Instituto Federal Catarinense (Campus Blumenau)
Professor: Ricardo de La Rocha Ladeira
Matéria: Padrões de Projeto
Nomes: Gabrielli Danker
Turma: BCC 2025.1
Data de entrega: 03 de Abril de 2025

Exercícios

1) Implemente, na linguagem de programação de sua preferência, o problema do restaurante, disponível em GOMES (s.d.).

   RESPOSTA: Atividade também feita no link a seguir: https://www.jdoodle.com/online-java-compiler

   Segue código:

```java
interface Pedido {
    void executar();
}


class PedidoHamburguer implements Pedido {
    private Cozinheiro cozinheiro;

    public PedidoHamburguer(Cozinheiro cozinheiro) {
        this.cozinheiro = cozinheiro;
    }

    @Override
    public void executar() {
        cozinheiro.fazerHamburguer();
    }
}


class PedidoMilkshake implements Pedido {
    private Cozinheiro cozinheiro;

    public PedidoMilkshake(Cozinheiro cozinheiro) {
        this.cozinheiro = cozinheiro;
    }

    @Override
    public void executar() {
        cozinheiro.fazerMilkshake();
    }
}


class Cozinheiro {
    public void fazerHamburguer() {
        System.out.println("Cozinheiro está preparando um hambúrguer...");
    }

```

```
38
39      public void fazerMilkshake() {
40          System.out.println("Cozinheiro está preparando um milkshake...");
41      }
42  }
43
44
45  class Garconete {
46      private Pedido pedido;
47
48      public void setPedido(Pedido pedido) {
49          this.pedido = pedido;
50      }
51
52      public void processarPedido() {
53          if (pedido != null) {
54              pedido.executar();
55          } else {
56              System.out.println("Nenhum pedido foi feito.");
57          }
58      }
59  }
60
61  public class Cliente {
62      public static void main(String[] args) {
63
64          Cozinheiro cozinheiro = new Cozinheiro();
65          Garconete garconete = new Garconete();
66
67          Pedido pedidoHamburguer = new PedidoHamburguer(cozinheiro);
68          Pedido pedidoMilkshake = new PedidoMilkshake(cozinheiro);
69
70          System.out.println("Cliente: Quero um hambúrguer!");
71          garconete.setPedido(pedidoHamburguer);
72          garconete.processarPedido();
73
74          System.out.println("Cliente: Quero um milkshake!");
75          garconete.setPedido(pedidoMilkshake);
76          garconete.processarPedido();
77      }
78  }
79
```

O código está na pasta Command -> Cliente.java

Segue o retorno:

```
Cliente: Quero um hambúrguer!
Cozinheiro está preparando um hambúrguer...
Cliente: Quero um milkshake!
Cozinheiro está preparando um milkshake...
```

2) Implemente um sistema de fila de impressão que utilize os padrões ***Command*** e ***Singleton***. O sistema deve permitir que documentos sejam adicionados à fila de impressão e processados sequencialmente. Além disso, deve ser possível cancelar a impressão do último documento enviado antes da conclusão da fila. A fila de impressão deve ser gerenciada por uma classe *singleton*, garantindo que apenas uma instância da fila exista durante a execução do programa. O padrão *Command* deve ser utilizado para encapsular cada solicitação de impressão, permitindo a execução e o cancelamento de impressões.

RESPOSTA:

Segue o código:

```java
import java.util.LinkedList;
import java.util.Queue;

interface ComandoImpressao {
    void executar();
    void cancelar();
}

class ImprimirDocumento implements ComandoImpressao {
    private Impressora impressora;
    private String documento;

    public ImprimirDocumento(Impressora impressora, String documento) {
        this.impressora = impressora;
        this.documento = documento;
    }

    @Override
    public void executar() {
        impressora.imprimir(documento);
    }

    @Override
    public void cancelar() {
        impressora.cancelarImpressao(documento);
    }

    public String getDocumento() {
        return documento;
    }
}

class Impressora {
    public void imprimir(String documento) {
        System.out.println("Imprimindo documento: " + documento);
    }

    public void cancelarImpressao(String documento) {
        System.out.println("Impressão cancelada: " + documento);
    }
}

class FilaImpressao {
    private static FilaImpressao instancia;
    private Queue<ComandoImpressao> fila = new LinkedList<>();
    private ComandoImpressao ultimoComando; // Para controle de cancelamento

    private FilaImpressao() {}

    public static synchronized FilaImpressao getInstancia() {
        if (instancia == null) {
            instancia = new FilaImpressao();
        }
        return instancia;
    }

    public void adicionarDocumento(ComandoImpressao comando) {
        fila.add(comando);
        ultimoComando = comando;
        System.out.println("Documento adicionado à fila: " + ((ImprimirDocumento) comando).getDocumento());
    }

    public void processarFila() {
        while (!fila.isEmpty()) {
            ComandoImpressao comando = fila.poll();
            comando.executar();
        }
    }
```

```java
    public void cancelarUltimoDocumento() {
        if (ultimoComando != null) {
            ultimoComando.cancelar();
            fila.remove(ultimoComando);
            ultimoComando = null;
        } else {
            System.out.println("Nenhum documento para cancelar.");
        }
    }
}

public class Cliente {
    public static void main(String[] args) {
        Impressora impressora = new Impressora();
        FilaImpressao fila = FilaImpressao.getInstancia();

        ComandoImpressao doc1 = new ImprimirDocumento(impressora, "Relatório.pdf");
        ComandoImpressao doc2 = new ImprimirDocumento(impressora, "Contrato.docx");
        ComandoImpressao doc3 = new ImprimirDocumento(impressora, "Imagem.png");

        fila.adicionarDocumento(doc1);
        fila.adicionarDocumento(doc2);
        fila.adicionarDocumento(doc3);

        fila.cancelarUltimoDocumento();

        fila.processarFila();
    }
}
```

O código esta na pasta Command -> Cliente2.java

Segue o que retorna: