MAC-441: Programação Orientada a Objetos

Relatório da Fase 2

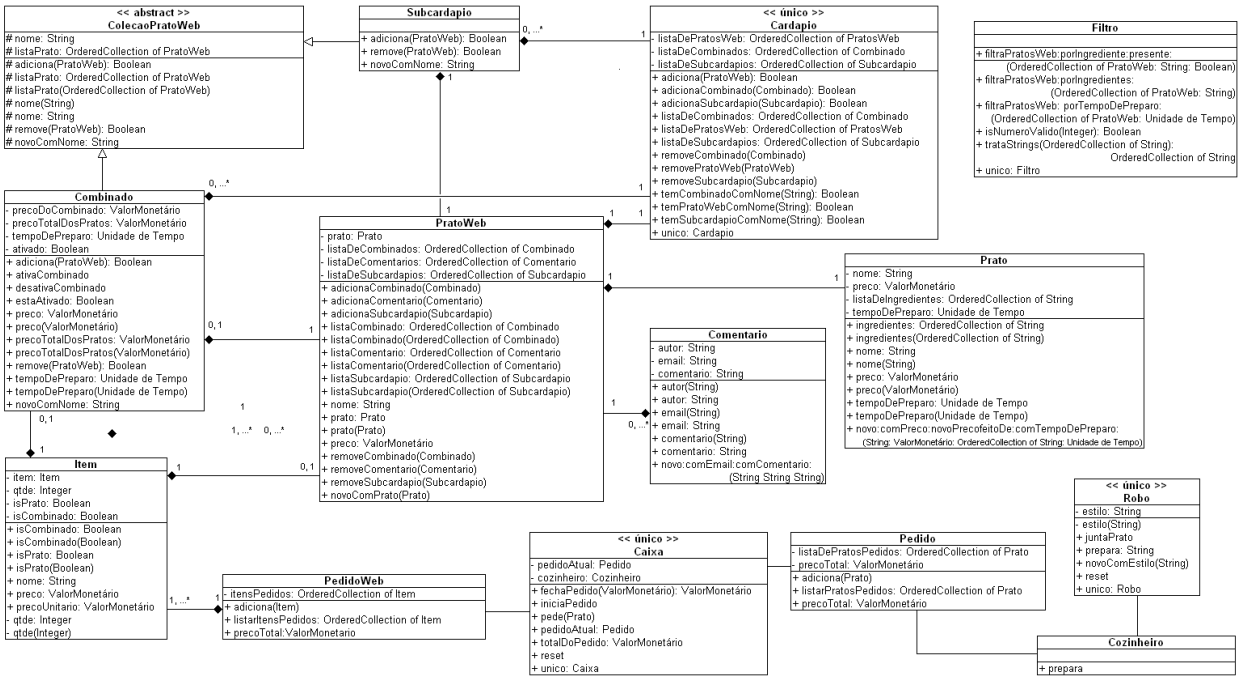
André Shoji Asato 5122749
Danielle Tiemy Paulo Miazaki 5122899
Mauricio Chui Rodrigues 5122930

Especificações

Nosso sistema foi implementado e executado nos ambientes a seguir.

- Squeak 3.8 – 6665 (Linux), com Seaside2.7a1-mb.205.mcz;
- Squeak 3.9 – 7067 (Windows), também com Seaside2.7a1-mb.205.mcz.

Diagrama UML do Sistema



(Favor verificar a imagem “uml.gif” no arquivo compactado)

Alterações nas Classes da Fase 1

Para a implementação do sistema, achamos necessário fazer três alterações na estrutura que entregamos como parte da fase anterior. Desconsideramos nesta seção a alteração dos pratos para conter tempos de preparo, por ser obrigatória. Cada nova funcionalidade resultante das alterações teve seus respectivos testes escritos.

Acréscimo da classe Item

Esta alteração fez com que mudássemos nosso modelo da fase anterior. Uma das interfaces web que desenvolvemos nesta fase é voltada à criação de pedidos, utilizada pelo garçom. Durante a implementação, entendemos que seria mais prático se bastasse a referência para um certo prato e fosse indicada a quantidade dele a ser preparada, mentalidade que não tínhamos antes. Tal característica de quantidade encontrava-se apenas no segundo modelo, através da classe Item. Ao adotá-la como parte da arquitetura, portanto, nosso modelo-base aproximou-se do segundo modelo da Fase 1.

```
Object subclass: #Item
        instanceVariableNames: 'item qtde isPrato isCombinado'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'EP2'
```

A classe que utilizamos é um pouco diferente da sugerida na primeira fase, como pode ser visto acima. Item representa uma certa quantidade de combinados ou pratos, quando anteriormente a única alternativa eram os pratos. As variáveis “isPrato” e “isCombinado” são úteis para diferenciar o tratamento dado pelo caixa ao item, pois, no caso de combinados, é preciso verificar cada prato nele contido.

Alteração na classe Cardapio

O cardápio, que continuou sendo singleton, deixou de guardar instâncias da classe Prato para guardar as da classe PratoWeb, que será explicada na próxima seção. Também passou a conter listas de combinados e sub-cardápios, parte das novas funcionalidades que optamos por desenvolver.

```
Object subclass: #Cardapio
        instanceVariableNames: 'listaDePratosWeb listaDeCombinados listaDeSubcardapios'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'EP1'
```

Entre os métodos que a classe disponibiliza está o “reiniciaComExemplo”, que permite o preenchimento do cardápio com uma boa quantidade de itens. A chamada deste método na correção é opcional, mas nos foi muito útil para os testes do sistema.

Alteração na classe Robo

O nosso primeiro robô apenas imprimia mensagens no Transcript, informando sobre preparo de ingredientes e junção de pratos. As informações foram mantidas, mas as mensagens agora são armazenadas como em um log, para que o gerente possa consultá-las através da interface quando achar conveniente. Para que as informações não fiquem armazenadas eternamente, o gerente tem a opção de limpar a lista de mensagens na própria interface web.

Desenvolvimento da Fase 2

O desenvolvimento desta fase contou com a implementação de mais classes para abrigar as funcionalidades que escolhemos, além da criação de interfaces web o mais amigáveis possível para os usuários. As classes podem ser verificadas no diagrama UML no início deste relatório, mas não estão inclusas as relacionadas a interfaces, devido ao número considerável de classes implementadas com esse fim.

Entre as funcionalidades disponíveis, optamos por estender o sistema antigo para suportar comentários dos clientes sobre os pratos (1 ponto); existência de combinados (1 ponto); sub-cardápios para melhor organização do cardápio (2 pontos); e filtragem do conteúdo dos cardápios (3 pontos). Portanto, como esperado, a soma mínima de 7 pontos foi atingida.

Os pacotes EP2 e EP2-Interfaces contêm as classes geradas para esta fase (inclusive a classe Item), de forma que o segundo é relacionado apenas às interfaces web. O pacote EP2-Testes contém os testes de unidade para as classes em EP2, já a interface foi testada com a ferramenta Selenium.

As Classes PratoWeb e PedidoWeb

Estas classes são fundamentais para o sistema. Um prato web é um objeto que armazena um prato, seu possível sub-cardápio e suas listas de combinados e comentários. Não vimos sentido em incluir todas essas informações em uma instância da classe Prato porque, para nós, um prato deveria ser similar à forma no mundo real, bastando que tenha preço, ingredientes, nome e tempo de preparo.

Um pedido web, por sua vez, tem grande importância no momento em que os pedidos são feitos. Foi visto, na primeira seção, que uma instância da classe Item representa um prato ou um combinado. Porém, um cliente pode incluir em seu pedido quantos itens quiser, para que então o caixa faça os cálculos. A forma que escolhemos de passar vários itens de uma vez para o caixa é a classe PedidoWeb, que armazena uma lista de itens.

Comentários dos Clientes

Cada comentário é representado por uma instância da classe Comentario, onde são encontradas as variáveis “autor”, “email” e “comentario”. Em nosso sistema, não só os clientes podem enviar seus comentários, como o próprio gerente pode fazer isso. No entanto, apenas o gerente pode remover os comentários inseridos.

```
Object subclass: #Comentario
        instanceVariableNames: 'autor email comentario'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'EP2'
```

Combinados e Sub-Cardápios

A implementação dos combinados é extremamente parecida com a de sub-cardápios, por ambos serem sub-conjuntos de pratos web do cardápio. Assim, implementamos as classes Combinado e Subcardapio como filhas de uma mesma classe, chamada ColecaoPratosWeb, onde estão as variáveis para nome da coleção e para a lista de pratos web nela contidos.

A classe Subcardapio apresenta apenas métodos para adição e remoção. A classe Combinado, no entanto, permite adição, remoção e ativação (o combinado pode existir mas estar inativo). Além disso, contém métodos para lidar com o tempo total de preparo do combinado e também para lidar com dois tipos de preços: o preço total dos pratos é nada mais do que a soma de todos os preços de pratos no combinado, enquanto o preço do combinado pode apresentar variações para mais ou para menos no preço total.

Quanto às interfaces, os combinados são criados e modificados em uma área administrativa separada dos sub-cardápios, porém o garçom visualiza tanto os combinados quanto os sub-cardápios (se existirem) em uma mesma interface, a que exibe o conteúdo do cardápio.

Filtro do Cardápio

Conforme especificado, o sistema permite a filtragem de pratos do cardápio pela presença ou não de um dado ingrediente; por um preço mínimo ou máximo; ou por um tempo de preparo mínimo ou máximo. Os filtros são independentes, cada um representado por um método diferente na implementação.

Foram entregues também um método e seu teste de unidade para filtragem pela presença ou ausência de múltiplos ingredientes, mas ele não é utilizado pelo sistema e só foi mantido por ter dado um certo trabalho antes de notarmos que o número de ingredientes era limitado a um.

Interfaces em Geral

Todas as interfaces são extensões da classe WAComponent, que permite a renderização no servidor do Seaside. Para que ficasse apresentável, cada interface recebeu um método chamado “style”, que delega a chamada para a classe Style, singleton. Esta classe tem como única função a devolução de todos os estilos (CSS) das páginas, realizada pelo método “devolveStyle”.

```
WAComponent subclass: #GerenteTelaInicial
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'EP2-Interfaces'

devolveStyle
	"Devolve o style para cada pagina."

	^'
body {background-color: #F2F2F2;}
```

A:active {color: #003399; font-family: Verdana; font-style: normal; font-size: 10pt; text-decoration: none}
A:visited {color: #003399; font-family: Verdana; font-style: normal; font-size: 10pt; text-decoration: none}
A:link {color: #003399; font-family: Verdana; font-style: normal; font-size: 10pt; text-decoration: none}
(..).

Testes de Aceitação da Interface Web

Os testes de aceitação foram entregues junto com o EP porque passaram pelas seguintes alterações:

1) Todos os comandos "submit" dos testes foram substituídos por "clickAndWait". Afinal, tínhamos botões em todas as ocasiões em que aquele comando era usado, e queríamos que eles fossem apertados.

2) O teste "teste-pedido-repetido.html" precisou ser mudado porque não funcionava. Executava-se o comando "openWindow", mas não se selecionava a nova janela (ela não era nem identificada com um id). Assim, tentavam-se executar os comandos seguintes na janela antiga. Para consertar, trocou-se o "openWindow" por "openWindowAndWait" (identificando devidamente a nova janela) e adicionou-se logo depois um "selectWindow" para selecionar a nova janela. Foi necessária a troca de "openWindow" por "openWindowAndWait" porque, depois do "openWindow", o Selenium tentava executar os outros comandos enquanto a janela ainda estava sendo aberta.

Observação

Caso aconteça a infelicidade de um pedido ficar preso, este comando deve ser executado no workspace: “GarcomNovoPedido pedido: nil”.