

Aula 3: Projeto de Sistemas Orientando a Objetos

Passando da Análise ao Projeto

Passando da Análise ao Projeto: O processo de passar da análise para o projeto utilizado neste curso será:

1. Identificação dos requisitos dos usuários em forma de diagramas de Caso de Uso, Especificações de Caso de Uso e Especificações Suplementares;
2. Desenvolvimento do Modelo Conceitual (Análise) através do Diagrama de Classes contendo as Classes, seus atributos e relacionamentos (sem incluir, neste momento, os métodos das classes);
3. Desenvolvimento dos Diagramas de Sequência para identificação da primeira solução e localização dos possíveis métodos das classes;
4. Desenvolvimento do Modelo de Domínio (Projeto) através do Diagrama de Classes, desta vez contendo as demais classes identificadas e seus métodos.
5. Desenvolvimento dos demais diagramas UML.

Entendendo o Projeto Orientado a Objetos

Enquanto que, durante a Análise, há uma ênfase em descobrir objetos e conceitos do domínio do problema.

No Projeto há a ênfase em definir objetos de software e como eles colaboram para atender requisitos. Isto inclui, portanto, classes com seus atributos e métodos.

Por exemplo, uma classe Veículo seria implementada em Java como abaixo:

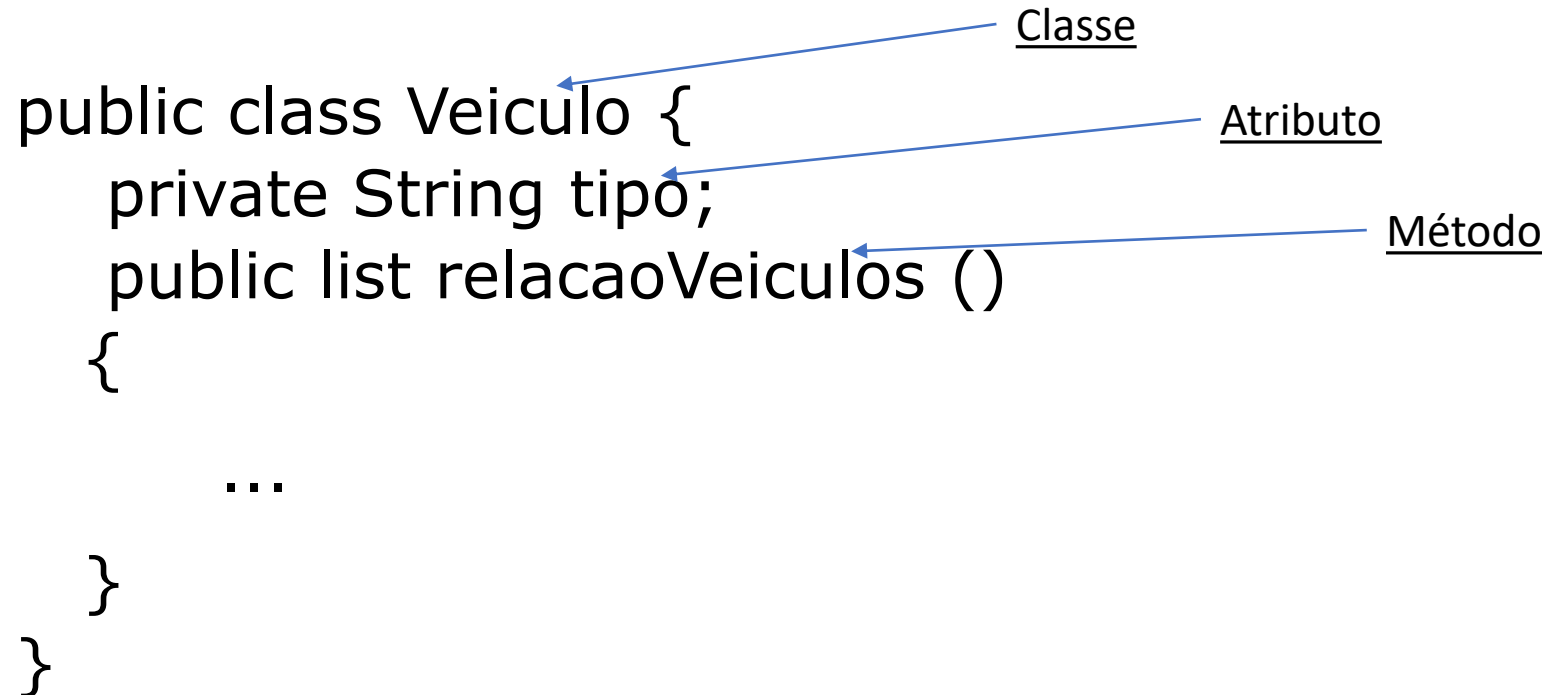
```
public class Veiculo {  
    private String tipo;  
    public List relacaoVeiculos ()  
    {  
        ...  
    }  
}
```

Entendendo o Projeto Orientado a Objetos

Enquanto que, durante a Análise, há uma ênfase em descobrir objetos e conceitos do domínio do problema.

No Projeto há a ênfase em definir objetos de software e como eles colaboram para atender requisitos. Isto inclui, portanto, classes com seus atributos e métodos.

Por exemplo, uma classe Veículo seria implementada em Java como abaixo:



```
public class Veiculo {  
    private String tipo;  
    public list relacaoVeiculos ()  
    {  
        ...  
    }  
}
```

The diagram illustrates the components of a Java class definition for `Veiculo`. Three blue arrows point from labels to specific parts of the code:
1. An arrow from the label Classe points to the `public class Veiculo {` line.
2. An arrow from the label Atributo points to the `private String tipo;` line.
3. An arrow from the label Método points to the `public list relacaoVeiculos ()` line.

O projeto orientado a objeto tem as seguintes características:

- Pensamos em coisas (objetos) em vez de funções;
- As funcionalidades são fornecidas em termos de serviços (métodos ou operações) oferecidos pelos objetos;
- Objetos são abstrações do mundo real;
- Objetos são independentes e encapsulam representações de informação e estado;
- Objetos se comunicam através de mensagens.

O projeto orientado a objeto tem as seguintes características:

- Pensamos em coisas (objetos) em vez de funções;
- As funcionalidades são fornecidas em termos de serviços (métodos ou operações) oferecidos pelos objetos;
- Objetos são abstrações do mundo real;
- Objetos são independentes e encapsulam representações de informação e estado;
- Objetos se comunicam através de mensagens.

Os objetos em um projeto OO estão relacionados à solução do problema que está sendo resolvido.

A programação OO realiza um projeto de software em uma linguagem de programação OO (Java ou C#).

Para passar de um modelo de análise para um modelo de projeto:

- Adicionamos detalhes às classes existentes (geralmente os métodos ou operações identificados nos diagramas de seqüência e comunicação (antigo diagrama de colaboração));
- Criamos novas classes para fornecer funcionalidade adicional.

Abaixo um exemplo de uma classe (Funcionário) contendo atributos e métodos:



Os objetos se comunicam através da passagem de mensagens. As mensagens são implementadas na prática como chamadas de métodos ou operações.

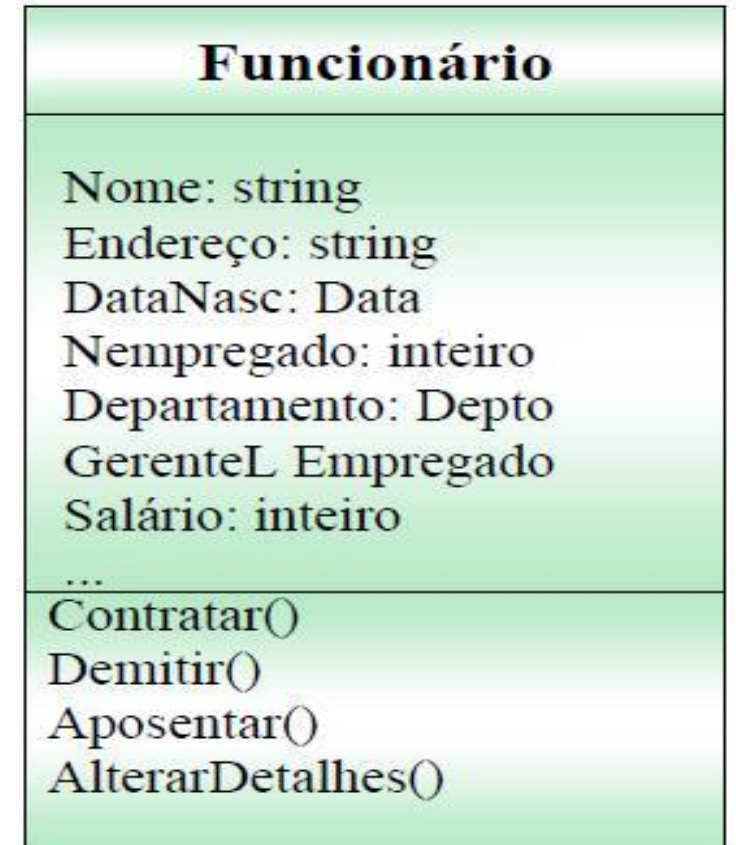
Por exemplo:

```
Funcionario f1 = new Funcionario();
```

```
f1.Contratar();
```

Durante o projeto fazemos as seguintes ações:

- Projetamos a arquitetura do sistema;
- Desenvolvemos os diagramas de comunicação;
- Refinamos os diagramas de sequência produzidos na análise;
- Especificamos as interfaces dos objetos.



Uma vez definidas as interações entre o sistema projetado e o ambiente, utilizamos estas informações para estabelecer a arquitetura do sistema.

É recomendável utilizarmos a arquitetura em camadas (n-tier).

- A camada de visão (interface gráfica) faz as interações com os atores;
- A camada de integração de dados gerencia os dados que serão persistidos;
- A camada de regras de negócio.

Uma arquitetura de software muito utilizada é a MVC.

Embora MVC não signifique exatamente desenvolvimento em camadas, os dois são muito utilizados.

MVC será discutido posteriormente.

Introdução à Arquitetura de Software

A adoção de uma arquitetura correta para o software ajuda a gerenciar a complexidade do mesmo.

A arquitetura do software define sua estrutura e inclui seus componentes e relacionamento entre eles.

Arquitetura enfatiza a separação de interesses em funcionalidade e interação.

Vantagens da Arquitetura de Software:

- Facilita o reuso (por exemplo, estilos (CSS) e padrões de projeto (MVC, Factory, etc.));
- Facilita a evolução do software;
- Facilita a manutenção do mesmo;
- Facilita a comunicação entre as partes envolvidas;

Estilos Arquiteturais definem famílias arquiteturais e não apenas um sistema. Exemplos:

- Cliente-Servidor
- Camadas (layered)

A arquitetura serve como base para as atividades de análise, projeto e implementação de software.

A definição de arquitetura pela ISO/IEEE 1471-2000 é a seguinte:

"Arquitetura é a organização fundamental de um sistema incorporada em seus componentes, seus relacionamentos com o ambiente, e os princípios que conduzem seu design e evolução."

A arquitetura, portanto, compreende estrutura (elementos ou componentes), relações e decisões, adicionada a uma importante preocupação: a evolução do software.

A arquitetura de um sistema deve definir os elementos que formarão o software. Tais elementos definem como o software é particionado em pedaços menores e, assim, definem como o software é entendido.

Elementos arquiteturais são divididos em dois tipos:

- elementos estáticos e
- elementos dinâmicos.

Os elementos estáticos de um sistema de software definem as partes do sistema e qual sua organização. Esse tipo de elemento reflete o sistema durante o design e é constituído de elementos de software (e.g., módulos, classes, pacotes, procedimentos, ou ainda serviços autocontidos), elementos de dados (e.g., entidades e tabelas de bancos de dados, arquivos de dados, ou classes de dados), e elementos de hardware (e.g., computadores em que o sistema vai executar, ou outros tipos de hardware que o sistema usará: roteadores, cabos, ou impressoras).

Elementos estáticos não consistem apenas das partes estáticas do sistema, mas também como eles se relacionam entre si. Associações, composições, e outros tipos de relações entre elementos de software, de dados, e de hardware formam o aspecto estático que compõe a arquitetura do sistema.

Por outro lado, elementos dinâmicos definem o comportamento do sistema. Esse tipo de elemento reflete o sistema durante a execução e nele estão incluídos processos, módulos, protocolos, ou classes que realizam comportamento. Elementos dinâmicos também descrevem como o sistema reage a estímulos internos e externos.

Uma arquitetura não deve ter suas estruturas definidas aleatoriamente, uma vez que são elas que permitem o sucesso relativo aos objetivos do sistema. Dessa maneira, é trabalho do arquiteto definir essas estruturas em meio às alternativas de design arquitetural existentes.

O arquiteto deve decidir entre as alternativas, particionando o sistema em elementos e relações que possibilitarão o atendimento aos atributos de qualidade. Essas decisões são chamadas decisões arquiteturais.