Relatório Trabalho Prático de Teste Baseado em Especificação

Método que faz o preenchimento à esquerda de uma string usando outra string.

Amanda Ferrari

Sumário

Introdução

Este relatório descreve o processo utilizado para testar o método `leftPad`, esse teste será baseado no livro de Mauricio Aniche como discutido em aula.

Passo 3 - Identificar as partições

Para cada entrada individualmente

Parâmetro `str`: `str` a string a ser preenchida.

A especificação diz que pode ser `null`. Como não temos muitas esspecificações, por enquanto teriamos o seguinte caso de teste:

- `str` com valor nulo;
- `str` vazio;

Parâmetro `padStr`: `padStr` a string usada para preencher.

A especificação diz que pode ser `null` ou vazio e que esses dois casos são tratados como espaço em branco. Dado essas especificações, vamos testar:

- `padStr` com valor nulo;
- `padStr` vazio;

Parâmetro `size`: O `size` é o tamanho máximo da string preenchida. Podemos testar das seguintes formas:

- `size` negativo;
- `size` menor que `str` e maior que 0;
- `size = 0`;

Para combinações de entradas

Como o método é para preenchimento de uma string com outra string, vamos testar combinações entre elas. - Podemos começar com o `size = 5`, `str = a` e `padStr = null` para assim verificar se teremos espaçoes em branco; - `str` vazia, `size = 3` e `padStr` = `-`; - `str` com o tamanho = `size`;

Para saídas esperadas

O método retorna uma string preenchida à esquerda, a string original se o preenchimento não for necessário ou `null` se uma string com o valor `null` for dada como entrada.

Saída:

- string original;
- `null`;

Pós jacoco

Verificando a cobertura do código, temos que o teste cobriu 100%, mas fazendo uma análise profunda, notamos que isso só foi possível pela simplicidade e tamanho do código e pelos casos de testes analisados minuciosamente para chegar a um bom resultado sem muita redundância. Talvez outro ponto que pode ter influenciado no resultado de cobertura foi a interpretação aberta para a solução do problema.

Passo 4 - Identificar os valores limite

Analisando os valores limites seguindo as partições:

Para string vazia:

- Sendo `str`, depois `padStr` e então `size`.

Para `size` com diferentes valores:

- `size < 0`;
- `size > 0`;
- `size = 0`;

Para string nula:

- Sendo `str`, depois `padStr` e então `size`.

Casos de teste criados a partir da utilização do critério MC/DC

No método `leftPad` encontramos uma condição com mais de um valor, sendo ele o `if (padStr == null || padStr.isEmpty())`, com isso derivamos a tabela verdade de condições a seguir:

T	**padStr = null**	**padStr vazio**	**resultado**
T1	V	V	V
T2	V	F	V
T3	F	V	V
T4	F	F	F

Inicialmente analisando o `padStr = null`:

- T1 temos todas as variaveis verdadeiras e o resultado também, então procuro um caso onde `padStr = null` é falso, ou seja o oposto de T1, o resultado também o oposto e o restante igual. Como não tem, parti para o proximo;
- Com T2 encontramos o T4 que vira o par de idependecia de T2;
- T3 não faz par com nenhum outro.

Para a condição `padStr.isEmpty()` temos:

- T3 fazendo par com T4.

No fim teremos:

- `padStr = null`: {T2,T4}
- `padStr.isEmpty()`: {T3,T4}

Com isso, temos os seguintes casos de teste: TR04 e TR01 que satisfaz o par {T2,T4}. E os casos TR05 e novamente TR01 que satisfaz o outro par {T3,T4}. Ambos casos já haviam sido escritos e estão definidos no proximo tópico. ## Passo 5 - Derivar os casos de teste

Seguindo os passos demonstrados no slide, vamos testar casos de excessão apenas uma vez e não combiná-los.

Casos de teste de excessão:

- TR01: `size` = len(str) + len(padStr);
- TR02: `str == null`;
- TR03: `str` vazio;
- TR04: `padStr == null`;
- TR05: `padStr` vazio;
- TR06: `size == 0`;
- TR07: `size > 0` e `size < len(str)`;
- TR08: `size < 0`;

Passo 6 - Automatizar os casos de teste usando JUnit

O principal desafio encontrado ao desenvolver os casos de teste foram escolher quais casos de testes realizar de modo que não fique muito repetitivo e que supra as especificações. Outro grande desafio encontrado desta vez foi o checkstyle, que em minha opinião as vezes mais atrapalhava do que realmente ajudava na qualidade, encontrei um problema na qual ele apenas apontava um número como mágico.

Eventuais erros e defeitos

Algo que não foi definido, tratado e nem pensado é o caso de padStr ter mais de um caracter, depois de pensar nesse caso, criei o TR09 onde falhou. Como não tem definição para o caso, minha sugestão seria levantar `IllegalArgumentException`.

Cobertura de código

Por ser um código relativamente simples e com condições também simples, a cobertura antes e depois de satisfazer os critérios MC/DC ficaram em 100%.