Instituto Superior Técnico

Software Testing and Validation

School Year 2023/2024 — 2nd Semester

1st Exam **A** - June 19, 2024 — Duration: 1:45 hours

- Clearly identify the first page of this test.

I. (1.5+0.5+0.5+0.75+0.75+1.0 = 5.0 val.)

a) Draw the control flow graph for the following method, clearly identifying its several code segments and the several *du-pairs* and corresponding *dc-paths* for the x variable. It may be useful to show in each segment if the variable was defined and/or used. To simplify the analysis, you can consider that each segment should contain a single statement.

```
public int doSomething(int y) {
  int x = g(y);

  while (x != 0 && y > 5) {
    y = h(x);
    x = g(y);
  }

  if (y > 50)
    x = g(y);

  return h(x + y);
}
```

b) Show a minimal set of paths that achieves statement coverage for this method.

____

____

____

____

____

c) Show a minimal set of paths that achieves branch coverage for this method.

____

____

____

____

____

d) Show a minimal set of paths that achieves *All-Defs* coverage for this method, concerning just variable x.

e) Show a minimal set of paths that achieves *All-Uses* coverage for this method, concerning just variable x.

f) Consider two test suites, *TC1* and *TC2*. *TC1* achieves a **branch** coverage equal to 100% while *TC2* achieves a **statement** coverage equal to 90%. If possible, indicate which of the two test suites achieves greater **statement** coverage? Justify your response.

II. (5.5 val.)

Consider the following class:

```
public class ParkingAccount {
  // ...
  public ParkingAccount(String licensePlate, float balance) throws InvalidOperationException { ... }

  public void setLicensePlate(String newLicense) throws InvalidOperationException { ... }
  public String getLicensePlate() {  ... }

  // add amount to the balance of this parking account
  public void addToBalance(int amount) InvalidOperationException { ... }
  public float getBalance() { ... }

  public void setFidelityNumber(int newFidel) InvalidOperationException { ... }
  public int getFidelityNumber() { ... }

  public void payParking(float amount) { ... }
}
```

This class represents a person's account in a parking management system. Each instance of this class has a license plate and maintains the balance and level of customer loyalty. The loyalty level is an integer ranging from 0 to 100. The balance is a number with two decimal places and can vary between 0 and $50.00 + fidel$, where $fidel$ represents the level of customer loyalty. Finally, the license plate is a string of 6 characters, and it is a unique identifier (you cannot have two accounts (represented by two instances different from `ParkingAccount`) with the same license plate). If the invocation of a method of this class invalidates one of these conditions, then the method should not have any effect and it should throw the *InvalidOperationException* exception.

Draw the test suite that checks the behavior of this class using the appropriate class scope test pattern. Describe the various steps of the applied test pattern. Implement two **successful** test cases and one **failure** test case from this test suite. The method under test in the **failure** test case cannot be the constructor of this class.

III. (5.0 val.)

a)

Consider the following application domain. A file has a name and a certain content (represented by a string of characters) and is represented by the `File` class. The `getContent()` method of this class returns the content of the file. Two files are duplicates if their content is the same. The `DirUtils` class offers some functionalities associated with The partial interface of these two classe is the following:

```
public class File {
  public File(String name, String content) { ... }
  public String getContent() { ... }
  public boolean isDuplicate(File f) { ... }
}
```

```
public class DirUtils {
  // ...
  public List<File> listFiles(String dir)
     throws InvalidArgumentException;
}
```

One of the features supported by the `DirUtils` class, implemented by method `public List<File> listFiles(String dir)`, consists of getting all the files existing in a given directory. If the directory represented by the *dir* argument does not exist, then the method throws the exception `InvalidArgumentException`. If the directory to be listed does not have any file, then this method returns an empty list.

Consider now that you have already developed the `FileUtils` class. This class offers some functionality to find duplicates of a file in a given file system (represented by an instance of `DirUtils`). Each `FileUtils` still holds the number of duplicates that it has already found:

```
public class FileUtils {
  private DirUtils dirUtils;
  private int numberDuplicates;

  public Fileutils(DirUtils dirUtils) { ... }
  public List<File> getDuplicates(File file, String dirName) throws FileException { ... }

  public int getNumberDuplicates() { ... }
}
```

The `getDuplicates(File file, Sting dirName)` method is responsible to determine the files existing in the directory represented by `dirName` that are duplicates of the file represented by `file`. This method is implemented based on the functionality offered by the `listFiles` method of `DirUtils`. This method should have the following functionality: if both arguments are valid, the method returns the files present in the indicated directory that are duplicates of the given file and it also updates the number of duplicates already found. When this method does not find any duplicated file, then it returns an empty list. If the file argument is valid and the specified directory does not exist, then the method should return `null`. If the specified file is invalid (it is the null reference), then this method should throw the `FileException` exception. The implementation of this method should avoid unnecessary invocations on the object `DirUtils`. The constructor of the `FileUtils` class is responsible for associating the supplied `DirUtils` object with the created object.

Implement three test cases that exercise this method in the following situations: (i) Directory with several files, but only some of them are duplicates of the given file; (ii) directory not existing; (iii) existing directory but the specified file is invalid. In the implementation of these test cases, the class `FileUtils` must be exercised in isolation from the class `DirUtils` and you can use the `TestNG` and `mockito` frameworks. You can consider that all the exception classes and the `File` class are already implemented.

IV. (2.0 + 1.5 + 1.0 = 4.5 val.)

a) Describe the *Static Analysis* technique and how the tool *ESC/Java* (describe in the paper *Extended Static Checking for Java* implemented this technique. Show the main advantages and disadvantages of this technique.

b) Compare the *Random Testing* and *Mutation Testing* techniques, briefly explaining how each one works.

c) Consider that you have two classes, `SuperA` and `SubA`, where `SubA` is a subclass of `SuperA`. `SuperA` defines the method `public void doSomething(int i)`. `SubA` does not override this method (that means `SubA` just inherites this method). `SuperA` was tested both at class and method scope. What is the impact of this inherited method on the tests that need to be implemented (at class-scope and method-scope) for class `SubA`?