

## Trabalho 8 – Grupo 12: Alexandre Monteiro - 51023 / João Afonso - 51111:

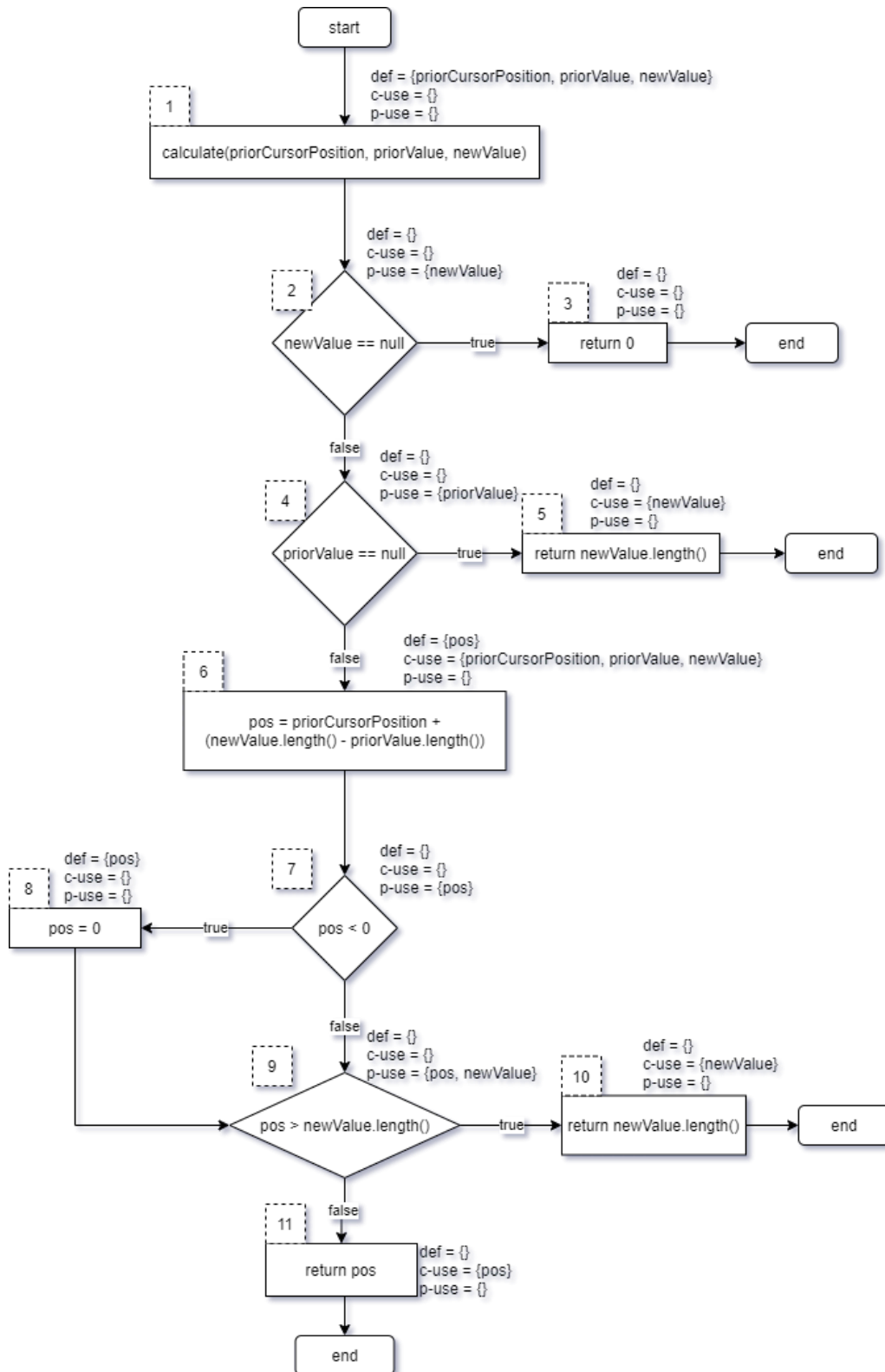
Justificação das funções escolhidas para testar e respetivo propósito:

Foram escolhidos os seguintes métodos:

- `calculate(int priorCursorPosition, String priorValue, String newValue)` da classe `CursorPositionCalculator.java`:
  - Este método recebe como input um inteiro “priorCursorPosition” e duas strings, “priorValue” e “newValue”, e devolve a posição calculada do cursor sobre a String “newValue”, no caso de o valor “priorValue” ser null então é devolvida a posição a seguir á “newValue”, e no caso de o “newValue” ser null então é devolvido 0 (zero). Esta função foi escolhida por apresentar um número razoável de variáveis e por conter if statements que podem influenciar o output.
- `join(Collection<?> s, String delimiter)` da classe `Util.java`:
  - Este método recebe como input uma coleção “s” e uma String “delimiter” e retorna uma String resultante da junção de todos os elementos da coleção, mas, entre cada um destes elementos, vai ser feito append do “delimiter”. Foi escolhida esta função porque, por um lado, apresenta um número razoável de variáveis (vai-se ter ao todo 4 variáveis) e, por outro, tem 2 if statements e um ciclo while, podendo surgir casos interessantes associados à forma como os dados são acedidos e modificados
- `fileName(String path)` da classe `Path.java`:
  - Dado um path para um ficheiro, retorna apenas o nome do ficheiro em causa. Escolheu-se este método porque, apesar de apresentar apenas 2 variáveis, a variável “path” é definida e usada (tanto em termos de c-use e p-use) várias vezes, tendo-se considerado que podia ser interessante ter uma função com menos variáveis, mas que essas variáveis fossem definidas e usadas um maior número de vezes.

## Dataflow Testing:

calculate:



Resumo tabular de cada uma das variáveis:

Variável– “priorCursorPosition”:

def-use pairs para a variável priorCursorPosition			
par id	def	use	path
1	1	6	<1,2,4,6>

Coverage criteria:

- all-defs: 1
- all-c-uses: 1
- all-p-uses: none
- all-uses: 1

Variável– “priorValue”:

def-use pairs para a variável priorValue			
par id	def	use	path
1	1	6	<1,2,4,6>
2	1	(4, T)	<1,2,4,5>
3	1	(4, F)	<1,2,4,6>

Coverage criteria:

- all-defs: 1
- all-c-uses: 1
- all-p-uses: 2, 3
- all-uses: 1, 2, 3

Variável– “newValue”:

def-use pairs para a variável newValue			
par id	def	use	path
1	1	5	<1,2,4,5>
2	1	6	<1,2,4,6>
3	1	10	<1,2,4,6,7,9,10>
4	1	10	<1,2,4,6,7,8,9,10>
5	1	(2, T)	<1,2,3>
6	1	(2, F)	<1,2,4>
7	1	(9, T)	<1,2,4,6,7,9,10>
8	1	(9, T)	<1,2,4,6,7,8,9,10>
9	1	(9, F)	<1,2,4,6,7,9,11>
10	1	(9, F)	<1,2,4,6,7,8,9,11>

Coverage criteria:

- all-defs: 5
- all-c-uses: 1, 2, 3
- all-p-uses: 5, 6, 7, 9
- all-uses: 1, 2, 3, 5, 6, 7, 9

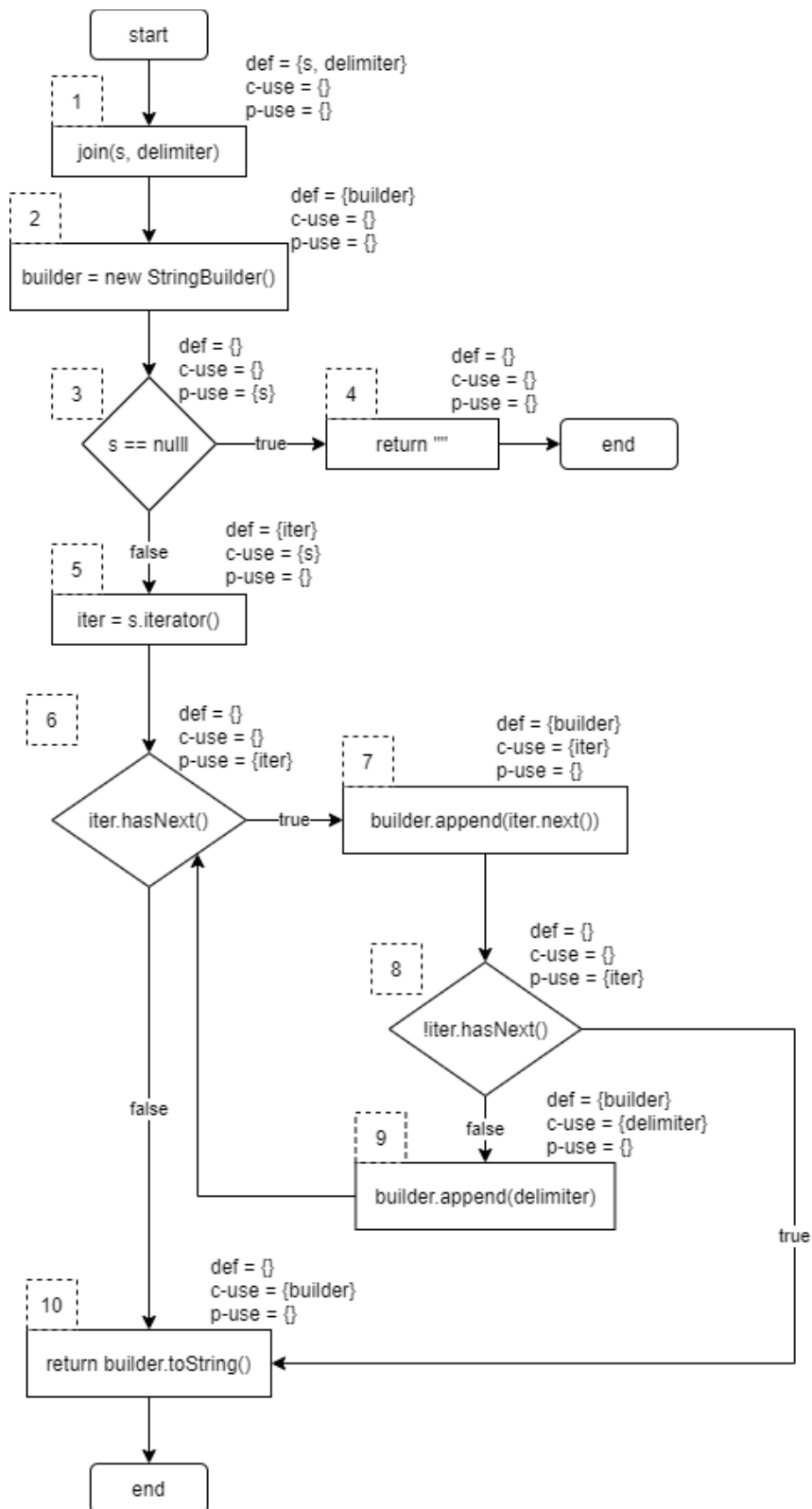
Variável– “pos”:

def-use pairs para a variável pos			
par id	def	use	path
1	6	11	<6,7,9,11>
2	6	(7, T)	<6,7,8>
3	6	(7, F)	<6,7,9>
4	6	(9, T)	<6,7,9,10>
5	6	(9, F)	<6,7,9,11>
6	8	11	<8,9,11>
7	8	(9, T)	<8,9,10>
8	8	(9, F)	<8,9,11>

Coverage criteria:

- all-defs: 2, 8
- all-c-uses: 1, 6
- all-p-uses: 2, 3, 4, 5, 7, 8
- all-uses: 1, 2, 3, 4, 5, 6, 7, 8

join:



Resumo tabular de cada uma das variáveis:

Variável– “s”:

def-use pairs para a variável s			
par id	def	use	path
1	1	5	<1,2,3,5>
2	1	(3, T)	<1,2,3,4>
3	1	(3, F)	<1,2,3,5>

Coverage criteria:

- all-defs: 1
- all-c-uses: 1
- all-p-uses: 2, 3
- all-uses: 1, 2, 3

Variável– “delimiter”:

def-use pairs para a variável delimiter			
par id	def	use	path
1	1	9	<1,2,3,5,6,7,8,9>

Coverage criteria:

- all-defs: 1
- all-c-uses: 1
- all-p-uses: none
- all-uses: 1

Variável– “builder”:

def-use pairs para a variável builder			
par id	def	use	path
1	2	10	<2,3,5,6,10>
2	7	10	<7,8,10>
3	9	10	<9,6,10>

Coverage criteria:

- all-defs: 1, 2, 3
- all-c-uses: 1, 2, 3
- all-p-uses: none
- all-uses: 1, 2, 3

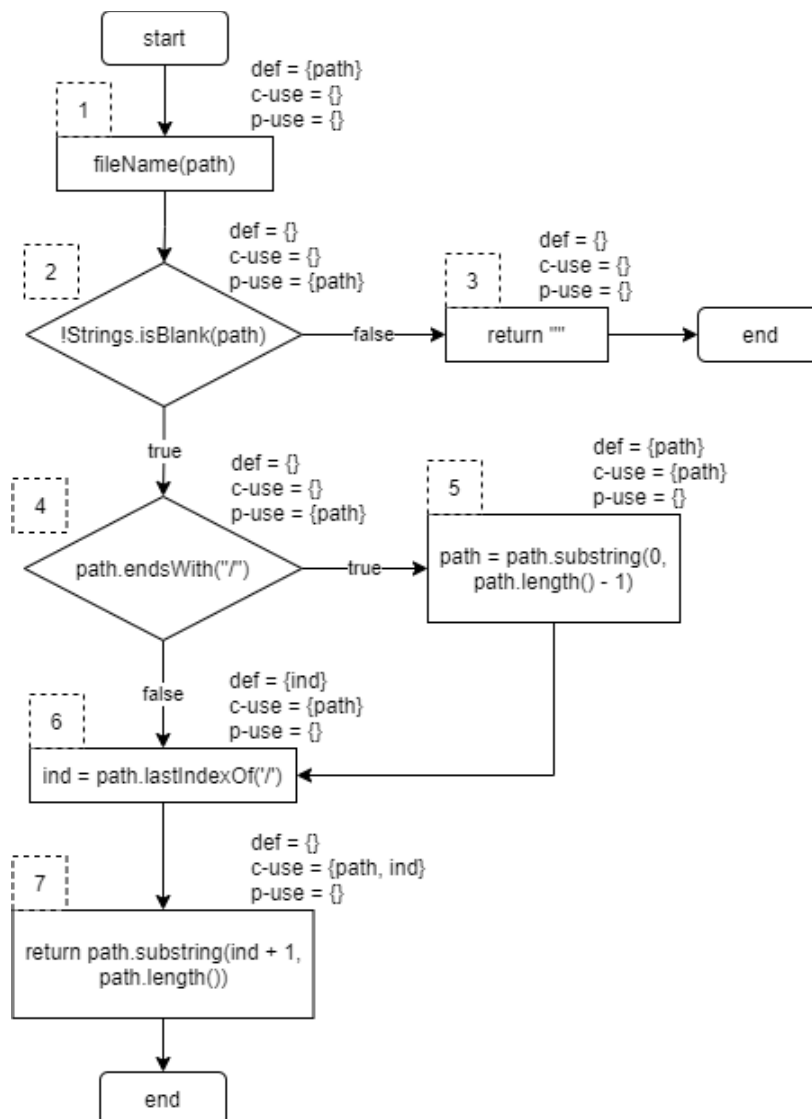
Variável– “iter”:

def-use pairs para a variável iter			
par id	def	use	path
1	5	7	<5,6,7>
2	5	(6, T)	<5,6,7>
3	5	(6, F)	<5,6,10>
4	5	(8, T)	<5,6,7,8,10>
5	5	(8, F)	<5,6,7,8,9>

Coverage criteria:

- all-defs: 1
- all-c-uses: 1
- all-p-uses: 2, 3, 4, 5
- all-uses: 1, 2, 3, 4, 5

fileName:



Resumo tabular de cada uma das variáveis:

Variável– “path”:

def-use pairs para a variável path			
par id	def	use	path
1	1	5	<1,2,4,5>
2	1	6	<1,2,4,6>
3	1	7	<1,2,4,6,7>
4	5	6	<5,6>
5	5	7	<5,6,7>
6	1	(2, T)	<1,2,4>
7	1	(2, F)	<1,2,3>
8	1	(4, T)	<1,2,4,5>
9	1	(4, F)	<1,2,4,6>

Coverage criteria:

- all-defs: 4, 6
- all-c-uses: 1, 2, 3, 4, 5
- all-p-uses: 6, 7, 8, 9
- all-uses: 1, 2, 3, 4, 5, 6, 7, 8, 9

Variável– “ind”:

def-use pairs para a variável ind			
par id	def	use	path
1	6	7	<6,7>

Coverage criteria:

- all-defs: 1
- all-c-uses: 1
- all-p-uses: none
- all-uses: 1



Testes unitários criados para cada uma das funções:

calculate:

```
@Test
public void testCalculateWithNewValueNull() {
    int priorCursorPosition = 1;
    String priorValue = "qwe";
    String newValue = null;
    int actual = CursorPositionCalculator.calculate(priorCursorPosition, priorValue, newValue);
    int expected = 0;

    assertEquals(expected, actual);
}

@Test
public void testCalculateWithPriorValueNull() {
    int priorCursorPosition = 1;
    String priorValue = null;
    String newValue = "qwerty";
    int actual = CursorPositionCalculator.calculate(priorCursorPosition, priorValue, newValue);
    int expected = 6;

    assertEquals(expected, actual);
}

@Test
public void testCalculateWithPosRedefiniton() {
    int priorCursorPosition = 0;
    String priorValue = "qwerty";
    String newValue = "qwe";
    int actual = CursorPositionCalculator.calculate(priorCursorPosition, priorValue, newValue);
    int expected = 0;

    assertEquals(expected, actual);
}

@Test
public void testCalculateWithPosWithinNewValueLimit() {
    int priorCursorPosition = 1;
    String priorValue = "qwe";
    String newValue = "qwerty";
    int actual = CursorPositionCalculator.calculate(priorCursorPosition, priorValue, newValue);
    int expected = 4;

    assertEquals(expected, actual);
}

@Test
public void testCalculateWithPosOverNewValueLimit() {
    int priorCursorPosition = 5;
    String priorValue = "qwe";
    String newValue = "qwerty";
    int actual = CursorPositionCalculator.calculate(priorCursorPosition, priorValue, newValue);
    int expected = 6;

    assertEquals(expected, actual);
}
```

Runs: 5/5

Errors: 0

Failures: 0

com.todotxt.todotxttouch.util.CursorPositionCalculatorDFTest [Runner: JUnit 4] ((

- testCalculateWithNewValueNull (0,001 s)
- testCalculateWithPosRedefiniton (0,000 s)
- testCalculateWithPriorValueNull (0,000 s)
- testCalculateWithPosOverNewValueLimit (0,000 s)
- testCalculateWithPosWithinNewValueLimit (0,000 s)

#### **testCalculateWithNewValueNull:**

Neste teste a variável newValue foi definida com null de modo a obter como output o valor 0. Obteve-se o resultado esperado.

Este teste engloba os def-use pairs:

1. Relativamente aos def-use pairs da variável "newValue", inclui o seguinte id: 5

Análise dos def-use pairs de acordo com a variável em causa:

1. O primeiro bloco condicional da função vai averiguar se newValue é null, como "newValue == null" dá true (tem-se o p-use (2, T)) então é devolvido 0 e a função termina.

#### **testCalculateWithPriorValueNull:**

Neste teste foi atribuído um valor não nulo a newValue e um valor nulo a priorValue de modo a ser devolvido o tamanho da string newValue. Obteve-se o resultado esperado.

Este teste engloba os def-use pairs:

1. Relativamente aos def-use pairs da variável "priorValue", inclui o seguinte id: 2
2. Relativamente aos def-use pairs da variável "newValue", inclui os seguintes ids: 1 e 6

Análise dos def-use pairs de acordo com a variável em causa:

1. O primeiro condicional a ser feito averigua se newValue é nulo, como é passado um valor não nulo então "newValue == null" é falso (tem-se o p-use (2, F)) e a função continua.
2. O segundo condicional averigua se priorValue é nulo, e como o valor é null então "priorValue == null" é verdadeiro (p-use em (4, T) de priorValue) e é devolvida o tamanho da string newValue (c-use em 5 de newValue).

#### **testCalculateWithPosRedefiniton:**

Neste teste foi atribuído o valor 0 a priorCursorPosition e foram atribuídos valores não nulos a priorValue e newValue de modo que a string priorValue seja maior que newValue obrigando a função a atribuir um valor negativo á variável pos para depois ser redefinido com um valor não negativo, neste caso 0, para ser então devolvido o valor 0. Obteve-se o resultado esperado. Observa-se também que uma vez que pos é redefinida com 0 então a condição "pos > newValue.length()" avalia sempre como falso uma vez que uma String não pode ter um comprimento negativo, sendo que o caminho <8, 9, 10> é impossível.

Este teste engloba os def-use pairs:

1. Relativamente aos def-use pairs da variável "pos", inclui os seguintes ids: 2, 6 e 8

Análise dos def-use pairs de acordo com a variável em causa:

1. Na definição de "pos" em 6 a equação atribui a esta variável um valor negativo de modo a condição "pos < 0" avaliar como true (p-use em (7, T) de pos) e redefinir pos com 0.
2. Com a definição de pos em 8 com o valor 0, a condição "pos > newValue.length()" vai devolver falso (p-use em (9, F)) e o valor de pos é devolvido pela função (c-use em 11).

#### **testCalculateWithPosWithinNewValueLimit:**

Neste teste foi atribuído o valor 1 a `priorCursorPosition` e valores não nulos a `priorValue` e `newValue` com tamanhos, respetivamente, 3 e 6 de modo que o cálculo da variável “pos” não fosse maior que o tamanho de `newValue` e fosse devolvido o valor de pos. Obteve-se o resultado esperado.

Este teste engloba os def-use pairs:

1. Relativamente aos def-use pairs da variável “`priorCursorPosition`”, inclui o seguinte id: 1
2. Relativamente aos def-use pairs da variável “`priorValue`”, inclui os seguintes ids: 1 e 3
3. Relativamente aos def-use pairs da variável “`newValue`”, inclui os seguintes ids: 2, 6 e 9
4. Relativamente aos def-use pairs da variável “`pos`”, inclui os seguintes ids: 1, 3 e 5

Análise dos def-use pairs de acordo com a variável em causa:

1. Como `priorValue` e `newValue` não são nulos então as duas primeiras condições avaliam a falso e a computação prossegue (p-use em (2, F) e (4, F)).
2. Para a definição de pos o seu valor é calculado somando a `priorCursorPosition` a diferença entre os tamanhos das strings `newValue` e `priorValue` (c-use em 6).
3. O valor de pos que foi calculado é positivo e então a condição “`pos < 0`” é avaliada como falso (p-use em 7).
4. Como `priorCursorPosition` é 1 e os tamanhos de `priorValue` e `newValue` são, respetivamente, 3 e 6 então pos tem como valor 4 ( $pos = 1 + (6 - 3)$ ). Como pos é menor que o tamanho de `newValue` então “`pos > newValue.length()`” é falso (p-use em (9, F)) e é devolvido o valor de pos (c-use em 11).

#### **testCalculateWithPosOverNewValueLimit:**

Neste teste foi atribuído o valor 5 a `priorCursorPosition` e valores não nulos a `priorValue` e `newValue` com tamanhos, respetivamente, 3 e 6 de modo que o cálculo da variável “pos” fosse maior que o tamanho de `newValue` e fosse devolvido o tamanho de `newValue`. Obteve-se o resultado esperado.

Este teste engloba os def-use pairs:

1. Relativamente aos def-use pairs da variável “`priorCursorPosition`”, inclui o seguinte id: 1
2. Relativamente aos def-use pairs da variável “`priorValue`”, inclui os seguintes ids: 1 e 3
3. Relativamente aos def-use pairs da variável “`newValue`”, inclui os seguintes ids: 2, 3, 6 e 7
4. Relativamente aos def-use pairs da variável “`pos`”, inclui os seguintes ids: 3 e 4

Análise dos def-use pairs de acordo com a variável em causa:



1. Como `priorValue` e `newValue` não são nulos então as duas primeiras condições avaliam a falso e a computação prossegue (p-use em (2, F) e (4, F)).
2. Para a definição de pos o seu valor é calculado somando a `priorCursorPosition` a diferença entre os tamanhos das strings `newValue` e `priorValue` (c-use em 6).
3. O valor de pos que foi calculado é positivo e então a condição “`pos < 0`” é avaliada como falso (p-use em 7).
4. Como `priorCursorPosition` é 5 e os tamanhos de `priorValue` e `newValue` são, respetivamente, 3 e 6 então pos tem como valor 8 ( $pos = 5 + (6 - 3)$ ). Como pos é maior que o tamanho de `newValue` então “`pos > newValue.length()`” é verdadeiro (p-use em (9, T)) e é devolvido o tamanho de `newValue` (c-use em 10).


join:




```
@Test
public void testJoinColWithThreeElements() {
    List<String> col = new ArrayList<>();
    col.add("one");
    col.add("two");
    col.add("three");
    String expected = "one#two#three";
    String actual = Util.join(col, "#");
    assertEquals(expected, actual);
}

@Test
public void testJoinNullCol() {
    List<String> col = null;
    String expected = "";
    String actual = Util.join(col, "#");
    assertEquals(expected, actual);
}

@Test
public void testJoinEmptyCol() {
    List<String> col = new ArrayList<>();
    String expected = "";
    String actual = Util.join(col, "#");
    assertEquals(expected, actual);
}
```

Runs: 3/3     Errors: 0     Failures: 0

✓  com.todobxt.todobxttouch.util.UtilJoinTest [Runner: JUnit 4] (0,010 s)

- ✓  testJoinColWithThreeElements (0,007 s)
- ✓  testJoinNullCol (0,001 s)
- ✓  testJoinEmptyCol (0,002 s)

### testJoinColWithThreeElements:

Neste teste, é criada uma lista (col) com os seguintes 3 elementos: "one", "two", "three". Vai ser feito "Util.join(col, "#")", ou seja, espera-se que seja retornada uma String que contém os 3 elementos da lista, pela mesma ordem em que foram inseridos na lista e separados por "#". Obteve-se o resultado esperado.

Este teste engloba os def-use pairs:

3. Relativamente aos def-use pairs da variável s, inclui os seguintes ids: 1 e 3
4. Relativamente aos def-use pairs da variável delimiter, inclui o seguinte id: 1
5. Relativamente aos def-use pairs da variável builder, inclui os seguintes ids: 1, 2 e 3
6. Relativamente aos def-use pairs da variável iter, inclui os seguintes ids: 1, 2, 4 e 5

Análise dos def-use pairs de acordo com a variável em causa:

1. Tem-se um if statement em que se averigua se a variável "s" é null, mas, como não é, não se executa o código contido no if (tem-se o p-use (3, F)). De seguida, é feito "Iterator<?> iter = s.iterator()", o que corresponde a outro uso de "s" (tem-se o c-use 5).
2. Só é feito um uso da variável "delimiter" quando se tem uma Collection "s" com pelo menos 2 elementos (uma vez que só é feito append "delimiter" entre 2 elementos de "iter"). A variável "s" corresponde a uma Collection com 3 elementos, portanto, como é feito "Iterator<?> iter = s.iterator()", tem-se que "while (iter.hasNext())" tem valor True. Dentro do ciclo, como "if (!iter.hasNext())" é False, ou seja, existe um outro elemento no Iterator, vai ser feito append do "delimiter" (tem-se o c-use 9).
3. A variável "builder" só tem um c-use na última linha do método em que é feito "builder.toString()". Esta variável corresponde a um StringBuilder, portanto, sempre que é feito um append, vai-se ter uma nova def de "builder". Esta variável é originalmente definida quando é feito "StringBuilder builder = new StringBuilder()" (tem-se def 1). Como foi passada uma Collection com 3 elementos, vai-se entrar dentro do ciclo while, fazendo-se "builder.append(iter.next())" (tem-se def 2). Como "if (!iter.hasNext())" é False, vai ser feito "builder.append(delimiter)" (tem-se o def 3). Por fim, vai ser feito um break, saindo-se do ciclo for e retornando-se "builder.toString()".
4. A variável "iter" é definida sempre que "s" é diferente de null e, como foi passado como parâmetro uma Collection com 3 elementos, sabe-se que vai ser definido "Iterator<?> iter = s.iterator()". Tendo em conta que "s" é diferente de null, tem-se que "while (iter.hasNext())" vai estar a True (tem-se o use (6, T)). Dentro do while, começa-se por fazer "builder.append(iter.next())" (tem-se o c-use 7) e, como "if (!iter.hasNext())" vai ter valor False (tem-se o p-use (8, F)), vai ser feita outra iteração do ciclo while. Na terceira iteração do ciclo while, "if (!iter.hasNext())" vai ter valor True (tem-se o p-use (8, T)) e é feito break, saindo-se do ciclo while.

**testJoinNullCol:**

Tal como esperado, foi retornada uma String vazia.

Este teste engloba os def-use pairs:

1. Relativamente aos def-use pairs da variável `s`, inclui o seguinte id: 2

Análise dos def-use pairs de acordo com a variável em causa:

1. Tem-se um if statement em que se averigua se a variável `s` é null. Como `s` tem valor null, `if (s == null)` vai ter valor True (tem-se o p-use (3, T)) e vai ser retornada uma String vazia.

**testJoinEmptyCol:**

Neste teste, inicializa-se uma lista `col`, mas nunca se adiciona qualquer elemento à lista, ou seja, vai simplesmente ser retornada uma lista vazia. Obteve-se o resultado esperado.

Este teste engloba os def-use pairs:

1. Relativamente aos def-use pairs da variável `iter`, inclui o seguinte id: 3

Análise dos def-use pairs de acordo com a variável em causa:



1. A única forma como se vai obter `while (iter.hasNext())` com valor False é se a Collection não tiver elementos porque, caso contrário, vai-se sair do ciclo for não porque a condição `iter.hasNext()` do while tem valor False, mas porque é feito um break dentro do ciclo. Como a Collection não tem qualquer elemento, `while (iter.hasNext())` tem valor false (tem-se o p-use (6, F)) e retorna-se `builder.toString()`.


fileName:




```
@Test
public void testFileNameWithPathNotBlank() {
    String path = "test/code/todo.java";
    String expected = "todo.java";
    String actual = Path.fileName(path);
    assertEquals(expected, actual);
}

@Test
public void testFileNameWithBackslashInTheEnd() {
    String path = "test/code/todo.java/";
    String expected = "todo.java";
    String actual = Path.fileName(path);
    assertEquals(expected, actual);
}

@Test
public void testParentPathWithPathBlank() {
    String expected = "";
    String actual = Path.fileName(" ");
    assertEquals(expected, actual);
}
```

Runs: 3/3     Errors: 0     Failures: 0

▼  com.todoxt.todoxttouch.util.PathFileNameTest [Runner: JUnit 4] (0,001 s)

-  testParentPathWithPathBlank (0,000 s)
-  testFileNameWithBackslashInTheEnd (0,000 s)
-  testFileNameWithPathNotBlank (0,000 s)

### **testFileNameWithPathNotBlank:**

Este teste engloba os def-use pairs:

1. Relativamente aos def-use pairs da variável path, inclui os seguintes ids: 2, 3, 6, 9
2. Relativamente aos def-use pairs da variável ind, inclui o seguinte id: 1

Análise dos def-use pairs de acordo com a variável em causa:

1. Se o path que é passado como parâmetro não terminar com o caracter "/", "if (!Strings.isBlank(path))" vai ter valor True (tem-se o p-use (2, T)) e "if (path.endsWith("/"))" vai ter valor False (tem-se o p-use (4, F)). Não sendo necessário remover o último caracter do path, é feito "int ind = path.lastIndexOf('/')" (tem-se o c-use 6) e, por fim, retorna-se a String "return path.substring(ind + 1, path.length())" (tem-se o c-use 7).
2. É feito "int ind = path.lastIndexOf('/')" para obter a posição da última ocorrência de "/" e, de seguida, é feito "return path.substring(ind + 1, path.length())", utilizando-se a variável "ind" para especificar a substring que se vai retornar (tem-se o c-use 7).

### **testFileNameWithBackslashInTheEnd:**

Neste teste, foi concebida uma String (path) cujo último caracter corresponde a "/": "test/code/todo.java/". Como tal, espera-se que o último "/" seja removido e que seja retomado apenas o todo.java (que se encontra no fim do path). Obteve-se o resultado esperado.

Este teste engloba os def-use pairs:

1. Relativamente aos def-use pairs da variável path, inclui os seguintes ids: 1, 4, 5, 8

Análise dos def-use pairs de acordo com a variável em causa:

1. Caso o path que é passado como parâmetro corresponda a uma String que termine em "/", então, para obter o nome do ficheiro, será necessário remover esse último caracter. Se a variável "path" corresponde a uma String cujo último caracter seja "/", então vai-se ter-se que "if (path.endsWith("/"))" tem valor True (tem-se o p-use (4, T)). De seguida, é feito "path = path.substring(0, path.length() - 1)" (tem-se o c-use 5 e o def 5). Saindo do if, é feito "int ind = path.lastIndexOf('/')" (tem-se o c-use 6) e, por fim, retorna-se a String "return path.substring(ind + 1, path.length())" (tem-se o c-use 7).

### **testParentPathWithPathBlank:**

Neste teste, decidiu-se que ia ser usada uma String composta exclusivamente por espaços. Deste modo, tal como era esperado, foi retornada uma String vazia.

Este teste engloba os def-use pairs:

1. Relativamente aos def-use pairs da variável path, inclui o seguinte id: 7

Análise dos def-use pairs de acordo com a variável em causa:

1. Caso o "path" passado como parâmetro corresponda a um null, empty ou blank, não se entra dentro do primeiro if. Tendo-se uma "path" com tais características, "if (!Strings.isBlank(path))" vai ter valor False (tem-se o p-use (2, F)) e é simplesmente retornado uma String vazia.



## Testes gerados para cada coverage criteria;

Os testes unitários representados previamente foram concebidos para cada coverage criteria da seguinte forma:

calculate:

- Var “priorCursorPosition”:
  - all-defs:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosOverNewValueLimit
  - all-c-uses:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosOverNewValueLimit
  - all-p-uses: none
  - all-uses:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosOverNewValueLimit
- Var “priorValue”:
  - all-defs:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosOverNewValueLimit
    - testCalculateWithPriorValueNull
  - all-c-uses:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosOverNewValueLimit
  - all-p-uses:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosOverNewValueLimit
    - testCalculateWithPriorValueNull
  - all-uses:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosOverNewValueLimit
    - testCalculateWithPriorValueNull
- Var “newValue”:
  - all-defs:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosOverNewValueLimit
    - testCalculateWithPriorValueNull
    - testCalculateWithNewValueNull
  - all-c-uses:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosOverNewValueLimit
    - testCalculateWithPriorValueNull
  - all-p-uses:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosOverNewValueLimit
    - testCalculateWithPriorValueNull
    - testCalculateWithNewValueNull
  - all-uses:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosOverNewValueLimit
    - testCalculateWithPriorValueNull

- testCalculateWithNewValueNull
- Var “pos”:
  - all-defs:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosOverNewValueLimit
    - testCalculateWithPosRedefinition
  - all-c-uses:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosRedefinition
  - all-p-uses:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosOverNewValueLimit
    - testCalculateWithPosRedefinition
  - all-uses:
    - testCalculateWithPosWithinNewValueLimit
    - testCalculateWithPosOverNewValueLimit
    - testCalculateWithPosRedefinition

join:

- Var “s”:
  - all-defs:
    - testJoinColWithThreeElements
  - all-c-uses:
    - testJoinColWithThreeElements
  - all-p-uses:
    - testJoinColWithThreeElements
    - testJoinNullCol
  - all-uses:
    - testJoinColWithThreeElements
    - testJoinNullCol
- Var “delimiter”:
  - all-defs:
    - testJoinColWithThreeElements
  - all-c-uses:
    - testJoinColWithThreeElements
  - all-p-uses: none
  - all-uses:
    - testJoinColWithThreeElements
- Var “builder”:
  - all-defs:
    - testJoinColWithThreeElements
  - all-c-uses:
    - testJoinColWithThreeElements
  - all-p-uses: none
  - all-uses:
    - testJoinColWithThreeElements
- Var “iter”:
  - all-defs:
    - testJoinColWithThreeElements

- all-c-uses:
  - testJoinColWithThreeElements
- all-p-uses:
  - testJoinColWithThreeElements
  - testJoinNullCol
- all-uses:
  - testJoinColWithThreeElements
  - testJoinNullCol

fileName:

- Var “path”:
  - all-defs:
    - testFileNameWithPathNotBlank
    - testFileNameWithBackslashInTheEnd
  - all-c-uses:
    - testFileNameWithPathNotBlank
    - testFileNameWithBackslashInTheEnd
  - all-p-uses:
    - testFileNameWithPathNotBlank
    - testFileNameWithBackslashInTheEnd
    - testParentPathWithPathBlank
  - all-uses:
    - testFileNameWithPathNotBlank
    - testFileNameWithBackslashInTheEnd
    - testParentPathWithPathBlank:
- Var “ind”:
  - all-defs:
    - testFileNameWithPathNotBlank
  - all-c-uses:
    - testFileNameWithPathNotBlank
  - all-p-uses: none
  - all-uses:
    - testFileNameWithPathNotBlank