# Correção das questões da Prova de Engenhara de software

## Hiago Pichi Radaeli

Questão 1

A resposta correta é a opção E (III e IV)

Questão 2

A resposta correta é a opção E (Requisito Não Funcional);

. Na descrição do problema, é mencionada a necessidade de verificar a identidade do cliente por meio da utilização de dados biométricos. Nesse contexto, a utilização de dados biométricos é um recurso que define como o sistema deve funcionar e não está diretamente relacionado a regras de negócios específicas, padrões de arquitetura ou requisitos funcionais.

Os requisitos não funcionais descrevem features do sistema que não estão diretamente relacionados às funcionalidades específicas, mas sim a propriedades como desempenho, segurança, usabilidade, confiabilidade, entre outras. Portanto, a opção E (Requisito Não Funcional) é a resposta correta.

Questão 3

A resposta correta é a opção D (Conceitos, associações e atributos);

A análise do problema em objetos individuais é um passo essencial na análise orientada a objetos. Um modelo conceitual representa esses conceitos em um domínio do problema. No contexto da orientação a objetos, um objeto é composto por seus atributos (características que descrevem o objeto), associações (relacionamentos entre objetos) e conceitos (entidades ou elementos do domínio do problema).

Portanto, a opção d) (Conceitos, associações e atributos) é a resposta mais precisa, pois engloba os elementos fundamentais de um modelo conceitual na orientação a objetos.

Questão 4

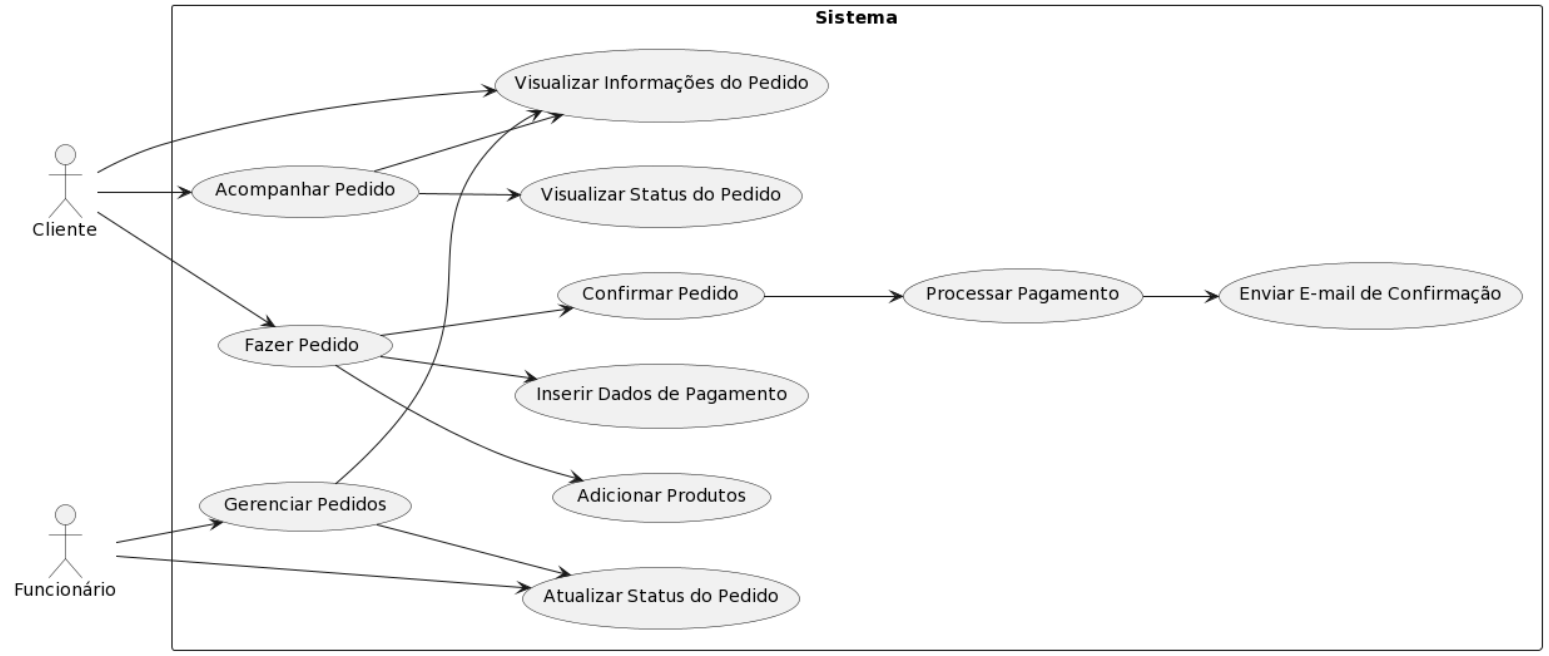
Cliente faz pedido online acessando o sistema e selecionando os produtos que deseja comprar. O cliente adiciona os produtos ao carrinho e verifica o carrinho. O cliente confirma o pedido e insere os dados de pagamento. A loja processa o pagamento, confirma o pedido e envia um e-mail de confirmação para o cliente. O cliente acompanha o status do pedido. O Funcionário da loja gerencia OS pedidos e atualiza O status do pedido. O cliente acessa O sistema e executa a opção "Acompanhar. Pedido. Loja exibe o status atual do pedido (processando, em trânsito, entregue). A loja permite que o cliente visualize as informações do pedido (produtos, data de entrega, etc.). O funcionário da loja gerencia os pedidos executando a opção "Gerenciar Pedidos". A loja exibe uma lista de todos os pedidos. O funcionário seleciona um pedido e pode visualizar as informações do pedido (produtos. cliente, etc.). 0 funcionário pode atualizar o status do pedido (processando, em trânsito, entregue). O Funcionário da loja atualiza ○ status do pedido. O funcionário executa a opção "Atualizar Status do Pedido". A loja exibe uma lista

de todos os pedidos. O funcionário seleciona um pedido e atualiza o status do pedido (processando, em trânsito, entregue).

Caso de uso em PlantUML

```
@startuml

left to right direction

actor Cliente as cliente
actor Funcionário as funcionario

rectangle Sistema {
  cliente --> (Fazer Pedido)
  cliente --> (Acompanhar Pedido)
  cliente --> (Visualizar Informações do Pedido)
  funcionario --> (Gerenciar Pedidos)
  funcionario --> (Atualizar Status do Pedido)

  (Fazer Pedido) --> (Adicionar Produtos)
  (Fazer Pedido) --> (Inserir Dados de Pagamento)
  (Fazer Pedido) --> (Confirmar Pedido)

  (Confirmar Pedido) --> (Processar Pagamento)
  (Processar Pagamento) --> (Enviar E-mail de Confirmação)

  (Acompanhar Pedido) --> (Visualizar Status do Pedido)
  (Acompanhar Pedido) --> (Visualizar Informações do Pedido)

  (Gerenciar Pedidos) --> (Visualizar Informações do Pedido)
  (Gerenciar Pedidos) --> (Atualizar Status do Pedido)
}
```

@enduml




Modelo de Domínio em PlantUML

@startuml

class Cliente {

- nome: String

- email: String

- endereco: String

+ fazerPedido()

+ adicionarProduto(Produto)

+ removerProduto(Produto)

+ verificarCarrinho()

+ listarProdutosCarrinho()

+ confirmarPedido()

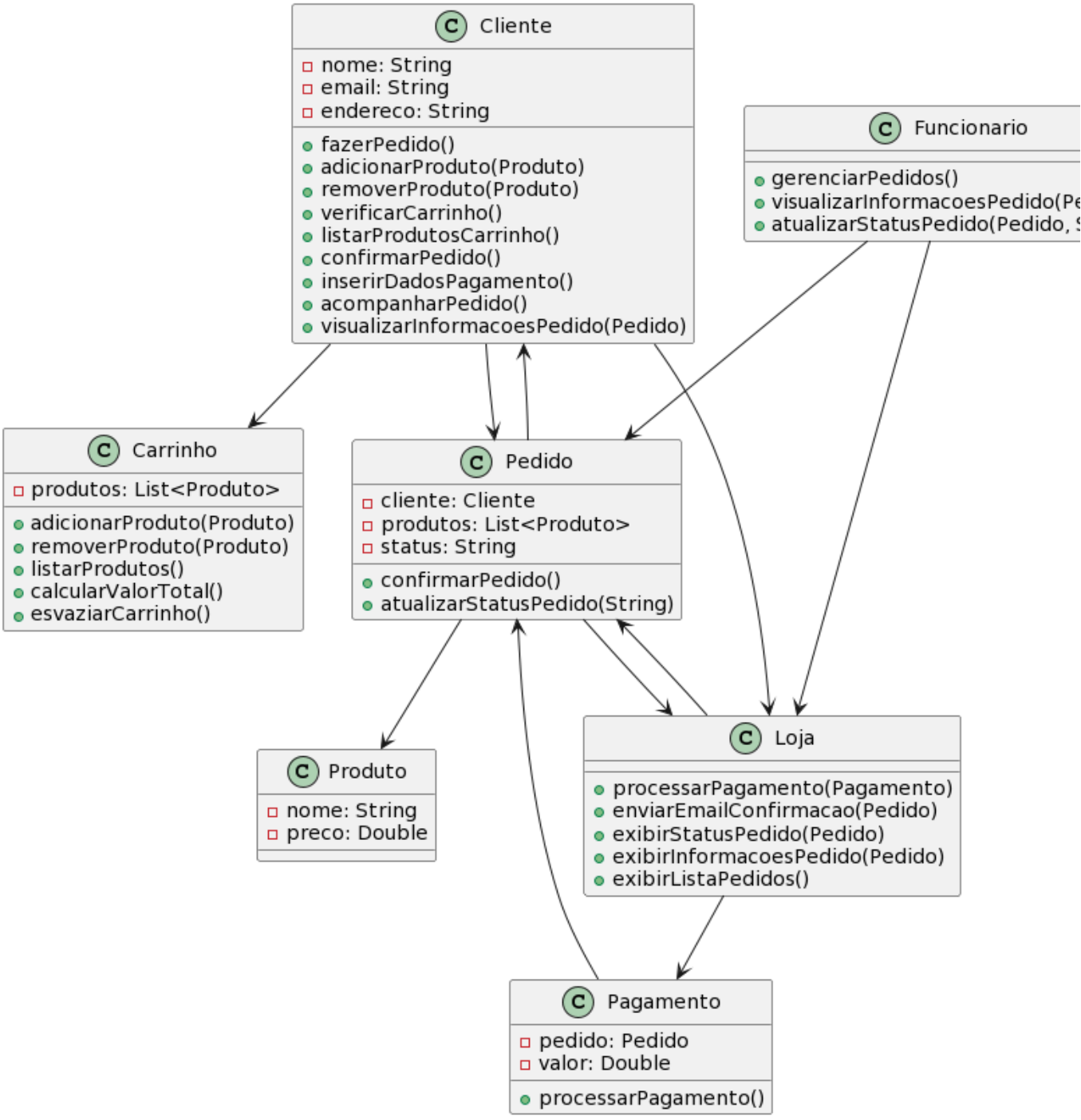
+ inserirDadosPagamento()

+ acompanharPedido()

+ visualizarInformacoesPedido(Pedido)

```
}

class Produto {
  - nome: String
  - preco: Double
}

class Carrinho {
  - produtos: List<Produto>
  + adicionarProduto(Produto)
  + removerProduto(Produto)
  + listarProdutos()
  + calcularValorTotal()
  + esvaziarCarrinho()
}

class Pedido {
  - cliente: Cliente
  - produtos: List<Produto>
  - status: String
  + confirmarPedido()
  + atualizarStatusPedido(String)
}

class Pagamento {
  - pedido: Pedido
  - valor: Double
  + processarPagamento()
}
```

```
class Loja {
  + processarPagamento(Pagamento)
  + enviarEmailConfirmacao(Pedido)
  + exibirStatusPedido(Pedido)
  + exibirInformacoesPedido(Pedido)
  + exibirListaPedidos()
}

class Funcionario {
  + gerenciarPedidos()
  + visualizarInformacoesPedido(Pedido)
  + atualizarStatusPedido(Pedido, String)
}

Cliente --> Carrinho
Cliente --> Pedido
Cliente --> Loja

Pedido --> Cliente
Pedido --> Produto
Pedido --> Loja

Pagamento --> Pedido

Loja --> Pagamento
Loja --> Pedido

Funcionario --> Pedido
Funcionario --> Loja

@enduml
```


Cliente
nome: String
email: String
endereco: String
fazerPedido()
adicionarProduto(Produto)
removerProduto(Produto)
verificarCarrinho()
listarProdutosCarrinho()
confirmarPedido()
inserirDadosPagamento()
acompanharPedido()
visualizarInformacoesPedido(Pedido)
Funcionario
gerenciarPedidos()
visualizarInformacoesPedido(Pe
atualizarStatusPedido(Pedido, S
Carrinho
produtos: List<Produto>
adicionarProduto(Produto)
removerProduto(Produto)
listarProdutos()
calcularValorTotal()
esvaziarCarrinho()
Pedido
cliente: Cliente
produtos: List<Produto>
status: String
confirmarPedido()
atualizarStatusPedido(String)
Produto
nome: String
preco: Double
Loja
processarPagamento(Pagamento)
enviarEmailConfirmacao(Pedido)
exibirStatusPedido(Pedido)
exibirInformacoesPedido(Pedido)
exibirListaPedidos()
Pagamento
pedido: Pedido
valor: Double
processarPagamento()