Avaliação 4 - Log do TDD

Rebeca Portes Stroh - 186407

O problema foi testado e resolvido seguindo as seguintes etapas:

1. Entendi o foco principal do programa:
   O foco é identificar arquivos duplicados.
   Pelos exemplos, o que difere um arquivo do outro é o conteúdo apenas.

2. Pensei em alguns casos a partir da definição. Assim como alguns casos limites:
   - Não existem arquivos na pasta => não duplicados
   - Um só arquivo => nunca é duplicado
   - Um só diretório vazio => nunca é duplicado
   - Dois arquivos vazios na mesma pasta => duplicados
   - Dois arquivos com mesmo conteúdo na mesma pasta => duplicados
   - Dois arquivos com dif conteúdos na mesma pasta => não duplicados
   - Dois arquivos vazios em pastas dif => duplicados
   - Dois arquivos com mesmo conteúdo em pastas dif => duplicados
   - Dois arquivos com dif conteúdos em pastas dif => não duplicados
3. Criei o primeiro teste, o mais simples e básico possível:
   Não existem arquivos na pasta => não duplicados
   `testEmptyList`

4. Rodei e verifiquei que deu erro (obviamente, pois não tem nem classe ainda).
5. Fiz o suficiente para este primeiro teste passar, criando:

```
public class FindDuplicate {
    public FindDuplicate() {

    }

    public List<List<DirOrFile>> findDuplicate(List<DirOrFile>
dirfiles) {
        return Collections.emptyList();
    }
}
public class DirOrFile {
    public DirOrFile() {

    }
}
```

6. Criei before all e outro teste:

```
public static FindDuplicate FD;

@BeforeAll
static void setUpBeforeClass() throws Exception {
    System.out.print("Configurando os testes ... ");

    FD = new FindDuplicate();
```

```
    System.out.println("OK");
}
```

```
testOneFileEmpty
```

7. Fiz ele passar criando:
```
public DirOrFile(String fileName, String content)
private String fileName;
private String content;
```

getters e setters

8. Criei outro teste:
```
testOneSubDirEmpty
```

9. Fiz ele passar criando:
```
public DirOrFile(String fileName, List<DirOrFile> myFiles)
private List<DirOrFile> myFiles;
```

getters e setters

10. Criei outro teste:
```
testOneSubDirWithOneFileEmpty
```

11. Fiz ele passar criando: PASSOU

12. Criei outro teste:
```
testTwoEmptyFiles
```

13. Fiz ele passar criando:
```
public List<List<DirOrFile>> findDuplicate(List<DirOrFile> dirfiles) {
    List<List<DirOrFile>> result = new ArrayList<>();

    result.add(dirfiles);

    return result;
}
```

14. O que fez com que outros testes falhassem, logo, refiz essa funcao
```
public List<List<DirOrFile>> findDuplicate(List<DirOrFile> dirfiles) {
    List<List<DirOrFile>> result = new ArrayList<>();

    for (int i=0; i < dirfiles.size()-1; i++) {
        for (int j=i+1; j < dirfiles.size(); j++) {
            if (dirfiles.get(i).getContent() ==
dirfiles.get(j).getContent()) {
                List<DirOrFile> duplicatedFiles = new ArrayList<>();
                duplicatedFiles.add(dirfiles.get(i));
```

```
                duplicatedFiles.add(dirfiles.get(j));
                result.add(duplicatedFiles);
            }
        }
    }

    return result;
}
```

15. Criei outro teste:
    `testTwoDifFiles`

16. Fiz ele passar criando: PASSOU

17. Criei outro teste:
    `testSubDirAndFileEmpties`

18. Fiz ele passar criando: PASSOU

19. Criei outro teste:
    `testSubDirWithTwoEmptyFiles`

20. Fiz ele passar criando a função auxiliar:

```
private List<DirOrFile> findFiles(List<DirOrFile> dirfiles) {
    List<DirOrFile> result = new ArrayList<>();

    for (int i=0; i < dirfiles.size(); i++) {
        if (dirfiles.get(i).getContent() != null) { // is a file
            result.add(dirfiles.get(i));
        } else { // is a folder
            List<DirOrFile> resultFolder =
findFiles(dirfiles.get(i).getMyFiles());
            result = Stream.concat(result.stream(),
resultFolder.stream())
                    .collect(Collectors.toList());
        }
    }

    return result;
}
```

21. Criei outro teste:
    `testTheeSubDirsWithFileEmpties`