# Unidade 4

Nos capítulos anteriores foi abordado todo o fundamento da POO, aqui é apresentada a UML, uma grande facilitadora da POO, com ela economizamos muito do nosso tempo gasto com o planejamento, tornamos a leitura e interpretação dos programas muito mais dinâmica e fluída. A base deste capítulo será apresentar diversas representações gráficas de diagramas de classes, relacionamento de objetos, associações, composições, agregações, assim como as principais convenções para seu uso e ainda uma introdução sobre MVC que será melhor abordado no Capítulo 5.

## Aula 1 - UML

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para

* Interpretar um diagrama de classes
* Desenhar um diagrama de classes
* Entender o relacionamento entre as classes

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 50 minutos de aula expositiva;
* 20 minutos para tirar as dúvidas dos alunos;
* 20 minutos para desenvolver as atividades propostas para a turma.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* O que é UML;
* Diagrama de classes;
* Classes, métodos e atributos;
* Interfaces;
* Classes abstratas.

### PONTOS IMPORTANTES

Para garantir ao aluno um aprendizado significativo, resgate os principais conceitos de cada tópico abordado na aula. Neste momento, é importante que você utilize dicas e sugestões para reforçar os temas trabalhados, oferecendo também exemplos que possibilitem a relação entre teoria e prática.

### 1.1 - O que é UML

Para ser um bom profissional é essencial que o programador aprenda a UML, pois é bem provável que a maioria dos programadores tenham que um dia projetar um software. Este cargo é chamado Analista de Sistemas e está na maioria das empresas. Se não formos o projetista, com certeza seremos o desenvolvedor, que terá que interpretar os diagramas do projetista.

### 1.2 - Diagrama de classes

Neste tópico começamos a explorar o UML com o diagrama de classes. O diagrama de classe é o principal dentre os diagramas da UML. Além de estar no senso comum é o mais amplamente utilizado.

#### 1.2.1. Perspectivas de construção do Diagrama de Classes

Educador, do grupo de diagramas estruturais, **o diagrama de classes é o mais importante** e relevante. Do grupo dos diagramas comportamentais, diagrama de casos de uso, sequência e comunicação. Os diagramas comportamentais são os menos utilizados pelos programadores, pois estão mais voltados a definir “o que fazer” do que “como deve ser feito”.

### 1.3 - Classes, métodos e atributos

Este tópico define como uma classe é representada no diagrama de classes. Uma classe é representada conforme a imagem abaixo:



Esta aula é bastante visual, já que se trata de diagramas.

### 1.4 – Interfaces

Este tópico define como uma interface é representada no diagrama de classes. As interfaces são



Esta aula é bastante visual, já que se trata de diagramas.

### 1.5 - Classes abstratas

Este tópico define como uma classe abstrata é representada no diagrama de classes.

Identificar métodos estáticos no diagrama de classes é opcional, pois descrever como as classes se comunicam não é sua função, mas o diagrama de sequência tem a função de fazê-lo. O diagrama de classes descreve a relação que uma classe tem com a outra e não como um método funciona.

Opcionalmente você pode representar um método (ou atributo) estático no diagrama de classes sublinhando o mesmo.

### 1.6 – Resumo

Retome os tópicos abordados na aula questionando aos seus alunos.

O que aprendemos hoje pessoal? Você pode fazer isso em forma de Quizz ou para engajar sua turma pode finalizar a aula, perguntando se alguém tem dúvidas sobre aula de hoje e quem se habilitaria a resumir o que foi visto, faça isso toda aula com um aluno diferente para treiná-los quanto ao desenvolvimento de comunicação e ao final da fala do aluno, resuma os tópicos.

Neste capítulo o aluno aprendeu sobre UML e seu vasto mundo de diagramas. Um deles foi o diagrama de classes.

### 1.7 – Exercícios

|  |  |
| --- | --- |
| **1.** | **O que é UML?**  Resposta: UML é uma sigla para Unified Modeling Language, ou seja, Linguagem de Modelagem Unificada. A UML é uma forma de representar graficamente um sistema de software. A UML não é uma metodologia ou regra, ou seja, ela não dita o que deve ser feito ou como o sistema deve ser projetado, mas ela nos auxilia a visualizá-lo de forma prática e a entender o relacionamento entre os objetos. |
| **2.** | **O que é um Diagrama de classes?**  Resposta: O Diagrama de Classes está no grupo de diagramas estruturais da UML e é um dos diagramas mais utilizados pelos programadores e analistas de sistemas, pois este permite uma visualização geral das Classes com seus atributos e métodos e também como os objetos se relacionam entre si. De maneira simplificada, podemos dizer que é uma representação gráfica do relacionamento entre os objetos. |
| **3.** | **Em qual fase do projeto o Diagrama de classes deve ser desenhado?**  Resposta: O Diagrama de classes é uma definição estratégica do sistema. É como um esboço do nosso programa. Este esboço é escrito e retrabalhado no início do projeto, na fase de Análise, antes da fase de desenvolvimento. |
| **4.** | **Quais são as perspectivas de construção do Diagrama de classes? Explique-as.**  Resposta: Quando projetamos, existem três perspectivas de classes que podemos utilizar: a Conceitual, a de Especificação e a de Implementação. Estas perspectivas são cruciais para desenhar e compreender os diagramas de classes.   * **Conceitual**: Um diagrama simples, que apresenta apenas análises do problema em foco, relacionadas às classes que irão executá-lo. Este apresenta apenas uma perspectiva do programa e não possui envolvimento com a linguagem a ser implementada. * **Especificação**: É obtido com a adição de mais detalhes ao modelo anterior, conforme a interface escolhida. Esta perspectiva é o projeto para a solução do problema em questão, conforme o software que foi escolhido. * **Implementação**: É a perspectiva usada com maior frequência e corresponde a implementação das classes na linguagem de programação escolhida. |
| **5.** | **Como classes, métodos e atributos são representados no Diagrama de classes? Explique com um exemplo.**  Resposta: Abaixo temos o exemplo da classe Carro representada no diagrama de classes:    Figura 4.3 |
| **6.** | **É possível que métodos e atributos ou até mesmo subdivisões sejam omitidos no diagrama? Caso a resposta for positiva, o que isto representa?**  Resposta: Sim. Representa que a classe não possui tais itens ou que já estão implícitos ou que são irrelevantes. |
| **7.** | **Como uma interface é representada no Diagrama de classes?**  Resposta: Abaixo temos o exemplo da interface Domesticable representada no diagrama de classes.    Figura 4.4 |
| **8.** | **Como as classes abstratas são representadas em um Diagrama de classes?**  Resposta: Abaixo temos o exemplo da classe abstrata Employee representada no diagrama de classes.    Figura 4.5 |

### 1.8 – TDP

#### 1.8.1 - A necessidade de diagramar

**Finalmente novos programadores chegaram para suprir a falta. Pedro irá te ajudar a programar, mas ele não está entendendo nada sobre o código e sobre o fluxo dos dados.**

**Você está aprendendo UML. Que tal implementar os diagramas agora que, com a ajuda de um novo programador, o prazo não está tão apertado assim? O diagrama vai ajudar Pedro a entender como é o funcionamento do código de uma maneira geral. Isso irá ajudar futuros programadores deste projeto também. Então ao invés de perder horas explicando linhas de código a Pedro (e a qualquer outro programador que participe do projeto), vamos criar um diagrama de classes.**

**Além disso, seu desempenho tem sido excelente. Logo os diretores irão te selecionar para coordenar outros projetos, e você não estará lá para explicar o código do projeto Filmes aos novos programadores. A UML fará isto pra você!**

#### Tarefa

**Faça o diagrama de classes dos models. Insira os atributos e métodos. Não implemente herança, interface e demais relações entre classes. Apenas crie as classes. Para trabalharmos nosso espírito de equipe, realize esta tarefa em dupla, e tenha suas anotações individuais em seu caderno.**

**Educador,** quase sempre, no mercado de trabalho, o programador estará em uma equipe. A importância de incentivar o aluno a realizar o trabalho em dupla é muito importante, pois além da atividade fluir melhor (duas cabeças pensam melhor do que uma), a carga de atividades sobre os alunos diminuirá e eles estarão prontos para trabalhar em equipes nas empresas.

A solução deste trabalho se encontra [nesta](https://drive.google.com/file/d/0B6bf60bnbpBwTHR1UUU2SDdxdlk/view?usp=sharing) imagem:



Para a diagramação desta etapa, não omitimos nenhum atributo que posteriormente gerará um relacionamento.

## Aula 2 - Relacionamento entre objetos

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para

* Interpretar o relacionamento das classes em um diagrama de classes

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 50 minutos de aula expositiva;
* 20 minutos para tirar as dúvidas dos alunos;
* 20 minutos para desenvolver as atividades propostas para a turma.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* Relacionamentos;
* Herança;
* Interfaces;
* Classes abstratas;
* Associação;
* Composição;
* Agregação.

### PONTOS IMPORTANTES

Para garantir ao aluno um aprendizado significativo, resgate os principais conceitos de cada tópico abordado na aula. Neste momento, é importante que você utilize dicas e sugestões para reforçar os temas trabalhados, oferecendo também exemplos que possibilitem a relação entre teoria e prática.

Educador, nesta aula serão apresentados diversos conceitos gráficos que são convenções para a UML. É interessante exemplificar no quadro utilizando exemplos das aulas anteriores, ssim se obtém uma relação entre um projeto UML e a POO.

### 2.1 – Relacionamentos

Relacionamento entre objetos são feitos a todo instante. Este tópico define ???

### 2.2 – Herança

Este tópico define como a relação de herança é representada no diagrama de classes.

Educador, seria interessante modelar toda árvore do reino animal, dado como exemplo na Aula 2 do Capítulo 3.

### 2.3 – Interfaces

Este tópico define como a relação de interface é representada no diagrama de classes.

### 2.4 - Classes abstratas

Este tópico define como a relação de classe abstrata é representada no diagrama de classes, mostre ao aluno os sinais gráficos e nomenclatura que as indicam.

### 2.5 – Associação

Associação é um tipo de relacionamento entre classes. Uma associação possui multiplicidade. Educador neste tópico explane a seus alunos sobre o conceito de multiplicidade.

Associações Ternárias ou N-árias são úteis para representar associações complexas, porém devem ser evitadas ao máximo, pois são difíceis de interpretar. Veja um exemplo de associação N-ária na seção 2.5.2 desta aula.

#### 2.5.1 - Binária

A associação binária representa o relacionamento entre objetos de duas classes diferentes. Esta é a mais comum entre as associações. Não se deixe confundir. Quando falamos de relacionamento, falamos a nível de objeto. O que tipifica a associação é o tipo da classe dos objetos que estão se relacionando e não quantos objetos se relacionam entre si. Um exemplo claro é PEDIDO possui ITENS:

../../../Desktop/cap4/poo_cap4_img16.png

Na imagem acima, não foi especificado multiplicidade, mas podemos deduzir que um objeto PEDIDO possui muitos objetos ITENS. São muitos objetos em jogo, mas apenas duas classes estão representadas ali.

#### 2.5.2 - Ternária ou N-árias

A associação ternária ou N-ária representa o relacionamento entre três ou mais objetos de classes diferentes. Ela é a associação mais difícil de se interpretar, pois sua leitura é omnidirecional, ou seja, qualquer sentido que você direcionar sua leitura, você terá uma interpretação diferente. Exemplo ATLETA – TREINADOR – CAMPO:



Veja que pode ser (e deve) interpretado de diferentes pontos de vista.

Educador,como este exemplo não foi dado no livro do aluno, você pode desenhá-lo junto com os alunos, no quadro negro ou nos computadores.

#### 2.5.3 – Unária

A associação unária (também chamada de reflexiva) é utilizada para representar o relacionamento entre objetos de um mesmo tipo de classe. Ela não é tão comum quanto a associação binária, mas nos deparamos muito com ela.

Geralmente para representar relações entre subclasses de uma mesma classe polimórfica, é utilizado a associação unária, com o intuito de flexibilizar as classes que se relacionam. Exemplo:

Uma pessoa, pode ser homem ou mulher, que pode ter filhos, ou ser filho de um homem ou mulher.



Educador,como este exemplo não foi dado no livro do aluno, você pode desenhá-lo junto com os alunos, no quadro negro ou nos computadores.

### 2.6 - Composição

Define que o objeto parte não existe sem o objeto todo. A lista de objetos que compõem outros objetos não faz sentido, ou não existe, sem a presença do objeto-todo. Por exemplo: Relacionamento entre Pedido e Itens. Um pedido não existe sem que haja pelo menos um item, como na Figura 4.15:

../../../Desktop/cap4/poo_cap4_img16.png

Figura 4.15

### 2.7 – Agregação

Define que o objeto parte pode existir livremente sem o objeto todo.

Educador, uma dica importante a leitura dos relacionamentos acima é que eles têm um modo padronizado para isso. Você os desenha da esquerda para a direita e lê-se neste sentido também. Para nomear um relacionamento, geralmente utilizamos verbos. Na Figura 4.17, leríamos:

Time **TEM** Atletas.

### 2.8 – Resumo

Retome os tópicos abordados na aula questionando aos seus alunos.

O que aprendemos hoje pessoal? Você pode fazer isso em forma de Quizz ou para engajar sua turma pode finalizar a aula, perguntando se alguém tem dúvidas sobre aula de hoje e quem se habilitaria a resumir o que foi visto, faça isso toda aula com um aluno diferente para treiná-los quanto ao desenvolvimento de comunicação e ao final da fala do aluno, resuma os tópicos.

Nesta aula você aprendeu sobre o relacionamento entre os objetos. Para permitir as execuções dos processos pelo nosso sistema, as classes relacionam-se entre si com o intuito de compartilhar informações que colaborem umas com as outras. Associação, herança, dependência, agregação e composição são relacionamentos entre as classes e nesta aula, você aprendeu a representá-las.

### 2.9 – Exercícios

|  |  |
| --- | --- |
| **1**. | **Como é a representação de Herança no Diagrama de classes?**  **Resposta:**    Figura 4.8 |
| **2**. | **Como definimos uma relação de interface no Diagrama de classes?**  **Resposta:**    Figura 4.9    Figura 4.10 |
| **3**. | **Qual é a representação utilizada para definir o relacionamento entre a classe abstrata e sua classe concreta?**  **Resposta:**  abstração.png  Figura 4.11 |
| **4**. | **O que é uma associação e qual o seu objetivo?**  **Resposta:** Associações são descrições de vínculos entre as classes ou mesmo vínculos com ela mesma, representados por linhas que conectam as classes envolvidas e podem conter setas de navegação entre elas. Tais vínculos indicam ligações de instâncias de uma classe com outras instâncias de outras classes. Além disso, indica troca de informações e compartilhamento de métodos entre as classes e também que uma instância de uma classe origine outras instâncias das outras classes associadas. Também identifica os níveis de dependência entre as classes envolvidas. |
| **5**. | **Quais os tipos de associação?**  **Resposta: Os tipos de associação são a** Composição, Agregação e as Multiplicidades. |
| **6**. | **O que são associações binárias? E Ternária ou N-árias? E Unária?**  **Resposta:** Relacionamento entre objetos de duas classes diferentes, relacionamento entre objetos de três ou mais classes diferentes e relacionamento entre apenas objetos da mesma classe. |
| **7**. | **O que determina a multiplicidade?**  **Resposta:** Qual das classes envolvidas em uma associação fornece informações para as outras, além de permitir especificar o nível de dependência de uma classe para com as outras envolvidas na associação |
| **8**. | **Dê um exemplo de multiplicidade ternária.**  **Resposta:**  Relação Professor - Sala de Aula - Aluno.  Relação Projeto – Empresa - Projetista.  Relação Treinador – Campo – Jogador. |
| **9**. | **Qual a diferença entre composição e agregação?**  **Resposta:** Um relacionamento do tipo agregação é caracterizado quando uma lista de objetos que está relacionada a outro objeto pode existir sem que sejam parte deste objeto. Em composição a lista de objetos que compõem um outro objeto não fazem sentido, ou não existem, sem a presença do objeto-todo. |
| **10**. | **Por que o relacionamento é entre objetos e não classes?**  **Resposta:** No diagrama de classe devemos pensar como se comportarão os objetos e não as classes. Multiplicidade não faria sentido se a relação fosse entre as classes e não objetos, afinal, existe apenas uma classe que descreve determinado objeto mas existem diversas instâncias desta classe com seus ciclos de vida. Estas instâncias se relacionam com outras instâncias. Lembre-se, a classe apenas descreve uma "coisa" ela não tem ação e nem propriedade, somente suas instâncias/objetos têm. |

### 2.10 – TDP

#### 2.10.1 Com UML não há dúvidas

**Você e Pedro modelaram as classes muito bem, porém é preciso mostrar a forma como os objetos se relacionam entre si.**

**Na UML não pode haver dúvidas. Uma UML bem definida faz com que o programador nem pense para programar. Ele só lê o diagrama e o traduz para a linguagem de programação.**

**Dentre os vários diagramas da UML, existe um chamado Diagrama de Comunicação, que detalha a comunicação entre os objetos. Com o Diagrama de classes podemos mostrar o relacionamento entre os objetos e dar uma boa noção ao programador na hora de implementar o código.**

#### Tarefa

1. **Defina os relacionamentos entre os objetos.**
2. **Leia sobre os outros diagramas importantes da UML: o Diagrama de caso de uso e o Diagrama de comunicação.**

#### Tarefa 1

A solução deste trabalho se encontra nesta imagem:



Educador,um detalhe a ser levado em consideração é com relação as classes que implementam a interface Nameable. Como ela implica que a classe que a implementa deve ter um atributo name, então omitimos este atributo de todas estas classes.

#### Tarefa 2

## Aula 3 - Arquitetura de software

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para

* Diferenciar um arquiteto de software e um desenvolvedor
* Entender a arquitetura MVC

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 50 minutos de aula expositiva;
* 20 minutos para tirar as dúvidas dos alunos;
* 20 minutos para desenvolver as atividades propostas para a turma.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* O que é arquitetura de software;
* O que faz um arquiteto de software;
* MVC;
* Sistema de Agenda Eletrônica;
* Criando os Models.

### PONTOS IMPORTANTES

Para garantir ao aluno um aprendizado significativo, resgate os principais conceitos de cada tópico abordado na aula. Neste momento, é importante que você utilize dicas e sugestões para reforçar os temas trabalhados, oferecendo também exemplos que possibilitem a relação entre teoria e prática.

### 3.1 - O que é arquitetura de software

#### 3.1.1 Definição

Para definir melhor, no caso de arquitetura, uma definição concisa é: "Arquitetura é a definição dos elementos que compõem uma estrutura e como eles se relacionam". Quando entramos em algo mais específico como "Arquitetura de Software", o que muda é a definição de quem são os elementos utilizados, qual a estrutura e as possibilidades de relacionamento entre eles. E neste caso a estrutura é o próprio software e os elementos constituintes dele, que podem ser classes, componentes, serviços e etc.

A arquitetura é um projeto, mas nem todo projeto pode ser considerado arquitetura, pois muitas decisões de projeto são deixadas de lado na arquitetura. A arquitetura serve para deixar as restrições para as atividades futuras, que devem produzir códigos ou projetos mais detalhados conforme a arquitetura.

#### 3.1.2 Importância

É uma atividade no desenvolvimento de software que define a estrutura de um sistema de alto nível, sem ater-se a tecnologia utilizada.

### 3.2 - O que faz um arquiteto de software

Um arquiteto de software já foi um programador alguma vez na vida. Ele não programa, mas não é só por isso que ele está afastado ou defasado em relação a programação. Como ele define os requisitos, ele deve também ser o revisor dos artefatos (entregáveis que geram valor ao projeto) para validar a conformidade do que está sendo desenvolvido com o que vou modelado confuso ???. Isto implica que o arquiteto fará também a revisão do código que está sendo desenvolvido, verificando se as boas práticas estão sendo aplicadas corretamente e etc. Resumindo, o arquiteto não programa, mas está em contato constante com a programação.

### 3.3 - O MVC

O MVC é uma arquitetura de software que organiza as classes em três camadas, ou categorias: a Model, a View e a Controller.

Em Smalltalk tudo é objeto: os números, as classes, os métodos, blocos de código, etc. Não há tipos primitivos, ao contrário de outras linguagens orientadas a objeto; strings, números e caracteres são implementados como classes em Smalltalk, por isso esta linguagem é considerada puramente orientada a objetos. Para complemento você educador deve se aprimorar, para maiores informações passe no site: <http://squeak.org/> . Revise a Unidade 1.

### 3.4 - Sistema Agenda Eletrônica

Educador, a partir daqui as aulas serão totalmente prática. Os conceitos gerais de POO já foram aplicados e agora o que teremos de teórico é no máximo uma definição ou outra sobre Java ou ferramentas do Java, nada que diz respeito especificamente à programação orientada a objetos.

Começaremos a criar um programa que funcionará como agenda de contatos. Este programa utilizará recursos de interface gráfica e recomenda-se que você já o tenha pronto em seu computador. O exemplo será disponibilizado mas é aconselhável que você reproduza os passo-a-passos também, antes de dar esta aula aos alunos.

#### 3.4.1 Requisitos do sistema

Mais uma vez estamos definindo os requisitos do software. Este papel geralmente é do gerente de projetos, mas em muitas empresas que não possuem um funcionário deste cargo, esta atividade é feita pelo analista de sistemas ou programador.

Neste tópico foi feita uma breve descrição dos requisitos do nosso próximo exemplo prático. Este exemplo será desenvolvido até o final deste curso, e é o maior exemplo até então.

Recomendamos que você, educador, finalize este exemplo antes mesmo de começar esta aula (Aula 3 – Unidade 4), pode ocorrer problemas de execução de código durante a aula que poderiam atrapalhar o andamento da aula.

### 3.5 - Criando os Models

Neste tópico criaremos a camada Model da nossa agenda de contatos.

#### 3.5.1 Enumerações

Educador, os exemplos de código serão mostrados em forma de passo-a-passo, mas os alunos não terão o código pronto para consultar. Muitos bugs (problemas no código ou execução do programa) acontecerão com os alunos, pois é natural que isto aconteça com programadores estudantes, então é primordial que os alunos recebam acompanhamento, seja este individual ou pelo quadro negro, repassando os exemplos, explicando ao aluno de forma diferente, tentando ajuda-lo a pesquisar uma solução na internet.

Apenas em último caso corrija os bugs do aluno. Corrija apenas em casos que o aluno está atrasando o andamento da aula ou caso observe uma frustração muito grande vinda do mesmo. É primordial para a formação do profissional da programação que ele saiba identificar e resolver bugs. Além do aluno começar a ser mais produtivo cada vez que ele consegue resolver os bugs sozinho, ele ganha habilidades para manutenção em código de terceiros.

### 3.6 – Resumo

Retome os tópicos abordados na aula questionando aos seus alunos:

O que aprendemos hoje pessoal? Você pode fazer isso em forma de Quizz ou para engajar sua turma pode finalizar a aula, perguntando se alguém tem dúvidas sobre aula de hoje e quem se habilitaria a resumir o que foi visto, faça isso toda aula com um aluno diferente para treiná-los quanto ao desenvolvimento de comunicação e ao final da fala do aluno, resuma os tópicos.

Nesta aula definimos e vimos a importância da Arquitetura de Software. O MVC é um padrão de arquitetura de software que propõe a separação da aplicação em três camadas: model, view e controller. É uma maneira de criar classes cada vez mais independentes de outras e que sejam focadas apenas em suas responsabilidades. Nesta aula, criamos a camada Model sendo sua aplicação importante para a manipulação de dados de um sistema.

### 3.7 – Exercícios

|  |  |
| --- | --- |
| 1 | **Defina arquitetura de software e sua importância.**  Resposta: A arquitetura de software surgiu como uma vertente da engenharia de software. De forma geral, é um conjunto de divisões e relações do todo com suas partes. Definição A arquitetura é um projeto, mas nem todo projeto pode ser considerado arquitetura, pois muitas decisões de projetos são deixadas de lado na arquitetura. A arquitetura serve para deixar as restrições para as atividades futuras, que devem produzir códigos ou projetos mais detalhados conforme a arquitetura. Importância A arquitetura de software apresenta uma abstração comum de um sistema que a maioria dos grupos de interesse (*stakeholders*) pode utilizar como base de negociação, consenso, entendimento e comunicação.  Também pode ser usada para abstração transferível do sistema, pois ela constitui um modelo relativamente simples da estrutura e da relação entre os elementos de um sistema e esse modelo pode ser utilizado em outros sistemas que sejam similares. |
| 2 | **Quais são as competências de um arquiteto de software?**  Resposta: De acordo com o tipo de sistema, as funções do arquiteto podem ser:   * Estabelecimento de requisitos: onde o arquiteto é responsável pelo entendimento e gestão dos requisitos não funcionais do sistema. Os arquitetos podem trabalhar diretamente com os *stakeholders.* * Avaliação do risco técnico do sistema: o arquiteto fornece um plano de risco e avalia o impacto e os riscos que alguma mudança terá no sistema. * Análise do domínio do sistema: o arquiteto deve dividir os problemas em partes e estruturar soluções que possam abordar as necessidades da organização. * Também pode ser revisor dos entregáveis do sistema, desenvolvedor e mentor de projetistas e desenvolvedores, e líder de equipe. |
| 3 | **O que é MVC?**  Resposta: A reusabilidade e manutenibilidade são parâmetros de qualidade importantes para a orientação a objetos e devem ser sempre melhorados pelo desenvolvedor. Para isso devem ser criadas classes cada vez mais independentes de outras e que sejam focadas apenas em suas responsabilidades. O MVC vem ao encontro disso quando propõe a separação da aplicação em três tipos de classes: **M***odel,* **V***iew e* **C***ontroller.*  A estrutura MVC é um padrão de arquitetura de software que foi proposta em 1979 pelo grupo que trabalhava na linguagem *Smalltalk.* Este padrão é utilizado para que o usuário visualize o mesmo modelo de dados simultaneamente sob diferentes pontos de vista, propondo a separação da aplicação em: *model*, que é a camada de manipulação dos dados; *view*, a camada de interação com o usuário e *controller*, que é a camada de controle. |
| 4 | **Defina model, view e controller.**  Resposta: |
|  | **Model**: Camada de manipulação de dados.  **View**: Camada de interação com o usuário.  **Controller**: Camada de controle, intermediadora entre view e model. |
| 5 | **O que são enumerações?**  Resposta: Enumerações ou enums são estruturas de dados definidas pelo programador utilizadas para enumerar determinado elemento. Muitas vezes precisamos trabalhar com um conjunto de dados que é representado por números inteiros mas simbolizam estados ou valores significativos a aplicação. Com o enum é possível apelidar e agrupar em um contexto estes números inteiros simbólicos |
| 6 | **Quais são as vantagens de utilizar enumerações?**  Resposta: Com o enum é possível restringir um conjunto de valores e facilitar a legibilidade do programador, se comparado a solução utilizando Integers. O que o torna vantajoso em inúmeras aplicações. |

### 3.8 – TDP

#### 3.8.1 - A fase de design

**Enquanto os arquitetos identificam as entidades e esboçam os diagramas, o time de designer esboça os protótipos e os mockups.**

**Um protótipo é um produto em fase de testes, que é utilizado para validação e verificação conceitual de uma ideia. No caso da programação, protótipos são desenhos de baixo nível que roteirizam as telas e o fluxo de navegação do aplicativo.**

**Os protótipos não chegam aos olhos dos desenvolvedores, pois servem para definir a organização estrutural do aplicativo. Eles são feitos de maneira rápida e passam pela validação do cliente. Uma vez reprovado, o protótipo é descartado e um novo é feito. Uma vez aprovado, o protótipo passa por um refinamento e enfim os mockups são gerados.**

**Os mockups, assim como os protótipos, são representações conceituais do software, mas neste caso, são de alto nível, com uma fidelidade maior ao produto final. São imagens que mostram a aparência exata que o software deverá conter.**

**Mariana, a designer, está com dificuldades em ter um protótipo aprovado. Ajude-a a trilhar o caminho do sucesso!**

#### Tarefa

**Pegue folhas de sulfite e caneta grossa e tente desenhar as telas do aplicativo como elas vêm em sua mente. Não tenha medo de errar.**

**Faça de uma maneira rápida que apenas passe a ideia do que você deseja nesta tela. Sendo assim, faça diversos protótipos até que você encontre um ideal.**

Educador, como modelo interessante a ser seguido para a resolução deste TDP, observe o protótipo abaixo:

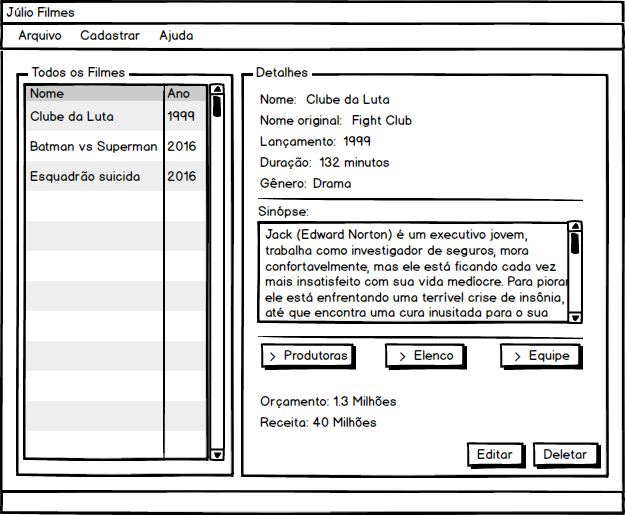


Figura 4. - Tela principal

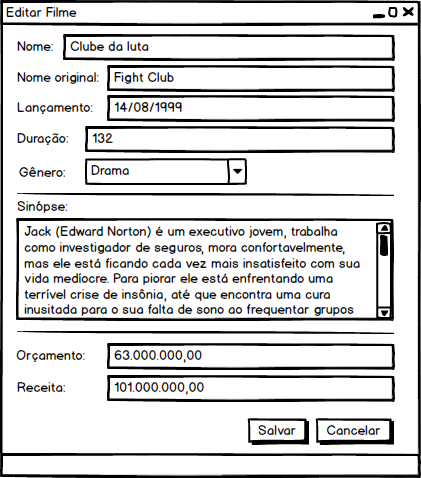


Figura 4. - Cadastro e Edição de Filmes

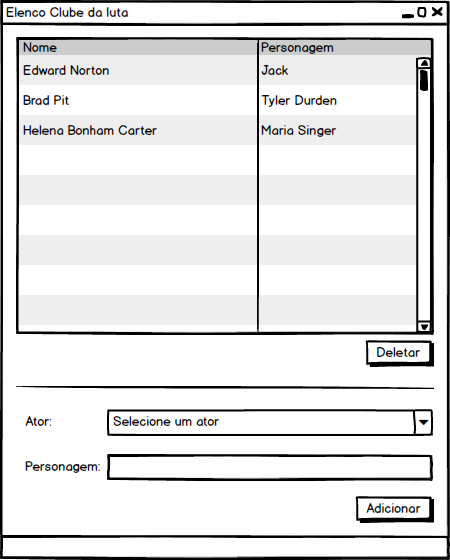


Figura 4. - Tela de elenco

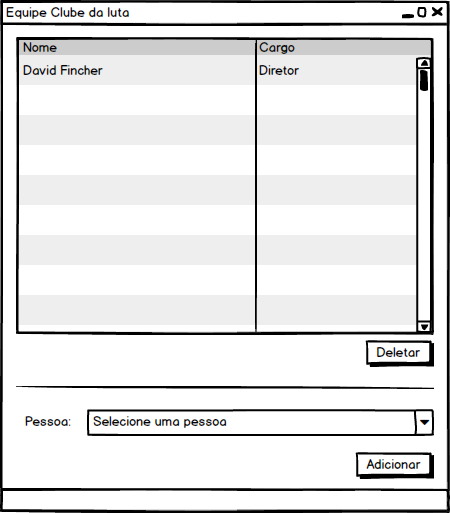


Figura 4. - Tela de equipe

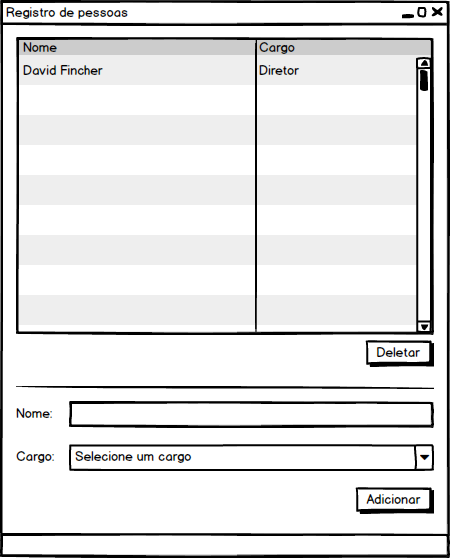


Figura 4. - Tela de registro de pessoas

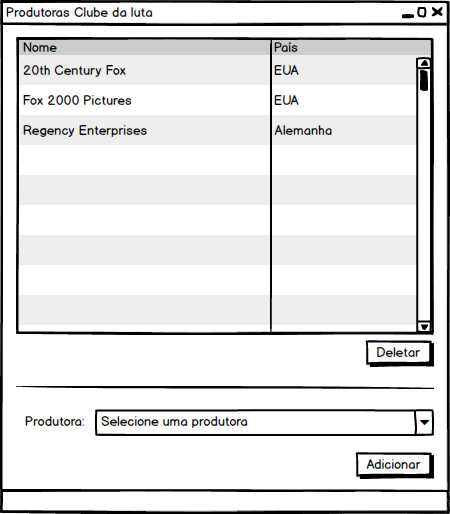


Figura 4. - Tela de produtoras do filme

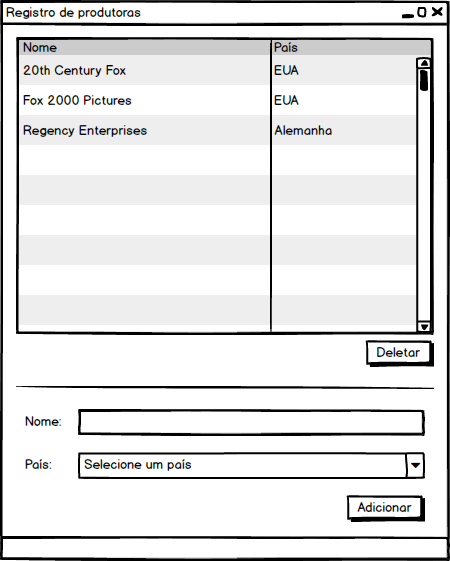


Figura 4. - Tela de registro de produtoras

**Educador,** o TDP é finalizado na Unidade 5. Sugerimos que você termine-o antes da aula 1 da Unidade 5, pois nesta aula os alunos começarão a desenvolver o projeto de fato, e é preciso que você tenha total domínio do mesmo. Outro detalhe é que a explicação e o intuito das telas mostradas neste protótipo estarão na aula 3 da unidade 5, juntamente com o passo-a-passo para reproduzi-las.