Teste Baseado na Especificação

Exemplo

Requisitos

Método: substringBetween(...)

Busca, em uma string, por substrings delimitadas por marcações de inicio e fim, e retorna todas as strings encontradas em um vetor.

- str - String que contém as substrings. Null retorna null; string vazia retorna outra string vazia.
- open - String que identifica o início da substring. String vazia returna null.
- close - String que identifica o fim da substring. String vazia returna null.

O método retorna um array de substrings, ou retorna null se não encontrar nada.

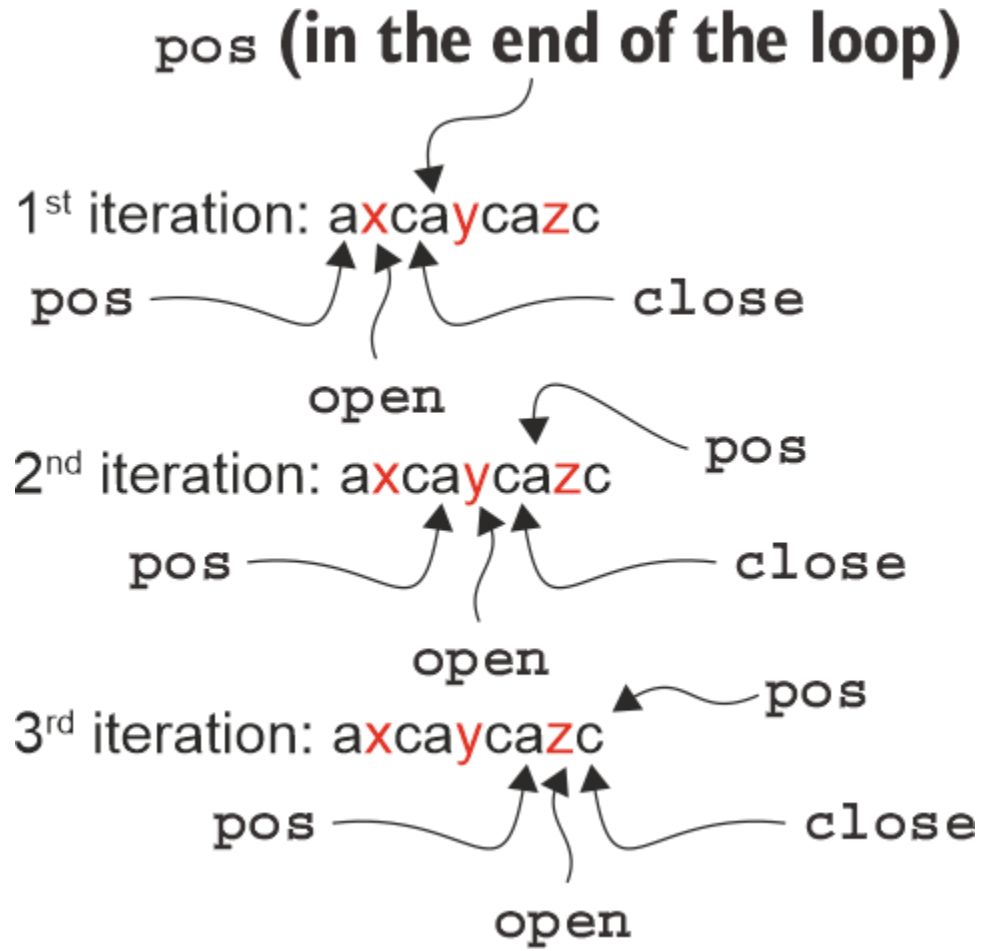
Requisitos

Exemplo:

Se str = “a**x**ca**y**ca**z**c”, open = “a”, and close = “c”, a saída será um array contendo [“x”, “y”, “z”].

Isso porque a substring “a<algo>c” aparece três vezes na string original: a primeira tem “x” no meio, a segunda tem “y,” e a ultima “z.”

```
public static String[] substringsBetween(final String str,
 final String open, final String close) {

  if (str == null || isEmpty(open) || isEmpty(close)) {
    return null;
  }

  int strLen = str.length();
  if (strLen == 0) {
    return EMPTY_STRING_ARRAY;
  }

  int closeLen = close.length();
  int openLen = open.length();
  List<String> list = new ArrayList<>();
  int pos = 0;
  while (pos < strLen - closeLen) {
    int start = str.indexOf(open, pos);

    if (start < 0) {
      break;
    }

    start += openLen;
    int end = str.indexOf(close, start);
    if (end < 0) {
      break;
    }

    list.add(str.substring(start, end));
    pos = end + closeLen;

  }

  if (list.isEmpty()) {
    return null;
  }
  return list.toArray(EMPTY_STRING_ARRAY);
}
```


pos (in the end of the loop)
1st iteration: axcaycazc
pos
close
open
2nd iteration: axcaycazc
pos
pos
close
open
3rd iteration: axcaycazc
pos
pos
close
open

Para vocês fazerem

Escrevam os casos de testes que vocês acham que devemos ter.

O formato não importa, pode ser algo como “todos os parâmetros nulos”.

Entendendo os requisitos, entradas e saída

- Requisitos normalmente são compostos de três partes:
  - O que o programa deve fazer
  - As entradas
  - A saída

Entendendo os requisitos, entradas e saída

- O objetivo do método é coletar todas as substrings de uma string que são delimitadas por uma substring de abertura e uma substring de fechamento (o usuário as fornece).
- O programa recebe três parâmetros:
  - a) str, que representa a string da qual o programa extrairá as substrings
  - b) open, que indica o início de uma substring
  - c) close, que indica o fim da substring
- O programa retorna um array composto por todas as substrings encontradas pelo programa.

Explorando o que o programa faz

- Testes que exploram o método de forma não sistemática pode aumentar sua compreensão dele.
- Isso foi verificado ao se observar desenvolvedores de software profissionais escrevendo casos de teste para métodos que nunca tinham visto antes (Aniche, Treude e Zaidman, 2021).
  - Maurício Aniche, Christoph Treude, and Andy Zaidman. 2021. “How Developers Engineer Test Cases: An Observational Study.” Transactions on Software Engineering (TSE)
- Esta etapa é mais relevante quando você não escreveu o código — se você o escreveu, essa fase de exploração pode não ser necessária.

Explorando o que o programa faz

- Um caso feliz: str = “abcd” com open = “a” e close = “d”. Saída esperada: [“bc”].
  Se esse teste passar, exploramos um pouco mais.
- Várias substrings: str = “abcdabcdab” com open = “a” e close = “d”. Saída esperada: [“bc”, “bc”].
- Strings de início e fim maiores que um caractere: str = “aabcddaabfddaab” com open = “aa” e close = “dd”. Saída esperada: [“bc”, “bf”].

```
@Test
void simpleCase() {
  assertThat(
    StringUtils.substringsBetween("abcd", "a", "d")
  ).isEqualTo(new String[] { "bc" });
}

@Test
void manySubstrings() {
  assertThat(
    StringUtils.substringsBetween("abcdabcdab", "a", "d")
  ).isEqualTo(new String[] { "bc", "bc" });
}

@Test
void openAndCloseTagsThatAreLongerThan1Char() {
  assertThat(
    StringUtils.substringsBetween("aabcddaabfddaab", "aa", "dd")
  ).isEqualTo(new String[] { "bc", "bf" });
}
```

Explorando entradas e saídas -> partições

- Devemos encontrar uma maneira de priorizar e selecionar um subconjunto de entradas e saídas que nos dê certeza suficiente sobre a correção do programa.

- Embora o número de entradas e saídas possíveis do programa seja quase infinito, alguns conjuntos de entradas fazem com que o programa se comporte da mesma maneira, ou seja, são da mesma partição:
  - str = “abcd” com open = “a” e close = “d”. Saída esperada: [“bc”].
  - str = “xyzw” com open = “x” e close = “w”. Saída esperada: [“yz”].

Explorando entradas e saídas -> partições

- Uma maneira sistemática de fazer explorar as entradas e saídas para encontrar partições:
  - Cada entrada individualmente: “Quais são as possíveis classes de entradas que posso fornecer?”
  - Cada entrada em combinação com outras entradas: “Que combinações posso tentar entre as substrings de abertura e fechamento?”
  - As diferentes classes de saída esperadas deste programa: “Ele retorna arrays? Ele pode retornar um array vazia? Ele pode retornar nulos?”

Explorando entradas e saídas -> partições

- Parâmetro `str`

Explorando entradas e saídas -> partições

- Parâmetro `str`
  a) String nula
  b) String vazia
  c) String de tamanho 1
  d) String de tamanho > 1 (qualquer string)

Explorando entradas e saídas -> partições

- Parâmetro `open`
  a) String nula
  b) String vazia
  c) String de tamanho 1
  d) String de tamanho > 1 (qualquer string)
- Parâmetro `close`
  a) String nula
  b) String vazia
  c) String de tamanho 1
  d) String de tamanho > 1 (qualquer string)

Explorando entradas e saídas -> partições

- Uma vez que as entrada são analisadas individualmente em detalhes, exploramos possíveis combinações entre elas.

Explorando entradas e saídas -> partições

- Parâmetros (str, open, close):

Explorando entradas e saídas -> partições

- Parâmetros (str, open, close): open e close podem ou não estar na string str. Além disso, open pode estar lá, mas close não (e vice-versa).
  a) str não contém nem open nem close.
  b) str contém open, mas não contém close.
  c) str contém close, mas não contém open.
  d) str contém open e close.
  e) str contém open e close várias vezes.

Note que isso pode depender, de certa forma, da experiência do testador.

Explorando entradas e saídas -> partições

Analisando a saída

- Array de strings
  - Array nulo
  - Array vazio
  - Item único
  - Vários itens
- Cada string individual
  - Vazia
  - Caractere único
  - Vários caracteres

Explorando entradas e saídas -> partições

Analisando a saída

- Pode parecer que não é necessário refletir sobre as saídas. Afinal, se você raciocinou corretamente sobre as entradas, provavelmente está exercitando todos os tipos possíveis de saídas. Este é um argumento válido.
- No entanto, para programas mais complexos, refletir sobre as saídas pode ajudá-lo a ver um caso de entrada que você não identificou antes.

Analisando as fronteiras

- Uma fronteira direta acontece quando a string str passa de vazia para não vazia, pois o programa para de retornar vazio e (possivelmente) começará a retornar algo. *Essa fronteira já é coberta pelas partições criadas.*
- Analisando as partições na categoria (str, open, close), o programa não pode ter substrings, uma substring ou várias substrings. E as strings open e close podem não estar na string; ou, mais importante, eles podem estar na string, mas sem substring entre eles. *Essa é uma fronteira que devemos exercitar.*

Analisando as fronteiras

- Fronteira
  - Quando `str` contém tanto `open` quanto `close` e o tamanho da substring encontrada muda de zero para maior que zero, o programa passa a retorna essa substring.
    - `str` contém `open` e `close`, sem caracteres entre elas.
    - `str` contém `open` e `close`, com caracteres entre elas.

    - O segundo teste não é necessário, pois outros testes já exercitam essa situação. Portanto, podemos descartá-lo.

Elaborando Casos de Testes

- Com as entradas, saídas e fronteiras devidamente dissecados, podemos elaborar casos de teste concretos.
- Idealmente, combinaríamos todas as partições que criamos para cada uma das entradas.
- O exemplo tem quatro categorias, cada uma com quatro ou cinco partições.

Elaborando Casos de Testes

- Parâmetro `str`
  a) String nula
  b) String vazia
  c) String de tamanho 1
  d) String de tamanho > 1
- Parâmetro `open`
  a) String nula
  b) String vazia
  c) String de tamanho 1
  d) String de tamanho > 1
- Parâmetro `close`
  a) String nula
  b) String vazia
  c) String de tamanho 1
  d) String de tamanho > 1

- Parâmetros (`str`, `open`, `close`)
  a) str não contém nem open nem close.
  b) str contém open, mas não contém close.
  c) str contém close, mas não contém open.
  d) str contém open e close.
  e) str contém open e close várias vezes.

Elaborando Casos de Testes

- Isso significa que deveríamos começar pela partição str = null e a combinaria com as partições das categorias open, close e (str, open, close).
- Terminaríamos com 4 × 4 × 4 × 5 = 320 testes.
- Escrever 320 testes pode ser um esforço que não valerá a pena.

Elaborando Casos de Testes

- Devemos, então, decidir pragmaticamente quais partições devem ser combinadas com outras e quais não devem.
- Uma primeira ideia para reduzir o número de testes é testar casos excepcionais apenas uma vez e não combiná-los.
- Por exemplo, a partição de str = null pode ser testada apenas uma vez e não mais do que isso.
  - O que ganharíamos ao combinar uma str nula com open sendo nulo, vazio, comprimento = 1 e comprimento > 1, bem como com close sendo nulo, vazio, comprimento = 1, comprimento > 1 e assim por diante? Não valeria a pena o esforço.
- O mesmo vale para uma string vazia: um teste pode ser bom o suficiente.
- Se aplicarmos a mesma lógica aos outros dois parâmetros e os testarmos como nulos e vazios apenas uma vez, já reduzimos drasticamente o número de casos de teste.

Elaborando Casos de Testes

- Pode haver outras partições que não precisam ser totalmente combinadas.
  - Para o caso da str de tamanho 1, dois testes podem ser suficientes: um em que o único caractere na string corresponde a open e close e outro em que não.
  - A menos que tenhamos uma boa razão para acreditar que o programa lida com tags de abertura e fechamento de diferentes comprimentos de maneiras diferentes, não precisamos das quatro combinações:
    - (tamanho de open = 1, tamanho de close = 1),
    - (tamanho de open > 1, tamanho de close = 1),
    - (tamanho de open = 1, tamanho de close > 1) e
    - (tamanho de open > 1, tamanho de close > 1).

    ✓Apenas (tamanho de open = 1, tamanho de close = 1) e (tamanho de open > 1, tamanho de close > 1) são suficientes.

Elaborando Casos de Testes

- Portanto, não devemos combinar partições cegamente, pois isso pode levar a casos de teste menos relevantes.
- Observar a implementação também pode ajudá-lo a reduzir o número de combinações.

Elaborando Casos de Testes

- Parâmetro `str`
  a) String nula
  b) String vazia
  c) String de tamanho 1
  d) String de tamanho > 1
- Parâmetro `open`
  a) String nula
  b) String vazia
  c) String de tamanho 1
  d) String de tamanho > 1
- Parâmetro `close`
  a) String nula
  b) String vazia
  c) String de tamanho 1
  d) String de tamanho > 1

- Parâmetros (`str`, `open`, `close`)
  a) str não contém nem open nem close.
  b) str contém open, mas não contém close.
  c) str contém close, mas não contém open.
  d) str contém open e close.
  e) str contém open e close várias vezes.

Elaborando Casos de Testes

- Os casos excepcionais:
  - T1: str é nula.
  - T2: str é vazia.
  - T3: open é nula.
  - T4: open é vazia.
  - T5: close é nula.
  - T6: close é vazia.

Elaborando Casos de Testes

- str de tamanho = 1
  - T7: O único caractere em str corresponde a open.
  - T8: O único caractere em str corresponde a close.
  - T9: O único caractere em str não corresponde nem a open nem a close
  - T10: O único caractere em str corresponde a open e a close.

Elaborando Casos de Testes

- Tamanho de str > 1, tamanho de open = 1, tamanho de close = 1
  - T11: str não contém open ou close.
  - T12: str contém open, mas não contém close.
  - T13: str contém a close, mas não contém close.
  - T14: str contém open e close.
  - T15: str contém open e close várias vezes.

Elaborando Casos de Testes

- Tamanho de str > 1, tamanho de open > 1, tamanho de close > 1
  - T16: str não contém open ou close.
  - T17: str contém open, mas não contém close.
  - T18: str contém a close, mas não contém close.
  - T19: str contém open e close.
  - T20: str contém open e close várias vezes.

Elaborando Casos de Testes

- Finalmente, o teste da fronteira
  - T21: str contém open e close sem caracteres entre elas.

Automatizando os Casos de Testes

```
import org.junit.jupiter.api.Test;
import static ch2.StringUtils.substringsBetween;
import static org.assertj.core.api.Assertions.assertThat;

public class StringUtilsTest {

  @Test
  void strIsNullOrEmpty() {
    assertThat(substringsBetween(null, "a", "b"))
      .isEqualTo(null);

    assertThat(substringsBetween("", "a", "b"))
      .isEqualTo(new String[]{});
  }

...
```

Automatizando os Casos de Testes

```
@Test
  void openIsNullOrEmpty() {
    assertThat(substringsBetween("abc", null, "b")).isEqualTo(null);
    assertThat(substringsBetween("abc", "", "b")).isEqualTo(null);
  }

  @Test
  void closeIsNullOrEmpty() {
    assertThat(substringsBetween("abc", "a", null)).isEqualTo(null);
    assertThat(substringsBetween("abc", "a", "")).isEqualTo(null);
  }
```

Automatizando os Casos de Testes

```
@Test
  void strOfLength1() {
    assertThat(substringsBetween("a", "a", "b")).isEqualTo(null);
    assertThat(substringsBetween("a", "b", "a")).isEqualTo(null);
    assertThat(substringsBetween("a", "b", "b")).isEqualTo(null);
    assertThat(substringsBetween("a", "a", "a")).isEqualTo(null);
  }
```

Automatizando os Casos de Testes

```
@Test
 void openAndCloseOfLength1() {
   assertThat(substringsBetween("abc", "x", "y")).isEqualTo(null);
   assertThat(substringsBetween("abc", "a", "y")).isEqualTo(null);
   assertThat(substringsBetween("abc", "x", "c")).isEqualTo(null);
   assertThat(substringsBetween("abc", "a", "c"))
     .isEqualTo(new String[] {"b"});
   assertThat(substringsBetween("abcabc", "a", "c"))
     .isEqualTo(new String[] {"b", "b"});
 }
```

Automatizando os Casos de Testes

```
@Test
 void openAndCloseTagsOfDifferentSizes() {
   assertThat(substringsBetween("aabcc", "xx", "yy")).isEqualTo(null);
   assertThat(substringsBetween("aabcc", "aa", "yy")).isEqualTo(null);
   assertThat(substringsBetween("aabcc", "xx", "cc")).isEqualTo(null);
   assertThat(substringsBetween("aabbcc", "aa", "cc"))
     .isEqualTo(new String[] {"bb"});
   assertThat(substringsBetween("aabbccaaeecc", "aa", "cc"))
     .isEqualTo(new String[] {"bb", "ee"});
 }
```

Automatizando os Casos de Testes

```
 @Test
void noSubstringBetweenOpenAndCloseTags() {
  assertThat(substringsBetween("aabb", "aa", "bb"))
    .isEqualTo(new String[] {""});
  }
}
```