Laboratório de Engenharia de Software

Lista 1

Nome	RA
Andressa Almeida	1680481522041
Daniel Augusto	1680481521030
Diego de Melo	1680481521036
Elias Sanai	1680481521016

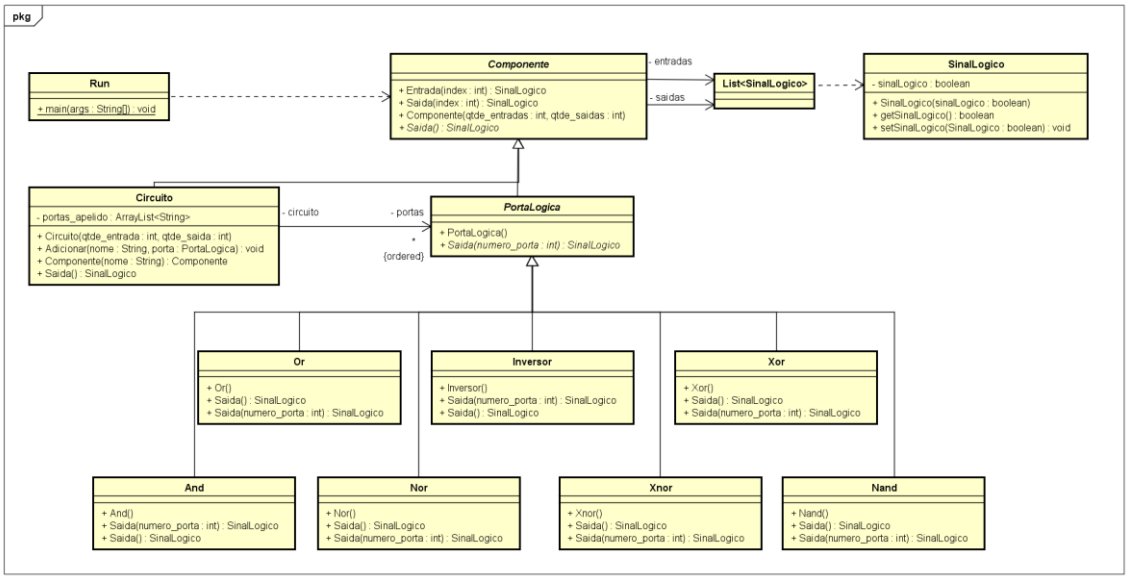
Requisito	Descrição
R001	O sistema deverá permitir a criação de **portas lógicas**: elementos simples que executarão operações lógicas tais como AND, OR, NOT, XOR etc.
R002	As portas lógicas deverão ter de 1 a 2 entradas e possuir apenas uma única saída que é a aplicação da sua função às entradas.
R003	O sistema deverá permitir a criação de **subsistemas lógicos**: um conjunto internamente conectado de portas lógicas realizando uma operação que é consequência dessas conexões.
R004	O número mínimo de entradas e de saídas dos subsistemas lógicos é 1 e o número máximo é arbitrário.
R005	O sistema deverá permitir a simulação dos circuitos por meio de um programa de testes que implemente tabelas verdade para os circuitos definidos.
R006	O sistema deverá ser implementado com uma interface em linha de comando, apresentando os testes de modo organizado.

O que é para fazer?

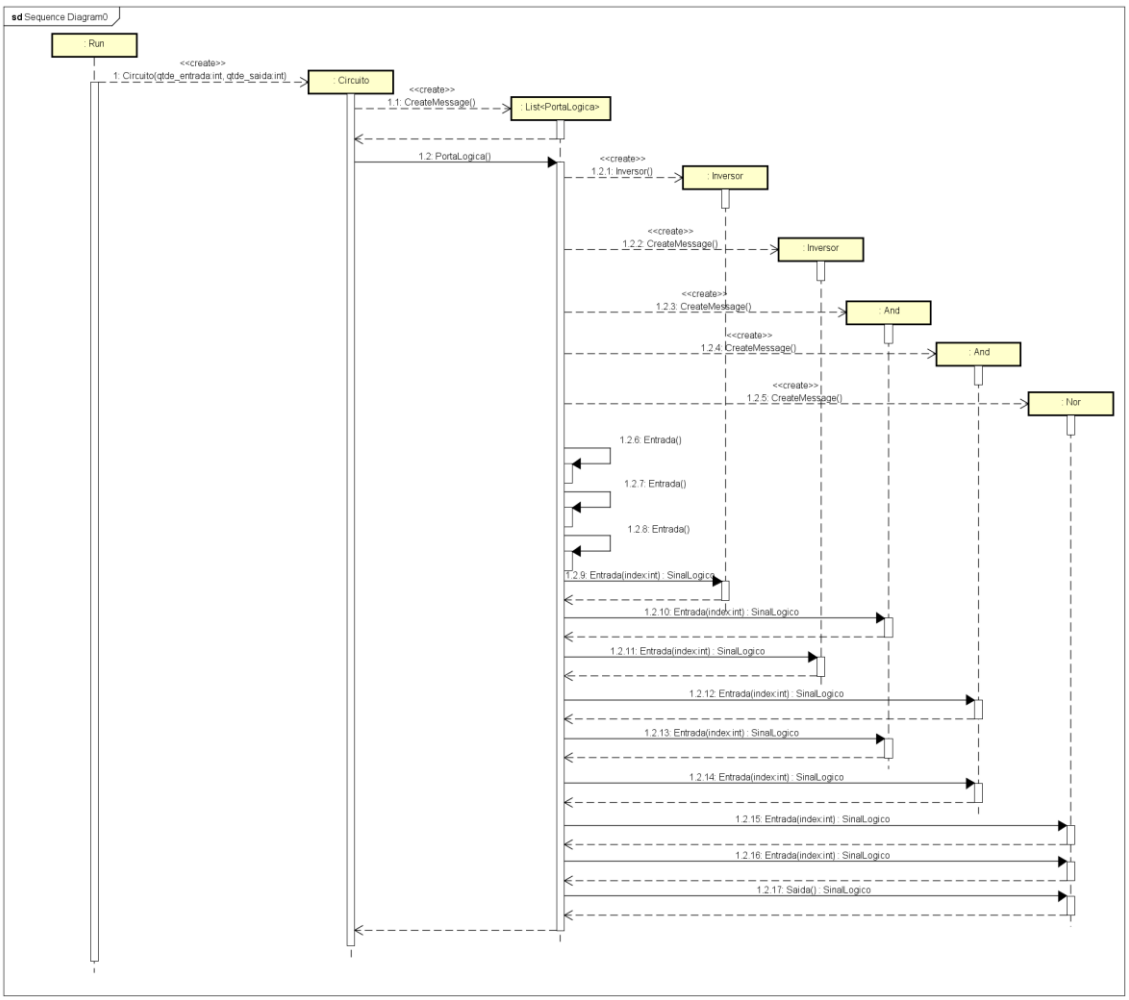
1. Procurar padrões de projeto que poderiam ser utlizados na resolução deste problema: indicar o nome, referência e justfcar sua escolha.

2. Elaborar um diagrama UML de classes do projeto do sistema e um diagrama de sequências UML que exemplifque um cenário de utlização do sistema (por exemplo, a construção e uso do segundo circuito de exemplo desta lista).

3. Implementar o sistema em Java. Não é necessário criar uma interface gráfca – o sistema pode funcionar em linha de comando, se preferir.



Foi utilizado o Composite porque um circuito é feito de várias PortaLógicas e além disso, uma Porta Lógica pode depender de outra porta lógica, a partir dessa informação o que acontece é a necessidade de hierarquia.




```
public class And extends PortaLogica {


    public And() {
        super();
    }

    @Override
    public SinalLogico Saida() {
        boolean r =( entradas.get(0).getSinalLogico() &&
entradas.get(1).getSinalLogico());
        saidas.add(0, new SinalLogico(r));
        return saidas.get(0);
    }

}

public class Xor extends PortaLogica{

    public Xor() {
        super();
    }
```

```
    @Override
    public SinalLogico Saida() {
        boolean r=(entradas.get(0).getSinalLogico() ==
entradas.get(1).getSinalLogico());
        saidas.add(0, new SinalLogico(r));
        return saidas.get(0);
    }

}



public abstract class Componente {
    protected final ArrayList<SinalLogico> entradas;
    protected final ArrayList<SinalLogico> saidas;

    public SinalLogico Entrada(int index) {
        return entradas.get(index - 1);
    }

    public SinalLogico Saida(int index) {
        return saidas.get(index - 1);
    }
    public Componente(int qtde_entradas, int qtde_saidas) {
        this.entradas = new ArrayList<>(qtde_entradas);
        this.saidas = new ArrayList<>(qtde_saidas);

        for(int i = 0; i < qtde_entradas; i++)
            this.entradas.add(new SinalLogico(false));

        for(int i = 0; i < qtde_saidas; i++)
            this.saidas.add(new SinalLogico(false));
    }
    public abstract SinalLogico Saida();


}

public class Circuito extends Componente {
    private final ArrayList<PortaLogica> portas;
    private final ArrayList<String> portas_apelido;

    public Circuito(int qtde_entrada, int qtde_saida) {
        super(qtde_entrada, qtde_saida);

        this.portas = new ArrayList<>();
        this.portas_apelido = new ArrayList<>();
    }
```

```
    public void Adicionar(String nome, PortaLogica porta) {
        portas.add(porta);
        portas_apelido.add(nome);
    }

    public Componente Componente(String nome)
    {
        for(int i = 0; i < portas_apelido.size(); i++)
        {
            if(portas_apelido.get(i).equals(nome))
                return portas.get(i);
        }

        return null;
    }

    @Override
    public SinalLogico Saida() {
        throw new UnsupportedOperationException("Not supported yet.");
//To change body of generated methods, choose Tools | Templates.
    }
}

public class Or extends PortaLogica {

    public Or() {
        super();
    }


    @Override
    public SinalLogico Saida() {
        boolean r=entradas.get(0).getSinalLogico() ||
entradas.get(1).getSinalLogico();
        saidas.add(0, new SinalLogico(r));
        return saidas.get(0);
    }

}

public class Inversor extends PortaLogica {

    public Inversor() {
        super();
    }

    @Override
    public SinalLogico Saida(int numero_porta) {
```

```
        boolean r=!entradas.get(0).getSinalLogico();
        saidas.add(numero_porta - 1, new SinalLogico(r));
        return saidas.get(numero_porta - 1);
    }
    public SinalLogico Saida() {
        boolean r=!entradas.get(0).getSinalLogico();
        saidas.add(0, new SinalLogico(r));
        return saidas.get(0);
    }

}

public class Nand extends PortaLogica{

    public Nand() {
        super();
    }

    @Override
    public SinalLogico Saida() {
        boolean r = entradas.get(0).getSinalLogico() &&
entradas.get(1).getSinalLogico();
        r=!r;
        saidas.add(0, new SinalLogico(r));
        return saidas.get(0);
    }

}

public class Nor extends PortaLogica {


    public Nor() {
        super();
    }


    @Override
    public SinalLogico Saida() {
        boolean r=(entradas.get(0).getSinalLogico() ||
entradas.get(1).getSinalLogico());
        r=!r;
        saidas.add(0, new SinalLogico(r));
        return saidas.get(0);
    }
}

public abstract class PortaLogica extends Componente{
```

```
    public PortaLogica() {
        super(2, 1);
    }

    @Override
    public abstract SinalLogico Saida(int numero_porta);



}

public class Run {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Circuito circuito = new Circuito(3, 1);
        circuito = new Circuito(3,1); //3 entradas e 1 saída
        //elemento 1
        circuito.Adicionar("INV1", new Inversor());
        //elemento 2
        circuito.Adicionar("INV2", new Inversor());
        //elemento 3
        circuito.Adicionar("AND1", new And());
        //elemento 4
        circuito.Adicionar("AND2", new And());
        //elemento 5
        circuito.Adicionar("NOR1", new Nor());
        //níveis das entradas
        circuito.Entrada(1).setSinalLogico(false);
        circuito.Entrada(2).setSinalLogico(true);
        circuito.Entrada(3).setSinalLogico(false);
        //conexões

circuito.Componente("INV1").Entrada(1).setSinalLogico(circuito.Entrada(1)
.getSinalLogico());

circuito.Componente("AND1").Entrada(2).setSinalLogico(circuito.Entrada(2)
.getSinalLogico());

circuito.Componente("INV2").Entrada(1).setSinalLogico(circuito.Entrada(2)
.getSinalLogico());

circuito.Componente("AND2").Entrada(2).setSinalLogico(circuito.Entrada(3)
.getSinalLogico());
```

```
circuito.Componente("AND1").Entrada(1).setSinalLogico(circuito.Componente
("INV1").Saida().getSinalLogico());

circuito.Componente("AND2").Entrada(1).setSinalLogico(circuito.Componente
("INV2").Saida().getSinalLogico());

circuito.Componente("NOR1").Entrada(1).setSinalLogico(circuito.Componente
("AND1").Saida().getSinalLogico());

circuito.Componente("NOR1").Entrada(2).setSinalLogico(circuito.Componente
("AND2").Saida().getSinalLogico());

        //saída – a função recalcula o circuito
        boolean
valor=circuito.Componente("NOR1").Saida().getSinalLogico(); //=
circuito.Saida(1).getSinalLogico();
        System.out.println("Resultado do circuito pré-programado: "+
valor);

    }
}

public class Xnor extends PortaLogica{


    public Xnor() {
        super();
    }

    @Override
    public SinalLogico Saida() {
        boolean r=entradas.get(0).getSinalLogico() ^
entradas.get(1).getSinalLogico();
        r=!r;
        saidas.add(0, new SinalLogico(r));
        return saidas.get(0);
    }

}
```