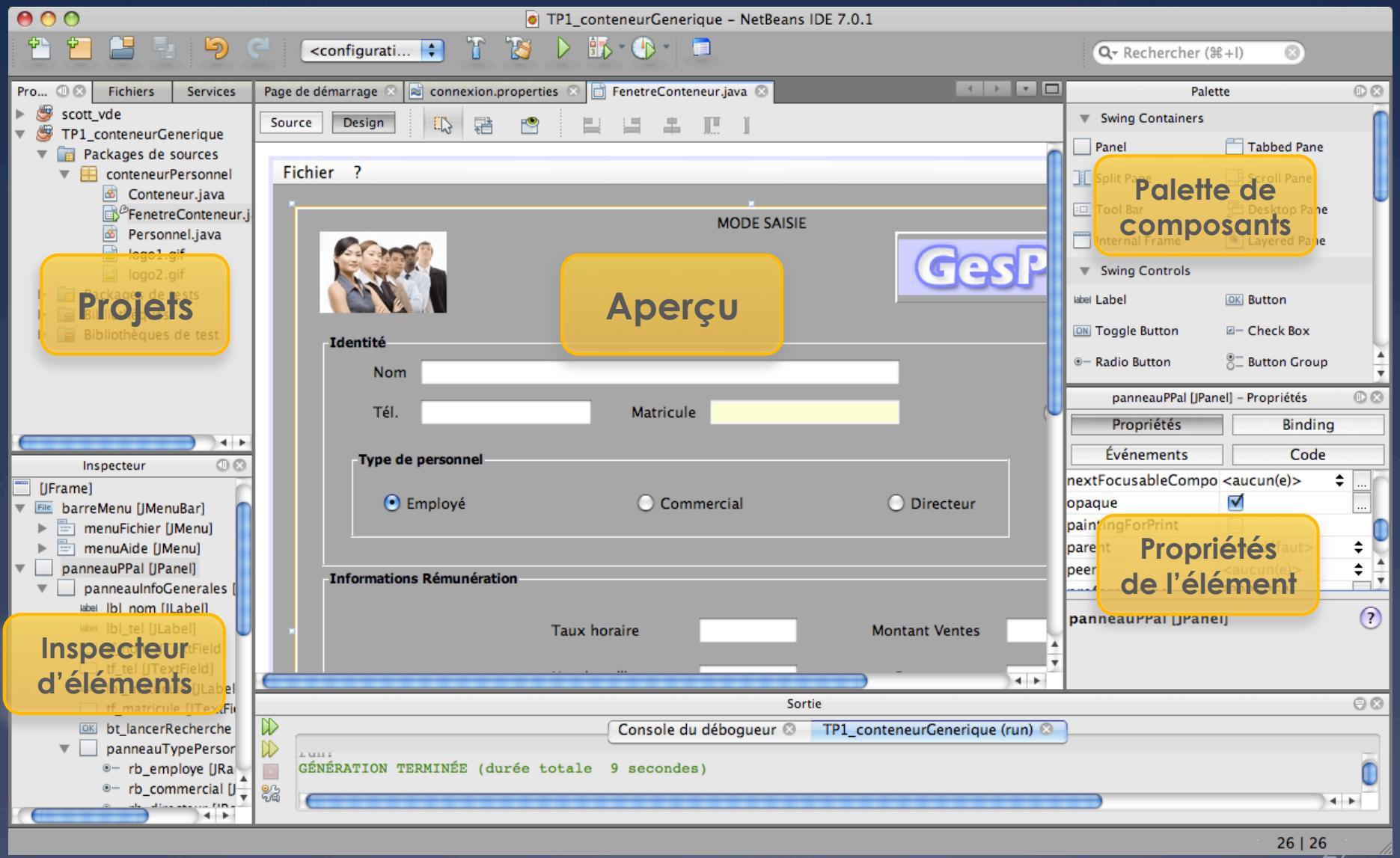
```
// indique si le conteneur est vide
public boolean estVide() {
    return tM.isEmpty();
}
```

```
public void testEstVide() {
    System.out.println("estVide");
    Conteneur instance = new Conteneur();
    boolean expResult = false;
    boolean result = instance.estVide();
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}
```

- * Code à modifier et compléter ! (sinon tjs *fail* !)
- * On peut ensuite exécuter
 - * Toute la classe de test : **clic droit – exécuter le fichier**
 - * Juste une méthode :
- * Cf <http://www.junit.org/> (il existe PHPunit et Nunit - pour .Net)

Swing GUI Builder

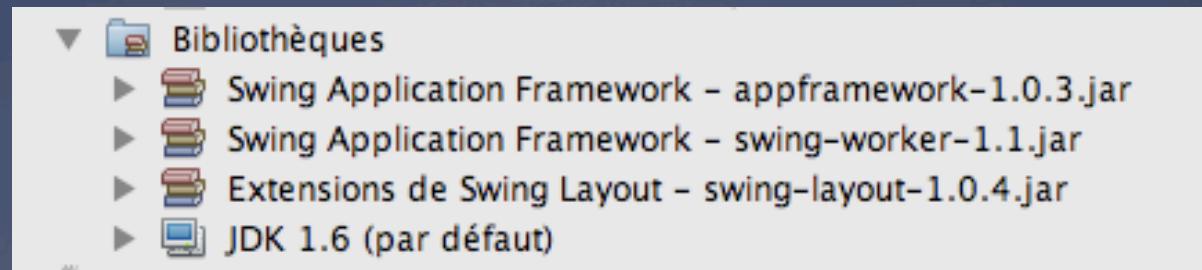
(non maintenue depuis 2008)



Présentation du GUI Builder



- * L'environnement Swing GUI Builder est un éditeur intuitif pour construire des interfaces utilisateur en glissant-déposant des composants en mode WYSIWYG
 - * *What You See Is What You Get*
- * C'était l'un des plus évolué des IDE
- * L'IDE suggère automatiquement l'alignement, l'espacement et le redimensionnement.
- * Les « layouts » sont gérés automatiquement.



Le Swing GUI Builder



TP à venir en proglHM...

- * Il existe maintenant un **débogueur visuel** permettant de tester votre interface !
- * Cf <https://netbeans.org/kb/docs/java/debug-visual.html>
 - * Avec un tutoriel sur un fichier exemple

Astuce n°1...



- * Pour **générer automatiquement les getters et setters** d'une classe, il suffit de faire :
 - * bouton droit sur cette classe (dans les différentes vues (**Projects, File**) ou l'éditeur java)
-> **refactor -> encapsulate fields**
(ou sur un attribut donné, sous éditeur)
 - * ou, depuis le menu principal
refactor -> encapsulate fields

Astuce n°2



- * Pour **organiser automatiquement ses imports** (par package, et dans le bon ordre) :
 - * Rappel : l'ordre doit être **java, javax, org, com**
- * Dans l'éditeur :
 - * bouton droit -> **Fix Imports**
 - * Si la résolution automatique peut porter à confusion (par exemple **java.util.Date** et **java.sql.Date**) : une boite de dialogue demande de faire son choix
 - * Mais le mieux est d'être explicite dans le code à chaque endroit où il peut y avoir confusion !

Astuce n°3



- * Pour **consulter les sources d'une classe** (par exemple String) :
 - * placer le pointeur de la souris au-dessus de la déclaration String et appuyer sur la touche Ctrl - Espace
 - * La déclaration String se transforme alors en lien hypertexte.
 - * Il suffit de cliquer dessus pour aller directement dans les sources.
- * Utiliser le raccourci clavier : **Alt-O**
 - * Il faut pour cela avoir indiqué où était la Javadoc !

Astuce n°4



Configuration du JDK et de la javadoc

- * Spécifier le chemin dans la boite de dialogue : menu **Outils -> Java Platform**
- * Sélectionner le *jdk* dans le panneau de gauche.
- * Cliquer dans le panneau de droite sur l'onglet *Sources* et, dans la boite de dialogue, cliquer sur **Add Jar/Folder**.
- * Indiquer ensuite le répertoire ou l'archive de la javadoc
 - * NB : préférer la Javadoc sous forme de répertoire que sous forme d'archive, sinon ça ralentit NB quand on l'utilise.

Astuce n°5



- * Pour formater correctement son code : menu **Source -> Format**
- * Ignorer une partie de son code en le mettant en commentaires :

* Icône



Modèles de code (template)



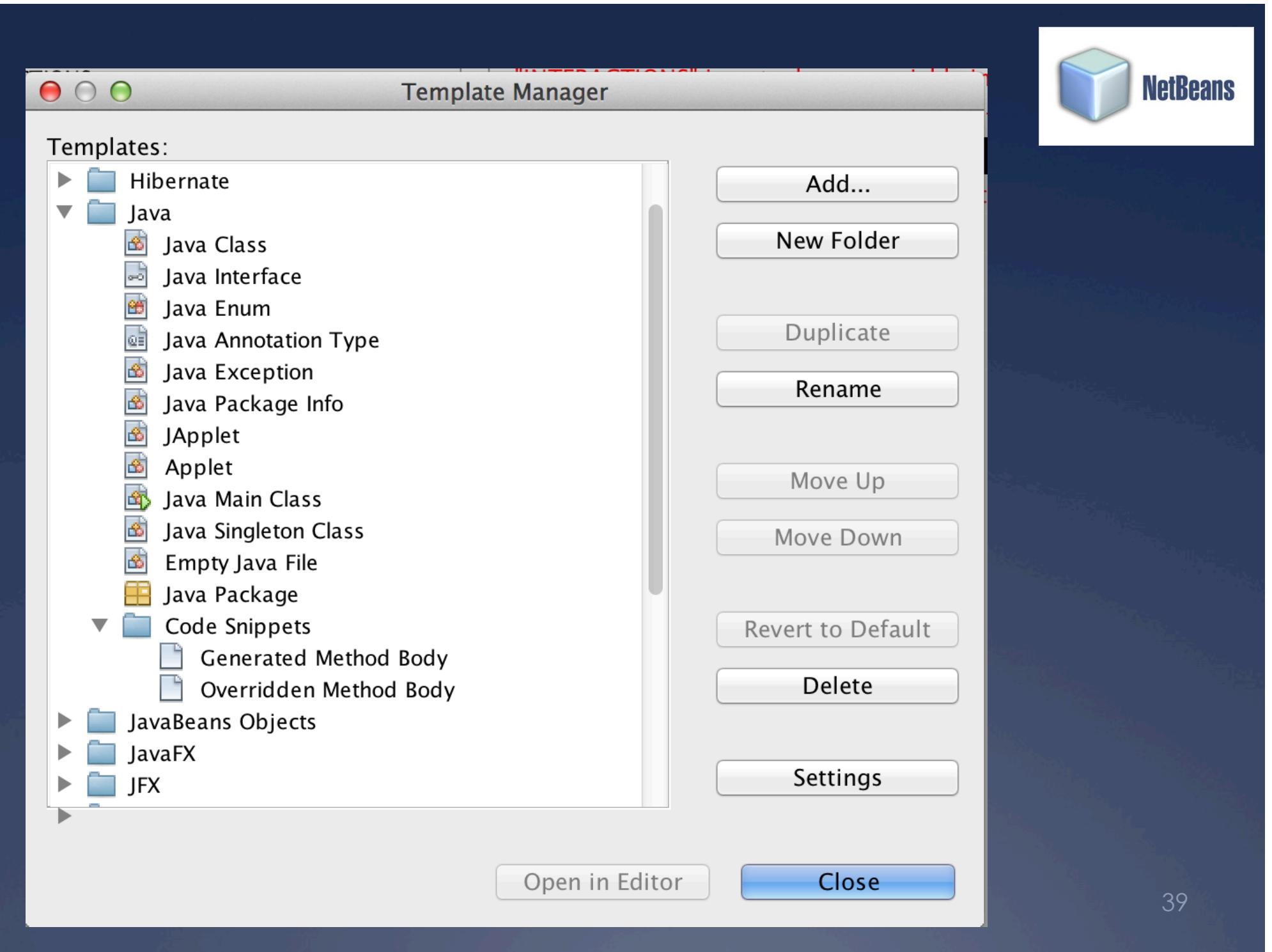
- * Les **modèles de code** sont des bouts de code accessibles via des raccourcis typographiques.
 - * Exemple : taper **sout** suivi de la touche *Tab*
→ **System.out.println("");** est inséré avec le curseur positionné entre les guillemets
 - * Autres exemples utiles :
 - * Boucle d'itération : taper **for** et *ctrl + espace*
 - * NetBeans suggérera 3 itérations possibles.
 - * Condition : taper **if** et *ctrl + espace*
 - * NetBeans créera la structure :

```
if (condition) {} else {}
```
 - * (Voir les templates prédéfinis : lors des choix de complétion)

Créer ses propres modèles de code



- * Pour créer ses propres templates de code :
 - * **Tools -> Template**
 - * Choisir l'onglet **Java**



Raccourcis clavier utiles



- * **Ctrl + espace** : complétion automatique
- * **Alt + Maj + F** : Fix imports (ajouter les imports nécessaires /supprimer les imports inutilisés)
- * **Alt + Maj + I** : Ajoute la directive d'import pour la classe où se situe le curseur
- * **Ctrl + Maj + P** : Recherche d'une chaîne de caractère, un type de fichier, ... dans les projets
- * **Alt + O** : Affichage de la source de l'objet (via la Javadoc)
- * **Alt + G** : Se positionner sur la déclaration de la variable (méthode) où le curseur est positionné
- * **Ctrl + Maj + F** : Mise en forme du code (vous pouvez sélectionner une zone de code pour restreindre le formatage)
- * **Ctrl + T / Ctrl + D** : Indentation/ Désindentation du code (vous pouvez sélectionner plusieurs lignes pour (dés)indenter toutes ces lignes de code)
- * **Ctrl + E** : Efface la ligne courante
- * **Ctrl + Maj + T / D** : Pour commenter/ Décommenter à l'aide de // les lignes sélectionnées
- * **Ctrl + J suivi de S / E** : Démarré / Arrête l'enregistrement d'une macro

