Linguagens de programação que utilizam orientação objetos

Alguns exemplos de linguagens orientadas a objetos modernas utilizadas por grandes empresas em todo o mundo são: Java, C#, C++, Object Pascal (Delphi), Ruby, Python, Lisp, etc.

**HISTÓRIA DO UML**

Era uma vez um sistema UML que tinha origem na compilação das "melhores práticas de engenharia" que provaram ter sucesso na modelagem de sistemas grandes e complexos. Sucedeu aos conceitos de Grady Booch, OMT (James Rumbaugh) e OOSE (Ivar Jacobson) fundindo-os numa única linguagem de modelagem comum e largamente utilizada. A UML pretende ser a linguagem de modelagem padrão para modelar sistemas concorrentes e distribuídos.

A UML ainda não é um padrão da indústria, mas esse objetivo está a tomar forma sob os auspícios do Object Management Group (OMG). O OMG pediu informação acerca de metodologias orientadas a objetos que pudessem criar uma linguagem rigorosa de modelagem de software. Muitos líderes da indústria responderam na esperança de ajudar a criar o padrão.

Os esforços para a criação da UML tiveram início em outubro de 1994, quando Rumbaugh se juntou a Booch na Rational. Com o objetivo de unificar os métodos Booch e OMT, decorrido um ano de trabalho, foi lançado, em outubro de 1995, o esboço da versão 0.8 do Unified Process - Processo Unificado (como era conhecido). Nesta mesma época, Jacobson se associou à Rational e o escopo do projeto da UML foi expandido para incorporar o método OOSE. Nasceu então, em junho de 1996, a versão 0.9 da UML.

Finalmente em 2000, a UML foi aprovada como padrão pelo OMG (Object Management Group), um consórcio internacional de empresas que define e ratifica padrões na área de Orientação a Objetos.

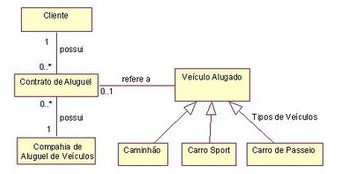
**DIAGRAMAS**

**Diagramas estruturais**

Os diagramas estruturais tratam o aspecto estrutural tanto do ponto de vista do sistema quanto das classes. Existem para visualizar, especificar, construir e documentar os aspectos estáticos de um sistema, ou seja, a representação de seu esqueleto e estruturas “relativamente estáveis”. Os aspectos estáticos de um sistema de software abrangem a existência e a colocação de itens como classes, interfaces, colaborações, componentes. E é representado em 7 tipos, segue abaixo as descrições:

**Diagramas de classes**

O Diagrama de Classes é utilizado para fazer a representação de estruturas de classes de negócio, interfaces e outros sistemas e classes de controle. Além disso, o diagrama de classes é considerado o mais importante para a UML, pois serve de apoio para a maioria dos demais diagramas. Se dá pela formação de conjunto de informações sobre determinadas classes, que unidas entre si formam um sentido geral do projeto.

Exemplos:

**Diagramas de objetos**

O diagrama de objetos representa os objetos de um diagrama de classes em um determinado instante de tempo, representando suas instâncias e seus relacionamentos, conforme definidos no diagrama de classes. Os objetos e suas instâncias demonstradas são utilizados para fazer a modelagem da visão estática do projeto de um sistema, a partir de situações da realidade ou de protótipos. Quando mostrado graficamente, é perceptível que o diagrama de objetos seja parecido com o de classes.

Exemplo: Uma classe carro, e o modelo Nissan 2007. O Nissan 2007 é objeto da classe carro.

**Diagramas de componentes**

Este diagrama mostra os artefatos de que os componentes são feitos, como arquivos de código fonte, bibliotecas de programação ou tabelas de bancos de dados.

Modelar software baseado em componentes.

Indicar os componentes do software e seus relacionamentos.

**Diagramas de implementação ou instalação**

O diagrama de utilização, também denominado diagrama de implantação, consiste na organização do conjunto de elementos de um sistema para a sua execução.

Mostra o layout físico de um sistema, revelando quais partes do software são executadas em quais partes do hardware.

Enfoca a estrutura física sobre a qual o software irá ser implantado e executado em termos de hardware.

Define como as máquinas estarão conectadas e através de quais protocolos se comunicarão.

É útil quando o sistema a ser modelado for ser executado sobre múltiplas camadas.

Seus elementos são os nós e os caminhos de comunicação.

**Diagramas de pacotes**

Um diagrama de pacotes é composto de:

Pacotes

Relacionamentos entre pacotes.

O diagrama de pacotes tem o objetivo de transformar as classes em pacotes. O critério para definir os pacotes é subjetivo e depende da visão e das necessidades do projetista. Este deve definir uma certa semântica e colocar os elementos similares e que tendem a serem modificados em conjunto num mesmo pacote. Como também, pode-se usar os pacotes para mostrar a arquitetura do sistema.

**Diagramas de estrutura composta**

O Diagrama de Estrutura Composta é utilizado para modelar Colaborações. Uma colaboração descreve uma visão de um conjunto de entidades cooperativas interpretadas por instâncias que cooperam entre si para executar uma função específica. O termo estrutura desse diagrama refere-se a uma composição de elementos interconectados, representando instâncias de tempo de execução colaboram, por meio de vínculos de comunicação, para atingir algum objetivo comum. Esse diagrama também pode ser utilizado para definir a estrutura interna de um classificador.

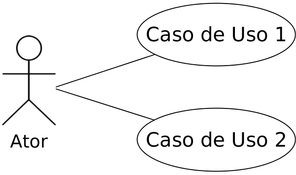
**Diagrama de perfil**

O Diagrama de perfil, definido a partir da UML 2.5, destina-se a criar uma visão (ou customização de um metamodelo existente com construções específicas para um determinado domínio) do relacionamento entre classes para atender determinado domínio.

**Diagrama comportamental**

Utilizado para visualizar, especificar, construir e documentar aspectos dinâmicos de um devido sistema. Considerando aspectos dinâmicos de um sistema como representação das suas partes que passam por alteração, assim como aspectos dinâmicos de uma casa abrangem a passagem de pessoas pelos cômodos, e a circulação de ar, também os aspectos dinâmicos de um sistema de software envolve itens como fluxo de mensagem ao longo do tempo.

UML também permite a representação de um sistema de forma padronizada, para facilitar a compreensão, e ainda auxilia a comunicação entre os objetos, e a identificação de processos.



3. Cada conceito é uma idéia ou um entendimento pessoal que temos do nosso mundo. Os conceitos que adquirimos nos permitem dar sentido sobre as coisas do nosso mundo. Essas coisas às quais nossos conceitos se aplicam são denominados objetos.

Objeto

* Um objeto pode ser real ou abstrato.
* Os objetos possuem informações (contém dados) e desempenham ações (possuem funcionalidade).
* Qualquer coisa à qual um conceito ou tipo de objeto se aplica – uma instância de um conceito ou tipo de objeto.
* Um objeto é uma instância de uma classe.

Exemplo:

* + Uma fatura;
  + Uma organização;
  + Um vôo de avião;
  + Uma pessoa;
  + Um lugar.

Na análise e no projeto OO, estamos interessados no comportamento do objeto. As operações são codificadas como métodos. A representação de software OO do objeto é, dessa forma, uma coleção de tipos de dados e métodos. No software OO: Um objeto é qualquer coisa, real ou abstrata, a respeito da qual armazenamos dados e os métodos que os manipulam.

Métodos:

* Os métodos especificam a maneira pela qual os dados de um objeto são manipulados.
* Uma especificação dos passos pelos quais uma operação deve ser executada.
* Ele é um script de implementação de uma operação.
* Diferentes métodos podem ser usados para executar a mesma operação.
* Os métodos de um tipo de objeto referenciam somente as estruturas de dados desse tipo de objeto.
* A ação que um objeto ou uma classe podem desempenhar.
* Os métodos são similares às funções e procedures do universo da programação estruturada.

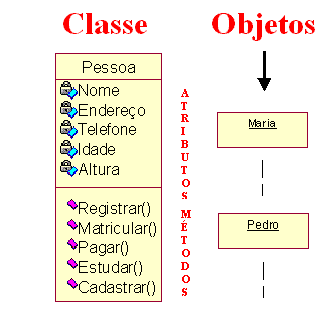
Um objeto é, dessa forma, uma coisa, com suas propriedades representadas pelos tipos de dados e seu comportamento representado pelos métodos.

Classe:

O termo classe refere-se à implementação de software de um tipo de objeto. Um tipo de objeto especifica uma família de objetos sem estipular como o tipo e o objeto são implementados. Os tipos de objetos são especificados durante a análise OO. Os tipos de objetos são implementados como módulos enquanto na linguagem orientada a objeto, os tipos de objetos são classificados como classes.

* Uma classe é uma implementação de um tipo de objeto.
* Uma classe especifica uma estrutura de dados e os métodos operacionais permissíveis que se aplicam a cada um de seus objetos.
* Uma classe pode ter sua própria estrutura de dados e métodos, bem como herda - lá de sua superclasse.

Exemplo: Classe- Objeto - Método - Atributo.



Herança:

É comum haver similaridades entre diferentes classes. Frequentemente, duas ou mais classes irão compartilhar os mesmos atributos e/ou métodos. Como nenhum de nós deseja reescrever várias vezes o mesmo código, seria interessante se algum mecanismo pudesse tirar proveito dessas similaridades. A herança é esse mecanismo. Por intermédio da herança, é possível modelar relacionamentos do tipo **"é"** ou **"é semelhante"**, o que nos permite reutilizar rotinas e dados já existentes.

Uma subclasse herda as propriedades de sua classe-mãe; uma subclasse herda as propriedades das subclasses e assim por diante. Uma subclasse pode herdar a estrutura de dados e os métodos, ou alguns dos métodos, de sua superclasse. Ela também tem métodos e às vezes, tipos de dados próprios.

**Subclasse** – uma classe que é um subtipo de uma ou mais classes (*denominadas superclasses*). Como tal, ela herda todas as características de suas superclasses. Em outras palavras, todas as características de uma classe são reusáveis por suas subclasses. Se a classe B herda de A, então dizemos que B é uma subclasse de A.

**Superclasse** – Uma classe que é um supertipo de uma ou mais classes (*tb chamadas de subclasses*). Como tal, ela é uma classe a partir da qual todas as suas características são herdadas por suas subclasses. Em outras palavras, todas as características de uma superclasse são reusáveis por aquelas classes que são seus subtipos. Se a classe B herda de A, então dizemos que A é uma superclasse de B.

**Exemplo 1**: A figura abaixo mostra uma classe e uma subclasse. A subclasse tem os mesmos métodos de sua superclasse, mas tem também o método G. Ás vezes, uma classe herda propriedades de mais de uma superclasse. Isso é denominado herança múltipla.

