INFORMATIQUE INDUSTRIELLE

Licence SEICOM – I.U.T. de Nantes

TP 2 : Programmation orientée objet avec l'atelier de développement Eclipse

Module M2-1 édition 2012

1. OBJECTIFS

Etre capable de :

- Manipuler un objet, ses différents membres en prenant en charge leur encapsulation ;
- Définir et utiliser de nouvelles classes dérivées d'une classe existante ;
- Comprendre l'utilisation de techniques POO : agrégation, classes abstraites, ...

⇒ **Les travaux à faire dans ce TP sont repérés ainsi.**

⇒ **Dans votre compte rendu vous ferez apparaître les réponses aux questions, les extraits de codes utiles (commentés) ainsi qu'un diagramme de classe général regroupant toutes les classes du projet.**

2. CONTEXTE

2.1 L'environnement de travail Eclipse

Eclipse est un environnement de travail gratuit que vous pouvez librement télécharger sur le site www.eclipse.org. Très convivial, il permet :

- d'éditer, en Java, chaque classe ;
- d'instancier un objet d'une classe et d'exécuter les méthodes de cet objet ;
- de contrôler cette exécution avec des points d'arrêt, d'observer le contenu de variables etc.
- de générer la documentation associée aux différentes classes.

De plus, un didacticiel est intégré dans l'environnement de travail Eclipse. Pour l'obtenir, cliquer sur menu « Aide », onglet « Bienvenue ».

2.2 Documentation Java

La documentation de l'API Java est accessible dans un dossier sur le répertoire partagé \\Zeus\etucom, sous le répertoire « \CommunGeneral\Java\docs\api\ » : double-cliquer sur « *index.html* » (ici) ou sur internet sur le site java.com.

2.3 Cahier des charges

Le but du projet est de développer une application Java, qui permet de simuler le fonctionnement de circuits logiques complexes.

Un circuit intégré logique est doté :

- d'entrées (de 0 à plusieurs)
- de sorties (de 0 à plusieurs)

et réalise une fonction logique simple (une porte ET, OU, OU EXCLUSIF (XOR), NON), une fonction logique complexe (multiplexeur, additionneur, …) ou gère la communication avec l'extérieur (interrupteur, micro-switch, led, …).

Entrées et sorties sont câblées par des fils (classe Fil) qui sont chargés *logiquement* (valeurs 0, 1 ou *indéfinie* (2), codées respectivement par les constantes de la classe FIL : ZERO, UN et X).

Comme dans la réalité, pour connecter l'entrée d'un circuit avec la sortie d'un autre circuit, il faut que ces circuits partagent une même occurrence de la classe Fil (un seul objet Fil).

Pour réaliser la simulation, les circuits sont dotés d'une méthode *simuler()* qui utilise les valeurs sur les fils d'entrée pour affecter une valeur aux fils de sortie.

3. TRAVAIL À RÉALISER

3.1 Création de votre projet

Sur Madoc, figurent les sources des programmes sur lesquels nous allons travailler.

⇒ **Lancer Eclipse**

⇒ **Sélectionner votre espace de travail (workspace)**

⇒ **Créer un nouveau projet Java nommé « SimulateurCircuits »**

⇒ **Importer dans le package par défaut les fichiers sources :**

- **Circuit.java**
- **Fil.java**
- **PorteET.java**
- **PorteOU.java**
- **TesteurComposant.java**

Notez que eclipse copie ces fichiers dans votre espace de travail.

⇒ **Corriger les éventuelles erreurs qui apparaissent en marge (marquées d'une « croix rouge »).** Pour cela, positionner le curseur sur la croix de la ligne en erreur et cliquer avec le bouton gauche : Eclipse propose une correction parmi d'autres, la mieux appropriée est en général la première de la liste. Enregistrer après correction, et la croix rouge doit passer alors en couleur grisée.

⇒ **Exécuter le main() des classes PorteET et PorteOU grâce au menu « Run > Run As > Application Java »**

La classe TesteurComposant a pour but de tester toutes les valeurs possibles en entrée d'un composant et d'afficher la valeur de sortie correspondante. Si vous tester cette classe en mode normal vous constaterez qu'une erreur persiste :

⇒ **Exécuter le main de la classe TesteurComposant en mode Debug pour trouver l'erreur (mettre un point d'arrêt dans la méthode *tester()*).**

3.2 Développements

3.2.1 Extension aux portes NON et XOR

⇒ **Sur le modèle des classes PorteET et PorteOU, proposez une implémentation pour les classes PorteNON et PorteXOR.**

⇒ **Tester vos classes grâce à la classe TesteurComposant**

3.2.2 Nommage des éléments

Vous remarquerez que tous les éléments (fils et composants) n'ont pas de nom. Il ne sera pas facile de distinguer les différents fils ou composants lorsque plusieurs d'entre eux seront utilisés en même temps.

Comme ce principe de nommage est un concepts commun à toutes les classes utilisées, nous définirons une classe *Nommage* avec un seul attribut *nom*, des constructeurs adaptés, une méthode publique getNom() et la méthode *toString()* surchargée (héritée de java.lang.Object).

⇒ **Proposer une implémentation de cette classe *Nommage***

⇒ **Faite hériter les classes Fil et Circuit de Nommage. Modifier si besoin les constructeurs disponibles. Modifier les méthodes toString de toutes les classes pour faire apparaître le nom des fils et des portes.** Exemple d'affichage d'une PorteET nommée « p1 » construite avec 3 fils nommés « a », « b » et « y » : « porteET p1 (a=1,b=0) [y=0] »

3.3 Circuits Complexes

Dans cette partie nous développerons des circuits plus complexes composés de plusieurs circuits de base. Les différents circuits possible hériterons d'une classe abstraite CircuitComplexe regroupant les propriétés et méthodes communes à tous.

3.3.1 Classe CircuitComplexe

Un circuit complexe étant tout d'abord un circuit comme un autre (avec des entrées-sorties, un nom, …), cette classe devra hériter de la classe Circuit. Comme un circuit complexe est composé de plusieurs objets Circuit dont les références seront mémorisés dans un tableau (attribut de CircuitComplexe).

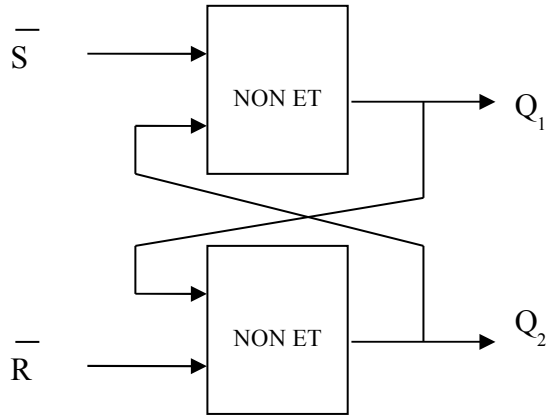
⇒ **Proposer une implémentation de cette classe (attribut et constructeurs)**

⇒ **La méthode *simuler()* de Circuit sera commune à tous les circuits complexes, proposer une implémentation.**

⇒ **Pourquoi cette classe qui ne contient pas de méthode abstraites doit-elle pourtant rester abstraite ?**

3.3.2 Exemples de circuits complexes bascule RS

⇒ **Réaliser une classe BasculeRS dont le shéma est donné ci dessous**



⇒ **Proposer une implémentation pour un cicuit additionneur complet de deux bits (classe AddComp2Bit)**

Un additionneur 2 bits est la brique de base d'un additionneur n bits. Il permet de faire la somme de 2 valeurs sur 1 bit (0+0=0; 0+1=1; 1+1=0 + retenue). Un additionneur complet prend en compte une retenue précédente. Il a donc

- 3 entrées : a et b les bits à additionner et r la retenue précédente
- 2 sorties : s la somme et R la retenue

On a les relations suivantes :

$$s = a \oplus b \oplus s \qquad R = ab + ar + br$$