

ESISAR GRENOBLE-INP



CS312 Programmation Objet

TP4 : Simulateur de Circuit Logique

El Mehdi Khalfi, Ioannis Parissis, Oum-El-Kheir Aktouf

On souhaite réaliser un système de simulation de circuits logiques constitué d'opérateurs qui sont des composants élémentaires permettant des assemblages ayant des fonctions particulières.

Ces composants que l'on qualifie souvent de « portes » peuvent soit transformer des entrées en des sorties (Transformateur) comme les portes « And », « Or », « Not », ... ; soit recevoir des niveaux logiques en entrée (Récepteur) comme une « Led », ...

Un composant est donc un opérateur qui possède zéro, un ou plusieurs (maximum deux dans ce TP) entrées, et zéro ou une sortie. Mais en tout état, il possède une entrée (le cas d'une « Led ») ou une sortie (le cas d'un « Button »). Par exemple, le composant « And » possède deux entrées et une sortie. Les nombres d'entrées/sorties d'un type de composant sont immuables.

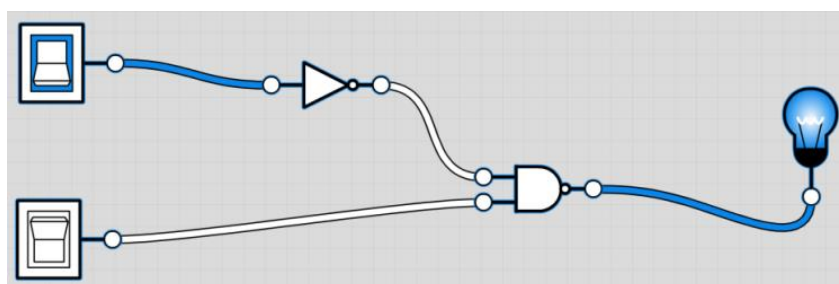


Figure 1 : Circuit Logique (depuis <http://logic.ly/demo>)

Un circuit est un assemblage de composants reliés entre eux par des connexions. Une connexion est orientée et relie une sortie d'un composant à l'entrée d'un autre. Une entrée d'un composant ne peut recevoir qu'une connexion, et une sortie ne peut être reliée qu'à une seule entrée.

Nous nous intéressons, dans ce TP, à des circuits particuliers appelés « circuits fermés », ceux pour lesquels aucune porte d'entrée ou de sortie n'est libre, c.-à-d. qu'il existe une connexion partant de chaque porte de sortie et une connexion arrivant sur chaque porte d'entrée. Nous considérons qu'un circuit n'ayant aucune porte libre est un circuit fermé. De plus, ce type de circuit doit contenir une seule « Led » qui recevra le résultat de la simulation du circuit.

Pour représenter les circuits, on propose un format textuel (xml¹) qui décrit les composants et les connexions qui les relient. Comme illustré dans la Figure 2, chaque composant est identifié par le nom du type auquel il appartient (par exemple, `<logicGate>LED</logicGate>`), ses coordonnées dans l'outil graphique (`<led x="410" y="195">`), et les autres composants reliés à ses portes d'entrée (`<in1></in1>` et `<in2></in2>`).

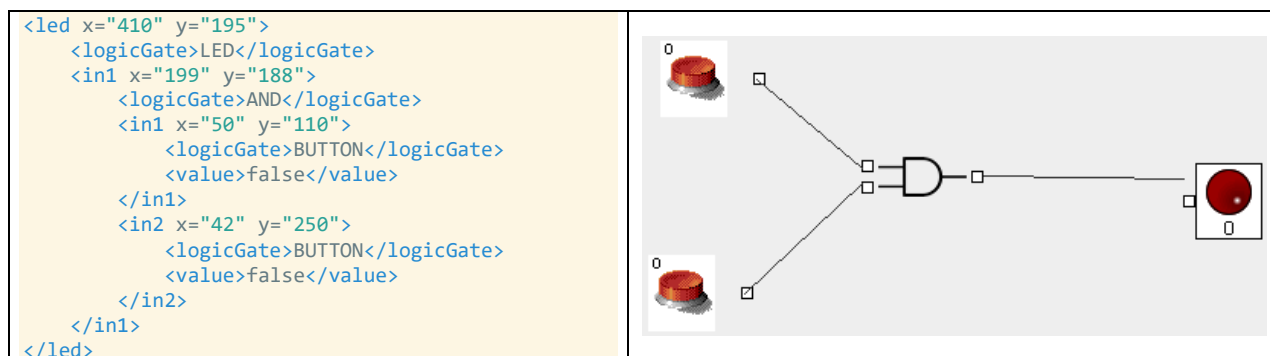


Figure 2 : Représentation Textuelle d'un Circuit Logique

¹ https://fr.wikipedia.org/wiki/Extensible_Markup_Language

Description Technique

Mise en Place de l'Application

- Télécharger le fichier TP.zip (sans le décompresser).
- Depuis Eclipse, cliquer sur File -> Import -> Existing Projects into Workspace
- Ensuite, sélectionner « Select archive file », et chercher le fichier .zip déjà téléchargé.
- Une fois le projet est exporté sur Eclipse, ajouter le bytecode de certaines classes déjà implémentées.
 - Cliquer avec le bouton droit de la souris sur le projet (TP)
 - Choisir Build Path -> Configure Build Path.
 - Dans l'onglet « Libraries », Cliquer sur « Add External Class Folder »
 - Sélectionner le répertoire « classes » (déjà contenu dans le projet Eclipse). Pour les utilisateurs de Windows (si on ne peut pas parcourir l'arborescence pour sélectionner « classes »), copier le chemin absolu du répertoire « classes » dans le champ « Dossier ».

Interface Graphique

- Le point d'entrée de l'application est la fonction `main` de la classe `CircuitLogiqueJFrame`. Cette classe hérite de `JFrame`, et représente la fenêtre principale de l'interface graphique. Elle contient `contentPane` comme panneau principal. Ce dernier inclut une barre d'outils et un autre panneau de type `LogicGateSimulatorJPanel` pour la réalisation de circuits logiques.
- Un panneau de type `LogicGateSimulatorJPanel` contient deux autres panneaux pour le choix de portes logiques et la réalisation du circuit logique.
- Chaque porte positionnée dans le `drawingPanel` est elle-même un panneau contenant (i) l'icône correspondant à la porte logique, et (ii) les connecteurs d'entrée/sortie.
- Les évènements (clic de souris, glisser déposer, ...) de chaque composant (panneau, bouton, fenêtre, label, ...) sont dans la classe correspondante.

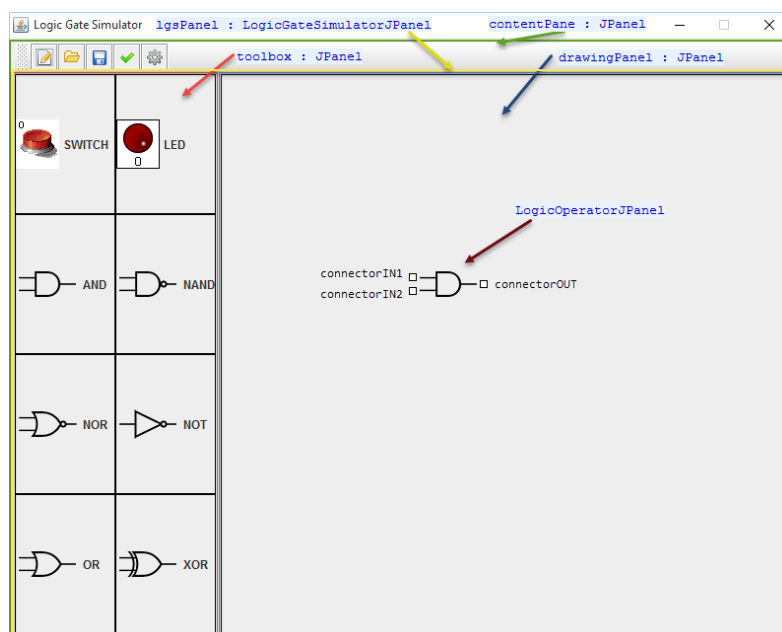







Figure 3 : Composants de l'Application

Partie 1 : IHM

Description de l'Application

L'application à implémenter est divisée en trois parties :

- ✓ Barre d'outils qui permet de :
 -  Créer un nouveau circuit logique,
 -  Charger un fichier (*.boole) contenant un circuit logique,
 -  Sauvegarder un circuit logique dans un fichier (*.boole),
 -  Vérifier si un circuit logique est fermé,
 -  Simuler le résultat d'un circuit logique.
- ✓ Boîte à outils contenant des portes logiques.
- ✓ Espace de dessin pour réaliser un circuit logique.

Question 1


- Lancer l'application à partir d'Eclipse (Bouton droit sur le fichier `CircuitLogiqueJFrame`, ensuite choisir « Run As -> Java Application »).
- Charger le fichier `test.boole`.
- Vérifier que le circuit est bien fermé.
- Vérifier le résultat de la simulation du circuit logique.

Question 2

Décrire les étapes d'initialisation contenues dans le constructeur de la classe

`LogicGateSimulatorJPanel`.

Question 3

- Pour créer un nouveau circuit logique, choisir une porte logique à partir de la boîte à outils, le curseur de la souris change d'icone, telle qu'elle correspond à la porte logique choisie.
- Pour positionner une porte logique sur `drawingPanel`, il faut compléter le corps de l'évènement `mouseClick` dans le constructeur de `LogicGateSimulatorJPanel`, de sorte que :
 - Si on clique sur le bouton droit, on annule le choix d'une porte :
 - Le type de porte logique sélectionnée est remis à `NONE`.
 - L'icône du curseur de la souris est remise à celle par défaut ().
 - Si on clique sur le bouton gauche de la souris :
 - On ajoute une porte logique (une instance de `LogicOperatorJPanel`) du type sélectionné à l'espace de dessin. Un opérateur est initialisé comme suit :
 - `g` : `null` dans le cas de création d'une nouvelle porte, sinon, l'instance d'une porte déjà existante.
 - `logicgate` : Type de la porte logique.
 - `x`, `y` : position de la porte dans le `drawingPanel`.
- Pour créer des connexions, à partir d'un connecteur de sortie d'une porte, cliquer et maintenir appuyé avec le bouton gauche de la souris pour relier cette sortie avec le connecteur d'entrée d'une autre porte logique.
- Sauvegarder le circuit réalisé.

Partie 2 : IHM

Importer la nouvelle version de TP2.zip. (Il faut d'abord supprimer le projet « TP » de la dernière séance, ne pas oublier de cocher l'option « *delete project contents on disk* »).

Question 1

Quelle est la différence entre `MouseListener`, `MouseAdapter` et `MouseMotionListener`, `MouseMotionAdapter` ?

A quoi sert chacun des évènements suivants : `mouseClicked`, `mousePressed`, `mouseReleased`, `mouseDragged`, `mouseEntered`, `mouseExited` ?

Dans cette version du TP, on remarque qu'après avoir positionné un opérateur logique dans `drawingPanel`, il restera immobile et on ne peut pas le glisser/déposer sur l'espace du dessin.

Question 2

Implémenter cette fonctionnalité en ajoutant des écouteurs d'évènements dans le constructeur de `LogicOperatorJPanel`. **La façon la plus simple consiste à définir la position de l'opérateur comme étant la position de la souris en faisant glisser l'opérateur.** Mais, ce n'est pas la solution la plus réaliste !

Indications :

Se servir des évènements `mousePressed` et `mouseDragged`.

Raisonnement par la position absolue (dans l'écran de la machine) de l'opérateur logique (et non pas sa position dans l'interface graphique).

Comme illustré dans la Figure 3, Tout opérateur a -au plus- deux connecteurs d'entrée et un connecteur de sortie. Pour une porte logique comme « Button », on ne lui ajoute pas `connectorIN1` et `connectorIN2`. « Not », par exemple, ne contient pas `connectorIN2`.

Question 3

Une connexion est un lien entre **deux** opérateurs. Pour créer une connexion, on relie -avec la souris- le `connectorOUT` d'un opérateur vers `connectorIN1` ou `connectorIN2` d'un autre. Cette fonctionnalité est implémentée dans des évènements ajoutés à ces trois connecteurs.

- Décrire les comportements des évènements `mouseEntered` et `mouseExited` dans chacun des trois connecteurs (`connectorIN1`, `connectorIN2`, `connectorOUT`).

Implémenter la création d'une connexion telle que :

Il faut se servir de `xBegin`, `xEnd`, `yBegin`, `yEnd`, qui représentent les coordonnées des deux extrémités de la connexion (Le dessin de la connexion étant déjà implémenté, il suffit de fournir les bonnes coordonnées).

Partie 3 : Opérations Logiques

Supprimer le répertoire « classes » du « build path » :

- Cliquer avec le bouton droit de la souris sur le projet (TP)
- Sélectionner « Build Path » -> « Configure Build Path »
- Dans l'onglet « Libraries », sélectionner la ligne contenant « classes », et cliquer sur « Remove »
- Maintenant que Eclipse ne connaît plus les types `and`, `button`, `gate`, `led`, `LogicGate`, `nand`, `nor`, `not`, `or`, `xor`, l'objectif est de les implémenter.

Commencer par créer un package nommé « gates » (Cliquer sur le bouton droit de la souris sur TP, ensuite sélectionner « new » -> « Package »)

Dans le package « gates », ajouter :

- Une énumération nommée « LogicGate » (new -> Enum). La définition de l'énumération est comme suit :

```
package gates;

public enum LogicGate {
    NONE, AND, NAND, NOR, NOT, OR, XOR, BUTTON, LED
}
```

Question 1

Quelles sont les différences entre une classe et une énumération en Java ?

- Une classe nommée « gate » :

```
package gates;

import javax.xml.bind.annotation.XmlAttribute;
import ihm.LogicOperatorJPanel;

/**
 * Cette classe simule le comportement d'un port logique
 *
 */

public abstract class gate {
    /**
     * Le type de la porte logique
     */
    public LogicGate logicGate;

    /**
     * Les coordonnées de la porte logique dans le dessin.
     */
    int x, y;

    /**
     * Les portes connectées dans les deux connecteurs in1 et in2.
     */
    gate in1, in2 = null;

    public gate getIn1() {
        return in1;
    }
}
```

```

public void setIn1(gate in1) {
    this.in1 = in1;
}

public gate getIn2() {
    return in2;
}

public void setIn2(gate in2) {
    this.in2 = in2;
}

public gate() {
}

public gate(int x, int y) {
    this.x = x;
    this.y = y;
}

public int getX() {
    return x;
}

@XmlAttribute
public void setX(int x) {
    this.x = x;
}

public int getY() {
    return y;
}

@XmlAttribute
public void setY(int y) {
    this.y = y;
}

/**
 * Valider s'il un connecteur non occupé dans le circuit
 */
abstract public boolean ValiderCircuitFerme();

/**
 * Retourner le résultat de l'opération logique .
 */
abstract public boolean Process();

public void GetGatesFromRoot() {
    LogicOperatorJPanel.lGates.add(this);
    if (this.in1 != null) {
        this.in1.GetGatesFromRoot();
    }
    if (this.in2 != null) {
        this.in2.GetGatesFromRoot();
    }
}
}

```

Question 2

Quelles sont les différences entre une classe abstraite et une interface en Java ?

À quoi sert une fonction abstraite ?

- Une classe Java (new -> Class) par opérateur logique (and, button, led, nand, nor, not, or, xor) qui hérite de la classe gate.
 - Ajouter l'annotation `@XmlRootElement` avant la définition de chaque classe (avant `public class ...`)
 - Pour la classe `button`, ajouter un attribut `boolean value = false;`

Question 3

Quelles sont les méthodes qui doivent être implémentées dans toutes les classes qui héritent de `gate` ? Pourquoi ?

Implémenter ces fonctions pour chacune des classes filles de `gate`.