INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO

CAMPUS GUARULHOS

BEATRIZ MAZZUCATTO - GU3046745

LANCHONETE QUASE TRÊS LANCHES

GUARULHOS - SP
2025

BEATRIZ MAZZUCATTO - GU3046745

LANCHONETE QUASE TRÊS LANCHES

Atividade apresentada à disciplina de API e Microsserviços, do Instituto Federal de Educação, Ciência e Tecnologia Campus Guarulhos como nota parcial para aprovação na disciplina do curso Engenharia de Computação, sexto semestre.

Professor Mestre: Giovani Fonseca Ravagnani Disperati

GUARULHOS - SP
2025

1. Objetivo

Este documento descreve a arquitetura, os padrões de projeto e as justificativas das decisões de design adotadas no **Sistema de Lanchonete**, desenvolvido em Java. O sistema simula o funcionamento básico de um estabelecimento de alimentação, permitindo o **cadastro de itens alimentícios**, **criação de pedidos** e **geração de notas fiscais**.

2. Arquitetura e Padrões de Projeto

2.1 Padrão de Herança (*Inheritance*)

Decisão de Design:
Implementação de uma hierarquia de classes com Item como classe abstrata base.

Justificativa:

- **Reutilização de código:** Atributos comuns (preco, dataValidade, peso) defi- nidos uma única vez.
- **Polimorfismo:** Permite tratar diferentes tipos de itens de forma uniforme.
- **Extensibilidade:** Facilita a adição de novos tipos de produtos sem alterar código existente.

Implementação:

```
abstract class Item
  { protected double
  preco;
  protected Date dataValidade;
  protected double peso;

  public abstract String descricao();
```

```
}
```

2.2 Padrão de Composição

Decisão de Design:
A classe `Pedido` contém uma lista de objetos `Item`.

Justificativa:

- **Flexibilidade:** Um pedido pode conter qualquer quantidade e tipo de itens.
- **Baixo acoplamento:** `Pedido` não depende da implementação interna de ca- da item.
- **Responsabilidade única:** Cada classe mantém sua função específica.

3. Estrutura das Classes

3.1 Classe `Item` (Abstrata)

Responsabilidade:
Definir a estrutura base para todos os produtos alimentícios.

Características:

- Atributos protegidos para acesso controlado por subclasses.
- Método abstrato descricao() obriga a implementação nas classes concretas.
- Encapsulamento garantido por métodos *getters* públicos.

3.2 Classes Filhas (`Pizza`, `Lanche`, `Salgadinho`)

Responsabilidade:
Definir características específicas de cada tipo de alimento.

Características:

- Atributos adicionais (ex.: `recheio`, `borda` para `Pizza`).
- Implementação concreta do método `descricao()`.
- Construtores que invocam o construtor da superclasse.

3.3 Classe `Pedido`

Responsabilidade:
Gerenciar a coleção de itens e executar cálculos de totais.

Características:

- Agregação de objetos `Item` por meio de `List<Item>`.
- Cálculos de **subtotal**, **taxa de serviço** e **total**.
- Geração de **nota fiscal** formatada.
- Função de **cálculo de troco**.

4. Funcionamento do Sistema

1. **Criação de Itens**

Pizza pizza = new Pizza(45.90, validade, 0.8, "Calabresa", "Recheada", "Tomate");

Lanche lanche = new Lanche(22.50, validade, 0.4, "Australiano", "Frango", "Mostarda");

2. **Criação do Pedido**

Pedido pedido = new Pedido("João Silva", 0.10); // Cliente + taxa de 10%

3. **Adição de Itens ao**

Pedido

pedido.adicionarItem(pizza);

pedido.adicionarItem(lanche);

4. **Processamento e Geração de Nota**
    - Cálculo do subtotal (soma dos preços dos itens).
    - Aplicação da taxa de serviço.
    - Cálculo do total (subtotal + taxa).
    - Formatação da nota fiscal.
5. **Cálculo de Troco**

```
double troco = pedido.calcularTroco(valorRecebido);
```

5. Vantagens do Design

5.1 Manutenibilidade

- Separação clara de responsabilidades.
- Alterações em um tipo de item não afetam outros.

5.2 Extensibilidade

- Inclusão de novos produtos (ex.: `Bebida`, `Sobremesa`) sem alterações estru- turais.
- Novos cálculos podem ser implementados na classe `Pedido`.

5.3 Reutilização

- Código comum centralizado na classe `Item`.
- Métodos de cálculo aplicáveis a diferentes cenários.

5.4 Polimorfismo

- Lista de `Item` pode armazenar qualquer subclasse.
- Invocação uniforme do método `descricao()`.

6. Fluxo de Execução

1. **Inicialização:** Instanciação de objetos de produtos.
2. **Agregação:** Inclusão dos produtos em um `Pedido`.
3. **Processamento:** Cálculos automáticos de valores e taxas.
4. **Saída:** Geração da nota fiscal e cálculo de troco.

7. Considerações de Design

7.1 Encapsulamento

- Atributos privados/protegidos com acesso via *getters*.

7.2 Abstração

- Interface uniforme via classe abstrata `Item`.
- Detalhes de implementação restritos às classes concretas.

7.3 Herança

- Reaproveitamento de atributos e métodos comuns.
- Especialização nas subclasses.

8. Conclusão

O design proposto garante **flexibilidade**, **clareza** e **facilidade de manutenção**. A arquitetura suporta novas funcionalidades com mínimo impacto no código existente, assegurando a escalabilidade do sistema.