

Polytech Lille IMA2A4
Conception Modélisation Objets (CMO)
TP5

1 Packages et encapsulation

1. Ranger toutes les classes (sauf la classe d'application `TestCircuits`) dans un package `circuits` :
 - créer un répertoire `circuits` et y ranger les fichiers `.java` de ces classes
 - placer en tête (première ligne, avant les `import`) de ces fichiers la déclaration :

```
package circuits;
```
 - les chemins d'accès aux packages doivent être listés dans la variable d'environnement `CLASSPATH` ou dans le paramètre `-cp` des commandes `javac` et `java`. Le répertoire courant (`.`) y est par défaut, il vous suffit donc de compiler en vous situant "au dessus" du répertoire `circuits`, soit :

```
> javac ./circuits/*.java
```
2. Ranger la classe `TestCircuits` dans un package séparé `test` :
 - créer un répertoire `test` au même niveau que `circuits` et y ranger le fichier `TestCircuits.java` avec son entête :

```
package test;
```
 - importer les classes du package `circuits` dans `TestCircuits.java` :

```
import circuits.*;
```
 - compiler :

```
> javac ./test/TestCircuits.java
```
3. Il se peut que la compilation ne passe plus à cause des modalités de visibilité.
 - Exemples d'erreurs possibles :
 - (a) si n'avez pas déclaré `public` la classe `circuits.Circuit` :

```
test/TestCircuits.java:23: circuits.Circuit is not public in circuits;
cannot be accessed from outside package
    static void test(Circuit circ) {
                ^
```

- (b) si vous n'avez pas déclaré `public` la méthode `getIns()` de `circuits.Circuit` :

```
test/TestCircuits.java:28: getIns() is not public in circuits.Circuit;
cannot be accessed from outside package
    for (Interrupteur in : circ.getIns())
        ^
```

- Ajuster ces modalités, notamment pour rendre `public` ce qui est exporté (classes et méthodes) par le package `circuits` et donc utilisable par `test.TestCircuits`, et cacher le reste. Généralement :

- déclarer les classes `public` (ou ne rien mettre si utilitaire interne = “package limited”)
 - déclarer les méthodes `public`, sinon ne rien mettre (“package limited”) ou `protected` (“subclass limited”).
 - encapsuler les variables d’instances : ne rien mettre (“package limited”) ou `protected`, ou encore `private` mais attention elles ne seront plus accessibles dans les sous-classes (même si elles sont dans le même package).
4. Exécution :
- ```
> java ./test/TestCircuits
```
5. “Jouer à cache-cache” en essayant différents niveaux de visibilité, par exemple :
- Si vous n’avez pas vécu les erreurs précédentes :
    - enlever la modalité `public` sur la classe `circuits.Circuit`. Recompiler le package `circuits`. Puis en recompilant `TestCircuits`, vous devez tomber sur l’erreur 3a
    - enlever la modalité `public` sur la méthode `getIns()` de `circuits.Circuit`. Faites de même, vous devez tomber sur l’erreur 3b
  - Classes `public` ou non :
    - La classe `circuits.Porte2Entrees` est un intermédiaire de factorisation de la hiérarchie des classes de composants interne au package `circuits` (non utilisée dans `test`). La cacher (non `public`), recompiler les 2 packages. Que se passe-t-il ?
    - Idem pour la classe `circuits.SondesTable`, utilitaire interne pour faciliter l’implantation des fonctionnalités `probe()/unProbe()` sur les circuits. Vérifier de même qu’elle peut rester locale.
    - Encapsuler fortement (`private`) les champs `in1` et `in2` de `Porte2Entrees`. Compiler le package `circuits`. Que se passe-t-il pour ses sous-classes ?
    - La liste `composants` d’un circuit doit être encapsulée (tout sauf `public`, au moins `protected`) car gérée par celui-ci :
      - Essayer dans `TestCircuits` de corrompre cette liste sans passer par le circuit, par exemple :
 

```
circ.composants.add(new And());
```

 que se passe-t-il ?
      - Rendre provisoirement cette liste `public` pour la corrompre comme précédemment et ré-essayer. Noter que la nomenclature n’est plus triée... (ni `description()`, ni `traceEtats()`).

## 2 Sauvegarde par sérialisation

La sérialisation va permettre de sauvegarder les circuits et de les recharger. Dans la classe `Circuit`, écrire un couple de méthodes :

- `public void save(String fileName)` qui sauvegarde dans le fichier de nom `fileName` le nom du circuit (`String` donc sérialisable) et sa liste `composants`. Pour cela il est nécessaire de rendre `Serializable` les classes de `Composant`.
- `public void load(String fileName)` qui fait l’inverse : charge le nom du circuit et sa liste `composants`.

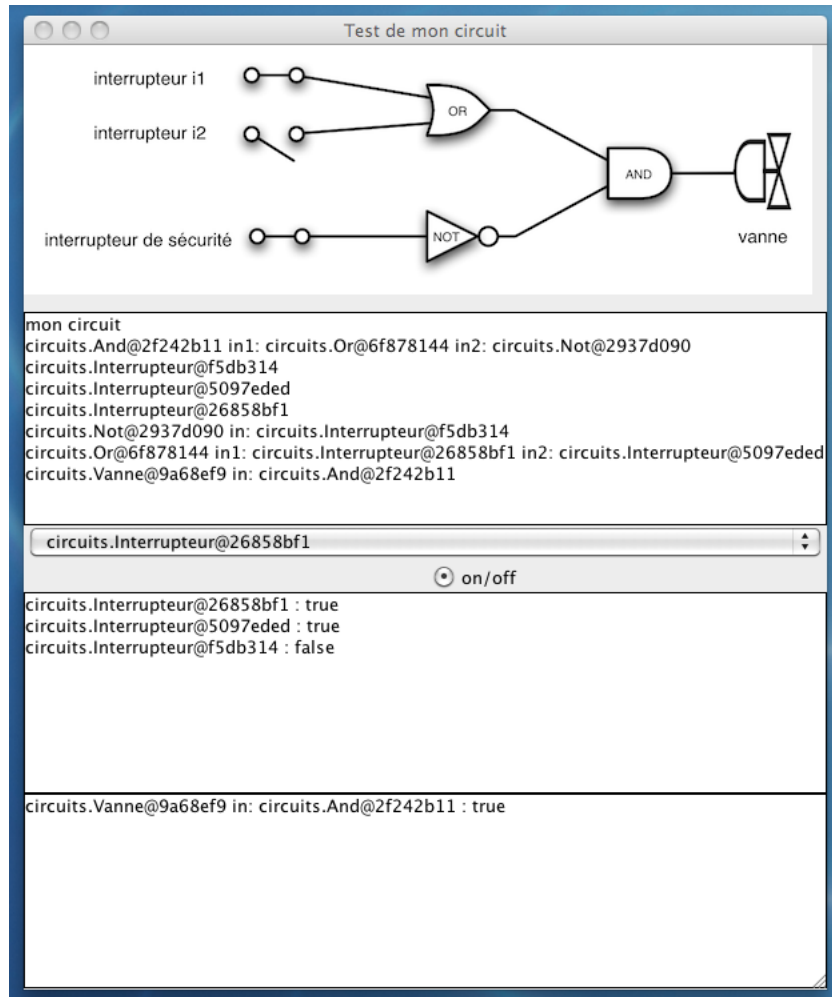
### Test

- Dans la classe `TestCircuits` sauvegarder l’état du circuit exemple en demandant à l’utilisateur un nom de fichier (soit `fileName` du genre “`circuit.bin`”).
- Programmer une autre classe de test `test.TestSerial` :
  - dont le `main` est paramétré par un fichier de sauvegarde créé comme précédemment (`fileName`)
  - créer un objet circuit à partir de ce fichier de la façon suivante :
 

```
Circuit circ = new Circuit();
circ.load(fileName);
```
  - lui applique `traceEtats()` pour vérifier le rechargement de la configuration.

### 3 Interface graphique

Nous allons concevoir une petite interface graphique **Testeur** qui permet de tester le circuit donné en exemple sauvegardé comme précédemment. L'interface a l'allure suivante :



De haut en bas :

- image du circuit (**Canvas** d'affichage d'un fichier `.png`)
- **JTextArea** (non éditable) contenant sa description
- **JComboBox** (liste de sélection) qui permet de sélectionner un interrupteur d'entrée
- bouton **on/off** (**JRadioButton**) qui commute l'interrupteur sélectionné
- **JTextArea** contenant l'état des interrupteurs (obtenu par leur `traceEtat()`)
- **JTextArea** contenant l'état des sorties (la vanne en l'occurrence).

L'ensemble forme une **JFrame** (qui porte dans sa bannière le nom du circuit chargé) et les composants d'interface sont positionnés selon une stratégie verticale (**BoxLayout** en **Y**).

L'application est paramétrée par un nom de fichier (soit `fileName`, sans extension, par exemple "circuit") auquel doit correspondre :

- une extension `.bin` contenant le circuit sérialisé à tester
- une extension `.png` contenant son image à afficher.

#### A faire :

- récupérer l'archive java (`.jar`) :  
~barre/public\_html/ima4-FSC/circuits/ihm.jar

- Désarchiver par :
  - > `jar xvf ihm.jar`
- Vous devez obtenir :
  - `circuit.png` : l'image du circuit exemple
  - `ihm/Testeur.java` le code (incomplet) de la classe `ihm.Testeur`
- Examiner le code de cette classe qui manipule les classes du package `circuits`
- Remarquer notamment que pour afficher la description du circuit dans la zone correspondante, l'interface nécessite les méthodes suivantes à programmer dans `circuits.Circuit` :
  - `String getName()` qui renvoie le nom du circuit
  - `String description()` qui renvoie sa description sous la forme d'une `String`.
- Compiler ensuite la classe `ihm.Testeur` et l'essayer sur le circuit exemple sérialisé précédemment :
  - > `java ihm.Testeur circuit &`
- Vous devez obtenir l'apparence précédente mais incomplète :
  - le bouton `on/off` n'a pas d'effet
  - il manque la zone de texte du bas (état des vannes).
- Compléter le code en suivant les commentaires `//GUIDE :...` pour :
  - ajouter un composant `JTextArea` pour afficher l'état des sorties (vanne)
  - mettre à jour les zones d'états quand on commute l'interrupteur sélectionné en cliquant sur `on/off`.
- Compiler et réessayer.