* Para ter chegado a este nível, você já deve ter conhecido diversos recursos de linguagem Java e tecnologias auxiliares, como Servidores Web e Bancos de Dados, para produzir aplicações informatizadas. No entanto, como podemos garantir que tudo o que produzimos ficará de fácil manutenção posteriormente? Em outras palavras, como podemos garantir que amanhã ou depois nosso código possa ser ampliado, refatorado ou modificado por terceiros, de forma que não ocorra prejuízos de entendimento e manutenção?

Para facilitar estas transformações que podem ocorrer no código, neste primeiro tópico, estudaremos, inicialmente, sobre o surgimento do MVC, com o intuído de melhorar e agilizar a produção de aplicações através de uma divisão de papeis, assim como a sua conceituação de forma geral. Também discutiremos sua origem e as motivações que nos levam a utilizar esse padrão arquitetural em projetos de pequeno, médio e até mesmo grande porte, além de entendermos como o MVC pode facilitar a manutenção desses projetos.

* Existe um segredo para que, em geral, não percamos os objetos sob nossa posse: deixá-los, após o uso, no mesmo local de origem e, se possível, no mesmo estado. Essa é a melhor forma de não ter dificuldades de encontrar o objeto quando precisar utilizá-lo.

Com os sistemas de informações não é tão diferente. É relevante aprender a programar de forma certa e organizada. Um bom programador deve ser responsável por manter a ordem em seu código-fonte.

Reflita: você se lembra perfeitamente de um código que fez há uma semana, um mês ou um ano? Quem ou o quê irá garantir que nosso código será de fácil entendimento para extensão por terceiros ou por nós mesmos, após um certo período de tempo?

Pensando nesses riscos, foram criados alguns protocolos, metodologias, regras e padrões arquiteturais para serem seguidos, com o intuito de viabilizar a melhor eficiência, disponibilidade, manutenibilidade, confiabilidade e organização do software. Assim, criou-se o padrão MVC, que visa prover essas funcionalidades.

* **1.1 O que é o Model View Controller?**

O Model View Controller (MVC), em português, Modelo Visão Controlador, é um padrão arquitetural de sistemas de informação que separa a lógica de programação em camadas, de modo que cada uma terá responsabilidades distintas para proporcionar a informação requisitada pelo usuário de maneira organizada e sem conflitos.

Assim, no modelo MVC, temos três camadas: model, view e controller. Cada uma tem sua função e são independentes uma das outras, facilitando a manutenção e escrita do código.

Percebe-se que a reutilização de código é um motivo relevante pelo qual esse padrão vem a ser constantemente utilizado no mercado de T.I, pois diminui o custo de projetos posteriores em tempo e, consequentemente, em dinheiro. Além de viabilizar o desenvolvimento paralelo de código, pois, em virtude do sistema se encontrar dividido em camadas, distintas equipes de programadores podem paralelizar a codificação para desenvolverem camadas distintas que irão se complementar e interagir ao uni-las em uma única estrutura.

Um projeto organizado em camadas padronizadas pode ser facilmente mantido e compreendido posteriormente, além de separar responsabilidades e características em comum sob distintos níveis de abstração.

### 1.2 Como surgiu o MVC?

O padrão arquitetural MVC surgiu em 1979 por um funcionário norueguês da empresa Xerox PARC, chamado Trygve Reenskaug. Até então, conceitos de padrões de projeto já existiam no mercado, porém nada claramente definido em virtude da não tão avançada tecnologia presente na época. A ideia de Reenskaug era separar o projeto de software em três camadas: modelo, visão e controlador, a fim de reduzir o acoplamento (grau que uma classe conhece a outra) e dependabilidade entre partes do sistema, além de melhorar a coesão (especificidade de uma classe, o quão objetiva ela é) das classes.

A manifestação deste padrão aconteceu quando Reenskaug estava desenvolvendo uma linguagem de programação, chamada Smalltalk (também orientada ao objeto) e, posteriormente, o MVC passou a ser implementado em outras linguagens orientadas aos objetos e também frameworks de diversas plataformas.

#### Saiba Mais



Você sabia que Trygve Reenskaug além de cientista da computação também era professor da Universidade de Oslo? E que a motivação para o surgimento do MVC se deu em virtude dos primeiros sistemas que utilizavam interfaces gráficas?

Conheça mais sobre o assunto em: [http://www.digitaldev.com.br/2013/01/18/entendendo-o-mvc-model-view-controller/](https://www.digitaldev.com.br/2013/01/18/entendendo-o-mvc-model-view-controller/).

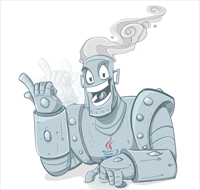
* Provavelmente você já tenha feito códigos grandes em uma mesma classe ou até no mesmo arquivo. A princípio, isso parece ser mais fácil e tranquilo, tudo parece bem organizado, disponível e de fácil acesso para quando você precisar. Porventura, isso demonstra facilidade de programação, talvez em outros paradigmas de programação seja algo viável. No entanto, em orientação a objetos, não podemos trabalhar dessa forma, pois a cada dia as pessoas querem aplicações que facilitam suas vidas, tenham inúmeras funcionalidades e que sejam, praticamente, a resolução de todos os problemas demandados. Todos esses requisitos exigem códigos simples e mais complexos, ou seja, significa o aumento da complexidade de um sistema que não pode fazer tudo em apenas um arquivo ou classe. Assim, é preciso ter boas estratégias para manter o projeto sempre organizado mesmo com a quantidade de código necessária. E é com esse objetivo que surge o padrão arquitetural MVC.

Além da organização, esse padrão, possibilita ganhar eficiência, escalabilidade e, principalmente, reutilizar um código. Não seria legal já deixar algo pronto para que no próximo projeto não tivéssemos trabalho dobrado? Acompanharemos como isso pode ser feito.

* No que diz respeito à organização, como já foi mencionado, o MVC é divido em três camadas, sendo que cada camada é desenvolvida independentemente uma da outra, ou seja, as funções que têm afinidades estão agrupadas em um único local. Dessa forma, gera eficiência ao programar uma aplicação ou realizar manutenções nesta, pois podemos agir diretamente no local certo, sem alterar e prejudicar outras partes do sistema.

A escalabilidade pode ser observada uma vez que um código bem dividido e organizado é facilmente distribuído e acrescido sem necessidade de um esforço maior para mantê-lo. Quanto à reutilização, temos pequenos módulos, gerados por esta arquitetura, que podem ser importados por outras, através de suas divisões em camadas.

#### Você Sabia?



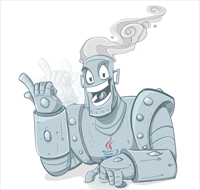
Um dos grandes desafios e metas das equipes de desenvolvimento de aplicações é produzir sistemas seguros, eficientes, de fácil manutenção e em prazos mais curtos. O padrão MVC proporciona tudo isso se for corretamente aplicado. É por essa razão que até hoje ele é bastante utilizado e cada vez mais são feitas adaptações para melhorá-lo.

* Outro motivo que deu origem ao surgimento do padrão MVC foi a necessidade da separação de responsabilidades em sistemas utilizadores de interfaces gráficas (Graphical User Interface, e em português, Interface gráfica de usuário). Todavia, o MVC não se manteve apenas neste setor, mas se prolongou para aplicações Webs, Desktops e, até mesmo, Mobile (Android, IOs, Windows Phone etc.), além da utilização em outras linguagens.

Em outras palavras, ao estudarmos o MVC, precisamos entender que este recurso não apenas será utilizado em um caso, mas sim em diversos tipos de aplicações que queiramos desenvolver e padronizar suas codificações. Você percebe a importância disso?

Aprender a implementar o MVC tornará possível a implementação de sistemas em múltiplas plataformas, utilizando um só padrão que diminuirá seu esforço e ajudará na estruturação de projetos de software dos mais variados tipos e plataformas.

#### Você Sabia?



Além disso, o MVC também é implementado em diversas linguagens de programação extremamente populares, como C#, Ruby, PHP, JavaScript, Python, entre outras que disponibilizam frameworks, implementando o padrão. Assim, você estará expandindo seu nível de conhecimento, até mesmo quando desejar aprender outra linguagem de programação que não seja Java.

* Conforme o que foi apresentado, o MVC é de grande importância para uma maior facilidade na hora da programação por conta da organização que ele estabelece através da divisão do código em blocos, facilitando a programação e a reutilização deste. Assim, fica mais fácil de compreender por que as empresas de desenvolvimento de software investem cada vez mais em padrões arquiteturais, principalmente no MVC. Elas visam o ganho de tempo, a qualidade e o menor custo.

Enfim, neste tópico, aprendemos, a origem e os motivos que ocasionaram o surgimento do padrão MVC, além de abordar sua relevância e oportunidade de reutilização em plataformas de desenvolvimento distintas. Conhecemos a história e a valorização desse padrão, além de observarmos os principais motivos pelos quais devemos utilizá-lo.

No próximo tópico, iremos entender como se dividem as três camadas que compõem o MVC e conheceremos a responsabilidade de cada uma e a relação entre elas.

No tópico anterior, estudamos como se deu o surgimento de um sistema que é dividido em camadas e que se distribui para proporcionar organização, escalabilidade e reuso do sistema. Estudamos as principais vantagens desse padrão e de sua utilização, assim como as facilidades que ele pode proporcionar em um projeto de programação.

Agora, neste tópico, iremos abordar as três camadas do MVC: camada Modelo, camada Visão e camada Controlador. Também discutiremos sobre suas principais responsabilidades dentro do sistema e observaremos a interação que existe entra cada uma delas. Você aprenderá que elas têm diferentes utilidades a partir do objetivo e da responsabilidade de cada uma.

* De modo geral, as camadas estão interligadas e cada uma cumpre uma função que está relacionada com a função das outras duas camadas.

A camada Modelo armazena informações e notifica as outras duas camadas, Visão e Controlador, que estão associadas a ela, quando ocorre alguma mudança em seu estado. Essas notificações permitem que a camada Visão produza saídas atualizadas sobre o estado atual do sistema em virtude de algum serviço requisitado pelo utilizador do sistema.

A camada Visão é responsável por produzir saídas atualizadas, isto é, tudo que permite a interação do usuário com o sistema, seja por meio de visualização, toque, gestos, cliques com o mouse ou por meio de navegação, com o uso do teclado. Neste exato momento, para que você esteja absorvendo essas informações, por meio de sentidos humanos, foi relevante que a equipe de desenvolvimento proporcionasse a camada de Visão, sempre retornando informações precisas e requisitadas pelos utilizadores do sistema.

Já a camada Controlador funciona como uma ponte entre a camada Visão e a camada Modelo, além de fornecer a conexão com outras aplicações externas. Todo o fluxo de requisições feitas na camada Visão irá passar pela camada Controle para que esta decida para onde sua requisição deverá ir. Em geral, serve para "avisar" ao modelo que atualize seu estado atual.

Agora que aprendemos um pouco sobre estas camadas e que sabemos que há uma interação evidente entre elas, observaremos de modo aprofundado cada uma das camadas, seus papeis e como elas podem ser divididas.

### 2.1.2 Camada Modelo

Essa camada é responsável pelo domínio da aplicação. Sobre o que se trata? É uma aplicação bancária? Comercial? De jogos? Basicamente é o que move a aplicação, pois grande parte da lógica de negócio, ou seja, a parte que se encarrega dos processos do negócio, como controle de inventário, compras, entre outros, deve se encontrar nessa camada. Entre alguns dos objetivos específicos estão:

* 1. Modelar os dados e comportamentos em razão do domínio de negócio;
  2. Manipular e gerar dados;
  3. Encapsular dados e comportamentos dos objetos, independente da visão.

A leitura, a escrita, a manipulação e a validação de informações se darão na camada Modelo. Digamos, por exemplo, que, em nosso sistema de controle de estoque, temos uma classe chamada Produto que contém informações particulares e genéricas dos produtos de um supermercado. Ela nos informa valores, quantidades, descrições e outras características.

Não é interessante que as descrições dos produtos sejam vazias ou que sua quantidade atual não seja informada ao ser realizado um cadastro, pois todas essas verificações, assim como a classe Produto fazem parte da camada Modelo.

* Não é uma regra, mas, em geral, quase tudo aquilo que você consegue perceber no mundo ao redor se trata da camada Modelo. Os objetos de negócio, os objetos que interagem diretamente com a lógica por trás das operações de negócio como, por exemplo, as operações de venda e o controle de estoque, irão se encontrar nesta camada.

Se estamos lidando com um sistema hospitalar, os objetos, Médico, Paciente, Enfermidade, Sala e todos os outros que se associam ao contexto irão fazer parte da camada Modelo, já que estes objetos estão interligados e interferem diretamente na lógica de negócio de um hospital, como consultas, gerar relatórios, entre outros.

Entre as regras de negócio, como as regras de um hospital, citam-se, por exemplo:

* 1. Nenhum paciente poderá ficar sem um médico associado;
  2. O objeto Enfermidade não poderá existir se não existir um objeto Paciente;
  3. Uma sala não pode ser ocupada por um objeto Enfermidade, somente pelos objetos: Médico e Paciente.

Estes devem e fazem parte da camada Modelo, pois lidam com questões de domínio hospitalar. Em resumo, temos como responsabilidade da camada Modelo: dados da aplicação, regras de negócio, lógica e funções.

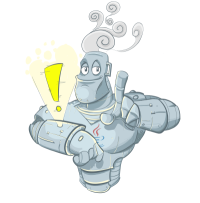
* A camada Modelo encapsula todo o estado e comportamento da aplicação, faz uma conexão praticamente direta com a fonte de dados da aplicação, em geral, o banco de dados. Já percebeu que diversas tabelas da sua base de dados estão representadas no código Java como classes de domínio? Como exemplo, temos: Pessoa, Produto, Aluno, Professor, Veículo e outras.

A lógica de negócio, que está inclusa na camada Modelo, não fica apenas separada da camada Visão como, por exemplo, uma interface gráfica, além disso, a primeira sequer sabe da existência desta. O modelo é a junção dos dados e dos métodos que os manipulam. Com isso, o reuso provido pelo MVC se dá, principalmente, na camada Modelo, sendo este o núcleo funcional da aplicação.

Quer uma prova desse reuso? Digamos que tenhamos que desenvolver uma aplicação Mobile, Web e outra Desktop. As classes de domínio do negócio podem ser feitas em apenas um projeto à parte como, por exemplo, o Projeto Núcleo e todos os outros três projetos terão acesso à camada Modelo desse núcleo, acessando as regras de negócio ao mesmo tempo. Qual parece ser a melhor opção? Fazer apenas um projeto ou replicar as regras de negócio e classes do Modelo nos três projetos em distintas plataformas? Além de duplicarmos o código, iríamos ter um trabalho triplicado ao desejar alterar alguma regra de negócio simples.

Apesar de a camada modelo realizar todas estas funções, o usuário final não tem acesso direto a esta camada, visto que esta não é sua funcionalidade, por isso, vamos agora estudar a camada Visão.

#### Atenção!



Uma grande vantagem de separar toda a lógica de negócio em nossa camada Modelo é deixar de maneira privada todas as regras de negócio, de tal forma, que as camadas Visão e Controle não poderão interferir e ocasionar conflitos em nosso domínio, mas irão auxiliar a camada Modelo atendendo suas requisições e atualizando seu estado atual.

### 2.1.3 Camada Visão

Diante do que você acabou de aprender sobre a camada Modelo, você deve ter compreendido como as informações mantidas nesta camada, ou seja, como toda a lógica de negócio, seus métodos e objetos são acessados pelo utilizador da aplicação. Assim, vem a questão: como o usuário irá perceber valores e até mesmo realizar requisições ao sistema? Qual a forma que o usuário poderá requisitar ou prover informações ao sistema?

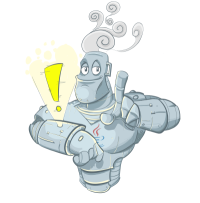
Tudo isso se dará por meio da camada Visão, onde o estado atual dos objetos da camada Modelo poderão ser facilmente acessados. A exemplo disso, temos a interface gráfica de um sistema que exibe informações pessoais de uma rede social: mensagens, curtidas, fotos, comentários etc. Através desta interface, o usuário tem acesso a todas as funcionalidades do sistema, que utilizam a lógica de negócio provida através da camada Modelo.

Assim, a principal função da camada de visão é a representação de dados, onde um usuário pode observar estes dados presentes na aplicação ou inserir dados que serão utilizados por essa camada.

* Nessa representação, a camada Visão tem o objetivo de representar a interface de apresentação entre a camada Modelo e o usuário do sistema. Dessa forma, nem toda informação proveniente da camada Modelo é interessante para que o usuário absorva, então a Visão, além de proporcionar a interpretação das informações, também desempenha o papel de filtro, apenas proporcionando detalhes que são relevantes para o usuário.

Um bom exemplo de camada de Visão, em aplicações desktop, são as GUIs (interfaces gráficas) repletas de botões, campos de textos, imagens, formulários, entre outros recursos que viabilizam o retorno e envio de dados entre sistema e usuário. Em uma plataforma Web que utilize o MVC, temos, como exemplo da camada Visão, as páginas HTMLs recheadas de estilos por meio do Cascading Style Sheets (CSS) e eventos com o uso do JavaScript. Tudo isso faz parte da camada Visão que tenta, da melhor forma possível, apresentar ao usuário o que se passa na camada de Modelo do software.

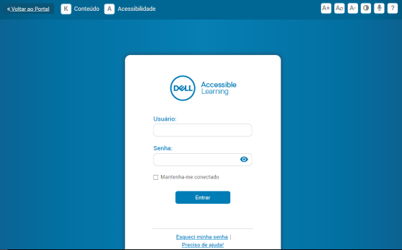
#### Atenção!



Podemos ter certeza de uma coisa: se você está captando as informações deste curso, é tudo por meio da camada Visão, pois sem ela as informações contidas nos servidores e bancos de dados não poderiam ser expostas de maneira amigável e de fácil discernimento por todos nós. Lembre-se de que o uso das interfaces gráficas foi a grande motivação para a comercialização do computador de uso pessoal, demonstrando a grande importância desta camada.

* A camada Visão não está preocupada em como ou onde a informação foi obtida, ela apenas foca em apresentar o conhecimento por meio de recursos que possibilitam a transmissão do dado. Vamos pensar juntos: uma interface gráfica apresenta dados? Uma caixa de som retorna informações por meio do áudio? A resposta em ambos os questionamentos é positiva, referem-se à camada Visão.

A interface direta com o usuário do sistema é feita por meio dessa camada, por exemplo, podemos citar a interface de acesso ao curso, onde o aluno pode interagir com o sistema, provendo informações para acessá-lo ou resolver questões, como reescrever estas informações através das aulas e videoaulas.

Figura 1 – Camada Visão de acesso à plataforma. Fonte: Dell Computadores/LE@D.

### 2.1.4 Camada Controle

O próprio nome desta camada já sugere a sua principal função: controlar. No caso, ela controla a comunicação entre a camada Visão e a camada Modelo. Já foi discutido anteriormente que a Visão apresenta as informações contidas na camada Modelo, porém a ponte entre ambas é realizada através da camada Controle. O fluxo de requisições e dados recebidos/enviados por ambas também é responsabilidade desta camada.

A importância da camada Controle se dá uma vez que todo o sistema é independente, ou seja, nenhuma camada tem acesso aos dados de uma outra camada, sendo necessário, assim, um intermediador de informações. Desta forma, podemos manter o controle sem interligar as camadas existentes.

Entre algumas de suas funções específicas estão:

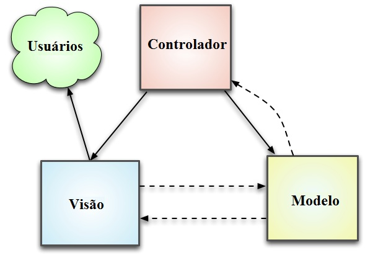
* 1. Controlar e mapear ações;
  2. Determinar o fluxo de apresentação;
  3. Ser a camada intermediadora entre as camadas Modelo e Visão.
* As requisições realizadas em um sistema de informação, por meio da camada Visão, devem chegar à camada Modelo por um controlador de requisições, eventos recebidos de uma interface gráfica por meio de cliques do mouse, comandos de voz, preenchimento de formulários, navegação por teclado, entre outros, ocasionam eventos no sistema que devem ser direcionados para o caminho correto. Não seria interessante, em um sistema de controle financeiro, o usuário fornecer um crédito em sua conta e o controlador direcionar aquela requisição para o módulo de débito, ocasionando um prejuízo financeiro por conta de um erro de controle do fluxo. Em virtude disso, os controladores são responsáveis por validar requisições e direcionar a logística para a camada de Modelo responsável em lidar com a intenção do usuário. Perceba que de tal maneira, não existe uma interação direta entre Visão e Modelo, organizando o código e impedindo que a lógica de negócio se misture com a interface gráfica do sistema.

Em regra, a camada Controle recebe a requisição, envia para o Modelo correto e espera o retorno desse Modelo. A camada Controle não se preocupa se a resposta será positiva ou negativa, apenas recebe requisições e retorna respostas. Dessa forma, podemos dizer que ela trabalha de modo a fazer o intermédio entre as outras duas camadas. Esse processo pode ser observado através do fluxo dos dados, quando há a interação entre as três camadas, o que discutiremos a seguir.

### 2.1.5 Modelo, Visão, Controle e suas Interações

Até o momento, estudamos as três camadas e suas principais funções, que são aspectos muito importantes para a compreensão do processo de funcionamento do MVC. O que também é bastante relevante é a interação que ocorre durante o desempenho de atividades das camadas, assim como a forma como elas se comunicam.

É importante ter em mente que cada camada tem sua função e que uma é totalmente independente da outra, ou seja, as camadas não interferem e nem modificam os arquivos e métodos que não são de sua responsabilidade. Por este motivo, é necessário haver uma forma destas camadas se comunicarem e entrarem em acordo. Para isto, temos a camada Controle, que coordena as outras duas camadas. Observe o fluxo de dados em uma aplicação MVC na imagem a seguir. Ela ilustra de maneira gráfica a interação entre as três camadas do sistema e o usuário.

Figura 2: Fluxo de dados em uma aplicação MVC. Fonte: Dell Computadores/LE@D.

* Perceba que o que ocorre durante o processo apresentado na imagem é:
  1. Os usuários requisitam para a camada de visão o retorno de algum clique, comando de voz ou inserção de dados pelo teclado.
  2. A camada de visão (única e exclusiva camada capaz de interagir com o usuário diretamente) avisa ao controlador que uma requisição foi solicitada;
  3. De pronto, o controlador avisa a camada Modelo a necessidade de uma resposta e retorno imediato para a requisição do usuário.
  4. O modelo recebe a requisição, processa as informações, talvez consulte um banco de dados externo e posteriormente retorna a resposta para o controlador.
  5. O controlador retorna a Visão o resultado, e este último descreve para o usuário por meio de gráficos, tabelas e imagens o retorno da requisição.

De modo mais dinâmico, acompanhe essa interação entre as camadas a partir do diálogo a seguir:

Visão (para Controlador): Fala, Controlador! O usuário acabou de pedir para acessar a plataforma dellead! Estes são os dados dele (envia os dados do usuário).

Controlador (para Visão): Beleza. Daqui a pouco te mando a resposta.

Controlador (para Modelo): Modelo, preciso que você verifique se um login pode ser realizado com esses dados (envia os dados do usuário).

Modelo (para Controlador, após a verificação): Os dados são válidos. Segue a resposta de login.

Controlador (para Modelo): Beleza! Obrigado.

Controlador (para Visão): Visão, o usuário informou os dados corretos. Aqui estão os dados dele (envia os dados do usuário). Você pode carregar a página para que ele tenha acesso ao curso.

Visão (para Controlador): Valeu (atualiza a página para o usuário).

* Durante este tópico, acompanhamos em detalhes a descrição das responsabilidades de cada camada do padrão arquitetural MVC. Podemos observar que cada camada tem sua finalidade específica e é interessante que uma não sobreponha a outra, também não tire uma responsabilidade que não é de sua respectiva competência. De tal forma, o projeto seguirá o padrão definido e cada camada irá trabalhar colaborando de acordo com seus objetivos.

No próximo tópico praticaremos a aplicação do padrão MVC em um projeto Java criado na IDE Eclipse, desde o início. Com toda a estrutura dividida nas três camadas descritas neste tópico, acreditamos que você esteja pronto para observar estes conhecimentos adquiridos na prática, então vamos lá!

##  TÓPICO 3 – Implementação MVC

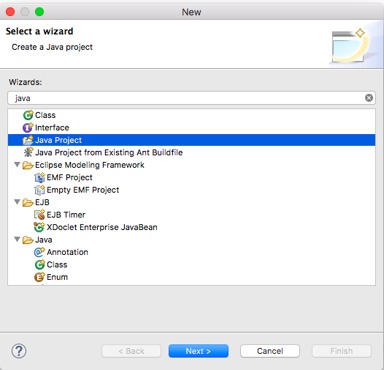
Objetivo

* Compreender o uso da arquitetura MVC em um projeto Java;
* Criar o arquétipo de um projeto baseado no padrão MVC.

Tudo o que estudamos até agora servirá como base para a utilização prática dos conceitos. Neste tópico, criaremos um projeto genérico Java na IDE Eclipse e iremos estruturar um arquétipo no padrão MVC, além de instanciar classes para observar o fluxo de interação entre as camadas Modelo, Visão e Controle.

No projeto, poderemos perceber a diferença de cada umas dessas camadas, além de observarmos como cada uma delas trabalha individualmente. Também aprenderemos a organizar e codificar este tipo de projeto. Desta forma, teremos uma noção prática da organização de um projeto MVC, observando de perto seu fluxo de dados.

* Para iniciarmos, vamos abrir nossa IDE eclipse. No IDE, vamos criar um novo Projeto Java através do menu file, na opção new e, finalmente, Java Project ou através do atalho Ctrl + N, onde você pode acessar este menu e procurar também a opção Java Project.

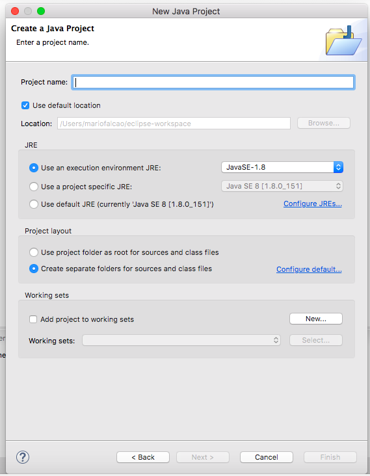
Figura 3 – Criação de um novo projeto Java Fonte: Dell Computadores/LE@D.

* A seguir definiremos o nome do nosso projeto na caixa de texto Project Name, o nome exemplo pode ser: "Projeto".

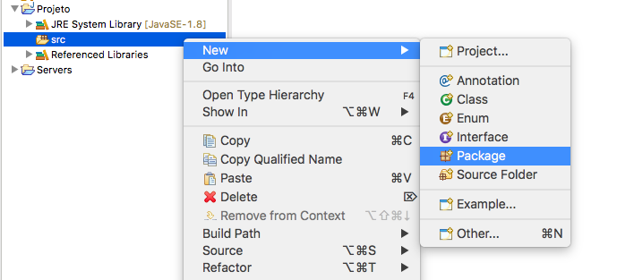
Após isso, clique em Finish ou pressione Alt + N para avançar e quando necessário Alt + F para finalizar. Você perceberá que o novo projeto está criado. Observe que, para realizarmos o arquétipo do nosso projeto utilizando o padrão MVC, não é necessário nenhuma biblioteca externa ou framework. Tudo se trata de uma maneira para organizar os arquivos e interações entre as classes.

Através desses fundamentos, teremos que criar três pacotes distintos, onde cada um irá representar uma camada do MVC, ou seja, são eles:

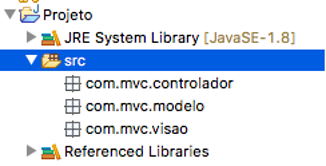
* 1. com.mvc.modelo
  2. com.mvc.visao
  3. com.mvc.controlador

Figura 4 – Seleção do nome para o projeto criado. Fonte: Dell Computadores/LE@D.

* Para criar esses pacotes, devemos acessar o menu file, selecionar a opção new e então o botão package, conforme descrito na imagem. Lembre-se que você também pode acessar este menu através do atalho ctrl + n e então selecionar a opção package.

Figura 5 – Criação de um novo pacote. Fonte: Dell Computadores/LE@D.

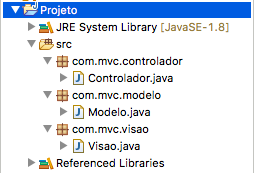
* Devemos realizar essa sequência de criação de pacotes três vezes para criar cada uma das camadas do MVC. Como resultado final, desta parte, teremos em nosso sistema de arquivos, três pacotes criados, cada um com a responsabilidade de armazenar os arquivos próprios da arquitetura MVC:

Figura 6 – Divisão de pacotes em um projeto MVC Fonte: Dell Computadores/LE@D.

Até aqui já temos o arquétipo montado, esses pacotes estão prontos para receberem a criação de cada uma das classes necessárias para que seja concretizada a interação entre as partes: modelo, visão e controlador.

* Para que as classes possam ser instanciadas, devemos criar as novas classes dentro dos pacotes recentemente criados, cada pacote deve armazenar a classe que faz parte do seu escopo.

Vamos utilizar este procedimento, descrito no parágrafo anterior, a fim de criar três classes (Modelo, Visao, Controlador) cada uma em um pacote do arquétipo MVC. Ao criar a classe Visão selecione a opção public static void main para que nossa função main seja automaticamente gerada dentro desta classe, ou na tela de criação da classe, pressione o atalho ctrl + v para escolher a criação automática.

Figura 7: Pacotes e suas respectivas classes em um projeto MVC. Fonte: Dell Computadores/LE@D.

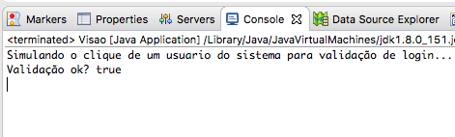
Talvez você esteja se perguntando por qual motivo temos uma função main dentro da classe Visão, todavia não temos nas outras. Para fins didáticos, a classe Visão terá essa função main para simular ações (requisições) do usuário que será o utilizador do sistema. Em outras palavras, seria o mesmo que cliques e/ou eventos que o usuário possa demandar para o sistema.

* A classe visão será descrita no código a seguir, observe que dentro da função main instanciamos o que seria a interface gráfica, ou seja, uma instância da classe Visão que chama o método simularCliqueUsuarioParaValidarLogin que irá receber como parâmetro duas strings: usuário e senha. Em outras palavras, de forma abstrata isso pode fazer referência a um formulário com duas caixas de texto que recebe usuário e senha, e em seguida o utilizador do sistema clica no botão OK enviando os parâmetros a fim de serem validados através da classe Modelo, lembre-se que a classe Controlador irá ser a ponte que transportará essas informações.
* package com.mvc.visao;
* import com.mvc.controlador.Controlador;
* public class Visao {
* public static void main(String[] args) {
* Visao visao = new Visao();
* visao.simularCliqueUsuarioParaValidarLogin("dellead","123456");
* }
* public void simularCliqueUsuarioParaValidarLogin (String usuario, String senha) {
* System.out.println("Simulando o clique de um usuario do sistema para validação de login..."); Controlador controlador = new Controlador();
* boolean resultado = controlador.validarLogin(usuario, senha);
* System.out.println("Validação ok? "+resultado);
* }
* }

* A classe Controlador, como discutido anteriormente, será a responsável em receber a requisição da visão e encaminhar para o Modelo correto, ou seja, a classe Modelo que tem a incumbência de validar o usuário e senha conforme enviados como parâmetros. Abaixo temos a implementação da classe controlador.
* package com.mvc.controlador;
* import com.mvc.modelo.Modelo;
* public class Controlador {
* public boolean validarLogin(String usuario, String senha) {
* Modelo modelo = new Modelo();
* return modelo.validarLogin(usuario, senha);
* }
* }

* Em aplicações mais completas a estrutura segue essa mesma lógica, todavia esse usuário e senha poderiam ser consultados de um servidor externo ou banco de dados, não é verdade? Algo mais próximo da realidade de produção. Todavia, não se engane! Estamos aprendendo a fazer o simples, porém de forma correta, posteriormente apenas devemos reproduzi-lo com um pouco mais de complexidade. Confira o código.
* package com.mvc.modelo;
* public class Modelo {
* private String usuario = "dellead";
* private String senha = "123456";
* public boolean validarLogin(String usuario, String senha) {
* return this.usuario.equalsIgnoreCase(usuario) && this.senha.equalsIgnoreCase(senha)? true : false;
* }
* }

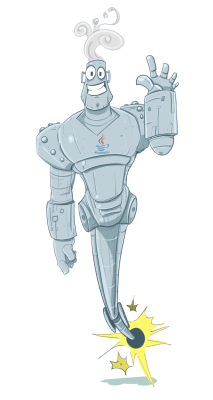
* Agora que temos nossas três classes prontas, iremos executar a classe Visão como uma aplicação Java, você pode por exemplo utilizar o atalho do teclado Alt + Shift + X + J. Ao realizá-lo notaremos no console que a validação foi positiva e o usuário está pronto para logar em nosso sistema, como ilustra a imagem, onde podemos observar o retorno do método que especificamos com a string “validação ok”, desta forma, caso o exemplo execute com sucesso, esta string é então exibida na tela do console:

Figura 8 – Fluxo de dados em uma aplicação MVC. Fonte: Dell Computadores/LE@D.

* Neste tópico, observamos na prática a implementação de um sistema utilizando o padrão MVC (Modelo Visão Controlador). Compreendemos que estas classes estão separadas uma das outras, através de pacotes diferentes, e que cada uma têm sua própria função definida neste sistema.

Apesar de não precisarmos utilizar nenhum framework ou aplicação externa, aprendemos que é simples codificar utilizando o padrão MVC e que através disto, temos um maior rendimento de tempo, através da reutilização de códigos, por exemplo.

### Fechamento



Chegamos ao final da nossa aula, aprendemos que o padrão arquitetural MVC surgiu com o intuito de melhorar o desenvolvimento de software, proporcionando fácil manutenção, escalabilidade e organização. Também entendemos que ele está dividido em três camadas: Modelo, Visão e Controlador, cada uma com uma responsabilidade específica. Sua implementação é algo global que atinge toda a estrutura do projeto de software de maneira positiva, continue estudando esse padrão, suas variantes e frameworks que dão suporte ao mesmo. O estudo não termina aqui, apenas começa! Até a próxima aula.

#### Referências

* + Caellum. Disponível em: <https://www.caelum.com.br/apostila-java-web/mvc-model-view-controller/>. Acesso em: 25 out. 2018.
  + MVC a final o que, Allan Ramos. Disponível em: <https://tableless.com.br/mvc-afinal-e-o-que/>. Acesso em: 25 out. 2018.
  + MVC. Rodrigo Rebouças. Disponível em: [http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/arqu/mvc/mvc.htm](https://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/arqu/mvc/mvc.htm" \t "_Blank). Acesso em: 25 out. 2018.