Classe TestPath

Descrição:

O método fileName não contém documentação, por isso analisámos a implementação e deu para perceber que este método tem como objetivo filtrar o caminho para um ficheiro, de modo a retornar só o nome do mesmo.

Escolhemos esta função pois recebe um parâmetro String que será filtrado e que será usado para criar o nome de um ficheiro. Assim, devido às restrições que são impostas pelos sistemas operativos a nomes de ficheiros, decidimos testa a validade deste método.

Parâmetro e características:

- É uma String que corresponde ao caminho de um ficheiro e têm como características:
    - Nao pode ser empty nem null
    - Nao pode ser só espaços
    - O caminho está dependente do sistema operativo. Em linux, o File.Separator é forward slash (“/”) e em Windows é backward slash (“\”).

Testes JUnit:

Então, de modo a averiguar a validação do software feito, procedemos à realização de testes usando a framework JUnit. Testou-se:

- Se dando o parâmetro como null, retornaria o nome do ficheiro como vazio. Passou no teste.
- Se dando o parâmetro como String vazia, retornaria também uma string vazia. Passou no teste.
- Se o caminho for um conjunto de nomes separados por backward slash (Teste para sistema operativo Windows) , retorna o último nome. Chumbou no teste
- Se o caminho for um conjunto de nomes separados por forward slash (Teste para sistema operativo Linux) , retorna o último nome. Passou no teste
- Se o caminho dado tinha espaços no meio dos nomes. Passou no teste.

Conclusões dos testes:

Este método cumpre os requisitos normais de um caminho de um ficheiro numa diretoria, retornando o nome do ficheiro. Contudo, para sistema Operativo WIndows, retorna o caminho do ficheiro em vez do nome do mesmo, sendo claramente um bug que pode afetar o funcionamento do sistema.

Classe TestStrings

Descrição:

O método insertPaddedIfNeeded têm como objetivo inserir uma string dentro de outra, numa determinada posição, e devolver o resultado da concatenação das mesmas.

Escolhemos esta função devido à inter-dependência dos 3 parâmetros. Devido à quantidade de combinações possíveis e restrições de input, achámos por bem testar este método.

Parâmetros e características:

- String inicial que corresponde à string na qual será inserida outra e têm como características:
    - Não pode ser null
- Índice onde será inserida a string. Têm como características:
    - Não pode ser um valor negativo nem maior que o tamanho do primeiro parâmetro
- String que será inserida na primeira. Têm como características:

    - Não pode ser null nem vazia.
    - Não pode estar presente no primeiro parâmetro

Testes JUnit:

Então, de modo a averiguar a validação do software feito, procedemos à realização de testes usando a framework JUnit. Testou-se:

- Se dando o primeiro parâmetro como null, o resultado final seria a segunda string. Contudo, detectou-se um bug. o programa devolve um NullPointerException quando se vai buscar o tamanho do primeiro parâmetro.
- Também se verificou que, dando o terceiro parâmetro como null, retorna a primeira string e, por isso, passou no teste.
- Dando alguma das strings como vazias, o programa, com sucesso, retorna a outra, com o espaçamento necessário.
- Caso o índice seja negativo ou maior que o tamanho da primeira string, o programa, com sucesso, retorna, como anunciado na documentação, retorna IndexOutOfBoundsException.
- Também se testou se, com os inputs corretos, o resultado era o esperado. Passou com sucesso.
- Também se averiguou que, para um índice maior que zero e menor que o tamanho da primeira string, o programa fazia a concatenação corretamente. O teste passou com sucesso.

Conclusões dos testes:

Este método cumpre com o que se propõe, como verificado nos testes. Contudo, verificou-se que, para o primeiro parâmetro null, o programa contempla este tipo de input e por isso “rebenta”.

Classe TestRelativeDate

Descrição:

O método getRelativeDate, de acordo com a documentação, devolve uma String com a data relativa, comparando a data dada como parâmetro ao dia de hoje.

Devido à particularidade do tipo do parâmetro e às comparações feitas entre o mesmo e a data atual, decidimos testar as características deste tipo de parâmetro e, que por sua vez, complementa a panóplia de parâmetros analisados neste relatório.

Parâmetros e características:

- O único parâmetro deste método é um objeto do tipo Calendar que representa uma data no calendário. Têm como característica:
    - Não poder ser null

Testes JUnit:

Então, de modo a averiguar a validação do software feito, procedemos à realização de testes usando a framework JUnit. Testou-se:

- Tendo em conta as características do Calendar dado, fez-se testes para:
    - Uma data correspondente a “ontem”;
    - Uma data correspondente a alguns dias, mas inferior a um mês de diferença relativamente a “hoje”;
    - Uma data correspondente a um mês;
    - Uma data correspondente a alguns meses;
    - Uma data correspondente a um ano;
    - Uma data futura;
- Todos os testes anteriormente mencionados passaram com sucesso.

- O parâmetro como null. Ao contrário do esperado, o programa crashou quando, no método, é calculado o número de milisegundos do parâmetro, resultando em NullPointerException. O teste chumbou.

Conclusões dos testes:

Este método cumpre os requisitos normais do cálculo relativo de uma data, relativamente ao dia de hoje. O único bug detectado é, caso seja dado um parâmetro null, o programa crashar devido a um NullPointerException**.**

Este método não está devidamente documentado para testes de blackboxing, em que os testers não sabem como é que foi feita a implementação, visto que o output da função recorre a uma função estática que formata datas e, por isso, só indo ver ao código qual o resultado esperado, é que dá para comparar os mesmos.

CursorPositionCalculator:

Descrição:

Função utilizada para calcular a nova posição de cursor quando existe uma mudança relativa ao input (String) da aplicação.

Parâmetros e características:

Esta função recebe três argumentos:

- Posição inicial do cursor *(Int)* **(Varia entre 0 e o tamanho do input antes da mudança);**
- Input antes da mudança mencionada *(String)* **(Que pode ser qualquer input que o utilizador deseje, caso não existisse nenhum, seria null)**;
- Input após a mudança mencionada *(String*) **(Que pode ser qualquer input que o utilizador não deseje, caso o input seja apagado, será null).**

Testes JUnit:

Começámos por testar com tamanhos negativos, de modo a garantir que uma má execução da função não terminaria a execução do programa:

  - Caso de receber Strings “null”
  - Caso de receber duas Strings válidas, mas posição de cursor inválida;
  - Caso de receber uma String que foi inserida, e uma posição do cursor superior ao tamanho da palavra.
  - Caso a segunda frase seja inexistente.
  - No caso de receber duas palavras, mas uma posição de cursor inválida.

Em todas as situações testadas, o programa funciona como esperado, ignorando posições negativas (considerando-as como 0), e posições acima do tamanho da palavra (considerando-as como o tamanho da String existente).

Conclusões dos testes:

A função está programada para estar protegida de todos os casos, mesmo para input que (em teoria) a função não deveria ser capaz de aceitar e trabalhar com.

Tree:

Descrição:

Tree é uma classe utilizada para guardar informação introduzida. É uma classe sem documentação, no entanto, as suas funções e organização são relativamente semelhantes a outras implementações da estrutura.

Foram, assim, testados variados métodos da classe.

Parâmetros e características:

- A estrutura aceita um tipo genérico <E> que pode ser utilizado daí em diante pela classe.
- As funções mais importantes desta classe são “addChild(E data), contains(E data) e getChild()

Testes JUnit:

Para testar a classe, começámos por fazer uma função de setup para ser utilizada antes de qualquer teste ser corrido, entre os quais são:

- Caso de ir buscar informação acabada de inserir e compará-la a ela própria;
- Caso de receber um filho inserido;
- Caso de ir buscar lista de filhos antes da dar loading;
- Caso de ir buscar lista de filhos depois de dar loading;
- Caso de procurar informação guardada na root. **(Contains retornou ‘False’ - Erro).**
- Caso de procurar informação guardada nos filhos;
- Caso de procurar informação guardada em profundidades inferiores (>=2). **(Contains retornou ‘False’ - Erro)**;
- Caso de procurar filhos fora do limite permitido.

Conclusões dos testes:

Implementação extremamente primitiva da estrutura de árvore. Não oferece nada de novo para além das já existentes, e transmite informação errada ao utilizador, indicando que elementos inseridos não existem na árvore (não procurando na root nem nas profundidades superiores a 1), permite mexer manualmente nos filhos (passando-os em getters), entre outros. Claramente a classe que mais precisa de uma adaptação.