Centro Universitário de Excelência
Sistemas de Informação

Modularização do Sistema de Gerenciamento de Pedidos de Restaurante

Autores: Antonio Gabriel Gomes Falcão do Nascimento
Angel Yraê Ribeiro Ferreira
Alexandre Roque Pereira Costa

Feira de Santana, 2025

Agenda

1. main.kt:
arquivo que contém toda a lógica do programa

2. model.kt:
arquivo que contém as funções que lidam com os dados

Essa apresentação visa explicar o funcionamento do nosso código que, foi criado para resolver o problema apresentado pelo professor Lucas.

Main.kt

- Contém toda a lógica do funcionamento do programa

- Imports do Model

Funções sendo usadas

```
atualizarItem(userIndexItem, produtoAlterado, novoValor, userIndexProduto)

cadastrarItem(nome, descricao, preco, quantidade)

valorTotal = adicionarProdutoAoPedido(produtoEscolhido, itensPedido, valorTotal, qtd)

val pedido = criarPedido(contadorPedidos, itensPedido, valorTotal)

atualizarPedido(pedidoSelecionado, novoStatus)

consultarPedidosPorStatus(StatusPedido.FEITO).forEach { println(it) }
```

Model.kt

- Contém todas as funções
- Contém Variáveis
- Contém as datas classes e enum classes

Funções

```
fun cadastrarItem (nome: String, descricao: String, preco: Double, quantidade: Int) {  1 Usage  copperlamb78
    codigoProduto += 1
    val produto = Produto(nome, descricao,  valor = preco, quantidade) //adiciona O produto a lista de produtos
    produtos.add(produto)
}
```

```
fun atualizarItem (userIndexItem: Int, produtoAlterado: Produto, novoValor: String, userIndexProduto: Int) {  1 Usage
    when (userIndexItem) { // Aqui alteramos o valor da característica do produto com seus respectivos tipos
        1 -> {
            produtoAlterado.nome = novoValor
        }

        2 -> {
            produtoAlterado.descricao = novoValor
        }

        3 -> {
            produtoAlterado.valor = novoValor.toDouble()
        }

        4 -> {
            produtoAlterado.quantidade = novoValor.toInt()
        }
    }
    produtos[userIndexProduto] = produtoAlterado //trocamos o produto antigo pelo o produto alterado
}
```

Funções

```
fun adicionarProdutoAoPedido(produtoEscolhido: Produto, itensPedido: MutableList<Produto>, valorTotal: Double, qtd: Int): Double {
    val produtoPedido = produtoEscolhido.copy(quantidade = qtd)
    itensPedido.add(produtoPedido)
    produtoEscolhido.quantidade -= qtd
    return valorTotal + (produtoPedido.valor * qtd)
}
```

```
fun criarPedido (contadorPedidos: Int, itensPedido: MutableList<Produto>, valorTotal: Double) : Pedido {
    val pedido = Pedido( id = contadorPedidos, itensPedido, valorTotal)
    pedidos.add(pedido)
    return pedido
}
```

Funções

```
fun atualizarPedido (pedidoSelecionado: Pedido, novoStatus: Int) {  1 Usage
        pedidoSelecionado.status = StatusPedido.entries[novoStatus - 1]
}
```

```
fun consultarPedidosPorStatus(status: StatusPedido): List<Pedido> =
    pedidos.filter { it.status == status }
```

Variáveis

```
var codigoProduto = 0  3 Usages

val produtos = mutableListOf<Produto>() //Lista de produtos  10 Usages
val pedidos = mutableListOf<Pedido>() // Lista de pedidos  7 Usages
```

Classes

```
data class Produto( //Classe do produt
    var nome: String,
    var descricao: String,
    var valor: Double,
    var quantidade: Int,
    val codigo: Int = codigoProduto
)
```

```
data class Pedido( // Classe para pedidos  5 Usages
    val id: Int,
    val itens: MutableList<Produto>,
    var valorTotal: Double,
    var status: StatusPedido = StatusPedido.ACEITO
)
```

```
enum class StatusPedido { // Clase para status do pedido  12 Usages  new *
    ACEITO, FAZENDO, FEITO, ESPERANDO_ENTREGADOR, SAIU_PARA_ENTREGA, ENTREGUE
}
```

Referências

- https://kotlinlang.org/docs/home.html