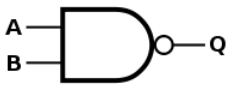
Programmation Java @ Et3
Polytech Paris-Sud | 2024-25

Nom : BENCHIKHI
Prénom : Anas

Exercice 4

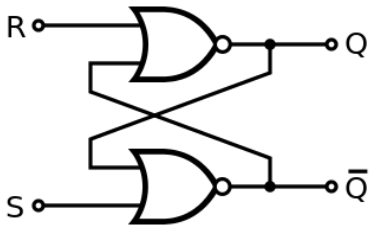
Les *portes logiques* sont des circuits électroniques qui possèdent des *entrées* et des *sorties* sur lesquelles on place et récupère des valeurs de *bits*. Les portes logiques ET-NON (*nand*) possèdent deux entrées **A** et **B**. La sortie **Q** est au niveau 0 (ou *false*) si toutes les entrées sont au niveau 1 (ou *true*). Une seule entrée au niveau 0 suffit pour que la sortie soit à 1. Les symboles de ce type de portes est représenté ci-dessous :



Les portes logiques OU-NON (*nor*), qui possèdent également deux entrées **A** et **B**, ont leur sortie au niveau 1 si toutes les entrées sont au niveau 0. Une seule entrée au niveau 1 suffit pour que la sortie soit à 0. Ce type de porte a pour symbole :

Les *circuits séquentiels* sont obtenus par combinaison de portes logiques, avec certaines sorties qui sont rebouclées en entrée. Contrairement aux portes logiques, l'état des sorties des circuits séquentiels dépend non seulement de l'état des entrées mais aussi de l'état antérieur des sorties: ce sont des circuits *dotés de mémoire*.

Les *bascules RS* sont des bistables qui peuvent prendre deux états. Ces bascules sont asynchrones. Elles possèdent deux entrées nommées **R** et **S** et deux sorties complémentaires nommées **Q** et $\bar{\mathbf{Q}}$. L'entrée **S** (set) met la bascule au travail et la sortie **Q** à la valeur 1. L'entrée **R** (reset) remet la bascule au repos et la sortie **Q** à la valeur 0. La représentation d'une bascule RS constituée de portes *nor* est donnée ci-dessous :



On envoie séparément et alternativement les signaux sur **S** et **R**. Dans le cas particulier où **S** = **R** = 1, l'état de la sortie est indéterminé. La table de vérité en fonction de l'état précédent $\mathbf{Q}_n$ est la suivante :

Q_n	S	R	Q_{n+1}
1	1	0	1
1	0	1	0
1	1	1	?
1	0	0	1
0	1	0	1
0	0	1	0
0	0	0	0
0	1	1	?

1. On souhaite modéliser des portes *nand* et *nor* par des classes en factorisant au mieux le code. Toute modification d'une des entrées doit entraîner la mise à jour de la sortie de la porte. Écrivez les classes nécessaires en les dotant de méthodes utiles pour l'affichage de l'état de leurs objets ainsi que des méthodes d'altération (*setters*) et de consultation (*getters*) utiles.

#4.1 code

```
abstract class PorteLogique {
    private boolean entree1;
    private boolean entree2;
    private boolean sortie;

    public PorteLogique() {
        this.entree1 = false;
        this.entree2 = false;
        this.sortie = calculerSortie();
    }

    public void setEntree1(boolean valeur) {
        this.entree1 = valeur;
        this.sortie = calculerSortie();
    }

    public void setEntree2(boolean valeur) {
        this.entree2 = valeur;
        this.sortie = calculerSortie();
    }

    public boolean getEntree1() {
        return entree1;
    }

    public boolean getEntree2() {
        return entree2;
    }

    public boolean getSortie() {
        return sortie;
    }
```

```
    protected abstract boolean calculerSortie();

    public void afficherEtat() {
        System.out.println("Entrée 1: " + entree1 + ", Entrée 2: "
+ entree2 + ", Sortie: " + sortie);
    }
}

class PorteNAND extends PorteLogique {
    @Override
    protected boolean calculerSortie() {
        return !(getEntree1() && getEntree2());
    }
}

class PorteNOR extends PorteLogique {
    @Override
    protected boolean calculerSortie() {
        return !(getEntree1() || getEntree2());
    }
}
```

2. Réalisez des tests au niveau de la classe permettant d'obtenir la table de vérité de chacune des portes. Pour cela, passez par des variables du type de votre classe représentant une porte logique (les méthodes invoquées ne dépendant pas du type concret particulier).

#4.2 code et exécution

```
abstract class PorteLogique {
    private boolean entree1;
    private boolean entree2;
    private boolean sortie;

    public PorteLogique() {
        this.entree1 = false;
        this.entree2 = false;
        this.sortie = calculerSortie();
    }

    public void setEntree1(boolean valeur) {
        this.entree1 = valeur;
        this.sortie = calculerSortie();
    }

    public void setEntree2(boolean valeur) {
        this.entree2 = valeur;
        this.sortie = calculerSortie();
    }

    public boolean getEntree1() {
        return entree1;
```

```
    }

    public boolean getEntree2() {
        return entree2;
    }

    public boolean getSortie() {
        return sortie;
    }

    protected abstract boolean calculerSortie();

    public void afficherEtat() {
        System.out.println("Entrée 1: " + entree1 + ", Entrée 2: "
+ entree2 + ", Sortie: " + sortie);
    }

    public void genererTableVerite() {
        System.out.println("Table de vérité:");
        System.out.println("Entrée 1 | Entrée 2 | Sortie");
        System.out.println("-------------------------");
        for (boolean e1 : new boolean[]{false, true}) {
            for (boolean e2 : new boolean[]{false, true}) {
                setEntree1(e1);
                setEntree2(e2);
                System.out.println("   " + e1 + "     |    " + e2
+ "     |  " + getSortie());
            }
        }
    }
}

class PorteNAND extends PorteLogique {
    @Override
    protected boolean calculerSortie() {
        return !(getEntree1() && getEntree2());
    }
}

class PorteNOR extends PorteLogique {
    @Override
    protected boolean calculerSortie() {
        return !(getEntree1() || getEntree2());
    }
}

public class Main {
    public static void main(String[] args) {
        // Tests pour les portes logiques
        testerPorteLogique(new PorteNAND(), "NAND");
        testerPorteLogique(new PorteNOR(), "NOR");
    }
```

```
    public static void testerPorteLogique(PorteLogique porte,
String nomPorte) {
        System.out.println("Table de vérité pour la porte " +
nomPorte + ":\n");
        porte.genererTableVerite();
        System.out.println();
    }
}
```

3. Écrivez à présent une classe modélisant une bascule RS composée de 2 portes *nor*, et implémentant les méthodes :
— `getS`, `getR`, `getQ`, `getNonQ` pour obtenir l'état des valeurs correspondantes ;
— les méthodes `setS` et `setR` pour changer l'entrée de la bascule et calculer un nouvel état ;
— une méthode pour afficher l'état de la bascule.

#4.3 code

```
class BasculeRS {
    private PorteNOR porte1; // NOR pour S et Q
    private PorteNOR porte2; // NOR pour R et non-Q
    private boolean S;
    private boolean R;

    public BasculeRS() {
        porte1 = new PorteNOR();
        porte2 = new PorteNOR();
        S = false;
        R = false;
        mettreAJourEtat();
    }

    public boolean getS() {
        return S;
    }

    public boolean getR() {
        return R;
    }

    public boolean getQ() {
        return porte1.getSortie();
    }

    public boolean getNonQ() {
        return porte2.getSortie();
    }

    public void setS(boolean valeur) {
        S = valeur;
        mettreAJourEtat();
```

```
    }

    public void setR(boolean valeur) {
        R = valeur;
        mettreAJourEtat();
    }

    private void mettreAJourEtat() {
        // Met à jour les entrées des portes NOR
        porte1.setEntree1(S);
        porte1.setEntree2(porte2.getSortie());

        porte2.setEntree1(R);
        porte2.setEntree2(porte1.getSortie());
    }

    public void afficherEtat() {
        System.out.println("État de la bascule RS:");
        System.out.println("S: " + S + ", R: " + R + ", Q: " +
getQ() + ", non-Q: " + getNonQ());
    }
}
```

4. On souhaite à présent ajouter aux portes logiques un nombre de cycles maximum au bout duquel la porte est à changer. On comptera comme cycle toute opération de lecture ou d'écriture sur la porte logique. Ajoutez le code nécessaire pour prendre en compte ce nombre de cycles maximum (qui pourra être défini au niveau de chaque porte particulière), puis mettez en œuvre la notion d'*exception* pour prendre en compte des erreurs au niveau des portes logiques ainsi qu'au niveau des bascules RS. Vous devrez avoir les trois types d'exceptions suivants : `ExceptionPorteLogique`, `ExceptionPorteAChanger` (sous-type de `ExceptionPorteLogique`), et `ExceptionBasculeAReparer` (qui devra renseigner sur quelle porte de la bascule doit être changée).

#4.4 code

```
//Les différentes exceptions :


class ExceptionPorteLogique extends Exception {
    public ExceptionPorteLogique(String message) {
        super(message);
    }
}

class ExceptionPorteAChanger extends ExceptionPorteLogique {
    public ExceptionPorteAChanger(String message) {
        super(message);
    }
}
```

```
class ExceptionBasculeAReparer extends Exception {
    private String porteDefectueuse;

    public ExceptionBasculeAReparer(String message, String
porteDefectueuse) {
        super(message);
        this.porteDefectueuse = porteDefectueuse;
    }

    public String getPorteDefectueuse() {
        return porteDefectueuse;
    }
}


//mise à jour de la classe porte logique


abstract class PorteLogique {
    private boolean entree1;
    private boolean entree2;
    private boolean sortie;
    private int cycles;
    private int maxCycles;

    public PorteLogique(int maxCycles) {
        this.entree1 = false;
        this.entree2 = false;
        this.sortie = calculerSortie();
        this.cycles = 0;
        this.maxCycles = maxCycles;
    }

    public void setEntree1(boolean valeur) throws
ExceptionPorteAChanger {
        incrementerCycles();
        this.entree1 = valeur;
        this.sortie = calculerSortie();
    }

    public void setEntree2(boolean valeur) throws
ExceptionPorteAChanger {
        incrementerCycles();
        this.entree2 = valeur;
        this.sortie = calculerSortie();
    }

    public boolean getEntree1() throws ExceptionPorteAChanger {
        incrementerCycles();
        return entree1;
    }

    public boolean getEntree2() throws ExceptionPorteAChanger {
```

```
        incrementerCycles();
        return entree2;
    }

    public boolean getSortie() throws ExceptionPorteAChanger {
        incrementerCycles();
        return sortie;
    }

    private void incrementerCycles() throws ExceptionPorteAChanger
{
        cycles++;
        if (cycles > maxCycles) {
            throw new ExceptionPorteAChanger("La porte logique a
dépassé son nombre maximum de cycles.");
        }
    }

    protected abstract boolean calculerSortie();
}


// mise à jour de bascule RS


class BasculeRS {
    private PorteNOR porte1; // NOR pour S et Q
    private PorteNOR porte2; // NOR pour R et non-Q
    private boolean S;
    private boolean R;

    public BasculeRS(int maxCyclesPorte) {
        porte1 = new PorteNOR();
        porte2 = new PorteNOR();
        S = false;
        R = false;
        try {
            mettreAJourEtat();
        } catch (ExceptionPorteLogique e) {
            System.out.println(e.getMessage());
        }
    }

    public boolean getS() {
        return S;
    }

    public boolean getR() {
        return R;
    }

    public boolean getQ() throws ExceptionBasculeAReparer {
        try {
```

```
            return porte1.getSortie();
        } catch (ExceptionPorteAChanger e) {
            throw new ExceptionBasculeAReparer("La porte 1 de la
bascule doit être changée.", "Porte 1");
        }
    }

    public boolean getNonQ() throws ExceptionBasculeAReparer {
        try {
            return porte2.getSortie();
        } catch (ExceptionPorteAChanger e) {
            throw new ExceptionBasculeAReparer("La porte 2 de la
bascule doit être changée.", "Porte 2");
        }
    }

    public void setS(boolean valeur) {
        S = valeur;
        try {
            mettreAJourEtat();
        } catch (ExceptionPorteLogique e) {
            System.out.println(e.getMessage());
        }
    }

    public void setR(boolean valeur) {
        R = valeur;
        try {
            mettreAJourEtat();
        } catch (ExceptionPorteLogique e) {
            System.out.println(e.getMessage());
        }
    }

    private void mettreAJourEtat() throws ExceptionPorteLogique {
        porte1.setEntree1(S);
        porte1.setEntree2(porte2.getSortie());

        porte2.setEntree1(R);
        porte2.setEntree2(porte1.getSortie());
    }

    public void afficherEtat() {
        try {
            System.out.println("S: " + S + ", R: " + R + ", Q: " +
getQ() + ", non-Q: " + getNonQ());
        } catch (ExceptionBasculeAReparer e) {
            System.out.println(e.getMessage() + " (" +
e.getPorteDefectueuse() + ")");
        }
    }
}
```