Trabalho 9 – Grupo 12: Alexandre Monteiro - 51023 / João Afonso - 51111:

Mutation Score dos testes unitários das tarefas anteriores:

Mutation Testing consiste na avaliação da qualidade de um conjunto de testes. Para este efeito, criam-se mutantes e, posteriormente, tenta-se matar esses mutantes. Tendo em consideração que um mutante é morto se pelo menos um dos testes falhar quando o mutante é executado, optámos por efetuar ligeiras alterações no código para garantir que, quando os testes são efetuados no código original (isto é, código que não corresponde a um mutante), então nenhum teste vai falhar. Assim sendo, se um teste falhar, irá ser devido a um mutante.

Foram realizadas as seguintes alterações:

1. No seguinte código ocorria um IndexOutOfBounds quando se usava um offset negativo. Como tal, foi acrescentada uma verificação que, quando o offset é inferior a zero, faz "offset = 0", nunca permitindo que o offset seja negativo.

```
public static String insertReplaceString(String original, String replace, int offset) {
    if(offset < 0) offset = 0;
    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;
 }
```

2. No seguinte código, quando "task" tinha valor null, ocorria um NullPointerException. Assim sendo, passou-se a averiguar se "task == null" e, na eventualidade de tal ser verdade, retoma-se null.

```
private static Task find(List<Task> tasks, Task task) {
    if(task == null) return null;
    Task partialMatch1 = null;
    Task partialMatch2 = null;
    for (Task task2 : tasks) {
        if (task2 == task) {
            return task2;
        }
        if (task2.getText().equals(task.getOriginalText())) {
            if (task2.getPriority() == task.getOriginalPriority()) {
                partialMatch1 = task2;
                if (task2.getId() == task.getId()) return task2;
            }

            // We prefer to find an exact match (both text and priority are
            // the same), but it is possible that priority has been lost
            // because the task has been completed, so we will consider
            // partial matches as a last resort.
            partialMatch2 = task2;
        }
    }
    if (partialMatch1 != null) return partialMatch1;
    return partialMatch2;
}
```

3. Ao analisar a seguinte função torna-se claro que a posição do cursor nunca devia de ser inferior a zero. Como tal, se "priorCursorPosition" for inferior a zero, não faz sentido estar a calcular a nova posição de um cursor partindo de uma posição negativa. Da mesma forma que "pos" é alterado para zero se o seu valor for inferior a zero, então, se "priorCursorPosition" for inferior a zero, o seu valor também vai ser colocado a zero.

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

    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;
}
```

Tendo-se realizado estas alterações no código original, utilizou-se a ferramenta PIT de modo a obter o seguinte Mutation Score para os testes unitários que tinham sido originalmente concebidos nas tarefas anteriores:

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
34	90%	1137/1260	70%	506/722	78%	506/651

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
com.todotxt.todotxttouch.task	25	90%	786/869	66%	295/444	75%	295/392
com.todotxt.todotxttouch.task.sorter	2	98%	103/105	86%	65/76	89%	65/73
com.todotxt.todotxttouch.util	7	87%	248/286	72%	146/202	78%	146/186

com.todotxt.todotxttouch.task

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
25	90%	786/869	66%	295/444	75%	295/392

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
AndFilter.java	100%	10/10	100%	4/4	100%	4/4
ByContextFilter.java	93%	14/15	90%	9/10	100%	9/9
ByPriorityFilter.java	91%	10/11	86%	6/7	100%	6/6
ByProjectFilter.java	93%	14/15	90%	9/10	100%	9/9
ByTextFilter.java	88%	14/16	69%	9/13	90%	9/10
ContextParser.java	93%	14/15	100%	4/4	100%	4/4
FilterFactory.java	93%	14/15	81%	13/16	81%	13/16
HiddenFilter.java	100%	2/2	100%	2/2	100%	2/2
HiddenParser.java	100%	5/5	100%	3/3	100%	3/3
JdotxtTaskBagImpl.java	91%	122/134	59%	32/54	65%	32/49
LinkParser.java	82%	14/17	50%	2/4	50%	2/4
LocalFileTaskRepository.java	84%	48/57	64%	16/25	64%	16/25
MailAddressParser.java	92%	12/13	50%	2/4	50%	2/4
OrFilter.java	92%	11/12	71%	5/7	83%	5/6
PhoneNumberParser.java	100%	5/5	100%	1/1	100%	1/1
Priority.java	95%	58/61	83%	20/24	95%	20/21
PriorityTextSplitter.java	93%	13/14	83%	5/6	100%	5/5
ProjectParser.java	93%	14/15	100%	4/4	100%	4/4
RecParser.java	100%	12/12	100%	3/3	100%	3/3
Task.java	88%	190/217	56%	92/164	68%	92/136
TaskBagFactory.java	67%	2/3	100%	1/1	100%	1/1
TaskBagImpl.java	90%	122/136	59%	33/56	67%	33/49
TextSplitter.java	98%	42/43	90%	9/10	100%	9/9
ThresholdDateFilter.java	100%	5/5	80%	4/5	80%	4/5
ThresholdDateParser.java	90%	19/21	100%	7/7	100%	7/7

Package Summary

com.todotxt.todotxttouch.task.sorter

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	98%	103/105	86%	65/76	89%	65/73

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
GenericSorter.java	88%	7/8	40%	2/5	50%	2/4
Sorters.java	99%	96/97	89%	63/71	91%	63/69

com.todotxt.todotxttouch.util

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
7	87%	248/286	72%	146/202	78%	146/186

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
CursorPositionCalculator.java	91%	10/11	75%	9/12	75%	9/12
Path.java	92%	12/13	100%	11/11	100%	11/11
RelativeDate.java	96%	24/25	61%	17/28	61%	17/28
Strings.java	93%	41/44	77%	30/39	81%	30/37
TaskIo.java	80%	49/61	70%	26/37	79%	26/33
Tree.java	100%	35/35	95%	18/19	95%	18/19
Util.java	79%	77/97	63%	35/56	76%	35/46

Partindo destes resultados iniciais, procurou-se matar o maior número de mutantes possível para maximizar o Mutation Score.

Criação de novos testes para matar os mutantes que estão vivos:

Package Summary - com.todotxt.todotxttouch.task.sorter:

Sorters.java:

Foram criados os seguintes testes:

1. Nenhum dos testes do conjunto de testes original executava o método getName. Foi criado um Sorter do tipo "PRIORITY" e, tal como esperado, ao fazer "getName()" obtém-se a String com o valor "Priority".

```
//Mutant 1
@Test
public void testSorterbyGetName() {

    String x = Sorters.PRIORITY.getName();

    assertEquals("Priority",x);
}
```

2. Relativamente ao método "compareLists", foi necessário criar um outro teste unitário para conseguir matar o mutante que surgiu ao fazer uma alteração na seguinte linha de código: "int res = s1 == null ? (s2 == null ? 0 : -1) : (s2 == null ? 1 : s1.compareTo(s2));". Foi criado um teste para o caso de ter S2 a null e S2 com um valor correspondente a uma String:

```
@Test
public void testSorterByCompareS2NullS1Not() {

    List<String> p1 = Arrays.asList(null, "y", "c");
    List<String> p2 = Arrays.asList(null, null, "z");

    int expected = 1;
    int actual = Sorters.compareLists(p1, p2);
    assertEquals(expected,actual);
}
```

Não se consegue alcançar mutation score de 100% na classe Sorters.java porque existem múltiplos mutantes equivalentes, ou seja, mutantes que não podem ser mortos.

O primeiro conjunto de mutantes equivalentes que se obteve resultaram de mutantes em que substituiu uma multiplicação por uma divisão:

- return Long.compare(t1.getId(), t2.getId()) * (ascending ? 1 : -1);
- return t1.getPriority().compareTo(t2.getPriority()) * (ascending ? 1 : -1);
- return t1.getText().compareToIgnoreCase(t2.getText()) * (ascending ? 1 : -1);
- return compareLists(p1, p2) * (ascending ? 1 : -1);
- return compareLists(p1, p2) * (ascending ? 1 : -1);

Em todos estes casos a multiplicação vai servir o mesmo propósito: manter o resultado inalterado (multiplicar por 1) ou trocar o sinal do resultado (multiplicar por -1) – como tal, como tanto uma multiplicação como uma divisão poderiam servir o mesmo propósito, se a multiplicação em qualquer um destes casos for alterado para uma divisão, apesar da mudança, o resultado vai permanecer sempre o mesmo, não sendo possível criar um teste que consiga matar este mutante.

O outro caso correspondente a um mutante equivalente é:

```
public static <E extends Comparable> int compareLists(List<E> l1, List<E> l2) {
    if (l1.equals(l2))
        return 0;

    for (int i = 0; i < Math.min(l1.size(), l2.size()); i++) {
        E s1 = l1.get(i);
        E s2 = l2.get(i);
        int res = s1 == null ? (s2 == null ? 0 : -1) : (s2 == null ? 1 : s1.compareTo(s2));
        if (res != 0)
            return res;
    }
    return l1.size() > l2.size() ? 1 : -1;
}
```

Neste caso, o mutante equivalente quando se altera “return l1.size() > l2.size() ? 1 : -1” para o seguinte: “return l1.size() >= l2.size() ? 1 : -1” uma vez que, para matar este mutante, seria necessário criar um teste em que l1 e de l2 tivessem o mesmo tamanho e em que esta linha fosse executada. Contudo, sempre que l1 e l2 são iguais, nunca se corre esta linha, não sendo possível criar um teste que consiga matar este mutante.

Package Summary - com.todotxt.todotxttouch.util:

CursorPositionCalculator.java:

Todos as mutações que não foram mortas nesta classe correspondiam a mutantes equivalentes que surgiram no método "calculate". Todos os mutantes equivalentes resultaram da alteração de um conditional boundary:

1. É feito "if(priorCursorPosition < 0)" para alterar o valor inicial dos cursores que apontam para uma posição negativa de modo a passarem a apontar para a posição com valor zero. Ao alterar "<" para "<=", se priorCursorPosition tiver valor zero, apenas se vai passar a fazer "priorCursorPosition = 0" quando o valor de "priorCursorPosition" já correspondia a zero para começar, ou seja, não se observa alterações. Nos restantes casos, não se vai constatar qualquer mudança.
2. Alterar "pos = pos < 0 ? 0 : pos" por "pos = pos <= 0 ? 0 : pos" não vai provocar qualquer alteração porque, quando pos = 0, mesmo que "<" seja alterado para "<=", o valor de "pos" vai ser sempre o mesmo.
3. Em "return pos > newValue.length() ? newValue.length() : pos", mesmo que ">" seja alterado por ">=", o valor retornado vai permanecer inalterado.

```
        public static final int calculate(int priorCursorPosition,
                        String priorValue, String newValue) {
2               if(priorCursorPosition < 0)
                        priorCursorPosition = 0;

1               if (newValue == null) {
                        return 0;
                }

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

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

Path.java:

Não tiveram de ser criados novos testes porque já se tinha mution score de 100%.

RelativeDate.java:

Foram criados os seguintes testes:

1. Não tinham sido realizados um conjunto de testes destinados a certos valores de fronteira, o que levava certos mutantes a não ser mortos:
   1.1. Calcular o dia relativo a 2 datas iguais. Tal como esperado, foi retornado "today".
   1.2. Calcular o dia relativa a 2 datas com uma diferença de 2 dias. Tal como esperado, foi retornado "2 days ago".
   1.3. Calcular o dia relativo a 2 datas com 1 mês de diferença, sendo esta diferença de exatamente 30 dias (para corresponder a um caso de fronteira). Tal como esperado, foi retornado "1 month ago".
   1.4. Calcular o dia relativo a 2 datas com 2 meses de diferença, sendo esta diferença de exatamente 60 dias (para corresponder a um caso de fronteira). Tal como esperado, foi retornado "2 months ago".
   1.5. Calcular o dia relativo a 2 datas no caso de se estar a fazer uma comparação com uma data no futuro. Tal como esperado, foi retornado "2021-08-12" (corresponde ao valor que foi que foi passado como parâmetro).
   1.6. Calcular o dia relativo a 2 datas com exatamente 1 ano de diferença. Tal como esperado, foi retornado "2019-08-13".

```
@Test
public void testRelativeDateToday() throws ParseException {
    String expected = "today";
    assertEquals(expected, RelativeDate.getRelativeDate(today, getCalendarOf("2020-08-12")));
}

@Test
public void testRelativeDate2DaysAgo() throws ParseException {
    String expected = "2 days ago";
    assertEquals(expected, RelativeDate.getRelativeDate(today, getCalendarOf("2020-08-10")));
}

@Test
public void testRelativeDate1MonthAgo30Days() throws ParseException {
    String expected = "1 month ago";
    assertEquals(expected, RelativeDate.getRelativeDate(today, getCalendarOf("2020-07-13")));
}

@Test
public void testRelativeDate2MonthsAgo60Days() throws ParseException {
    String expected = "2 months ago";
    assertEquals(expected, RelativeDate.getRelativeDate(today, getCalendarOf("2020-06-13")));
}

@Test
public void testRelativeDateFuture() throws ParseException {
    String expected = "2021-08-12";
    assertEquals(expected, RelativeDate.getRelativeDate(today, getCalendarOf("2021-08-12")));
}

@Test
public void testRelativeDateFarInThePast() throws ParseException {
    String expected = "2019-08-13";
    assertEquals(expected, RelativeDate.getRelativeDate(today, getCalendarOf("2019-08-13")));
}
```

2. Foi criado um teste para calcular o dia relativo utilizando uma variável do tipo "Date" em vez de "Calendar" (como tinha sido feito nos restantes teste). Neste caso, como apenas se passa como parâmetro uma única data, o dia atual vai ser utilizado como referência. Como tal, foi necessário escolher uma data que tivesse ocorrido há mais de um ano para garantir que o valor retornado vai sempre o mesmo (se uma data ocorreu há mais de 1 ano, então vai ser retornada uma String com a própria data que foi passada como parâmetro). Obteve-se o resultado que era esperado.

```
@Test
public void testRelativeDateWithDate() throws ParseException {
    String expected = "1999-08-12";
    Calendar testDay = getCalendarOf("1999-08-12");
    Date date = testDay.getTime();
    assertEquals(expected, RelativeDate.getRelativeDate(date));
}
```

Strings.java:

Foram criados os seguintes testes:

1. Nos métodos "insertPaddedIfNeeded" e "insertPadded" não se estava a testar o caso em que a String a ser inserida tinha valor null. Em ambos os casos, tal como era esperado, a String na qual ia ser feita a inserção permaneceu inalterada.

```
@Test
public void testIfNeededInsertNull() {
    String s = "abc";
    String stringToInsert = null;
    int insertAt = 0;
    String expected = "abc";
    String actual = Strings.insertPaddedIfNeeded(s, insertAt, stringToInsert);
    assertEquals(expected, actual);
}

@Test
public void testInsertPaddedInsertNull() {
    String s = "abc";
    String stringToInsert = null;
    int insertAt = 0;
    String expected = "abc";
    String actual = Strings.insertPadded(s, insertAt, stringToInsert);
    assertEquals(expected, actual);
}
```

2. Nos métodos “insertPadded” e “insertPaddedIfNeeded” não se estava a ter em consideração o caso em que a String na qual é feita a inserção já termina num espaço, não sendo necessário acrescentar um outro espaço no fim. Em ambos os casos, como a String já termina num espaço, não se acrescenta um outro espaço.

```
@Test
/*
 * Method - insertPaddedIfNeeded
 */
public void testIfNeededEndInSpace() {
    String s = "xyz ";
    String stringToInsert = "xyz";
    int insertAt = 2;
    String expected = "xyz ";
    String actual = Strings.insertPaddedIfNeeded(s, insertAt, stringToInsert);
    assertEquals(expected, actual);
}

@Test
/*
 * Method - insertPaddedIfNeeded
 */
public void testInsertPaddedEndInSpace() {
    String s = "xyz ";
    String stringToInsert = "xyz";
    int insertAt = 4;
    String expected = "xyz xyz ";
    String actual = Strings.insertPadded(s, insertAt, stringToInsert);
    assertEquals(expected, actual);
}
```

3. No método “insertPaddedIfNeeded” nunca se tinha testado o caso em que, por um lado, a String que se pretende inserir ocorre várias vezes na String onde é feita a inserção e, por outro lado, nas primeiras vezes onde há repetição, a sequência de caracteres que formam a String a inserir não está separada por espaços, ou seja, não é considerado uma palavra, sendo necessário continuar a percorrer a String onde se pretende fazer a inserção para procurar a próxima ocorrência de uma repetição para averiguar se está dividida por espaços, isto é, se corresponde a uma palavra já existente. Obteve-se o resultado esperado.

```
@Test
/*
 * Method - insertPaddedIfNeeded
 */
public void testIfNeededCycleIteration() {
    String s = "abcxx xa x";
    String stringToInsert = "x";
    int insertAt = 3;
    String expected = "abcxx xa x ";
    String actual = Strings.insertPaddedIfNeeded(s, insertAt, stringToInsert);
    assertEquals(expected, actual);
}
```

4. No método "insertPaddedIfNeeded" não se tinha feito um teste para o caso em que a String a inserir é igual à String que se quer inserir. Neste caso, é apenas colocado um espaço em branco a seguir à String onde ia ser feita a inserção.

```
@Test
/*
 * Method - insertPaddedIfNeeded
 */
public void testIfNeededEqual() {
    String s = "xyz";
    String stringToInsert = "xyz";
    int insertAt = 2;
    String expected = "xyz ";
    String actual = Strings.insertPaddedIfNeeded(s, insertAt, stringToInsert);
    assertEquals(expected, actual);
}
```

Não foi possível ter 100% de Mutation Coverage porque existem 2 mutantes equivalentes:

1. No método "insertPadded", apesar de "if (insertAt > 0)" ser alterado por "if (insertAt >= 0)", o resultado de correr o bloco de código dentro do "if" ou dentro do respetivo "else" para o caso em que insertAt = 0 vai ser o mesmo.

```
56          StringBuilder newText = new StringBuilder();
57 2        if (insertAt > 0) {
58                  newText.append(s.substring(0, insertAt));
59 2                if (newText.lastIndexOf(SINGLE_SPACE) != newText.length() - 1) {
60                          newText.append(SINGLE_SPACE);
61                  }
62                  newText.append(stringToInsert);
63                  String postItem = s.substring(insertAt);
64 1                if (postItem.indexOf(SINGLE_SPACE) != 0) {
65                          newText.append(SINGLE_SPACE);
66                  }
67                  newText.append(postItem);
68          } else {
69                  newText.append(stringToInsert);
70 1                if (s.indexOf(SINGLE_SPACE) != 0) {
71                          newText.append(SINGLE_SPACE);
72                  }
73                  newText.append(s);
74          }
```

2. No método "insertPaddedIfNeeded", mesmo que se altere "while ((startPos < s.length()) && (! found))" para passar a ter "startPos <= s.length()", o resultado nunca vai ser alterado porque, quando "startPos" é igual a "s.length()", "pos" vai ter sempre valor -1.

```
106 3       while ((startPos < s.length()) && (! found))
107         {
108                 int pos = s.indexOf(stringToInsert, startPos);
109
110 2               if (pos < 0)
111                         break;
112
113 1               startPos = pos + 1;
114 1               int     before = pos - 1;
115 1               int after = pos + stringToInsert.length();
116
117 2               if (((pos == 0) || (Character.isWhitespace(s.charAt(before)))) &&
118 3                ((after >= s.length()) || (Character.isWhitespace(s.charAt(after)))))
119                         found = true;
120         }
121
```

Tree.java:

Só foi criado um novo teste, tendo-se conseguido obter 100% mutation score nesta classe:

1. No método "public boolean contains(E data)" ainda não se tinha testado a situação em que existem vários child nodes, mas nenhum desses corresponde ao que se está a procurar:

```
@Test
public void testContainsData4() {
    Tree<String> parent = new Tree<>("Olá");
    parent.setLoaded();
    parent.addChild("Ok");
    parent.addChild("sim");
    parent.addChild("nao");
    assertFalse(parent.contains("algo"));
}
```

Util.java:

Foram criados os seguintes testes:

1. Ainda não tinha sido criado um teste em que se chamasse o método "isDeviceReadable()".
2. Foi necessário criar um teste para o caso de se tentar copiar um ficheiro que não existe. Neste caso, tal como esperado, foi lançada uma exceção.

```
@Test
public void testIsDeviceReadable() {
    assertTrue(Util.isDeviceReadable());
}

@Test(expected = TodoException.class)
public void testcopyFileNotExists() throws IOException {
    File origFile = new File(DEFAULTDIR + File.separator + "origFile.txt");
    File newFile = new File(DEFAULTDIR + File.separator + "newFile.txt");
    newFile.createNewFile();
    Util.copyFile(origFile, newFile, true);
    origFile.delete();
    newFile.delete();
}
```

Package Summary - com.todotxt.todotxttouch.task:

AndFilter.java:

Não tiveram de ser criados novos testes porque já se tinha mution score de 100%.

ByContextFilter.java, ByPriorityFilter.java e ByProjectFilter.java:

Nestas 3 classes a única mutação que não foi morta está associada com um método que apenas que apenas é para ser usado para testes, não devendo de ser usado pela aplicação. Como tal, optou-se por não matar estes mutantes. Qualquer mutante que possa surgir posteriormente e que faça parte de um método destinado apenas a teste também não será morto.

ByTextFilter.java:

Os testes feitos para esta classe foram criados no JdotxtTaskBagImplTest.java. Os testes que foram feitos aqui foram para testar as funções getText() e isCaseSensitive(). Foi testado se o construtor guardava os argumentos que foram passados a este e obteve-se os resultados esperados.

```
@Test
public void testByTextFilterContent1() {
    ByTextFilter filt = new ByTextFilter("Hello", false);
    assertEquals("HELLO",filt.getText());
    assertFalse(filt.isCaseSensitive());
}

@Test
public void testByTextFilterContent2() {
    ByTextFilter filt = new ByTextFilter("Hello", true);
    assertEquals("Hello",filt.getText());
    assertTrue(filt.isCaseSensitive());
}
```

ContextParser.java:

Não tiveram de ser criados novos testes porque já se tinha mution score de 100%.

HiddenFilter.java e HiddenParser.java:

Não tiveram de ser criados novos testes porque já se tinha mution score de 100%.

LinkParser.java:

Foram criados 2 novos testes:

1. Testar o caso em que se tentar fazer parse de um URL com valor null. Neste caso, vai ser retornada uma lista vazia. Obteve-se o resultado esperado.
2. Caso em que se passa como parâmetro um URL válido. Tal como era esperado, obteve-se uma lista que contém o respetivo URL.

```
@Test
public void testLinkParserNull() {
    List<URL> actual = LinkParser.getInstance().parse(null);
    List<URL> expected = new ArrayList<URL>();
    assertEquals(expected, actual);
}

@Test
public void testTest() throws MalformedURLException {
    List<URL> actual = LinkParser.getInstance().parse("https://www.youtube.com/");
    List<URL> expected = new ArrayList<URL>();
    URL link = new URL("https://www.youtube.com/");
    expected.add(link);
    assertEquals(expected, actual);
}
```

MailAddressParser.java:

Foram criados 2 novos testes:

1. Testar o caso em que se tentar fazer parse de um endereço de email com valor null. Neste caso, vai ser retornada uma lista vazia. Obteve-se o resultado esperado.
2. Caso em que se passa como parâmetro um endereço de email válido. Tal como era esperado, obteve-se uma lista que contém o respetivo email.

```
@Test
public void tesMailAddressParserNull() {
    List<String> actual = MailAddressParser.getInstance().parse(null);
    List<String> expected = new ArrayList<>();
    assertEquals(expected, actual);
}

@Test
public void testMailAddressParserValid() throws MalformedURLException {
    List<String> actual = MailAddressParser.getInstance().parse("RandomEmailAccount123@gmail.com");
    List<String> expected = new ArrayList<String>();
    String mail = "RandomEmailAccount123@gmail.com";
    expected.add(mail);
    assertEquals(expected, actual);
}
```

Priority.java:

Foram criados 4 novos testes:

- Foi testado se as funções inListFormat() e inDetailFormat() devolviam o resultado correto e obteve-se o resultado esperado
- Foi testado se o range em que as duas pontas eram a mesma Priority devolvia uma lista com essa Priority, obteve-se o resultado esperado
- Foi testado se não era devolvida nenhuma Priority (Priority.NONE) ao passar como argumento uma string que não exista em nenhum dos enumerados da classe Priority, foi obtido o resultado esperado.

```
@Test
public void testInListFormat() {
    String expected = "A";
    String actual = Priority.A.inListFormat();
    assertEquals(expected,actual);
}

@Test
public void testInDetailFormat() {
    String expected = "A";
    String actual = Priority.A.inDetailFormat();
    assertEquals(expected,actual);
}

@Test
public void testPrioritySmallRangeInCode() {
    List<String> expected = new ArrayList<>();
    expected.add("C");
    assertEquals(expected,Priority.rangeInCode(Priority.C,Priority.C));
}

@Test
public void testToPriorityWithUnsualString() {
    Priority actual = Priority.toPriority("Ola");
    Priority expected = Priority.NONE;
    assertEquals(expected,actual);
}
```

PriorityTextSplitter.java:

Foi necessário criar um teste para o caso em que se tenta fazer split de null. Obteve-se o resultado esperado.

```
@Test
public void testPrioritySplitResultNull() {
    PriorityTextSplitter splitter = PriorityTextSplitter.getInstance();
    PriorityTextSplitter.PrioritySplitResult actual = splitter.split(null);
    PrioritySplitResult expected = new PrioritySplitResult(Priority.NONE, "");

    boolean res = false;
    if(actual.priority.equals(expected.priority) && actual.text.equals(expected.text)) {
        res = true;
    }
    assertTrue(res);
}
```

Task.java:

Maior parte dos mutantes que se encontravam vivos situavam-se na função de hashcode. Para testar o valor devolvido pela função foi criada uma função auxiliar para calcular o hash de um objeto task seguindo a mesma lógica encontrada dentro da classe. Foi criado um teste com a criação de uma task básica com id 0 e com rawText e defaultPrependedDate a null, e foi criado outro teste com uma task que já contem outros atributos como por exemplo o contexto, projetos da task e a prioridade. Foram obtidos os resultados esperados.

```
@Test
public void testTaskHashcode4() {
    Task task = new Task(0, null, null);
    int expected = computeTaskHashCode(task);
    int actual = task.hashCode();
    assertEquals(expected,actual);
}

@Test
public void testTaskHashcode5() throws ParseException {
    Task task = new Task(73, "Task @C1 +C1 http://java.sun.com someone@gmail.com", getMockDate("2000-01-01"));
    task.markComplete(getMockDate("2000-05-01"));
    task.setPriority(Priority.A);
    task.delete();
    int expected = computeTaskHashCode(task);
    int actual = task.hashCode();
    assertEquals(expected,actual);
}

private int computeTaskHashCode(Task t) {
    final int prime = 31;
    int result = 1;
    result = prime * result + (t.isCompleted() ? 1231 : 1237);
    result = prime * result + ((t.getCompletionDate() == null) ? 0 : t.getCompletionDate().hashCode());
    result = prime * result + ((t.getContexts() == null) ? 0 : t.getContexts().hashCode());
    result = prime * result + (t.isDeleted() ? 1231 : 1237);
    result = prime * result + (int) (t.getId() ^ (t.getId() >>> 32));
    result = prime * result + ((t.getLinks() == null) ? 0 : t.getLinks().hashCode());
    result = prime * result + ((t.getMailAddresses() == null) ? 0 : t.getMailAddresses().hashCode())
            + ((t.getPhoneNumbers() == null) ? 0 : t.getPhoneNumbers().hashCode());
    result = prime * result + ((t.getPrependedDate() == null) ? 0 : t.getPrependedDate().hashCode());
    result = prime * result + ((t.getPriority() == null) ? 0 : t.getPriority().hashCode());
    result = prime * result + ((t.getProjects() == null) ? 0 : t.getProjects().hashCode());
    result = prime * result + ((t.getRelativeAge() == null) ? 0 : t.getRelativeAge().hashCode());
    result = prime * result + ((t.getText() == null) ? 0 : t.getText().hashCode());
    return result;
}
```

Resultados finais:

Após conceber todos os testes previamente descritos, obteve-se os seguintes resultados:

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
34	92%	1161/1261	80%	579/723	86%	579/673

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
com.todotxt.todotxttouch.task	25	92%	802/870	77%	343/445	84%	343/407
com.todotxt.todotxttouch.task.sorter	2	100%	105/105	92%	70/76	92%	70/76
com.todotxt.todotxttouch.util	7	89%	254/286	82%	166/202	87%	166/190

com.todotxt.todotxttouch.task

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
25	92%	802/870	77%	343/445	84%	343/407

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
AndFilter.java	100%	10/10	100%	4/4	100%	4/4
ByContextFilter.java	93%	14/15	90%	9/10	100%	9/9
ByPriorityFilter.java	91%	10/11	86%	6/7	100%	6/6
ByProjectFilter.java	93%	14/15	90%	9/10	100%	9/9
ByTextFilter.java	100%	16/16	92%	12/13	92%	12/13
ContextParser.java	93%	14/15	100%	4/4	100%	4/4
FilterFactory.java	93%	14/15	81%	13/16	81%	13/16
HiddenFilter.java	100%	2/2	100%	2/2	100%	2/2
HiddenParser.java	100%	5/5	100%	3/3	100%	3/3
JdotxtTaskBagImpl.java	91%	122/134	61%	33/54	67%	33/49
LinkParser.java	88%	15/17	100%	4/4	100%	4/4
LocalFileTaskRepository.java	84%	48/57	76%	19/25	76%	19/25
MailAddressParser.java	100%	13/13	100%	4/4	100%	4/4
OrFilter.java	92%	11/12	71%	5/7	83%	5/6
PhoneNumberParser.java	100%	5/5	100%	1/1	100%	1/1
Priority.java	100%	61/61	96%	23/24	96%	23/24
PriorityTextSplitter.java	100%	14/14	100%	6/6	100%	6/6
ProjectParser.java	93%	14/15	100%	4/4	100%	4/4
RecParser.java	100%	13/13	100%	4/4	100%	4/4
Task.java	90%	195/217	75%	123/164	87%	123/142
TaskBagFactory.java	67%	2/3	100%	1/1	100%	1/1
TaskBagImpl.java	90%	122/136	59%	33/56	67%	33/49
TextSplitter.java	100%	43/43	100%	10/10	100%	10/10
ThresholdDateFilter.java	100%	5/5	80%	4/5	80%	4/5
ThresholdDateParser.java	95%	20/21	100%	7/7	100%	7/7

com.todotxt.todotxttouch.task.sorter

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	100%	105/105	92%	70/76	92%	70/76

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
GenericSorter.java	100%	8/8	100%	5/5	100%	5/5
Sorters.java	100%	97/97	92%	65/71	92%	65/71

com.todotxt.todotxttouch.util

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
7	89%	254/286	82%	166/202	87%	166/190

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
CursorPositionCalculator.java	91%	10/11	75%	9/12	75%	9/12
Path.java	92%	12/13	100%	11/11	100%	11/11
RelativeDate.java	96%	24/25	86%	24/28	86%	24/28
Strings.java	98%	43/44	95%	37/39	95%	37/39
TaskIo.java	80%	49/61	78%	29/37	88%	29/33
Tree.java	100%	35/35	100%	19/19	100%	19/19
Util.java	84%	81/97	66%	37/56	77%	37/48

Pode-se constatar um aumento de 10% de Mutation Score. É importante realçar que não se conseguiu ter um aumento maior devido a mutantes equivalentes e, caso o cálculo do Mutation Score excluísse estes mutantes, seria possível observar uma melhoria muito maior. Também houve um conjunto de testes que não foram realizados por fazerem parte de funções.

Em todos os packages observou-se um aumento significativo:

- com.todotxt.todotxttouch.task: 66% -> 77%
- com.todotxt.todotxttouch.task.sorter: 86% -> 92%
- com.todotxt.todotxttouch.util: 72% -> 82%