UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CORNÉLIO PROCÓPIO

Lista de Exercícios: Teste caixa-preta
Tabela de decisão (TD)

Prof. André Takeshi Endo

Para cada um dos exercícios a seguir, aplique o critério de teste funcional: ***tabela de decisão.*** *Caso uma descrição de classes não seja fornecida, elabora as classes e implemente os casos de teste com o auxílio do* ***JUnit e Mockito****.*

Os exercícios de listas anteriores podem ser refeitas com a diferença que para projetar os casos de teste os critérios de teste sejam aplicados.

(Exercício 1) Cálculo da hipoteca (Mousavi). Considere os seguintes requisitos:

- R1- O sistema deve receber três valores como entrada: gênero (true → feminino e false → masculino), idade ([18, 55]) e salário ([0-10000]). Como saída, o sistema deve calcular o valor máximo da hipoteca para essa pessoa.
- R2- O valor máximo da hipoteca é calculado pela multiplicação do valor do salário com um fator (tabela no R4).
- R3- Mensagens de erro específicas devem ser geradas para valores inválidos de idade e salário.
- R4- O fator para calcular a hipoteca (R2) é definido pela tabela a seguir:

Categoria	Homem	Fator	Mulher	Fator
Jovem	18-35 anos	75	18-30 anos	70
Médio	36-45 anos	55	31-40 anos	50
Idoso	46-55 anos	30	41-50 anos	35

(Exercício 2) Considere um método para adição segura. A assinatura do método é apresentada a seguir:

int safe_add(int a, int b) throws OverflowException, UnderflowException;

Tal método faz a adição segura, tratando possíveis casos de overflow e underflow por meio do lançamento de exceções.

Dica: Em Java, as constantes `Integer.MIN_VALUE` e `Integer.MAX_VALUE` são usadas para referenciar o limite inferior e superior de variáveis do tipo int.

(Exercício 3) Um funcionário recebe um salário mensal e pode ganhar várias bonificações esporádicas ao longo do ano. Com essas informações, o software deve calcular o ganho mensal do funcionário e retornar qual a sua aliquota de imposto de renda (ver a tabela a seguir).

Base de cálculo mensal em R$	Alíquota %
Até 1.903,98	–
De 1.903,99 até 2.826,65	7,5
De 2.826,66 até 3.751,05	15,0
De 3.751,06 até 4.664,68	22,5
Acima de 4.664,68	27,5

(Exercício 4) Considere uma classe que receba como entrada o peso em quilos e a altura em metros e, com base no cálculo do IMC e na tabela a seguir, fornece a situação atual. Caso as entradas não atendam as seguintes restrições, exceções específicas precisam ser geradas:

- peso de 40 a 200 quilos.
- altura de 1,20 m a 2,5 m.

Resultado	Situação
Abaixo de 17	Muito abaixo do *peso*
Entre 17 e 18,49	Abaixo do *peso*
Entre 18,5 e 24,99	*Peso* normal
Entre 25 e 29,99	Acima do *peso*
Entre 30 e 34,99	*Obesidade* I
Entre 35 e 39,99	*Obesidade* II (severa)
Acima de 40	*Obesidade* III (mórbida)

(Exercício 5) Considere as classes abaixo.

Pessoa.java

```
public class Pessoa {
   int codigo, idade;
   String nome;

   //getters and setters
   public int getCodigo() {
      return codigo;
   }
   …
}
```

RHService.java

```
public interface RHService {
   public ArrayList<Pessoa> getAllPessoas();
}
```

PessoaDAO.java

```
public class PessoaDAO {

   RHService rhservice;

   public PessoaDAO(RHService rhservice) {
      this.rhservice = rhservice;
   }

   public boolean existePessoa(String nome) {
      ArrayList<Pessoa> pessoas = rhservice.getAllPessoas();
      for(Pessoa p : pessoas) {
         if(p.getNome().equalsIgnoreCase(nome))
            return true;
      }
      return false;
   }
}
```

Implemente casos de teste em JUnit para o método “existePessoa(..)”.

(Exercício 6) Considere as classes abaixo.

MathOps.java

```
public interface MathOps {
   public int fatorial(int n);
}
```

Somatoria.java

```
public class Somatoria {
   MathOps mathOps;

   public Somatoria(MathOps mathOps) {
      this.mathOps = mathOps;
   }

   /**
    * @param numeros
    * @return a somatoria do fatorial de cada inteiro no array numeros
    */
   public int somaDeFatoriais(int numeros[]) {
      //TODO

      return 0;
   }
}
```

Implemente o método “somaDeFatoriais()” segundo o que está especificado no comentário e CTs em JUnit.

(Exercício 7) [Adaptado de Acharya2015] Simulação de cotação de ações. O software observa tendências do mercado e:

- compra novas ações
- vende ações existentes

Considere que o sistema possui as seguintes classes:

- MarketWatcher
- Portfolio
- StockBroker
- Stock com os atributos symbol, companyName e price.

Os testes serão realizados sob o método perform() da classe StockBroker, apresentada a seguir. O método perform() funciona da seguinte forma:

- aceita um portfolio e uma ação (stock)
- recupera o preço atual de mercado
- compara o preço atual com a média das ações compradas
- Se o preço atual subiu 10%, ele vende 10 ações
  - Caso contrário, ele compra ações.

A classe Portfolio lê

```
public class StockBroker {
  private final static BigDecimal LIMIT
            = new BigDecimal("0.10");

  private final MarketWatcher market;

  public StockBroker(MarketWatcher market) {
    this.market = market;
  }

  public void perform(Portfolio portfolio,Stock stock) {
    Stock liveStock = market.getQuote(stock.getSymbol());
    BigDecimal avgPrice = portfolio.getAvgPrice(stock);
    BigDecimal priceGained =
        liveStock.getPrice().subtract(avgPrice);
    BigDecimal percentGain = priceGained.divide(avgPrice);
    if(percentGain.compareTo(LIMIT) > 0) {
      portfolio.sell(stock, 10);
    }else if(percentGain.compareTo(LIMIT) < 0){
      portfolio.buy(stock);
    }
  }
}
```

informações de um banco de dados e a classe MarketWatcher conecta na Internet para recuperar as cotações atuais. Nesse caso, para testar o método perform() essas funcionalidades não estão disponíveis.

Implemente casos de teste em JUnit e Mockito para testar o método perform().

(Exercício 9) ***Tabela de decisão.*** Ao adquirir um cartão de crédito, existem três condições. Primeiro, se você é um novo cliente no banco, você possui 10% de desconto na anuidade. Segundo, se você é um cliente ouro do banco, você tem 12% de desconto na anuidade. Por fim, se você possui
um cupom, você tem 25% de desconto (mas não pode ser usado com o desconto de novo cliente). Os descontos podem ser somados, se aplicáveis.

Implemente uma classe Java que calcula o desconto (apenas o esqueleto da classe) e casos de teste JUnit derivados da tabela de decisão. Envie junto uma planilha que ilustra a tabela de decisão criada.

(Exercício 11) Uma loja possui um software que auxilia o vendedor a tomar uma decisão quando o pagamento não é realizado com dinheiro. No caso, o pagamento pode ser realizado em cheque ou em cartão de crédito. O cliente ter cadastro na loja também é um fator a ser considerado. Considere que:

- A venda só é realizada diretamente se a compra for menor que R$50,00, o pagamento em cheque e o cliente tiver cadastro.
- O supervisor deve ser chamado se o pagamento for em cheque e (i) o cliente não tem cadastro ou (ii) o cliente tem cadastro mas a compra é maior que R$50,00.
- O sistema do comércio local deve ser consultado se a compra for menor que R$50,00 e o pagamento é via cartão de crédito.
- O sistema do SPC deve ser consultado se o pagamento for em cartão de crédito e (i) o cliente tem cadastro mas a compra é maior que R$50,00 ou (ii) o cliente não tem cadastro.

Com base na classe Java a seguir, implement os casos de teste em JUnit derivados da tabela de decisão. Envie junto uma planilha que ilustra a tabela de decisão criada.

```
public class TomadorDeDecisao {
    /**
     * @param valorDaCompra - valor da compra que sera realizada
     * @param tipoPagamento - pode ser "dinheiro", "cartao" ou "cheque"
     * @param clienteTemCadastro - verdadeiro caso o cliente tenha cadastro na loja
     *                          e falso, caso contrário.
     * @return possiveis saidas sao:
     *   - "pode realizar venda"
     *   - "chame o supervisor"
     *   - "consulte sistema comercial"
     *   - "consulte sistema SPC"
     */
            public String tomarDecisao(float valorDaCompra, String tipoPagamento, boolean
clienteTemCadastro) {
        //TODO
        return "";
    }
}
```

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CORNÉLIO PROCÓPIO

(Exercício 13) Considere os seguintes requisitos:

- R1- O sistema deve receber três valores como entrada: gênero (1 → feminino e 0 → masculino), idade ([21, 70]) e salário ([0-10000]). Como saída, o sistema deve calcular o valor máximo da hipoteca para essa pessoa.
- R2- O valor máximo da hipoteca é calculado pela multiplicação do valor do salário com um fator (tabela no R3).
- R3- O fator para calcular a hipoteca (R2) é definido pela tabela a seguir:

Masculino	**Fator**	**Feminino**	**Fator**
21-40 anos	80	21-43 anos	60
41-70 anos	40	44-70 anos	30

Implemente uma classe Java que calcula o valor máximo da hipoteca (apenas o esqueleto da classe) e casos de teste JUnit derivados da tabela de decisão. Elabore uma planilha que ilustra a tabela de decisão criada.

(Exercício 15) Uma loja possui um software que auxilia o vendedor a tomar uma decisão quando o pagamento não é realizado com dinheiro. No caso, o pagamento pode ser realizado em cheque ou em cartão de crédito. O cliente ter cadastro na loja também é um fator a ser considerado. Considere que:

- A venda só é realizada diretamente se a compra for menor que R$50,00, o pagamento em cheque e o cliente tiver cadastro.
- O supervisor deve ser chamado se o pagamento for em cheque e (i) o cliente não tem cadastro ou (ii) o cliente tem cadastro mas a compra é maior que R$50,00.
- O sistema do comércio local deve ser consultado se a compra for menor que R$50,00 e o pagamento é via cartão de crédito.
- O sistema do SPC deve ser consultado se o pagamento for em cartão de crédito e (i) o cliente tem cadastro mas a compra é maior que R$50,00 ou (ii) o cliente não tem cadastro.

Elabore uma tabela de decisão para a descrição apresentada. Derive casos de teste da tabela e implemente os casos de teste em JUnit.

(Exercício 17) ***Tabela de decisão.*** A classe Calculadora é usada para calcular o salário dos empregados de acordo com as seguintes regras:

- Se o tipo de empregado é “Assalariado40H” seu salário é 4.000.
- Se o tipo de empregado é “Assalariado20H” seu salário é 1.500.
- Se o tipo de empregado é “Horista” seu salário é o número de horas trabalhadas vezes 15.
  - Caso o horista trabalhe exatamente 40 horas nenhuma pendência é gerada (atributo pendencia da classe *Salario*).
  - Caso o horista trabalhe menos de 40 horas, a pendência “relatorio de ausencia” é gerada.
  - Caso o horista trabalhe mais de 40 horas, a pendência “autorizacao de hora-extra” é gerada.

Elabore uma tabela de decisão para a descrição apresentada. Derive casos de teste da tabela e implemente os casos de teste em JUnit, considerando as classes a seguir.

Salario.java

```
public class Salario {
    int valorSalario;
    String pendencia;

    public String getPendencia() {
        return pendencia;
    }
    public int getValorSalario() {
        return valorSalario;
    }
}
```

Calculadora.java

```
public class Calculadora {
    public Salario calcularSalario(String tipoEmpregado, int horasTrabalhadas) {
        //todo
        return null;
    }
}
```

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CORNÉLIO PROCÓPIO

(*Exercício 19)* A classe Autenticacao é usada para realizar o login de um dado usuário de acordo com as seguintes regras:

- Se o e-mail ou a senha estiverem vazio, o método retorna a mensagem "e-mail/senha não podem ser vazio.".
- Se o e-mail não estiver no formato "cccc@ccc.ccc" o método retorna "e-mail fora do formato".
- Se a senha tiver menos que 4 caracteres, o método retorna "a senha tem ao menos 4 caracteres".
- Passando por essas verificações, o método consulta o banco de dados e pode retornar as seguintes situações:
  - "usuario não existe" se o e-mail não está cadastro.
  - "senha incorreta" se o usuário está cadastrado mas a senha fornecida não é igual a armazenada.
  - "logado como admin" se o usuário é do tipo admin.
  - "logado" se o usuário é do tipo normal.

Elabore uma tabela de decisão para a descrição apresentada. Derive casos de teste da tabela e implemente os casos de teste em JUnit, considerando as classes a seguir. Envie junto uma planilha que ilustra a tabela de decisão criada.

Usuario.java

```
public class Usuario {
    private String email, senha, tipo;

    public Usuario(String email, String senha, String tipo) {
        this.email = email;
        this.senha = senha;
        this.tipo = tipo;
    }
    public String getEmail() {
        return email;
    }
    public String getSenha() {
        return senha;
    }
    public String getTipo() {
        return tipo;
    }
}
```

Autenticacao.java

```
public class Autenticacao {
    public String login(Usuario usuario) {
        return "";
    }
}
```

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CORNÉLIO PROCÓPIO

Referências

- Sujoy Acharya. “Mockito for Spring”, 2015. 178 pages, Packt Publishing Limited.