

# **Sistema de Controle de Pedidos de um Restaurante**

## **Etapa 2: Refatoração**

**Autores:** Fátima Pereira Santos Pinho  
Ismael Rodrigues de Oliveira Neto  
Rebeca Helen Batista Amorim

# Agenda

Apresentar a evolução do projeto da Equipe Marselha, desenvolvido em Kotlin, focando na aplicação de boas práticas de arquitetura de software.

## **1. Introdução:**

Explicar a reestruturação do código e os benefícios esperados.

## **2. Estrutura do Projeto:**

Apresentar a nova organização do sistema.

## **3. Principais Operações do Sistema:**

Demonstrar funcionalidades-chave e como elas interagem na nova arquitetura.

## **4. Fluxo de uma Funcionalidade:**

Ilustrar como a comunicação entre a interface e a lógica de negócio foi implementada.

## **5. Considerações Finais:**

Destacar os resultados alcançados.

# Introdução

Nosso objetivo foi **refatorar** o código.



Aplicamos princípios de arquitetura de software para garantir a qualidade e a manutenibilidade do projeto.

# Estrutura do Projeto



Sistema de Controle de Pedidos de um Restaurante

Etapa 2: Refatoração

# Estrutura do Projeto



- **Model.kt: A Lógica do Negócio**
  1. Estrutura dos Dados com data class
  2. Consistência com enum class
  3. Funções Puras

# Estrutura do Projeto



- **Estrutura de Dados no Model.kt**

enum class para Status:

```
enum class StatusPedido {  
    ACEITO,  
    FAZENDO,  
    FEITO,  
    ESPERANDO_ENTREGADOR,  
    SAIU_PARA_ENTREGA,  
    ENTREGUE  
}
```

data class atualizada:

```
data class Pedido(  
    val codigo: Int,  
    val itens: MutableList<Item>,  
    var total: Double,  
    var cupom: Boolean,  
    var status: StatusPedido =  
        StatusPedido.ACEITO  
)
```

# Estrutura do Projeto



- **Main.kt: O Ponto de Entrada**

1. Função `main()` e o loop de menu `do-while`.
2. Captura os dados do usuário (`readln`).
3. Chama as funções do `Model.kt` para executar as ações.
4. Exibe os resultados para o usuário (`println`).

# Estrutura do Projeto



- **Maint.kt: Responsável pela "conversa" com o usuário**

As funções têm o prefixo **ui** (User Interface) para deixar claro seu propósito:

```
fun uiCadastrarItem()
```

```
fun uiAtualizarItem()
```

```
fun uiCriarPedido()
```

```
fun uiAtualizarPedido()
```

```
fun uiConsultarPedidos()
```



# Principais Operações do Sistema **unex**

- **Interação entre as Camadas:** a Interface (Main) delega as tarefas para a Lógica (Model).

## Consultar Pedido:

Passo 1

**Main.kt** pergunta como o usuário deseja filtrar os pedidos.

Passo 2

**Model.kt** executa a busca, filtra os resultados e os devolve para a interface.

Passo 3

**Main.kt** exibe os dados formatados para o usuário.

Sistema de Controle de Pedidos de um Restaurante

Etapa 2: Refatoração

# Fluxo de uma Funcionalidade



- **Exemplo Prático:** Cadastrando um Item



Passo 1: A Interface solicita os dados (Main.kt)

```
fun uiCadastrarItem() {  
    println("\n-> CADASTRAR ITEM\n")  
    print("Insira o nome do item: ")  
    ...  
    cadastrarItem(nome, descricao,  
    preco, estoque)
```



Passo 2: O Model executa a ação (Model.kt)

```
fun cadastrarItem(nome: String..) {  
    val novoItem = Item  
        codigo = contadorItem++,  
    ...  
    itensCadastrados.add(novoItem)
```

# Considerações Finais



A separação clara entre lógica e interface torna o projeto muito mais fácil de entender e dar manutenção.

# Referências



- KOTLIN. Kotlin documentation. Disponível em: <https://kotlinlang.org/docs/home.html>. Acesso em: 03 out. 2025.