COLBERT Lycée Numérique	**Licence Technologies Nouvelles et Autonomie de la Personne**	3ème année
	Java : P.O.O. multiplateforme	Travaux pratiques

Programmation multiplateforme avec Java

Objectifs

Rédiger des programmes en langage Java, à l'aide d'une plateforme de développement, pour satisfaire à un cahier des charges.
Utilisation d'outils de génération de code pour réaliser des interfaces homme/machine rapidement et pour gagner en productivité

Compétence visée par ces travaux pratiques

- Réaliser
- Intégrer un module logiciel

Savoirs associés à maitriser à l'issu des travaux pratiques

- Programmation orientée objet
- Langages de programmation
- Outils de génération de code

Matériels et applications utilisés

- Plateforme de développement Eclipse sous Linux

I. Environnement de travail

Eclipse est un EDI : Environnement de Développement Intégré (IDE : Integrated Development Environment), c'est-à-dire un logiciel qui simplifie la programmation en proposant un certain nombre de raccourcis et d'aide à la programmation.

Il a été développé par IBM qui a fait don du code initial en 2001. Il est gratuit et disponible pour la plupart des systèmes d'exploitation.

Un grand nombre de langages de programmation sont pris en charge par cette plateforme, dont le langage Java.

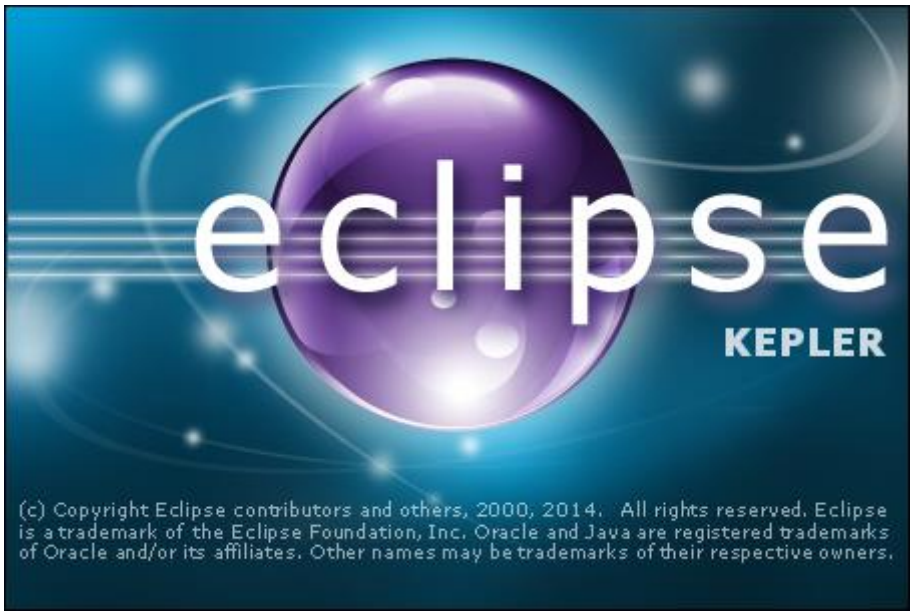
Le développement en Java avec Eclipse nécessite d'utiliser des outils permettant de produire des programmes exécutables.

Nous utiliserons l'outil MinGW (sous Windows), Linux GCC (sous Linux) qui rassemble les éléments permettant d'obtenir une chaîne de compilation cohérente et complète (un compilateur C+, l'outil make et un débogueur).

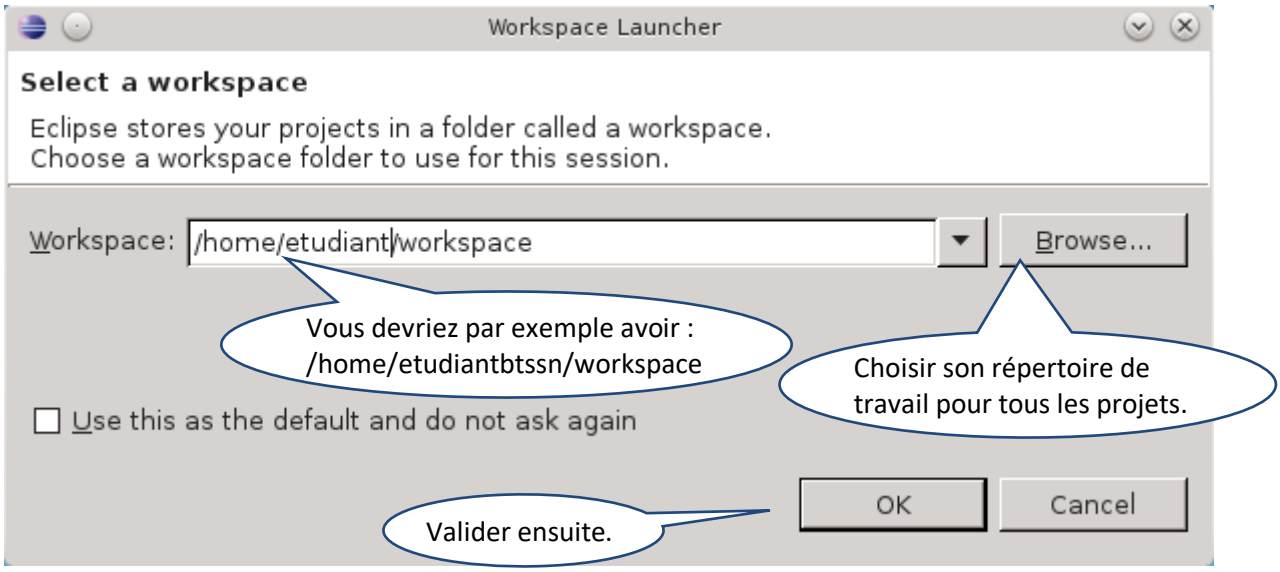
II. Mise en œuvre

Exécuter l'application Eclipse en cherchant l'icone  :

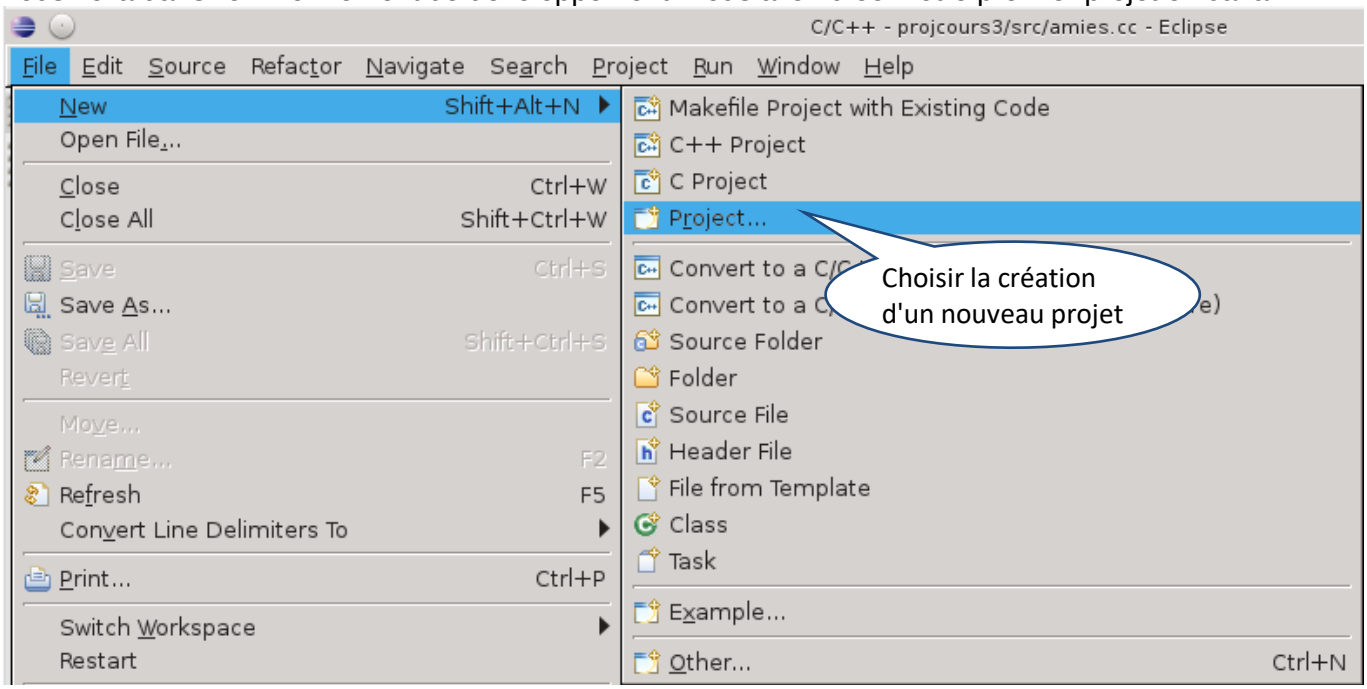
Une fenêtre s'affiche équivalente à celle ci-dessous :



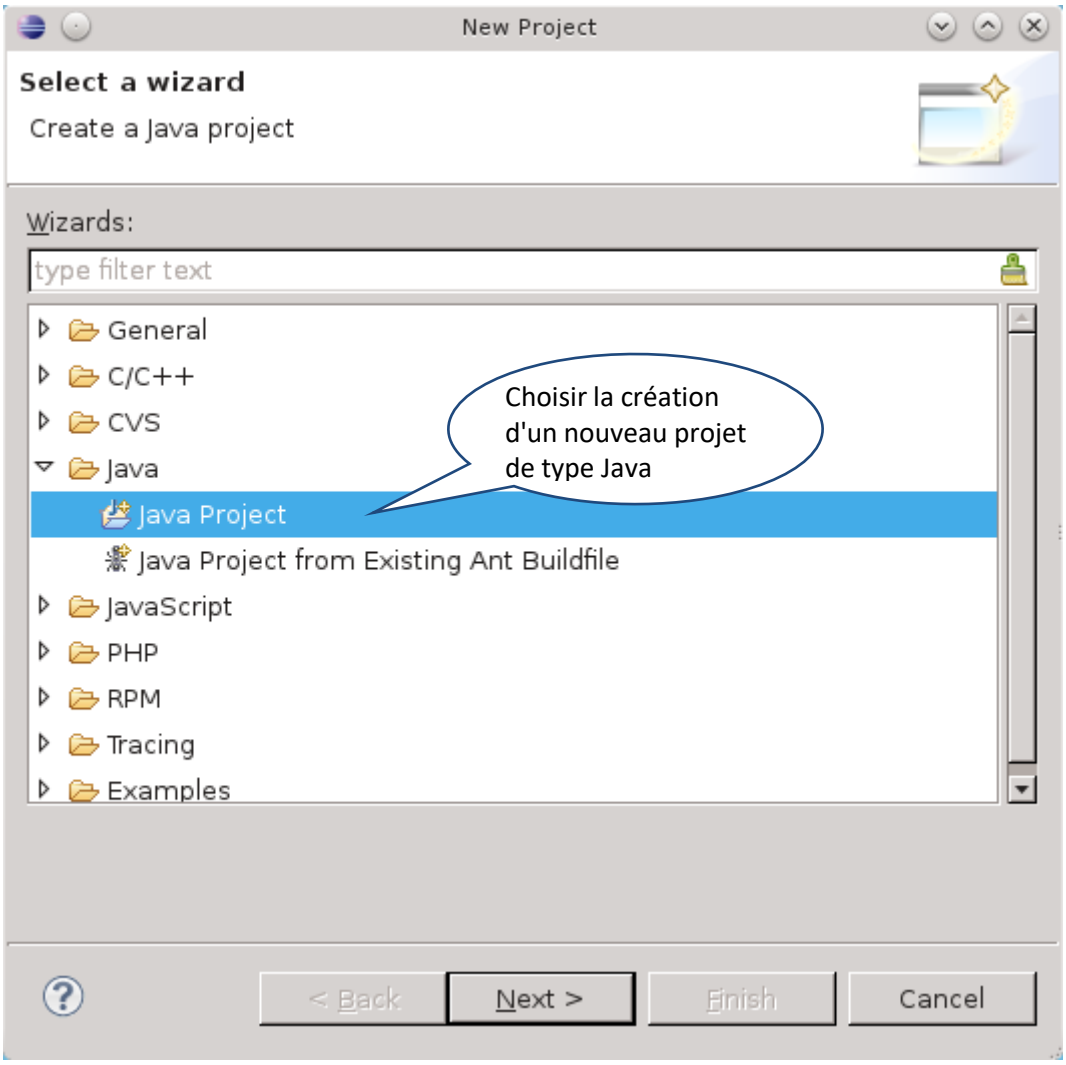
Une autre fenêtre apparaît comme celle ci-dessous pour demander votre répertoire de travail,:



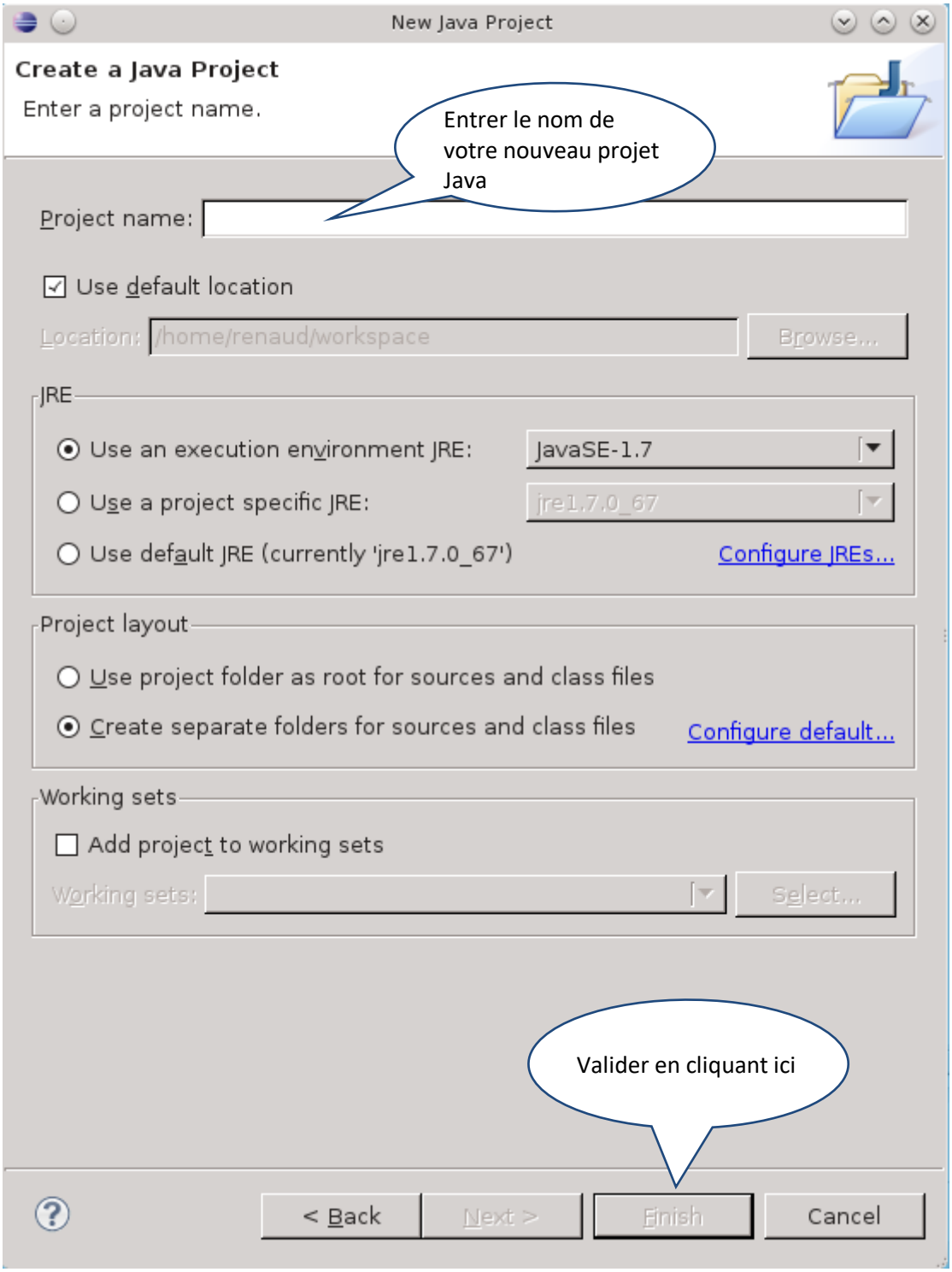
Vous voila dans l'environnement de développement. Vous allez créer votre premier projet en Java

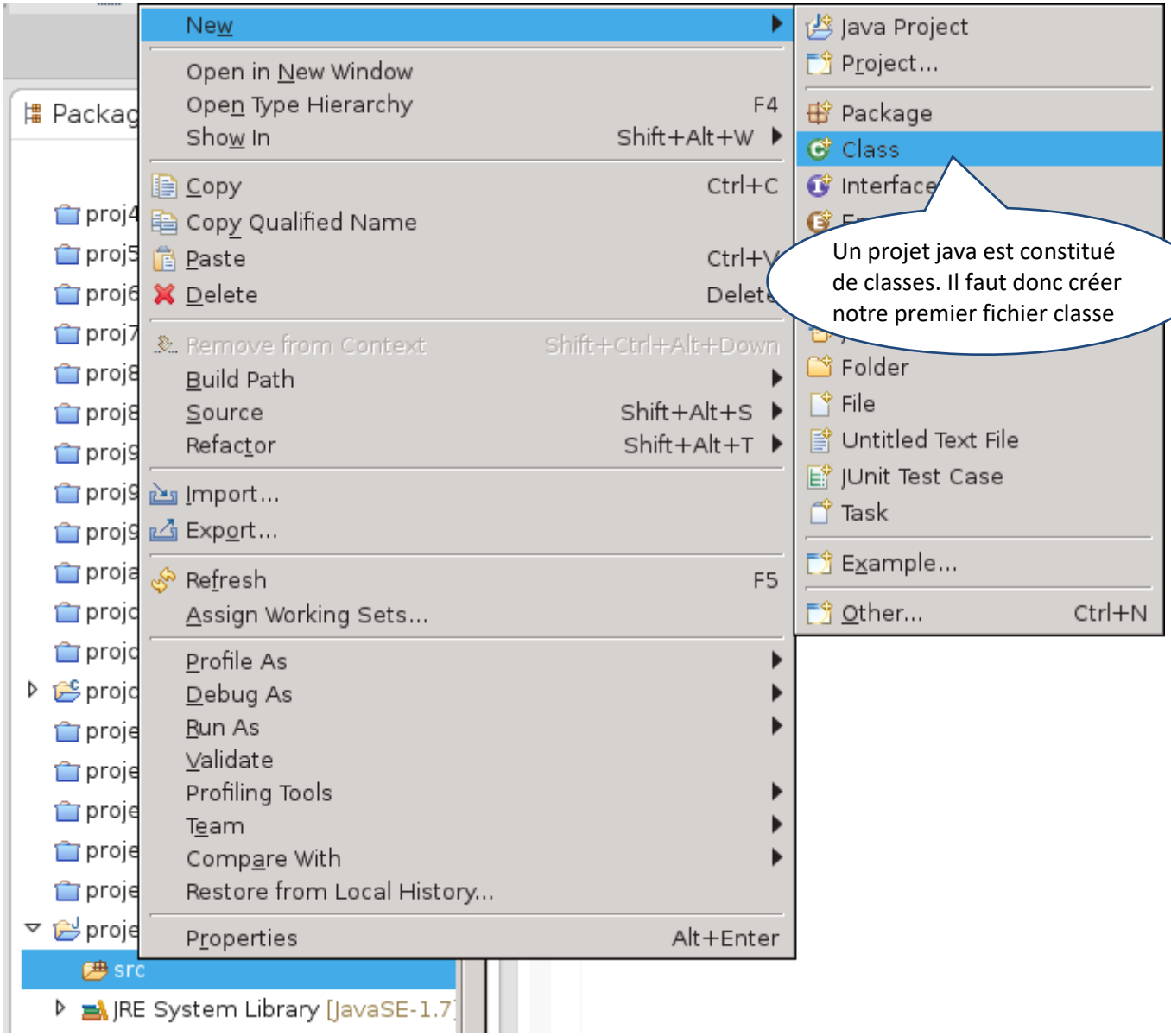


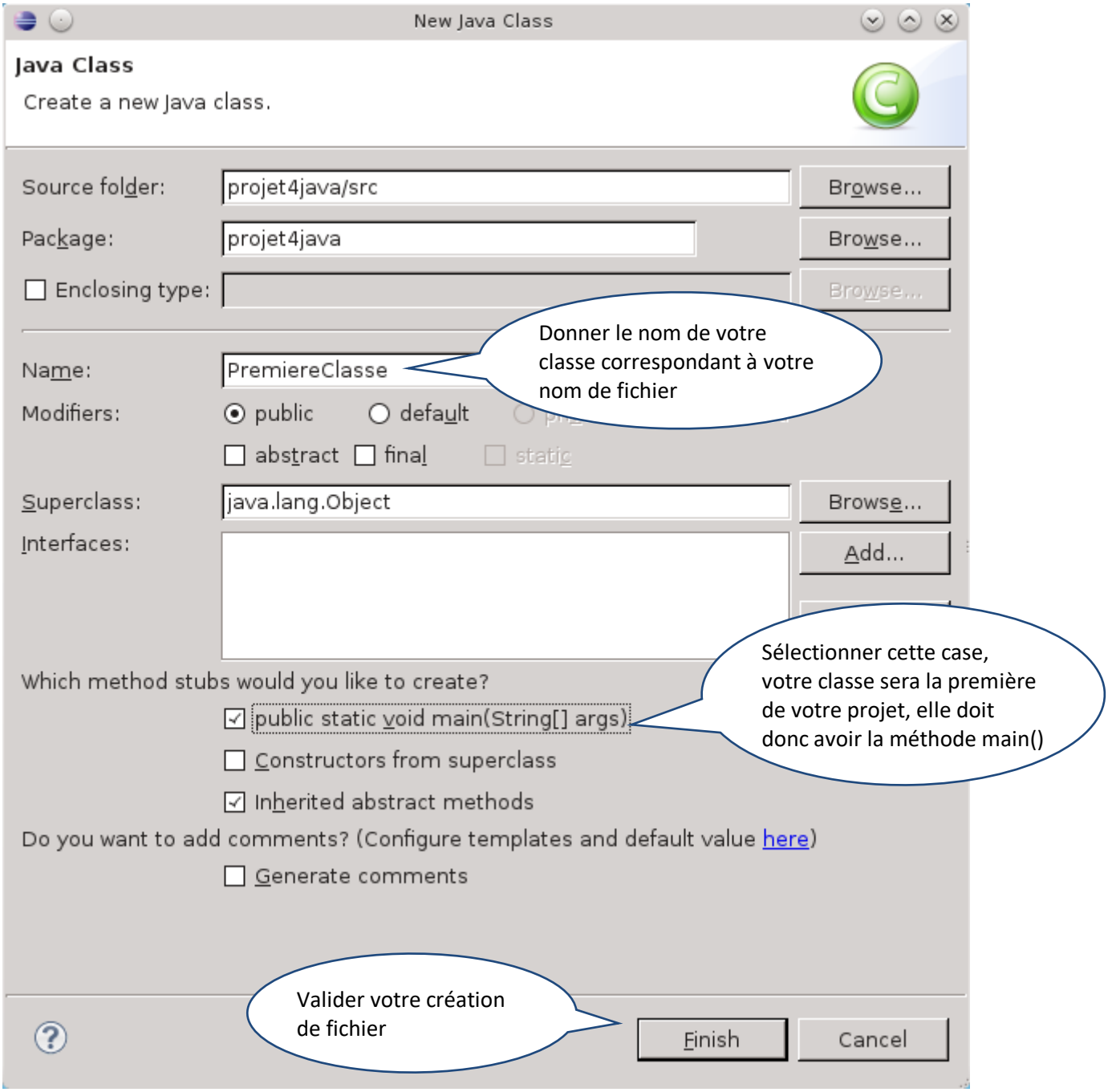
Sélectionner le type de projet Java parmi ceux possibles à développer avec Eclipse. Si vous ne voyez pas le projet Java, il faudra alors ajouter le plug-in projet java à Eclipse.



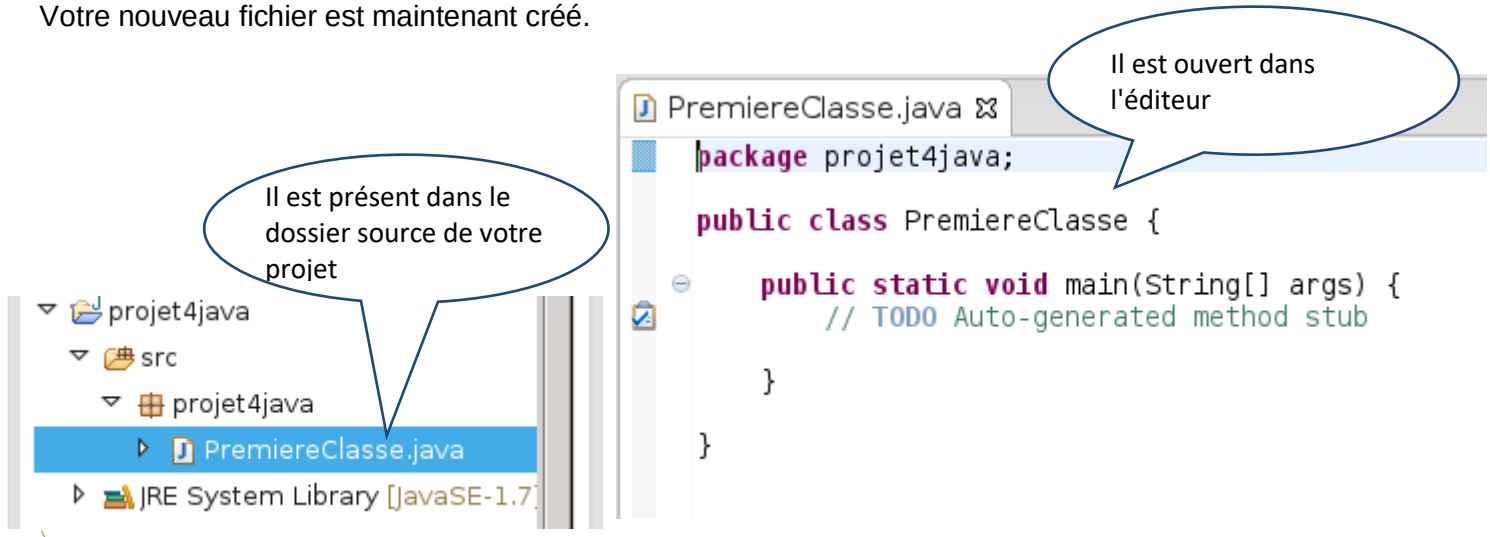
Il faut maintenant choisir le nom du projet.




New
Open in New Window
Open Type Hierarchy F4
Show In Shift+Alt+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Shift+Ctrl+Alt+Down
Build Path
Source Shift+Alt+S
Refactor Shift+Alt+T
Import...
Export...
Refresh F5
Assign Working Sets...
Profile As
Debug As
Run As
Validate
Profiling Tools
Team
Compare With
Restore from Local History...
Properties Alt+Enter
Java Project
Project...
Package
Class
Interface
Folder
File
Untitled Text File
JUnit Test Case
Task
Example...
Other... Ctrl+N
Un projet java est constitué de classes. Il faut donc créer notre premier fichier classe
src
JRE System Library [JavaSE-1.7]



Votre nouveau fichier est maintenant créé.



Compléter le fichier source de votre premier programme Java, comme ci-dessous :

```
package projet1java;

public class PremiereClasse {
    public static void main (String[] args){
      System.out.println("Bonjour le monde !");
      }
    }
```

Enregistrer le fichier puis tester son fonctionnement.

Petit rappel sur les classes et les objets :

Les classes peuvent posséder des attributs et des méthodes.

- Les attributs décrivent la classe
- Les méthodes définissent les actions qu'une classe peut effectuer

Les objets sont des éléments qui respectent les définitions de classes, "créer une instance d'un objet" signifie créer une copie de cet objet dans la mémoire de l'ordinateur en respectant la définition de sa classe.

III. Un projet avec plusieurs classes

Nous allons maintenant créer une classe AnimalDomestique contenant certaine méthodes et attributs. Celle-ci sera un fichier à part entière ; il ne devra pas inclure une fonction main().

Créer un nouveau projet.

Créer un nouveau fichier de type classe nommé AnimalDomestique sans fonction main.

Recopier la classe AnimalDomestique ci-dessous contenant les actions que l'objet devra effectuer.

```
package proj2AnimalJava;

public class AnimalDomestique {
  int age;                  // attribut age de type integer
  float poids,tailles;      // attributs poids et tailles de type float
  String couleur;           // attribut couleur de type string

  // méthode affichant un message
  public void dormir(){
        System.out.println("Je suis fatigué, je vais me reposer.");
        }

  // méthode affichant un message
  public void manger(){
        System.out.println("J'ai besoin de me nourrir ...");
        }

  // méthode renvoyant une chaine de caractères
  // prenant comme argument une chaine de caractères
  public String dire(String unMot){
        String reponseAnimal = "OK !! OK !! " + unMot;
        return reponseAnimal;
        }
  }
```

Créer un nouveau fichier de type class nommé MaitreAnimal avec fonction main.

Recopier la classe MaitreAnimal ci-dessous contenant l'instanciation de la classe AnimalDomestique, ainsi que l'appelle aux méthodes associées.

```
package proj2AnimalJava;

public class MaitreAnimal{
  // méthode main instanciant la classe AnimalDomestique
  // utilisant les méthodes de la classe AnimalDomestique
  public static void main (String[] args){
  AnimalDomestique monAnimal = new AnimalDomestique();
  monAnimal.manger();
  System.out.println(monAnimal.dire("Ouaf !! Ouaf !!"));
  monAnimal.dormir();
  }
}
```

Compiler puis exécuter le projet complet.

Les résultats affichés sont-ils ceux attendus ?

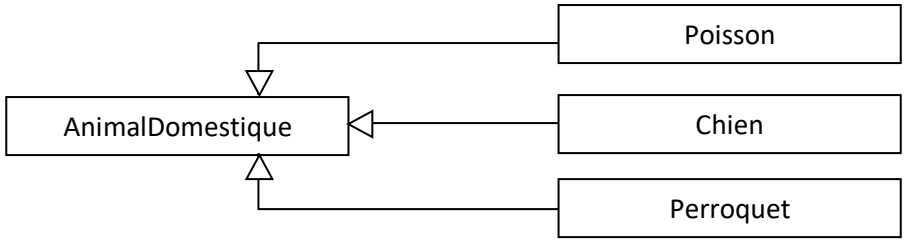
A quoi sert la ligne d'instruction `AnimalDomestique nomAnimal = new AnimalDomestique();` ?

Héritage de classe

Notre classe AnimalDomestique va nous aider à découvrir un autre concept important de Java, appelé *héritage*. Dans la vie réelle, chaque personne hérite des caractéristiques de l'un ou l'autre de ses parents. De la même façon, dans le monde Java, tu peux, à partir d'une classe, en créer une nouvelle.

La classe AnimalDomestique possède un comportement et des attributs partagés par de nombreux animaux familiers -ils mangent, dorment, certains d'entre eux émettent des bruits, leurs peaux peuvent être de différentes couleurs, etc

D'un autre côté, les animaux domestiques sont différents les uns des autres - les chiens aboient, les chat miaules, les perroquets parlent mieux que les chiens. Mais tous mangent, dorment, ont un poids et une taille. C'est pourquoi il est plus facile de créer une classe Poisson qui *héritera* certains comportements et attributs communs de la classe AnimalDomestique, que de créer Chien, Perroquet ou Poisson à partir de rien à chaque fois.



Le mot-clé spécial extends est là pour ça :

```
class Poisson extends AnimalDomestique {
}
```

On peut dire que notre Poisson est une *sous-classe* (*subclass*) de la classe AnimalDomestique et que la classe AnimalDomestique est une *superclasse* (*superclass*) de la classe Poisson. Autrement dit, on utilise la classe AnimalDomestique comme un modèle pour créer la classe Poisson.

Même si l'on se contente de laisser la classe Poisson telle qu'elle est, on peut toujours utiliser chacun des attributs et méthodes hérités de la classe AnimalDomestique. :

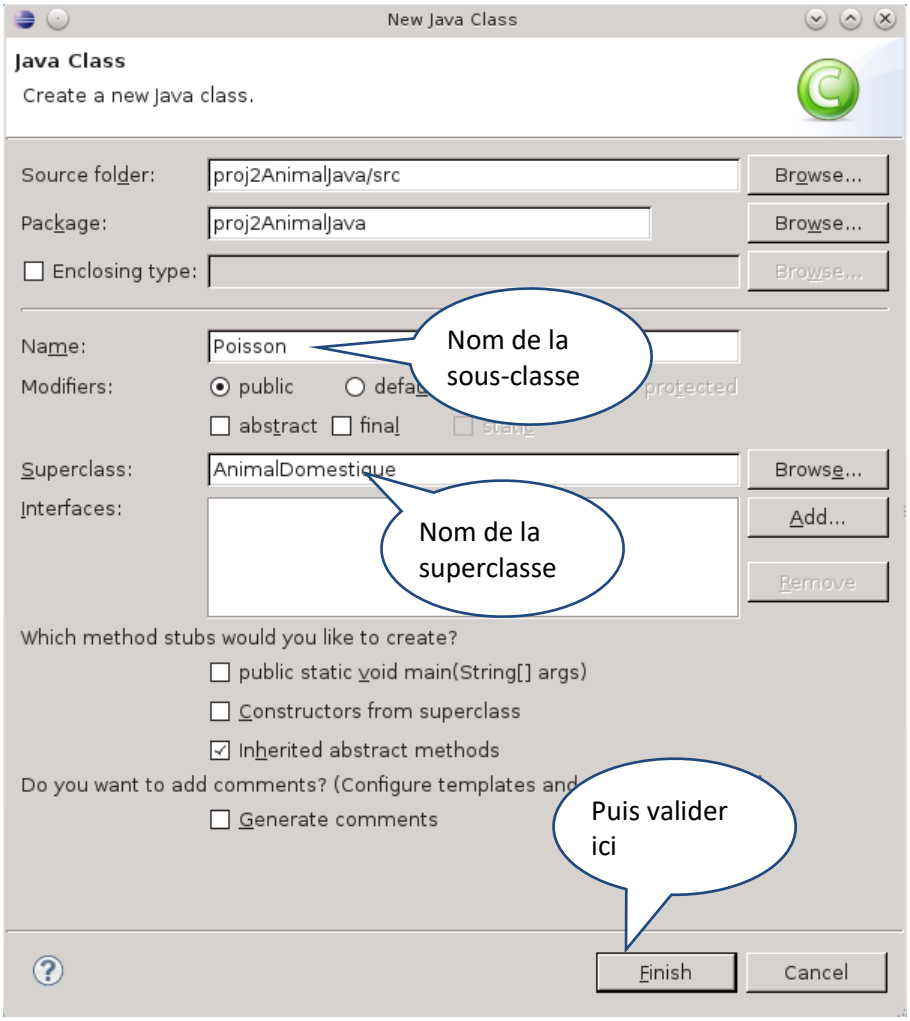
```
Poisson monPoisson = new Poisson();
monPoisson.dormir();
```

Même si nous n'avons pas encore déclaré de méthode dans la classe Poisson, nous avons le droit d'appeler la méthode dormir() de sa superclasse !

Ajoutons maintenant des propriétés à notre classe Poisson.

Pour ajouter une classe qui hérite des propriétés d'une autre, il faut l'indiquer comme ci-dessous dans éclipse :

1- Bouton droit de votre souris pour ajouter une nouvelle classe
2- Compléter les champs appropriés



Vous obtenez alors les premières lignes de votre sous-classe comme ci-après :

```
package proj2AnimalJava;

public class Poisson extends Animaldomestique {
```

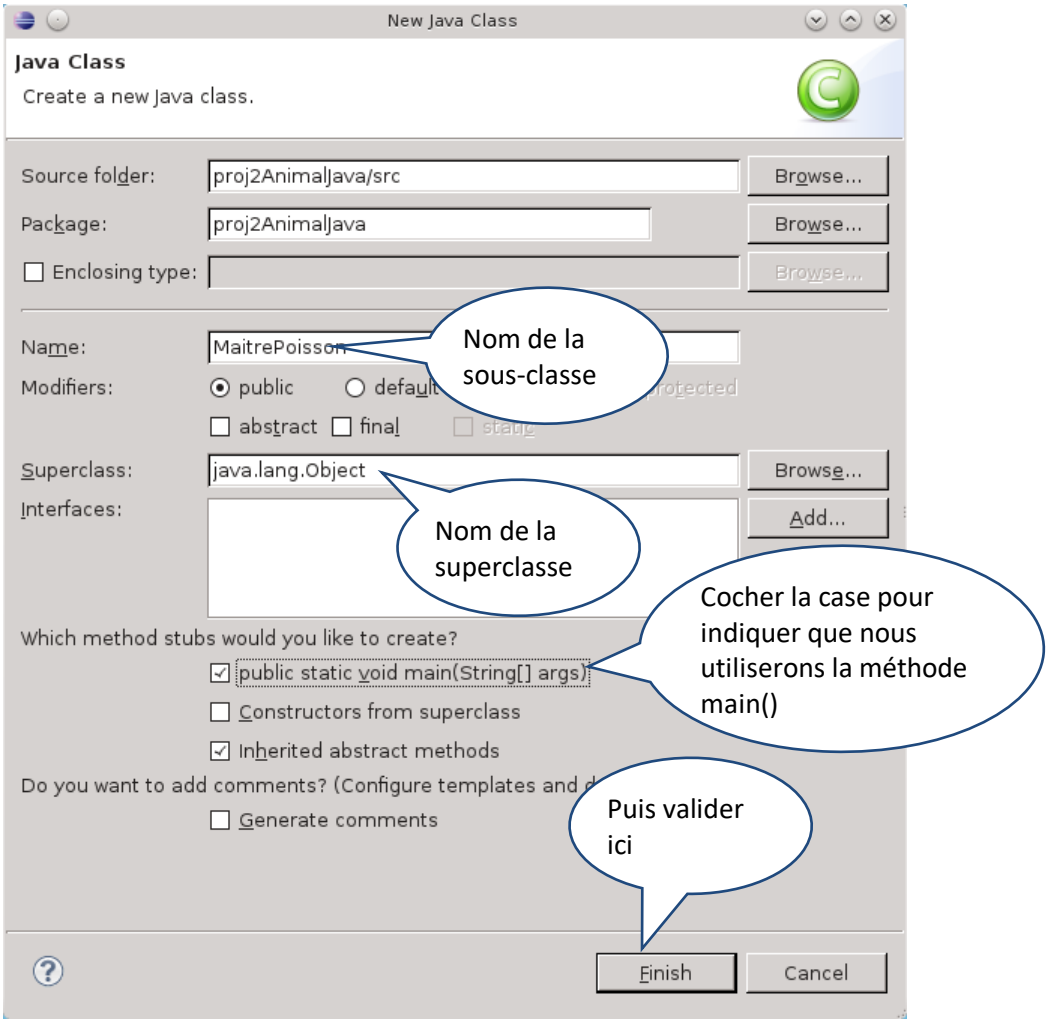
```
}
```

Recopier les méthodes et attribut de cette nouvelle classe comme ci-dessous :

```
package proj2AnimalJava;

public class Poisson extends Animaldomestique {

    int profondeurActuelle = 0; // initialisation de l'attribut

    // méthode ajoutant la valeur passée en argument
    // à l'attribut profondeurActuelle, puis l'affichant à l'écran
    // et retournant en sortie de méthode la valeur profondeurActuelle
    public int plonger (int combienDePlus){
      profondeurActuelle = profondeurActuelle + combienDePlus;
      System.out.println("Plongée de " + combienDePlus + " mètres");
      System.out.println("Je suis à " + profondeurActuelle + " mètres sous le
    niveau de la mer");
      return profondeurActuelle;
      }
    }
```

La méthode plonger() a un *argument* combienDePlus qui indique au poisson de combien il doit plonger. Nous avons aussi déclaré la variable de classe profondeurActuelle qui enregistre la nouvelle profondeur courante à chaque fois que la méthode plonger() est appelée. Cette méthode renvoie la valeur courante de la variable profondeurActuelle à la classe appelante.

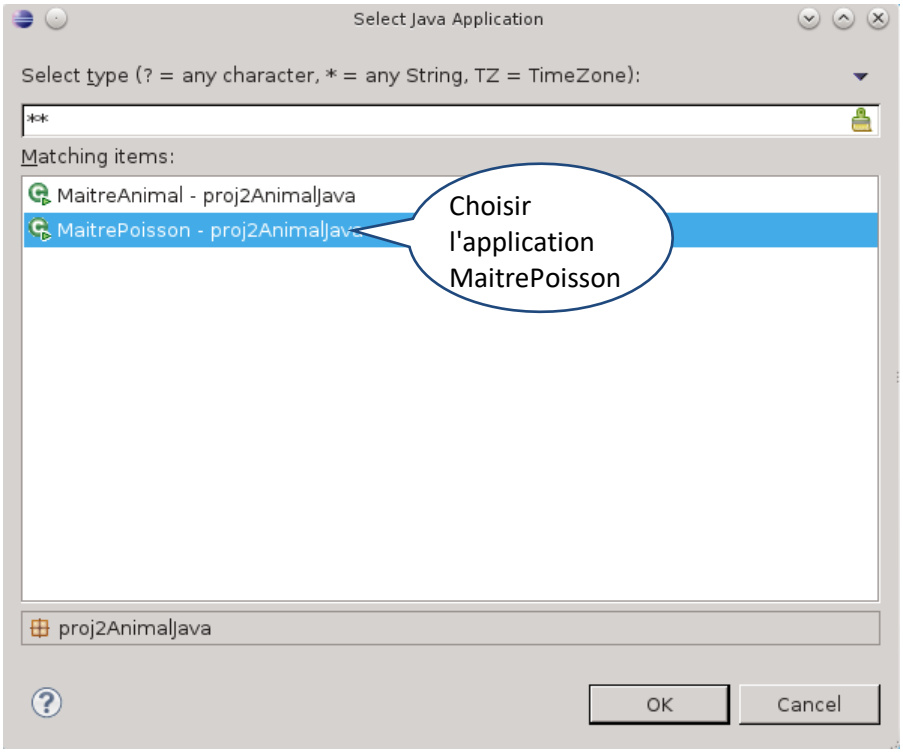
Créer maintenant une autre classe nommée MaitrePoisson qui ressemble à ceci :



Compléter ensuite le reste de la classe comme indiqué ci-dessous :

```
package proj2AnimalJava;

public class MaitrePoisson{

  public static void main (String[] args){
  Poisson monPoisson = new Poisson(); // instancie la classe Poisson
  monPoisson.plonger(2); // appel de la méthode plonger
  monPoisson.plonger(3);
  monPoisson.dormir();   // appel de la méthode dormir
  }
}
```

Au moment d'exécuter le programme, il faut choisir qu'elle classe contenant la méthode main() exécuter, comme indiqué dans la fenêtre qui s'ouvre à chaque exécution du projet :



La méthode main() instancie l'objet Poisson et appelle sa méthode plonger() deux fois, avec des arguments différents. Ensuite, elle appelle la méthode dormir().

La classe MaîtrePoisson appelle aussi des méthodes de sa superclasse AnimalDomestique, alors qu'elle ne fait pas partie de la classe AnimalDomestique. C'est tout l'intérêt de l'héritage : il n'y a pas besoin de copier et coller le code de la classe AnimalDomestique, il suffit d'utiliser le mot extends et la classe Poisson peut utiliser les méthodes d' AnimalDomestique !

Redéfinition d'une méthode

Les poissons ne parlent pas. Mais notre classe Poisson hérite de la classe AnimalDomestique qui possède la méthode dire(). Ceci signifie que rien n'empêche d'écrire une instruction comme celle-ci :

```
monPoisson.dire("Un poisson qui parle !");
```

Eh bien, notre poisson a commencé à parler… Pour éviter que cela se produise, il faut que la classe Poisson redéfinisse la méthode dire() de la classe AnimalDomestique. Pour ce faire, il faut qu'une méthode ait exactement la même signature dans la sous-classe que dans la superclasse, la méthode de la sous-classe sera alors utilisée à la place de celle de la superclasse.

Ajouter la méthode dire() à la classe Poisson, indiqué dans les pointillés :

```
package proj2AnimalJava;

public class Poisson extends Animaldomestique {
   int profondeurActuelle = 0;

   public int plonger (int combienDePlus){
     profondeurActuelle = profondeurActuelle + combienDePlus;
     System.out.println("Plongée de " + combienDePlus + " mètres");
     System.out.println("Je suis à " + profondeurActuelle + " mètres sous le
   niveau de la mer");
     return profondeurActuelle;
     }

   public String dire(String unMot) {
     return "Ne sais-tu pas que les poissons ne parlent pas ?";
     }

   }
```

Ajouter maintenant l'appel suivant dans la méthode main() de la classe MaitrePoisson, comme écrit ci-dessous :

```
package proj2AnimalJava;

public class MaitrePoisson{
   public static void main (String[] args){
         Poisson monPoisson = new Poisson();
         monPoisson.plonger(2);
         monPoisson.plonger(3);
         monPoisson.dormir();

         String reactionPoisson;
         reactionPoisson = monPoisson.dire("Salut");
         System.out.println (reactionPoisson);

         }
   }
```

Exécuter le programme MaitrePoisson

Que constatez-vous ?

Les constructeurs de classe

Comme en langage C++, l'opérateur **new** permet de créer des instances d'objets en mémoire. Par exemple :

```
Poisson monPoisson = new Poisson();
```

Les parenthèses après le mot Poisson signifient que cette classe a une méthode nommée Poisson(), appelée constructeur et qui a les caractéristiques suivantes :

- Les constructeurs ne sont appelés qu'une fois au cours de la construction d'un objet en mémoire.
- Ils doivent avoir le même nom que la classe elle-même.
- Ils ne retournent pas de valeur ; il n'est même pas nécessaire d'utiliser le mot-clé void dans la signature d'un constructeur.

Toute classe peut avoir plusieurs constructeurs. Si aucun constructeur n'est créé pour une classe, Java le crée automatiquement, lors de la compilation, sans argument par défaut.

C'est pour cette raison que le compilateur Java n'a jamais réclamé une déclaration permettant d'écrire new Poisson(), alors que la classe Poisson ne définit aucun constructeur.
En général, les constructeurs sont utilisés pour affecter des valeurs initiales aux variables membres d'une classe.

Ajouter une méthode constructeur pour donner la profondeur initiale de votre classe Poisson

```
package proj2AnimalJava;
public class Poisson extends Animaldomestique {
   int profondeurActuelle;

   Poisson(int pronfondeurDepart) {
     profondeurActuelle = profondeurDepart;
     }

   public int plonger (int combienDePlus){
     profondeurActuelle = profondeurActuelle + combienDePlus;
     System.out.println("Plongée de " + combienDePlus + " mètres");
     System.out.println("Je suis à " + profondeurActuelle + " mètres sous le
   niveau de la mer");
     return profondeurActuelle;
     }
   public String dire(String unMot) {
     return "Ne sais-tu pas que les poissons ne parlent pas ?";
     }
   }
```

Mettre une valeur dans l'appel de la méthode constructeur :

```
   package proj2AnimalJava;

   public class MaitrePoisson{
     public static void main (String[] args){
           Poisson monPoisson = new Poisson(20);
           monPoisson.plonger(2);
           monPoisson.plonger(3);
           monPoisson.dormir();
           String reactionPoisson;
           reactionPoisson = monPoisson.dire("Salut");
           System.out.println (reactionPoisson);
           }
     }
```

La classe MaitrePoisson peut créer une instance de Poisson et affecter la position initiale du poisson.

Si un constructeur avec arguments est défini dans une classe, il n'est plus possible d'utiliser le constructeur sans argument par défaut.

Le mot clé this

Le mot-clé **this** est utile lorsqu'il faut se référer à l'instance de l'objet dans lequel le programme se trouve. Ainsi la classe Poisson pourrait être écrite comme suit :

```
package proj2AnimalJava;
public class Poisson extends Animaldomestique {
   int profondeurActuelle;
   Poisson(int pronfondeurActuelle) {
     this.profondeurActuelle = profondeurActuelle;
     }
   public int plonger (int combienDePlus){
     profondeurActuelle = profondeurActuelle + combienDePlus;
     System.out.println("Plongée de " + combienDePlus + " mètres");
     System.out.println("Je suis à " + profondeurActuelle + " mètres sous le
   niveau de la mer");
     return profondeurActuelle;
     }

   public String dire(String unMot) {
     return "Ne sais-tu pas que les poissons ne parlent pas ?";
     }
   }
```

Le mot-clé **this** permet d'éviter des conflits de nom. Ici dans la classe Poisson, **this**.profondeurActuelle fait référence à l'attribut membre profondeurActuelle, alors que profondeurActuelle fait référence à la variable passée comme argument dans la méthode Poisson.

IV. Intégration en ligne de code d'une interface graphique homme-machine

La particularité de Java réside dans le fait que ce langage peut être exécuté sur n'importe qu'elle machine ou système d'exploitation sur lequel une machine virtuelle Java est installée. Cela en fait un langage hautement disponible et facile à déployer ou utiliser. Il n'est donc plus dédié à une architecture de processeur en particulier et les applications qui sont écrites peuvent donc être utiliser sur toute plateforme qui intègre une machine virtuelle Java.

Pour rendre l'utilisation aussi aisée que possible entre le programme et l'utilisateur, il est souvent plus pratique de permettre un dialogue à l'aide de la souris ou tactile que par des lignes de commandes et le clavier comme élément de dialogue.

Nous allons donc nous intéresser à la création d'interfaces graphiques et à l'utilisation de bibliothèques de classes dédiées.
Nous utiliserons donc la bibliothèque AWT (nécessaire pour des anciennes versions de Java) qui permet de travailler avec du graphique, contenant un ensemble de classes telles que des boutons, des champs textuels, des libellés et bien d'autres encore.
Nous utiliserons aussi la bibliothèque SWING (fonctionnelle sur des versions de Java récente) qui permet d'obtenir également des boutons, des champs textuels et d'autres contrôles mais de manière plus évolué.

⚠ Il est nécessaire d'indiquer au compilateur Java où trouver les bibliothèques des classes utilisées dans le programme. La déclaration **`import`** indique au compilateur quelle bibliothèque ajouter dans le projet.

✎ **Créer** un nouveau projet Interface1 et recopier le code suivant :

Paquetage de bibliothèque standard

```
package proj3InterfaceIHMGraphique;
```

Bilbiothèques graphiques

```
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.FlowLayout;
```

Classe principale

```
public class InterfaceGraphique {
```

Méthode principale

```
  public static void main(String[] args) {
```

Création du panneau

```
    JPanel contenuFenetre = new JPanel();
```

Affectation d'un gestionnaire de disposition au panneau

```
    FlowLayout dispositionFenetre = new FlowLayout();
    contenuFenetre.setLayout(dispositionFenetre);
```

Création des éléments de contrôles

```
    JLabel etiquette = new JLabel("Nombre 1 :");
    JTextField champTexte = new JTextField(10);

    JButton boutonEcriture = new JButton("Ecrire");
```

Ajout des éléments au panneau

```
    contenuFenetre.add(etiquette);
    contenuFenetre.add(champTexte);
    contenuFenetre.add(boutonEcrire);
```

Création du cadre

```
    JFrame cadre = new JFrame("Ma première interface graphique");
```

Ajout du panneau au cadre

```
    cadre.setContentPane(contenuFenetre);
```

Dimensionner et rendre visible

```
    cadre.setSize(400,100);
    cadre.setVisible(true);
```

Lecture d'un évènement lors du clic sure boutonEcrire

```
    InterfaceEvenement interfaceEvenement = new InterfaceEvenement();
    boutonEcrire.addActionListener(interfaceEvenement);
    }
  }
```

Créer une nouvelle classe **`InterfaceEvenement`** dans le projet et recopier le code suivant :

```
package proj3InterfaceIHMGraphique;
```
Paquetage de bibliothèque standard

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
```
Bilbiothèques des évènements

```
import javax.swing.JOptionPane;
import javax.swing.JButton;
```
Bilbiothèques graphique

```
public class InterfaceEvenement implements ActionListener {
```
Classe

```
  public void actionPerformed(ActionEvent evenement){
```
Méthode principale

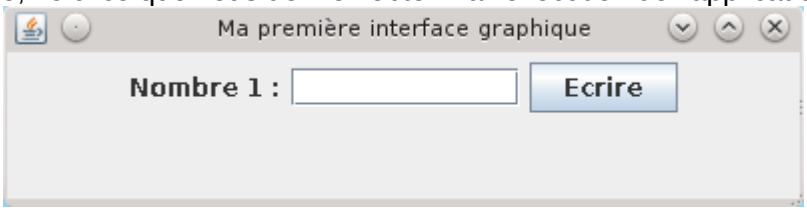
```
    JButton boutonClique = (JButton) evenement.getSource();
```
Lecture de la source de l'évènement

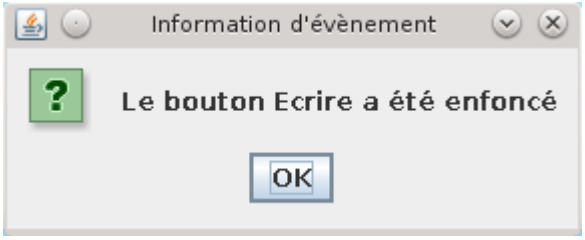
```
    String etiquetteBoutonClique = boutonClique.getText();
```
Lecture de l'étiquette du bouton

```
    JOptionPane.showConfirmDialog(null,
    "Le bouton " + etiquetteBoutonClique + " a été enfoncé",
    "Information d'évènement",
    JOptionPane.PLAIN_MESSAGE);
```
Ajout de texte à la boite de dialogue surgissant.

```
    }
  }
```

Tester le programme, voici ce que vous devriez obtenir à l'exécution de l'application :





Les événements utilisateurs sont gérés par plusieurs interfaces `EventListener`.
Les interfaces `EventListener` permettent de définir les traitements en réponse à des événements utilisateurs généré par un composant. Une classe doit contenir une interface auditrice pour chaque type d'événements à traiter :

- `ActionListener` : clic de souris ou enfoncement de la touche Enter
- `ItemListener` : utilisation d'une liste ou d'une case à cocher
- `MouseMotionListener` : événement de souris
- `WindowListener` : événement de fenêtre

L'ajout d'une interface `EventListener` impose plusieurs ajouts dans le code.

Dans la méthode `actionPerformed`, la méthode `getSource()` de la classe `ActionEvent` est utilisée pour savoir sur quel bouton a appuyé l'utilisateur - la variable événement est une référence à cet objet qui est présente quelque part dans la mémoire vive de l'ordinateur. La documentation Java nous indique que cette méthode retourne la source de l'événement sous la forme d'une instance du type Object, qui est la superclasse de toutes les classes Java, y compris les composants de fenêtre.

C'est fait ainsi de façon à avoir une méthode universelle qui fonctionne avec tous les composants. Dans la fenêtre nous avons pour le moment 1 seul bouton, donc seul le bouton peut être à l'origine de l'événement d'action, c'est pourquoi nous effectuons une conversion de type explicite de l'Object retourné en un `JButton`, en indiquant le type `(JButton)` entre parenthèses devant l'appel de la méthode :

```
JButton boutonClique = (JButton) evenement.getSource();
```

Nous déclarons une variable de type `JButton` à la gauche du signe égale et, bien que la méthode `getSource()` retourne des données du type `Object`, nous précisons à Java que nous convertissons le type Object en type `JButton`, ce qui implique que nous sommes certain d'obtenir une instance de `JButton`.
C'est seulement après avoir effectué la conversion d'`Object` en `JButton` que nous pouvons appeler la méthode `getText()` définie par la classe `JButton`

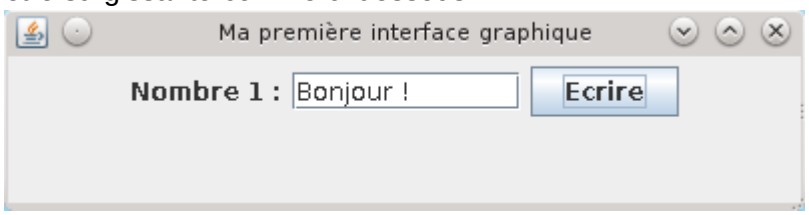
Explication de `showConfirmDialog()`:

```
JOptionPane.showConfirmDialog(null,
  "Le bouton " + etiquetteBoutonClique + " a été enfoncé",
  "Information d'évènement",
  JOptionPane.PLAIN_MESSAGE);
```

Il y a différentes version de la méthode `showConfirmDialog()`. Dans la version utilisée, le mot-clé **null** signifie que cette boîte de message n'a pas de fenêtre mère, le deuxième argument contient le message, le troisième le titre de la boîte de message et le quatrième permet de choisir le(s) bouton(s) à inclure dans la boîte (PLAIN_MESSAGE signifie que seul un bouton OK est affiché).

Nous allons modifier un petit peu le programme pour séparer la méthode constructeur de la classe `InterfaceGraphique` de la méthode `main` contenue dans la classe `InterfaceGraphique`, et ainsi définir des attributs globaux dans cette classe, qui pourront être alors vu par d'autres classes.

Nous allons ainsi pouvoir interagir avec la zone de texte (champTexte), via la classe fille `InterfaceEvenement`, qui normalement n'est pas accessible depuis la classe fille, puisqu'elle fait partie de la classe parent `InterfaceGraphique` . Cette zone de texte sera donc modifiée lors de l'appuie sur le bouton OK de la fenêtre surgissante comme ci-dessous :



Modifier le code précédent de la classe InterfaceGraphique "maitresse" pour obtenir les mêmes lignes de codes comme ci-dessous :

Paquetage de bibliothèque standard

```
package proj3InterfaceIHMGraphique;
```

Bilbiothèques graphique

```
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.FlowLayout;
```

Classe

```
public class InterfaceGraphique {

   JLabel etiquetteTexte;
   JTextField champTexte;
   JButton boutonEcrire;
```

Constructeur de la classe InterfaceGraphique

```
   // constructeur de la classe InterfaceGraphique
   InterfaceGraphique(){
     JPanel contenuFenetre = new JPanel();

     FlowLayout dispositionFenetre = new FlowLayout();
     contenuFenetre.setLayout(dispositionFenetre);

     etiquetteTexte = new JLabel("Nombre 1 :");
     champTexte = new JTextField(10);
     boutonEcrire = new JButton("Ecrire");

     contenuFenetre.add(etiquetteTexte);
     contenuFenetre.add(champTexte);
     contenuFenetre.add(boutonEcrire);

     JFrame cadre = new JFrame("Ma première interface graphique");
     cadre.setContentPane(contenuFenetre);

     cadre.setSize(400,100);
     cadre.setVisible(true);

   // utilisation de this faisant référence aux propriétés
   // de la classe InterfaceGraphique lors d'un évènement
     InterfaceEvenement interfaceEvenement = new InterfaceEvenement(this);
     boutonEcrire.addActionListener(interfaceEvenement);
     }
```

Méthode principale

```
   // méthode main de tout le projet
   public static void main(String[] args) {
     InterfaceGraphique interfaceIHM = new InterfaceGraphique();
     }
```

```
}
```

Paquetage de bibliothèque standard

```
package proj3InterfaceIHMGraphique;
```

Bilbiothèques des évènements

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
```

Bilbiothèques graphique

```
import javax.swing.JOptionPane;
import javax.swing.JButton;
import javax.swing.JTextField;
```

Classe

```
public class InterfaceEvenement implements ActionListener {

  // une référence globale à la classe InterfaceGraphique
  InterfaceGraphique parent;
```

Constructeur de la classe InterfaceEvenement

```
  // Le constructeur stocke la référence à la classe
  // InterfaceGraphique dans la variable membre parent
  InterfaceEvenement(InterfaceGraphique parent) {
  this.parent = parent;
  }
```

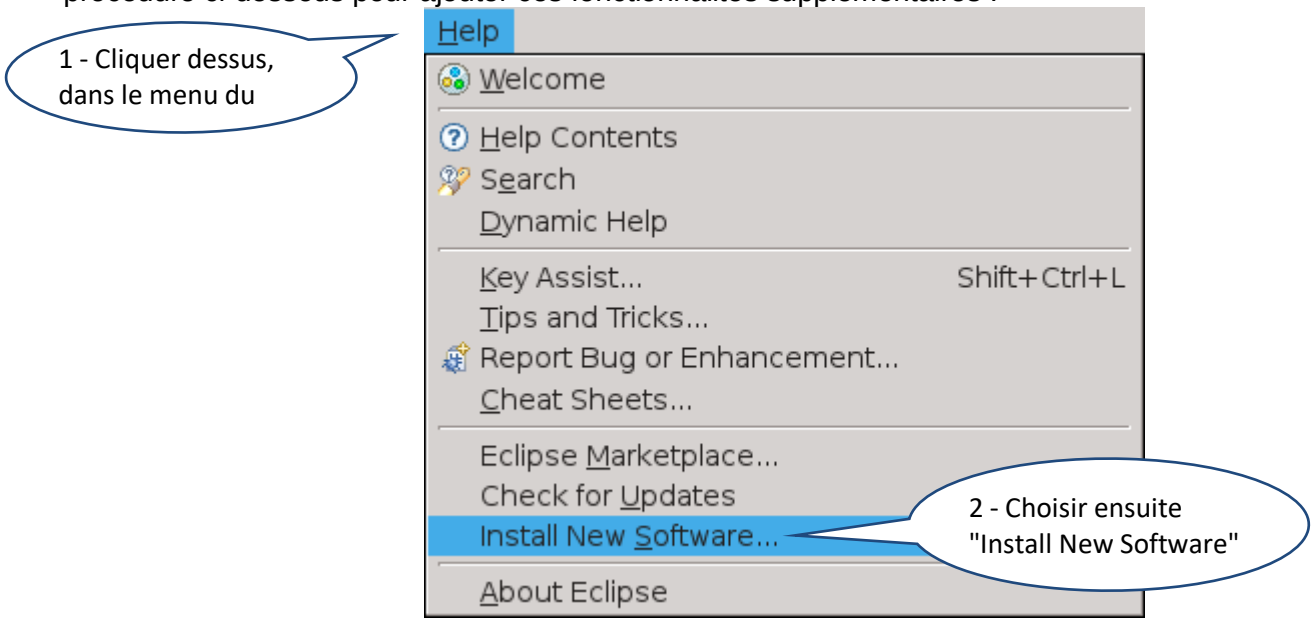
Méthode principale

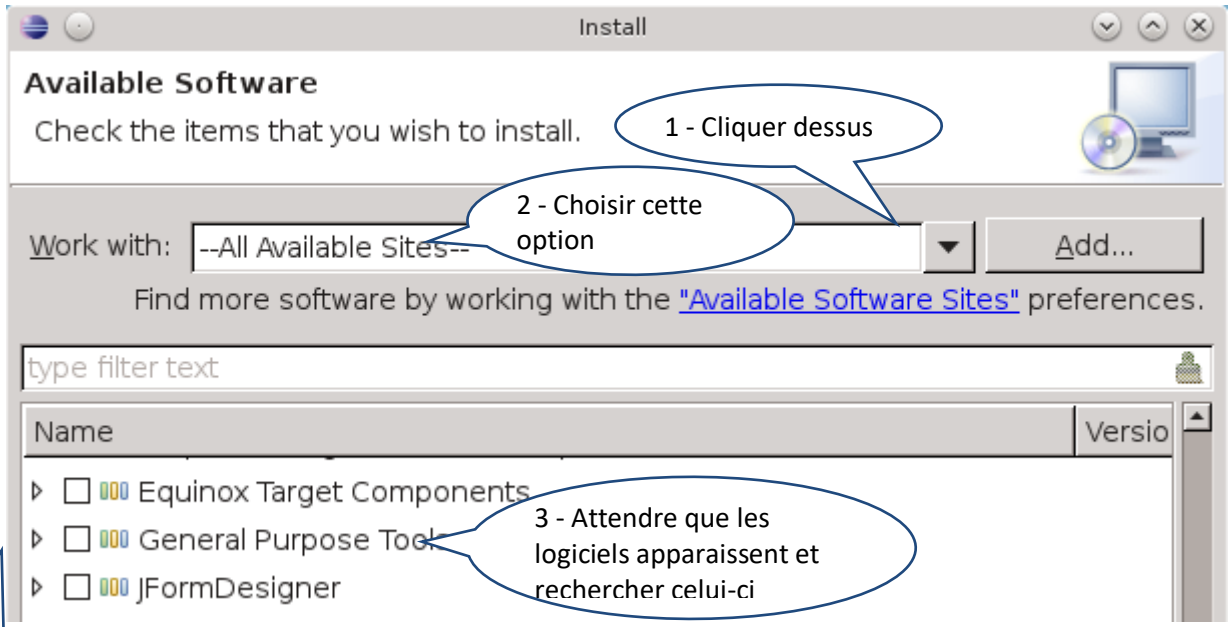
```
public void actionPerformed(ActionEvent evenement){
  JButton boutonClique = null;

  // lecture de la source de l'évènement
  // enregistrement dans une variable de type Object
  Object sourceEvenement = evenement.getSource();
  // si la source est un objet de type JButton
  if (sourceEvenement instanceof JButton){
    // on enregistre la provenance de l'évènement
    // dans la variable boutonClique
    boutonClique = (JButton) sourceEvenement;
    }

  // Retrouve le libellé du bouton
  String libelleBoutonClique = boutonClique.getText();
  // Concatène le libellé du bouton au texte
  // de la boîte de message
  JOptionPane.showConfirmDialog(null,
  "Le bouton " + libelleBoutonClique + " a été enfoncé",
  "Information d'évènement",
  JOptionPane.PLAIN_MESSAGE);

  // référence au champTexte de la classe parent InterfaceGraphique
  parent.champTexte.setText("Bonjour !");
  }

}
```

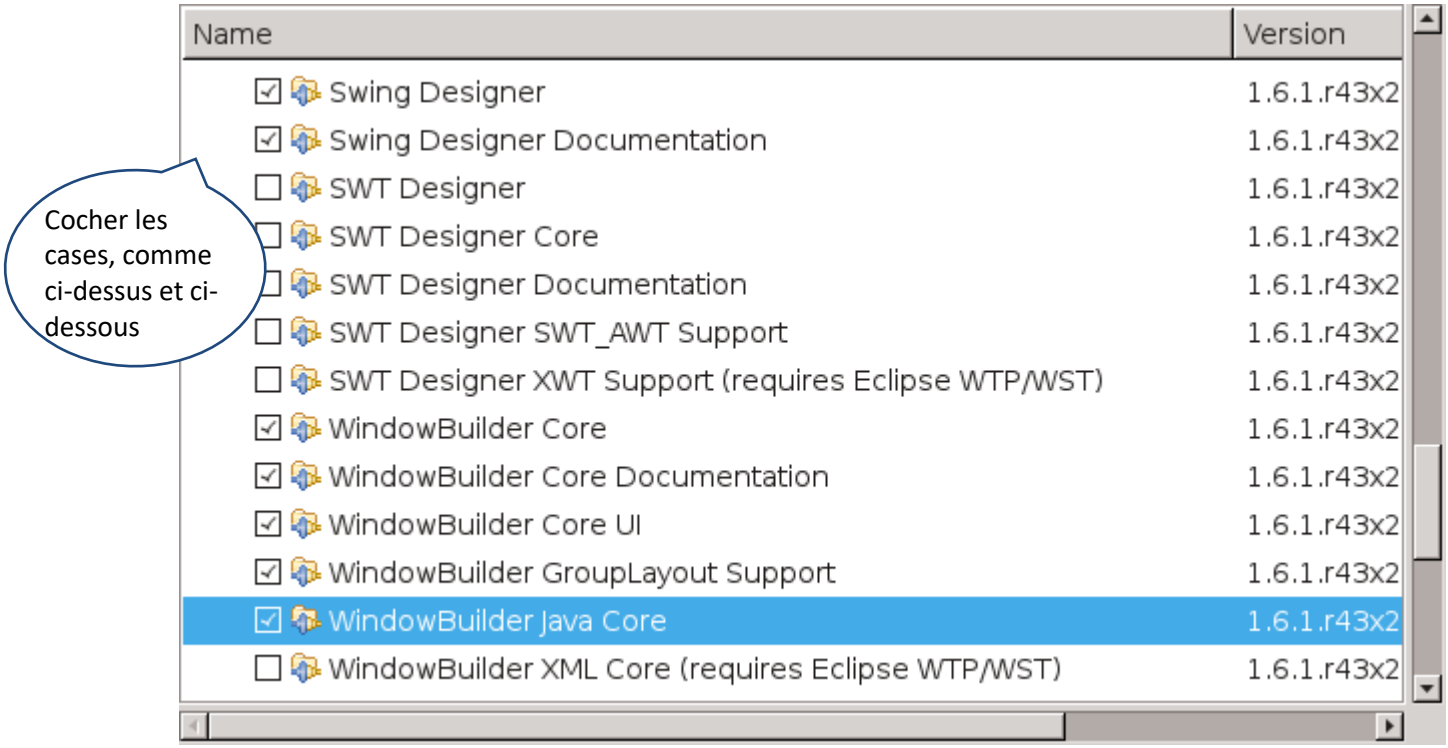
V. *Intégration avec un éditeur graphique d'une interface graphique homme-machine*

L'objectif de cette dernière partie est de présenter le développement d'interface utilisateur graphique (GUI : Graphical User Interface), pour développer des applications ou applets en Java, avec un éditeur graphique en diminuant au maximum l'écriture de code pour toutes les parties graphiques et évènements.

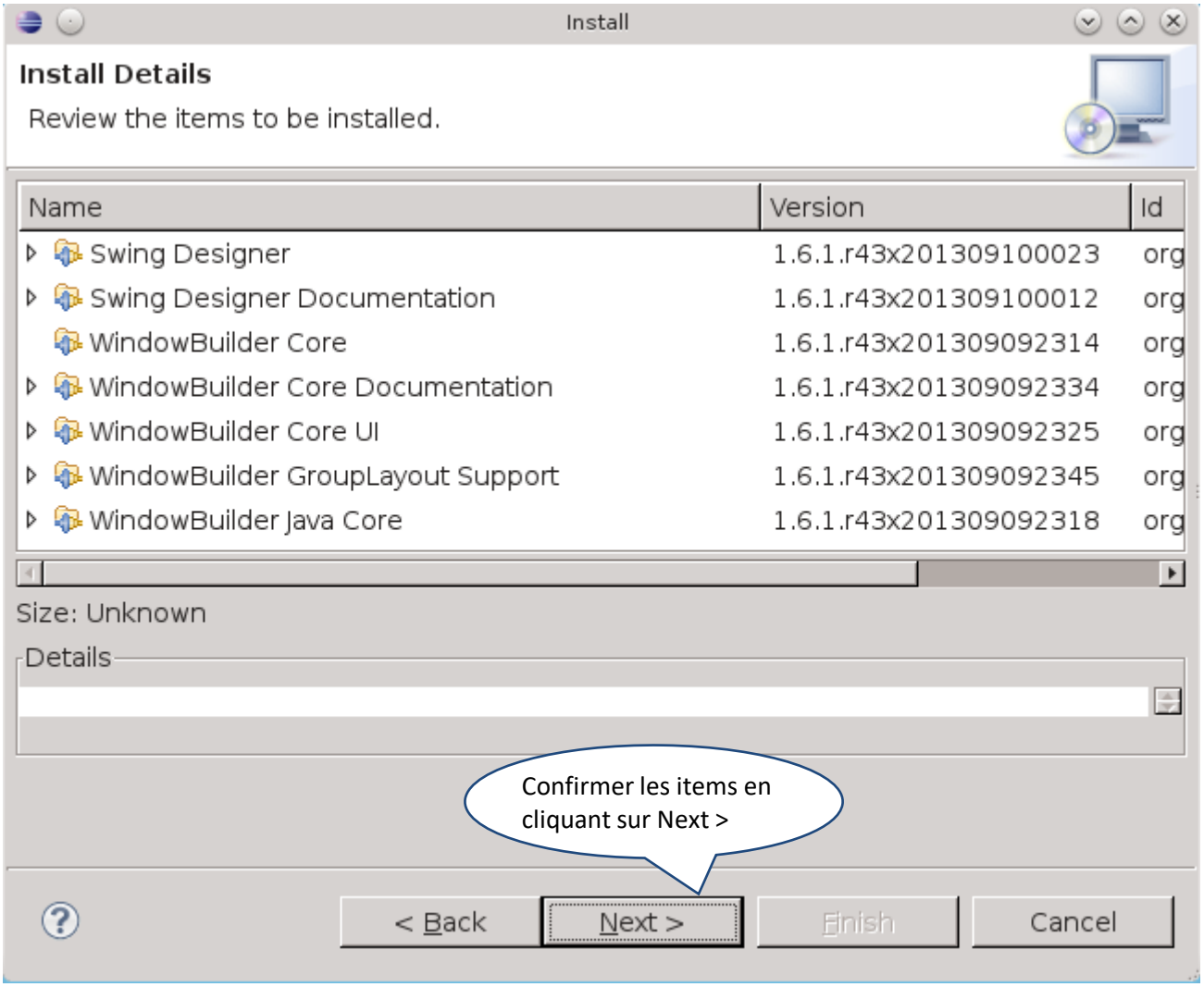
Il faut installer des logiciels supplémentaires à Eclipse qui permettent l'utilisation de tels outils. Suivez la procédure ci-dessous pour ajouter ces fonctionnalités supplémentaires :

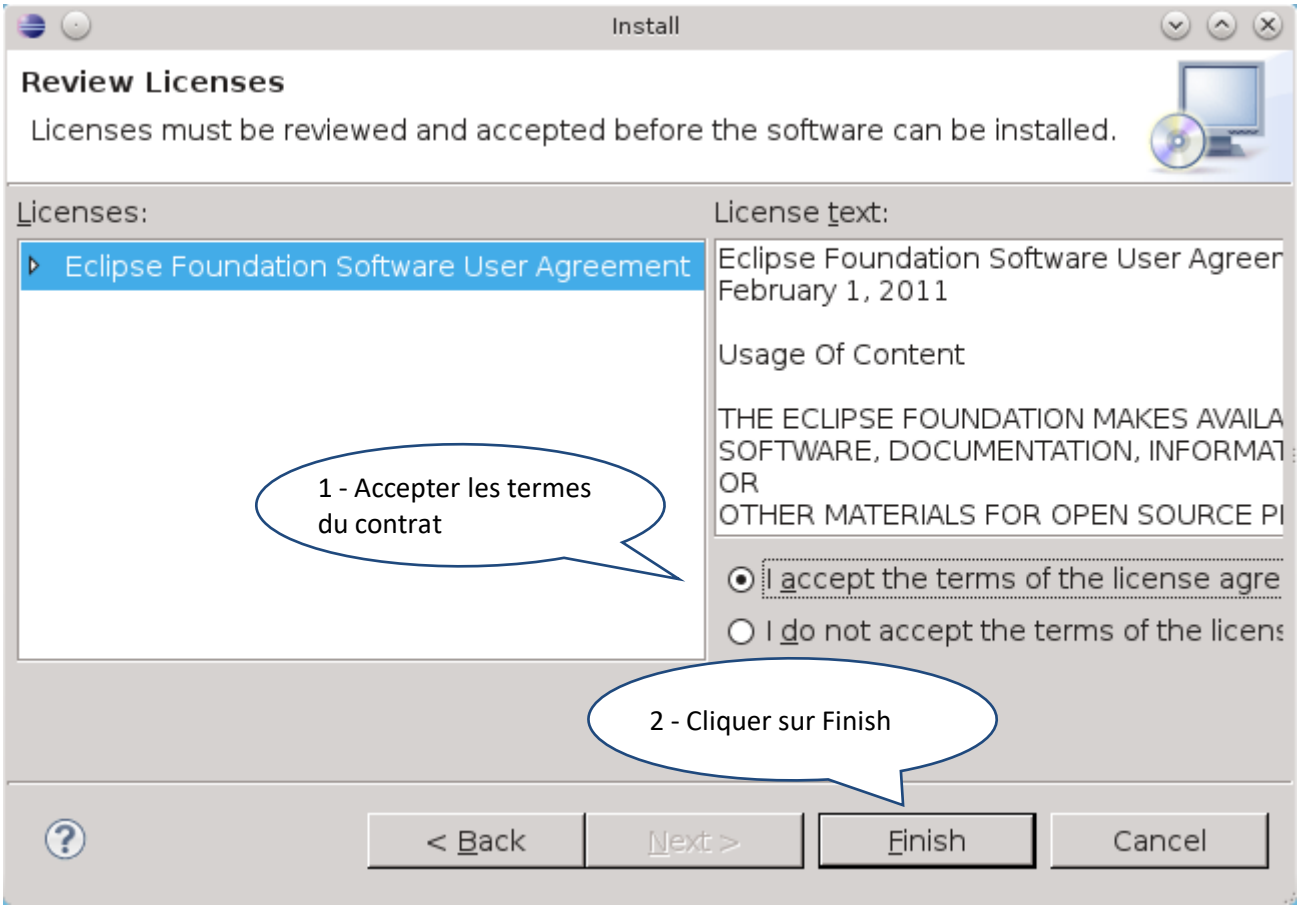


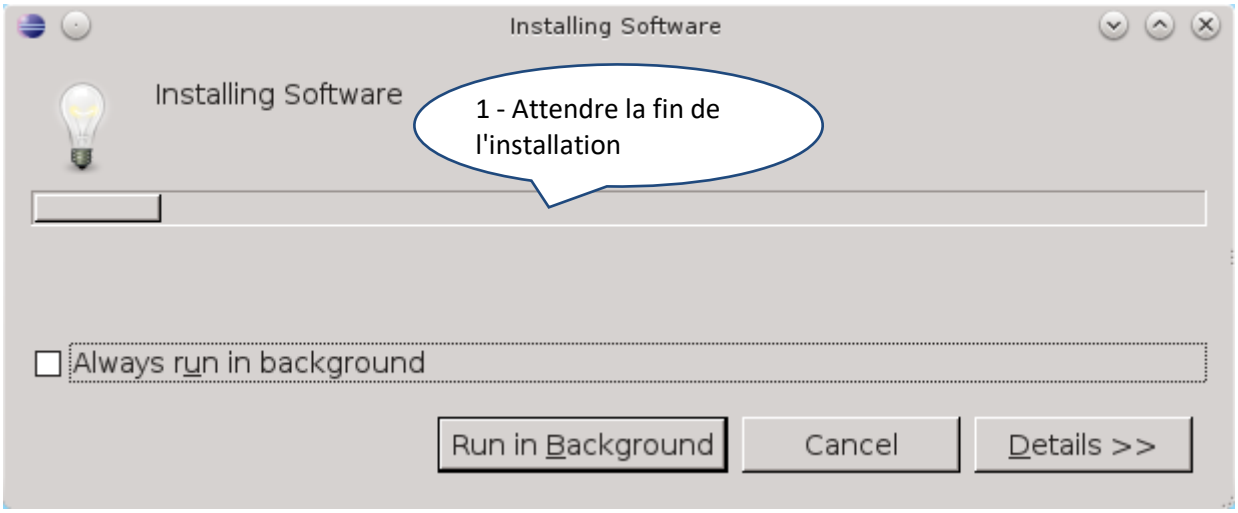


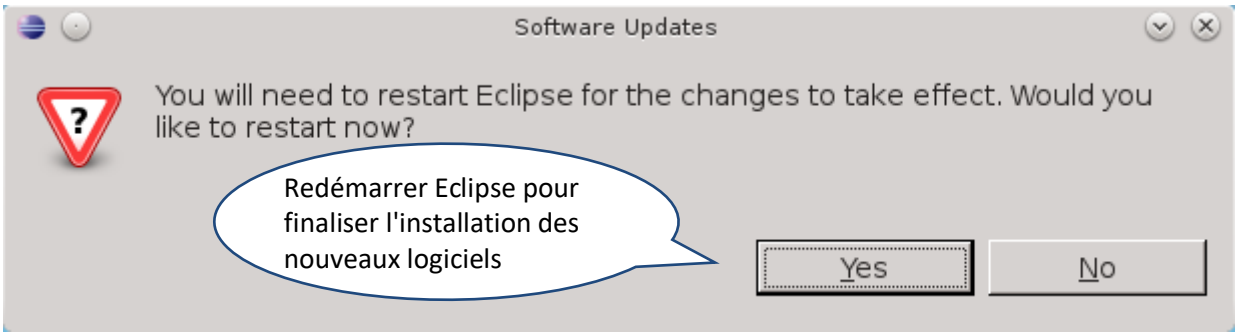


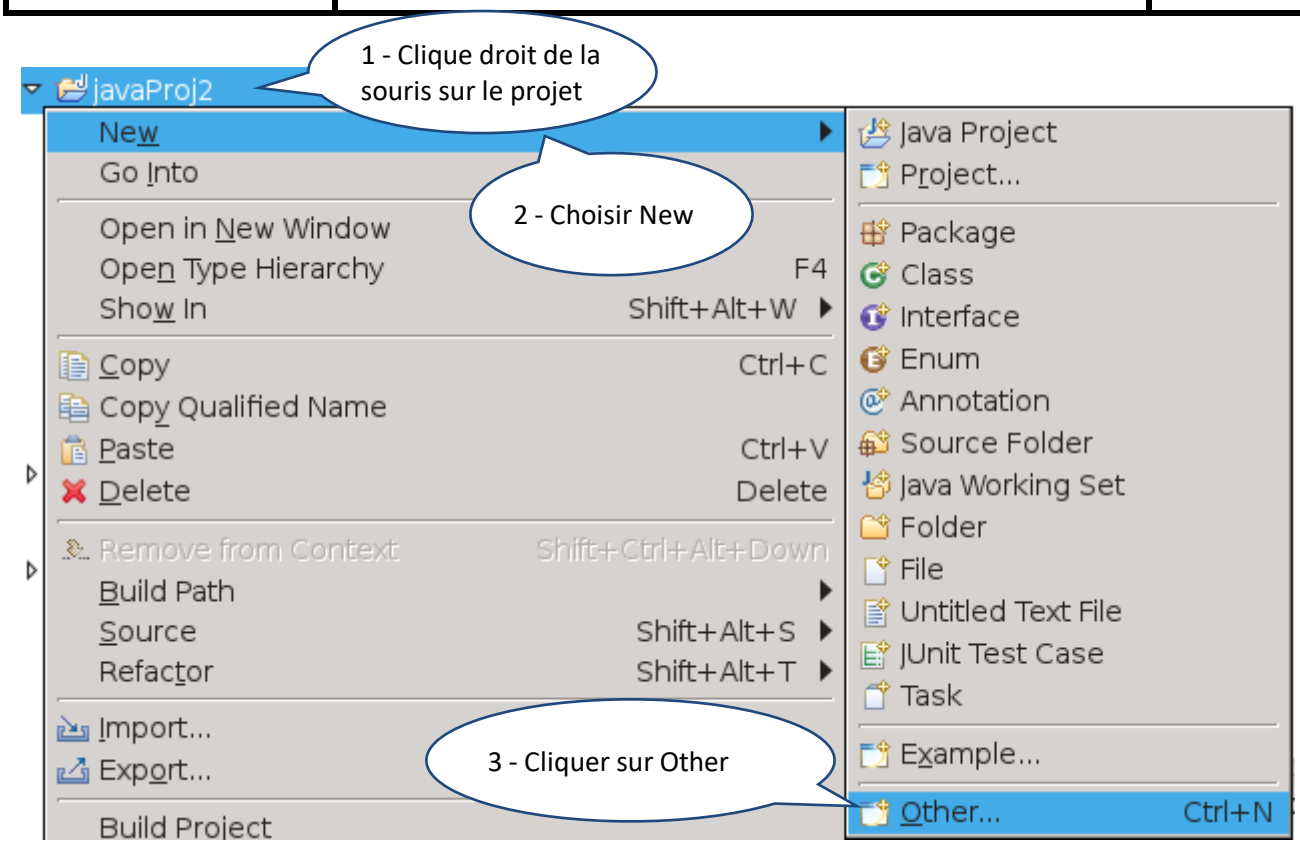
Valider votre choix en cliquant sur le bouton 

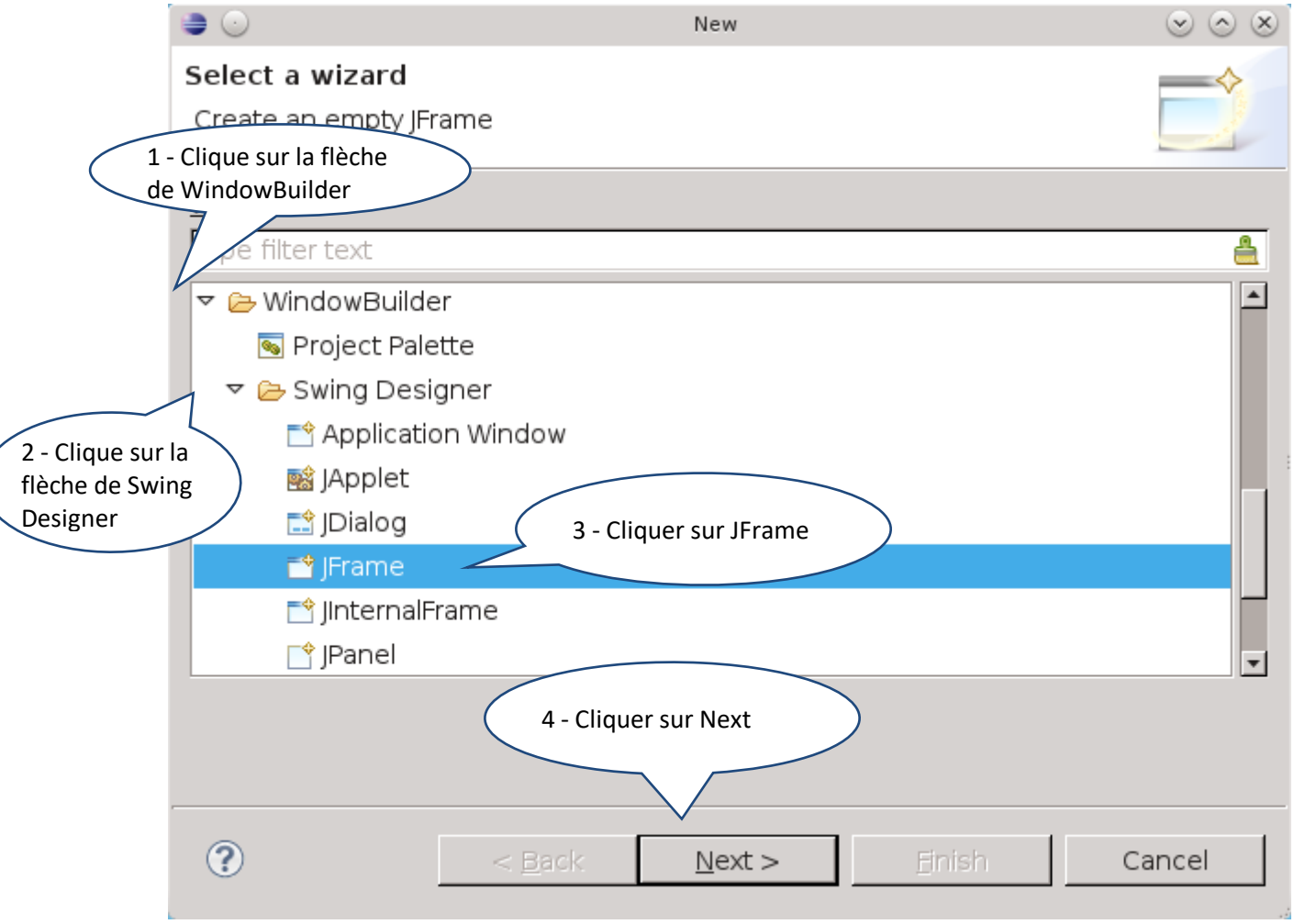

Install
Install Details
Review the items to be installed.
Name
Version
Id
Swing Designer
1.6.1.r43x201309100023
org
Swing Designer Documentation
1.6.1.r43x201309100012
org
WindowBuilder Core
1.6.1.r43x201309092314
org
WindowBuilder Core Documentation
1.6.1.r43x201309092334
org
WindowBuilder Core UI
1.6.1.r43x201309092325
org
WindowBuilder GroupLayout Support
1.6.1.r43x201309092345
org
WindowBuilder Java Core
1.6.1.r43x201309092318
org
Size: Unknown
Details
Confirmer les items en cliquant sur Next >
< Back
Next >
Finish
Cancel

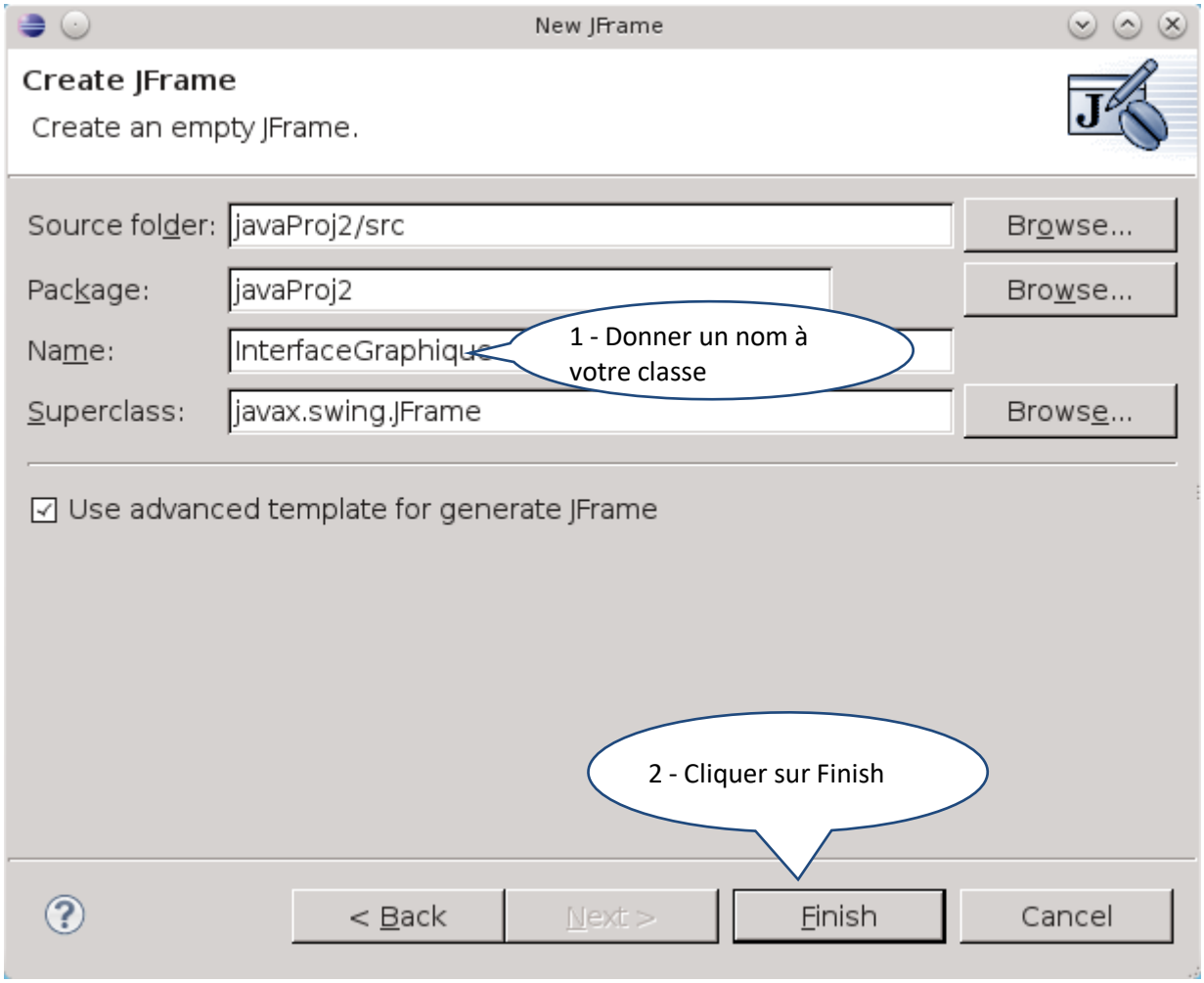

Install
Review Licenses
Licenses must be reviewed and accepted before the software can be installed.
Licenses:
License text:
Eclipse Foundation Software User Agreement
Eclipse Foundation Software User Agreer
February 1, 2011
Usage Of Content
THE ECLIPSE FOUNDATION MAKES AVAILA
SOFTWARE, DOCUMENTATION, INFORMAT
OR
OTHER MATERIALS FOR OPEN SOURCE PI
1 - Accepter les termes du contrat
I accept the terms of the license agre
I do not accept the terms of the licens
2 - Cliquer sur Finish
< Back
Next >
Finish
Cancel

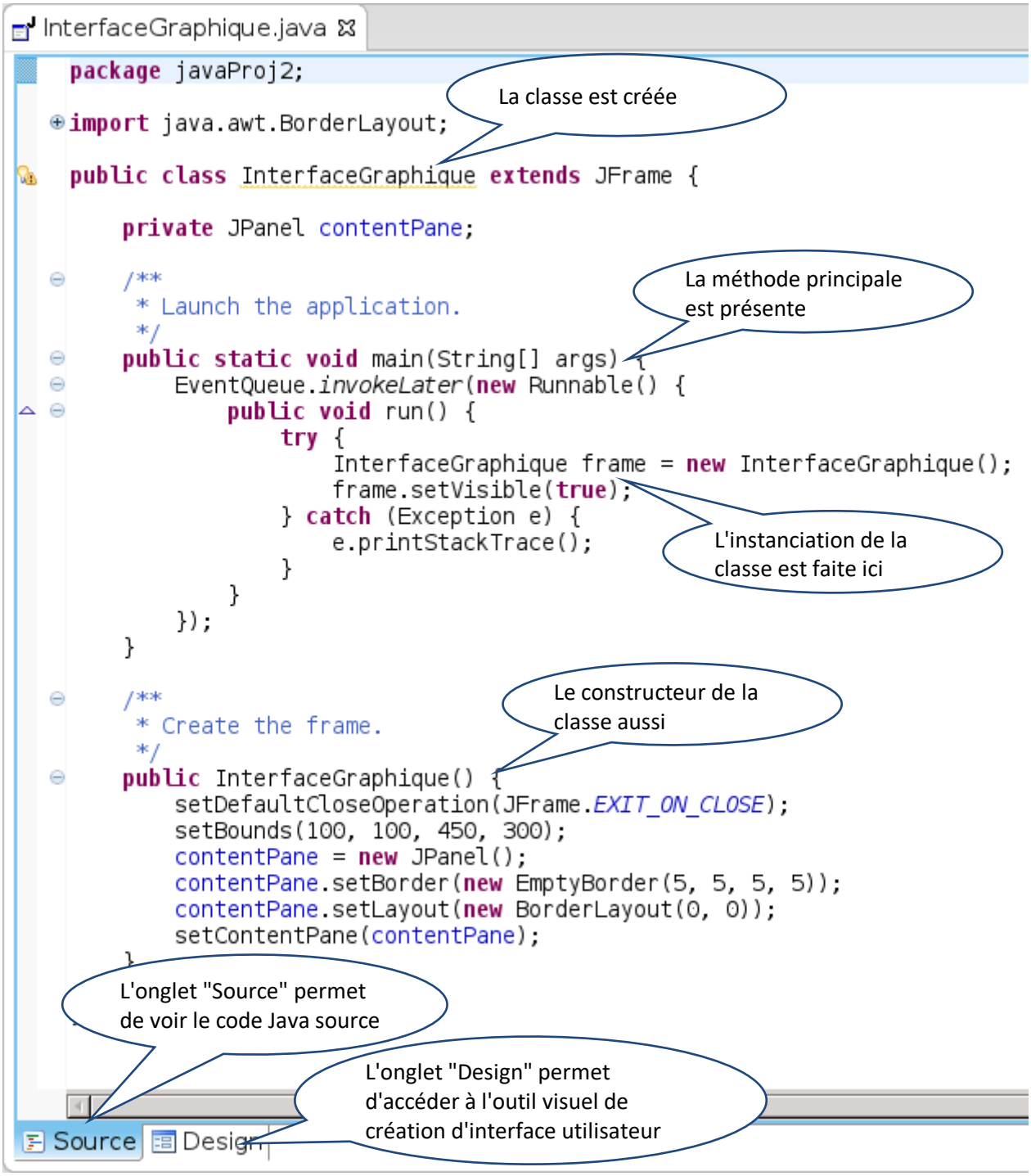

Installing Software
Installing Software
1 - Attendre la fin de l'installation
Always run in background
Run in Background
Cancel
Details >>


Software Updates
You will need to restart Eclipse for the changes to take effect. Would you like to restart now?
Redémarrer Eclipse pour finaliser l'installation des nouveaux logiciels
Yes
No

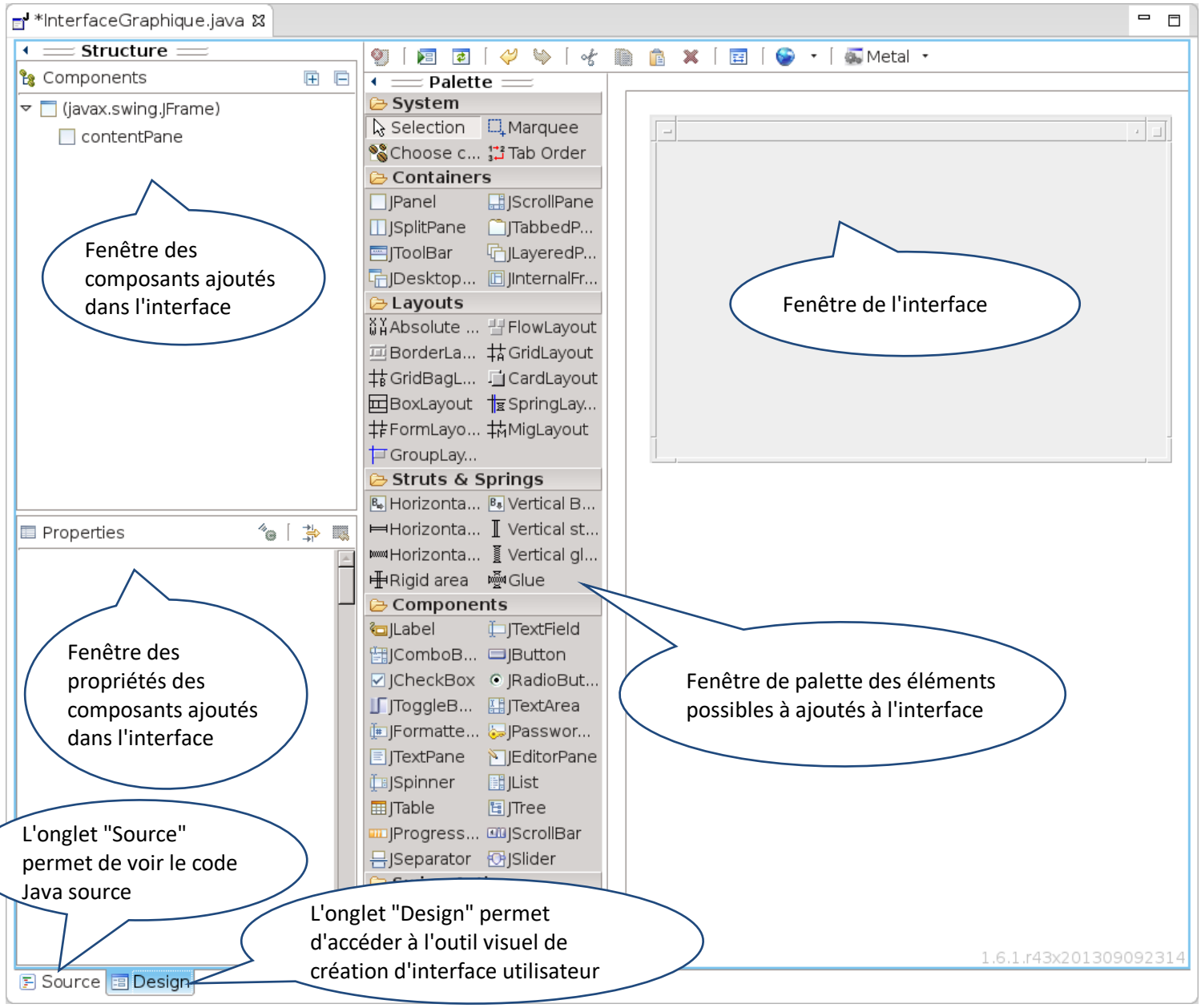

1 - Clique droit de la souris sur le projet
javaProj2
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Shift+Alt+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Shift+Ctrl+Alt+Down
Build Path
Source Shift+Alt+S
Refactor Shift+Alt+T
Import...
Export...
Build Project
2 - Choisir New
3 - Cliquer sur Other
Java Project
Project...
Package
Class
Interface
Enum
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Task
Example...
Other... Ctrl+N


New
Select a wizard
Create an empty JFrame
1 - Clique sur la flèche de WindowBuilder
2 - Clique sur la flèche de Swing Designer
WindowBuilder
Project Palette
Swing Designer
Application Window
JApplet
JDialog
JFrame
JInternalFrame
JPanel
3 - Cliquer sur JFrame
4 - Cliquer sur Next
< Back
Next >
Finish
Cancel


New JFrame
Create JFrame
Create an empty JFrame.
Source folder:
javaProj2/src
Browse...
Package:
javaProj2
Browse...
Name:
InterfaceGraphique
1 - Donner un nom à votre classe
Superclass:
javax.swing.JFrame
Browse...
Use advanced template for generate JFrame
2 - Cliquer sur Finish
< Back
Next >
Finish
Cancel

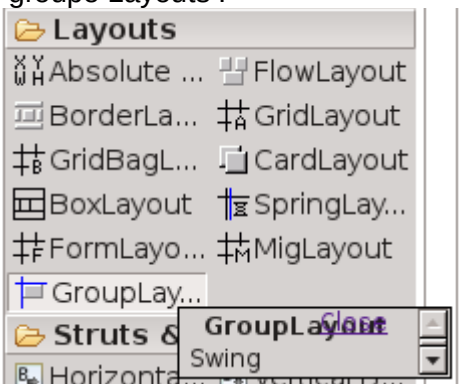



```
package javaProj2;

import java.awt.BorderLayout;

public class InterfaceGraphique extends JFrame {

    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    InterfaceGraphique frame = new InterfaceGraphique();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public InterfaceGraphique() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        contentPane.setLayout(new BorderLayout(0, 0));
        setContentPane(contentPane);
    }
```

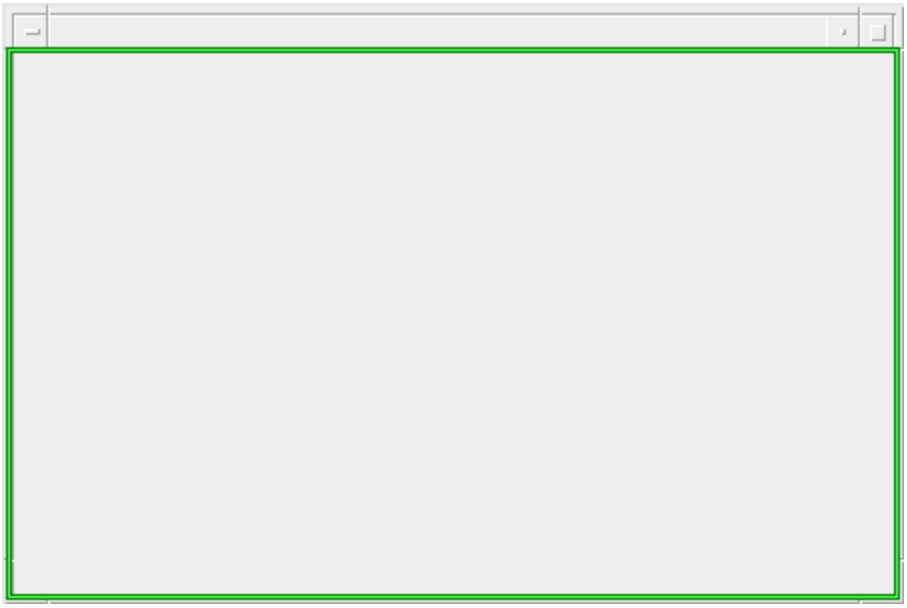


Nous allons ajouter un champ texte, une étiquette décrivant le champ texte ainsi qu'un bouton, comme dans l'interface précédente.

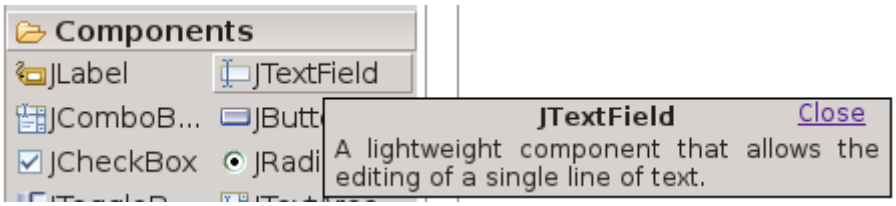
Cliquer sur GroupLayout dans le groupe Layouts :



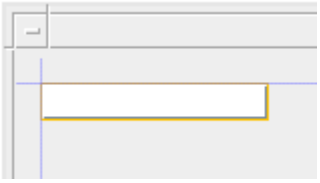
Puis **déplacer** la souris sur la fenêtre de l'interface, celle-ci devient verte comme ci-dessous, à coté du le curseur un + est ajouté, cliquer alors dans la fenêtre :



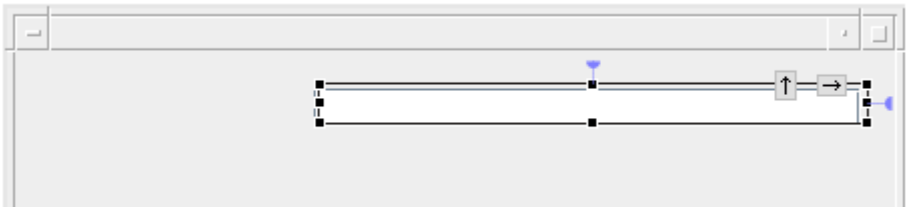
Cliquer sur JTextField dans le groupe Components, pour ajouter une zone de texte dans la fenêtre de l'interface :



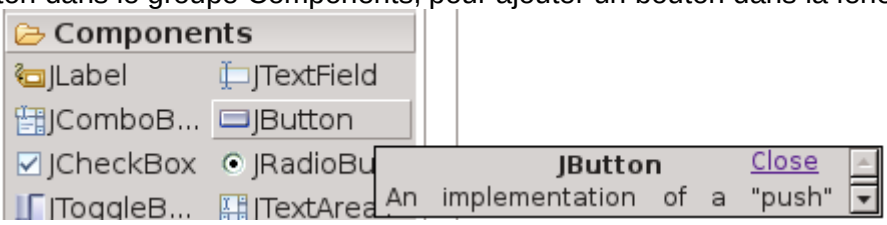
Puis **déplacer** la souris dans la fenêtre de l'interface pour ajouter la zone de texte :



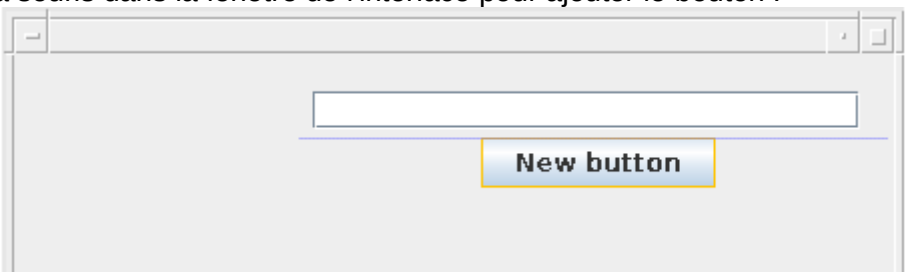
Modifier ensuite la zone de texte pour que la largeur soit comme ci-dessous :



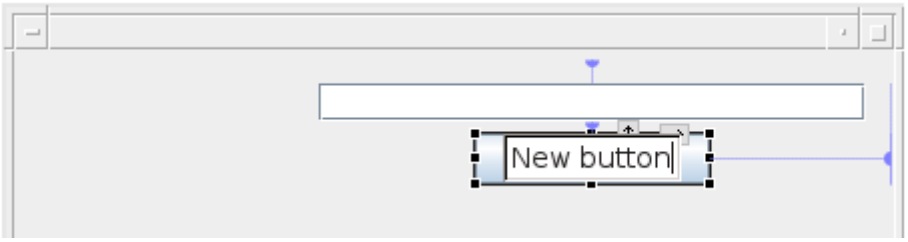
Cliquer sur JButton dans le groupe Components, pour ajouter un bouton dans la fenêtre de l'interface :

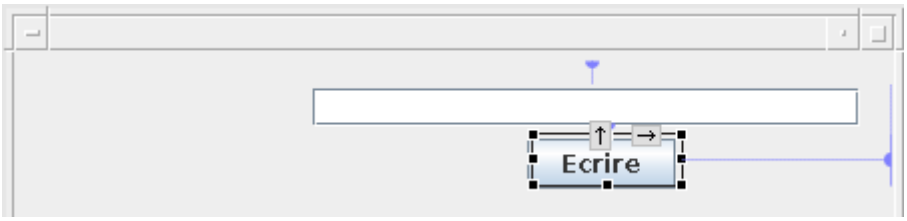


Puis **déplacer** la souris dans la fenêtre de l'interface pour ajouter le bouton :

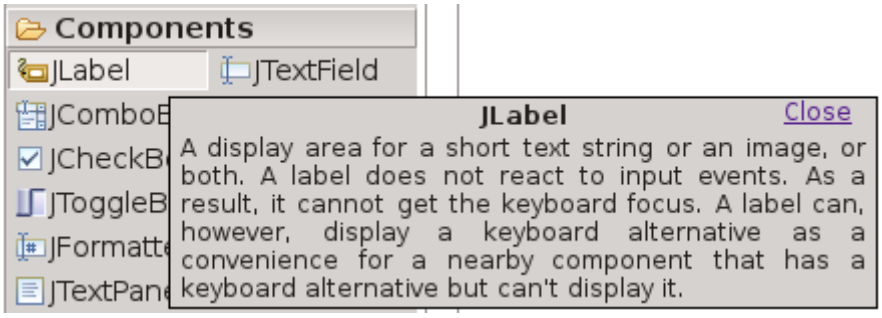


Ecrire une valeur dans le bouton : Ecrire

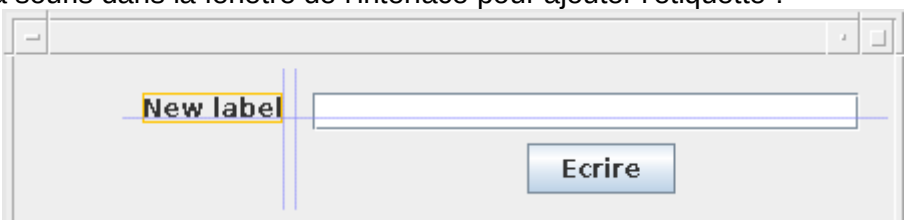




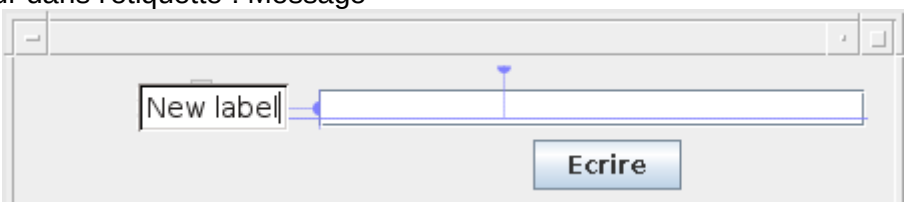
Cliquer sur JLabel dans le groupe Components, pour ajouter une étiquette près de la zone de texte dans la fenêtre de l'interface :

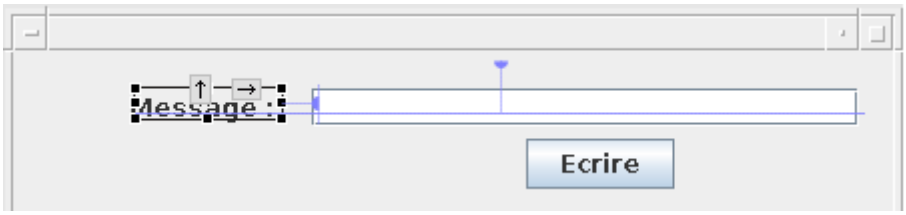


Puis **déplacer** la souris dans la fenêtre de l'interface pour ajouter l'étiquette :

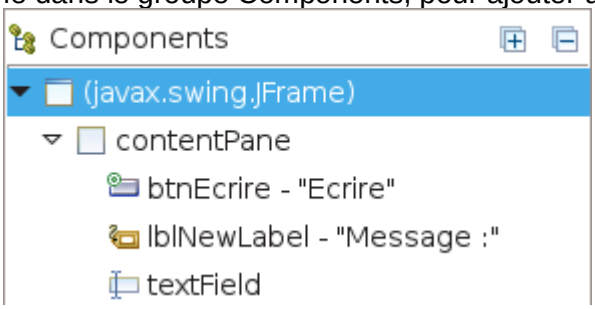


Ecrire une valeur dans l'étiquette : Message

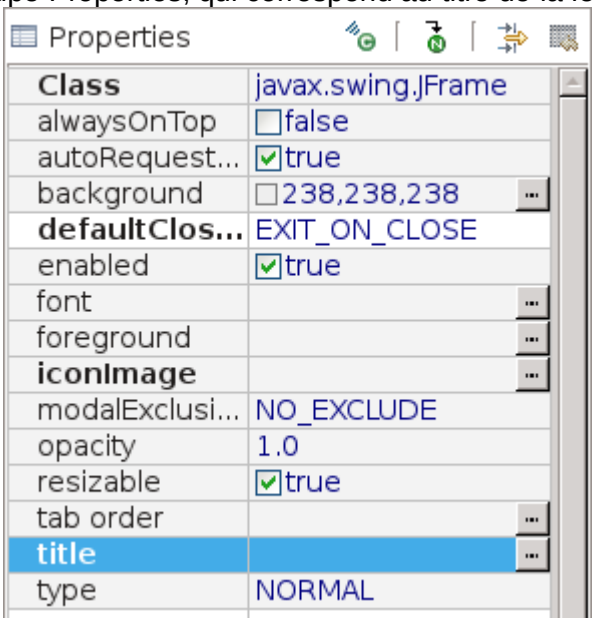




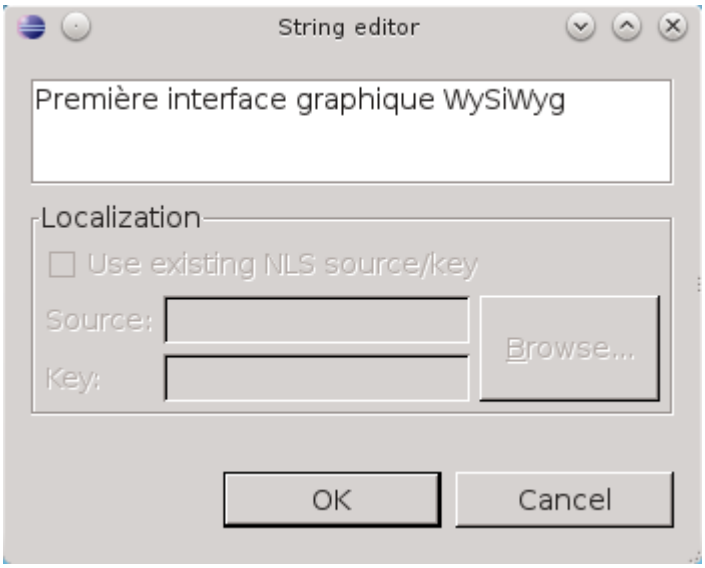
Cliquer sur javax.swing.frame dans le groupe Components, pour ajouter un titre à la fenêtre interface :



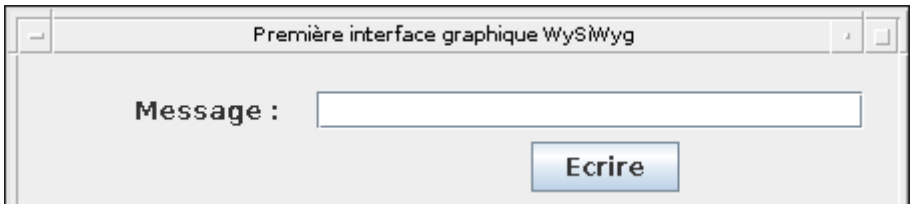
Cliquer sur title dans le groupe Properties, qui correspond au titre de la fenêtre interface :



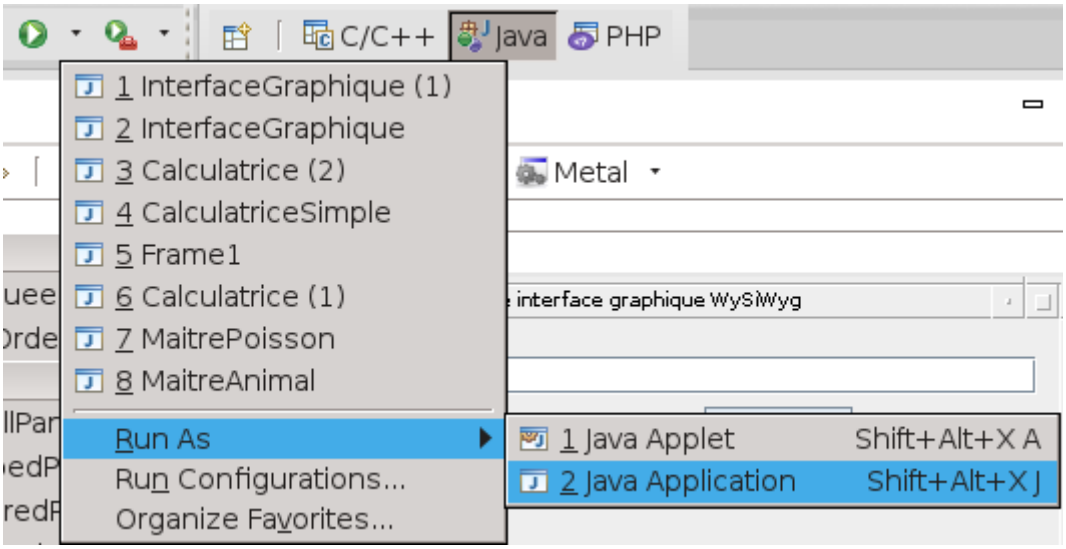
Ecrire le titre comme ci-dessous :



Il apparaît dans le haut de la fenêtre de l'interface :



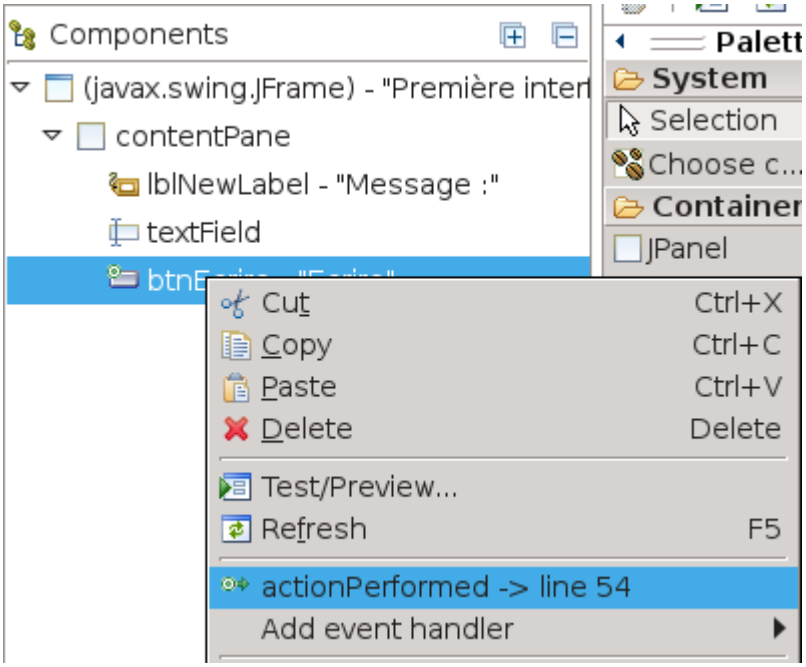
Cliquer sur la petite flêche noir pour faire apparaître le menu déroulant comme ci-dessous, pour exécuter l'application :



Ce qui doit donner :

Nous allons maintenant ajouter une interaction au bouton "Ecrire" pour écrire un message dans la zone de texte.

Avec le bouton droit de la souris, **cliquer** sur "btnEcrire", puis dans le menu déroulant qui apparaît, **cliquer** sur "actionPerformed":



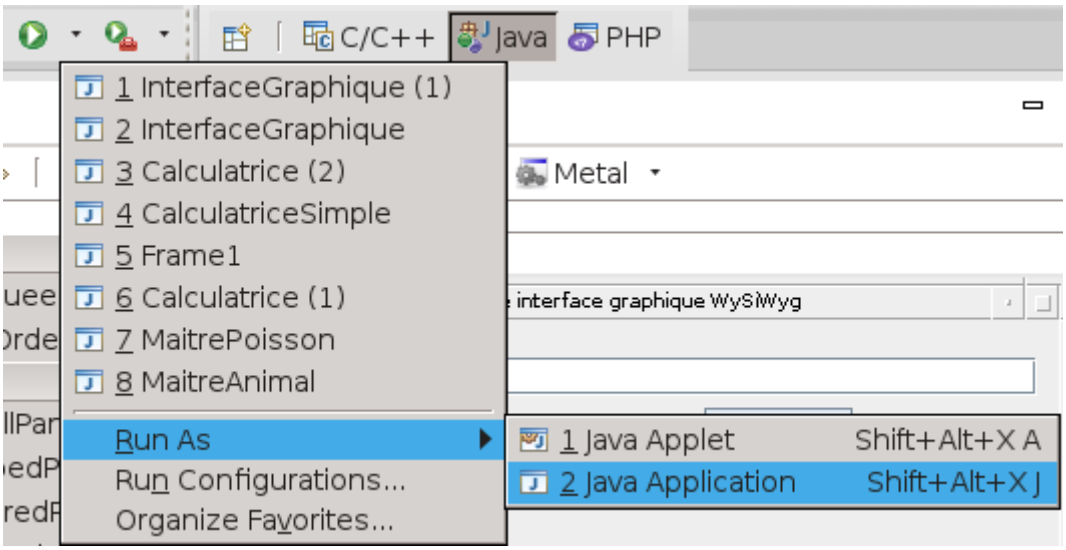
Vous êtes renvoyé à la fenêtre "Source" et sur le code correspondant à l'action du bouton :

```
JButton btnEcrire = new JButton("Ecrire");
btnEcrire.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
    }
});
```

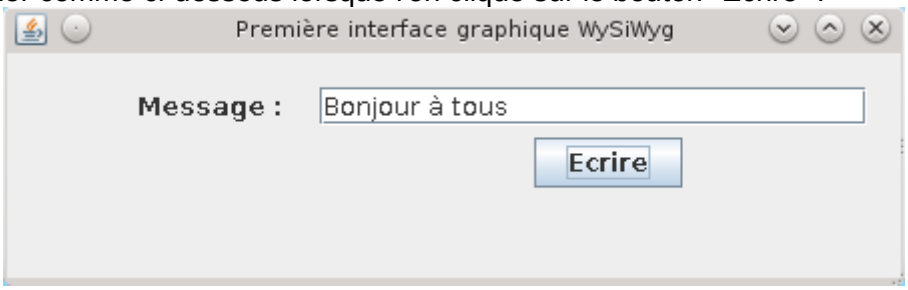
Ajouter la ligne de code encadrée ci-dessous :

```
JButton btnEcrire = new JButton("Ecrire");
btnEcrire.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        textField.setText("Bonjour à tous");
    }
});
```

Cliquer sur la petite flèche noir pour faire apparaître le menu déroulant comme ci-dessous, pour exécuter l'application :



Ce qui doit donner comme ci-dessous lorsque l'on clique sur le bouton "Ecrire" :



VI. Conclusion

Au travers des activités, vous avez appris comment écrire des classes en Java et comment celles-ci peuvent interagir en elles.
Vous avez utilisé des API avec Java pour créer des fenêtres graphiques et ajouter des interactions avec le code
Enfin vous avez utilisé un environnement de développement graphique WySiWyg (What you see is what you get) pour développer plus rapidement une interface graphique ainsi que mettre en œuvre des évènements liés à l'appui d'un bouton sur cette interface graphique.