Hortolândia, 18 de Maio de 2021.

Aulas 11 e 12 – Padrão de Projeto MVC

1. Introdução

A aula de hoje prossegue com o assunto de **Padrões de Projeto**.

Será visto em detalhes o **Padrão MVC** (Modelo-Visão-Controle).

Em seguida, serão realizados exercícios em laboratório.

2. Padrão de Projeto MVC

O padrão de projeto MVC (Modelo-Visão-Controle) é um padrão de projeto (*design pattern*) do tipo **estrutural**.

Ele separa a representação da informação da interação do usuário com a mesma. O modelo (*model*) consiste nos dados da aplicação, regras de negócios, lógica e funções. Uma visão (*view*) pode ser qualquer saída de representação dos dados, como uma tabela ou um diagrama. É possível ter várias visões do mesmo dado, como um gráfico de barras para gerenciamento e uma visão tabular para contadores. O controlador (*controller*) faz a mediação da entrada, convertendo-a em comandos para o modelo ou visão. As ideias centrais por trás do MVC são a **reusabilidade de código** e **separação de conceitos** (WIKIPEDIA, 2016).

A Figura 1 ilustra o padrão MVC. A Figura apresenta verbos nas ligações entre os elementos **Usuário** (*User*), **Controlador** (*Controller*), **Modelo** (*Model*) e **Visão** (*View*). Os verbos indicam o papel de cada um desses elementos. Por exemplo, o usuário “vê” as informações da **Camada de Visão**, e “utiliza” os recursos da **Camada de Controle**.

A **Camada de Controle**, por sua vez, manipula os dados da camada de modelo, respondendo às ações do usuário. A **Camada de Modelo**, por fim, ao sofrer atualizações, comunica à camada de Visão as alterações sofridas nos dados, gerando assim uma nova representação dos mesmos para o usuário.

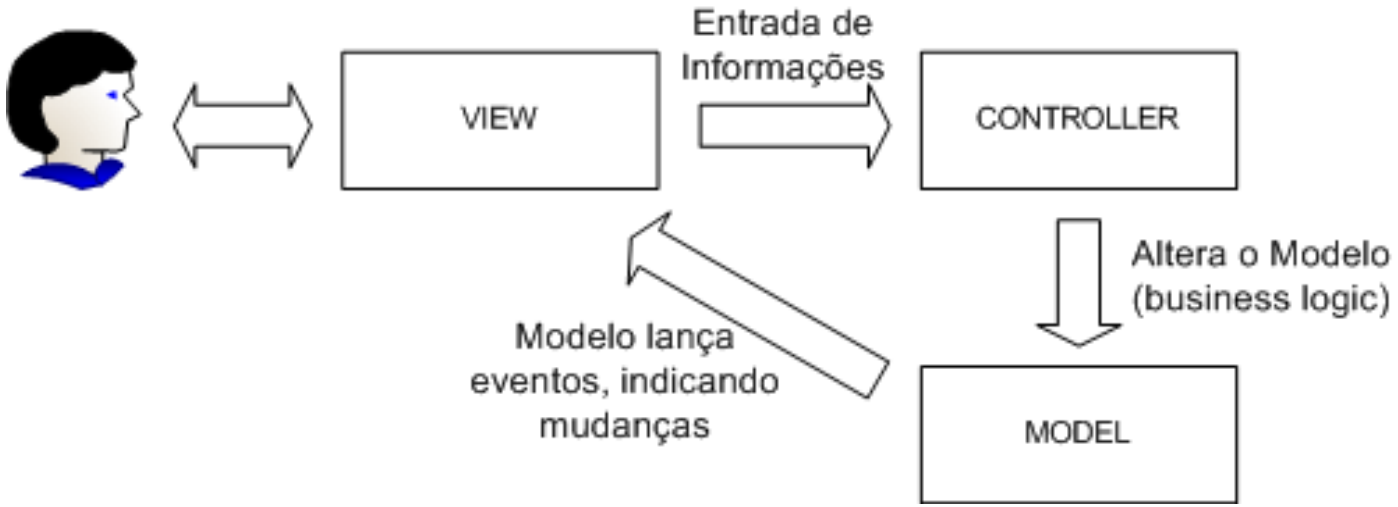

Figura 1: Padrão de Projeto **MVC**.

(http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/arqu/mvc/mvc.htm)

A Seção 3 detalha os benefícios do padrão MVC.

Já a Seção 4 apresenta um exemplo de aplicação em Java usando MVC.

3. Benefícios do Padrão MVC

O **Padrão de Projeto MVC** agrega às aplicações que o utilizam os seguintes **benefícios** (http://www.careerride.com/MVC-benefits.aspx):

- Separação de papéis e responsabilidades entre camadas;
- A **Camada de Visão** pode ser representada de diversas formas, sem alteração das camadas subjacentes;
- Especialização dos desenvolvedores em camadas específicas.
- Desenvolvimento em paralelo por múltiplas equipes;
- Manutenção facilitada do produto final.

A subseção seguinte apresenta as responsabilidades de cada camada do Modelo MVC.

3.1. Camadas do Modelo MVC

O **padrão MVC**, em resumo, organiza as camadas da seguinte forma:

O **Modelo** representa os dados, e não faz nada além disso. O Modelo não depende do Controlador ou da Visão.

A **Visão** apresenta os dados do modelo, e envia ações do usuário (por exemplo, cliques de botões) para o Controlador. A Visão pode ser: a) independente tanto do modelo quanto do controlador; ou b) ser acoplada ao controlador, e depender portanto do modelo.

O **Controlador** provê dados do modelo para a Visão, e interpreta as ações do usuário, tais como cliques de botões. O controlador depende da Visão e do Modelo. Em alguns casos, o controlador e a visão estão no mesmo objeto (DALLING, 2009).

4. Exemplo de Aplicação em Java

Um exemplo de projeto empregando o padrão MVC está descrito em TUTORIALSPOINT.

No caso de uma aplicação Orientada a Objetos, em particular utilizando Java, cada camada pode ser definida da seguinte forma:

- **Modelo** – representa um ou mais objetos do tipo Java POJO (*Plain Old Java Object*), contendo dados e métodos de acesso a estes dados.
- **Visão** – representa a visualização dos dados que o modelo contém.
- **Controlador** – atua tanto sobre o modelo quanto sobre a visão. Ele controla o fluxo de dados no(s) objeto(s) do modelo e atualiza a visão sempre que os dados forem alterados. Ele garante a separação entre o Modelo e a Visão.

A aplicação descrita em TUTORIALSPOINT (2016) possui quatro classes: **Student**, **StudentView**, **StudentController** e **MVCPatternDemo**.

O Diagrama de Classes da aplicação de exemplo está representado na Figura 2.

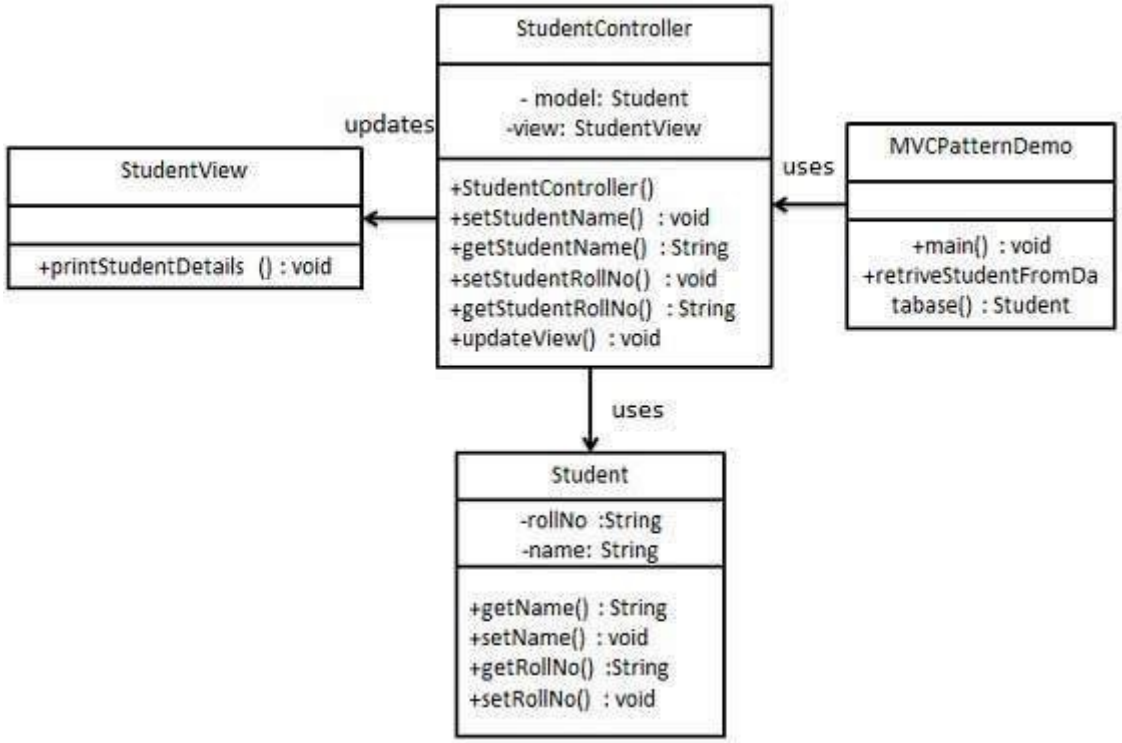

Figura 2: Diagrama de Classe de Exemplo empregando o **MVC.**
(http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm)

A classe **Student** armazena os dados de um estudante, e o seu código-fonte pode ser observado na Listagem 1.

```
public class Student {
   private String rollNo;
   private String name;
   public String getRollNo() {
      return rollNo;
   }
   public void setRollNo(String rollNo) {
      this.rollNo = rollNo;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
}
```

Listagem 1: Código-fonte da classe **Student**.
(http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm)

A classe **StudentView** exibe os dados relativos ao estudante na Linha de Comandos. Ela é uma classe simples que implementa a Camada de Visão. Seu código-fonte está representado na Listagem 2.

```
public class StudentView {
         public   void   printStudentDetails(String   studentName,   String
studentRollNo){
      System.out.println("Student: ");
      System.out.println("Name: " + studentName);
      System.out.println("Roll No: " + studentRollNo);
   }
}
```

Listagem 2: Código-fonte da classe **StudentView**.
(http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm)

A classe **StudentController** possui acesso aos objetos da Visão e do Modelo. Ela possui métodos para a atualização e a recuperação de dados do Modelo, bem como um método (**`updateView()`**) para atualização da camada de Visão.

```
   public class StudentController {
      private Student model;
      private StudentView view;
     public StudentController(Student model, StudentView view){
         this.model = model;
         this.view = view;
      }
      public void setStudentName(String name){
```

```
         model.setName(name);
      }
      public String getStudentName(){
         return model.getName();
      }
      public void setStudentRollNo(String rollNo){
         model.setRollNo(rollNo);
      }
      public String getStudentRollNo(){
         return model.getRollNo();
      }
      public void updateView(){
view.printStudentDetails(model.getName(), model.getRollNo());
      }
   }
```

Listagem 3: Código-fonte da classe **StudentController**.
(http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm)

Já a classe **MVCPatternDemo**, representada na Listagem 4, realiza a execução da aplicação de exemplo, da seguinte forma:

a) Inicialmente, um objeto do tipo **Student** (Modelo) é criado, a partir de dados supostamente provindos de uma base de dados. Na prática, os dados são fictícios, e não envolvem o acesso a um SGBD. O conceito, entretanto, é o mesmo.

b) Em seguida, é criado um objeto do tipo **StudentView**. Este objeto representa a camada de Visão.

c) Na sequência, um objeto do tipo **StudentController** é criado, sendo passados ao seu construtor as referências para os objetos de Modelo e Visão.

d) O **Controlador**, logo a seguir, atualiza a camada de Visão, com os dados provindos do Modelo.

e) A aplicação de exemplo prossegue alterando agora alguns dados do modelo – no caso, o nome do estudante – e em seguida o Controlador atualiza novamente a camada de Visão.

```
   public class MVCPatternDemo {
      public static void main(String[] args) {
//fetch student record based on his roll no from the database
         Student model  = retriveStudentFromDatabase();
         //Create a view : to write student details on console
         StudentView view = new StudentView();
    StudentController controller = new StudentController(model, view);
         controller.updateView();
         //update model data
         controller.setStudentName("John");
         controller.updateView();
      }
      private static Student retriveStudentFromDatabase(){
         Student student = new Student();
         student.setName("Robert");
         student.setRollNo("10");
```

```
            return student;
        }
    }
```

Listagem 4: Código-fonte da classe **MVCPatternDemo**.
(http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm)

Este exemplo poderia perfeitamente ser implementado de forma gráfica, incluindo inclusive campos de entrada de textos que permitam a entrada de dados a partir do usuário, bem como botões de controle que permitam o acionamento direto da camada de Controle por parte do usuário.

De toda forma, a aplicação cumpre o seu objetivo de ilustrar, de forma didática, o mínimo de objetos necessários para a composição de uma aplicação que implemente o padrão MVC.

5. Considerações Finais

O padrão MVC está presente hoje em diversas aplicações, é suportado pelas principais linguagens de programação atuais, bem como é provido pelos principais *frameworks* de desenvolvimento, tanto para Web quanto para *Desktop*.

Outros exemplos de aplicações básicas e didáticas do padrão MVC podem ser vistos em:

- ECKSTEIN, R. **Java SE Application Design With MVC**. Oracle Corp. http://www.oracle.com/technetwork/articles/javase/index-142890.html
- **CS3443**. Examples of Model-View-Controller Pattern. http://www.cs.utsa.edu/~cs3443/mvc-example.html

Como exemplos de *Frameworks* que implementam o padrão MVC, pode-se citar (organizados pela linguagem de referência que utilizam):

```
ASP
    ASP Xtreme Evolution
    Toika
    AJAXED
.NET
    ASP.NET MVC – oficial da Microsoft
    C# – oficial da Microsoft
    Versões Existentes : MVC 2, MVC 3, MVC 4,MVC 5
    ASP .NET MVC na prática – Comunidade
Harmony
    Harmony Framework – Oficial da Vilesoft
Java
    Apache Struts
    Brutos Framework
    Click Framework
    JSF
    Mentawai
    Neo Framework
    PlayFramework
    Spring MVC
    Tapestry
```

```
        VRaptor
        WebWork
    Perl
        Catalyst
        Mojolicious
        Gantry
    PHP
        Akelos
        CakePHP - para as versões 4 e 5
        CodeIgniter - para as versões 4 e 5
        FuelPHP -    para versões 5.3+
        iGrape
        Kohana Framework - para a versão 5
        LightVC - para a versão 5.
        Laravel - para a versão 5.3+
        Megiddo - para a versão 5
        Oraculum PHP Framework - para a versão 5
      PageletBox - IDE para Celular WAP com framework MVC embutido em PHP 5
       Phalcon - Extensão em C visando alta performance e baixo consumo de
recursos - para a versão 5
        PHPBurn
        PHPonTrax - para a versão 5
        PRADO - para a versão 5
        Seagull
```

```
        Spaghetti*
        Symfony – para a versão 5
        Vórtice Framework
        XPT Framework – para a versão 5
        Yii Framework – para a versão 5
        Zend Framework – da ZEND, mantenedora oficial do PHP 5 no padrão MVC
        Zend Framework – da ZEND, mantenedora oficial do PHP 6 no padrão MVC
    Python
        Django
        TurboGears
        Web2py
        Zope / Plone
    Ruby
        Rails
        Merb
```

(WIKIPEDIA, 2016. MVC. https://pt.wikipedia.org/wiki/MVC).

Tal extensão do uso do Padrão MVC demonstra sua ampla utilização no mercado de desenvolvimento de software, envolvendo tanto aplicações para a Web quanto para desktop.

Sua compreensão e domínio são, portanto, altamente recomendáveis para profissionais da área de Computação, em particular os de desenvolvimento de Sistemas.

6. Referências Bibliográficas

ALEXANDER, C., ISHIGAWA, S., SILVERSTEIN, M., IACOBSON, M., FIKSDAHL-KING, I., e ANGEL, S.. ***A Pattern Language***. Oxford University Press, New York, 1977.

DALLING, T. ***Model View Controller Explained***. 31/05/2009. Disponível em: http://www.tomdalling.com/blog/software-design/model-view-controller-explained/.

DESTRO, Daniel. ***Implementando Design***. 2011. Disponível em: http://www.guj.com.br/articles/137.

GAMA, E., HELM, R., JOHNSON, R., VLISSIDES, J. ***Design patterns – elements of reusable object-oriented software***. Addison Wesley Longman. 1995.

TUTORIALSPOINT. ***Design Patterns – MVC Pattern***. Disponível em: http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

WIKIPEDIA. ***Model-view-controller (MVC)***. Disponível em: https://pt.wikipedia.org/wiki/MVC