Relatório Trabalho Prático de Teste Baseado em Especificação

Validação do método preenche(String str, int size, String stringPreenche)

Breno Farias da Silva

Sumário

Introdução

Este relatório descreve o processo de teste da função `preenche`, responsável por preencher uma string à esquerda até atingir um tamanho especificado, utilizando uma string de preenchimento fornecida. O método recebe três parâmetros: a string original (`str`), o tamanho desejado (`size`) e a string de preenchimento (`stringPreenche`). O relatório é dividido entre os testes baseados em especificação e os testes estruturais utilizando o critério MC/DC, além da análise de cobertura com a ferramenta Jacoco.

Teste baseado em especificação

Análise da especificação

A especificação define os seguintes comportamentos:

- Se `str` for `null`, o método retorna `null`.

- Se `stringPreenche` for `null` ou vazia, deve ser tratada como um espaço em branco.
- Se o tamanho de `str` for maior ou igual a `size`, o método retorna `str`.
- Caso contrário, a string é preenchida à esquerda com `stringPreenche` até atingir o tamanho `size`.

Identificação de partições de equivalência

- `str`:
  - `null` ⟶ retorna `null`
  - não nula:
    - vazia: `""`
    - 1 caractere: `"x"`
    - múltiplos caracteres: `"abc"`
- `stringPreenche`:
  - `null` ou `""` ⟶ tratado como `" "`
  - não vazia:
    - 1 caractere: `"*"`
    - mais de 1 caractere: `"-="`
- `size`:
  - menor que `str.length()` ⟶ retorna `str`
  - igual a `str.length()` ⟶ retorna `str`
  - maior que `str.length()`:
    - diferença de 1 caractere
    - diferença de múltiplos caracteres (para avaliar padrão de repetição)

Casos de teste derivados

Caso	`str`	`size`	`stringPreenche`	Esperado	Justificativa
1	`null`	5	“-”	`null`	`str` é `null`
2	`""`	3	“.”	“...”	`str` vazia, preenchida completamente
3	“a”	3	“-”	“–a”	preenchimento de 2 com 1 caractere
4	“abc”	3	“*”	“abc”	tamanho já suficiente
5	“abc”	4	“*”	“*abc”	precisa de 1 caractere
6	“abc”	6	“-=”	“-=-abc”	padrão de preenchimento longo
7	“abc”	5	`""`	” abc”	stringPreenche vazia ⟶ espaço
8	“abc”	5	`null`	” abc”	stringPreenche null ⟶ espaço
9	“abcdef”	4	“#”	“abcdef”	já tem tamanho maior que size
10	“abc”	10	“123”	“1231231abc”	múltiplos preenchimentos + truncamento

Caso	`str`	`size`	`stringPreenche`	Esperado	Justificativa
11	“abc”	5	” ”	” abc”	espaço explícito como preenchimento

Teste estrutural (MC/DC)

Passo 1: Definir as condições atômicas do método

- **C1:** `str == null` (true/false)
- **C2:** `stringPreenche == null` (true/false)
- **C3:** `stringPreenche.isEmpty()` (true/false) — só avaliada se `C2 = false`, mas para a tabela verdade, considerada independente
- **C4:** `fillLength <= 0` (true/false), onde `fillLength = size - str.length()`

Como temos 4 condições binárias, o total será $2^4 = 16$ linhas na tabela verdade completa.

Passo 2: Tabela verdade completa

Caso	C1	C2	C3	C4	Resultado esperado	Observação
T1	1	0	0	0	retorna null	str null
T2	1	0	0	1	retorna null	str null
T3	1	0	1	0	retorna null	str null
T4	1	0	1	1	retorna null	str null
T5	1	1	0	0	retorna null	str null
T6	1	1	0	1	retorna null	str null
T7	1	1	1	0	retorna null	str null
T8	1	1	1	1	retorna null	str null
T9	0	0	0	0	preenche com stringPreenche	Caso normal preenche
T10	0	0	0	1	retorna str	fillLength <= 0
T11	0	0	1	0	preenche com espaço	stringPreenche.isEmpty() = true
T12	0	0	1	1	retorna str	fillLength <= 0
T13	0	1	0	0	preenche com espaço	stringPreenche == null
T14	0	1	0	1	retorna str	fillLength <= 0
T15	0	1	1	0	preenche com espaço	stringPreenche == null e isEmpty

Caso	C1	C2	C3	C4	Resultado esperado	Observação
T16	0	1	1	1	retorna str	fillLength <= 0

Passo 3: Explicação dos pares de independência para MC/DC

O critério MC/DC exige que cada condição atômica:

- Afete a decisão final
- Seja demonstrada por pares de casos de teste onde apenas aquela condição muda e as outras permanecem iguais

Passo 4: Seleção de pares independentes para cada condição

- **C1:** Pares (T8, T16)
- **C2:** Pares (T9, T13)
- **C3:** Pares (T9, T11)
- **C4:** Pares (T9, T10)

Passo 5: Tabela final reduzida para MC/DC

Caso	C1	C2	C3	C4	Resultado esperado
T8	1	1	1	1	retorna null
T16	0	1	1	1	retorna str
T9	0	0	0	0	preenche com stringPreenche
T13	0	1	0	0	preenche com espaço
T11	0	0	1	0	preenche com espaço
T10	0	0	0	1	retorna str

Passo 6: Testes JUnit para esses casos

```
@Test
void testStrNull() {
    assertNull(preenche(null, 5, "-"));
}

@Test
void testPreencheComStringPreenche() {
    assertEquals("***abc", preenche("abc", 6, "*"));
}

@Test
void testPreencheComEspacoStringPreencheNull() {
    assertEquals("  abc", preenche("abc", 5, null));
}
```

```java
@Test
void testPreencheComEspacoStringPreencheEmpty() {
    assertEquals("  abc", preenche("abc", 5, ""));
}

@Test
void testRetornaStrFillLengthZero() {
    assertEquals("abcdef", preenche("abcdef", 4, "#"));
}
```

Análise de cobertura com Jacoco

A cobertura de código foi medida utilizando a ferramenta Jacoco em dois momentos:

Sem os testes MC/DC

- **Instruções cobertas:** 93% (40 de 43 instruções)
- **Branches cobertos:** 100% (10 de 10 branches)
- **Linhas cobertas:** 12 de 13 linhas
- **Métodos cobertos:** 2 de 2 métodos
- **Classes cobertas:** 1 de 1 classe

Com os testes MC/DC

- **Instruções cobertas:** 93% (40 de 43 instruções)
- **Branches cobertos:** 100% (10 de 10 branches)
- **Linhas cobertas:** 12 de 13 linhas
- **Métodos cobertos:** 2 de 2 métodos
- **Classes cobertas:** 1 de 1 classe

Conclusão

O critério MC/DC foi aplicado com sucesso, identificando as condições atômicas relevantes e garantindo que cada uma seja testada de forma independente. Os casos selecionados cobrem todas as combinações necessárias para garantir o correto funcionamento da função `preenche`. A ferramenta **Jacoco** pode ser usada para validar a cobertura dos testes.