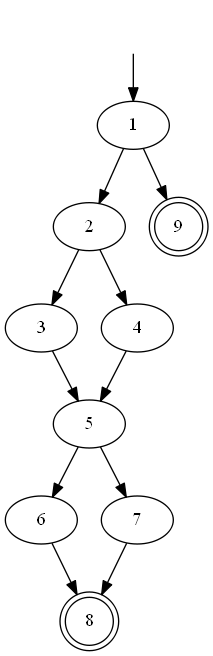
**Testes Unitários**

***Obs:*** *Em alguns testes foram utilizadas as técnicas de* ***particionamento do espaço de entradas****, enquanto que em outros foi utilizada a técnica de* ***cobertura por caminhos primos****. Em ambos os casos, a cobertura dos testes dos métodos testados chegou a 100%. Testes construídos sem a utilização de nenhuma técnica foram marcados como “Outros Testes”. Não foram implementados testes para todas as classes desenvolvidas. Os problemas de codificação que geraram alguns testes “infeasibles” foram mantidos.*

**Classe Pedido**

**Método addItem(Item item)**



**Testes requeridos segundo critério de caminhos primos:**

**TR = {[1,9], [1,2,3,5,7,8], [1,2,3,5,6,8], [1,2,4,5,6,8], [1,2,4,5,7,8]}**

**T1 = [1,9]:**

* Adição de um item válido a um pedido, porém sem estoque suficiente, ou;
* Adição de um item inválido a um pedido;

**T2 = [1,2,3,5,7,8]:**

* Adição de um item válido a um pedido, com estoque suficiente, e já presente no pedido (item é bebida pronta e pedido não possui itens pendentes de preparo na cozinha);

**T3 = [1,2,3,5,6,8]:**

* Adição de um item válido a um pedido, com estoque suficiente, e já presente no pedido (item não é bebida pronta ou pedido possui itens pendentes de preparo na cozinha);

**T4 = [1,2,4,5,6,8]:**

* Adição de um item válido a um pedido, com estoque suficiente, e ainda não presente no pedido (item é bebida pronta e pedido não possui itens pendentes de preparo na cozinha);

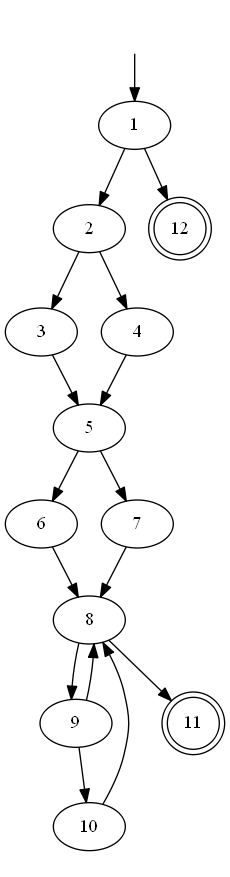
**T5 = [1,2,4,5,7,8]**

* Adição de um item válido a um pedido, com estoque suficiente, e ainda não presente no pedido (item não é bebida pronta ou pedido possui itens pendentes de preparo na cozinha);

**Erros encontrados:**

1-O estado do pedido não estava sendo atualizado quando se inseria um item novo no pedido devido a um erro no método public boolean addItem(Item item).

**Método removeItem(Item item)**



**Testes requeridos segundo critério de caminhos primos:**

**TR = {[1,2,4,5,7,8,9,10], [8,9,10,8], [9,10,8,9], [9,8,11], [9,8,9], [8,9,8], [1,12], [1,2,4,5,7,8,9,10], [8,9,10,8], [10,8,9,10], [9,10,8,11], [9,10,8,9], [1,2,4,5,7,8,9,10], [8,9,10,8], [9,10,8,11], [1,2,3,5,7,8,11], [1,2,4,5,6,8,11], [1,2,3,5,6,8,11], [1,2,4,5,7,8,11], [1,2,3,5,7,8,9,10], [8,9,10,8], [9,10,8,11], [1,2,3,5,6,8,9,10], [8,9,10,8], [9,10,8,11], [1,2,4,5,6,8,9,10], [8,9,10,8], [9,10,8,11]}**

**T1 = [1,2,4,5,7,8,9,10,8,9,8,9,8,11]:**

* Remoção de um item válido presente no pedido, cuja quantidade no pedido seja igual a 1 unidade, e que após a remoção o pedido não fique vazio. Após a remoção do item o pedido deve ter um item que necessite de preparo na cozinha e dois itens que não necessitem preparo;

**T2 = [1,12]:**

* Remoção de um item válido, porém não presente no pedido, ou;
* Remoção de um item inválido de um pedido;

**T3 = [1,2,4,5,7,8,9,10,8,9,10,8,11]:**

* Remoção de um item válido presente no pedido, cuja quantidade no pedido seja igual a 1 unidade, e que após a remoção o pedido não fique vazio. Após a remoção do item o pedido deve ter dois itens que necessitem de preparo na cozinha e nenhum que não necessite preparo;

**T4 = [1,2,4,5,7,8,9,10,8,11]:**

* Remoção de um item válido presente no pedido, cuja quantidade no pedido seja igual a 1 unidade, e que após a remoção o pedido não fique vazio. Após a remoção do item o pedido deve ter um item que necessite de preparo na cozinha e nenhum que não necessite preparo;

**T5 = [1,2,3,5,7,8,11]:**

* Remoção de um item válido presente no pedido, cuja quantidade no pedido seja maior que 1 unidade, e que após a remoção o pedido não fique vazio. Após a remoção o pedido não deve conter nenhum item;

Este teste é “infeasible”, pois não é possível que o pedido fique não vazio e vazio ao mesmo tempo. Para este caso foi implementado o caminho **[1,2,3,5,7,8,9,11]**, que considera que após a remoção do item o pedido ainda contenha 1 item que não necessite de preparo na cozinha.

**T6 = [1,2,4,5,6,8,11]:**

* Remoção de um item válido presente no pedido, cuja quantidade no pedido seja igual a 1 unidade, e que após a remoção o pedido fique vazio. Após a remoção o pedido não deve conter nenhum item;

**T7 = [1,2,3,5,6,8,11]:**

* Remoção de um item válido presente no pedido, cuja quantidade no pedido seja maior do que 1 unidade, e que após a remoção o pedido fique vazio. Após a remoção o pedido não deve conter nenhum item que necessite de preparo na cozinha;

Este teste é “infeasible”, pois não é possível remover um item cuja quantidade seja maior que 1 unidade e que após a remoção de 1 unidade o pedido fique vazio.

**T8 = [1,2,4,5,7,8,11]:**

* Remoção de um item válido presente no pedido, cuja quantidade no pedido seja igual a 1 unidade, e que após a remoção o pedido não fique vazio. Após a remoção o pedido não deve conter nenhum item;

Este teste é “infeasible”, pois não é possível que o pedido fique não vazio e vazio ao mesmo tempo. Para este caso foi implementado o caminho **[1,2,4,5,7,8,9,11]**, que considera que após a remoção do item o pedido ainda contenha 1 item que não necessite de preparo na cozinha.

**T9 = [1,2,3,5,7,8,9,10,8,11]:**

* Remoção de um item válido presente no pedido, cuja quantidade no pedido seja maior que 1 unidade, e que após a remoção o pedido não fique vazio. Após a remoção o pedido deve conter 1 item que necessite de preparo na cozinha;

**T10 = [1,2,3,5,6,8,9,10,8,11]:**

* Remoção de um item válido presente no pedido, cuja quantidade no pedido seja maior do que 1 unidade, e que após a remoção o pedido fique vazio. Após a remoção, o pedido ainda deve conter um item que necessite de preparo na cozinha;

Este teste é “infeasible”, pois não é possível remover um item cuja quantidade seja maior que 1 unidade e que após a remoção de 1 unidade o pedido fique vazio.

**T11 = [1,2,4,5,6,8,9,10,8,11]:**

* Remoção de um item válido presente no pedido, cuja quantidade no pedido seja igual a 1 unidade, e que após a remoção o pedido fique vazio. Após a remoção o pedido deve conter um item que necessite de preparo na cozinha;

Este teste é “infeasible”, pois não é possível que o pedido fique não vazio e vazio ao mesmo tempo.

**Erros encontrados:**

1- Ao se remover um item do pedido, o estoque não estava sendo atualizado corretamente devido a um erro no método public boolean removeItem(Item item).

**Outros testes**

Teste testDefaultValues()

Testes cobertos:

* Verificação das condições iniciais dos atributos do Pedido.

**Erros encontrados:**

1. Ocorria uma NullPointerException caso fosse solicitado um código de item inválido ou não constante no pedido devido a um erro no método *public int getQuantidadeItem(int codigoItem)*. O método foi corrigido para retornar zero nestes casos.

Teste testGettersAndSetters()

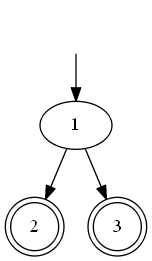
Testes cobertos:

* Verificação se os atributos da classe Reserva são setados e retornados corretamente.

**Nenhum erro foi encontrado neste teste.**

**Classe ReceitaTest**

**Método addIngrediente(String ingrediente, double quantidade)**



**Testes requeridos segundo critério de caminhos primos:**

**TR = {[1,2], [1,3]}**

**T1 = [1,2]:**

* Inserção de um ingrediente que já existe na receita.

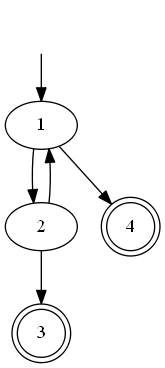
**T2 = [1,3]:**

* Inserção de um ingrediente que ainda não existe na receita.

**Erros Encontrados:**

1-Se o ingrediente já existia na receita, adicionar uma nova quantidade sobrescrevia a quantidade existente ao invés de somá-la, devido a um erro no método *addIngrediente(String descricao, double quantidade)*. O método então foi alterado para verificar se o ingrediente já existe antes de adicionar, somando as quantidades caso positivo.

**Método isEstoqueSuficiente()**



**Testes requeridos segundo critério de caminhos primos:**

**TR = {[1,2,1], [2,1,4], [1,2,3], [2,1,2]}**

**T1 = [1,2,1,4]:**

* Consulta de uma receita contendo um ingrediente que já existe no estoque.

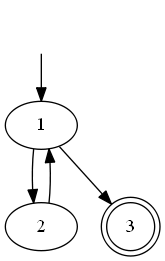
**T2 =** **[1,2,1,2,3]:**

* Consulta de uma receita contendo um ingrediente que já existe no estoque e um que não existe.

**Erros Encontrados:**

1-Quando a quantidade do ingrediente consultado fosse exatamente igual ao valor existente no estoque era retornado *false* ao invés de *true* no método isEstoqueSuficiente() devido a um erro na lógica de comparação.

**Método devolveItensReceitaParaEstoque()**



**Testes requeridos segundo critério e caminhos primos:**

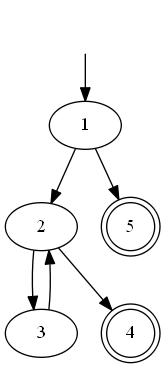
**TR = {[1,2,1], [2,1,2], [2,1,3]}**

**T1 = [1,2,1,2,1,2,1,3]:**

* Devolve para o estoque uma receita contendo três ingredientes quaisquer.

**Nenhum erro foi encontrado neste teste.**

**Método removeItensReceitaDoEstoque()**



**Testes requeridos segundo critério e caminhos primos:**

**TR = {[1,2,3], [2,3,2], [3,2,4], [3,2,3], [1,5], [1,2,4]}**

**T1 = [1,2,3,2,3,2,4]**

* Remove do estoque uma receita contendo dois ingredientes quaisquer presentes no estoque.

**T2 = [1,5]**

* Remove do estoque uma receita contendo um ingrediente qualquer não presente no estoque.

**T3 = [1,2,4]**

* Remove do estoque uma receita vazia.

**Nenhum erro foi encontrado neste teste.**

**Outros Testes:**

Teste testDefaultValues()

Testes cobertos:

* Verificação das condições iniciais dos atributos do Pedido.

**Nenhum erro foi encontrado neste teste.**

**Classe ItemTest**

**Para a classe Item, os testes foram realizados utilizando a técnica de particionamento do espaço de entradas, tentando fazer as combinações entre todos os tipos de entradas (itens) possíveis (entradas, sopas, saladas, etc), e com diferentes combinações entre as mesmas.**

**Foram testados os métodos isEstoqueSuficiente(), removeItemDoEstoque() e insereItemNoEstoque(), sendo que os dois últimos foram testados de forma conjuntas nos casos de teste de forma a aproveitar a sinergia entre os mesmos.**

**Teste testDefaultValues()**

Testes cobertos:

* Verificação das condições iniciais dos atributos do Item.

Erros encontrados:

**Nenhum erro foi encontrado neste teste.**

**Teste testIsEstoqueSuficienteItemNovo()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item novo (sem categoria definida);

**Erros encontrados:**

1-Era gerada uma NullPointerException no método isEstoqueSuficiente() quando o item consultado era novo (sem categoria definida).

**Teste testAddRemoveItemNovoNoEstoque()**

Testes cobertos:

* Adição de um item sem categoria no estoque;
* Remoção do estoque de um item sem categoria.

**Erros encontrados:**

1-Era gerada uma NullPointerException no método insereItemNoEstoque() quando o item a ser inserido era novo (sem categoria definida).

**Teste testIsEstoqueSuficienteEntrada()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item do tipo ‘Entrada’ onde não há estoque os ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘Entrada’ onde não há no estoque quantidade suficiente dos ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘Entrada’ onde há no estoque quantidade suficiente dos ingredientes presentes na receita;

**Erros encontrados:**

1-Foi encontrado o mesmo erro do teste ReceitaTest.isEstoqueSuficiente().

**Teste testAddRemoveEntradaNoEstoque()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item do tipo ‘Entrada’ onde não há estoque os ingredientes presentes na receita;
* Verifica se é possível remover do estoque um item do tipo ‘Entrada’ onde não há estoque os ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘Entrada’ onde há no estoque quantidade suficiente dos ingredientes presentes na receita;
* Verifica se é possível remover do estoque um item do tipo ‘Entrada’ onde há no estoque os ingredientes presentes na receita.

**Nenhum erro foi encontrado neste teste.**

**Teste testIsEstoqueSuficienteSalada()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item do tipo ‘Salada’ onde não há estoque os ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘Salada’ onde não há no estoque quantidade suficiente dos ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘Salada’ onde há no estoque quantidade suficiente dos ingredientes presentes na receita;

**Erros encontrados:**

1-Foi encontrado o mesmo erro do teste ReceitaTest.isEstoqueSuficiente().

**Teste testTestAddRemoveSaladaNoEstoque()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item do tipo ‘Salada’ onde não há estoque os ingredientes presentes na receita;
* Verifica se é possível remover do estoque um item do tipo ‘Salada’ onde não há estoque os ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘Salada’ onde há no estoque quantidade suficiente dos ingredientes presentes na receita;
* Verifica se é possível remover do estoque um item do tipo ‘Salada’ onde há no estoque os ingredientes presentes na receita.

**Nenhum erro foi encontrado neste teste.**

**Teste void testIsEstoqueSuficienteSopa()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item do tipo ‘Sopa’ onde não há estoque os ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘Sopa’ onde não há no estoque quantidade suficiente dos ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘Sopa’ onde há no estoque quantidade suficiente dos ingredientes presentes na receita;

**Erros encontrados:**

1-Foi encontrado o mesmo erro do teste ReceitaTest.isEstoqueSuficiente().

**Teste testTestAddRemoveSopaNoEstoque()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item do tipo ‘Sopa’ onde não há estoque os ingredientes presentes na receita;
* Verifica se é possível remover do estoque um item do tipo ‘Sopa’ onde não há estoque os ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘Sopa’ onde há no estoque quantidade suficiente dos ingredientes presentes na receita;
* Verifica se é possível remover do estoque um item do tipo ‘Sopa’ onde há no estoque os ingredientes presentes na receita.

**Nenhum erro foi encontrado neste teste.**

**Teste void testIsEstoqueSuficientePratoPrincipal()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item do tipo ‘PratoPrincipal’ onde não há estoque os ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘PratoPrincipal’ onde não há no estoque quantidade suficiente dos ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘PratoPrincipal’ onde há no estoque quantidade suficiente dos ingredientes presentes na receita;

**Erros encontrados:**

1-Foi encontrado o mesmo erro do teste ReceitaTest.isEstoqueSuficiente().

**Teste testTestAddRemovePratoPrincipalNoEstoque()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item do tipo ‘PratoPrincipal’ onde não há estoque os ingredientes presentes na receita;
* Verifica se é possível remover do estoque um item do tipo ‘PratoPrincipal’ onde não há estoque os ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘PratoPrincipal’ onde há no estoque quantidade suficiente dos ingredientes presentes na receita;
* Verifica se é possível remover do estoque um item do tipo ‘PratoPrincipal’ onde há no estoque os ingredientes presentes na receita.

**Nenhum erro foi encontrado neste teste.**

**Teste void testIsEstoqueSuficienteSobremesa()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item do tipo ‘Sobremesa’ onde não há estoque os ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘Sobremesa’ onde não há no estoque quantidade suficiente dos ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘Sobremesa’ onde há no estoque quantidade suficiente dos ingredientes presentes na receita;

**Erros encontrados:**

1-Foi encontrado o mesmo erro do teste ReceitaTest.isEstoqueSuficiente().

**Teste testTestAddRemoveSobremesaNoEstoque()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item do tipo ‘Sobremesa’ onde não há estoque os ingredientes presentes na receita;
* Verifica se é possível remover do estoque um item do tipo ‘Sobremesa’ onde não há estoque os ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘Sobremesa’ onde há no estoque quantidade suficiente dos ingredientes presentes na receita;
* Verifica se é possível remover do estoque um item do tipo ‘Sobremesa’ onde há no estoque os ingredientes presentes na receita.

**Nenhum erro foi encontrado neste teste.**

**Teste void testIsEstoqueSuficienteBebidaPreparada()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item do tipo ‘BebidaPreparada’ onde não há estoque os ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘BebidaPreparada’ onde não há no estoque quantidade suficiente dos ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘BebidaPreparada’ onde há no estoque quantidade suficiente dos ingredientes presentes na receita;

**Erros encontrados:**

1-Foi encontrado o mesmo erro do teste ReceitaTest.isEstoqueSuficiente().

**Teste testTestAddRemoveBebidaPreparadaNoEstoque()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item do tipo ‘BebidaPreparada’ onde não há estoque os ingredientes presentes na receita;
* Verifica se é possível remover do estoque um item do tipo ‘BebidaPreparada’ onde não há estoque os ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘BebidaPreparada’ onde há no estoque quantidade suficiente dos ingredientes presentes na receita;
* Verifica se é possível remover do estoque um item do tipo ‘BebidaPreparada’ onde há no estoque os ingredientes presentes na receita.

**Nenhum erro foi encontrado neste teste.**

**Teste void testIsEstoqueSuficienteBebidaPronta()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item do tipo ‘BebidaPronta’ onde não há estoque os ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘BebidaPronta’ onde não há no estoque quantidade suficiente dos ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘BebidaPronta’ onde há no estoque quantidade suficiente dos ingredientes presentes na receita;

**Erros encontrados:**

1-Era gerada uma NullPointerException no método isEstoqueSuficiente() quando o item consultado era uma bebida pronta (sem ingredientes).

**Teste testTestAddRemoveBebidaProntaNoEstoque()**

Testes cobertos:

* Verifica se o estoque é suficiente para um item do tipo ‘BebidaPronta’ onde não há estoque os ingredientes presentes na receita;
* Verifica se é possível remover do estoque um item do tipo ‘BebidaPronta’ onde não há estoque os ingredientes presentes na receita;
* Verifica se o estoque é suficiente para um item do tipo ‘BebidaPronta’ onde há no estoque quantidade suficiente dos ingredientes presentes na receita;
* Verifica se é possível remover do estoque um item do tipo ‘BebidaPronta’ onde há no estoque os ingredientes presentes na receita.

**Nenhum erro foi encontrado neste teste.**

**Classe ReservaTest**

**Para a classe Reserva, os testes foram realizados utilizando a técnica de particionamento do espaço de entradas para o método isReservaVálida().**

**Teste testIsReservaValida()**

Testes cobertos:

* Verifica se uma reserva é válida após 29 minutos da data estipulada;
* Verifica se uma reserva é válida após 31 minutos da data estipulada;
* Verifica se uma reserva recém-criada é válida (data default igual à data de criação).

**Nenhum erro foi encontrado neste teste.**

**Outros testes:**

**Teste testGettersAndSetters()**

Testes cobertos:

* Verificação se os atributos da classe Reserva são setados e retornados corretamente.

**Nenhum erro foi encontrado neste teste.**

**Classes AtendenteTest, AuxiliarDeCozinhaTest, CozinheitoTest, GarcomTest, GerenteTest**

**Outros testes:**

**Teste testDefaultValues()**

Testes cobertos:

* Verificação das condições iniciais dos atributos dos objetos.

**Nenhum erro foi encontrado neste teste.**

**Teste testGettersAndSetters()**

Testes cobertos:

* Verificação se os atributos das classes são setados e retornados corretamente.

**Nenhum erro foi encontrado neste teste.**

**Teste testCodigo()**

Testes cobertos:

* Verifica se os códigos dos objetos são criados automaticamente e incrementados a cada novo objeto.

**Nenhum erro foi encontrado neste teste.**

**Classe DadosTest**

**Para a classe Dados (banco de dados), os testes foram realizados utilizando a técnica de particionamento do espaço de entradas, tentando fazer as combinações entre todos os tipos de entradas possíveis para cada método, e com diferentes combinações entre as mesmas.**

**Foram testados os métodos relativos ao estoque (consultaQuantidadeIngredientesNoEstoque(String indrediente), addNoEstoque(String ingrediente, double quantidade), removeDoEstoque(String ingrediente, double quantidade)), os métodos reativos aos funcionários (addFuncionario(Funcionario f), removeFuncionario(Funcionario f), getListaDeFuncionariosAtivos(), getPorcentagemComissao(String cargo) e getSalarioFixo(String cargo)) e os métodos relativos aos itens do cardápio (getItemDoCardapio(int codigoItem)), sendo que os métodos que apresentavam relações entre si foram testados de forma conjuntas nos casos de teste de forma a aproveitar a sinergia entre os mesmos.**

**Teste testConsultaQuantidadeIngredientesNoEstoque()**

Testes cobertos:

* Consulta um ingrediente inválido;
* Consulta um ingrediente válido mas não existente no estoque;
* Consulta um ingrediente válido e existente no estoque;
* Adiciona um ingrediente inválido;
* Adiciona um ingrediente válido e não existente no estoque;
* Adiciona um ingrediente válido e já existente no estoque.
* Adiciona uma quantidade inválida de um ingrediente válido;

**Erros encontrados:**

1-Se a quantidade de ingrediente a ser adicionada no estoque era negativa, o método subtraía ao invés de somar. O método public void addNoEstoque(String ingrediente, Double quantidade) então foi alterado para verificar se a quantidade é negativa antes de prosseguir, e não faz nada neste caso.

2-Se o ingrediente fosse invalido (null), era criado um registro no estoque para o item nulo. O método public void addNoEstoque(String ingrediente, Double quantidade) então foi alterado para verificar se o ingrediente é inválido antes de prosseguir, e não faz nada neste caso.

**Teste testAddRemoveIngredientesNoEstoque()**

Testes cobertos:

* Tenta remover do estoque um ingrediente inválido;
* Tenta remover do estoque um ingrediente válido mas não existente no estoque;
* Tenta remover um ingrediente válido e existente no estoque;
* Tenta remover um ingrediente válido e existente no estoque, porém numa quantidade maior à existente;
* Tenta remover um ingrediente válido e existente no estoque, porém numa quantidade inválida (< 0).

**Erros encontrados:**

1-Se a quantidade de ingrediente a ser removida do estoque era inválida (negativa), o método *public boolean removeDoEstoque(String ingrediente, Double quantidade)*somava ao invés de diminuir. O método então foi alterado para verificar se a quantidade é negativa antes de prosseguir, e retorna false neste caso.

**Teste testAddRemoveFuncionarios()**

Testes cobertos:

* Tenta adicionar na lista de funcionários ativos um funcionário inválido;
* Tenta adicionar na lista de funcionários um funcionário de cada tipo;
* Verifica se cada funcionário adicionado na lista realmente se encontra presente;
* Tenta remover da lista um funcionário inválido;
* Tenta remover da lista um funcionário válido mas não inserido na lista;
* Tenta remover da lista um funcionário válido e presente na lista.

**Erros encontrados:**

1-Se o funcionário adicionado era inválido (null), o método *public void addFuncionario(Funcionario f)* gerada uma NullPointerException. O método então foi alterado para verificar se o funcionário é null antes de prosseguir, e não faz nada neste caso.

**Outros Testes:**

**Teste testGetSalarioEComissaoCargos()**

Testes cobertos:

* Verifica se os valores dos salários fixos e das porcentagens de comissão de cada tipo de funcionário foram setados corretamente conforme consta no arquivo de configuração;
* Verifica os mesmos valores para uma classe de funcionário não existente.

**Não foram encontrados erros neste teste.**

**Teste testCardapio()**

Testes cobertos:

* Verifica se os itens do cardápio foram lidos corretamente a partir do arquivo de configuração.
* Testa um atributo de cada tipo de item lido;
* Testa um item não existente no cardápio.

**Erros encontrados:**

1-Foram encontrados alguns erros no arquivo de configuração (algumas bebidas não possuíam valor de venda, havia itens com códigos repetidos, alguns itens apresentavam formatação fora do padrão, etc).

Estes erros foram corrigidos na versão do arquivo de configuração utilizada atualmente.