6SANDUBAS.java

Daniel Fernandes Campos

Engenharia de computação – Universidade Estadual de Feira de Santana (UEFS)

CEP: 44036-900 – Feira de Santana – BA – Brasil

dfc152@hotmail.com

Resumo. Com a comodidade trazida por aplicativos de entrega de comida e seu sucesso diversas empresas estão fazendo a migração para plataformas digitais. Empresas como a 6SANDUBAS que desejando melhorar sua organização e eficiência contratou a EcompJr para desenvolver um sistema de controle para seus pedidos, assim me foi solicitado que fizesse o *software* de controle dos pedidos.

1. Introdução

Como dito por Schawab e Davis,2018, com a nova revolução industrial a sociedade de forma geral vem passando por uma reformulação. Essa reformulação vem afetando principalmente o setor de serviços, dentre eles o de venda e entrega de alimentos, que reinventando-se na forma de organizar os pedidos e entrega-los na busca cada vez mais por melhorar a eficiência e lucros.

Assim sendo 6SANDUBAS visando melhorias na organização e eficiência contratou a EcompJr para desenvolver para eles um sistema de controle de pedidos. Portanto foi me solicitado que criasse um software capaz de passar em testes pré-determinados, como exemplificado na Figura 1.

User Stories

	Título	Breve Descrição
1	Cadastrar cliente	Um novo cliente é adicionado ao sistema, os clientes mais recentes devem ser armazenados no final da lista.
2	Remover cliente	Informando o telefone, todos clientes com este telefone são removidos do sistema se não tiverem pedidos associados.
3	Consultar cliente	Faz consulta por nome ou telefone, e lista clientes cadastrados que tenham estes dados (mesmo parcial) conforme a ordem que foram armazenados.
4	Criar novo pedido	Um novo pedido é criado e inserido ao final da lista de pedidos.
5	Inserir itens ao pedido	Após cadastro inicial do pedido, itens do cardápio são inseridos conforme solicitação do cliente, e o valor total do pedido é apresentado ao final.
6	Atualizar situação do pedido	O primeiro pedido em aberto é atualizado para pedido fechado.
7	Listar pedidos em aberto	Todos os pedidos em aberto são exibidos pela ordem que foram criados.
8	Adicionar opção ao cardápio	Uma nova opção é adicionada ao cardápio do sistema.
9	Apresenta cardápio	Apresenta as opções do cardápio do sistema.

Figura 1.Exemplo de testes pré-determinados.

Então torna-se necessário estudar e conhecer temas como POO (Programação orientada a objetos), listas encadeadas, pilhas e filas. Posteriormente sendo necessário criar 5 classes principais sendo elas: Cardápio, ItemPedido, Cliente, Pedido e System, e mais uma para lista e uma para fila.

2. Metodologia

Para começar a resolver este problema foram feitas discussões em sessões tutoriais de o que cada classe deveria ter ou como estas classes se relacionam umas com as outras. Inicialmente foram feitas as classes mais simples sendo elas: Cliente, Cardapio, ItemPedido, MyLinkedList e MyQueue. Assim sendo para explicar o que são classes parafrasearei Augusto, André e Mary, Cecília (1996), classes são e estruturas que servem de molde para objetos guardando dados, chamados de atributos (suas características como cor ou forma ou velocidade ...), e funções relacionadas, chamadas de métodos da classe (sendo ações que poderão ser executadas pelo objeto), em uma forma de envelope e objetos seriam instâncias dessa classe, assim sendo uma classe pode ter várias instâncias se necessário.

Como nosso trabalho se resume a criar as classes que passem nos testes não é necessário ou preciso criar uma espécie de *maincode* (código principal que iria interagir com o usuário, pegando suas entradas) apenas confeccionar as classes que passem nos testes que nos foram passados, assim sendo o diagrama de classes da Figura 2 foi feito em uma das sessões tutoriais para nos dar um norte geral de como as classes iriam interagir e quais seus atributos sendo deixado de fora seus métodos.

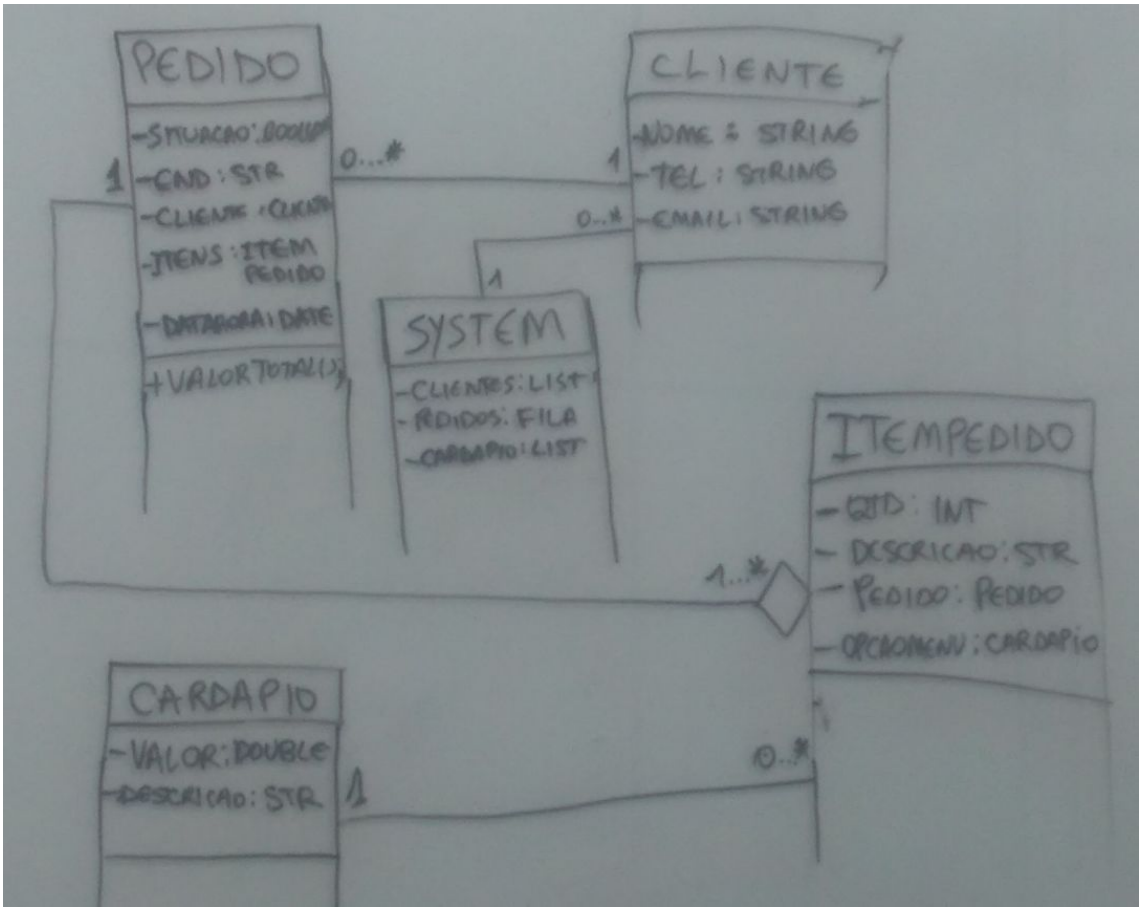

Figura 2. Diagrama de classes inicial apenas com atributos feito em sessão tutorial.

Os testes passados são testes de unidade com o framework JUnit, versão 4.12, que segundo Castro, Diogo (2007) é um framework que oferece um ambiente completo para realização de testes de unidade para o JAVA e estes teste servem para garantir com maior segurança que os métodos testados estão funcionando e facilitam a identificação de erros ou mal funcionamentos no código.

Um dos conceitos usados durante a resolução deste problema é o de herança e superclasse, como explicado por Horstmann, Cay (2004) no seu livro “Big Java”, herança é uma forma de uma classe passa seus métodos e atributos para outra classe, por exemplo se temos pessoa física e pessoa jurídica ambos derivam de pessoa por tanto criamos uma classe pessoa e fazermos elas herdam desta classe, para isso usamos o “*extends <nomeDaClasseMae>”* e chamamos essa classe mãe de superclasse das outras duas sendo só possível usar o “*extends”* apenas uma vez por classe.

Foi necessário também a confecção de uma lista encadeada que segundo Lafore, Robert (2004) em uma tradução livre seria um conjunto de objetos que contêm algum dado (informação guardada) e um *Link* para o próximo objeto da lista que tem a mesma estrutura que esse (seriam da mesma classe). Assim é possível criar uma sequência que guarda dados de forma dinâmica sendo chamado de lista encadeada.

Para percorrer essa lista é necessário a criação de um elemento chamado iterator, que tomando como referência Horstmann, Cay (2004) é um objeto criado para percorrer uma lista contendo dois métodos sendo eles "*hasNext*", que verifica se existem elementos para "iterar" (ou seja se existe elemento onde o ponteiro do iterator esta, se ele nao é *null*) , e "*next*", que é uma função que retorna o objeto atual e move o ponteiro do iterador para o próximo elemento.

Usando do mesmo princípio das listas encadeadas temos as filas, que usando como referência Lafore (2004) explica que filas são estrutura de dados semelhantes a listas encadeadas, porém o primeiro elemento a entrar é o primeiro elemento a sair (*Fist-In-Fist-Out*).

Esse código foi desenvolvido usando a IDE CoolBeans versão: 2018.12, tambem foi usado a biblioteca Junit versão 4.12 e foi desenvolvido no Windowns 10.

3. Resultados e Discussões.

Alguns dos principais problemas enfrentados na produção deste código ocorreram na elaboração da classe System, que é a responsável por controlar quase todos os objetos que foram criados. Nela foi necessário encontrar uma forma de integra os pedidos fechados aos pedidos abertos, a construção de um iterator (que até o momento não havia se mostrado necessário de criar) e o maior problema foi acessar métodos específicos de classes criadas utilizando uma lista genérica.

Então indo por partes, para a resolução desta integração dos pedidos fechados com os abertos, pois quando um pedido aberto é fechado é requerido que ele seja adicionado à lista de fechados, portanto foi criado um novo construtor para *pedidos* no qual era passado uma referência da lista de pedidos fechados para essa inserção acontecer dentro de um método do objeto de pedidos abertos. Já quanto ao iterador foi

apenas necessário criá-lo, inicialmente foi feito dentro da própria classe MyLinkedList, classe da lista encadeada, mas posteriormente foi passado para outra classe e esta foi integrada com a lista encadeada.

No java todas as classes são "filhas" da classe *Object* então em uma função com esse retorno podemos voltar qualquer objeto, porém sem fazer um casting (onde se força a transformação de uma variável de um dado tipo se torna de outro tipo) nesse retorno não podemos acessar os métodos específicos do objeto retornado, por tanto como havia necessidade desta interação foi criado uma superclasse chamada "*QualquerCoisa*" que apresentava uma espécie de "embrião" das funções porém sem implementação específica (sendo uma classe abstrata, que não pode se tornar objeto, que não pode ser inicializada), sendo as implementações dos métodos específicos feita na classe irá ser usado. Então usando uma lista encadeada cujo retorno seja essa superclasse criada podemos acessar qualquer método que será usado.

Com a existência desta lista especial para o PBL, funções para procurar por cliente, remover cliente, mostra cardápio e outras que envolvem a integração de lista e alguma outra classe foram colocadas nela, pois esta lista diferente da "MyLinkedList" foi desenvolvida especialmente para esse problema logo estas funções foram integradas a ela também.

Para o programa as entradas existentes são os arquivos de testes, que executam os testes unitários das classes criadas. Nesses testes são inicializados alguns objetos para clientes, cardápios, itens pedidos e pedidos e é realizado uma simulação de funcionamento, tendo como saída do programa apenas a porcentagem de resolução dos testes.

Como resultado deste problema temos um sistema composto pela seguinte classes:

1 - Classe de *clientes*: Responsável por guardar dados de um cliente específico, contendo seu nome, telefone e e-mail.

2 - Classe de *Cardapio*: Responsável por guardar dados relacionados às opções do menu disponíveis, guardando o nome do prato e o seu preço.

3 - Classe de *ItensPedidos*: Responsável por guardar as informações dos itens do cardápio de um determinado pedido, ele tem um ponteiro para o pedido a qual pertence, guarda também qual opção do cardápio ele representa, a quantidade daquela opção e por último uma possível modificação do prato.

4 - Classe de *Pedido*: Responsável por guardar os dados de um determinado pedido. Nela é guardado a qual cliente esse pedido pertence, o endereço de entrega, o horário que o pedido foi feito, se ele já foi entregue ou não e uma lista encadeada de todos os itens pedidos do cardápio.

5 - Classe System: Responsável controla o sistema, desde pedidos a cliente ou ao cardápio. Ela armazena uma lista para os clientes, uma para o cardápio, uma com todos os pedidos, uma para os pedidos fechados e uma fila para os abertos.

6 - Outras classes: Outras classes que servem como suportes para as citadas acima dentre elas listas encadeadas, filas e iterator.

Sendo que como interações de classes temos que a System contém as listas de pedidos, clientes e opções disponíveis no cardápio. Um pedido pertence a apenas um cliente este podendo ter mais de um pedido, ainda contém uma lista de itens pedidos, que por sua vez contém uma opção do cardápio.

3. Conclusão.

Neste PBL todos os objetivos foram completos, sendo conseguido 100 no *AllTestes*, passando em todos os 25 testes.

Uma possível melhoria seria talvez torna as classes mais independentes ou integrá-lo a um sistema.

Referências:

Schawab, Klaus e Davis, Nicholas (2018). **Shaping The Fourth Industrial Revolution**. World Economic Forum.

Augusto, André e Mary, Cecília (1996). **A LINGUAGEM DE PROGRAMAÇÃO JAVA**: orientação a objetos. unicamp .Available on: <http://www.ic.unicamp.br/~cmrubira/aacesta/java/javatut9.html>.Access in: 22/06/2019.

Castro, Diogo (2007). **Testes de unidade com JUnit**. DEVMEDIA .Available on: <https://www.devmedia.com.br/testes-de-unidade-com-junit/4637>.Access in: 22/06/2019.

Horstmann, Cay (2004). **Big Java**. Bookmann.

Lafore, Robert (2004). **Estruturas de dados & algoritmos em Java**. Ciência Moderna.