Exercice 02

Création d'une première IHM

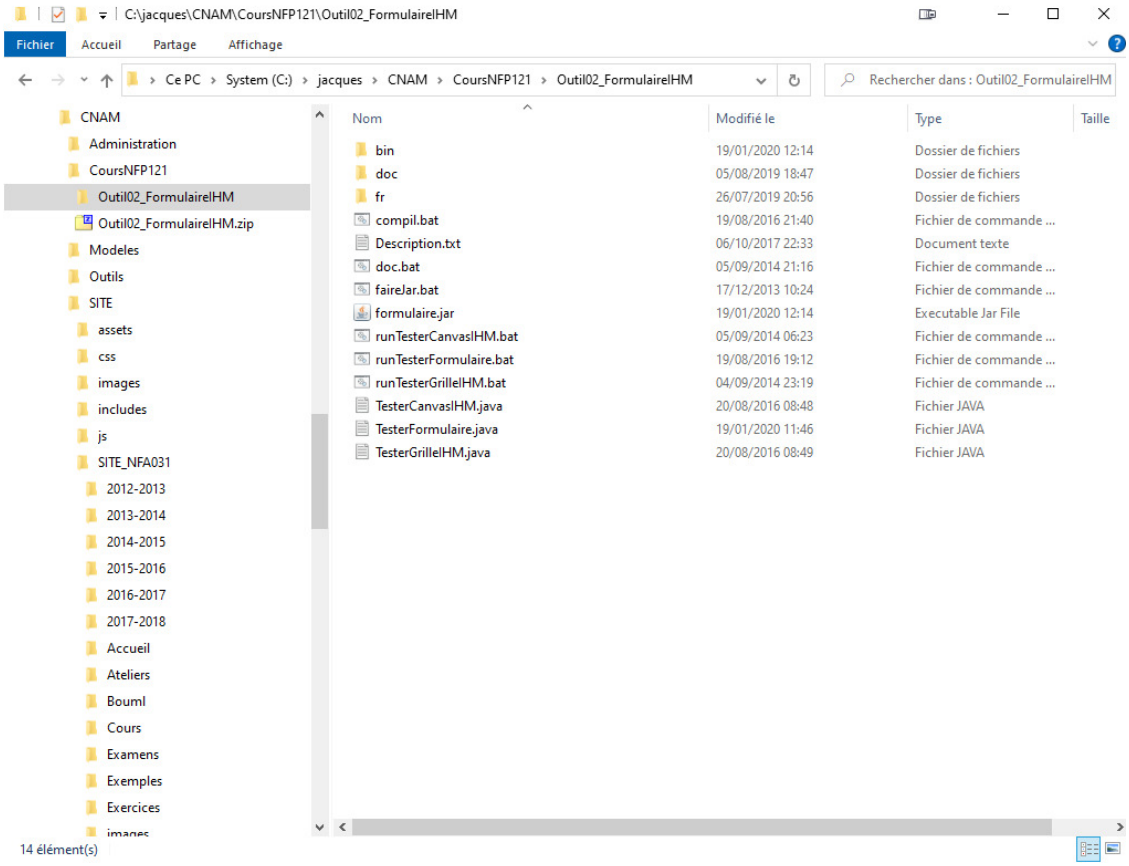
Apprentissage de l'utilisation de l'outil de création d'IHM : la classe Formulaire.
Apprentissage d'Eclipse.

1. Créer un programme d'IHM

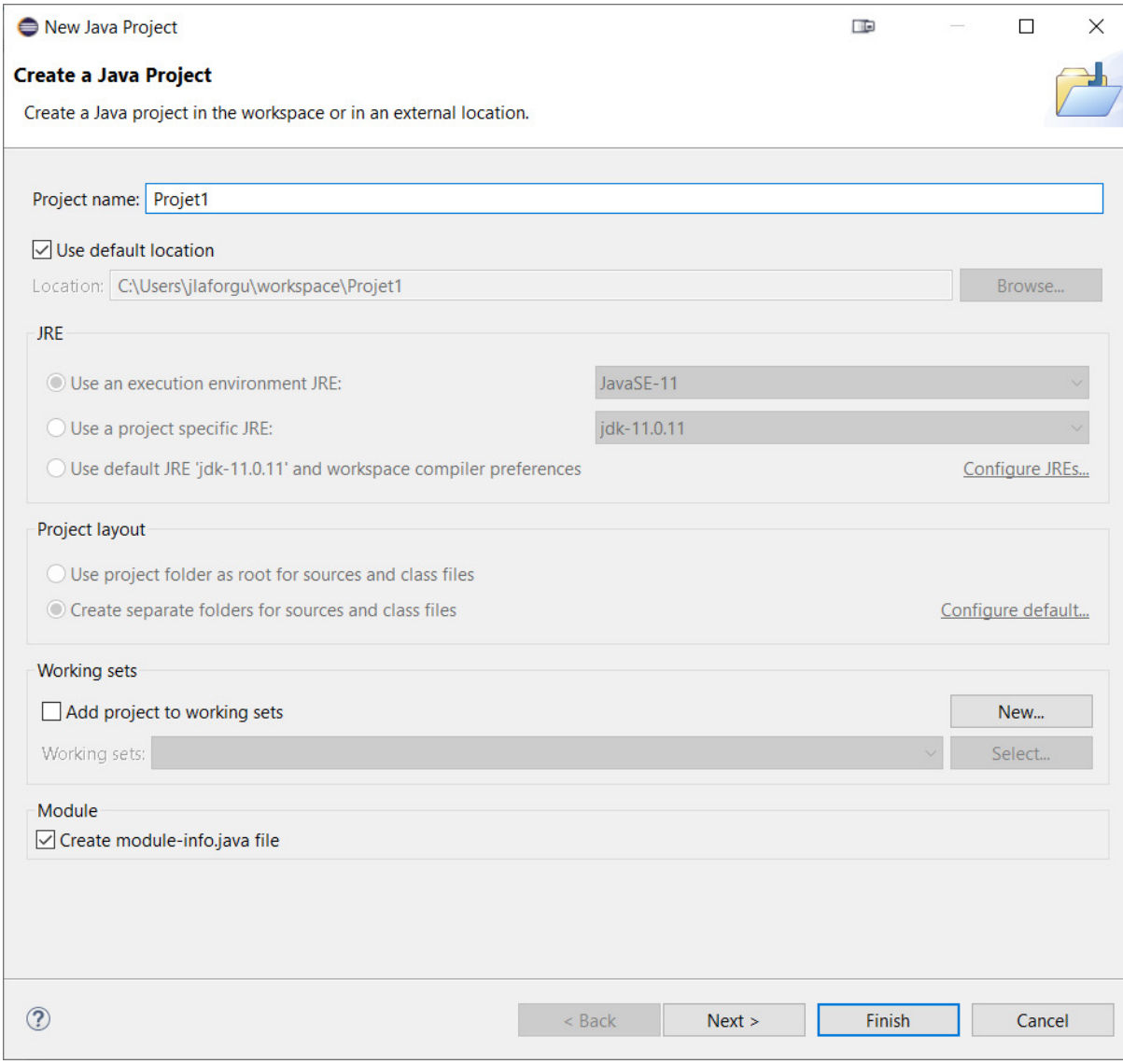
1.1. Initialiser son projet avec le package d'outil d'IHM

Télécharger dans un répertoire quelconque, par exemple CNAM/CoursNFP121, l'outil permettant de créer ses IHM :
http://coursjava.fr/NFA031_Outils.php?repertoire=Outil02_FormulaireIHM

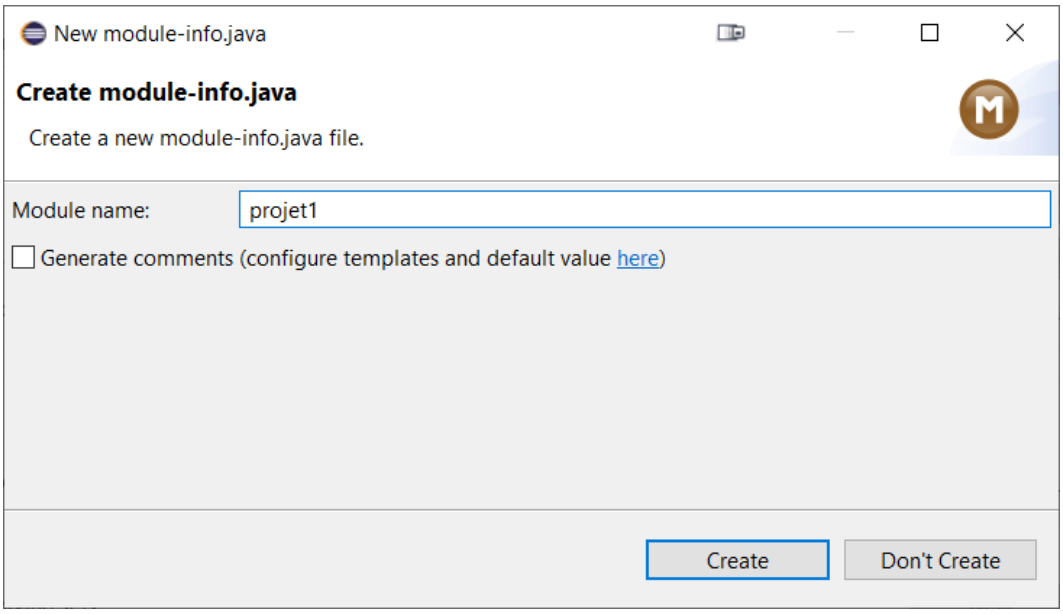
Extraire le zip. On obtient ceci :



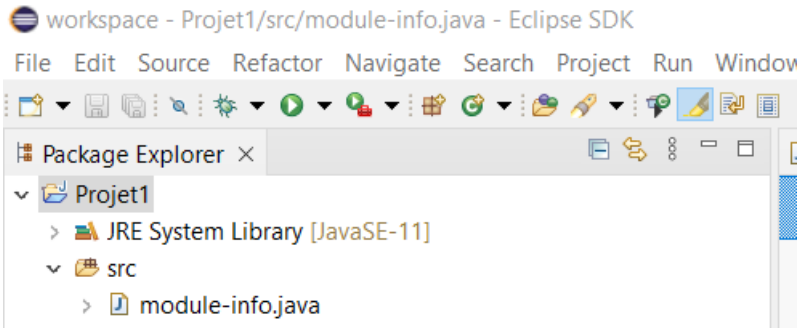
Dans Eclipse, créer un nouveau projet Java : >File>New>Java Project



Saisir le nom de votre projet : Projet1
>Finish

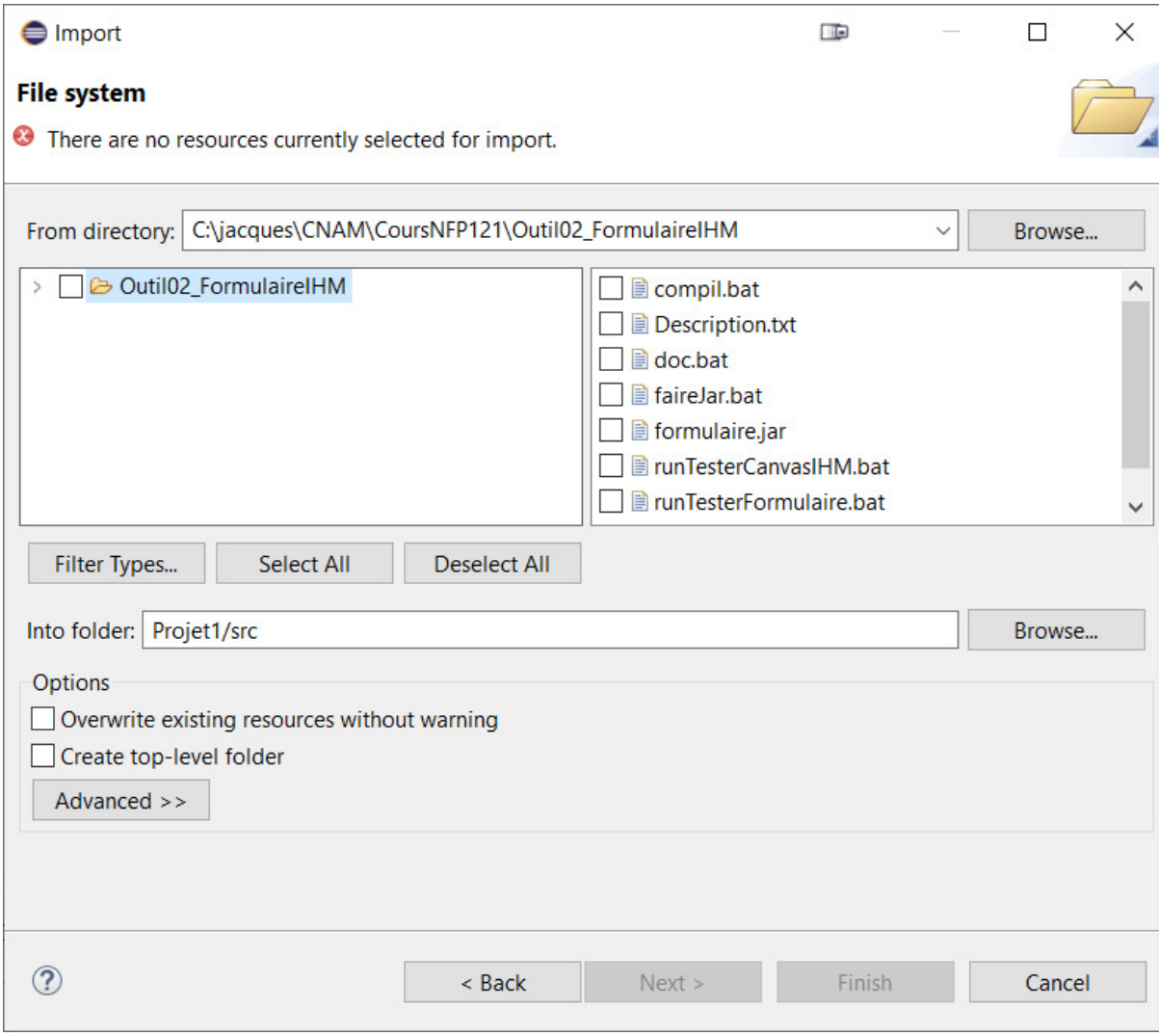


Saisir : projet1 (pas en majuscule)
>Create

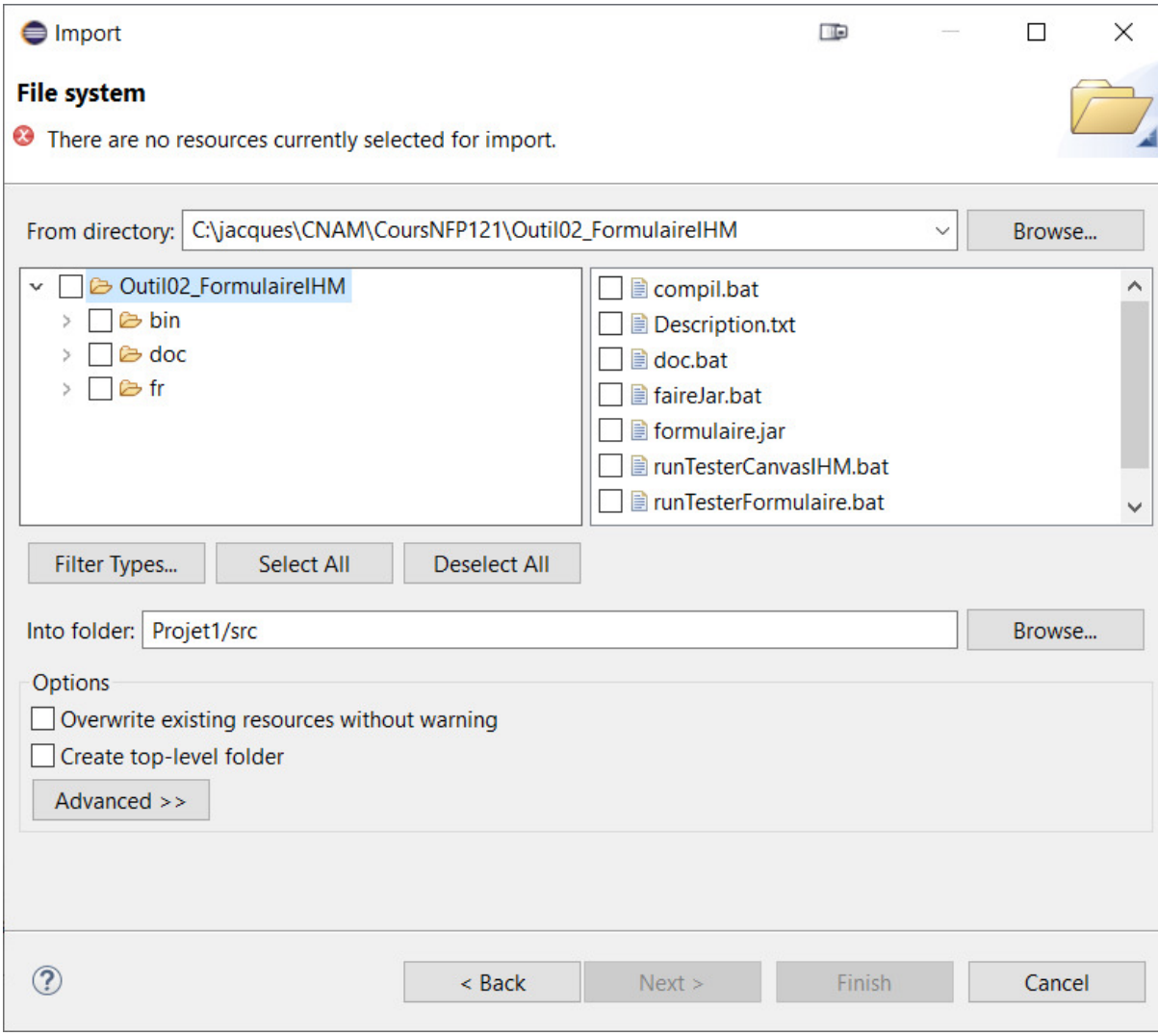


Sélectionner **src**. Clic-droit > Import > File System> Next > Browse > CNAM\CoursNFP121
Dossier : Outil02_FormulaireIHM
>Sélectionner un dossier

On obtient la fenêtre suivante :



Ouvrir l'arborescence de la racine :



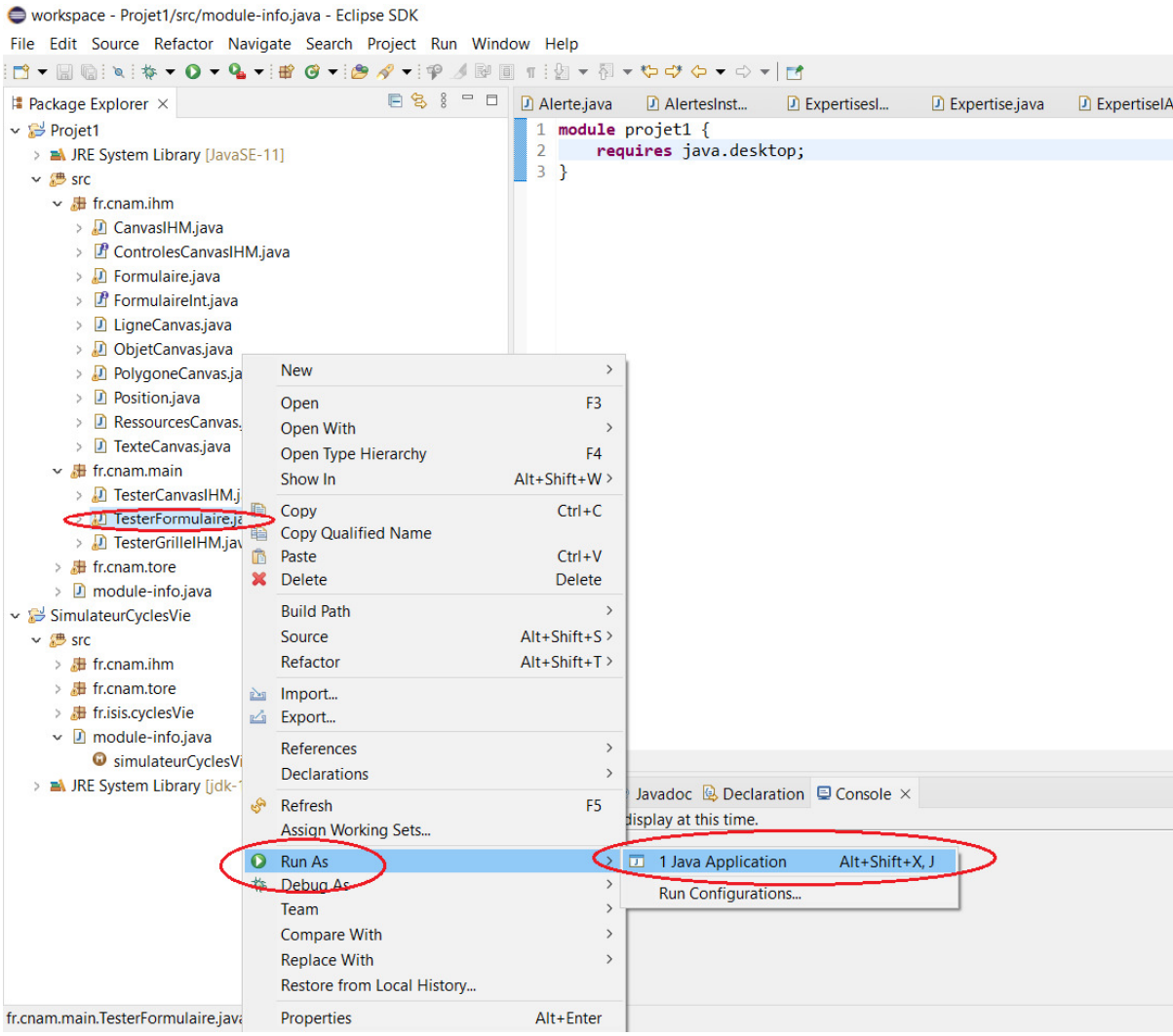
Cette fenêtre permet de choisir ce que l'on va importer (on ne veut pas importer tout l'outil qui contient la doc, les scripts windows). On ne veut importer que le package qui va nous permettre de fabriquer nos ihm.

Cocher le package **fr** > Finish

Modifier le fichier **module-info.java** :

```
module projet1 {
	requires java.desktop;
}
```

On va lancer le programme main de test de la création de Formulaire :

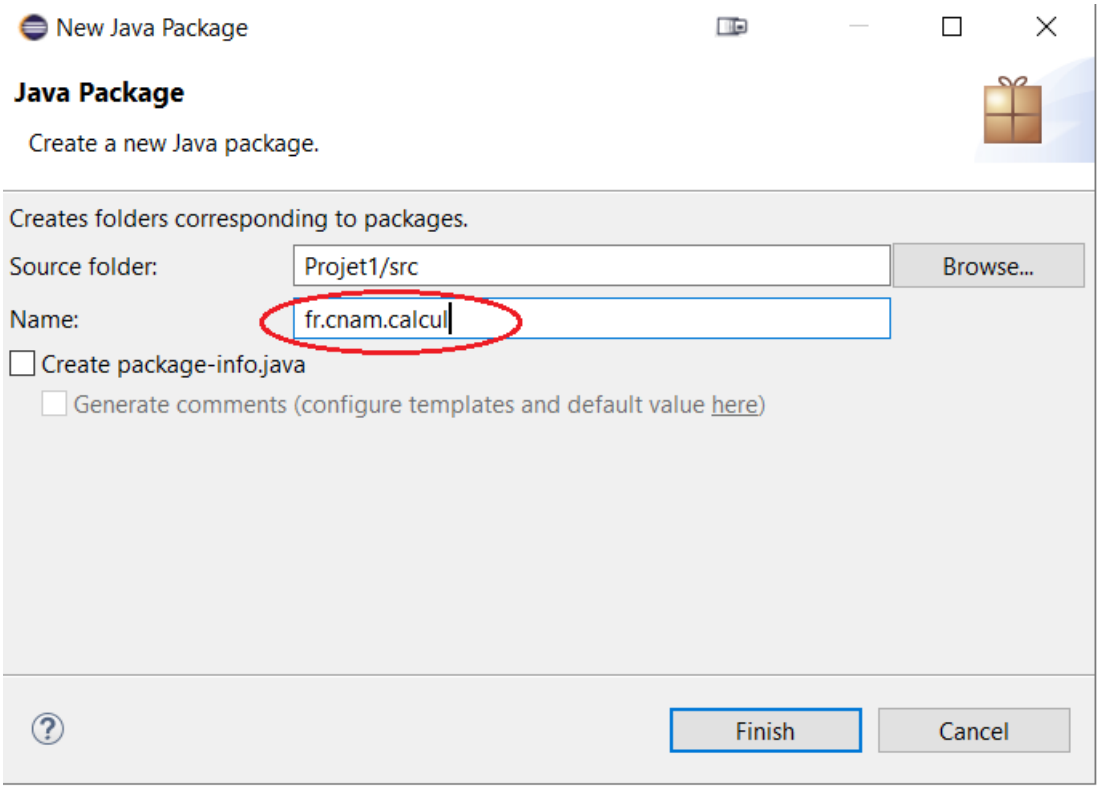


1.2. Créer un programme d'IHM

On se propose de créer une classe qui réalise un calcul quelconque **Calculer** et une classe d'IHM **CalculerIHM** qui va permettre de saisir les paramètres de calcul, lancer le traitement de calcul, récupérer le résultat calculé, et afficher ce résultat.

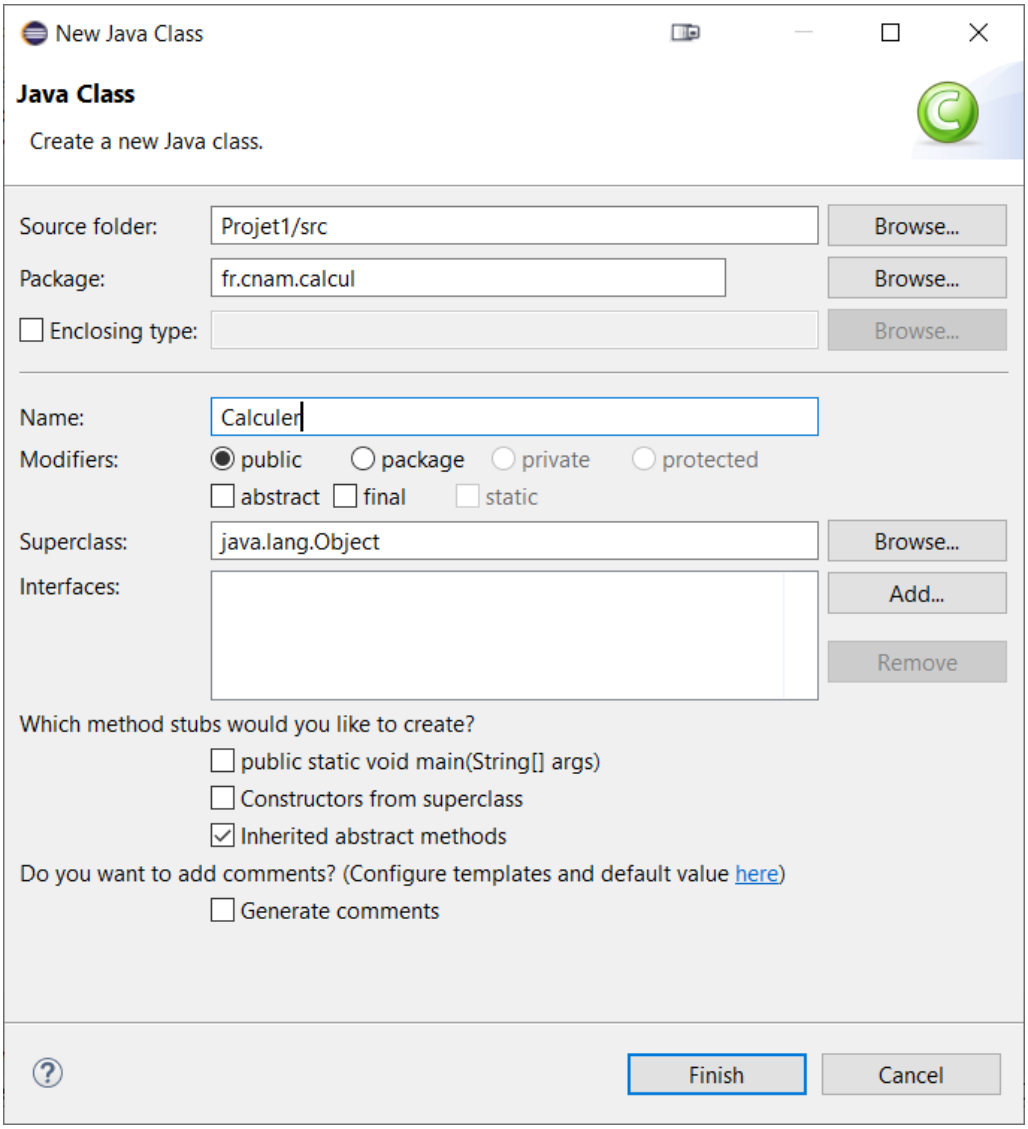
Créer un nouveau package : **fr.cnam.calcul** comme ceci :

Clic droit sur **src** > New > package



> Finish

Créer la classe **Calculer** :
Click droit sur fr.cnam.calcul > New > Class


New Java Class
Java Class
Create a new Java class.
Source folder:
Projet1/src
Browse...
Package:
fr.cnam.calcul
Browse...
Enclosing type:
Browse...
Name:
Calculer
Modifiers:
public
package
private
protected
abstract
final
static
Superclass:
java.lang.Object
Browse...
Interfaces:
Add...
Remove
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments
Finish
Cancel

> Finish

Mettre le code suivant dans la classe **Calculer** :

```
package fr.cnam.calcul;

import fr.cnam.ihm.*;

public class Calculer implements FormulaireInt {

private int v1;
private int v2;
private int resultat;

public void additionner()
{
   resultat = v1 + v2;
}

public int getV1() {return v1;}
public void setV1(int v1) {this.v1 = v1;}
public int getV2() {return v2;}
public void setV2(int v2) {this.v2 = v2;}
public int getResultat() {return resultat;}

// Méthode appelée par le Formulaire d'IHM
public void submit(Formulaire form,String action) {

   if (action.equals("ADDITIONNER")) {
      String s1=form.getValeurChamp("VALEUR1");
      String s2=form.getValeurChamp("VALEUR2");

      setV1(Integer.parseInt(s1));
      setV2(Integer.parseInt(s2));

      additionner();

      form.setValeurChamp("RESULTAT", ""+getResultat());
   }
}
}
```

De même créer, dans le packqge fr.cnam.calcul, la classe **CalculerIHM**.
Y coller le code suivant :

```
package fr.cnam.calcul;

import fr.cnam.ihm.Formulaire;

public class CalculerIHM {

private Formulaire form;

public CalculerIHM(Calculer calculer) {

   form= new Formulaire("CALCULER",calculer,300,300,true);

   form.setPosition(10,10);
   form.addText("VALEUR1", "Valeur 1", true, "");
   form.addText("VALEUR2", "Valeur 2", true, "");
   form.addButton("ADDITIONNER","Additionner");
   form.addText("RESULTAT", "Résultat : ", false, "");

   form.afficher();
}
}
```

De même créer, dans le packqge fr.cnam.calcul, la classe **Programme**.
Y coller le code suivant :

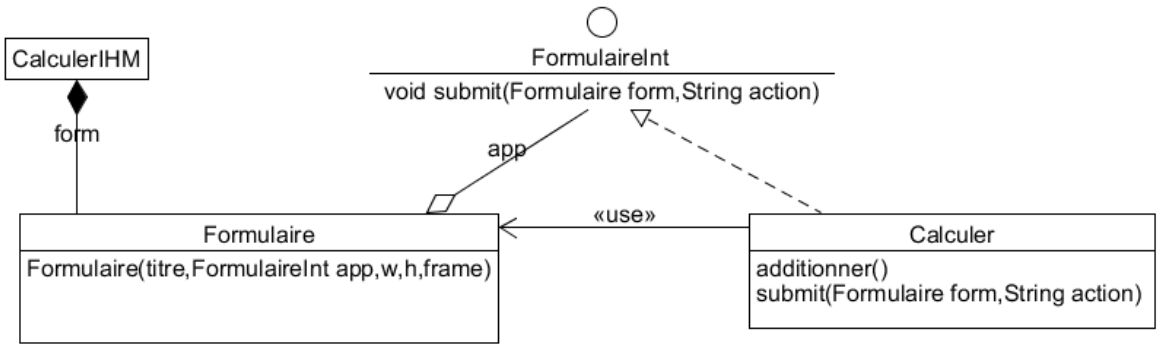
```
package fr.cnam.calcul;

public class Programme {

public static void main(String[] args) {

   Calculer calculer = new Calculer();
   CalculerIHM ihm = new CalculerIHM(calculer);
}
}
```

Vérifier qu'il n'y a pas d'erreur de compilation.

Exécuter le programme : click-droit sur la classe Programme. Puis : Run AS > Java Application

1.3. Principe de fonctionnement

L'objectif est de séparer les classes applicatives (on parle de « MODELE ») et les classes de l'interface graphique (on parle de « VUE ») qui permet de modifier ce modèle et d'exécuter les traitements de ce modèle.



Le code du programme principal :

```
Calculer calculer = new Calculer();
CalculerIHM ihm = new CalculerIHM(calculer);

ihm.getForm().afficher();
```

La classe **Calculer** est créée. Elle implémente la méthode **submit** de l'interface **FormulaireInt**.

La classe **CalculerIHM** est créée. Cette classe crée un objet **Formulaire**.

```
form= new Formulaire("CALCULER",calculer,300,300,true);
   form.setPosition(10,10);
   form.addText("VALEUR1", "Valeur 1", true, "");
   form.addText("VALEUR2", "Valeur 2", true, "");
   form.addButton("CALCULER","Lancer le calcul");
   form.addText("RESULTAT", "Résultat : ", false, "");

   form.afficher();
```

qui prend en entrée de son constructeur tout objet qui implémente l'interface **FormulalireInt**. Ce qui est le cas de la classe **Calculer**.

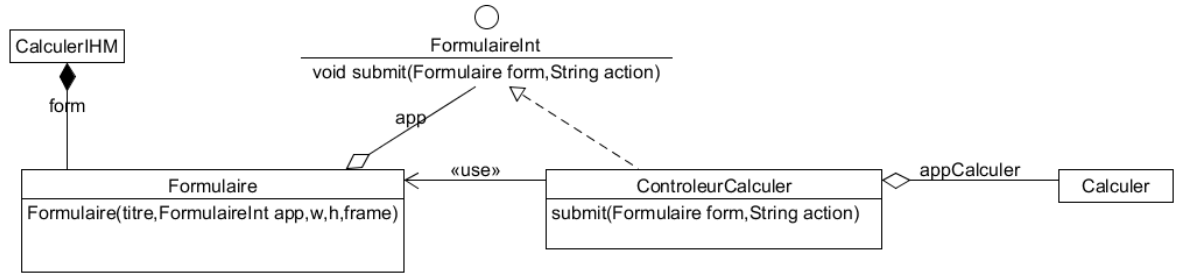
Quand on fait une action dans un des éléments graphiques du formulaire, la méthode **submit** est appelée.
Cette méthode qui est implémentée dans la classe de calcul peut donc faire un traitement.
Et comme en entrée de cette méthode se trouve le formulaire et l'action réalisée (le nom de l'élément graphique qui a déclenché une action (un bouton par exemple)) alors la classe de calcul :

- peut déterminer quelle action a été réalisée
- utiliser des méthodes du formulaire lui permettant de :
  - récupérer des valeurs saisies dans les éléments graphiques
  - modifier le contenu de n'importe quel élément graphique.

```
if (action.equals("CALCULER")) {
     String s1=form.getValeurChamp("VALEUR1");
     String s2=form.getValeurChamp("VALEUR2");
     int v1=Integer.parseInt(s1);
```

```
        int v2=Integer.parseInt(s2);

        int resultat = v1 + v2;

        form.setValeurChamp("RESULTAT", ""+resultat);
    }
```

Remarque : On s'aperçoit que la classe applicative **Calculer** est « polluée » par la méthode submit qui fait appel à des éléments de l'interface graphique. Il est possible de créer une classe intermédiaire qui contiendrait ce code. Cette classe est appelée un contrôleur. On n'obtient alors un modèle que l'on appelle MVC (Modèle-Vue-Controleur).



Mais cela est une autre histoire 😊.
NB : Tel décrit ce modèle MVC est incomplet. Mais nous verrons cela plus tard.

2. Gérer les erreurs d'exécution

Présentation en séance de TP

3. Utiliser le debugger

Exécuter le programme : click-droit sur la classe Programme. Puis : Debug AS > Java Application

Présentation en séance de TP

4. La notion de workspace

Présentation en séance de TP

5. Exercices

5.1. Exercice 1

Faites évoluer le programme pour réaliser le traitement de la multiplication

5.2. Exercice 2

Faites évoluer le programme pour gérer correctement le cas d'erreur quand on saisie une valeur qui n'est pas un entier. Un texte d'erreur est affiché dans l'IHM.

5.3. Exercice 3

Modifier le programme qui utilise un contrôleur (modèle MVC).