

Trabalho 5 – Grupo 12: Alexandre Monteiro - 51023 / João Afonso - 51111:

Line and branch coverage das tarefas 2:

Branch coverage:

A seguinte imagem representa a branch coverage dos métodos da classe “Strings.java”:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructio...
▼ Strings.java	96,4 %	188	7	195
▼ Strings	96,4 %	188	7	195
insertPadded(String, int,	97,8 %	88	2	90
insertPaddedIfNeeded(S	97,6 %	80	2	82
isBlank(String)	100,0 %	11	0	11
isEmptyOrNull(String)	100,0 %	9	0	9

1. Classe – Strings.java / Método – insertPaddedIfNeeded(String s, int insertAt, String stringToInsert):

```
public static String insertPaddedIfNeeded(String s, int insertAt,
    String stringToInsert) {
    if (Strings.isEmptyOrNull(stringToInsert)) {
        return s;
    }

    boolean found = false;
    int startPos = 0;

    while ((startPos < s.length()) && (! found))
    {
        int pos = s.indexOf(stringToInsert, startPos);

        if (pos < 0)
            break;

        startPos = pos + 1;
        int before = pos - 1;
        int after = pos + stringToInsert.length();

        if (((pos == 0) || (Character.isWhitespace(s.charAt(before)))) &&
            ((after >= s.length()) || (Character.isWhitespace(s.charAt(after)))))
            found = true;
    }

    if (found)
    {
        StringBuilder newText = new StringBuilder(s);

        if (newText.lastIndexOf(SINGLE_SPACE) != newText.length() - 1) {
            newText.append(SINGLE_SPACE);
        }

        return( newText.toString() );
    }
    else
        return( Strings.insertPadded(s, insertAt, stringToInsert) );
}
```

2. Classe – Strings.java / Método – insertPadded (Strings s, int insertAt, String stringToInsert):

```
public static String insertPadded(String s, int insertAt,
    String stringToInsert) {
    if (Strings.isEmptyOrNull(stringToInsert)) {
        return s;
    }

    if (insertAt < 0) {
        throw new IndexOutOfBoundsException("Invalid insertAt of ["
            + insertAt + "] for string [" + s + "]");
    }

    StringBuilder newText = new StringBuilder();
    if (insertAt > 0) {
        newText.append(s.substring(0, insertAt));
        if (newText.lastIndexOf(SINGLE_SPACE) != newText.length() - 1) {
            newText.append(SINGLE_SPACE);
        }
        newText.append(stringToInsert);
        String postItem = s.substring(insertAt);
        if (postItem.indexOf(SINGLE_SPACE) != 0) {
            newText.append(SINGLE_SPACE);
        }
        newText.append(postItem);
    } else {
        newText.append(stringToInsert);
        if (s.indexOf(SINGLE_SPACE) != 0) {
            newText.append(SINGLE_SPACE);
        }
    }
    newText.append(s);
}
return newText.toString();
}
```

3. Classe – Strings.java / Método – isBlank (Strings s):

```
public static boolean isEmptyOrNull(String s) {
    return s == null || s.length() == 0;
}
```

4. Classe – Strings.java / Método – isEmptyOrNull (Strings s):

```
public static boolean isBlank(String s) {
    return isEmptyOrNull(s) || s.trim().length() == 0;
}
```

5. Classe – CursorPositionCalculator.java / Método – calculate(int priorCursorPosition, String priorValue, String newValue):

A seguinte imagem representa a branch coverage dos métodos da classe “CursorPositionCalculator.java”:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructio...
CursorPositionCalculator.java	91,4 %	32	3	35
CursorPositionCalculator	91,4 %	32	3	35
calculate(int, String, Stri	100,0 %	32	0	32

```
public static final int calculate(int priorCursorPosition,
    String priorValue, String newValue) {
    if (newValue == null) {
        return 0;
    }

    if (priorValue == null) {
        return newValue.length();
    }

    int pos = priorCursorPosition
        + (newValue.length() - priorValue.length());
    pos = pos < 0 ? 0 : pos;
    return pos > newValue.length() ? newValue.length() : pos;
}
```

Line coverage:

A seguinte imagem representa a line coverage dos métodos da classe “Strings.java”:

Strings.java	93,2 %	41	3	44
Strings	93,2 %	41	3	44
insertPadded(String, int, Strin	95,0 %	19	1	20
insertPaddedIfNeeded(String	95,2 %	20	1	21
isBlank(String)	100,0 %	1	0	1
isEmptyOrNull(String)	100,0 %	1	0	1

1. Classe – Strings.java / Método – insertPaddedIfNeeded(String s, int insertAt, String stringToInsert)
2. Classe – Strings.java / Método – insertPadded (Strings s, int insertAt, String stringToInsert)
3. Classe – Strings.java / Método – isBlank (String s)
4. Classe – Strings.java / Método – isEmptyOrNull (String s)

5. Classe – CursorPositionCalculator.java / Método – calculate(int priorCursorPosition, String priorValue, String newValue):

A seguinte imagem representa a line coverage dos métodos da classe “CursorPositionCalculator.java”:

Element	Coverage	Covered Lines	Missed Lines	Total Lines
> com.todotxt.todotxttouch.task.sorter	0,0 %	0	89	89
▼ com.todotxt.todotxttouch.util	2,9 %	8	270	278
▼ CursorPositionCalculator.java	88,9 %	8	1	9
▼ CursorPositionCalculator	88,9 %	8	1	9
calculate(int, String, String)	100,0 %	8	0	8
> Path.java	0,0 %	0	13	13
▼ RelativeData.java	0,0 %	0	25	25

Line and branch coverage das tarefas 3:

Branch coverage:

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

A seguinte imagem representa a branch coverage do método da classe “Jdotxt.java”:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
insertReplaceString(String original, String replace, int offset)	100,0 %	38	0	38

```

public static String insertReplaceString(String original, String replace, int offset) {
    String a = original.substring(0, Math.min(offset, original.length()));
    String b;
    if (original.length() > (offset + replace.length())) b = original.substring(offset + replace.length(), original.length());
    else b = "";
    return a + replace + b;
}

```

Line coverage:

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

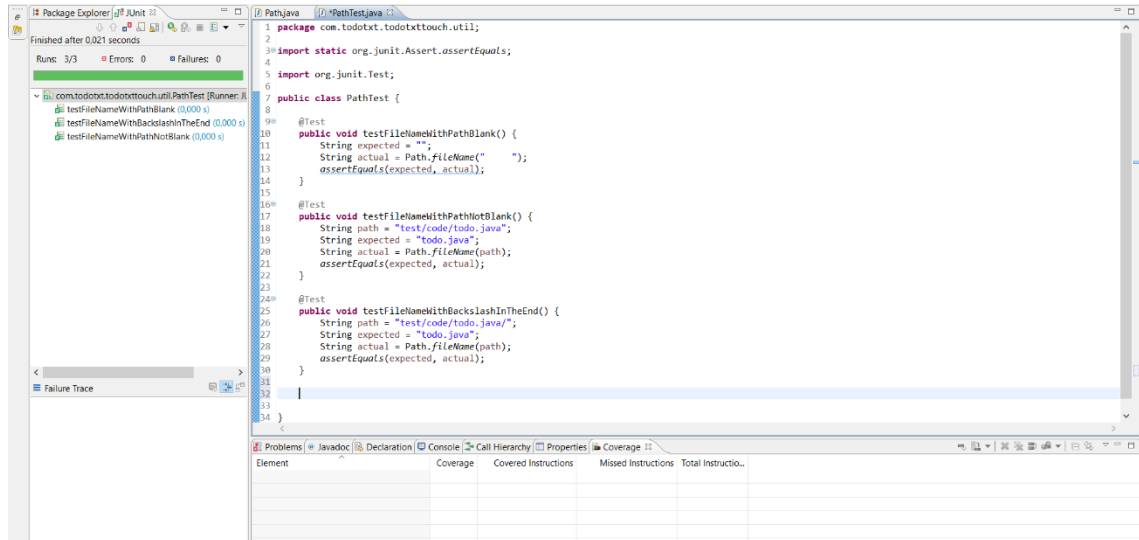
A seguinte imagem representa a line coverage do método da classe “Jdotxt.java”:

▼ jdotxt	0,1 %	6	4.097	4.103
▼ src/main/java	0,1 %	6	4.097	4.103
▼ com.chschmid.jdotxt	6,9 %	6	81	87
▼ Jdotxt.java	6,9 %	6	81	87
▼ Jdotxt	8,6 %	6	64	70
addFileModifiedListener(File)	0,0 %	0	2	2
archiveTodos()	0,0 %	0	8	8
insertReplaceString(String, String, int)	100,0 %	4	0	4
isMacOSX()	0,0 %	0	1	1
isWindows()	0,0 %	0	1	1
loadPreferences()	0,0 %	0	1	1

Line and branch coverage com os novos testes criados:

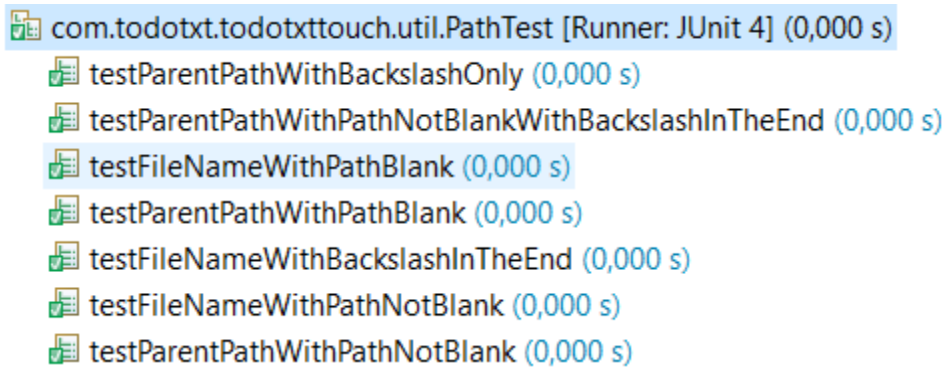
1. Package – com.todo.txt.todo.txttouch.util/ Classe – Path.java:

1.1. Método – fileName(String path):



- 1.1.1. `testFileNameWithPathBlank`: Testar o comportamento do método quando recebe uma string em branco. Tal como era esperado, o resultado correspondeu a uma string vazia.
- 1.1.2. `testFileNameWithPathNotBlank`: Testar o comportamento do método quando recebe um caminho que não tem barra no fim. Tal como esperado, foi retornado "todo.java".
- 1.1.3. `testFileNameWithPathBackslashInTheEnd`: Testar o comportamento do método quando recebe um caminho com barra no fim. Passou-se como parâmetro um "path" igual ao que tinha sido passado no teste 1.2, mas que continha uma barra no fim. Tal como esperado, apesar de se ter a barra no fim, foi retornado "todo.java" (tal como no 1.2).

1.2. Package – com.todotxt.todotxttouch.util / Classe – Path.java / Método – parentPath(String path):



com.todotxt.todotxttouch.util.PathTest [Runner: JUnit 4] (0,000 s)

- testParentPathWithBackslashOnly (0,000 s)
- testParentPathWithPathNotBlankWithBackslashInTheEnd (0,000 s)
- testFileNameWithPathBlank (0,000 s)
- testParentPathWithPathBlank (0,000 s)
- testFileNameWithBackslashInTheEnd (0,000 s)
- testFileNameWithPathNotBlank (0,000 s)
- testParentPathWithPathNotBlank (0,000 s)

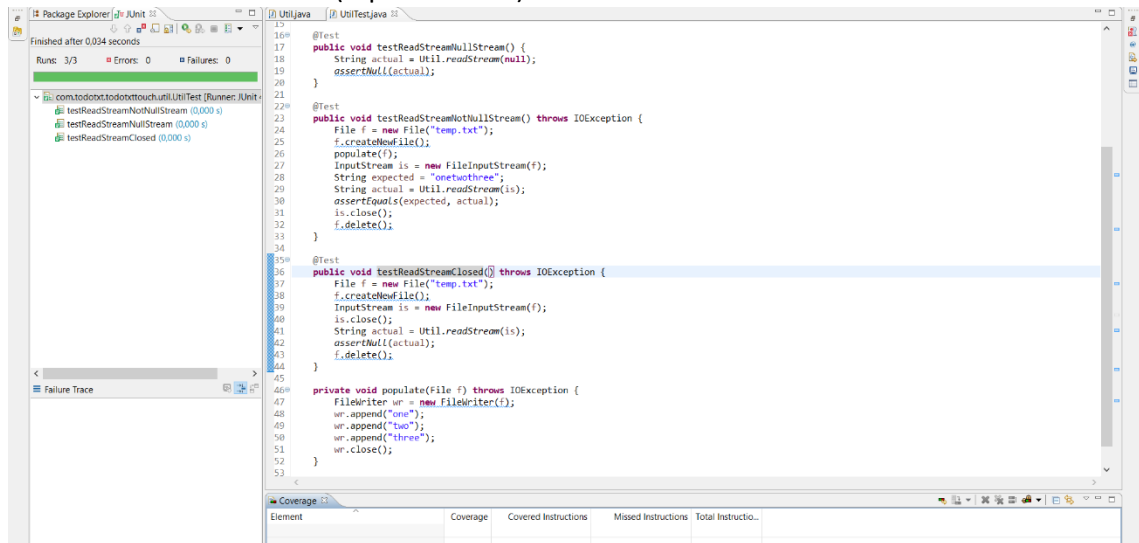


```
21 assertEquals(expected, actual);
22 }
23
24 @Test
25 public void testFileNameWithBackslashInTheEnd() {
26     String path = "test/code/todo.java\\";
27     String expected = "todo.java";
28     String actual = Path.fileName(path);
29     assertEquals(expected, actual);
30 }
31
32 @Test
33 public void testParentPathWithPathBlank() {
34     String expected = "";
35     String actual = Path.parentPath("");
36     assertEquals(expected, actual);
37 }
38
39 @Test
40 public void testParentPathWithBackslashOnly() {
41     String expected = "";
42     String actual = Path.parentPath("\\");
43     assertEquals(expected, actual);
44 }
45
46 @Test
47 public void testParentPathWithPathNotBlank() {
48     String path = "test/code/todo.java";
49     String expected = "test/code/";
50     String actual = Path.parentPath(path);
51     assertEquals(expected, actual);
52 }
53
54 @Test
55 public void testParentPathWithPathNotBlankWithBackslashInTheEnd() {
56     String path = "test/code/todo.java\\";
57     String expected = "test/code/";
58     String actual = Path.parentPath(path);
59     assertEquals(expected, actual);
60 }
61 }
62
63 }
```

- 1.2.1.testParentPathWithPathBlank: Testar o comportamento do método quando recebe como input um caminho em branco. Quando a String passada como parâmetro corresponde a uma String em branco ou a uma barra ("/"), retorna-se uma String vazia. Deste modo, tal como esperado, foi retornada uma String vazia.
- 1.2.2.testParentPathWithBackslashOnly: Testar o comportamento do método quando recebe como input apenas um backslash. Quando a String passada como parâmetro corresponde a uma String em branco ou a uma barra ("/"), retorna-se uma String vazia. Deste modo, tal como esperado, foi retornada uma String vazia.
- 1.2.3.testParentPathWithPathNotBlank: Testar o comportamento do método quando recebe como input um caminho não vazio e que não termina com uma barra no fim.
- 1.2.4.testParentPathWithPathNotBlankWithBackslashInTheEnd: Testar o comportamento do método quando recebe como input um caminho não vazio e que termina com uma barra no fim.

2. Package – com.todo.txt.todo.txttouch.util / Classe – Util.java:

2.1. Método – readStream(InputStream is):



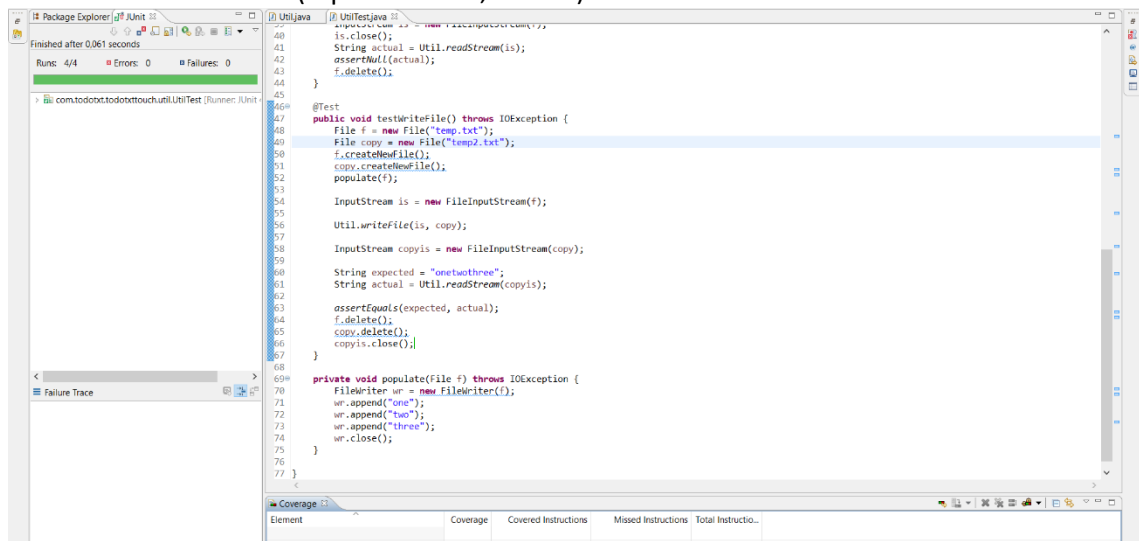
2.1.1. `testReadStreamNullStream`: Testar o comportamento do método se input for null.

Tal como esperado, foi retornado o valor null.

2.1.2. `testReadStreamNotNullStream`: Testar o comportamento do método ao passar como parâmetro uma stream que ainda não foi fechada. Foi criado um método privado para fazer populate de um dado ficheiro. Por sua vez, este ficheiro foi usado para criar o `InputStream` e, como previsto, foi retornada uma string com o conteúdo desse ficheiro.

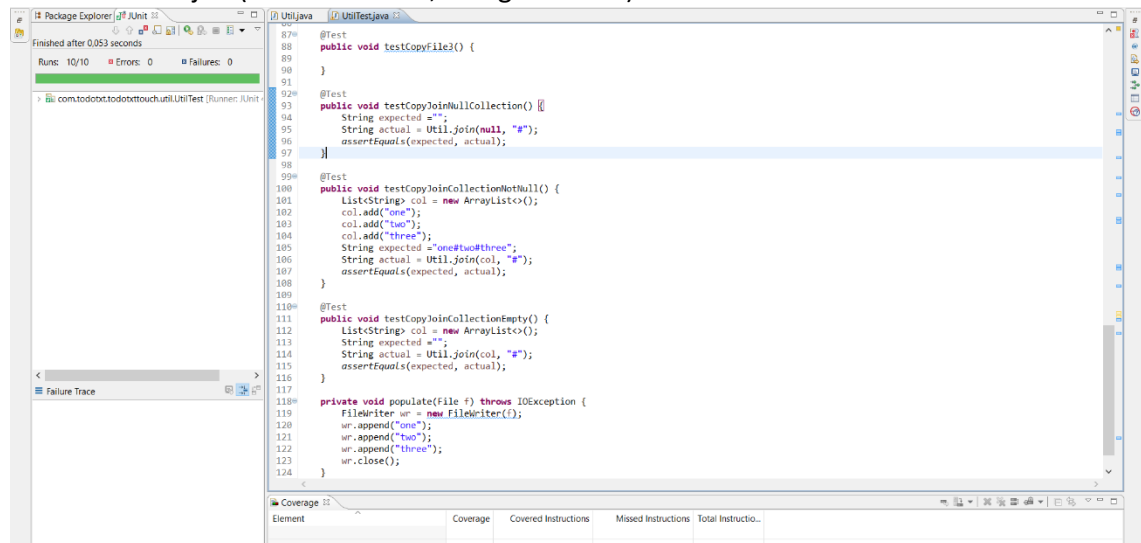
2.1.3. `testReadStreamClosed`: Testar o comportamento do método quando o input stream passado como parâmetro já tiver sido fechado. Tal como esperado, foi retornado null.

2.2. Método – writeFile(InputStream is, File file):



2.2.1. `testWriteFile`: Testar o comportamento do método ao receber um ficheiro válido para o qual pode copiar o conteúdo de um input stream (também válido). Copiou-se o conteúdo de um `InputStream`, "is", para o ficheiro "copy", tendo-se obtido o resultado esperado.

2.3. Método – join(Collection<?> s, String delimiter):

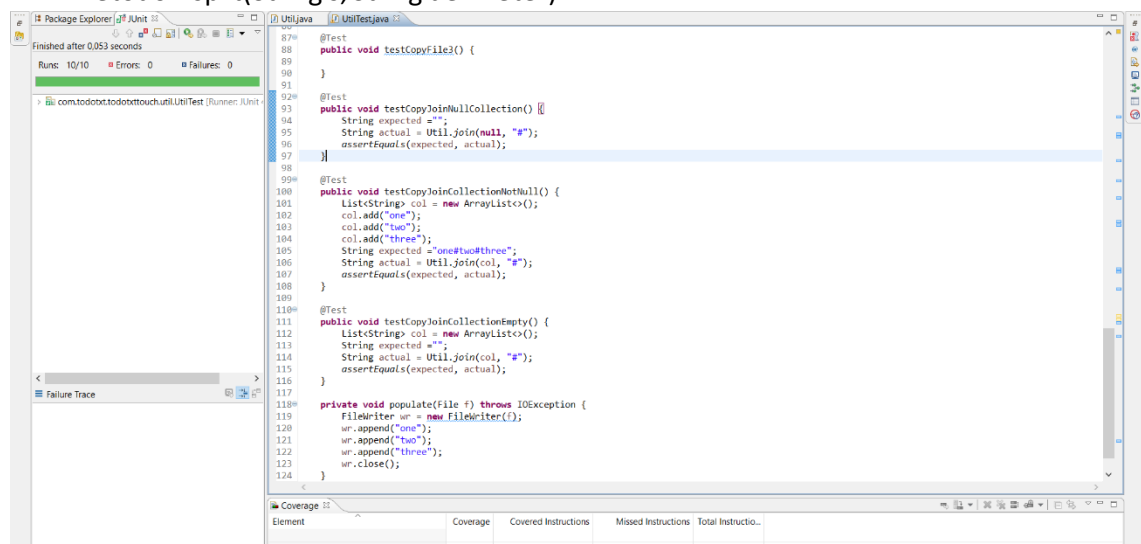


2.3.1. `testCopyJoinNullCollection`: Testar o comportamento do método ao passar como parâmetro uma Collection null. Como esperado, foi retornada uma String vazia.

2.3.2. `testCopyJoinCollectionNotNull`: Testar o comportamento do método quando se passa como parâmetro uma Collection com um dado número de elementos. Adicionou-se um conjunto de elementos à Collection e, como esperado, foi retornada uma String com os elementos da Collection separados pelo delimitador.

2.3.3. `testCopyJoinCollectionEmpty`: Testar o comportamento do método quando se passa como parâmetro uma Collection vazia. Como esperado, foi retornada uma String vazia.

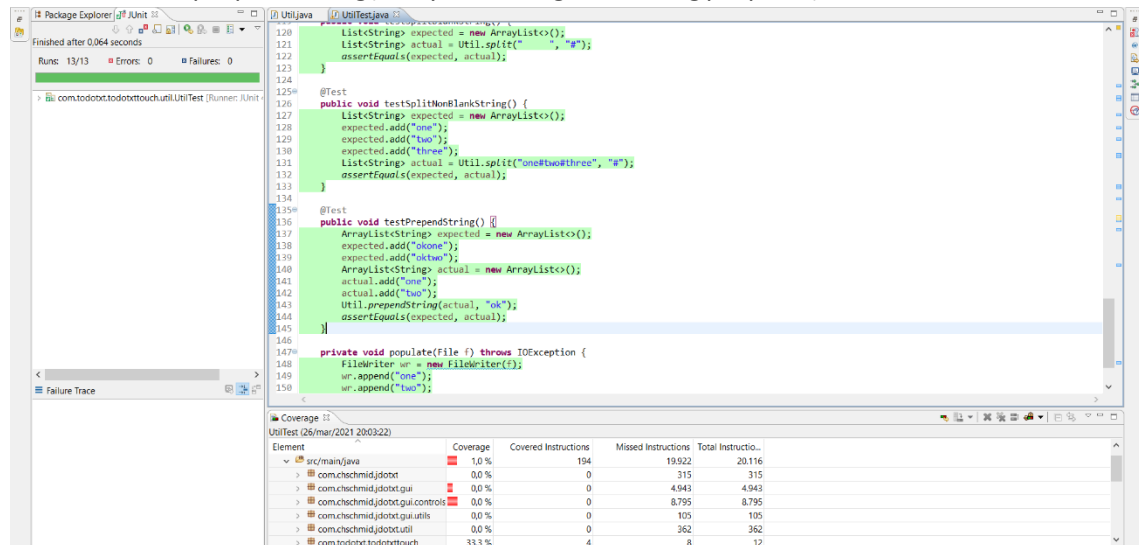
2.4. Método – split(String s, String delimiter):



2.4.1. `testSplitBlankString`: Testar o comportamento do método quando se tenta fazer split de uma string em branco. Como esperado, foi retornada uma lista vazia.

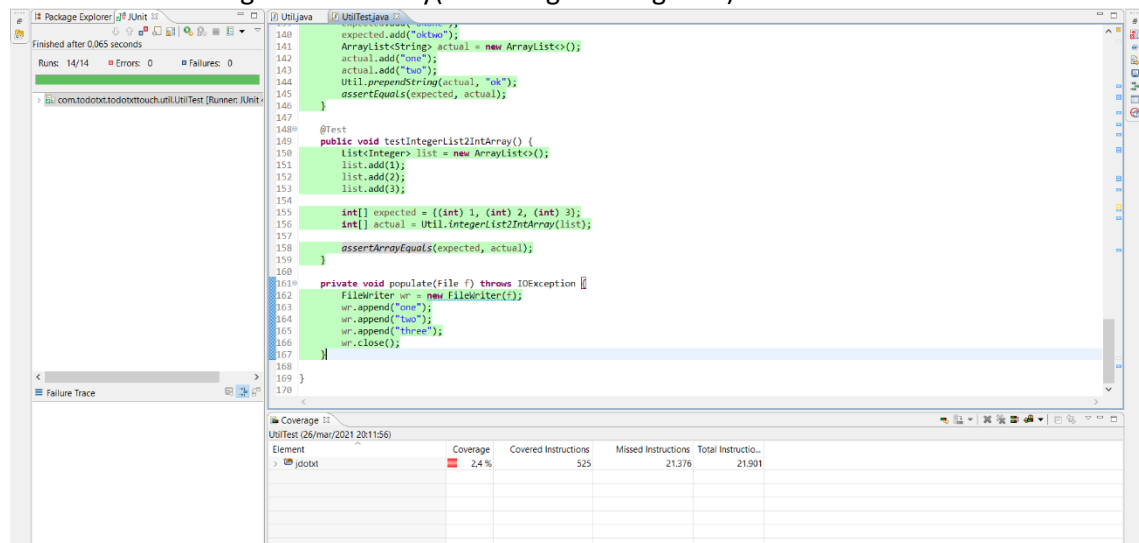
2.4.2. `testSplitNonBlankString`: Testar o comportamento do método quando se passa como parâmetro uma string não vazia que contém algumas instâncias do delimitador. Tal como esperado, a string foi dividida num array de acordo com o delimitador.

2.5. Método – prependString(ArrayList<String> list, String prepend):



2.5.1. `testPrependString`: Testar o comportamento do método quando se passa como parâmetro uma lista com valor diferente de null e uma string (também diferente de null) para poder ser adicionada como prefixo em cada um dos elementos da lista.

2.6. Método – integerList2IntArray(List<Integer> integerlist):



2.6.1. `testIntegerList2IntArray`: Testar o comportamento do método quando se passa como parâmetro uma lista não vazia de inteiros

3. Package – com.todoxtxt.todoxtxtouch.task / Classe – LocalFileTaskRepository.java:

Runs: 4/4 Errors: 0 Failures: 0

com.todoxtxt.todoxtxtouch.task.LocalFileTaskRepositoryTest [Runner: JUnit 4] (0,000 s)

- testStoredDoneTasks (0,000 s)
- testArchivedTodoTasks (0,000 s)
- testStoredTodoTasks (0,000 s)
- testArchivedDoneTasks (0,000 s)

```
36 public void testStoredTodoTasks() {
37     ArrayList<Task> tasks = new ArrayList<>();
38     tasks.add(new Task(0, "task1"));
39     tasks.add(new Task(1, "task2"));
40     tasks.add(new Task(2, "task3"));
41     repo.store(tasks);
42     ArrayList<Task> expected = tasks;
43     ArrayList<Task> actual = repo.load();
44     repo.purge();
45     assertEquals(expected, actual);
46 }
47 @Test
48 public void testStoredDoneTasks() {
49     ArrayList<Task> tasks = new ArrayList<>();
50     tasks.add(new Task(0, "task1"));
51     tasks.add(new Task(1, "task2"));
52     tasks.add(new Task(2, "task3"));
53     repo.storeDoneTasks(tasks);
54     ArrayList<Task> expected = tasks;
55     ArrayList<Task> actual = repo.loadDoneTasks();
56     repo.purge();
57     assertEquals(expected, actual);
58 }
59 @Test
60 public void testArchivedTodoTasks() {
61     ArrayList<Task> tasks = new ArrayList<>();
62     tasks.add(new Task(0, "task1"));
63     tasks.add(new Task(1, "task2"));
64     tasks.add(new Task(2, "task3"));
65     tasks.get(0).markComplete(new Date());
66     tasks.get(1).markIncomplete(new Date());
67     tasks.get(2).markComplete(new Date());
68     repo.archive(tasks);
69     ArrayList<Task> loaded = repo.load();
70     assertEquals(tasks.get(1).getText(), loaded.get(0).getText());
71 }
72 @Test
73 public void testArchivedDoneTasks() {
74     ArrayList<Task> tasks = new ArrayList<>();
75     tasks.add(new Task(0, "task1"));
76     tasks.add(new Task(1, "task2"));
77     tasks.add(new Task(2, "task3"));
78     tasks.get(0).markComplete(new Date());
79     tasks.get(1).markIncomplete(new Date());
80     tasks.get(2).markComplete(new Date());
81     repo.archive(tasks);
82     ArrayList<Task> loaded = repo.loadDoneTasks();
83     assertEquals(tasks.get(0).loaded.get(0));
```

3.1. Método – store(ArrayList<Task> tasks):

3.1.1.testStoredTodoTask: Testar o comportamento do método quando recebe como parâmetro um ArrayList de tarefas (com um dado número de elementos), fazendo-se “store” no ficheiro “todo.txt” das tarefas da lista. De seguida, faz-se “load” da lista de tarefas do ficheiro “todo.txt” e compara-se com a lista de tarefas original. Tal como era esperado, tendo em conta que não foram feitas quaisquer alterações nas tarefas contidas no “todo.txt”, a lista que foi loaded é igual à lista original.

3.2. Método – storeDoneTasks(ArrayList<Task> tasks):

3.2.1.testStoredDoneTasks: Testar o comportamento do método quando recebe um ArrayList de tarefas com um dado número de elementos. De seguida, fez-se “loadDoneTasks” para obter um ArrayList com as tarefas que já estavam feitas. Tendo em conta que foi feito “storeDoneTasks” à lista de tarefas original, fazendo “loadDoneTasks”, vai-se obter uma lista de tarefas igual à lista original.

3.3. Método – archive(ArrayList<Task> tasks):

3.3.1.testArchivedTodoTasks: Criou-se um ArrayList com 3 tarefas, tendo-se marcado “task1” e “task3” como “Complete” e tendo-se marcado “task2” como “Incomplete”. Fez-se “archive” desta lista de 3 tarefas e, de seguida, foi realizado um “load” para obter a lista de tarefas contidas no “todo.txt”. Tendo em conta que só uma das 3 tarefas não estava completa, então, só foi retornada essa tarefa como resultado do “load”.

3.3.2.testArchivedDoneTasks: Tal como no teste realizado no 3.1, criou-se uma lista de tarefas (neste caso, “task1” foi marcada como “Complete” e “task2” e “task3” foram marcadas como “Incomplete”). Tendo-se feito “archive” desta lista de tarefas, procedeu-se a fazer load das tarefas que já estavam feitas (“loadDoneTasks”) e, tal como esperado, foi só retornada a “task1” que era a única que estava marcada como “Complete”.

4. Teste – TreeTest / Package – com.todotxt.todotxttouch.task / Classe – LocalFileTaskRepository.java:

Runs: 12/12 Errors: 0 Failures: 0

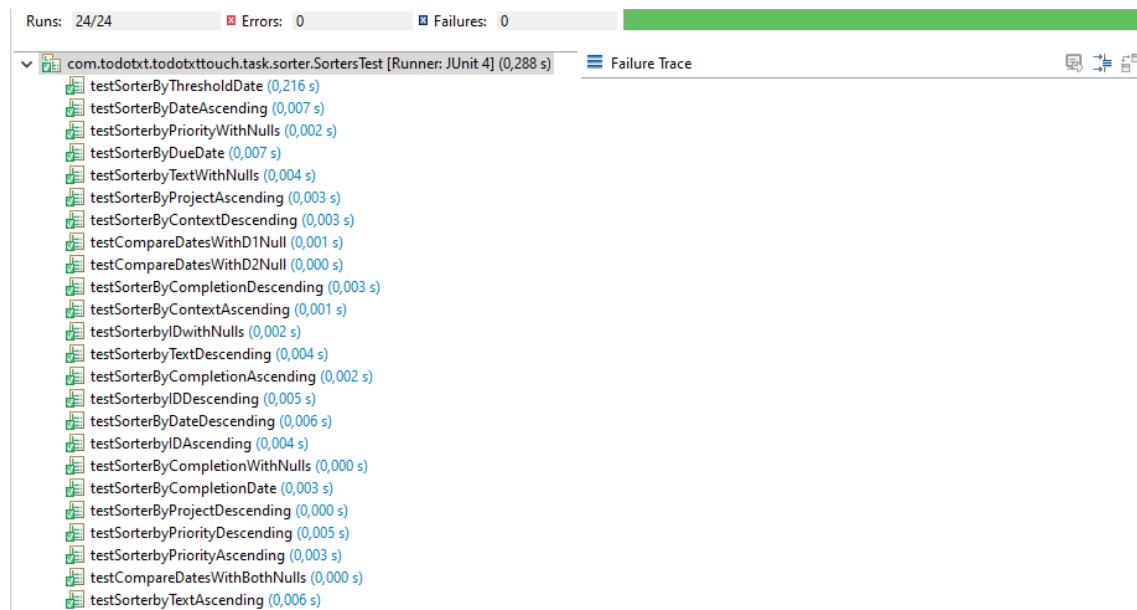
com.todotxt.todotxttouch.util.TreeTest [Runner: JUnit 4] (0,032 s)

- testGetChild (0,031 s)
- testGetData (0,000 s)
- testContainsData1 (0,000 s)
- testContainsData2 (0,000 s)
- testContainsData3 (0,000 s)
- testIsLoaded1 (0,000 s)
- testIsLoaded2 (0,000 s)
- testGetParent (0,000 s)
- testGetChildrenNull (0,000 s)
- testContainsChild1 (0,000 s)
- testContainsChild2 (0,000 s)
- testContainsChild3 (0,000 s)

```
9
10 public class TreeTest {
11
12     @Test
13     public void testIsLoaded1() {
14         Tree<Integer> tree = new Tree<>(2);
15         assertFalse(tree.isLoaded());
16     }
17
18     @Test
19     public void testIsLoaded2() {
20         Tree<Integer> tree = new Tree<>(2);
21         tree.setLoaded();
22         assertTrue(tree.isLoaded());
23     }
24
25     @Test
26     public void testGetData() {
27         Tree<String> tree = new Tree<>("01a");
28         assertEquals("01a", tree.getData());
29     }
30
31     @Test
32     public void testGetChildrenNull() {
33         Tree<String> tree = new Tree<>("01a");
34         assertNull(tree.getChildren());
35     }
36
37     @Test
38     public void testGetParent() {
39         Tree<String> parent = new Tree<>("01a");
40         Tree<String> child = parent.addChild("Adeus");
41         assertEquals(parent, child.getParent());
42     }
43
44     @Test
45     public void testContainsChild1() {
46         Tree<String> parent = new Tree<>("01a");
47         Tree<String> child = new Tree<>(parent, "Adeus");
48         parent.addChild(child);
49         assertTrue(parent.contains(child));
50     }
51
52     @Test
53     public void testContainsChild2() {
54         Tree<String> parent = new Tree<>("01a");
55         Tree<String> child = new Tree<>(parent, "Adeus");
56         assertFalse(parent.contains(child));
57     }
58
59     @Test
60     public void testContainsChild3() {
61         Tree<String> parent = new Tree<>("01a");
62         Tree<String> child = new Tree<>(parent, "Adeus");
63         parent.setLoaded();
64         assertFalse(parent.contains(child));
65     }
66
67     @Test
68     public void testContainsData1() {
69         Tree<String> parent = new Tree<>("01a");
70         assertFalse(parent.contains("Adeus"));
71     }
72
73     @Test
74     public void testContainsData2() {
75         Tree<String> parent = new Tree<>("01a");
76         parent.setLoaded();
77         assertFalse(parent.contains("Adeus"));
78     }
79
80     @Test
81     public void testContainsData3() {
82         Tree<String> parent = new Tree<>("01a");
83         parent.setLoaded();
84         parent.addChild("Ok");
85         parent.addChild("sim");
86         parent.addChild("nao");
87         assertTrue(parent.contains("sim"));
88     }
89
90     @Test
91     public void testGetChild() {
92         Tree<String> parent = new Tree<>("01a");
93         Tree<String> expected = new Tree<>("sim");
94         parent.addChild("Ok");
95         parent.addChild(expected);
96         parent.addChild("nao");
97         assertEquals(expected, parent.getChild(1));
98     }
99
100 }
101
```

- 4.1. testIsLoaded1: Consiste em testar o método "isLoaded". Neste caso, é criada uma árvore e, sem fazer "setLoaded", averigua-se se a árvore realmente foi carregada. Tal como era esperado, "isLoaded" retornou falso.
- 4.2. testIsLoaded2: Consiste em testar o método "isLoaded". Ao contrário do que foi feito no teste 4.1, vai-se fazer "setLoaded" e, ao fazer "isLoaded", vai ser retornado verdadeiro.
- 4.3. testGetData: Consiste em criar uma árvore em que se define "data". Comparando o "data" originalmente usado para a criação da árvore com o valor retornado de tree.getData(), observa-se que, tal como esperado, estes valores vão ser iguais.
- 4.4. testGetChildrenNull: Consiste em criar uma árvore em que se define apenas "data". De seguida, faz-se tree.getChildren() e, como ainda não se tinha adicionado nenhum child, o valor retornado vai ser null.
- 4.5. testGetParent: Consiste em criar uma árvore em que se define apenas "data". De seguida, é feito "parent.addChild("Adeus")", ou seja, é chamado o método "addChild" que apenas recebe como parâmetro um "data". Neste método vai ser criada uma nova árvore (Tree<E> child) e, consequentemente, é feito "addChild(child)". Tendo em conta que ainda não tinha sido adicionado nenhum child, children vai estar null, fazendo-se "children = new ArrayList<Tree<E>>()". Por fim, o child é adicionado à lista de children, define-se "child.parent = this" e retorna-se o child. Como tal, observa-se que a árvore correspondente ao parent que tinha sido criada no início do teste vai ser igual ao child.getParent().
- 4.6. testContainsChild1: Foi criada uma árvore (parent) em que se definiu apenas "data". De seguida, foi criada uma outra árvore (child) em que se definiu o "parent" (árvore criada inicialmente) e "data" e fez-se "parent.addChild(child)" para adicionar esta árvore a "parent". Tendo em conta que "child" foi adicionado a "parent", observa-se que "parent" vai conter "child".
- 4.7. testContainsChild2: Foi criada uma árvore (parent) em que se definiu apenas "data". Tal como foi feito no ponto 4.6, também foi criada uma árvore um "child", mas nunca foi adicionado ao "parent", pelo que "parent" não vai conter child.
- 4.8. testContainsChild3: Foi criada uma árvore (parent) em que se definiu apenas "data" e criou-se "child = new Tree<>(parent, "Adeus)". Fez-se "parent.setLoaded()" e observou-se que o "parent" não continha o "child".
- 4.9. testContainsData1: Criou-se "Tree<String> parent = new Tree<>("Olá)". Fez-se "parent.contains("Adeus")" e, tal como esperado, deu falso.
- 4.10. testContainsData2: Criou-se "Tree<String> parent = new Tree<>("Olá)" e fez-se "parent.setLoaded()". Averiguou-se se o "parent" continha "Adeus" e, tal como esperado, deu falso.
- 4.11. testContainsData3: Tal como tinha sido feito no ponto 4.10, criou-se "Tree<String> parent = new Tree<>("Olá)" e fez-se "parent.setLoaded()". Contudo, neste teste, adicionaram-se alguns filhos e averiguou-se se o "parent" continha "data" de um desses filhos. Tal como esperado, o resultado foi verdadeiro.
- 4.12. testGetChild: Criou-se "Tree<String> parent = new Tree<>("Olá)" e "Tree<String> expected = new Tree<>("sim)". Adicionou-se alguns filhos ao "parent", entre os quais, o "expected" (que foi o segundo filho a ser adicionado). Como tal, "parent.getChild(1)" deu igual ao "expected".

5. Teste – SortersTest/ Package – com.todoxt.todoxttouch.task.sorter / Classe – Sorters.java:



- 5.1. testSorterbyIDAscending: Começou-se por criar um conjunto de 6 tarefas (numeradas de 1 a 6). Foram criadas 2 listas: uma com as tarefas ordenadas numericamente de menor para maior (expected) e outra com as mesmas tarefas, só que ordenadas de forma aleatória (actual). Fez-se “Collections.sort(actual, Sorters.ID.ascending())” para ordenar as tarefas da lista desordenada por ordem crescente do ID. Tal como esperado, obteve-se que as 2 listas eram iguais.
- 5.2. testSorterbyIDDescending: Começou-se por criar um conjunto de 6 tarefas (numeradas de 1 a 6). Foram criadas 2 listas: uma com as tarefas ordenadas numericamente de maior para menor (expected) e outra com as mesmas tarefas, só que ordenadas de forma aleatória (actual). Fez-se “Collections.sort(actual, Sorters.ID.descending())” para ordenar as tarefas da lista desordenada por ordem decrescente do ID. Tal como esperado, obteve-se que as 2 listas eram iguais.
- 5.3. NullPointerException: Criou-se uma lista de tarefas e adicionou-se 2 tarefas a null. Fez-se sort por ordem crescente do ID. Como esperado, obteve-se null pointer exception.
- 5.4. testSorterbyPriorityAscending: Começou-se por criar um conjunto de 8 tarefas (numeradas de 1 a 8). Em cada uma dessas tarefas foi atribuída uma prioridade. Marcaram-se 3 tarefas (t3, t6 e t7) como completas. Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, só que ordenadas de acordo com a prioridade e por ordem crescente (expected). Fez-se “Collections.sort(actual, Sorters.PRIORITY.ascending())” para ordenar as tarefas de acordo com a prioridade e por ordem ascendente. Tal como esperado, obteve-se 2 listas iguais.
- 5.5. testSorterbyPriorityDescending: Criou-se um conjunto de 8 tarefas (numeradas de 1 a 8). Em cada uma dessas tarefas foi atribuída uma prioridade. Marcaram-se 3 tarefas (t3, t6 e t7) como completas. Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, mas ordenadas de acordo com a prioridade e por ordem decrescente (expected). Fez-se “Collections.sort(actual, Sorters.PRIORITY.descending())” para ordenar as tarefas de acordo com a prioridade e por ordem descendente. Tal como esperado, obteve-se 2 listas iguais.

- 5.6. `testSorterbyPriorityWithNulls`: Criou-se uma lista de tarefas e adicionou-se 2 tarefas a null. Fez-se sort por ordem crescente de prioridade. Obteve-se null pointer exception.
- 5.7. `testSorterbyTextAscending`: Começou-se por criar um conjunto de 6 tarefas (numeradas de 1 a 6). Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, só que ordenadas de acordo com o texto e por ordem crescente (expected). Fez-se `"Collections.sort(actual, Sorters.TEXT.ascending())"` para ordenar as tarefas de acordo com o texto e por ordem ascendente. Tal como esperado, obteve-se 2 listas iguais.
- 5.8. `testSorterbyTextDescending`: Começou-se por criar um conjunto de 6 tarefas (numeradas de 1 a 6). Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, só que ordenadas de acordo com o texto e por ordem decrescente (expected). Fez-se `"Collections.sort(actual, Sorters.TEXT.descending())"` para ordenar as tarefas de acordo com o texto e por ordem descendente. Obteve-se 2 listas iguais.
- 5.9. `testSorterbyTextWithNulls`: Criou-se uma lista de tarefas e adicionou-se 2 tarefas a null. Fez-se sort por ordem crescente do texto. Obteve-se um null pointer exception.
- 5.10. `testSorterByDateAscending`: Começou-se por criar um conjunto de 6 tarefas (numeradas de 1 a 6). Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, só que ordenadas de acordo com a data e por ordem crescente (expected). Fez-se `"Collections.sort(actual, Sorters.DATE.ascending())"` para ordenar as tarefas de acordo com a data e por ordem ascendente. Tal como esperado, obteve-se 2 listas iguais.
- 5.11. `testSorterByDateDescending`: Começou-se por criar um conjunto de 6 tarefas (numeradas de 1 a 6). Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, só que ordenadas de acordo com a data e por ordem decrescente (expected). Fez-se `"Collections.sort(actual, Sorters.DATE.descending())"` para ordenar as tarefas de acordo com a data e por ordem descendente. Obteve-se 2 listas iguais.
- 5.12. `testSorterByCompletionDate`: Criou-se um conjunto de 6 tarefas (numeradas de 1 a 6). Marcou-se 2 tarefas como estando completas (t2 e t4). Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, só que ordenadas de acordo com a data de completude e por ordem crescente (expected). Fez-se `"Collections.sort(actual, Sorters.COMPLETION_DATE.ascending())"` para ordenar as tarefas de acordo com a data de completude e por ordem ascendente. Obteve-se 2 listas iguais.
- 5.13. `testSorterByThresholdDate`: Começou-se por criar um conjunto de 6 tarefas (numeradas de 1 a 6). Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, só que ordenadas de acordo com a data-limite e por ordem crescente (expected). Fez-se `"Collections.sort(actual, Sorters.THRESHOLD_DATE.ascending())"` para ordenar as tarefas de acordo com a data-limite e por ordem ascendente. Obteve-se 2 listas iguais.
- 5.14. `testSorterByDueDate`: Começou-se por criar um conjunto de 6 tarefas (numeradas de 1 a 6). Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, só que ordenadas de acordo com a data-limite e por ordem crescente (expected). Fez-se `"Collections.sort(actual, Sorters.THRESHOLD_DATE.ascending())"` para ordenar as tarefas de acordo com a data de vencimento e por ordem ascendente. Tal como esperado, obteve-se 2 listas iguais.

- 5.15. `testSorterByCompletionAscending`: Começou-se por criar um conjunto de 6 tarefas (numeradas de 1 a 6). Marca-se 3 das tarefas como completas (t3, t4 e t6). Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, só que ordenadas de acordo com a completude e por ordem crescente (expected). Fez-se `"Collections.sort(actual, Sorters.COMPLETION.ascending())"` para ordenar as tarefas de acordo com a completude e por ordem ascendente. Tal como esperado, obteve-se 2 listas iguais.
- 5.16. `testSorterByCompletionDescending`: Começou-se por criar um conjunto de 6 tarefas (numeradas de 1 a 6). Marca-se 3 das tarefas como completas (t3, t4 e t6). Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, só que ordenadas de acordo com a completude e por ordem decrescente (expected). Fez-se `"Collections.sort(actual, Sorters.COMPLETION.descending())"` para ordenar as tarefas de acordo com a completude e por ordem descendente. Tal como esperado, obteve-se 2 listas iguais.
- 5.17. `testSorterByCompletionWithNulls`: Cria-se uma lista e adicionou-se 2 tarefas a null. Fez-se sort por ordem crescente de completude. Obteve-se null pointer exception.
- 5.18. `testSorterByProjectAscending`: Começou-se por criar um conjunto de 6 tarefas (numeradas de 1 a 6). Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, só que ordenadas de acordo com os projetos e por ordem crescente (expected). Fez-se `"Collections.sort(actual, Sorters.PROJECTS.ascending())"` para ordenar as tarefas de acordo com os projetos e por ordem ascendente. Obteve-se 2 listas iguais.
- 5.19. `testSorterByProjectDescending`: Começou-se por criar um conjunto de 6 tarefas (numeradas de 1 a 6). Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, só que ordenadas de acordo com os projetos e por ordem decrescente (expected). Fez-se `"Collections.sort(actual, Sorters.PROJECTS.descending())"` para ordenar as tarefas de acordo com os projetos e por ordem descendente. Tal como esperado, obteve-se 2 listas iguais.
- 5.20. `testSorterByContextAscending`: Começou-se por criar um conjunto de 6 tarefas (numeradas de 1 a 6). Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, só que ordenadas de acordo com o contexto e por ordem crescente (expected). Fez-se `"Collections.sort(actual, Sorters.CONTEXTS.ascending())"` para ordenar as tarefas de acordo com o contexto e por ordem ascendente. Tal como esperado, obteve-se 2 listas iguais.
- 5.21. `testSorterByContextDescending`: Começou-se por criar um conjunto de 6 tarefas (numeradas de 1 a 6). Foram criadas 2 listas de tarefas: uma com as tarefas ordenadas numericamente de menor para maior (actual) e outra com as mesmas tarefas, só que ordenadas de acordo com o contexto e por ordem decrescente (expected). Fez-se `"Collections.sort(actual, Sorters.CONTEXTS.descending())"` para ordenar as tarefas de acordo com o contexto e por ordem descendente. Obteve-se 2 listas iguais.
- 5.22. `testCompareDatesWithBothNulls`: Fez-se `"Sorters.compareDates(null, null, true)"` e, como d1 e d2 tinham valor null, foi retornado 0.
- 5.23. `testCompareDatesWithD1Null`: Fez-se `"Sorters.compareDates(null, new Date(), true)"` e, como apenas d1 tinha valor null, foi retornado 1.
- 5.24. `testCompareDatesWithD2Null`: Fez-se `"Sorters.compareDates(new Date(), null, true)"` e, como apenas d2 tinha valor null, foi retornado -1.

6. Teste – JdotxtTaskBagImplTest / Package – com.todotxt.todotxttouch.task / Classe – JdotxtTaskBagImpl.java:
- 6.1. testBagAddAsTask: Começa-se por adicionar 3 tarefas a uma lista de tarefas. Essas mesmas 3 tarefas são adicionadas (pela mesma ordem) ao task bag. Cria-se uma nova lista de tarefas com as tarefas do task bag. Compara-se as 2 listas de tarefas e, como esperado, observa-se que as 2 são iguais.
 - 6.2. testBagUpdate: Nos 3 testes seguintes adiciona-se 3 tarefas a uma lista de tarefas. De seguida, percorre-se todas as tarefas da lista de tarefas e adiciona-se cada tarefa no task bag.
 - 6.2.1. testBagUpdate1: Cria-se um calendário e define-se o seu tempo a partir de um `new Date()`. De seguida, adiciona-se 1 dia ao calendário e faz-se `"dt = c.getTime()"`. É criada uma nova tarefa (expected) usando `"dt": "Task expected = new Task(1, "anyTask", dt)"`. Finalmente, atualiza-se o task bag com expected (updatedTasks) e, como esperado, observa-se que `updatedTasks.get(1)` (que era onde estava a tarefa atualizada) é igual a expected.
 - 6.2.2. testBagUpdate2: Faz-se update de null no task bag, pelo que a lista de tarefas criada inicialmente vai ser igual à `bag.getTasks()`.
 - 6.2.3. testBagUpdate3: Foi criada uma nova tarefa que não faz parte do task bag. Ao tentar fazer update desta tarefa no task bag, como esperado, obtém-se um `TaskPersistException`.
 - 6.3. testBagDelete: São criadas 3 tarefas que são inseridas numa lista de tarefas. De seguida, percorre-se os elementos dessa lista de tarefas e adiciona-se os elementos no task bag.
 - 6.3.1. testBagDelete1: Cria-se uma tarefa (toRemove) que vai ser representativa da tarefa que se vai remover e que vai corresponder ao elemento da lista de tarefas na posição 1. É feito `"bag.delete(toRemove)"` para remover toRemove do task e bag e também é feito `"tasks.remove(toRemove)"` para remover a mesma tarefa da lista de tarefa. Desta forma, como se removeu a mesma tarefa em ambos os casos, então a lista de tarefas resultante vai ser igual a `"bag.getTasks()"`.
 - 6.3.2. testBagDelete2: É criada uma tarefa "toRemove" que não faz parte do task bag. Como tal, ao tentar remover essa tarefa do task bag vai-se obter um `"TaskPersistException"`.
 - 6.4. testBagGetPriorities: São criadas 3 tarefas que são inseridas numa lista de tarefas. Para cada tarefa da lista de tarefas vai ser atribuída uma dada prioridade (para as tarefas 0, 1 e 2 vai ser atribuído A, D e E respetivamente). Vai-se percorrer os elementos da lista e tarefas e adicionar os elementos ao task bag. Cria-se uma lista de prioridades (expected) e adiciona-se as prioridades A, D e E. É feito `bag.getPriorities()` para obter a lista de prioridades do task bag e, tal como esperado, observa-se que vai ser igual a expected.
 - 6.5. testBagGetContexts: Nos 2 testes seguintes adiciona-se 3 tarefas a uma lista de tarefas. De seguida, percorre-se todas as tarefas da lista de tarefas e adiciona-se cada tarefa no task bag.
 - 6.5.1. testBagGetContexts1: É criada uma lista de strings (expected) e adiciona-se 4 strings ("Cont1", "Cont2", "Cont3" e "Cont4"). Como é feito `bag.getContexts(false)`, não vai ser adicionado "-" na posição 0 do `Set<String>` que é retornado, pelo que o valor retornado vai ser igual ao expected.
 - 6.5.2. testBagGetContexts2: É criada uma lista de strings (expected) e adiciona-se 5 strings ("-", "Cont1", "Cont2", "Cont3" e "Cont4"). Neste caso especificou-se que

- getContexts(true), portanto é adicionado “-” no início, pelo que vai ser igual ao expected.
- 6.6. testBagGetProjects: Nos 2 testes seguintes adiciona-se 3 tarefas a uma lista de tarefas. De seguida, percorre-se todas as tarefas da lista de tarefas e adiciona-se cada tarefa no task bag.
- 6.6.1.testBagGetProjects1: É criada uma lista de strings (expected) e adiciona-se 4 strings (“Proj1”, “Proj 2”, “Proj 3” e “Proj 4”). Como é feito bag.getProjects(false), não vai ser adicionado “-” na posição 0 do Set<String> que é retornado, pelo que o valor retornado vai ser igual ao expected.
- 6.6.2.testBagGetProjects2: É criada uma lista de strings (expected) e adiciona-se 4 strings (“-”, “Proj1”, “Proj 2”, “Proj 3” e “Proj 4”). Como é feito bag.getContexts(true), vai ser adicionado “-” na posição 0 do Set<String> que é retornado, pelo que o valor retornado vai ser igual ao expected.
- 6.7. testBagSize: Cria-se uma lista de tarefas com 3 tarefas. Percorre-se os elementos da lista de tarefas, adicionando cada tarefa da lista ao task bag. Sendo que a linha tinha 3 elementos, então o task bag também vai ter tamanho 3.
- 6.8. testBagStore: Cria-se uma lista de tarefas (expected) com 3 tarefas. Percorre-se os elementos da lista de tarefas, adicionando cada tarefa da lista ao task bag. É feito store e depois reload do task bag. Tal como esperado, bag.getTasks() vai ser igual a expected.
- 6.9. testBagArchive: Testar a capacidade de arquivar e desarquivar elementos do task bag.
- 6.9.1.testBagArchive1
- 6.9.2.testBagArchive2
- 6.9.3.testBagArchive3
- 6.9.4.testBagArchive4
- 6.10. testBagFilterByPriority: São criadas 6 tarefas e são atribuídas prioridades a cada uma destas. Estas tarefas são adicionadas ao task bag pela mesma ordem com que foram criadas. Cria-se uma lista de prioridades e adiciona-se apenas a prioridade “A”. Cria-se uma lista de tarefas (expected) que contém as tarefas t2 e t3. Por fim, obtém-se as tarefas do task bag que contém prioridade A, o que corresponde apenas t2 e t3, pelo que o valor obtido vai ser igual a expected.
- 6.11. testBagFilterByContext: São criadas 6 tarefas. Estas tarefas são adicionadas ao task bag pela mesma ordem com que foram criadas. Cria-se uma lista de Strings e adiciona-se apenas a String “Cont1”. Cria-se uma lista de tarefas (expected) que contém as tarefas t1, t3 e t4. Por fim, obtém-se as tarefas do task bag que contém contexto “Cont1”, o que corresponde também a t1, t3 e t4, pelo que o valor obtido vai ser igual a expected.
- 6.12. testBagFilterByProject: São criadas 6 tarefas que são adicionadas ao task bag. É criada uma lista de Strings que representa os projetos e adiciona-se apenas o projeto “P2”. Tendo em conta que, de entre as tarefas inicialmente criadas as únicas que continham “P2” eram t2 e t4, criou-se uma lista tasks (expected) apenas com estas 2 tarefas. Por fim, obtém-se as tarefas do task bag que contém apenas “P2”, o que também vai corresponder às tarefas t2 e t4.
- 6.13. testBagFilterByTestWithCaseSensitive: Foram criadas 6 tarefas que foram adicionadas ao task bag. É criada uma lista de tarefas (expected) que só contém a tarefa t2. É criada uma String filterText = “faZer”. Define-se um boolean “caseSensitive = true” e obtém-se uma lista de tarefas (actual) do task bag que contenham a String “faZer”, sendo que se considera que existe uma diferença entre letras maiúsculas e minúsculas. Portanto, apesar de existirem várias tarefas com a palavra “fazer”, apenas a tarefa t2

tem a letra “Z” maiúscula e as restante letras minúsculas, pelo que actual vai ser igual a expected.

- 6.14. `testBagFilterByTestWithOutCaseSensitive`: Foram criadas 6 tarefas que foram adicionadas ao task bag. É criada uma lista de tarefas (expected) que contém as tarefas t1, t2 e t3. É criada uma String filterText = “faZer”. Define-se um boolean “caseSensitive = false” e obtém-se uma lista de tarefas (actual) do task bag que contenham a String “faZer”, sendo que não existem diferenças entre letras maiúsculas e minúsculas. Portanto, todas as tarefas com a palavra “fazer” vão estar incluídas no actual, ou seja, actual vai conter as tarefas t1, t2 e t3 e vai ser igual a expected.
- 6.15. `expected.testBagFilterByHidden`: Foram criadas 6 tarefas que foram adicionadas ao task bag. É criada uma lista de tarefas (expected) que vai conter os elementos t1, t3 e t5 que correspondem às tarefas que não estão escondidas. Obtém-se uma lista de tarefas (actual) que corresponde às tarefas que não estavam escondidas e observa-se que, tal como esperado, vai ser igual a expected.
- 6.16. `testBagFilterByThreshold`: Foram criadas 6 tarefas que foram adicionadas ao task bag. É criada uma lista de tarefas (expected) que vai conter os elementos t1, t2, t5 e t6 que representam as tarefas cuja data limite já passou. Obtém-se uma lista de tarefas (actual) que corresponde às tarefas que cuja data limite já passou. Tal como esperado, actual vai ser igual a expected.
- 6.17. `testBagAndFilter`: Foram criadas 6 tarefas que foram adicionadas ao task bag. São criadas 2 listas de tarefas: “List<String> contexts = new ArrayList<>()” e adiciona-se “C2” a esta lista; “List<String> projects = new ArrayList<>()” e adiciona-se “P2” a esta lista. Vai-se criar “AndFilter filter = new AndFilter()” e adiciona-se contexts e projects ao filter. É criada uma lista de tarefas que vai conter os elementos t1 e t3 porque contêm tanto “C2” como “P2”. Obtém-se os elementos de task bag que contêm “C2” e “P2” e vai dar um resultado igual a expected.
- 6.18. `testBagOrFilter`: Foram criadas 6 tarefas que foram adicionadas ao task bag. São criadas 2 listas de tarefas: “List<String> contexts = new ArrayList<>()” e adiciona-se “C2” a esta lista; “List<String> projects = new ArrayList<>()” e adiciona-se “P2” a esta lista. Vai-se criar “AndFilter filter = new AndFilter()” e adiciona-se contexts e projects ao filter. É criada uma lista de tarefas que vai conter os elementos t1, t3, t4 e t5 porque contêm “C2” ou “P2”. Obtém-se os elementos de task bag que contêm “C2” ou “P2” e observa-se que vai dar um resultado igual a expected.
- 6.19. `testBagFilterByPriorityWithNothing`: Foram criadas 6 tarefas e atribuiu-se prioridade a cada uma das tarefas. As tarefas criadas foram adicionadas ao task bag. Criou-se uma lista de tarefas (expected) e adicionou-se as tarefas criadas inicialmente a essa lista. Criou-se uma lista de tarefas (actual) fazendo-se bag.getTasks(new ByPriorityFilter(null), Sorters.ID.ascending()). Como esperado, actual deu igual a expected.
- 6.20. `testBagFilterByContextWithNothing`: Foram criadas 6 tarefas. As tarefas criadas foram adicionadas ao task bag. Criou-se uma lista de tarefas (expected) e adicionou-se as tarefas criadas inicialmente a essa lista. Criou-se uma lista de tarefas (actual) fazendo-se bag.getTasks(new ByContextFilter(null), null). Como esperado, actual deu igual a expected.
- 6.21. `testBagFilterByContextWithNoContext`: Foram criadas 6. As tarefas criadas foram adicionadas ao task bag. Criou-se uma lista de tarefas (expected) e adicionou-se as tarefas t1 e t6. Criou-se uma lista de Strings e adicionou-se apenas a String “-” à lista. Criou-se uma lista de tarefas (actual) fazendo-se bag.getTasks(new

ByContextFilter(ctxts), null) para obter as tarefas que não têm contexto. Como esperado, actual deu igual a expected.

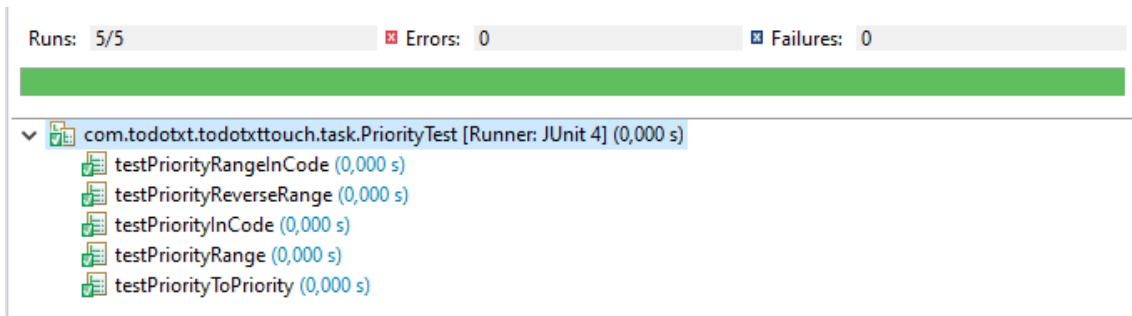
- 6.22. testBagFilterByProjectWithNothing: Foram criadas 6. As tarefas criadas foram adicionadas ao task bag. Criou-se uma lista de tarefas (expected) e adicionou-se as tarefas criadas inicialmente a essa lista. Criou-se uma lista de tarefas (actual) fazendo-se bag.getTasks(new ByProjectFilter(null), null). Como esperado, actual deu igual a expected.
- 6.23. testBagFilterByProjectWithNoProjects: Foram criadas 6. As tarefas criadas foram adicionadas ao task bag. Criou-se uma lista de tarefas (expected) e adicionou-se as tarefas t2 e t3. Criou-se uma lista de Strings e adicionou-se apenas a String “-” à lista. Criou-se uma lista de tarefas (actual) fazendo-se bag.getTasks(new ByProjectFilter(projs), null) para obter as tarefas que não têm projetos. Como esperado, actual deu igual a expected.
- 6.24. testBagFilterByTest: Foram criadas 6. As tarefas criadas foram adicionadas ao task bag. Criou-se uma lista de tarefas (expected) e adicionou-se as todas as tarefas. Criou-se uma lista de tarefas (actual) e definiu-se “bag.getTasks(new ByTextFilter(null, true), null)”. Tal como esperado, actual deu igual a expected.

7. Teste – TaskBagImplTest / Package – com.todoxt.todoxttouch.task / Classe – TaskBagImpl.java:

Nota: Tendo em conta a semelhanças que podem ser observadas entre esta classe e a classe JdotxtTaskBagImpl.java (existe muito código repetido). Como tal, optou-se por apresentar apenas os testes que fossem suficientemente diferentes ao ponto de justificar uma nova descrição.







- 7.1. testBagUpdate: Consiste em testar o comportamento do task bag quando é feito o update, podendo-se:
 - 7.1.1. testBagUpdate1: Fazer update de uma tarefa existente e válida
 - 7.1.2. testBagUpdate2: Fazer update de null
 - 7.1.3. testBagUpdate3: Fazer update de uma tarefa que não existe







8. Teste – PriorityTest / Package – com.todoxt.todoxttouch.task / Classe – Priority.java:



- 8.1. testPriorityRange: Começou por se criar uma lista de prioridades e adicionou-se um conjunto de 9 prioridades (a variar entre C e K) a essa lista. Comparou-se esta lista com Priority.range(Priority.C,Priority.K) e, tal como esperado, as 2 listas são iguais.
- 8.2. testPriorityReverseRange: Começou por se criar uma lista de prioridades e adicionou-se um conjunto de 9 prioridades (a variar entre K e C) a essa lista. Comparou-se esta lista com Priority.range(Priority.K,Priority.C) e, tal como esperado, as 2 listas são iguais.
- 8.3. testPriorityRangeInCode: Criou-se uma lista de Strings e adicionou-se ao todo 9 Strings a essa lista (com valores a variar "C" e "K"). Comparou-se esta lista com Priority.rangeInCode(Priority.C,Priority.K) e obteve-se que as 2 Strings eram iguais.
- 8.4. testPriorityInCode: Criou-se uma lista de Strings (expected) e adicionou-se 4 Strings (com os valores C, E, D e Z, respetivamente). Criou-se uma lista de prioridades (prios) e, tal como tinha sido feito no expected, foram adicionadas 4 prioridades com os valores C, E, D e Z. Fez-se Priority.inCode(prios) para converter a lista de prioridades numa lista de Strings e obteve-se uma lista igual a expected.
- 8.5. testPriorityToPriority: Criou-se uma lista de prioridades (expected) e adicionou-se 5 prioridades (com os valores C, E, NONE, D e Z, respetivamente). Criou-se uma lista de Strings (priosStr) e, tal como tinha sido feito no expected, foram adicionadas 5 prioridades com os valores C, E, None, D e Z. Fez-se Priority.toPriority(priosStr) para converter a lista de Strings numa lista de prioridades e obteve-se uma lista igual a expected.

Line e branch coverage dos unit tests concebidos neste trabalho:

Element	Coverage	Covered Lines	Missed Lines	Total Lines
>  com.todotxt.todotxttouch	40,0 %	2	3	5
>  com.todotxt.todotxttouch.task	 90,5 %	781	82	863
>  com.todotxt.todotxttouch.task.sorter	96,6 %	86	3	89
>  com.todotxt.todotxttouch.util	 87,1 %	242	36	278

Element	Coverage	Covered Branches	Missed Branches	Total Branches
>  com.todotxt.todotxttouch		0	0	0
>  com.todotxt.todotxttouch.task	 79,5 %	326	84	410
>  com.todotxt.todotxttouch.task.sorter	86,8 %	59	9	68
>  com.todotxt.todotxttouch.util	 82,5 %	137	29	166

Branch Coverage			
	Coverage (%)	Covered	Total
com.todotxt.todotxttouch	0,0	0	0
com.todotxt.todotxttouch.task	79,5	326	410
com.todotxt.todotxttouch.task.sorter	86,8	59	68
com.todotxt.todotxttouch.util	82,5	137	166
TOTAL	81,1	522	644

Line Coverage			
	Coverage (%)	Covered	Total
com.todotxt.todotxttouch	40,0	2	5
com.todotxt.todotxttouch.task	90,5	781	863
com.todotxt.todotxttouch.task.sorter	96,6	86	89
com.todotxt.todotxttouch.util	87,1	242	278
TOTAL	90,0	1111	1235