

Trabalho 3 – Grupo 12: Alexandre Monteiro - 51023 / João Afonso - 51111:

Escolha das funções, propósito e justificação:

Tendo em consideração que o objetivo do trabalho consistia na realização de black-box testing, optou-se por escolher as funções que estavam associadas com alguma fonte de conhecimento (apresentavam documentação) ou cujo comportamento era perceptível. Deste modo, foi possível testar o software sem conhecer a implementação, design e estrutura interna do mesmo.

Tendo isto em consideração, foram escolhidas as seguintes funções:

1. Package – com.todotxt.todotxttouch.util / Classe – CursorPositionCalculator.java / Método – calculate (int priorCursorPosition, String priorValue, String newValue):
 - **Propósito:** Usada para calcular a nova posição do cursor quando uma string é alterada (face à posição inicial do cursor na string antes da mudança)
 - **Justificação:** Como existe um ponteiro indicativo da posição do cursor na string priorValue, vai-se ter 3 partições (priorCursorPosition pode: corresponder a uma posição válida de string / ter um valor negativo / ser superior ao tamanho da string), sendo importante averiguar se o comportamento da função é adequado nestas partições e respetivas fronteiras. Também se deve certificar se o software realmente tem o comportamento descrito na documentação quando priorValue ou newValue têm valor null
2. Package – com.todotxt.todotxttouch.util / Classe – Strings.java / Método – insertPadded (String s, int insertAt, String stringToInsert):
 - **Propósito:** Usada para inserir uma palavra numa dada posição de uma string (a palavra inserida vai estar sempre entre espaços, podendo-se adicionar um espaço antes e/ou depois caso não haja uma separação com o texto já escrito)
 - **Justificação:** Tem-se um ponteiro, insertAt, que indica a posição da string s onde se quer inserir stringToInsert, ou seja, tem-se 3 partições (e 2 fronteiras) que podem ser interessantes de analisar:
 - insertAt corresponde a uma posição válida de s ($0 \leq \text{insertAt} \leq \text{s.length}$)
 - insertAt tem um valor negativo ($\text{insertAt} < 0$)
 - insertAt superior ao tamanho de s ($\text{insertAt} > \text{s.length}$)
3. Package – com.chschmid.jdotxt / Classe – Jdotxt.java / Método – insertReplaceString (String original, String replace, int offset):
 - **Propósito:** Usada para substituir parte de uma string (original) por outra string. Tem-se um offset que indica a sequência inicial de caracteres da string original que não se quer substituir. Se o número de caracteres resultantes da junção de offset caracteres de original com replace não exceder o tamanho de original, substitui-se essa sequência e o resto de original permanece inalterado.
 - **Justificação:** Existe um offset correspondente ao número de caracteres iniciais de original que se quer manter antes da substituição, ou seja, vai-se ter 3 partições (semelhante aos 2 casos anteriores). Também é necessário ter em consideração que o comportamento da função difere caso original.length() seja superior a replace.length() + offset (original não é substituído até ao fim)

Boundary Value Analysis:

1. Package – com.todo.txt.todo.txttouch.util / Classe – CursorPositionCalculator.java / Método – calculate (int priorCursorPosition, String priorValue, String newValue):

- 1.1. Partições:

- 1.1.1. priorCursorPosition < 0 → o cursor tem um valor negativo
- 1.1.2. 0 <= priorCursorPosition <= priorValue.length() - 1 → o cursor aponta para uma posição de priorValue (priorCursorPosition depende do tamanho de priorValue)
- 1.1.3. priorCursorPosition > priorValue.length() - 1 → o cursor tem um valor superior ao tamanho de priorValue (priorCursorPosition depende do tamanho de priorValue)
- 1.1.4. priorValue != null → priorValue corresponde a uma string com qualquer tamanho (por default, assume-se que, a menos que seja claramente especificado que priorValue = null, então todos os testes vão ter priorValue != null)
- 1.1.5. priorValue == null → se priorValue = null, a posição calculada corresponde à posição logo após a string
- 1.1.6. newValue != null → newValue corresponde a uma string com qualquer tamanho (por default, assume-se que, a menos que seja claramente especificado que newValue = null, então todos os testes vão ter newValue != null)
- 1.1.7. newValue == null → se newValue = null, a posição calculada corresponde a zero

- 1.2. Fronteiras (on-point):

- 1.2.1. priorCursorPosition == 0
- 1.2.2. priorCursorPosition == priorValue.length() - 1
- 1.2.3. priorValue == null
- 1.2.4. newValue == null

- 1.3. Casos de teste:

- 1.3.1. priorCursorPosition = 0 → on-point / fronteira: 1.2.1 / partição: 1.1.2
- 1.3.2. priorCursorPosition = -1 → off-point / fronteira: 1.2.1 / partição: 1.1.1
- 1.3.3. priorCursorPosition = priorValue.length() - 1 → on-point / fronteira: 1.2.1 / partição: 1.1.1
- 1.3.4. priorCursorPosition = priorValue.length() → off-point / fronteira: 1.2.1 / partição: 1.1.3
- 1.3.5. priorCursorPosition = -n → valor dentro da partição: 1.1.1
- 1.3.6. priorCursorPosition = priorValue.length() - n → valor dentro da partição: 1.1.2
- 1.3.7. priorCursorPosition = priorValue.length() + n → valor dentro da partição: 1.1.3
- 1.3.8. priorValue = null → on-point / fronteira: 1.2.3 / partição: 1.1.4
- 1.3.9. newValue = null → on-point / fronteira: 1.2.4 / partição: 1.1.6

2. Package – com.todoxt.todoxttouch.util / Classe – Strings.java / Método – insertPadded (String s, int insertAt, String stringToInsert):
 - 2.1. Partições:
 - 2.1.1. $0 > \text{insertAt}$ → inserir a string numa posição negativa
 - 2.1.2. $0 \leq \text{insertAt} \leq \text{s.length}()$ → inserir a string numa posição válida de s
 - 2.1.3. $\text{insertAt} > \text{s.length}()$ → inserir a string numa posição maior do que s
 - 2.2. Fronteiras:
 - 2.2.1. $\text{insertAt} == 0$
 - 2.2.2. $\text{insertAt} == \text{s.length}()$
 - 2.3. Casos de teste:
 - 2.3.1. $\text{insertAt} = 0$ → on-point / fronteira: 2.2.1 / partição: 2.1.2
 - 2.3.2. $\text{insertAt} = -1$ → off-point / fronteira: 2.2.1 / partição: 2.1.1
 - 2.3.3. $\text{insertAt} = \text{s.length}()$ → on-point / fronteira: 2.2.2 / partição: 2.1.2
 - 2.3.4. $\text{insertAt} = \text{s.length}() + 1$ → off-point / fronteira: 2.2.2 / partição: 2.1.3
 - 2.3.5. $\text{insertAt} = -n$ → valor dentro da partição: 2.1.1
 - 2.3.6. $\text{insertAt} = \text{s.length}() - n$ → valor dentro da partição: 2.1.2
 - 2.3.7. $\text{insertAt} = \text{s.length}() + n$ → valor dentro da partição: 2.1.3

3. Package – com.chschmid.jdotxt / Classe – Jdotxt.java / Método – insertReplaceString (String original, String replace, int offset):

3.1. Partições:

- 3.1.1. $0 > \text{offset} \rightarrow$ offset tem valor negativo
- 3.1.2. $0 \leq \text{offset} \leq \text{original.length()} - 1 \rightarrow$ offset aponta para uma posição contida dentro da string original
- 3.1.3. $\text{offset} > \text{original.length()} - 1 \rightarrow$ offset é maior do que a string original
- 3.1.4. $\text{original.length()} \leq \text{replace.length()} + \text{offset} \rightarrow$ a string original não é substituída até ao fim
- 3.1.5. $\text{original.length()} > \text{replace.length()} + \text{offset} \rightarrow$ a string original é substituída a partir de um dado e até ao fim
- 3.1.6. $0 \leq \text{offset} \leq \text{original.length()} - 1 \ \&\& \ \text{original.length()} \leq \text{replace.length()} + \text{offset} \rightarrow$ offset aponta para uma posição contida dentro da string original e, por sua vez, esta não vai ser substituída até ao fim
- 3.1.7. $0 \leq \text{offset} \leq \text{original.length()} - 1 \ \&\& \ \text{original.length()} > \text{replace.length()} + \text{offset} \rightarrow$ offset aponta para uma posição contida dentro da string original e, por sua vez, esta é substituída a partir de um dado ponto e até ao fim

3.2. Fronteiras:

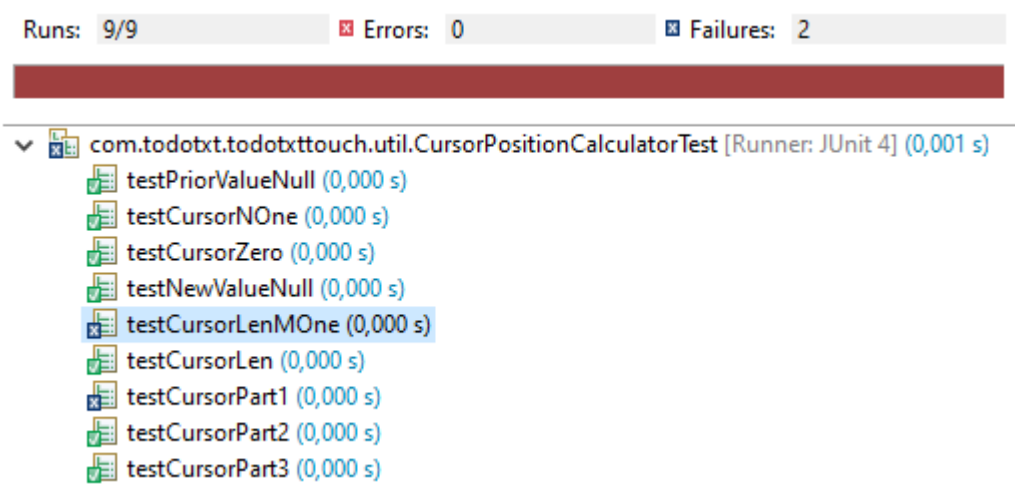
- 3.2.1. $\text{offset} == 0$
- 3.2.2. $\text{offset} == \text{original.length} - 1$
- 3.2.3. $\text{original.length()} == \text{replace.length()} + \text{offset}$
- 3.2.4. $\text{offset} == 0 \ \&\& \ \text{original.length()} == \text{replace.length()} + \text{offset}$
- 3.2.5. $\text{offset} == \text{original.length} - 1 \ \&\& \ \text{original.length()} == \text{replace.length()} + \text{offset}$

3.3. Casos de teste:

- 3.3.1. $\text{offset} = 0 \ \&\& \ \text{original.length()} > \text{offset} + \text{replace.length()} \rightarrow$ on-point / fronteira: 3.2.4 / partição: 3.1.7
- 3.3.2. $\text{offset} = 0 \ \&\& \ \text{original.length()} < \text{offset} + \text{replace.length()} \rightarrow$ on-point / fronteira: 3.2.4 / partição: 3.1.6
- 3.3.3. $\text{offset} = 0 \ \&\& \ \text{original.length()} = \text{offset} + \text{replace.length()} \rightarrow$ on-point / fronteira: 3.2.4 / partição: 3.1.6
- 3.3.4. $\text{offset} = \text{original.length()} - 1 \ \&\& \ \text{original.length()} > \text{offset} + \text{replace.length()} \rightarrow$ on-point / fronteira: 3.2.5 / partição: 3.1.7
- 3.3.5. $\text{offset} = \text{original.length()} - 1 \ \&\& \ \text{original.length()} < \text{offset} + \text{replace.length()} \rightarrow$ on-point / fronteira: 3.2.5 / partição: 3.1.6
- 3.3.6. $\text{offset} = \text{original.length()} - 1 \ \&\& \ \text{original.length()} = \text{offset} + \text{replace.length()} \rightarrow$ on-point / fronteira: 3.2.5 / partição: 3.1.6
- 3.3.7. $\text{offset} = -1 \rightarrow$ off-point / fronteira: 3.2.1 / partição: 3.1.1
- 3.3.8. $\text{offset} = \text{original.length()} \rightarrow$ off-point / fronteira: 3.2.2 / partição: 3.1.3
- 3.3.9. $\text{offset} = -n \rightarrow$ valor dentro da partição: 3.1.1
- 3.3.10. $\text{offset} = \text{original.length()} - n \ \&\& \ \text{original.length()} > \text{offset} + \text{replace.length()} \rightarrow$ valor dentro da partição: 3.1.7
- 3.3.11. $\text{offset} = \text{original.length()} - n \ \&\& \ \text{original.length()} < \text{offset} + \text{replace.length()} \rightarrow$ valor dentro da partição: 3.1.6
- 3.3.12. $\text{offset} = \text{original.length()} - 1 \ \&\& \ \text{original.length()} = \text{offset} + \text{replace.length()} \rightarrow$ on-point / fronteira: 3.2.5 / partição: 3.1.6
- 3.3.13. $\text{offset} = \text{original.length()} + n \rightarrow$ valor dentro da partição: 3.1.3

Testes e resultados:

1. Package – com.todotxt.todotxttouch.util / Classe – CursorPositionCalculator.java / Método – calculate (int priorCursorPosition, String priorValue, String newValue):

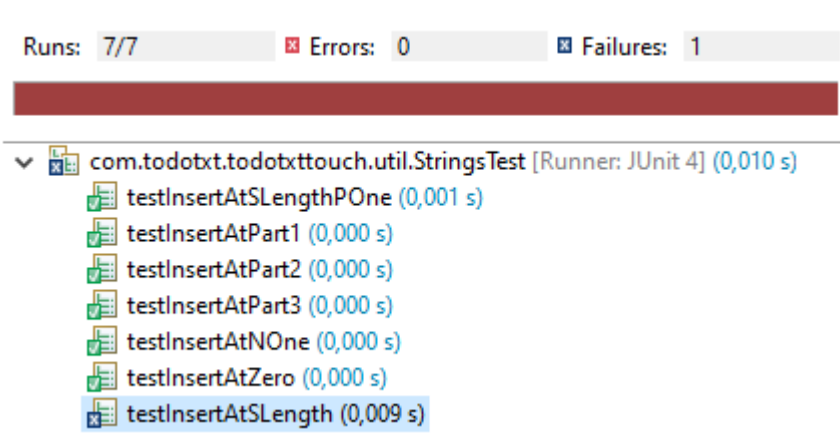


Descrição dos testes e resultados (para todos estes casos vai-se usar as variáveis newValue = “querty” e priorValue = “qwe”):

- testCursorZero:
 - Define-se int priorCursorPosition = 0
 - O resultado esperado (expected) corresponde a 3
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testCursorNone:
 - Define-se int priorCursorPosition = 2
 - O resultado esperado (expected) corresponde a 5
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testCursorLenMOne:
 - Define-se int priorCursorPosition = -1
 - O resultado esperado (expected) corresponde a 3
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se um valor indesejado
 - O actual value obtido foi 2, o que permite concluir que, apesar de priorCursorPosition apontar para uma posição supostamente inválida (porque o cursor não poderia possivelmente estar numa posição negativa de uma string), o cálculo é feito utilizando este valor inválido, concluindo-se que isto corresponde a uma falha
- testCursorLen:
 - Define-se int priorCursorPosition = 3
 - O resultado esperado (expected) corresponde a 6
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testCursorPart1:
 - Define-se int priorCursorPosition = -3
 - O resultado esperado (expected) corresponde a 3

- Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se um valor indesejado
- O actual value obtido foi 0, o que permite concluir que, apesar de priorCursorPosition apontar para uma posição supostamente inválida (porque o cursor não poderia possivelmente estar numa posição negativa de uma string), o cálculo é feito utilizando este valor inválido, concluindo-se que isto corresponde a uma falha
- testCursorPart2:
 - Define-se int priorCursorPosition = 1
 - O resultado esperado (expected) corresponde a 4
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testCursorPart3:
 - Define-se int priorCursorPosition = 5
 - O resultado esperado (expected) corresponde a 6
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testPriorValueNull:
 - Define-se int priorCursorPosition = 1
 - O resultado esperado (expected) corresponde a 6
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testNewValueNull:
 - Define-se int priorCursorPosition = 1
 - O resultado esperado (expected) corresponde a 0
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado

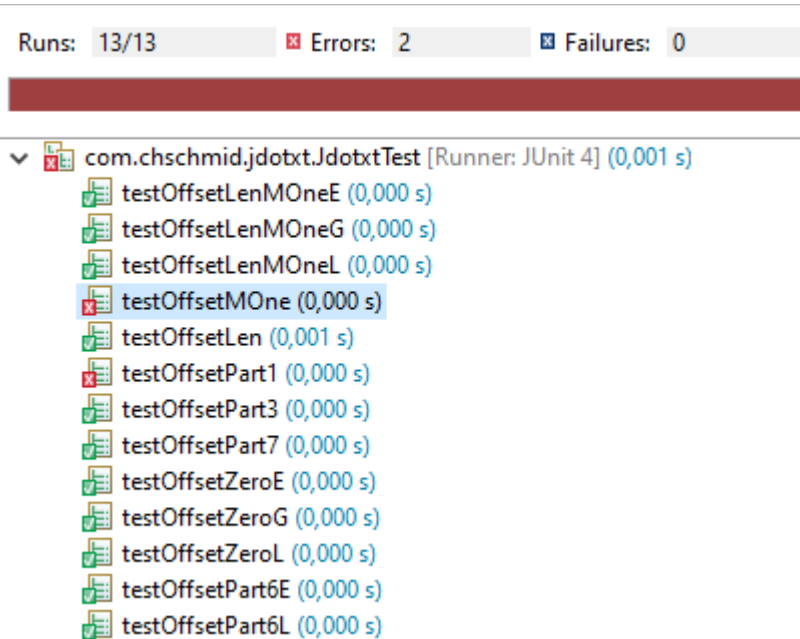
2. Package – com.todotxt.todotxttouch.util / Classe – Strings.java / Método – insertPadded (String s, int insertAt, String stringToInsert):



Descrição dos testes e resultados (para todos estes casos vai-se usar as variáveis `s = "qwerty"` e `stringToInsert = "abc"`):

- `testInsertAtZero`:
 - Define-se `int insertAt = 0`
 - O resultado esperado (expected) corresponde a `"abcqwerty"`
 - Calculou-se o valor verdadeiro (actual) e fez-se `assertEquals(expected, actual)`, obtendo-se o valor esperado
- `testInsertAtNOne`:
 - Define-se `int insertAt = -1`
 - De acordo com a documentação (`insertAt < 0`), deve de ocorrer um `IndexOutOfBoundsException`
 - Chamou-se a função e, tal como esperado, ocorreu um `IndexOutOfBoundsException`
- `testInsertAtSLength`:
 - Define-se `int insertAt = 6`
 - O resultado esperado (expected) corresponde a `"qwerty abc"`
 - Calculou-se o valor verdadeiro (actual) e fez-se `assertEquals(expected, actual)`, obtendo-se um valor indesejado
 - O actual value obtido foi `"qwerty abc "` tendo-se acrescentado um espaço no fim (sem haver uma justificação para tal estar a acontecer) concluindo-se que isto corresponde a uma falha
- `testInsertAtSLengthPOne`:
 - Define-se `int insertAt = 7`
 - De acordo com a documentação (`insertAt` maior do que o tamanho de `s`), deve de ocorrer um `IndexOutOfBoundsException`
 - Chamou-se a função e, tal como esperado, ocorreu um `IndexOutOfBoundsException`
- `testInsertAtPart1`:
 - Define-se `int insertAt = -5`
 - De acordo com a documentação (`insertAt < 0`), deve de ocorrer um `IndexOutOfBoundsException`
 - Chamou-se a função e, tal como esperado, ocorreu um `IndexOutOfBoundsException`
- `testInsertAtPart2`:
 - Define-se `int insertAt = 3`
 - O resultado esperado (expected) corresponde a `"qwe abc rty"`
 - Calculou-se o valor verdadeiro (actual) e fez-se `assertEquals(expected, actual)`, obtendo-se o valor esperado
- `testInsertAtPart3`:
 - Define-se `int insertAt = 9`
 - De acordo com a documentação (`insertAt` maior do que o tamanho de `s`), deve de ocorrer um `IndexOutOfBoundsException`
 - Chamou-se a função e, tal como esperado, ocorreu um `IndexOutOfBoundsException`

3. Package – com.todotxt.todotxttouch.util / Classe – Strings.java / Método – insertPadded (String s, int insertAt, String stringToInsert):



Descrição dos testes e resultados:

- testOffsetZeroG:
 - Define-se int offset = 0 e as strings original = "qwerty" e replace = "man"
 - O resultado esperado (expected) corresponde a "manrty"
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testOffsetZeroL:
 - Define-se int offset = 0 e as strings original = "qwerty" e replace = " abcdefgh"
 - O resultado esperado (expected) corresponde a " abcdefgh"
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testOffsetZeroE:
 - Define-se int offset = 0 e as strings original = "qwerty" e replace = " abcdef "
 - O resultado esperado (expected) corresponde a " abcdef "
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testOffsetLenMOneG:
 - Define-se int offset = 5 e as strings original = "qwerty" e replace = ""
 - O resultado esperado (expected) corresponde a " qwerty "
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testOffsetLenMOneL:
 - Define-se int offset = 5 e as strings original = "qwerty" e replace = "man"
 - O resultado esperado (expected) corresponde a " qwertman"
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado

- testOffsetLenMOneE:
 - Define-se int offset = 3 e as strings original = "qwerty" e replace = "man"
 - O resultado esperado (expected) corresponde a "qweman"
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testOffsetMOne:
 - Define-se int offset = -1 e as strings original = "qwerty" e replace = "man"
 - O resultado esperado (expected) corresponde a "qwertyman"
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), ocorrendo um StringIndexOutOfBoundsException, concluindo-se que se tem uma falha
- testOffsetLen:
 - Define-se int offset = 6 e as strings original = "qwerty" e replace = "man"
 - O resultado esperado (expected) corresponde a "manrty"
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testOffsetPart1:
 - Define-se int offset = -1 e as strings original = "qwerty" e replace = "man"
 - O resultado esperado (expected) corresponde a "manrty"
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), ocorrendo um StringIndexOutOfBoundsException, concluindo-se que se tem uma falha
- testOffsetPart6L:
 - Define-se int offset = 2 e as strings original = "qwerty" e replace = "man"
 - O resultado esperado (expected) corresponde a "qwmany"
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testOffsetPart6E:
 - Define-se int offset = 2 e as strings original = "qwerty" e replace = "mens"
 - O resultado esperado (expected) corresponde a "qwmens"
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testOffsetPart7:
 - Define-se int offset = 3 e as strings original = "qwerty" e replace = "mens"
 - O resultado esperado (expected) corresponde a "qwemens"
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado
- testOffsetPart3:
 - Define-se int offset = 10 e as strings original = "qwerty" e replace = "man"
 - O resultado esperado (expected) corresponde a "qwertyman"
 - Calculou-se o valor verdadeiro (actual) e fez-se assertEquals(expected, actual), obtendo-se o valor esperado