Instituto Superior Técnico

Teste e Validação de Software

Ano Lectivo de 2020/2021 — 2º Semestre

2º Teste **A** - 25de Junho de 2021, — Duração: 1:30 horas

- Certifique-se que a sua identificação está legível na primeira folha do enunciado.
- Não utilize folhas de rascunho.
- Escreva as respostas no próprio enunciado.

I. (2.0+0.75+0.75 = 3.5 val.)

a) Desenhe o respectivo grafo de controlo de fluxo do seguinte método, identificando claramente os vários segmentos do código e os vários *du-pairs* e respectivos *dc-paths* para a variável *x*.

```
public int doSomething(int z) {
  int x = f(z);
  int a = 0;

  if (x > 0) {
    a = x + 1;
    x = f(a);
  }

  if (a > 0)
    z = 5;
  else
    z = -4;

  if (z > 0) {
    a = f(a) + z;
    z = h(a);
  } else
    x = h(a + z);

  return x * z;
}
```

b) Indique um conjunto mínimo de caminhos completos que garanta a cobertura *All-Uses* para a variável *x*.

c)Indique um conjunto mínimo de caminhos completos que garanta a cobertura *All DU-Paths* para a variável *x*.

Número:_____ Nome:_____

II. (3.5 val.)

Considere o seguinte diagrama de classes

que representa a associação entre as entidades *User* e *Tweet* numa dada aplicação que é responsável por gerir a informação respeitante aos *tweets* feitos por cada utilizador. Cada utilizador (representado pela classe *User*) tem um dado *nickname*. Cada *tweet*, representado pela classe *Tweet*, tem um determinado texto e está associado a um único utilizador, designado como autor, que é o utilizador que criou o *tweet*. A interface destas classes é a seguinte:

```
public class User {
   /*...*/
  public User(String nickname)  {/*...*/}

  // changes the nickname of this user
  public void  setNickname(String nickname)
  {/*...*/}

  // add the tweet to this user
  public void add(Tweet tw)  {/*...*/}

  // returns all existing University instances
  static User[] getAllInstances() {/*...*/}
}
```

```
public class Tweet {
  /*...*/
  public Tweet(String text, User author) {/*...*/}

  // returns the nickname of the author that made
  // this tweet
  public String getAuthorNickname() { /*...*/ }

  // removes this tweet from the system
  public void destroy() { /*...*/ }

  // returns all existing tweets instances
  static Tweet[] getAllInstances() {/*...*/}
}
```

Considere ainda que não é possível remover do sistema instâncias de *User*, mas é possível remover do sistema um dado tweet através do método *destroy* presente na classe `Tweet`. Quando se remove um tweet, o seu autor permanece no sistema, não é removido. Aplicando o padrão de teste correcto, desenhe a correspondente bateria de testes que verifica a correcta concretização da associação existente entre estas duas entidades.

III. (6 val.)

a) Considere o seguinte domínio de aplicação. Suponha que quer fazer um cliente do *Twitter* que recebe *tweets* feitos por utilizadores e submete-os depois ao servidor que será responsável por disseminar os *tweets* pela rede social. Cada *tweet* e utilizador são representados, respectivamente, pelas classes `Tweet` e `User`:

```
public class Tweet  {
    private String message;

    public Tweet(String m) { message = m; }
    public String getMessage() { return message; }
}
```

```
public class User  {
    private String username;

    public User(String u) { username = u; }
    public String getUsername() { return username; }
}
```

A funcionalidade do servidor *Twitter* é representada pela seguinte entidade:

```
public interface ITwitterServer {
       boolean submit(Tweet tweet, User user) throws InvalidUserException;
}
```

O método `submit` permite enviar um tweet para o servidor. Este método devolve um boleano que indica se a submissão teve sucesso ou não. O valor `false` é devolvido caso o tweet seja inválido. A submissão pode ainda falhar caso o utilizador não esteja registado no servidor. Neste caso, o método lança a excepção `InvalidUserException`.

Pretende-se então desenvolver a classe que representa a funcionalidade do cliente do *Twitter*. Esta classe mantém o número de tweets (no campo `erros`) que não foram submetidos ao servidor com sucesso. Por razões de auditoria, cada instância desta classe mantém ainda um `log` dos *tweets* submetidos com sucesso. Este *log* consiste numa cadeia de caracteres que representa a concatenação das mensagens dos vários *tweets* submetidos com sucesso. A submissão de um *tweet* ao servidor é realizada através o método `add` desta classe. Adicionalmente, este método tem ainda a responsabilidade de actualizar o `log` e o número de *tweets* submetidos com erro tendo em conta o resultado da submissão do *tweet* ao servidor. Um *tweet* é submetido com sucesso apenas se o método `submit` do servidor devolver o valor `true`. Caso o utilizador seja a referência `null`, então a submissão não deve ser feita e o método devolve `false`. O método `add` devolve um valor boleano que indica se o tweet foi submetido com sucesso ou não. Considere que o estado actual da classe `TwitterClient` é o seguinte:

```
public class TwitterClient  {
    private;ITwitterServer server;
    private String log = "";
    private int errors = 0;

    public TwitterClient (ITwitterServer s) { server = s; }

    public boolean add(Tweet tw, User user) { /* to implement */ }

    public String log() { return log; }
    public int errors() { return errors; }
}
```

Concretize o método `add` e os quatro seguintes casos de teste que exercitam o correcto funcionamento deste método nos seguintes cenários: (i) submissão com sucesso de vários *tweets*; (ii) submissão de vários *tweets*, todos com utilizadores registados, mas um dos *tweets* é inválido; (iii) submissão de um *tweet* válido mas com o utilizador `null`; e (iv) submissão de um *tweet* válido mas com um utilizador não registado no servidor. Na concretização destes casos de teste, a classe `TwitterClient` deve ser exercitada de forma isolada da concretização da entidade `ITwiterServer`, devendo para isso utilizar a framework *JMockit*. Pode considerar que as restantes classes já estão concretizadas.

IV. (1.5 + 1.5 = 3 val.)

a) Descreva o padrão de teste *Sandwich integration testing*, indicando o seu objectivo, estratégia e principais vantagens e desvantagens.

b) Na metodologia *Test-driven development* o programador deve seguir um determinado conjunto de regras. Uma das regras é que a concretização de cada funcionalidade só deve ser feita após se ter um caso de teste relativo à funcionalidade em questão que falha. Quais as vantagens de se seguir esta regra?

V. (2 + 2 = 4 val.)

a) Descreva a técnica de teste *Random Testing*, indicando também as suas principais vantagens e desvantagens. Descreva resumidamente a solução apresentada pelo artigo *ARTOO: Adaptive Random Testing for Object-Oriented Software*, indicando quais as desvantagens desta técnica que são abordadas nesta solução.

b) Descreva a técnica *Mutation Testing*, indicando também as principais vantagens e desvantagens desta técnica. Descreva a aplicação desta técnica indicada no artigo *MuJava : An Automated Class Mutation System*, indicando quais as desvantagens indicadas anteriormente que foram resolvidas na solução apresentada no artigo.