Trabalho Prático 1 - Estacionamento

Arthur Xavier
xavier@dcc.ufmg.br

Alexandre Pretti
alexandre.pretti@dcc.ufmg.br

Vitor de Melo
vitormelo@dcc.ufmg.br

13 de Abril de 2016

1 Introdução

Este trabalho consiste na implementação de um sistema de alocação de vagas para um estacionamento, visando a automatização do processo de entrada e saída do estabelecimento.

O estacionamento em questão possui quatro níveis e cada nível possui dez vagas, divididas entre motocicletas, veículos pequenos e grandes e vagas para portadores de necessidades especiais.

Ao chegar ao estacionamento, o cliente acessa um totem onde ele informa o tipo do seu carro. O sistema deve então informar se existe ou não vaga disponível que ele pode utilizar, em caso afirmativo, ele deve informar em qual nível e posição essa vaga fica, caso contrário, deve informar LOTADO. As vagas são numeradas sequencialmente de acordo com o nível e tipo de vaga.

Na saída do estacionamento, o cliente deve receber um ticket com o valor que deve ser pago. O custo é calculado de acordo com um valor associado ao tipo da vaga utilizada e à quantidade de tempo.

2 Implementação

O código lê por entrada padrão linha por linha e enquanto tiver linha chama a função realizarAtendimento. A saída também é padrão.

A função totem.realizarAtendimento gera um tipo Ticket. Se está entrando, a vaga é setada através do método estacionar. Pega placa e veículo e retorna uma string com a vaga ou diz que está lotado. Se está saindo, pega a placa e veículo, pega o horário de saída e o horário de entrada e subtrai. Seta a vaga como null para desocupá-la.

A função gerarTicket da classe Totem faz um tokenização da entrada, através das funções Pattern e Matcher da biblioteca jdk (java.util.regex), criando um ticket separando em 4 parâmetros.

A classe Ticket é bem trivial e contém vaga, horário, veículo e o tipo(entrada ou saída).

A classe mãe, Vaga recebe alguns parâmetros e dependendo do tipo de veículo define uma vaga e cada tipo de vaga define seu preço. As filhas são essas definições, portanto se o veículo é um carro grande, um carro pequeno, uma motocicleta, ou um veículo com necessidades especiais, o veículo é alocado em seu lugar correto e é definido pelo preço de sua determinada categoria.

A classe Predio define níveis, e níveis possuem vagas que podem ser dos 4 tipos já citados.

Outras funções, classes, são bem triviais e podem ou não ter sido citadas na documentação com suas funções, mesmo não citando seus nomes em si. Por exemplo a função setVaga definida na classe pública Ticket, que pela implementação é facilmente notada que seta uma vaga.

3 Testes

Foram criadas três rotinas de teste de aceitação automatizadas baseadas nos critérios especificados pelo problema. As três rotinas de teste criadas visam testar diferentes casos que podem ocorrer durante a execução do programa. O processo de teste implementado possui um arquivo de entrada e um com a saída esperada supondo o correto funcionamento do programa. O programa é, então, executado sobre a entrada fornecida e a saída comparada com o arquivo contendo a saída esperada. Caso haja alguma divergência, o teste falha.

Os três testes implementados foram:

1. **Normal -** testa o funcionamento do programa em condições normais, onde nenhum erro ou comportamento tratável ocorre;
2. **Lotado -** testa o comportamento e a reação do programa ao verificar que o estacionamento encontra-se em capacidade máxima e um veículo tenta estacionar;
3. **Inválido -** testa o comportamento do programa quando de uma entrada inválida.

Os testes possuem cobertura suficiente para atestar o correto funcionamento do programa de acordo com as especificações e suposições.

4 Conclusão

O objetivo deste trabalho relativo à disciplina de Programação Modular é buscar compreender e aplicar os conceitos aprendidos em sala de aula quanto a *Orientação a Objetos* e linguagem *Java*. Aplicar estes conceitos em exemplos reais é o que completa o aprendizado.

As dificuldades de compreensão do problema, como por exemplo qual algoritmo de alocação de veículos em vagas utilizar foram resolvidas por meio de suposições a fim de completar as lacunas cognitivas abertas pela descrição do problema. Acerca da implementação, as principais dificuldades se deram na importação e uso de bibliotecas padrão da JDK 7.

O resultado, entretanto, foi além de satisfatório, confirmado pelos testes executados e pela qualidade do código.

5 Bibliografia

1. Oracle. *Java^{TM} Platform, Standard Edition 7 API Specification.* Abr 2016. `https://docs.oracle.com/javase/7/docs/api/`