# Capítulo 4 - UML

Agora que aprendemos a programar em Java e os conceitos principais de POO, vamos mergulhar mais afundo, para a Análise de Projeto Orientado a Objetos. Digamos que 70% do tempo e esforço gasto em um software orientado a objetos está depositado na fase de análise e planejamento. Em comparação com a linguagem procedural, o tempo gasto com o planejamento é visto por muitos como uma desvantagem, mas no final das contas não é. Para garantir todas as vantagens de POO que conhecemos, a fase de análise é importante. É nela que modelamos nossas classes de forma que elas sejam coesas, fáceis de serem entendidas e reutilizáveis. A Análise de Projetos Orientados a Objetos é uma área da Ciência da Computação bastante ampla. Iremos apenas pincelar alguns conceitos como diagrama de classes, relacionamento entre objetos e arquitetura de software.

## Aula 1 - UML

### 1.1 - O que é UML

No mundo dos projetos é mais rápido desenhar e redesenhar do que fazer uma correção em um software que já está sendo usado pelo usuário final. A UML (*Unified Modeling Language*) ou Linguagem de Modelagem Unificada é, como o próprio nome diz, uma linguagem de modelagem que veio para representar sistemas de uma forma padrão.

Antes de começarmos a codificar, o projeto passa por uma série de planejamentos e modelagens. A UML é uma forma de representar graficamente um sistema de software.



Figura 4. 1 – Logo UML

(<http://www.diegomacedo.com.br/introducao-a-uml-e-seus-diagramas/>)

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.

Em sua versão 2.2, a UML possui 14 tipos de diagramas (que são representações gráficas), divididos em duas grandes categorias: Estruturais e Comportamentais.

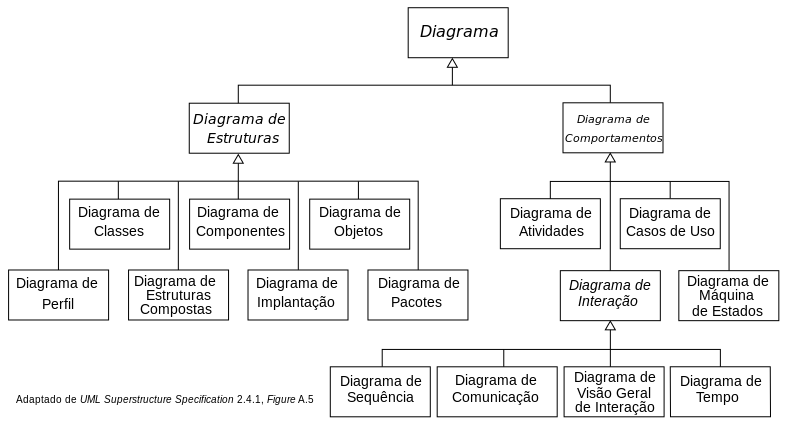


Figura 4. – Especificação estrutural da UML

(<https://pt.wikipedia.org/wiki/UML>)

### 1.2 - Diagrama de classes

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.

Na fase de análise, não é preciso se preocupar com código e implementação. Esqueça programação, pois o que importa no diagrama de classes é ter uma visão geral do sistema como um todo. Nesta fase não devemos levar em conta nem a linguagem de programação que o software será programado, pois a UML é uma linguagem de modelagem unificada que vale para qualquer linguagem de programação orientada a objetos.

Modelar o relacionamento entre as classes é importante pois nos possibilita detectar possíveis falhas de planejamento antes mesmo de ter um software rodando, e tendo detectado estas possíveis falhas fica mais rápido corrigi-las nesta fase de modelagem do que quando o software já está codificado.

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

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 pela implementação das classes na linguagem de programação escolhida.

### 1.3 - Classes, métodos e atributos

Vários dos conceitos que veremos agora já foram aprofundados nas aulas anteriores, o que iremos fazer é criar um paralelo para a criação de diagramas de classes.

As classes aqui, são representadas por retângulos, com três subdivisões. Como sabemos que classes são um conjunto de atributos agindo sobre seus métodos, elas serão representadas nos diagramas por estas mesmas propriedades.

Na primeira subdivisão está representada a **Descrição ou Nome da classe**. Na segunda estão os **atributos** (com seus tipos e a visibilidade) da classe descrita, e na última subdivisão está a lista dos **métodos** da classe, como no exemplo a seguir (classe Carro), onde vários atributos como marca, modelo e cor são representados logo após a descrição da classe e na subdivisão seguinte, são mostrados os métodos drive() e stop().



Figura 4. – Representação de uma classe no diagrama de classes

Podem-se notar na representação anterior, sinais positivos e negativos. Estes se referem à visibilidade dos atributos e métodos e qualificam seus modificadores de acesso, onde **“+”** representa o acesso public, **“-”** representa private e protected é representado pela ausência de sinais.

Nem sempre todos os atributos e métodos da classe são representados no diagrama, portanto não é necessário que as três subdivisões sejam preenchidas. É possível que haja um diagrama de classe apenas com nome e atributos ou então nome e métodos, ou somente o nome da classe (única subdivisão obrigatória para representar o diagrama de classes). A omissão de algumas subdivisões representa uma abstração no detalhamento da classe, ou seja, necessita-se de poucos detalhes na representação da classe e evita-se que tenhamos um diagrama poluído e de difícil compreensão. Outro motivo da omissão da sessão de atributos ou métodos é que a classe simplesmente não possua atributos ou métodos.

### 1.4 - Interfaces

Como vimos anteriormente, uma interface define um conjunto de operações que outras classes devam seguir. A partir de agora vamos chamar as classes que implementam uma interface de classe concreta da interface. Em diagrama de classes, uma interface é representada com uma etiqueta acima do nome da classe, da seguinte maneira:



Figura 4. – Representação de uma interface no diagrama de classes

No exemplo acima, dos animais domesticáveis, mostramos uma interface chamada Domesticável. Em uma explicação complementar, esta interface obriga que todos as classes que a implementam tenham um comportamento específico. No caso do diagrama acima, a interface define que um animal domesticável terá um nome (este nome pode ser dado através do método set e obtido através do get) e terá um dono. Estas são duas características que animais domesticáveis deverão seguir. Este é só um exemplo ilustrativo, um animal domesticável pode ter muitas outras características e seu diagrama pode ser desenhado de inúmeras maneiras.

Na próxima aula você ficará sabendo como representamos o ato de implementar uma interface (ou a relação entre a interface e suas classes concretas).

### 1.5 - Classes abstratas

Como as interfaces, a classe abstrata é identificada com uma etiqueta. Diferente das interfaces, as classes abstratas podem conter atributos, e não somente métodos, então ela é representada da seguinte maneira:



Figura 4. – Representação de uma classe abstrata no diagrama de classes

No exemplo Funcionário definimos que a classe Employee deveria ser abstrata pois deveríamos separar os funcionários por cargos, e a classe em questão não consegue definir um cargo por si só. Utilizamos herança e polimorfismo para isto. No diagrama de classes, uma classe abstrata é representada pela etiqueta <<abstract>> e diferente de uma interface, a classe abstrata pode conter métodos e atributos.

### 1.6 – Resumo

Definitivamente é mais fácil entender um desenho do que um punhado de código. A UML vem para facilitar a vida dos programadores que caem de paraquedas no meio de um projeto. Com ela temos uma representação objetiva do projeto como um todo, sem ater-se a detalhes como lógica de programação. Neste capítulo você conheceu a UML e seu vasto mundo de diagramas. Um deles foi o diagrama de classes. Vimos como algumas entidades da programação orientada a objetos são representadas no diagrama de classes.

### 1.7 – Exercícios

|  |  |
| --- | --- |
| 1 | O que é UML? |
| 2. | O que é Diagrama de classes? |
| 3. | Em qual fase do projeto o Diagrama de classes deve ser desenhado? |
| 4. | Quais são as perspectivas de construção do Diagrama de classes? Explique-as. |
| 5. | Como classes, métodos e atributos são representados no Diagrama de classes? Explique com um exemplo. |
| 6. | É possível que métodos e atributos ou até mesmo subdivisões sejam omitidos no diagrama? Caso a resposta for positiva, o que isto representa? |
| 7. | Como uma interface é representada no Diagrama de classes? |
| 8. | Como as classes abstratas são representadas em um Diagrama de classes? |

### 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

Junte-se em dupla. 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.

## Aula 2 - Relacionamento entre objetos

### 2.1 - Relacionamentos

As classes relacionam-se entre si com o intuito de compartilhar informações que colaborem umas com as outras para permitir as execuções dos processos pelo sistema. Na verdade, o diagrama mostra as relações entre os objetos e outras informações das classes, e não entre as classes em si. Você entenderá mais a seguir.

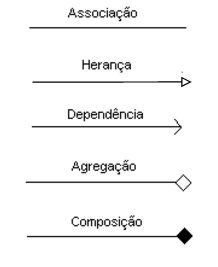


Figura 4. 6 – Representação dos relacionamentos no diagrama de classes

### 2.2 - Herança

Na aula anterior aprendemos como representar uma classe, uma interface e uma classe abstrata com diagrama de classes. Mas como é representada a relação entre superclasse e subclasse? É mais simples do que você imagina. No capítulo anterior utilizamos uma estrutura de árvore de herança para representar o relacionamento entre as classes que a compõem. Pois bem, a representação de herança é exatamente aquela que utilizamos. A seta utilizada para representar uma relação de herança é a seguinte:



Figura 4. 7 – Representação de relacionamento de herança no diagrama de classes

A direção que a seta aponta é sempre para a classe generalizada (superclasse) e a origem vem da classe especializada (subclasse). Para que seja mais intuitivo, sempre que possível posicione a classe mãe acima das classes filhas, similar a uma estrutura de árvore genealógica. Veja como seria o exemplo dos Animais com uma modelagem mais detalhada:



Figura 4. – Representação de uma herança no diagrama de classes

No diagrama acima especificamos um pequeno exemplo de herança entre Felino e suas subclasses Gato e Leão. Para nível de entendimento, definimos que um felino tem *cor, expectativa de vida, peso, dias em gestação, família* e que ele *caça e emite sinal sonoro.*

Sua subclasse Leão terá uma *espécie, idade, habitat* e como ações ele *reina e dorme.* Já o gato tem *raça, vidas, nome* e como ações ele *ronrona e ignora o dono.* As subclasses também possuem as características de Felino, pois são especializações da mesma

### 2.3 - Interfaces

Para definirmos uma relação de interface precisamos de uma seta parecida com a de herança, com a diferença que esta é tracejada.



Figura 4. - Representação do relacionamento de interface no diagrama de classes

O elemento de origem da seta é sempre a classe concreta, ou seja, a classe que a implementa, e o elemento o qual ela aponta é a interface em si.



Figura 4. – Exemplo do reino animal utilizando diagrama de classes

O exemplo acima mostra como é feito o relacionamento entre uma interface e uma classe concreta. Traduzindo podemos ver que Gato e Papagaio são domesticáveis.

### 2.4 - Classes abstratas

A representação utilizada para definir o relacionamento entre a classe abstrata e sua classe concreta é o mesmo utilizado pela herança, já que a maneira de utilizar uma classe abstrata é especializando-a em uma subclasse. O diferencial está apenas na etiqueta que acompanha o nome da classe. Modelando o exemplo de Funcionários, teremos o seguinte diagrama:

abstração.png

Figura 4. – Exeplo de relacionamento de herança com classe abstrata em UML

Como as subclasses de Funcionário não possuem nenhum método ou atributo que diferencie as especializações, vemos as seções de atributos e métodos vazias, isto porque a herança feita neste exemplo tem como único propósito calcular o bônus que cada funcionário receberá de acordo com seu cargo.

### 2.5 - Associação

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.

Em casos mais comuns, há a associação entre duas classes, chamada de associação binária. O caso em que a classe esta vinculada a si mesma chamamos de associação unária e quando uma mesma associação é compartilhada por várias classes, chamamos de associação ternária ou N-ária, em que esta ultima é um tipo mais raro e complexo. Resumindo, o objetivo das associações é definir a maneira como as classes se interligam e interagem entre si, estabelecendo um fluxo de informações entre elas.

A representação das associações por linhas e setas indicam a navegabilidade das associações, que indica o fluxo em que as informações são transmitidas entre as classes associadas. O uso das setas não é obrigatório mas traz um modo de leitura mais fluído e organizado.

As associações podem conter títulos, úteis quando é necessário esclarecer qual é o tipo de vínculo entre as classes.

#### 2.5.1 - Binária

Associações binárias são vínculos entre duas classes e é a mais comum no diagrama de classes. Nesta associação, um objeto pode se relacionar ou não com instâncias de outra classe, dependendo de sua multiplicidade

**Multiplicidade**

A multiplicidade procura determinar 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. A seguir, uma tabela explicativa para cada tipo de multiplicidade.

|  |  |
| --- | --- |
| **Multiplicidade** | **Significado** |
| 0..1 | No mínimo zero (nenhum) e no máximo um.  Os objetos das classes associadas não precisam obrigatoriamente estar relacionados, mas se estiverem indica que apenas uma instância da classe se relaciona com as instâncias da outra classe. |
| 1..1 | Um e somente um. Indica que apenas um objeto da classe se relaciona com os objetos da outra classe. |
| 0..\* | No mínimo nenhum e no máximo muitos. Indica que pode ou não haver instâncias da classe participando do relacionamento. |
| \* | Muitos. Indica que muitos objetos da  classe estão envolvidos no relacionamento. |
| 1..\* | No mínimo um e no máximo muitos. Indica que há pelo menos um objeto envolvido no relacionamento, podendo haver muito envolvidos. |
| 3..5 | No mínimo três e no máximo cinco. Indica que existem pelo menos três instâncias envolvidas no relacionamento e que podem ser quatro ou cinco as instâncias envolvidas, mas não mais do que isso. |

Tabela 4. - Multiplicidades

#### 2.5.2 - Ternária ou N-árias

Conectam mais de duas classes e é representada por um losango para onde convergem todas as ligações.

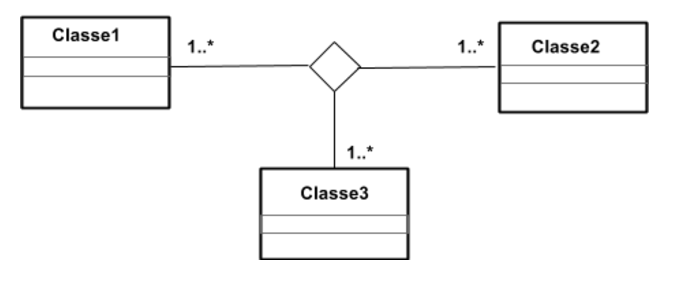


Figura 4. – Relacionamento ternário no diagrama de classes

Estas associações indicam associações mais complexas, no entanto devem ser evitadas para que sua leitura não seja difícil de interpretar.

#### 2.5.3 - Unária

Ocorre quando existe o relacionamento de uma classe consigo mesma. Veja a Figura:

### ../../../Desktop/cap4/poo_cap4_img13.png

Figura 4. – Relacionamento unário no diagrama de classes

### 2.6 - Composição

No caso da composição é diferente. 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.

../../../Desktop/cap4/poo_cap4_img14.png

Figura 4. – Representação do relacionamento de composição no diagrama de classes

Exemplo:

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

Figura 4. – Exemplo de relacionamento de composição no diagrama de classes

### 2.7 – Agregação

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. Por exemplo, um time é formado por atletas, ou seja, os atletas são parte integrante de um time, mas os atletas existem independentemente de um time existir. Nesse caso, chamamos esse relacionamento de agregação.

../../../Desktop/cap4/poo_cap4_img15.png

Figura 4. - Representação do relacionamento de composição no diagrama de classes

Exemplo:

../../../Desktop/cap4/poo_cap4_img17.png

Figura 4. 17 - Exemplo de relacionamento de agregação no diagrama de classes

### 2.8 – Resumo

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? |
| 2. | Como definimos uma relação de interface no Diagrama de classes? |
| 3. | Qual é a representação utilizada para definir o relacionamento entre a classe abstrata e sua classe concreta? |
| 4. | O que é uma associação e qual o seu objetivo? |
| 5. | Quais os tipos de associação? |
| 6. | O que são associações binárias? E Ternária ou N-árias? E Unária? |
| 7. | O que determina a multiplicidade? |
| 8. | Dê um exemplo de multiplicidade ternária. |
| 9. | Qual a diferença entre composição e agregação? |
| 10. | Por que o relacionamento é entre objetos e não classes? |

### 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 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.

## Aula 3 - Arquitetura de software

### 3.1 - O que é arquitetura de software

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.

#### 3.1.1 - 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 restrições nas atividades futuras, que devem produzir códigos ou projetos mais detalhados conforme a arquitetura.

#### 3.1.2 - 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.

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

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.

### 3.3 - O MVC

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.

### 3.4 - Sistema Agenda Eletrônica

Para entendermos melhor a arquitetura MVC, vamos criar nosso último exemplo. O sistema de cadastro de contatos. Este sistema será completo e terá interface gráfica, que iremos criar no próximo capítulo. A partir de agora, todas as etapas e conhecimentos a seguir serão em função da agenda eletrônica, afinal vamos aprender na prática tudo isso!

#### 3.4.1 - Requisitos do sistema

Vamos começar pelos requisitos.

O sistema Agenda permitirá ao usuário cadastrar, visualizar, editar e deletar um contato na agenda eletrônica. O sistema deverá atender os requisitos para as seguintes funcionalidades:

* **Cadastrar contato:** O usuário deverá informar o *nome*, o *email*, o *endereço* e um *grupo* para o *contato*. O contato poderá participar de um dos seguintes grupos: *escola*, *trabalho*, *família*, *amigos* e *conhecidos*.
* **Editar contato:** O usuário poderá selecionar e *editar* todos os dados de um contato.
* **Deletar contato:** O usuário poderá selecionar e *deletar* um contato.
* **Visualizar contato:** O usuário poderá selecionar e *visualizar* os detalhes de um contato.

A fase de definição de requisitos é bem maior e detalhada. Como este não é o foco do curso, definimos alguns requisitos de forma mínima suficiente para conseguirmos prosseguir com o desenvolvimento do sistema Agenda.

### 3.5 - Criando nossos Models

Agora que você já tem os requisitos vamos criar os models da agenda.

No decorrer do curso demos destaque total à camada Model. Na prática a camada Model é a representação de objetos ou entidades do mundo real ou do contexto do sistema, ou seja, nos nossos exemplos estivemos sempre manipulando os dados ou representando-os. O exemplo do reino animal, dos carros e dos funcionários apresentam o uso da camada Model. Agora vamos identificar os candidatos a modelo.

* O usuário irá criar um **contato**. Contato será uma classe de Model.
* O contato participará de ***grupos***. Os grupos já foram pré-determinados acima. Para este exemplo temos apenas um componente na camada Model.

Agora que definimos a entidade que irá compor a camada Model, vamos modelá-la utilizando o diagrama de classes:

1. Defina uma enumeração para os grupos.
2. Identifique e defina os Atributos de Contato.
3. Identifique e defina os métodos de Contato (lembrando que no diagrama de classes não explicitamos métodos de encapsulamento).

Pronto. Até então estamos realizando o papel de arquiteto de software para o Sistema Agenda Eletrônica. Coletamos os requisitos e agora estamos modularizando-os. Você reparou que nesta etapa conseguimos focar apenas nos Models? Esta é uma das vantagens da POO, a modularização, que é desenvolver partes independentes do software, que posteriormente irão se relacionar. Organizando o código para o padrão MVC, simplesmente conseguimos planejar a camada. Model sem se preocupar com o restante do software. Continuemos então com as definições da nossa camada Model.

#### 3.5.1 - Enumerações

Temos um detalhe importante na agenda, as enumerações. Vamos entender um pouco mais!

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.

No exemplo temos uma sequência de valores predefinidos que representam os grupos de contatos. Teríamos que grupo é do tipo Integer e corresponde aos seguintes valores

* escola = 0
* trabalho = 1
* família = 2
* amigos = 3
* conhecidos = 4

Desta forma organizamos os possíveis grupos para corresponder a um número inteiro. Assim, por exemplo, quando definíssemos um valor ao atributo *grupo* do usuário faríamos da seguinte maneira:

contact.group = 1   // Diz que o contato está no grupo Trabalho

Como boa prática, antes de começar a programar, vamos diagramar. Veja o diagrama para o exemplo até então:

../../../Desktop/cap4/poo_cap4_img18.png

Figura 4. – Classe Contact

Podemos perceber que tratar com números é mais difícil de interpretar. Dizemos que é menos semântico, e nada orientado a objetos. A medida que o aplicativo crescer, surgirão novos grupos para que o usuário possa classificar seus contatos, e isso será cada vez mais difícil de ser gerenciado pelo programador. Por isso existe o enum.

Com uma estrutura enum criada podemos substituir o atributo tipo de *grupo* para o enum Grupo, ao invés de Integer. O enum se parece com um tipo de variável, podendo ser assimilado como tal, mas não é uma classe, é apenas uma estrutura de dados. O diagrama de classes de Contato atualizado com enum ficaria da seguinte maneira.



Figura 4. – Relação Contact e Group

Perceba que o atributo *grupo* foi retirado da classe, isso porque o relacionamento entre Contato e Grupo implica que Contato pertence a um grupo.

DICA: Na hora de traduzir o diagrama para código não podemos esquecer de analisar os relacionamentos!

É comum esquecermos de levar em conta os relacionamentos, pois na prática, um relacionamento geralmente resulta em um atributo na classe. Neste exemplo, o relacionamento resultará em um atributo group do tipo Group na classe Contact.

Antes de definirmos como pronto, é boa prática que exista um atributo que sirva como identificador único ao objeto. Adicionaremos o atributo id ao nosso modelo Contact, portanto ele pode ser omitido do diagrama de classes, por se tratar de um atributo que se torna repetitivo entre os diversos models do projeto.

Nossa camada Model está pronta! View é o próximo passo, já que o Controller seria como o intermediador entre o Model e a View, por isso o deixaremos por último. Na verdade ele está bastante relacionado com o funcionamento da View, por isso abordaremos juntos a View e o Controller no exemplo.

### 3.6 – Resumo

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. |
| 2 | Quais são as competências de um arquiteto de software? |
| 3 | O que é MVC? |
| 4 | Defina model, view e controller. |
| 5 | O que são enumerações? |
| 6 | Quais são as vantagens de utiliar enumerações? |

### 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 idéia. 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 por 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.