Relatório do Projeto

EP2 de MAC441 – Programação Orientada a Objetos – 12/06/2007

Marcos Bonci Cavalca – 5133237 – BCC/IME/USP

Observações iniciais:

Importante:

- Rodar o arquivo `inicializa.html` no Selenium antes dos testes! Esse arquivo cria os ingredientes necessários e o suco de laranja no cardápio. Pode também ser rodado a qualquer momento para voltar a esse estado.

- Os endereços estão em `localhost:8080` (me parece que em algumas configurações aparece o número `9090`, mas a versão baixada já vinha configurada com o número `8080`).

Problemas conhecidos:

- Apesar do modo `WAKomEncoded`, várias vezes há problemas com acentos.

- Além de problemas em tempo de execução, houve sérios problemas com acentos na hora de carregar o `.mcz` salvo (utitlizando Windows XP). Todos os caracteres acentuados ficam defeituosos. Aparentemente os testes no Selenium não são afetados (mesmo os que incluem palavras acentuadas).

- Aparentemente devido ao modelo com singletons (`Cardapio`, `ListaDeIngredientes`, `Robo`), ocorrem problemas na hora de retornar das telas desses objetos. Geralmente contorna-se o problema chamando a tela (do garçom ou gerente) diretamente depois de clicar no botão de saída. (O problema se inicia quando ocorre algum erro no objeto singleton ou quando ele expira, e só deixa de acontecer quando o objeto é reinicializado.)

- No caso de acontecer esse problema com um objeto `Pedido`, o sistema irá acusar "`Pedido em aberto`". Para cancelar esse pedido, basta chamar o `Gerente`.

- O Firefox pode impedir a abertura automática de janela nos testes do Selenium.

Testes de aceitação no Selenium:

Observações:

- Localização dos objetos raiz (ver "Objetos implementados") no Selenium:
  `Garcom:` http://localhost:8080/seaside/garcom
  `Gerente:` http://localhost:8080/seaside/gerente
  `ResteSingletons:` http://localhost:8080/seaside/reset.

Modificações:

>> `assertTitle` alterado para `assertNotBodyText`

- Não encontrei o comando para definir o "`title`". Como me parece coisa de pouca de importância, decidi fazer essa modificação.

>> Inserção de preço retirada

- Como o projeto permite pratos do cliente, os ingredientes são pré-definidos, e os preços dos pratos são determinados pelos preços dos seus ingredientes.

>> Tratamento com ingredientes modificado

- Também por conta dos ingredientes pré-definidos.

>> Algumas modificações no tratamento com elementos de listas e atualização de telas

- Por questões práticas e estéticas.

>> `submit` alterado para `clickAndWait`

- `submit`?..

Funcionalidades adicionais implementadas:

>> Filtro no Cardápio (3 pontos):

- As opções de filtro são exibidas junto com o cardápio, exceto quando ele está sendo editado. Na tela o filtro é dividido em dois: filtro por valor numérico (`a partir de` / `até`) e filtro por ingrediente (`com` / `sem`), mas a função utilizada internamente é a mesma. O código referente ao filtro está dentro do objeto `Cardapio`, dividido em duas funções: de exibição (`renderFiltroOn:`) e do filtro propriamente dito (`filtrarPor: relacao: valor:`).

>> Montagem de Prato (3 pontos)

- Na tela de pedido o usuário pode inserir um prato personalizado, o que na verdade executa a mesma chamada que a inserção de novo prato no cardápio (`self call: Prato new`), inserindo-o apenas no pedido, porém.

>> Limite nas Calorias (1 ponto):

- O usuário define junto com seu pedido o máximo de calorias que deseja. Se a soma total ultrapassar esse valor, o pedido não é fechado e volta-se à tela de edição do pedido (extamante da mesmo modo que é tratado o valor pago em relação ao preço total do pedido).

Observações sobre escolhas de desenvolvimento:

Objetos referentes a objetos reais:

- A escolha desses objetos foi feita de forma bem intuitiva, atribuindo um objeto a cada um dos objetos reais envolvidos na interface de atendimento do restaurante. Eles foram divididos em duas categorias: objetos raiz (representam pessoas e suas possíveis ordens) e objetos de formulário (com as informações solicitadas e/ou detalhes das ordens dadas, referentes aos respectivos objetos reais). Todos eles foram implementados como objetos `WAComponent` - o que me parece, inclusive, ser um processo padrão para o Seaside.

Objetos singleton:

`>> Cardapio:`
- Como a lista de pratos disponíveis é uma só, nãz faz sentido haver mais de um cardápio. Especialmente em um restaurante tipo fast-food.

`>> ListaDeIngredientes:`
- Do mesmo modo, a lista de ingredientes disponíveis é uma só.

`>> Robo:`
- Como só ha uma fila de pedidos, deve haver também apenas um robô recebendo as ordens de pedidos. Na organização interna da cozinha ele pode comandar outros robôs com papéis diferentes, que seriam representados por outro(s) objetos(s), mas isso foge do escopo do projeto.

Outros objetos:

- Além dos objetos citados acima, há outros quatro adicionais (um raiz, dois que apenas exibem notificações ao usuário e um auxiliar, sem exibição). Estes e os acima citados são descritos com mais detalhes na seção seguinte.

Distribuição das funções dentro dos objetos:

- A escolha das funções a serem implementadas e suas respectivas localizações também foram feitas de forma bem intuitiva, em geral criando uma função para cada ação que pode ser pedida pelo usuário. Também foram subdivididas algumas funções de exibição (mantendo o padrão de nome `render`*`Coisa`*`On:`) para facilitar a leitura e a reutilização.

Detalhes do modelo completo:

- O modelo completo, de acordo com o modo que foi implementado, pode ser visto mais adiante no diagrama UML anexo (também disponível no arquivo `ep2uml.pdf`, para facilitar a leitura).

Objetos implementados:

Objetos com exibição (`WAComponent`):

Objetos raiz (`WAComponent(Object) canBeRoot = true`):

>> `Gerente`:
- Exibe o menu de opções do gerente e define `Prato emAndamento: false`. (Ver observações iniciais.)

>> `Garcom`:
- Exibe o menu de opções do garçom/cliente.

>> `ResetSingletons`:
- Reinicializa o `Cardapio`, a `ListaDeIngredientes` e o `Robo`. Utilizado no início do arquivo `inicializa.html`.

Objetos de formulário:

>> `Cardapio`:
- Singleton. Exibição: Lista de pratos (`Garcom`) ou edição dessa lista (`Gerente`). Decide qual dos dois modos pelo atributo `Cardapio editavel`. Possui subrotinas de exibição utilizadas por `Pedido` (`renderCardapioOn:` e `renderFiltroOn:`).

>> `ListaDeIngredientes`:
- Singleton. Exibição: edição da lista de ingredientes.

>> `Robo`:
- Singleton. Exibição: apenas o log de mensagens e a opção de limpá-lo.

>> `Pedido`:
- Formulário do pedido. Exibe o cardápio para seleção de pratos. Ao fechamento, ativa o `Robo`.

>> `Prato`:
- Criação/edição de prato. Utilizado por `Cardapio` e `Pedido`.

>> `Ingrediente`:
- Edição de ingrediente. Utilizado apenas por `ListaDeIngredientes`.

Objetos de notificação:

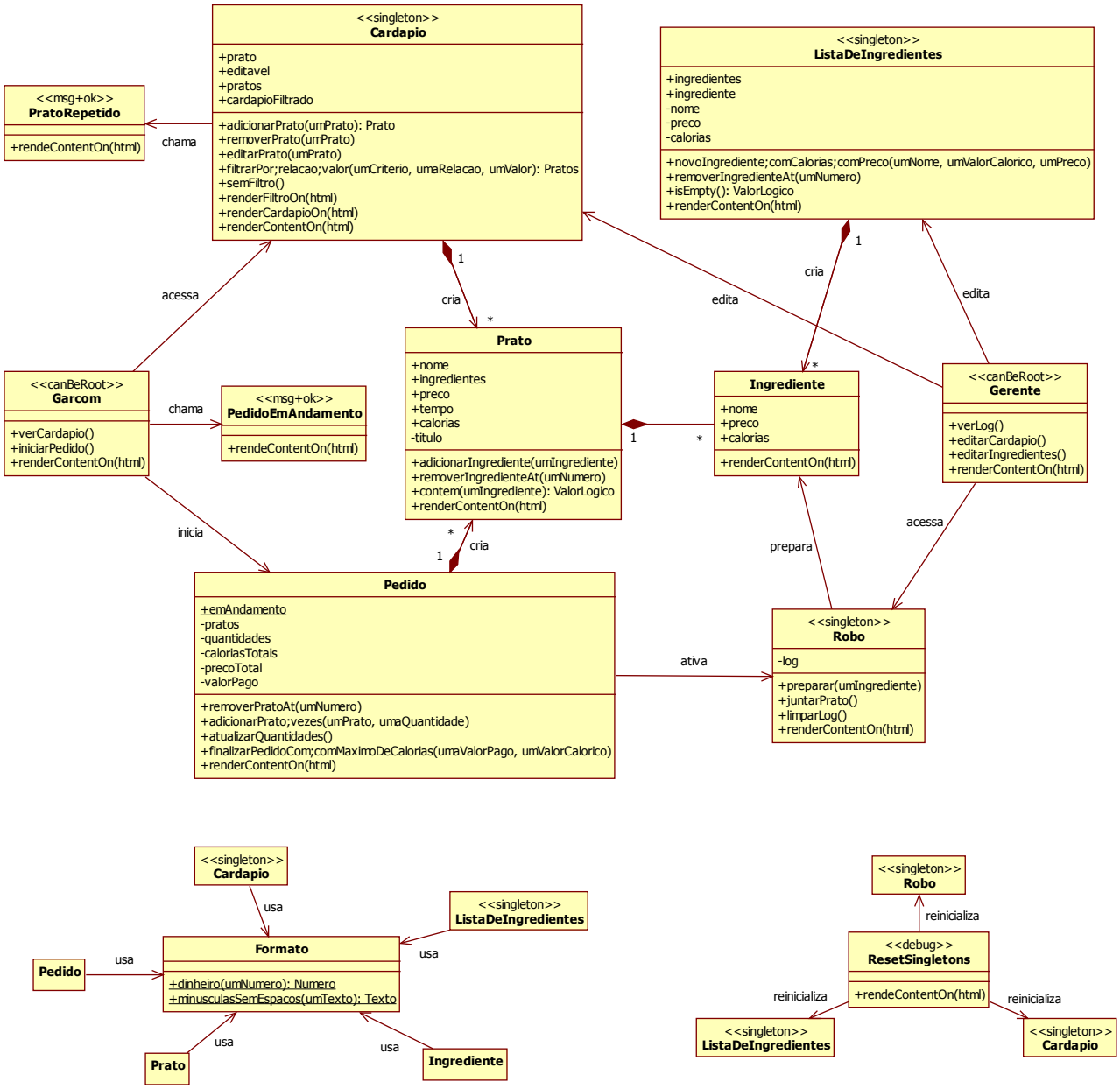
>> `PratoRepetido`:
- Exibe mensagem de prato com nome já existente.

>> `PedidoEmAndamento`:
- Exibe mensagem de pedido em andamento/em aberto.

Objetos sem exibição (`Object`):

>> `Formato`:
- Objeto auxiliar. Tem como único objetivo armazenar funções de formatação de texto utilizadas no projeto. (`Formato dinheiro:` e `Formato minusculasSemEspacos:`)

Diagrama UML detalhado do modelo utilizado:



Considerações finais:

Sobre os programas utilizados no desenvolvimento:

A idéia de implementar uma interface web com objetos é realmente interessante, assim como também é interessante a idéia de um ambiente virtual formado apenas por objetos que interagem entre si, que serve ao mesmo tempo de plataforma de desenvolvimento e como ambiente de execução, com efeitos em tempo real. Porém, como observado no fórum, a experiência com o Squeak foi realmente frustrante – chegando a ser quase traumatizante.

Na prática, o ambiente que deveria ser "amigo" do desenvolvedor acaba sendo "uma ameça constante". As ações mais triviais são pouco ou nada intuitivas, e várias vezes provocam efeitos inesperados (algumas vezes desastrosos). Em resumo, o que se sente ao utilizar o Squeak é uma grande falta de controle sobre o trabalho sendo feito – exatamente o contrário do que qualquer desenvolvedor deseja.