UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CORNÉLIO PROCÓPIO

Lista de Exercícios: Teste Caixa-Branca
GFC, Line Coverage e Branch Coverage

Prof. André Takeshi Endo

- Para todos os exercícios a seguir, elabore casos de teste em JUnit e Mockito de forma a alcançar 100% dos critérios de teste estrutural apresentados.
- Desenhe o grafo de fluxo de controle (GFC) para os alguns métodos.
- Comente os casos de teste com o caminho que foi gerado de acordo com o GFC.
- Use a funcionalidade de cobertura de código do Eclipse para confirmar que os critérios de fluxo de controle foram adequadamente cobertos.

(Exercício 1) Execute as atividades com as classes e métodos implementados até o momento durante a disciplina.

(Exercício 2) Considere os métodos a seguir.

```
public int test01(int a)
{
    return (a > 0) ? 1 : -1;
}

public int test03(int a)
{
    if(a > 10 || a < -10)
        return 0;
    else
        return 1;
}

public int test02(int a)
{
    switch(a)
    {
    case 1: return 1;
    case 2:
        return 2;
    default:
        return 0;
    }
}
```

(Exercício 3) Considere os métodos a seguir.

```java
// return C = A * B
public static double[][] multiply(double[][] A, double[][] B) {
    int mA = A.length;
    int nA = A[0].length;
    int mB = B.length;
    int nB = A[0].length;
    if (nA != mB) throw new RuntimeException("Illegal matrix dimensions.");
    double[][] C = new double[mA][nB];
    for (int i = 0; i < mA; i++)
        for (int j = 0; j < nB; j++)
            for (int k = 0; k < nA; k++)
                C[i][j] += (A[i][k] * B[k][j]);
    return C;
}
```

(Exercício 4) Realize os exercícios de listas anteriores considerando agora teste caixa-branca.

(Exercício 5) Utilize os projetos fornecidos anexos a esta lista.

(Exercício 6) Considere a classe abaixo:

```
public class Calculadora {
   /**
    * @param vetor
    * @param inicioInterv deve ser >= 0 e menor que fimInterv
    * @param fimInterv    deve ser >= 0 e maior que inicioInterv
    * @return (i) media dos inteiros do vetor que estão no intervalo [inicioInterv, fimInterv]
    *             (ii) -1 se os parametros forem invalidos
    */
   public float calcularMedia(int vetor[], int inicioInterv, int fimInterv) {
      if (inicioInterv < 0 || fimInterv < 0)
         return -1;

      if (inicioInterv >= fimInterv)
         return -1;

      float soma = 0, n = 0;
      for (int i = 0; i < vetor.length; i++) {
         if(vetor[i] >= inicioInterv && vetor[i] <= fimInterv) {
            soma = soma + vetor[i];
            n++;
         }
      }

      return soma/n;
   }
}
```

Elabore casos de teste em JUnit de forma a alcançar 100% de cobertura do critério de teste estrutural todos-arcos (branch coverage). Desenhe o grafo de fluxo de controle (GFC) para o método e comente cada caso de teste com o caminho que foi executado de

acordo com o GFC. O desenho do GFC pode ser feito no espaço em branco no final da avaliação ou encaminhada como uma imagem junto com o projeto; enumere os nós para identificar corretamente os caminhos.

(Exercício 8) Considere as duas classes e as três interfaces a seguir:

Usuario.java

```
public class Usuario {
   private String nome, email;

   public String getNome() {
      return nome;
   }
   public void setNome(String nome) {
      this.nome = nome;
   }
   public String getEmail() {
      return email;
   }
   public void setEmail(String email) {
      this.email = email;
   }
}
```

UsuarioDAO.java

```
public interface UsuarioDAO {
   public ArrayList<Usuario> getAllUsuarios();
}
```

ServidorDeEmail.java

```
public interface ServidorDeEmail {
   public boolean enviar(String email);
}
```

Criptografia.java

```
public interface Criptografia {
   public String criptografar(String mensagem);
}
```

Emissario.java

```
public class Emissario {
   private UsuarioDAO usuarioDAO;
   private ServidorDeEmail servidorEmail;
   private Criptografia criptografia;

   public Emissario(UsuarioDAO usuarioDAO, ServidorDeEmail servidorEmail) {
      this.usuarioDAO = usuarioDAO;
      this.servidorEmail = servidorEmail;
   }

   public void setCriptografia(Criptografia criptografia) {
      this.criptografia = criptografia;
   }

   public String enviarPara(ArrayList<String> nomes) {
      if(nomes == null)
         return "nomes nao informados";

      ArrayList<Usuario> usuarios = usuarioDAO.getAllUsuarios();
      if(usuarios == null || usuarios.size() == 0)
         return "nao ha usuarios";

      boolean msgsEnviadas = false;
      for(String nome : nomes) {
         for (Usuario usuario : usuarios) {
            if(usuario.getNome().equals(nome)) {
               String mensagem = criptografia.criptografar("mensagem secreta");
               boolean foiEnviado = servidorEmail.enviar("TO: " + usuario.getEmail() +
                                                         " " + mensagem);
               if(foiEnviado) {
                  msgsEnviadas = true;
                  break;
```

```
                }
                else
                    return "servidor de email offline";
            }
        }
    }

    if(msgsEnviadas)
        return "mensagens enviadas";
    else
        return "usuarios nao encontrados";
  }
}
```

Elabore casos de teste usando JUnit e Mockito de forma a alcançar 100% de cobertura do critério de teste estrutural todos-arcos (*branch coverage*). Desenhe o grafo de fluxo de controle (GFC) para o método “enviarPara(..)” da classe Emissario e comente cada caso de teste com o caminho que foi executado de acordo com o GFC. O desenho do GFC pode ser feito em uma folha em branco ou encaminhada como uma imagem junto com o projeto; enumere os nós para identificar corretamente os caminhos.

(Exercício 9) Implemente casos de teste em JUnit com o intuito de alcançar 100% de cobertura do critério de teste estrutural todos-arcos (*branch coverage*). Desenhe o grafo de fluxo de controle (GFC) para seus métodos e comente cada caso de teste com o caminho que foi executado de acordo com o GFC.

```
/** extraído de G. Fraser and A. Arcuri, “Whole Test Suite Generation,” IEEE Transactions on Software Engineering, vol. 39, iss. 2, pp. 276-291, 2013. **/

public class Stack {
  int[] values = new int[3];
  int size = 0;

  void push(int x) {
    if(size >= values.length)
      resize() ;
   if(size < values.length)
     values[size++] = x;
  }

  int pop() {
    if(size > 0)
      return values[size−−];
    else
      throw new EmptyStackException();
  }

  private void resize(){
    int[] tmp = new int[values.length ∗ 2];
    for(int i = 0; i < values.length; i++)
      tmp[i] = values[i];
    values = tmp;
  }
}
```

(Exercício 10) Considere a classe a seguir:

```
public class Calculadora {
   /**
    * @param n - inteiro
    * @param valorMaximo - valor maximo que pode ter o somatorio
    * @return - o somatorio de 0 ate |n|, caso somatorio seja <= valorMaximo
    * @throws Exception - caso o somatorio seja > valorMaximo
    */
   public int somatoriaLimitada(int n, int valorMaximo) throws Exception {
      int resultado = 0, i = 0;
      if(n < 0) {
         n = -n;
      }

      while(i<=n && resultado <= valorMaximo) {
         resultado = resultado + i;
         i++;
      }

      if(resultado > valorMaximo)
         throw new Exception("valor maximo foi ultrapassado");
      else
         return resultado;
   }
}
```

Elabore casos de teste em JUnit de forma a alcançar 100% de cobertura do critério de teste estrutural todos-arcos (branch coverage). Desenhe o grafo de fluxo de controle (GFC) para o método e comente cada caso de teste com o caminho que foi executado de acordo com o GFC. O desenho do GFC pode ser feito no espaço em branco no final da avaliação ou encaminhada como uma imagem junto com o projeto; enumere os nós para identificar corretamente os caminhos.

(Exercício 12) Considere as duas classes e as duas interfaces a seguir:

Usuario.java

```
public class Usuario {

        private String nome, senha,
senhaConfirmada;

        public String getNome() {
                return nome;
        }

        public void setNome(String nome) {
```

UsuarioDAO.java

```
public interface UsuarioDAO {
        public boolean
                existe(String nomeDoUsuario);
}
```

SenhaValidator.java

```
public interface SenhaValidator {
        public boolean verificar(String senha);
}
```

```
                this.nome = nome;
        }

        public String getSenha() {
                return senha;
        }

        public void setSenha(String senha) {
                this.senha = senha;
        }

        public String getSenhaConfirmada() {
                return senhaConfirmada;
        }

        public void setSenhaConfirmada(String
                                senhaConfirmada)
{
                this.senhaConfirmada =
                                senhaConfirmada;
        }
}
```

UsuarioValidator.java

```
public class UsuarioValidator {
        private UsuarioDAO usuarioDao;
        private SenhaValidator senhaValidator;

        public UsuarioValidator(UsuarioDAO usuarioDao) {
                this.usuarioDao = usuarioDao;
        }

        public void setSenhaValidator(SenhaValidator senhaValidator) {
                this.senhaValidator = senhaValidator;
        }

        public boolean ehUsuarioValido(Usuario u) throws Exception {
                if(u.getNome().length() <= 3)
                        throw new Exception("nome do usuario precisa de pelo menos 3 caracteres");

                if(! u.getSenha().equals(u.getSenhaConfirmada()))
                        throw new Exception("senhas diferentes");

                if(usuarioDao.existe( u.getNome() ))
                        throw new Exception("usuario ja existe");

                if(senhaValidator.verificar( u.getSenha() ))
                        throw new Exception("senha invalida");

                String nome = u.getNome();
                boolean isValid = true;
                for(int i = 0; i < nome.length(); i++) {
                        char atual = nome.charAt(i);
                        if(!Character.isAlphabetic(atual) && !Character.isDigit(atual))
                                isValid = false;
                }
```

```
                return isValid;
        }

}
```

(a) Desenhe o grafo de fluxo de controle (GFC) para o método "ehUsuarioValido(..)" da classe UsuarioValidator. O desenho do GFC pode ser feito em uma folha em branco ou encaminhada como uma imagem junto com o projeto; enumere os nós para identificar corretamente os caminhos.

(b) Elabore casos de teste usando JUnit e Mockito de forma a alcançar 100% de cobertura do critério de teste estrutural todos-arcos (*branch coverage*).

(c) Comente cada caso de teste com o caminho que foi executado de acordo com o GFC.

(Exercício 13) Considere a interface e a classe a seguir:

RedeMovel.java

```
public interface RedeMovel {
    public boolean estaConectado();
    public boolean enviarSMS(String telefone, String mensagem);
}
```

ContadorNaRede.java

```
public class ContadorNaRede {
    RedeMovel rede;

    public ContadorNaRede(RedeMovel rede) {
        this.rede = rede;
    }

    public String enviarNumeros(String telefone, int v[ ]) {
        int npar = 0, nNegativo = 0;
        for (int i = 0; i < v.length; i++) {
            if(v[i] % 2 == 0)
                npar++;

            if(v[i] < 0)
                nNegativo++;
        }
        String msg = "Enviado. pares: " + npar + "; negativos: " + nNegativo;

        if(rede == null || ! rede.estaConectado())
            throw new RuntimeException("Sem internet");

        if(rede.enviarSMS(telefone, msg))
            return msg;
        else
            return "Erro no envio. Verifique.";
    }
}
```

Elabore casos de teste em JUnit de forma a alcançar 100% de cobertura do critério de teste caixa-branca todos-arcos (*branch coverage*). Desenhe o grafo de fluxo de controle (GFC) para o método “enviarNumeros(..)” e comente cada caso de teste com o caminho que foi executado de acordo com o GFC. Enumere os nós para identificar corretamente os caminhos.

(Exercício 15) Considere a interface e a classe a seguir:

CartorioEleitoral.java

```
public interface CartorioEleitoral {

    /** Esse método pode retornar:
     * - "nao existe": se o cpf não possui
          titulo associado
     * - "pendencia": o titulo possui
          alguma pendencia
     * - "OK": situacao regularizada
          para o título  */

    public String verificar(String cpf);

}
```

VerificadorEleitoral.java

```
public class VerificadorEleitoral {
    private CartorioEleitoral cartorioEleitoral;

    public VerificadorEleitoral(CartorioEleitoral cartorioEleitoral) {
        this.cartorioEleitoral = cartorioEleitoral;
    }

    public String consultarSituacao(int idade, String cpf) throws Exception {
        if(idade < 0 || idade > 200) throw new Exception("idade invalida");

        if(cpf == null || cpf.length() != 11) throw new Exception("cpf invalido");

        if(idade < 16) return "nao pode votar";

        String status = cartorioEleitoral.verificar(cpf);
        if(status.equals("nao existe"))
            return "faca um titulo";
        else if(status.equals("pendencia"))
            return "regularize seu titulo";
        else if(status.equals("OK")) {
            String ret = "voto obrigatorio";
            if(idade <= 17 || idade > 70)
                ret = "voto facultativo";

            return ret;
        }
        return "erro desconhecido";
    }
}
```

Elabore casos de teste em JUnit e Mockito de forma a alcançar 100% de cobertura do critério de teste caixa-branca todos-arcos (*branch coverage*). Desenhe o grafo de fluxo de controle (GFC) para o método “consultarSituacao(..)” e comente cada caso de teste com o caminho que foi executado de acordo com o GFC. O desenho do GFC pode ser feito em uma folha em branco ou como uma imagem junto com o projeto; enumere os nós no código para identificar corretamente os caminhos.

(Exercício 17) Considere a classe a seguir:

Lambda.java

```
public class Lambda {
	/**
	 * @param strings
	 * @return um array com 2 inteiros contando quantas palavras
	 * comecam com 'a' e 'z', respectivamente
	 */
	public int[] countWordsStartingWithAandZ(List<String> strings) {
		int[] ret = {0, 0};

		strings.stream().forEach(s -> {
			if(s.startsWith("a"))
				ret[0]++;

			if(s.startsWith("z"))
				ret[1]++;
		});

		return ret;
	}
}
```

(a) Desenhe o grafo de fluxo de controle (GFC) para o método "countWordsStartingWithAandZ(..)" da classe UsuarioValidator. O desenho do GFC pode ser feito em uma folha em branco ou encaminhada como uma imagem junto com o projeto; enumere os nós para identificar corretamente os caminhos.
Obs. *Na construção do GFC, considere o forEach como um loop while.*

(b) Elabore casos de teste usando JUnit de forma a alcançar 100% de cobertura do critério de teste estrutural todos-arcos (*branch coverage*).

(c) Comente cada caso de teste com o caminho que foi executado de acordo com o GFC.