



IMD0412 – INTRODUÇÃO AO TESTE DE SOFTWARE

PROF. EIJI ADACHI M. BARBOSA

Roteiro de Laboratório – Sistema de Cobrança de Estacionamento

Objetivo:

O objetivo desta atividade é exercitar os conceitos de testes funcionais (caixa-preta) vistos em sala de aula, além de exercitar prática de implementação de testes executáveis com JUnit. Esta atividade valerá 10,0, mas com peso 20% na primeira unidade.

Observação:

Junto a este enunciado, veio um projeto do Eclipse para você iniciar sua implementação. Para tanto, basta você descompactar o .zip e importar o projeto pelo menu *File > Import > Existing Project*. Caso você use outra IDE, o projeto é um projeto Maven, então basta descompactar o .zip e usar a opção da sua IDE de importar um projeto Maven.

Cobrança de Estacionamento em Aeroportos:

O novo aeroporto da cidade está implantando seu novo sistema de cobranças de taxas de estacionamento. Você é o responsável por implementar e testar tal funcionalidade. A gerência do aeroporto estabeleceu uma série de critérios que devem ser implementados no programa de cobrança. O modelo de cobrança para uso do estacionamento do novo aeroporto prevê três tipos de uso: (i) Estacionamento a Curto Prazo, (ii) Estacionamento a Longo Prazo, e (iii) Estacionamento VIP. O tipo “Estacionamento a Curto Prazo” é pensado nos usuários que usarão estacionamento do aeroporto por apenas algumas horas. O tipo “Estacionamento a Longo Prazo” é pensado nos usuários que usarão o estacionamento do aeroporto por mais do que algumas horas, possivelmente usando por alguns dias. Já o tipo “Estacionamento VIP” é pensado nos usuários que querem conforto e segurança extra ao usar o estacionamento do aeroporto; tal serviço inclui serviço de valet e sala de espera VIP. O usuário deverá definir qual tipo deseja usar já na entrada do estacionamento do aeroporto, pois o tipo deverá ser registrado no sistema junto a hora de entrada. Ao sair do aeroporto, o sistema deverá calcular o valor a ser pago pelo usuário conforme as regras abaixo.

No “Estacionamento a Curto Prazo”, o usuário irá pagar R\$ 8,00 pela primeira hora e uma taxa de R\$ 2,00 a cada hora extra além da primeira hora; se apenas um minuto passar de uma determinada hora, a taxa de hora extra será cobrada integralmente. Por exemplo: se o usuário entrar às 12:30 e sair às 13:45 do mesmo dia, ele irá pagar R\$ 10,00: R\$ 8,00 pela hora inicial e R\$ 2,00 pelos minutos que excederam a primeira hora. Caso o usuário do “Estacionamento a Curto Prazo” use o estacionamento por mais de 24 horas, ele deverá pagar uma taxa extra de R\$ 50,00, além de pagar a taxa referente às horas extras. A cada 24 horas de permanência esta taxa de R\$ 50,00



será cobrada novamente, além da taxa referente à cada hora extra. Após o 7º dia de uso do estacionamento, a taxa de diária extra é reduzida para R\$ 30,00.

No “Estacionamento a Longo Prazo”, o usuário irá pagar R\$ 70,00 pelas primeiras 24 horas. Após as primeiras 24 horas, o usuário irá pagar uma taxa de R\$ 50,00 por cada diária extra; mesmo que apenas um minuto tenha passado do horário que se completa uma diária, a taxa da diária extra será cobrada integralmente. Após o 7º dia de uso do estacionamento, a taxa de diária extra é reduzida para R\$ 30,00. Além disso, a cada 30 dias de uso do estacionamento, uma taxa extra de R\$ 500,00 é cobrada, mantendo-se ainda a taxa de diária no valor de R\$ 30,00.

No “Estacionamento VIP”, o usuário irá pagar uma taxa de R\$ 500,00 que lhe dará direito a usar o estacionamento por uma semana. Após a primeira semana, será cobrada uma taxa de R\$ 100,00 para cada diária extra. Após o 14º dia de uso do estacionamento, a taxa de diária extra é reduzida para R\$ 80,00.

Implementação:

Para implementar o programa que calcula os valores devidos pelos usuários do estacionamento, você irá implementar e testar o módulo que de fato implementa a calculadora do estacionamento. Tal Calculadora deverá ser implementada no método `calculateParkingCost` da classe `Calculator`, o qual possui a seguinte assinatura:

Float calculateParkingCost(String checkin, String checkout, ParkingLotType type)

As strings `checkin` e `checkout` representam as datas de entrada e saída, respectivamente, e seguem o padrão “yyyy.MM.dd HH:mm”. Ex.: “2017.11.31 10:30”. Veja a classe `Main` para ver um exemplo de como manipular strings e datas, além de saber como calcular a diferença entre duas datas. Altere os valores das strings de entrada. Altere, por exemplo, a data de `checkin` para dois anos antes da data de `checkout`. Perceba que a diferença é calculada apenas em termos de dias, e não de anos.

Considere que os valores válidos para as entradas são:

- Checkin:
 - Ano [1970, 2017]
 - Mês [1, 12]
 - Dia [1, 31]
- Checkout:
 - Ano [1970, 2018]
 - Mês [1, 12]
 - Dia [1, 31]

Caso a string de entrada esteja fora do padrão definido acima, deverá ser lançada a exceção `imd0412.parkinglot.exception.DateFormatException` (veja na `Main` como uma string fora do padrão pode ser



identificada). Caso algum valor esteja fora do seu limite, ou uma data inválida seja passada (ex.: 30 de fevereiro), ou a data de check-out vier antes da data de check-in, lance a exceção `imd0412.parkinglot.exception.InvalidDataException`. Anos bissextos devem ser testados.

Para este problema, não comece pela implementação do método. Comece pela especificação dos casos de testes. Para isso, comece definindo as classes de equivalência e os limites das entradas. Em seguida, com base nas classes de equivalência, crie uma tabela de decisão para especificar o comportamento esperado do programa e, por fim, crie os casos de testes, cobrindo todas as regras identificadas. Só após toda esta sistematização de especificação do programa e dos casos de testes, comece a implementação do seu programa. Começar escrevendo os casos de testes e só em seguida implementar uma funcionalidade é o que se chama de “Test First”, prática base do *Test-Driven Development* (TDD).

Entregável:

Como nas atividades passadas, você deverá implementar classes de testes parametrizáveis: uma para os casos normais e outra para os casos excepcionais, além de uma classe “suíte” para agregar estas classes parametrizáveis. Também faz parte do entregável um documento descrevendo as classes de equivalência, os limites e a tabela de equivalência. Nos casos de testes executáveis implementados com JUnit, deverá ter alguma referência a qual regra da tabela de equivalência aquele caso de teste está relacionado (ex.: um comentário indicando algum ID da tabela de equivalência).

Ponto extra: quem entregar a calculadora com uma interface gráfica (web, ou desktop) ganhará até 1,0 ponto extra.

Gere um arquivo .zip nomeado seguindo o padrão `SeuNome_-SeuSobrenome-MATRICULA`. Ex.: `Eiji_-Adachi-201701234`. Este arquivo zip deverá conter o projeto `ParkingLot` com a implementação da calculadora e dos testes. A entrega será até terça-feira da próxima semana (29/08/2017). Venha pronto para apresentar sua solução, pois no dia irei sortear um aluno para apresentar a todos sua solução. Além disso, não mude a assinatura do método `calculateParkingCost`, pois irei trocar a sua implementação pela implementação de outro aluno para ver se os resultados dos testes são equivalentes.