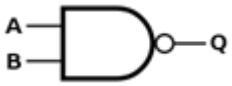
Programmation Java @ Et3
Polytech Paris-Sud | 2024-25

Nom : BRACQ
Prénom : Paolo

Exercice 4

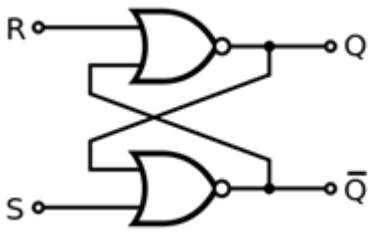
Les *portes logiques* sont des circuits électroniques qui possèdent des *entrées* et des *sorties* sur lesquelles on place et récupère des valeurs de *bits*. Les portes logiques ET-NON (*nand*) possèdent deux entrées **A** et **B**. La sortie **Q** est au niveau 0 (ou *false*) si toutes les entrées sont 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

```
// Classe parents
```

```
package Portes;

public class PorteLogique {
    protected boolean A;
    protected boolean B;
    protected boolean Q;

    /**
     * Constructeur de porte logique
     * @param A boolean
     * @param B boolean
     */
    public PorteLogique(boolean A, boolean B){
        this.A=A;
        this.B=B;
    updateQ();
}

// Méthode abstraite pour calculer la sortie
protected abstract void updateQ();

/**
 * Affichage de A
 * @return A boolean
 */
public boolean getA(){
    return A;

}
/**
 * Affichage de B
 * @return B boolean
 */
public boolean getB(){
    return B;
}
/**
```

```
 * Modification de A
 * @return
 */
public void setA(boolean a) {
    A=a;

    updateQ();
}
/**
 * Modification de B
 * @return
 */
public void setB(boolean b) {
    B=b;

    updateQ();
}
/**
 * Affichage de Q
 * @return Q boolean
 */
public boolean getQ(){

    return Q;
}
}
```

```
// Classe enfant
```

```
package Portes;

public class nand extends PorteLogique{
    /**
     * constructeur de porte nand
     * @param A boolean
     * @param B boolean
     */
    public nand (boolean A,boolean B){
        super(A,B, maxCycles);
        updateQ();
    }
    /**
     * Redéfintion de la méthode de mise à jour de la valeur Q
     */
    @Override
    protected void updateQ(){
        Q=! (A && B);
    }
}
```

```
package Portes;

public class nor extends PorteLogique{
    /**
     * Constructeur d'une porte nor
     * @param A boolean
     * @param B boolean
     */
    public nor (boolean A,boolean B){
        super(A,B, maxCycles);
        updateQ();
    }
```

```
    /**
     * Redéfintion de la méthode de mise à jour de la valeur Q
     */
    @Override
    protected void updateQ(){
        Q=! (A||B);
    }
}
```

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

```
package Portes;

public class Test_porte {
    public static void main(String[] args){
        // Test de la porte NAND
        System.out.println("Table de vérité de la porte NOR");
        nor norGate = new nor(true, false);
        System.out.println(" A: " + norGate.getA() + ", B: " + norGate.getB() +
", Q: " + norGate.getQ());

        norGate.setA(false);
        System.out.println("A: " + norGate.getA() + ", B: " + norGate.getB() +
", Q: " + norGate.getQ());

        norGate.setB(true);
        System.out.println(" A: " + norGate.getA() + ", B: " + norGate.getB() +
", Q: " + norGate.getQ());

        norGate.setA(true);
        norGate.setB(true);
        System.out.println("A: " + norGate.getA() + ", B: " + norGate.getB() +
", Q: " + norGate.getQ());

        nand nandGate = new nand(true, false);

        System.out.println("Table de vérité de la porte NANd");
        System.out.println( "A: " + nandGate.getA() + ", B: " + nandGate.getB()
+ ", Q: " + nandGate.getQ());

        nandGate.setA(false);
        System.out.println("A: " + nandGate.getA() + ", B: " + nandGate.getB() +
", Q: " + nandGate.getQ());

        nandGate.setB(true);
        System.out.println(" A: " + nandGate.getA() + ", B: " + nandGate.getB()
+ ", Q: " + nandGate.getQ());

        nandGate.setA(true);
        nandGate.setB(true);
        System.out.println("A: " + nandGate.getA() + ", B: " + nandGate.getB() +
", Q: " + nandGate.getQ());
    }
```

```
}
```

```
Table de vérité de la porte NOR
A: true, B: false, Q: false
A: false, B: false, Q: true
A: false, B: true, Q: false
A: true, B: true, Q: false
Table de vérité de la porte NANd
A: true, B: false, Q: true
A: false, B: false, Q: true
A: false, B: true, Q: true
A: true, B: true, Q: false

Process finished with exit code 0
```

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

```
package Portes;

public class BasculeRS {
    private boolean S;
    private boolean R;
    private boolean Q;
    private boolean nonQ;
    private nor nor1;
    private nor nor2;

    public BasculeRS(boolean S, boolean R){
        this.S=S;
        this.R=R;
        this.Q = false;
        this.nonQ = true;
        this.nor1 = new nor(R,nonQ);
        this.nor2 = new nor(S, Q);
        majEtat();
    }
    public boolean getS(){
        return S;
    }
    public boolean getR(){
        return R;
    }
    public boolean getQ(){
        return Q;
    }
    public boolean getNonQ(){
```

```
        return nonQ;
    }
    public void setR(boolean r) {
        R=r;
        majEtat();
    }
    public void setS(boolean s) {
        S=s;
        majEtat();
    }
    private void majEtat() {
        nor1.setA(S);
        nor1.setB(nonQ);
        Q = nor1.getQ();

        nor2.setA(R);
        nor2.setB(Q);
        nonQ = nor2.getQ();
    }

    public void etatBascule(){
        System.out.println("S: "+S+ " R:" + R+ " Q:"+Q+" NonQ: "+ nonQ);
    }
```

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

```
package Portes;

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

```
package Portes;

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

```
package Portes;

public class ExceptionBasculeAReparer extends Exception {
    private String porte;
```

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

    public String getPorte() {
        return porte;
    }
}
```

```
package Portes;

abstract class PorteLogique {
    protected boolean A;
    protected boolean B;
    protected boolean Q;
    protected int maxCycles;
    protected int cycles;

    /**
     * Constructeur de porte logique
     *
     * @param A         boolean
     * @param B         boolean
     * @param maxCycles int
     */
    public PorteLogique(boolean A, boolean B, int maxCycles) {
        this.A = A;
        this.B = B;
        this.maxCycles = maxCycles;
        this.cycles = 0;
        updateQ();
    }

    // Méthode abstraite pour calculer la sortie
    protected abstract void updateQ();

    /**
     * Affichage de A
     *
     * @return A boolean
     */
    public boolean getA() throws ExceptionPorteLogique {
        incrementCycles();
        return A;

    }

    /**
     * Affichage de B
     *
     * @return B boolean
     */
    public boolean getB() throws ExceptionPorteLogique {
        incrementCycles();
        return B;
    }

    /**
     * Modification de A
     *
     * @return
```

```
     */
    public void setA(boolean a) throws ExceptionPorteLogique {
        A = a;
        incrementCycles();
        updateQ();
    }

    /**
     * Modification de B
     *
     * @return
     */
    public void setB(boolean b) throws ExceptionPorteLogique {
        B = b;
        incrementCycles();
        updateQ();
    }

    /**
     * Affichage de Q
     *
     * @return Q boolean
     */
    public boolean getQ() throws ExceptionPorteLogique {
        incrementCycles();
        return Q;
    }

    protected void incrementCycles() throws ExceptionPorteLogique {
        cycles++;
        if (cycles > maxCycles) {
            throw new ExceptionPorteAChanger("La porte logique doit être chan-
gée.");
        }
    }
}
```

```
package Portes;

public class nor extends PorteLogique{
    /**
     * Constructeur d'une porte nor
     * @param A boolean
     * @param B boolean
     */
    public nor (boolean A,boolean B, int maxCycles){
        super(A,B, maxCycles);
        updateQ();
    }

    /**
     * Redéfintion de la méthode de mise à jour de la valeur Q
     */
    @Override
    protected void updateQ(){
        Q=! (A||B);
    }
}
```

```
package Portes;

public class nand extends PorteLogique{
```

```
    /**
     * constructeur de porte nand
     * @param A boolean
     * @param B boolean
     */
    public nand (boolean A,boolean B, int maxCycles){
        super(A,B, maxCycles);
        updateQ();
    }
    /**
     * Redéfintion de la méthode de mise à jour de la valeur Q
     */
    @Override
    protected void updateQ(){
        Q=! (A && B);
    }
}
```

```
package Portes;

public class BasculeRS {
    private boolean S;
    private boolean R;
    private boolean Q;
    private boolean nonQ;
    private nor nor1;
    private nor nor2;

    public BasculeRS(boolean S, boolean R, int maxCycles)throws ExceptionPorte-
Logique, ExceptionBasculeAReparer{
        this.S=S;
        this.R=R;
        this.Q = false;
        this.nonQ = true;
        this.nor1 = new nor(S,true, maxCycles);
        this.nor2 = new nor(R, false, maxCycles);
        majEtat();
    }
    public boolean getS() throws ExceptionPorteLogique{
        return S;
    }
    public boolean getR()throws ExceptionPorteLogique{
        return R;
    }
    public boolean getQ()throws ExceptionPorteLogique{
        return Q;
    }
    public boolean getNonQ()throws ExceptionPorteLogique{
        return nonQ;
    }
    public void setR(boolean r)  throws ExceptionPorteLogique, ExceptionBascu-
leAReparer{
        R=r;
        majEtat();
    }
    public void setS(boolean s)  throws ExceptionPorteLogique, ExceptionBascu-
leAReparer {
        S=s;
        majEtat();
    }
    private void majEtat() throws ExceptionPorteLogique, ExceptionBasculeARepa-
rer {
        try {
```

```
            nor1.setA(R);
            nor1.setB(nonQ);
            Q = nor1.getQ();

            nor2.setA(S);
            nor2.setB(Q);
            nonQ = nor2.getQ();
        } catch (ExceptionPorteAChanger e) {
            // Catch and wrap specific exception
            throw new ExceptionBasculeAReparer("La bascule RS doit être répa-
rée.", e.getMessage());
        }
    }

    public void etatBascule(){
        System.out.println("S: "+S+ " R:" + R+ " Q:"+Q+" NonQ: "+ nonQ);
    }

}
```