Trabalho 1 – Grupo 12: Alexandre Monteiro - 51023 / João Afonso - 51111:

Descrição do projeto:

O projeto aqui em causa é o jdotxt. Este projeto corresponde a uma ferramenta para gerir uma lista de afazeres. Toda a informação é armazenada em 2 ficheiros txt (localmente).

O projeto contém 3 source folders:

1. src/main/java – Contém o próprio código e é formado pelos seguintes pacotes:
    a. jdotxt – Inicialização do programa
    b. jdotxt.gui – Interface gráfica
    c. jdotxt.gui.controls – Tratar do input para a interface
    d. jdotxt.gui.utils – Utils para converter entre mapa e string

    (Nota: Os seguintes pacotes têm ficheiros que fazem parte do Todo.txt Touch)

    e. todotxttouch – Constantes
    f. todotxttouch.task – Tratar de input/filtrar
    g. todotxttouch.task.sorter – Ordenar de strings e listas
    h. todotxttouch.util – Funções genéricas que são usadas por vários ficheiros
2. src/main/resources – Recursos associados com a interface gráfica
3. src/test/java – Testes que se realizam para testar o código

Testes estáticos:

Técnica de teste de software que permite encontrar as faltas existentes numa aplicação através da análise dos artefactos do programa. Esta análise é feita sem executar o software que está a ser analisado.

Tendo em consideração a tarefa proposta, foi utilizada uma ferramenta automática para fazer a análise estática. Contudo, é de realçar que também existe um outro tipo de teste estático que poderia ter sido utilizado: examinação manual do código (review).

Algumas das características mais importantes da realização de testes estáticos incluem:

- Identificar faltas que podem ter um maior impacto em momentos posteriores de teste;
- Melhor manutenção do código e design;
- Detetar faltas numa fase mais inicial, permitindo aprender a prevenir que as mesmas se possam repetir posteriormente;
- Ferramentas de análise estática tendem a ser mais eficientes do que as de análise dinâmica.

Ferramenta de teste estático utilizada:

Optou-se por utilizar a ferramenta: PMD. Fez-se esta escolha com base nos seguintes fatores:

- Tem uma interface intuitiva, sendo fácil perceber e fazer distinção entre as regras;
- Permite selecionar e ordenar as regras de acordo com a preferência (existem também vários critérios que facilitam ainda mais este processo, permitindo ao utilizador escolher os fatores que quer ter em conta aquando da escolha das regras);
- Tem um conjunto vasto de regras built-in que permitem encontrar vários bugs;
- As explicações dos bugs são intuitivas e incluem exemplos e referências para ajudar os utilizadores a informarem-se melhor sobre os problemas.

Face ao elevado número de bugs que surgiram inicialmente, foi necessário selecionar apenas as regras que considerámos ser mais relevantes:

1. Para começar, selecionou-se apenas as regras cuja linguagem era Java;
2. Removeu-se as regras associadas com a documentação e com multithreading;
3. Removeu-se quaisquer regras que estivessem relacionados com alguma forma de convenção de nomes;
4. Relativamente às regras cuja prioridade era classificada como "importante", decidimos manter apenas as que estavam sujeitas a erros (error prone). Esta decisão foi tomada porque as restantes regras não eram particularmente interessantes e era desejável limitar os bugs apresentados de modo a dar um maior foco aos bugs mais relevante;
5. Aquando da análise do código, surgiram alguns bugs cuja descrição detalhada indicava que se referia a problemas descontinuados. Face a isto, excluímos todos os bugs que surgiam com esta indicação na descrição detalhada.

Relatório gerado:

Cada linha do relatório gerado representa um dos bugs que foram encontrados e contém a seguinte informação:

- Path do ficheiro;
- Linha onde se encontra o bug;
- Nome da regra associado com o bug;
- Breve descrição do bug.

Bugs escolhidos:

1. No ficheiro Util.java, linha 121, tem-se um bug de prioridade "Blocker" que não cumpre a regra "AvoidFileStream". De acordo com a descrição da regra, as classes FileInputStream e FileOutputStream (neste exemplo só está a ser usada a classe FileOutputStream) têm de ser finalizadas e isto vai fazer com que ocorram pausas devido ao overhead gerado pelo garbage collector.
2. No ficheiro PriorityTextSplitter.java, linha 51, tem-se um bug de prioridade "Critical" que não cumpre a regra "AvoidReassigningParameters". De acordo com a descrição da regra, não é apropriado atribuir novos valores aos parâmetros de entrada.
3. No ficheiro SortUtils.java, linha 17, tem-se um bug de prioridade "Urgent" que não cumpre a regra "CompareObjectsWithEquals". De acordo com a descrição da regra, deve-se de comparar referências a objetos utilizando equals(). Ao utilizar "==" está-se a averiguar se as duas referências em causa correspondem ao mesmo objeto. Por sua vez, o equals() é usado para verificar se as duas referências que estão a ser comparadas têm as mesmas propriedades (isto é algo que depende da própria classe).
4. No ficheiro Sorters.java, linha 198, tem-se um bug de prioridade "Urgent" que não cumpre a regra "UnusedAssignment". De acordo com a descrição da regra, o valor que era inicialmente atribuído à variável result nunca ia ser utilizado porque era logo substituído.
5. No ficheiro Tree.java, linha 80, tem-se um bug de prioridade "Urgent" que não cumpre a regra "OnlyOneReturn". De acordo com a descrição da regra, um método só devia de ter um exit point (no fim do método).

Descrição da correção dos bugs:

1. Para resolver o bug que não cumpria a regra “AvoidFileStream”, substituiu-se a classe FileOutputStream pelo método newOutputStream da classe Files. Desta forma, não se tem de fazer a finalização, ou seja, vai-se prevenir a pausa que era gerada pelo overhead do garbage collector.

```
public static void writeFile(InputStream is, File file)
        throws IOException {
    //FileOutputStream os = new FileOutputStream(file);
    try(OutputStream os = Files.newOutputStream(Paths.get(file.getName()))) {
        int c;
        byte[] buffer = new byte[8192];
        while ((c = is.read(buffer)) != -1) {
            os.write(buffer, 0, c);
        }
        closeStream(os);
    } finally {
        closeStream(is);
        //closeStream(os);
    }
}
```

2. Com o intuito de tratar o bug que não cumpria a regra “AvoidReassigningParameters”, como o valor de text só era alterado dentro do segundo if statement do método, substituiu-se a linha de código em que text era alterado por: return new PrioritySplitResult(priority, priorityMatcher.group(2)) – ou seja, em vez de alterar o valor de text dentro do if statement para poder retornar PrioritySplitResult(priority, text), pode-se retornar logo o PrioritySplitResult dentro do próprio if sem substituir o valor de text.

```
public PrioritySplitResult split(String text) {
    if (text == null) {
        return new PrioritySplitResult(Priority.NONE, "");
    }
    Priority priority = Priority.NONE;
    Matcher priorityMatcher = PRIORITY_PATTERN.matcher(text);
    if (priorityMatcher.find()) {
        priority = Priority.toPriority(priorityMatcher.group(1));
        return new PrioritySplitResult(priority, priorityMatcher.group(2));
        //text = priorityMatcher.group(2);
    }
    return new PrioritySplitResult(priority, text);
}
```

3. Com o intuito de tratar o bug que não cumpria a regra “CompareObjectsWithEquals”, substitui-se o if(s != last) por if(!s.equals(last)).

```
public static String writeSort(List<Map.Entry<Sorters, Boolean>> sortList) {
    Map.Entry<Sorters, Boolean> last = sortList.get(sortList.size()-1);
    StringBuilder sb = new StringBuilder();
    for (Map.Entry<Sorters, Boolean> s: sortList) {
        sb.append(s.getKey().name());
        sb.append(":");
        sb.append(s.getValue().toString());
        if (!s.equals(last)) // Antes a condição era "s != last"
            sb.append("|");
    }
    return sb.toString();
}
```

4. Com o intuito de tratar o bug que não cumpria a regra “UnusedAssignment”, altera-se a primeira linha do método de modo a não inicializar a variável result.

```
public static <E extends Comparable> int compareDates(E d1, E d2, final boolean ascending) {
    int result;

    // Similar to priorities -- we want tasks with threshold date go first.

    if      (d1 == null && d2 == null) { result =  0; }
    else if (d1 == null              ) { result =  1; }
    else if (d2 == null              ) { result = -1; }
    else {
      result = d1.compareTo(d2);
    }

    if (!ascending)
      result = -result;

    return result;
}
```

5. Com o intuito de tratar o bug que não cumpria a regra “OnlyOneReturn”, criou-se uma variável res do tipo boolean que foi inicializada a true. Alterou-se o if statement de modo que, caso children estivesse a null ou children não contivesse child, então res seria alterado para false. Por fim, retorna-se o res.

```
public boolean contains(Tree<E> child) {
    boolean res = true;
    if (children == null || !children.contains(child)) {
        res = false;
    }

    return res;
}
```