

ECM251 – Linguagens de Programação I

Aula 24 – L1/1, L2/1 e L3/1

Engenharia da Computação – 3ª série

***Padrões de Projetos*
*(L1/1, L2/1 e L3/1)***

2024

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

Padrões de Projetos

Tópico

- Padrão

Definição



- Em termos gerais, um "padrão" refere-se a um modelo, exemplo ou conjunto de regras amplamente aceitas e estabelecidas que serve como uma referência ou guia para abordar problemas comuns ou realizar tarefas específicas;
- São utilizados em várias áreas da vida, incluindo ciência, tecnologia, projeto, linguagem e muitos outros domínios;
- Base de comparação, algo que o consenso geral ou um determinado órgão oficial consagrou como um modelo aprovado.

Padrão

Definição



- São valiosos por fornecer eficiência, consistência e um ponto de referência comum e serem usados para resolver problemas de maneira eficaz, promover a interoperabilidade e a compreensão mútua entre diferentes partes interessadas, tendo algumas definições e exemplos em diferentes contextos:
 1. Padrões na Tecnologia e Engenharia;
 2. Padrões em *Design* e Arte;
 3. Padrões na Linguagem e Comunicação;
 4. Padrões na Ciência e Matemática; e
 5. Padrões Culturais e Sociais.

Características



1. Padrões na Tecnologia e Engenharia:

- Em engenharia de *software*, um "padrão de projeto", como os padrões *MVC*, *Singleton* ou *Factory*, é uma solução geralmente aceita para um problema recorrente no projeto de sistemas de *software*;
- Padrões de *hardware* se referem a especificações técnicas amplamente reconhecidas para componentes eletrônicos, como portas *USB* ou padrões de rede, como o *Ethernet*.

Características



2. Padrões em *Design* e Arte:

- Em *design* gráfico, padrões são elementos de repetição em uma composição, como um padrão de cores ou um padrão de textura;
- Na moda, os padrões se referem a desenhos repetitivos em tecidos ou roupas, como listras, xadrez ou floral.

Características



3. Padrões na Linguagem e Comunicação:

- Padrões linguísticos se referem às regras e convenções que governam a gramática, ortografia e uso da linguagem em uma determinada língua;
- Em comunicação, um "padrão de discurso" pode se referir à forma como uma pessoa se expressa de maneira consistente, como usar certas frases ou palavras com frequência.

Características



4. Padrões na Ciência e Matemática:

- Em matemática, um padrão é uma sequência ou relação que se repete de maneira previsível, como a sequência de números pares (2, 4, 6, 8, ...) ou a sequência de Fibonacci (1, 1, 2, 3, 5, ...);
- Em ciências naturais, padrões podem se referir a tendências observadas em dados, como padrões climáticos, padrões de crescimento de plantas ou padrões de comportamento animal.

Características



5. Padrões Culturais e Sociais:

- Em sociedade, padrões culturais são comportamentos, tradições e valores que são considerados típicos de um grupo ou sociedade específica;
- Em psicologia, um "padrão de comportamento" pode se referir a comportamentos recorrentes ou hábitos de uma pessoa.

Exemplo



- Padrão de Beleza:
 - Estaria a beleza nos olhos de quem vê?
 - Haveria uma base objetiva para tal julgamento?



Padrões de Projetos

Tópico

- Padrões de Projetos

Padrões de Projetos

Definição



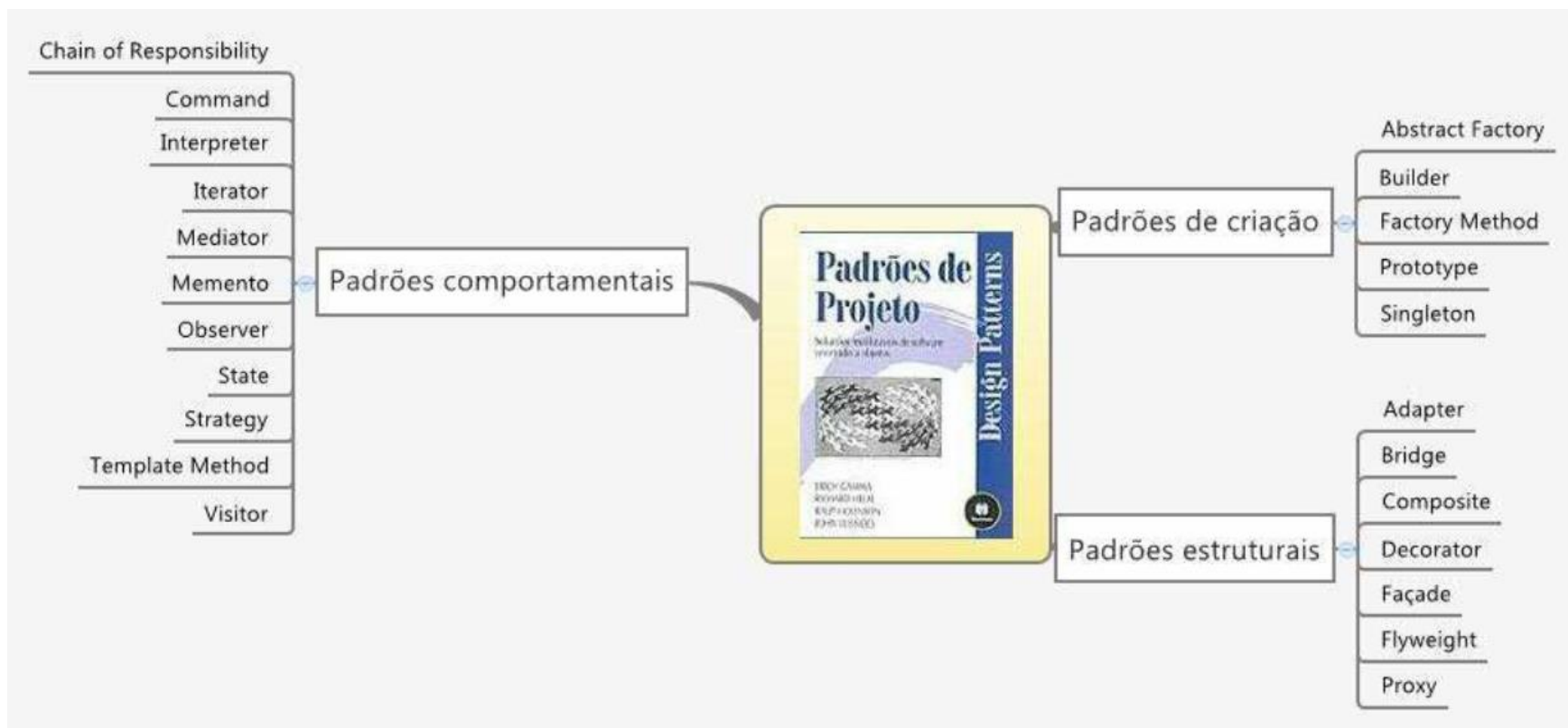
- Também conhecidos por *Design Patterns*, porque devem ser estudados e utilizados:
 - ✓ Melhorar a comunicação da equipe;
 - ✓ Melhorar o aprendizado individual;
 - ✓ Aumentar a capacidade de correção e manutenção do código;
 - ✓ Fazer reuso das soluções anteriores; e
 - ✓ Reduzir os recursos necessários para o desenvolvimento do projeto.

ECM251 – Linguagens de Programação I

Padrões de Projetos

Exemplos

- Principais Padrões de Projetos de *Software*:

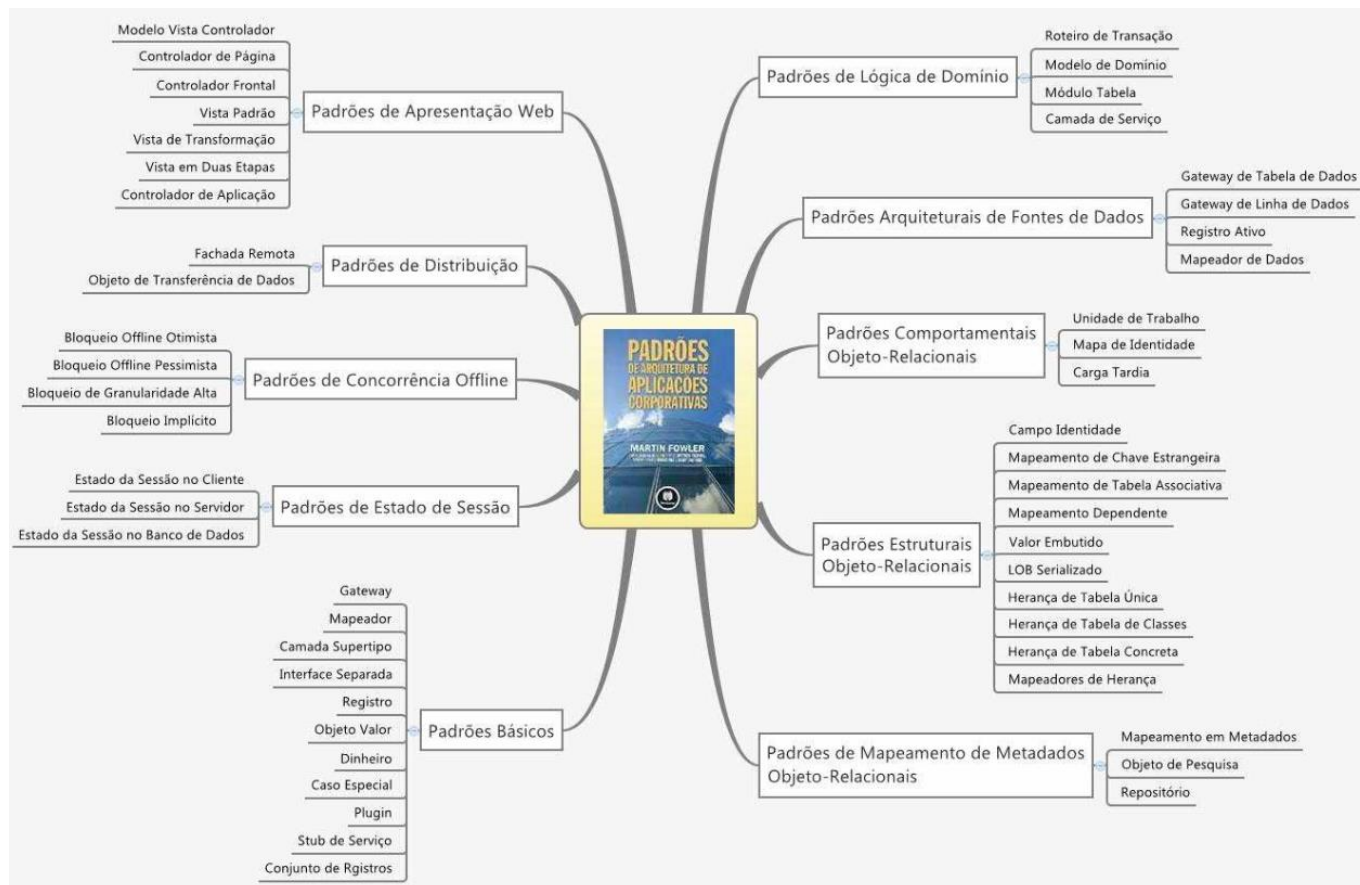


ECM251 – Linguagens de Programação I

Padrões de Projetos

Exemplos

- Principais Arquiteturas de Aplicações Corporativas:



Padrões de Projetos

Tópico

- Padrões GRASP

Padrões GRASP

Definição



- GRASP: *General Responsibility and Assignment Software Patterns*;
- Refletem práticas pontuais de Orientação a Objeto – OO;
- Descrevem os princípios fundamentais da atribuição de responsabilidades a objetos, expressas na forma de padrões;
- Ajudam a compreender melhor a utilização de vários paradigmas OO em projetos mais complexos;
- Exploram os princípios fundamentais de sistemas OO:
 - ✓ 5 padrões fundamentais; e
 - ✓ 4 padrões avançados.

Definição



- Introduzidos no livro “*Applying UML and Patterns*”, ou “*Utilizando UML e Padrões*”, de Craig Larman:



Características



- Padrões Básicos:
 - ✓ Especialista na Informação – *Expert*;
 - ✓ Criador – *Creator*;
 - ✓ Acoplamento Fraco, de avaliação – *Low Coupling*;
 - ✓ Coesão Alta, de avaliação – *High Cohesion*; e
 - ✓ Controlador – *Controller*.

Características



- Padrões Avançados:
 - ✓ Polimorfismo – *Polymorphism*;
 - ✓ Indireção – *Indirection*;
 - ✓ Invenção Pura – *Pure Fabrication*; e
 - ✓ Variações Protegidas – *Protected Variations*.

Padrões de Projetos

Tópico

- Padrão MVC

Padrão MVC

Definição



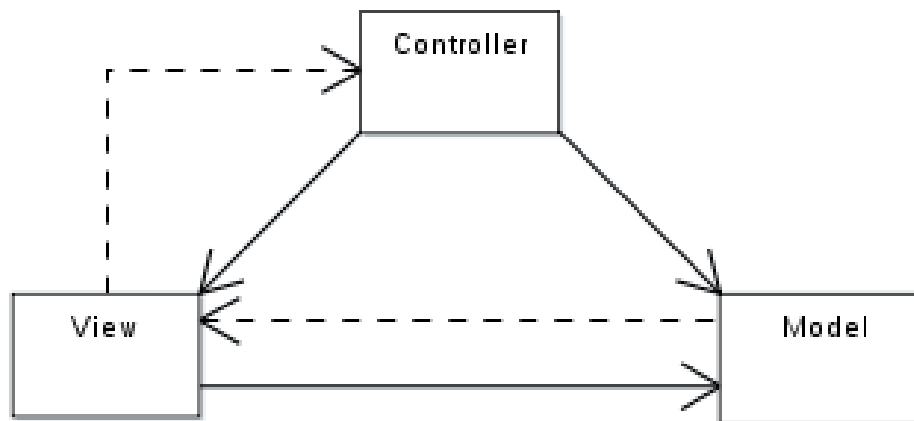
- MVC – *Model-View-Controller*, é um padrão de arquitetura de *software*;
- Com o aumento da complexidade das aplicações, torna-se fundamental a separação entre os dados, o *Model* (ou Modelo), e a interface com o usuário, a *View* (ou Visão);
- Alterações feitas na interface com o usuário não devem afetar a manipulação de dados e os dados podem ser reorganizados sem alterar a interface com o usuário;
- Separa as tarefas de acesso aos dados e lógica de negócio, lógica de apresentação e de interação com o usuário, introduzindo um novo componente entre os dois: o *Controller* (ou Controlador).

Padrão MVC

Características



- É usado em padrões de projeto de *software*;
- Abrange mais da arquitetura de uma aplicação do que é típico para um padrão de projeto; e
- Define como os componentes da aplicação interagem, sendo considerado como um *Design Pattern*.



Padrão MVC

Características



- Quando se dividem os componentes em camadas, pode-se aplicar o MVC;
- A camada de negócios é a *Model*, a apresentação é a *View* e o controle é realizado pelo *Controller*;
- Não confundir MVC com separação de camadas, pois as camadas dizem como agrupar os componentes e o MVC diz como os componentes da aplicação interagem;
- O *Controller* despacha as solicitações ao *Model*; e
- A *View* observa o *Model*.

Padrão MVC

Model



- A representação "domínio" específica da informação em que a aplicação opera, por exemplo: aluno, professor e turma fazem parte do domínio de um sistema acadêmico;
- É comum haver confusão pensando que *Model* é um outro nome para a camada de domínio;
- Lógica de domínio adiciona sentido a dados crus, por exemplo, calcular se hoje é aniversário do usuário; e
- Muitas aplicações usam um mecanismo de armazenamento persistente para armazenar dados e o MVC não cita especificamente a camada para acesso a esses dados, porque se subentende que estes métodos estão encapsulados na *Model*.

Padrão MVC

View



- Visualiza o *Model* em uma forma específica para a interação, geralmente uma interface de usuário e exibe os dados do modelo ao usuário de forma compreensível;
- Capta as interações do usuário, como cliques em botões, preenchimento de formulários e outras ações do usuário e envia os comandos apropriados para o *Control*;
- Deve ser projetada de forma a ser o mais independente possível do modelo subjacente, sem realizar operações de lógica de negócios ou armazenamento de dados diretamente, mas, em vez disso, solicitar as informações ao *Control* ou ao *Model*;
- Atualiza dinamicamente a interface do usuário quando os dados do *Model* mudam.

Padrão MVC

Controller



- Processa e responde a eventos, geralmente ações do usuário, e pode invocar alterações no *Model*;
- Onde é feita a validação dos dados; e
- Onde os valores inseridos pelos usuários são filtrados;
- Recebe as solicitações dos usuários por meio da *View*, incluindo cliques em botões, envio de formulários e outras ações do usuário;
- Comunica-se com a *Model* para acessar ou atualizar dados e pode solicitar informações a ela, atualizá-las com base nas ações do usuário e, em seguida, refletir suas alterações.

Conclusão



- O MVC visa separar a lógica de negócio da lógica de apresentação, permitindo desenvolvimento, teste e manutenção dos seus componentes de maneira isolada e independente.



Conclusões



- Os Padrões de Projetos em *software* são diretrizes e soluções de *design* amplamente aceitas que ajudam a melhorar eficiência, qualidade e manutenção;
- Representam uma abordagem comprovada para resolver problemas recorrentes;
- Sua adoção é essencial para o desenvolvimento de sistemas de *software* padronizados, eficazes e confiáveis.

Exemplo



- Criação e Gerenciamento de Lista de Tarefas com MVC:

Classe *TarefaModel.java*

Exemplo



- Criação e Gerenciamento de Lista de Tarefas com MVC:

```
1 package Model;
2 import View.*;
3 import Controller.*;
4
5 public class TarefaModel
6 {   private String descricao;
7     private boolean concluida;
8
9     public TarefaModel(String descricao)
10    {   this.descricao = descricao;
11        this.concluida = false;
12    }
13
14    public String getDescricao()
15    {   return descricao;
16    }
17
```


Exemplo



- Criação e Gerenciamento de Lista de Tarefas com MVC:

```
18     public boolean isConcluida()  
19     { return concluida;  
20     }  
21  
22     public void marcarComoConcluida()  
23     { concluida = true;  
24     }  
25  
26     public void marcarComoNaoConcluida()  
27     { concluida = false;  
28     }  
29 }  
30
```


Exemplo



- Criação e Gerenciamento de Lista de Tarefas com MVC:

Classe *TarefaView.java*

Exemplo



- Criação e Gerenciamento de Lista de Tarefas com MVC:

```
1 package View;
2 import Model.*;
3 import Controller.*;
4 import java.util.List;
5
6 public class TarefaView
7 {   public void exibirTarefas(List<TarefaModel> tarefas)
8     {   System.out.println("Lista de Tarefas:");
9         for (TarefaModel tarefa : tarefas)
10         {   String status = tarefa.isConcluida() ? "[X]" : "[ ]";
11             System.out.println(status + " " + tarefa.getDescricao());
12         }
13         System.out.println();
14     }
15 }
16
```

Exemplo



- Criação e Gerenciamento de Lista de Tarefas com MVC:

Classe *TarefaController.java*

Exemplo



- Criação e Gerenciamento de Lista de Tarefas com MVC:

```
1 package Controller;
2 import Model.*;
3 import View.*;
4
5 import java.util.ArrayList;
6 import java.util.List;
7
8 public class TarefaController
9 {   private List<TarefaModel> tarefas = new ArrayList<>();
10     private TarefaView view = new TarefaView();
11
12     public void adicionarTarefa(String descricao)
13     {   TarefaModel novaTarefa = new TarefaModel(descricao);
14         tarefas.add(novaTarefa);
15     }
16 }
```

Exemplo



- Criação e Gerenciamento de Lista de Tarefas com MVC:

```
17     public void removerTarefa(int indice)
18     {   tarefas.remove(indice);
19     }
20
21     public void marcarTarefaComoConcluida(int indice)
22     {   if (indice >= 0 && indice < tarefas.size())
23         {   tarefas.get(indice).marcarComoConcluida();
24         }
25     }
26
27     public void marcarTarefaComoNaoConcluida(int indice)
28     {   if (indice >= 0 && indice < tarefas.size())
29         {   tarefas.get(indice).marcarComoNaoConcluida();
30         }
31     }
32
33     public void atualizarView()
34     {   view.exibirTarefas(tarefas);
35     }
36 }
37
```

Exemplo



- Criação e Gerenciamento de Lista de Tarefas com MVC:

Classe AplicacaoTarefas.java

Exemplo



- Criação e Gerenciamento de Lista de Tarefas com MVC:

```
1 package Main;
2 import Model.*;
3 import View.*;
4 import Controller.*;
5
6 public class AplicacaoTarefas
7 {   public static void main(String[] args)
8     {   TarefaController controller = new TarefaController();
9         controller.atualizarView();
10        controller.adicionarTarefa("Fazer compras");
11        controller.atualizarView();
12        controller.adicionarTarefa("Estudar Java");
13        controller.atualizarView();
14        controller.adicionarTarefa("Exemplo de MVC");
15        controller.atualizarView();
16        controller.marcarTarefaComoConcluida(0);
17        controller.atualizarView();
18        controller.marcarTarefaComoConcluida(1);
19        controller.atualizarView();
```

Exemplo



- Criação e Gerenciamento de Lista de Tarefas com MVC:

```
20     controller.marcarTarefaComoConcluida(2);
21     controller.atualizarView();
22     controller.marcarTarefaComoNaoConcluida(0);
23     controller.atualizarView();
24     controller.marcarTarefaComoNaoConcluida(1);
25     controller.atualizarView();
26     controller.marcarTarefaComoNaoConcluida(2);
27     controller.atualizarView();
28     controller.removerTarefa(0);
29     controller.atualizarView();
30     controller.removerTarefa(0);
31     controller.atualizarView();
32     controller.removerTarefa(0);
33     controller.atualizarView();
34 }
35 }
36
```


Padrões de Projetos

Exemplo



- Criação e Gerenciamento de Lista de Tarefas com MVC:

Classe *AplicacaoTarefas.java* em execução

Exemplo



- Criação e Gerenciamento de Lista de Tarefas com MVC:

Lista de Tarefas:

Lista de Tarefas:

☐ Fazer compras

Lista de Tarefas:

☐ Fazer compras

☐ Estudar Java

Lista de Tarefas:

☐ Fazer compras

☐ Estudar Java

☐ Exemplo de MVC

Lista de Tarefas:

☒ Fazer compras

☐ Estudar Java

☐ Exemplo de MVC

Lista de Tarefas:

☒ Fazer compras

☒ Estudar Java

☐ Exemplo de MVC

Lista de Tarefas:

☒ Fazer compras

☒ Estudar Java

☒ Exemplo de MVC

Lista de Tarefas:

☐ Fazer compras

☒ Estudar Java

☒ Exemplo de MVC

Lista de Tarefas:

☐ Fazer compras

☐ Estudar Java

☒ Exemplo de MVC

Lista de Tarefas:

☐ Fazer compras

☐ Estudar Java

☐ Exemplo de MVC

Lista de Tarefas:

☐ Estudar Java

☐ Exemplo de MVC

Lista de Tarefas:

☐ Exemplo de MVC

Lista de Tarefas:

Exercícios



1. Digite e compile as classes *TarefaModel.java*, *TarefaView.java* e *TarefaController.java* e *AplicacaoTarefas.java*, distribuídas entre as páginas 30 e 40 deste material, e executar o método *main()* da classe *AplicacaoTarefas.java*. Analisar, concluir e registrar o funcionamento das mesmas;

Exercícios



2. Mantendo a arquitetura do programa segundo o *design pattern* MVC, trocar a comunicação com o usuário por uma interface gráfica, utilizando os modelos de *layouts* estudados anteriormente nesta disciplina (*flowlayout*, *borderlayout*, *gridlayout* ou os automaticamente gerados pela *IDE NetBeans*), eliminando toda e qualquer forma de comunicação com o usuário através do console e/ou por interface *Swing*, criando nessa nova interface gráfica os botões e os campos específicos para: “Incluir nova tarefa” na lista; “Marcar tarefa como concluída” na lista; “Marcar tarefa como não concluída” na lista; “Excluir tarefa” da lista; e “Visualizar lista” de tarefas.

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 24 – 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 24 – L1/1, L2/1 e L3/1

Bibliografia Complementar (continuação)



- LECHETA, Ricardo R. Web Services RESTful: aprenda a criar Web Services RESTfulem 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 Phython. Rio de Janeiro: Alta Books, 2012. 506 p.
ISBN 9788576083849.

Continua...

Bibliografia Complementar (continuação)

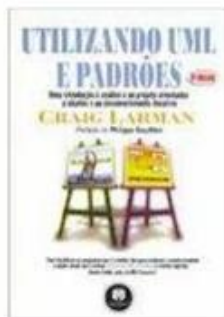


- YING, Bai. Practical database programming with Java. New Jersey: John Wiley & Sons, c2011. 918 p.
- ZAKAS, Nicholas C. The principles of object-oriented JavaScript. San Francisco, CA: No Starch Press, c2014. 97 p. ISBN 9781593275402.
- TANENBAUM, Andrew S.; MAARTEN, V. S. Sistemas Distribuídos: princípios e paradigmas, Pearson Education. 2ª edição. 2008.
- GOETZ et. al. Java Concurrency in Practice, 1st edition, 2006.
- CALVETTI, Robson. Programação Orientada a Objetos com Java. Material de aula, São Paulo, 2020.

ECM251 – Linguagens de Programação I

Aula 24 – L1/1, L2/1 e L3/1

Bibliografia Complementar (continuação)



JAVA DESIGN PATTERNS

http://www.allapplabs.com/java_design_patterns/java_design_patterns.htm

Java Design Patterns At a Glance

<http://www.javacamp.org/designPattern/>

Java Design Patterns Reference and Examples

<http://www.fluffycat.com/Java-Design-Patterns/>

ECM251 – Linguagens de Programação I

Aula 24 – L1/1, L2/1 e L3/1

FIM

ECM251 – Linguagens de Programação I

Aula 24 – L1/2, L2/2 e L3/2

Engenharia da Computação – 3ª série

Padrões de Projetos
(L1/2, L2/2 e L3/2)

2022

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

Exercícios



- Terminar, entregar e apresentar ao professor para avaliação, os exercícios propostos na aula de teoria, deste material;
- Trabalhar, em grupo, nos requisitos e no desenvolvimento do Projeto II, para avaliação no 2º semestre.

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 24 – L1/1, L2/1 e L3/1

FIM