

ECM251 – Linguagens de Programação I

Aula 18 – L1/1, L2/1 e L3/1

Engenharia da Computação – 3ª série

Testes Unitários
(L1/1, L2/1 e L3/1)

2025

ECM251 – Linguagens de Programação I

Aula 18 – L1/1, L2/1 e L3/1

Horário

Terça-feira: 2 x 2 aulas/semana

- L1/1 (07h40min-09h20min): *Prof. Calvetti*;
- L1/2 (09h30min-11h10min): *Prof. Calvetti*;
- L2/1 (07h40min-09h20min): *Prof. Menezes*;
- L2/2 (11h20min-13h00min): *Prof. Calvetti*;
- L3/1 (09h30min-11h10min): *Prof. Evandro*;
- L3/2 (11h20min-13h00min): *Prof. Evandro*.

Testes Unitários

Tópico

- Testes

Testes

Definição



- Teste é um processo que demonstra que algo funciona corretamente;
- No universo dos *softwares*, existem diversos tipos de testes:
 - ✓ Teste de Integração;
 - ✓ Teste Funcional;
 - ✓ Teste de Carga;
 - ✓ Teste de Segurança;
 - ✓ *Smoke Test*;
 - ✓ Teste Unitário;
 - ✓ Etc.

Testes

Tópico

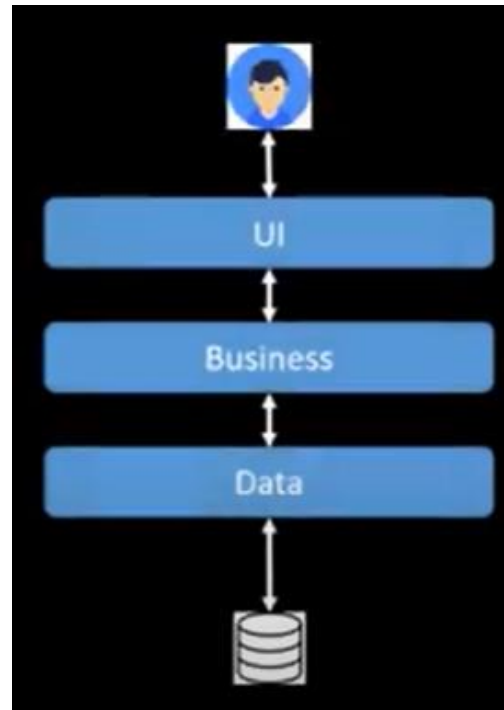
- Testes Funcionais

Testes Funcionais

Definição



- Imaginando que uma aplicação contém 3 (três) camadas: UI, Business e Data:



Testes Funcionais

Definição



- O **Teste Funcional** é quando se entrega a aplicação inteira, ou uma parte significativa da mesma, para seu usuário final testá-la;
- Daí, o usuário fará um teste completo da aplicação ou do módulo entregue a ele, exercitando todas as funcionalidades que deveriam estar presentes nessa entrega;
- Verifica, então, se a funcionalidade é executada sem erro, que todas as regras de negócio foram respeitadas, confirma se o tempo de resposta às transações está adequado, fazendo uma avaliação funcional da aplicação, ou parte dela, em teste.

Testes

Tópico

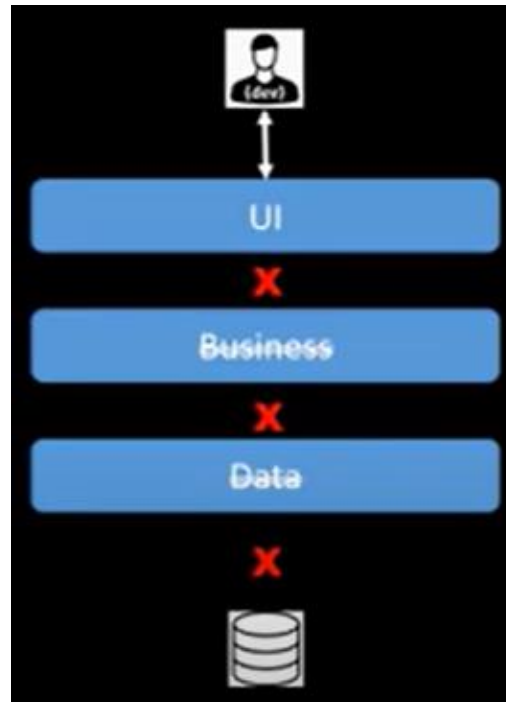
- *Smoke Tests*

Smoke Tests

Definição



- Imaginando a mesma aplicação, que contém 3 (três) camadas: UI, Business e Data:



Smoke Tests

Definição



- **Smoke Test** é o teste que o desenvolvedor faz antes de entregar a aplicação ao usuário, para ser realizado o teste funcional;
- No *smoke test*, é realizada uma “pequena” verificação, só para atestar que a aplicação está funcionando e não é necessário passar por todas as camadas da aplicação e testá-las;
- São realizados poucos acessos, apenas para confirmar que a aplicação está funcional e pronta para a fase de testes funcionais.

Testes

Tópico

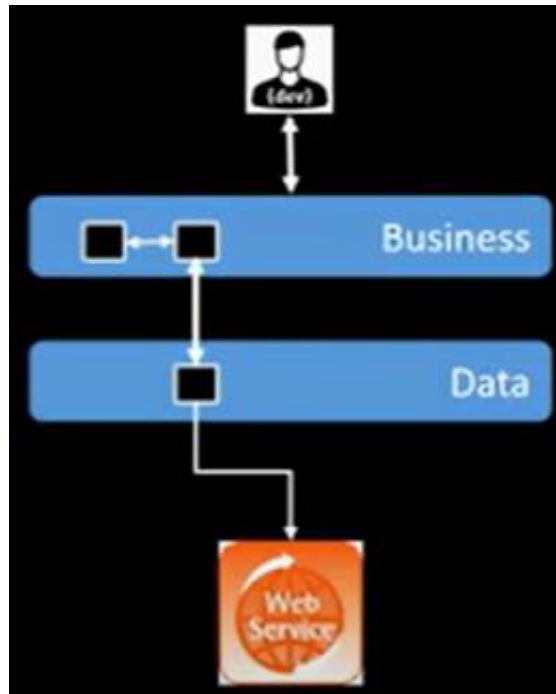
- Testes de Integração

Testes de Integração

Definição



- Imaginando a mesma aplicação, que contém 3 (três) camadas: UI, Business e Data:



Testes de Integração

Definição



- **Teste de Integração** é o teste que foca os componentes, os módulos que estão sendo desenvolvidos na aplicação e como que interagem entre si;
- Conforme as partes da aplicação são finalizadas, são anexadas ao todo, verificando se funcionam adequadamente no conjunto;
- Os testes de integração, são feitos, geralmente, nas camadas mais internas da aplicação, por exemplo, se a aplicação acessa um serviço de *web*, é feito esse teste a partir do término do componente da aplicação responsável por essa comunicação.

Testes

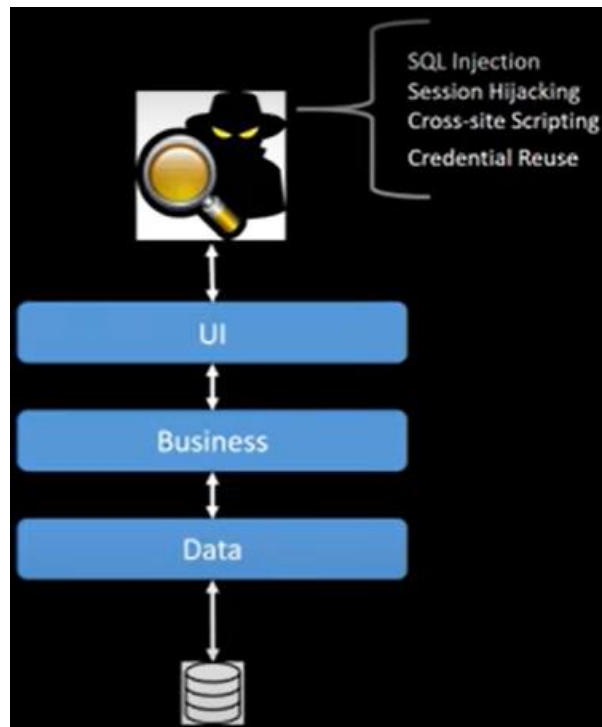
Tópico

- Testes de Segurança

Definição



- Imaginando a mesma aplicação, que contém 3 (três) camadas: UI, Business e Data:



Testes de Segurança

Definição



- **Teste de Segurança** é o teste que verifica se a aplicação é segura, se não é suscetível a ataques e se os responde com segurança;
- Com este teste, verifica-se a integridade da aplicação, do ponto de vista de segurança, quando exposta a diversas possibilidades de ataques, por exemplo:
 - ✓ *SQL Injection;*
 - ✓ *Session Hijacking;*
 - ✓ *Cross-site Scripting;*
 - ✓ *Credential Reuse;*
 - ✓ *Etc.*

Testes

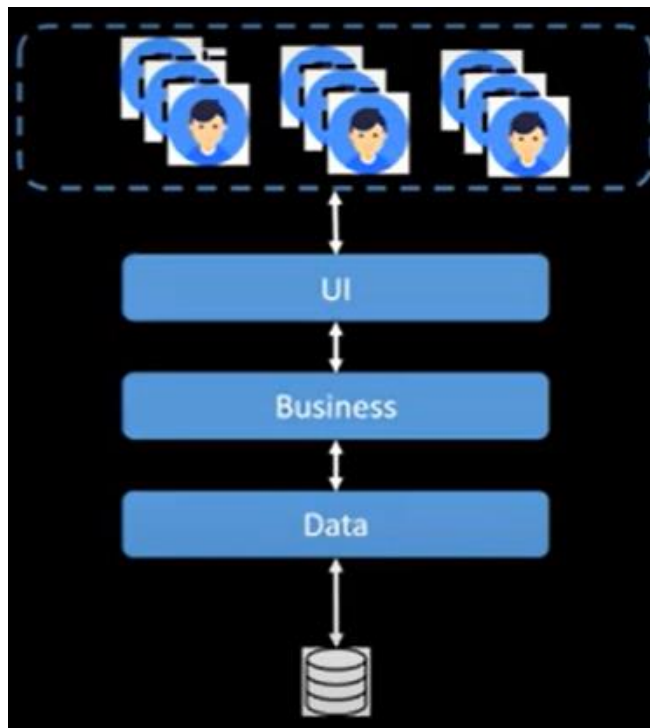
Tópico

- Testes de Carga

Definição



- Imaginando a mesma aplicação, que contém 3 (três) camadas: UI, Business e Data:



Testes de Carga

Definição



- **Teste de Carga** é o teste que verifica como a aplicação se comporta quando está sob grande carga, ou demanda, por exemplo, de acessos simultâneos, processamentos concorrentes etc.;
- Muitas vezes, uma aplicação pode ser aprovada em testes como funcional, integração e outros quando exposta a pouca demanda e ser reprovada em testes de carga, sob quantidades de cargas consideráveis e simultâneas.

Testes

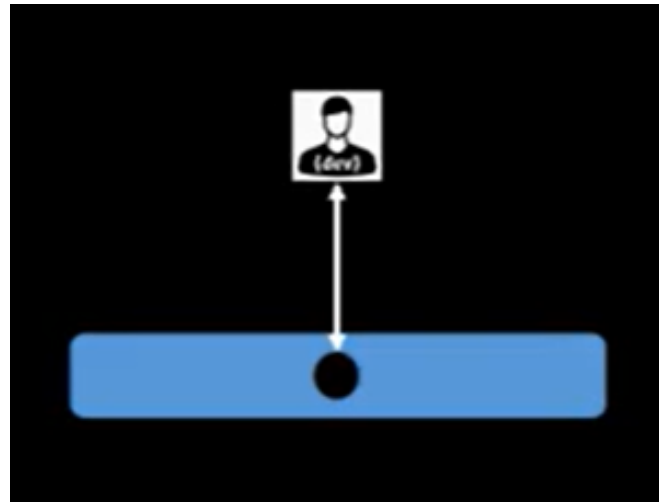
Tópico

- Testes Unitários

Definição



- Imaginando a mesma aplicação, que contém 3 (três) camadas: UI, Business e Data:



Testes Unitários

Definição



- **Teste Unitário** é feito pelo próprio desenvolvedor e foca exclusivamente uma unidade da aplicação, por exemplo, uma classe, um método da classe, uma função etc.;
- É um trecho de código, que executa outro trecho de código, que por sua vez verifica se o trecho de código sob testes executou corretamente seu objetivo;
- É um teste centrado na menor unidade da aplicação, que no caso da OO, é um método, em uma classe;

Definição



- **Teste Unitário**, inegavelmente, aumenta a qualidade do *software* desenvolvido, pois reduz enormemente a quantidade de erros (BECK, 2000, p.118);
- Resultados melhores nos testes são obtidos quando as baterias de testes são repetidas a cada alteração significativa do código, os chamados testes de regressão;
- Porém, testar tudo novamente sempre é uma tarefa cara e tediosa e, por este motivo, utilizam-se ferramentas de automação de testes (BECK; GAMMA, 2004);

Testes Unitários

Definição



- **Teste Unitário** é um código, escrito por um programador, com o objetivo de testar uma funcionalidade específica do código a ser testado;
- Seu alvo é a menor unidade de código, por exemplo, um método em uma classe na OO;
- São testes denominados **Testes de Caixa Preta**, pois verificam apenas a saída gerada por certo número de parâmetros de entrada, não se preocupando com o que ocorre dentro do método;

Testes Unitários

Definição



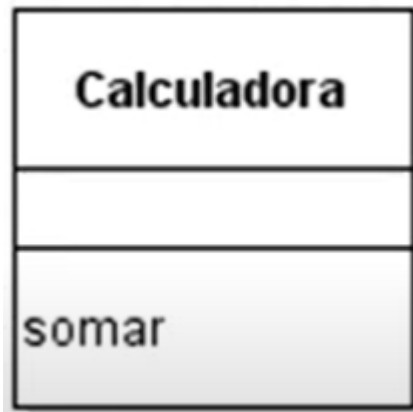
- Nos **Testes Unitários**, o que deve ser testado é um assunto sempre controverso, por exemplo, um código trivial, como métodos *get* e *set* da OO não precisam ser testados, pois geram trabalho e pouco resultado;
- Por outro lado, um código mais complexo, como métodos que contenham regras de negócio, estes, sim, devem ser alvos de diversos testes;
- Um conjunto sólido de testes protege o sistema de erros de regressão, que é, basicamente, estragar “sem querer” o código que não foi mexido, quando acontecem alterações em outras partes do sistema.

Testes Unitários

Exemplo 1



- Numa classe “simples” denominada **Calculadora** tem-se:



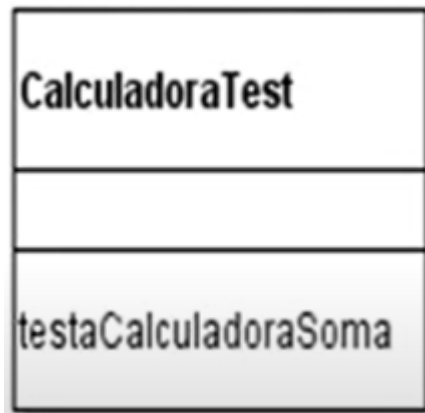
```
1 public class Calculadora {  
2 |  
3 | public int somar(int a, int b){  
4 |  
5 |     return a + b;  
6 | }  
7 }  
8
```

Testes Unitários

Exemplo 1



- Um **Teste Unitário** para essa classe **Calculadora** seria, então:



```
1 public class CalculadoraTest {
2
3     private Calculadora calculadora;
4
5     public void testaCalculadoraSoma(){
6
7         calculadora = new Calculadora();
8         int resultadoEsperado = 4;
9
10        int resultado = calculadora.somar(2, 2);
11
12        if (resultado == resultadoEsperado) {
13            System.out.println("Teste OK");
14        } else{
15            System.out.println("Teste Falhou");
16        }
17    }
18 }
```

Exemplo 2



- Numa classe denominada **Retangulo** tem-se:

Retangulo
-base -altura
+calcularArea() +calcularPerimetro()

```
1
2 public class Retangulo {
3     private int base;
4     private int altura;
5
6     public Retangulo(int base, int altura){
7         this.base = base;
8         this.altura = altura;
9     }
10
11     public int calcularArea(){
12         return base * altura;
13     }
14
15     public int calcularPerimetro(){
16         return 2*base + 2*altura;
17     }
18
19 }
20
```

Testes Unitários

Exemplo 2



- Um **Teste Unitário** para essa classe **Retangulo** seria, então:

RetanguloTest
+retangulo
+testCalcularArea() +testCalcularPerimetro()

```
1
2 public class RetanguloTest {
3     Retangulo retangulo;
4
5     public boolean testCalcularArea(){
6         retangulo = new Retangulo(10, 2);
7         int resultadoEsperado = 20;
8
9         int resultado = retangulo.calcularArea();
10
11         if(resultado == resultadoEsperado){
12             return true;
13         } else {
14             return false;
15         }
16     }
17
18     public boolean testCalcularPerimetro(){
19         retangulo = new Retangulo(10, 2);
20         int resultadoEsperado = 24;
21
22         int resultado = retangulo.calcularPerimetro();
23
24         if(resultado == resultadoEsperado){
25             return true;
26         } else {
27             return false;
28         }
29     }
30 }
31
32
33
```

ECM251 – Linguagens de Programação I

Testes Unitários

Exemplo 2



- A classe **RetanguloMain** para o **Teste Unitário** seria, então:

```
1
2 public class RetanguloMain {
3
4     public static void main(String[] args) {
5         RetanguloTest teste = new RetanguloTest();
6         boolean resultado;
7         |
8         resultado = teste.testCalcularArea();
9         System.out.println("testCalcularArea: " + resultado);
10
11         resultado = teste.testCalcularPerimetro();
12         System.out.println("testCalcularPerimetro: " + resultado);|
13
14     }
15
16 }
17
```

```
<terminated> RetanguloMain (1) [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (18 de jul de 2018 20:30:28)
testCalcularArea: true
testCalcularPerimetro: true
```


Testes Unitários

Exemplo 2



- A classe **RetanguloMain** para o **Teste Unitário** com erro seria, então:

```
1
2 public class RetanguloMain {
3
4     public static void main(String[] args) {
5         RetanguloTest teste = new RetanguloTest();
6         boolean resultado;
7
8         resultado = teste.testCalcularArea();
9         System.out.println("testCalcularArea: " + resultado);
10
11         resultado = teste.testCalcularPerimetro();
12         System.out.println("testCalcularPerimetro: " + resultado);
13
14     }
15
16 }
17
```

```
<terminated> RetanguloMain (1) [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (18 de jul de 2018 20:31:15)
testCalcularArea: true
testCalcularPerimetro: false
```

Testes

Tópico

- *Checksum*

Checksum

Definição



- O cálculo do **checksum** (ou soma de verificação), em uma comunicação entre duas máquinas, é uma técnica utilizada para garantir a integridade dos dados transmitidos de uma para a outra, por meio de uma conexão serial, arquivo etc.;
- O **checksum** é uma soma de valores associados aos dados transmitidos, sendo ela anexada aos dados antes do envio;
- Ao receber os dados, o receptor recalcula o **checksum** e verifica se ele corresponde ao valor calculado recebido;
- Se houver uma discrepância entre o valor calculado e o valor recebido, isso indica a possibilidade de erro na transmissão e permite que o receptor identifique e descarte os dados corrompidos;

Checksum

Definição



- Os principais objetivos e finalidades do cálculo do **checksum** em uma comunicação são:
 1. Integridade dos Dados:
 - O **checksum** ajuda a detectar erros de transmissão, como ruído na linha, interferência, alteração e/ou perda de dados durante essa comunicação;
 - Se os dados forem corrompidos, a verificação do **checksum** falhará, indicando que os dados recebidos não são confiáveis.

Checksum

Definição



- Os principais objetivos e finalidades do cálculo do **checksum** em uma comunicação são:
 2. Confirmação de Recebimento:
 - Ao verificar o **checksum**, o receptor pode confirmar que os dados foram recebidos corretamente;
 - Isso é especialmente importante em comunicações críticas, onde a precisão dos dados é crucial.

Checksum

Definição



- Os principais objetivos e finalidades do cálculo do **checksum** em uma comunicação são:
 3. Redução de Erros:
 - O uso de **checksums** reduz a probabilidade de erros não detectados em comunicações, por exemplo, nas seriais;
 - Sem o **checksum**, erros sutis ou ocasionais podem passar despercebidos, levando a decisões errôneas ou dados corrompidos.

Checksum

Definição



- Os principais objetivos e finalidades do cálculo do **checksum** em uma comunicação são:
 4. Detecção de Fraudes:
 - Em alguns casos, o **checksum** é usado para detectar tentativas de fraude ou adulteração de dados durante a transmissão;
 - Se alguém tentar modificar os dados, o **checksum** não coincidirá, sinalizando uma possível tentativa de fraude.

Checksum

Definição



- Existem diferentes algoritmos para se calcular o **checksum**, sendo o algoritmo de soma de verificação simples (também conhecido por *checksum*) um dos mais comuns;
- Nesse método, os *bytes* de dados são somados juntos e o resultado é anexado ao final dos dados enviados;
- O receptor realiza o mesmo cálculo e compara seu resultado com o valor recebido;
- Se eles forem iguais, os dados são considerados íntegros.

Checksum

Definição



- No entanto, vale ressaltar que o **checksum** simples tem limitações e pode não detectar todos os possíveis erros;
- Em casos críticos, como comunicações em sistemas de controle industrial ou transmissões de dados sensíveis, cálculos de **checksum** mais avançados, como **CRC** (*Cyclic Redundancy Check*), são frequentemente preferidos, pois oferecem uma detecção de erros mais robusta e uma maior segurança.

Checksum

Tópico

- *Checksum* pelo método de Soma e Complemento de 2

Checksum

Definição



- O método de **Soma e Complemento de 2** é uma técnica comum para calcular um *checksum* ou verificar a soma de verificação em comunicações de dados;
- Usado para detectar erros de transmissão em conjuntos de dados binários;
- A seguir, um exemplo, passo a passo, de como calcular o *checksum* usando o método de Soma e Complemento de 2.

Checksum

Exemplo



- Supondo-se que há um conjunto de dados binários de 8 *bits*, representados por caracteres, definidos pela tabela ASCII, que se quer enviar:

Dados em caracteres ASCII: 'C', 'a', 's', 'a' e '1'

Exemplo



1. Dividir os dados em grupos:

- Normalmente, os dados são divididos em grupos de tamanho fixo, em binário, com 8 *bits* cada, para facilitar o cálculo:

Dados em hexadecimal ASCII: **0x43, 0x61, 0x73, 0x61, 0x31**

Dados em binário ASCII:

0100 0011,

0110 0001,

0111 0011,

0110 0001,

0011 0001

Exemplo



2. Somar os grupos de dados:

- No exemplo, somar os dois grupos de 4 *bits* para cada dado:

Dados em binário ASCII:

0100 0011

0110 0001

0111 0011 +

0110 0001

0011 0001

Soma em binário: 1 1010 1001

Checksum

Exemplo



3. Descartar o *bit* excedente, à esquerda, da soma:
- Se a soma resultar em um número maior que o tamanho dos grupos (neste caso, 8 *bits*), descartar o bit excedente, mais significativo (MSB - *Most Significant Bit*), por exemplo:

Soma em binário: **1 1010 1001**

Descartar bit excedente da soma: **1010 1001**

Checksum

Exemplo



4. Calcular o Complemento de 2:

- Para calcular o **Complemento de 2**, complementa-se (inversão) todos os *bits* da soma (tornando 0 em 1 e vice-versa) e soma-se 1 ao resultado complementado:

Descartar *bit* excedente da soma: **1010 1001**

Complementar os *bits* da soma: **0101 0110**

Somar 1 no complementado: **1** +

Complemento 2: **0101 0111**

Descartar *bit* excedente, se houver: **0101 0111**

Checksum

Exemplo



5. Anexar o Complemento de 2 ao final:
- Deve-se anexar o **Complemento de 2** ao final dos dados originais, para obter-se o resultado final a ser transmitido:

Dados em binário ASCII:

0100 0011,

0110 0001,

0111 0011,

0110 0001,

0011 0001,

0101 0111

Exemplo



5. Anexar o Complemento de 2 ao final:
- Deve-se anexar o **Complemento de 2** ao final dos dados originais, para obter-se o resultado final a ser transmitido:

Dados em hexadecimal: **0x43, 0x61, 0x73, 0x61, 0x31, 0x57**

Dados em caracteres ASCII: **'C', 'a', 's', 'a', '1' e 'W'**

Testes Unitários

Exercício 1



- Baseado nos conceitos da programação OO, desenvolver, em Java, uma classe denominada **Checksum**, que contenha, entre outras coisas, um método denominado *calcularChecksum()*, recebendo um vetor de caracteres digitados pelo usuário e retornando o cálculo do respectivo *checksum*, baseado no algoritmo da **Soma e Complemento de 2**;
- Desenvolver, baseado nos conceitos de Testes Unitários, uma classe de **testes unitários** para a classe **Checksum**, capaz de realizar os testes unitários e automatizados de todos os métodos da classe **Checksum** (exceto do construtor).

Testes Unitários

Exercício 2



- Baseado na solução do exercício 1, anterior, acrescentar à classe **Checksum** o método *calcularChecksumDoArquivoTexto()*, capaz de ler os caracteres de um arquivo texto e executar o cálculo do *checksum*, baseado no algoritmo da **Soma e Complemento de 2**, da mesma forma como feito no exercício anterior, quando lia os caracteres via teclado;
- Após o cálculo do *checksum*, gravar o valor encontrado ao final de um outro arquivo texto, logo após os caracteres fornecidos;
- Desenvolver, baseado nos conceitos de Testes Unitários, uma classe de **testes unitários** para a classe **Checksum**, capaz de realizar os testes unitários e automatizados de todos os métodos da classe **Checksum** (exceto do construtor).

Exercício 3 - Desafio



- Pesquisar o método de *checksum* através do cálculo de CRC e implementar o método *calcularCRC()*, adicionando-o à classe **Checksum** do exercício anterior e executando todas as atividades solicitadas nos exercícios 1 e 2.

ECM251 – Linguagens de Programação I

Aula 18 – L1/1, L2/1 e L3/1

Bibliografia Básica



- MILETTO, Evandro M.; BERTAGNOLLI, Silvia de Castro. Desenvolvimento de software II: introdução ao desenvolvimento web com HTML, CSS, javascript e PHP (Tekne). Porto Alegre: Bookman, 2014. E-book. Referência Minha Biblioteca:
<https://integrada.minhabiblioteca.com.br/#/books/9788582601969>
- WINDER, Russel; GRAHAM, Roberts. Desenvolvendo Software em Java, 3ª edição. Rio de Janeiro: LTC, 2009. E-book. Referência Minha Biblioteca:
<https://integrada.minhabiblioteca.com.br/#/books/978-85-216-1994-9>
- DEITEL, Paul; DEITEL, Harvey. Java: how to program early objects. Hoboken, N. J: Pearson, c2018. 1234 p. ISBN 9780134743356.

Continua...

Bibliografia Básica (continuação)



- HORSTMANN, Cay S; CORNELL, Gary. Core Java. SCHAFRANSKI, Carlos (Trad.), FURMANKIEWICZ, Edson (Trad.). 8. ed. São Paulo: Pearson, 2010. v. 1. 383 p. ISBN 9788576053576.
- LIANG, Y. Daniel. Introduction to Java: programming and data structures comprehensive version. 11. ed. New York: Pearson, c2015. 1210 p. ISBN 9780134670942.
- TURINI, Rodrigo. Desbravando Java e orientação a objetos: um guia para o iniciante da linguagem. São Paulo: Casa do Código, [2017]. 222 p. (Caelum).

ECM251 – Linguagens de Programação I

Aula 18 – L1/1, L2/1 e L3/1

Bibliografia Complementar



- HORSTMANN, Cay. Conceitos de Computação com Java. Porto Alegre: Bookman, 2009. E-book. Referência Minha Biblioteca:
<https://integrada.minhabiblioteca.com.br/#/books/9788577804078>
- MACHADO, Rodrigo P.; FRANCO, Márcia H. I.; BERTAGNOLLI, Silvia de Castro. Desenvolvimento de software III: programação de sistemas web orientada a objetos em java (Tekne). Porto Alegre: Bookman, 2016. E-book. Referência Minha Biblioteca:
<https://integrada.minhabiblioteca.com.br/#/books/9788582603710>
- BARRY, Paul. Use a cabeça! Python. Rio de Janeiro: Alta Books, 2012. 458 p.
ISBN 9788576087434.

Continua...

ECM251 – Linguagens de Programação I

Aula 18 – L1/1, L2/1 e L3/1

Bibliografia Complementar (continuação)



- LECHETA, Ricardo R. Web Services RESTful: aprenda a criar Web Services RESTful em Java na nuvem do Google. São Paulo: Novatec, c2015. 431 p.
ISBN 9788575224540.
- SILVA, Maurício Samy. JQuery: a biblioteca do programador. 3. ed. rev. e ampl. São Paulo: Novatec, 2014. 544 p.
ISBN 9788575223871.
- SUMMERFIELD, Mark. Programação em Python 3: uma introdução completa à linguagem Python. Rio de Janeiro: Alta Books, 2012. 506 p.
ISBN 9788576083849.

Continua...

ECM251 – Linguagens de Programação I

Aula 18 – L1/1, L2/1 e L3/1

Bibliografia Complementar (continuação)



- YING, Bai. Practical database programming with Java. New Jersey: John Wiley & Sons, c2011. 918 p.
- ZAKAS, Nicholas C. The principles of object-oriented JavaScript. San Francisco, CA: No Starch Press, c2014. 97 p. ISBN 9781593275402.
- CALVETTI, Robson. Programação Orientada a Objetos com Java. Material de aula, São Paulo, 2020.

ECM251 – Linguagens de Programação I

Aula 18 – L1/1, L2/1 e L3/1

FIM

ECM251 – Linguagens de Programação I

Aula 18 – L1/2, L2/2 e L3/2

Engenharia da Computação – 3ª série

Testes Unitários
(L1/2, L2/2 e L3/2)

2025

ECM251 – Linguagens de Programação I

Aula 18 – L1/2, L2/2 e L3/2

Horário

Terça-feira: 2 x 2 aulas/semana

- L1/1 (07h40min-09h20min): *Prof. Calvetti*;
- L1/2 (09h30min-11h10min): *Prof. Calvetti*;
- L2/1 (07h40min-09h20min): *Prof. Menezes*;
- L2/2 (11h20min-13h00min): *Prof. Calvetti*;
- L3/1 (09h30min-11h10min): *Prof. Evandro*;
- L3/2 (11h20min-13h00min): *Prof. Evandro*.

Testes Unitários

Exercícios



- Terminar, entregar e apresentar ao professor para avaliação, os exercícios propostos na aula de teoria, deste material.

ECM251 – Linguagens de Programação I

Aula 18 – L1/2, L2/2 e L3/2

Bibliografia (apoio)



- LOPES, ANITA. GARCIA, GUTO. Introdução à Programação: 500 algoritmos resolvidos. Rio de Janeiro: Elsevier, 2002.
- DEITEL, P. DEITEL, H. Java: como programar. 8 Ed. São Paulo: Prentice-Hall (Pearson), 2010;
- BARNES, David J.; KÖLLING, Michael. Programação orientada a objetos com Java: uma introdução prática usando o BlueJ . 4. ed. São Paulo: Pearson Prentice Hall, 2009.

ECM251 – Linguagens de Programação I

Aula 18 – L1/2, L2/2 e L3/2

FIM