Projeto T2Ti ERP 3.0

Análise e Projeto Orientado a Objetos

A T2Ti

Apresentação

A T2Ti nasce do sonho de três colegas que trabalhavam no maior banco da américa latina.

Tudo começa em 2007 com o lançamento do curso Java Starter. Logo depois veio o Siscom Java Desktop seguido de outros treinamentos.

Desde então a Equipe T2Ti se esforça para produzir material de qualidade que possa formar profissionais para o mercado, ensinando como desenvolver sistemas de pequeno, médio e grande porte.

Um dos maiores sucessos da Equipe T2Ti foi o Projeto T2Ti ERP que reuniu milhares de profissionais num treinamento dinâmico onde o participante aprendia na prática como desenvolver um ERP desde o levantamento de requisitos. Foi através desse treinamento que centenas de desenvolvedores iniciaram seu negócio próprio e/ou entraram no mercado de trabalho.

Em 2010 a T2Ti lança sua primeira aplicação para produção, o Controle Financeiro Pessoal. O sucesso foi tanto que saiu até em matéria no site Exame, ficando entre os 10 aplicativos mais baixados da semana.

Começa então a era de desenvolvimento de sistemas para alguns clientes exclusivos, pois o foco ainda era em desenvolvimento de treinamentos. A T2Ti desenvolve sistemas para o mercado nacional e internacional.

Atualmente a T2Ti se concentra nas duas vertentes: desenvolver sistemas e produzir treinamentos.

_____

Sumário

Análise e Projeto Orientado a Objetos

Modelagem de Sistemas

Introdução.

Orientação a Objetos

Paradigma.
Conceitos | Classe, Abstração, Objeto, Mensagem.
Princípios | Encapsulamento, Polimorfismo, Generalização [Herança], Agregação [Composição].

Modelagem de Sistemas

UML | História, Fases de Desenvolvimento, Notação, Visões, Modelos de Elementos, Mecanismos Gerais, Diagramas.

Modelagem de Sistemas

Introdução

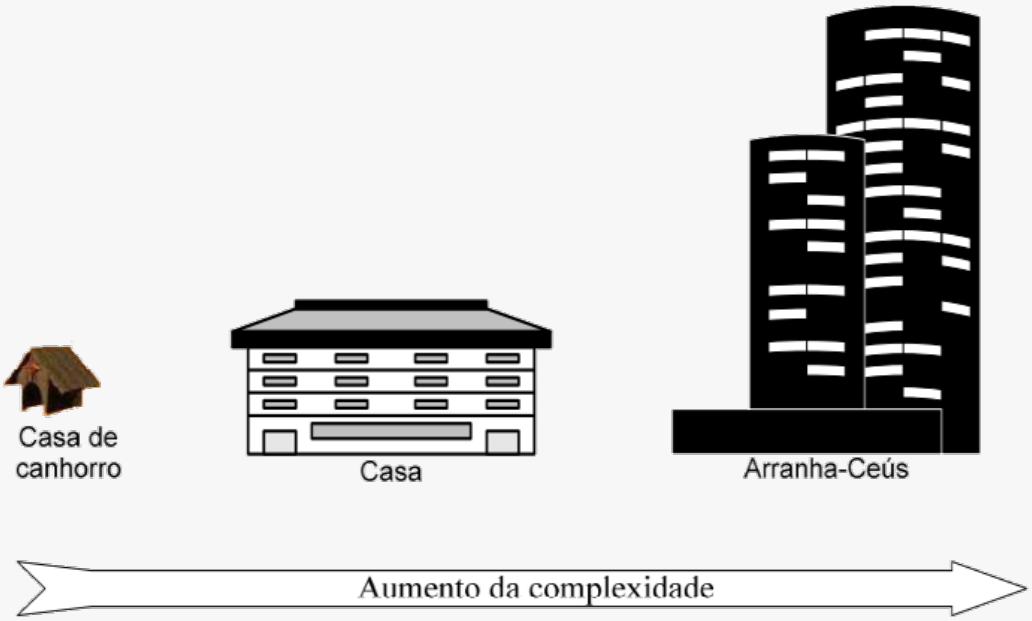
Em consequência do crescimento da importância da informação, surgiu a necessidade de gerenciar informações de uma forma adequada e eficiente e, desta necessidade, surgiram os Sistemas de Informações.

Um Sistema de Informação (SI) é uma combinação de pessoas, dados, processos, interfaces, redes de comunicação e tecnologia que interagem com o objetivo de dar suporte e melhorar o processo de negócio de uma organização com relação às informações.

O Principal objetivo da construção de um SI é a adição de valor à organização. O SI compreende os módulos funcionais computadorizados que interagem entre si para proporcionar a automatização de diversas tarefas. Uma característica intrínseca do desenvolvimento de sistemas de software é a complexidade.

Para gerenciar a complexidade são criados modelos. No caso de um projeto de construção de uma casa ou de um edifício são criadas maquetes e/ou plantas da parte hidráulica e elétrica. Esses são exemplos de modelos.

Isso também ocorre com o desenvolvimento de sistemas.



Modelagem de Sistemas

Introdução

Alguns veem a construção de modelos como uma atividade que atrasa o desenvolvimento do software propriamente dito. Mas essa atividade propicia:

- O gerenciamento da complexidade inerente ao desenvolvimento do software.
- A comunicação entre as pessoas envolvidas.
- A redução dos custos no desenvolvimento.
- A predição do comportamento futuro do sistema.

Quando tratamos da modelagem do software lidamos com digramas e documentação.

No contexto de desenvolvimento de software, os diagramas correspondem a desenhos gráficos que seguem algum padrão lógico.

Podemos também dizer que um diagrama é uma apresentação de uma coleção de elementos gráficos que possuem um significado predefinido.

Diagramas normalmente são construídos de acordo com regras de notação bem definidas.

Ou seja, cada forma gráfica utilizada em um diagrama de modelagem tem um significado específico. Diagramas permitem a construção de uma representação concisa de um sistema a ser construído.

A modelagem não pode ser composta apenas por diagramas. É preciso que haja informações textuais. Assim sendo, a união dos diagramas com as informações textuais de um sistema formam a a documentação do modelo do sistema.

A modelagem de sistemas de software consiste na utilização de notações gráficas e textuais com o objetivo de construir modelos que representam as partes essenciais de um sistema, considerando-se diversas perspectivas diferentes e complementares.

Paradigma

Um paradigma é uma forma de abordar um problema. No contexto da modelagem de um sistema de software, um paradigma tem a ver com a forma pela qual esse sistema é entendido e construído.

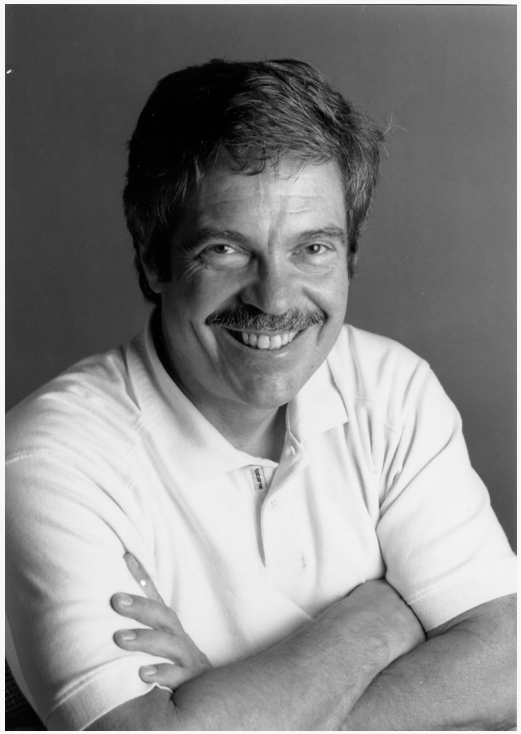
A primeira abordagem usada para modelagem de sistemas de software foi o paradigma estruturado, que tinha as seguintes características:

- Uso da técnica de decomposição funcional.
- Divisão sucessiva de um problema complexo em subproblemas.

No final dos anos 60 surgiu o paradigma da Orientação a Objetos. Levou um tempo até que esse paradigma suplantasse o estruturado, mas isso ocorreu e hoje é comum que os sistemas sejam construídos com o paradigma OO.

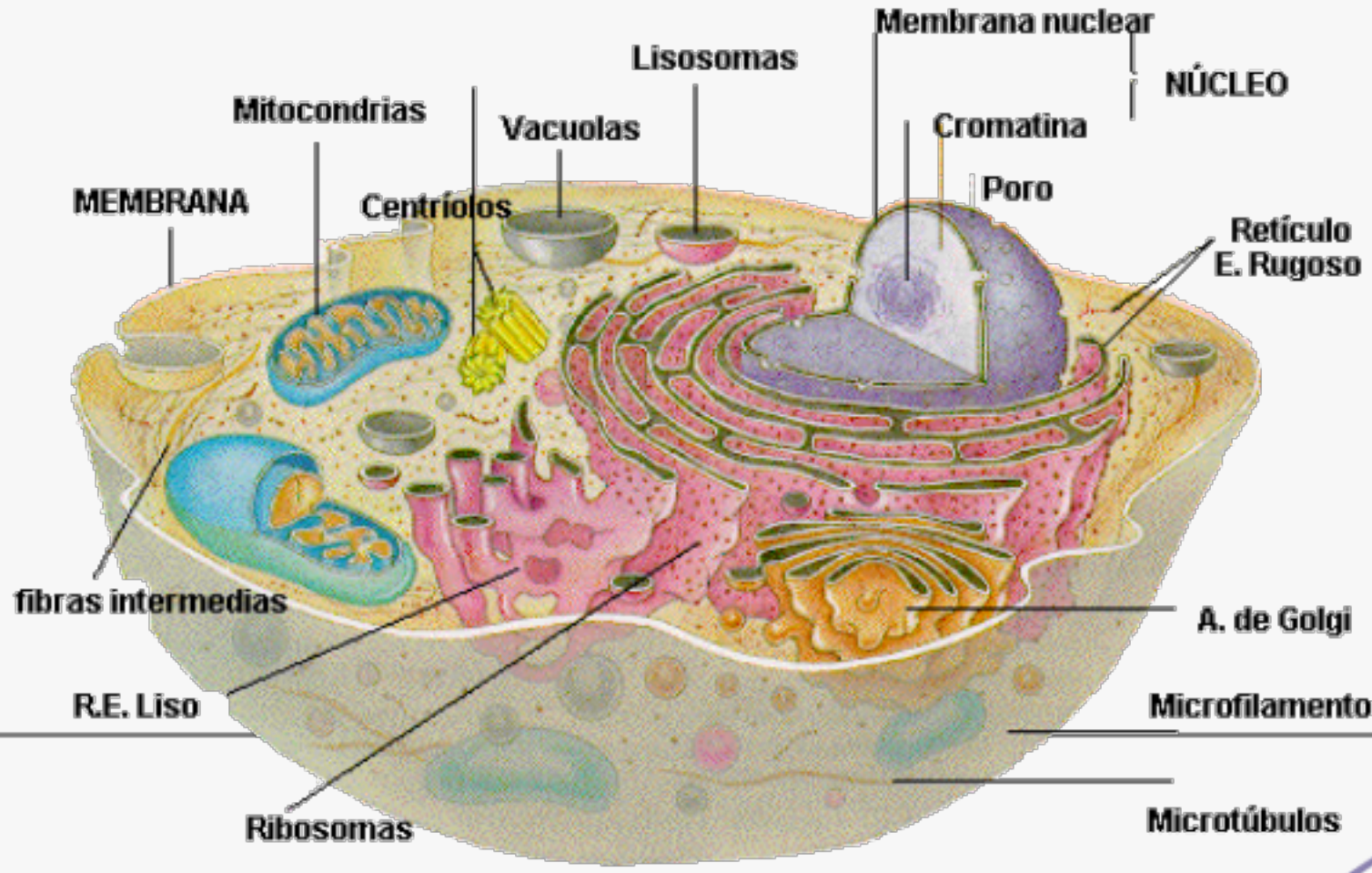
Alan Kay, um dos pais desse paradigma, formulou a chamada analogia biológica.

Como seria um sistema de software que funcionasse como um ser vivo?



Paradigma

Cada célula interagiria com outras células através do envio de mensagens para realizar um objetivo comum. Adicionalmente, cada célula se comportaria como uma unidade autônoma. De uma forma mais geral, Kay pensou em como construir um sistema de software a partir de agentes autônomos que interagem entre si.

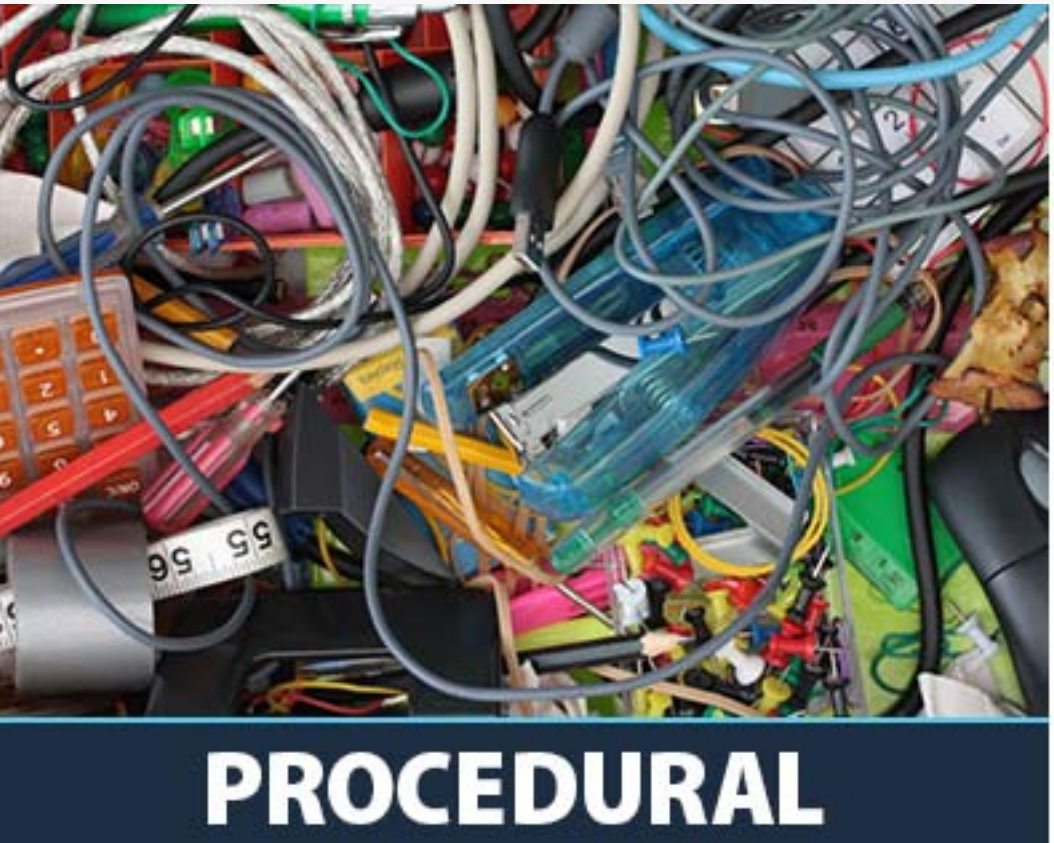


Orientação a Objetos

Paradigma

Através de sua analogia biológica, Alan Kay definiu os fundamentos da orientação a objetos.

1. Qualquer coisa é um objeto.
2. Objetos realizam tarefas através da requisição de serviços a outros objetos.
3. Cada objeto pertence a uma determinada classe. Uma classe agrupa objetos similares.
4. A classe é um repositório para comportamento associado ao objeto.
5. Classes são organizadas em hierarquias.



Paradigma

O paradigma da orientação a objetos visualiza um sistema de software como uma coleção de agentes interconectados chamados objetos. Cada objeto é responsável por realizar tarefas específicas.

É através da interação entre objetos que uma tarefa computacional é realizada.

Um sistema de software orientado a objetos consiste de objetos em colaboração com o objetivo de realizar as funcionalidades deste sistema. Cada objeto é responsável por tarefas específicas.

É através da cooperação entre objetos que a computação do sistema se desenvolve.

A Orientação a Objetos contém conceitos e princípios:

Conceitos:

- Classe
- Objeto
- Mensagem

Princípios:

- Encapsulamento
- Polimorfismo
- Generalização (Herança)
- Composição

O bom entendimento desses conceitos e princípios fará com que o leitor compreenda o paradigma da orientação a objetos e domine a técnica de desenvolver sistemas dessa maneira.

Orientação a Objetos

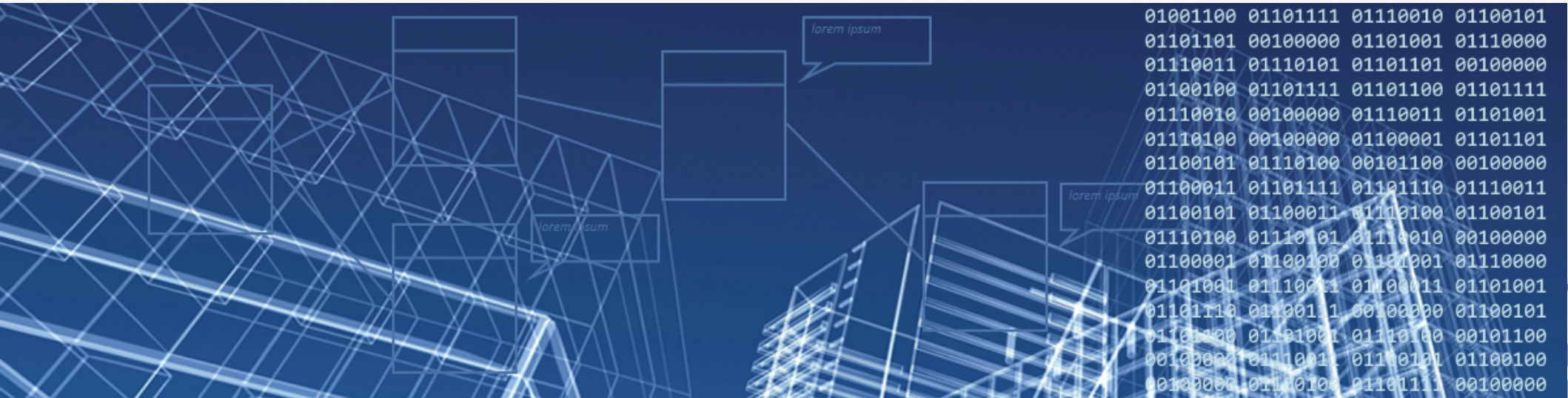
Conceitos | Classe

O mundo real é formado de coisas. Na terminologia de orientação a objetos, estas coisas do mundo real são denominadas objetos. Seres humanos costumam agrupar os objetos para entendê-los. A descrição de um grupo de objetos é denominada classe de objetos, ou simplesmente de classe.

Objetos de estrutura e comportamento idênticos são descritos como pertencendo a uma classe, de tal forma que a descrição de suas propriedades pode ser feita de uma só vez, de forma concisa, independente do número de objetos idênticos.

Uma classe é um molde para objetos. Diz-se que um objeto é uma instância de uma classe. Uma classe é uma abstração das características relevantes de um grupo de coisas do mundo real.

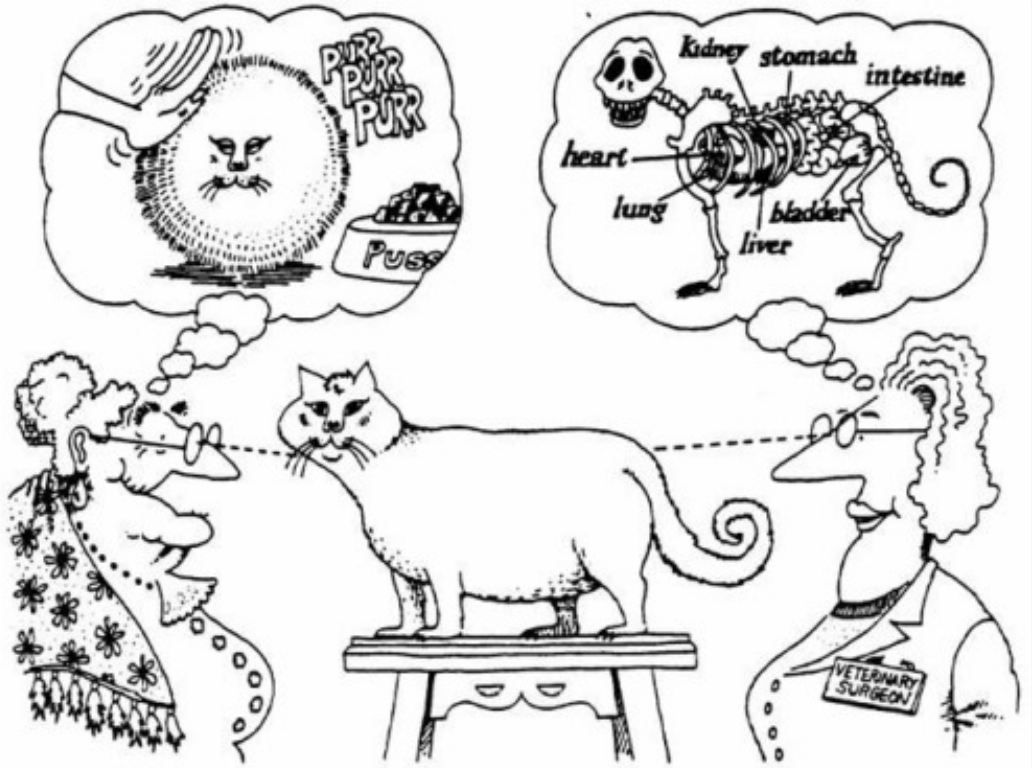
Uma classe provê toda a informação necessária para construir e utilizar objetos de um tipo particular, ou seja, descreve a forma da memória privada e como se realizam as operações das suas instâncias. Os métodos residem nas classes, uma vez que todas as instâncias de uma classe possuem o mesmo conjunto de métodos.

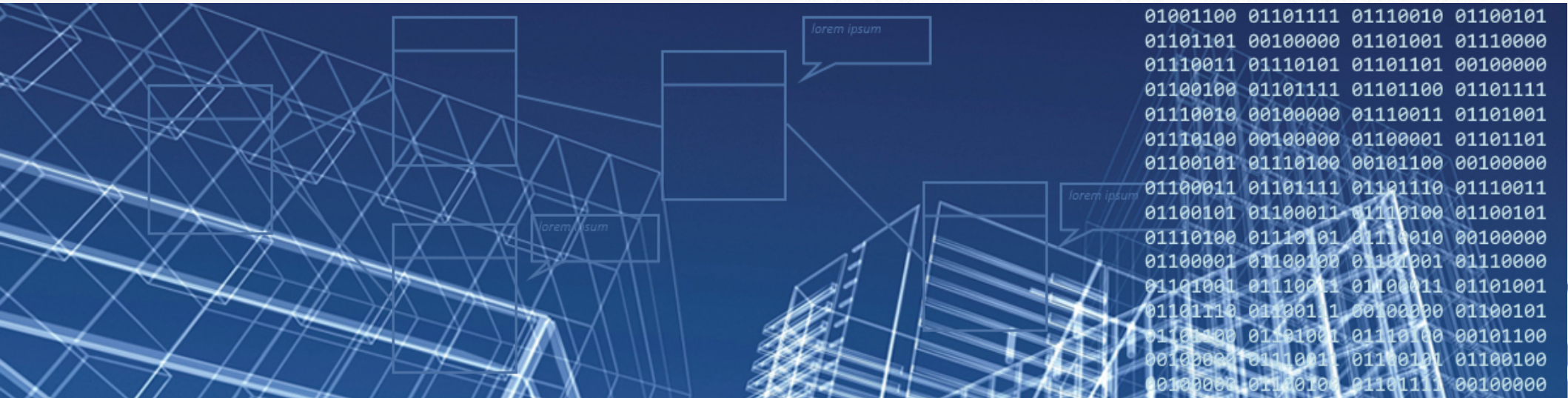


Orientação a Objetos

Conceitos | Abstração

Uma abstração é qualquer modelo que inclui os aspectos relevantes de alguma coisa, ao mesmo tempo em que ignora os menos importantes. Em termos de sistemas, o desenvolvedor abstrai um objeto quando ele sabe como ele se comporta, mas não precisa conhecer exatamente como funciona cada aspecto do comportamento do objeto. Se a classe FUNCIONARIO sabe como receber o salário e para isso espera apenas um valor, basta passar para o objeto dessa classe o valor do salário e deixar que ele realize o procedimento, sem precisar saber como ele faz isso.



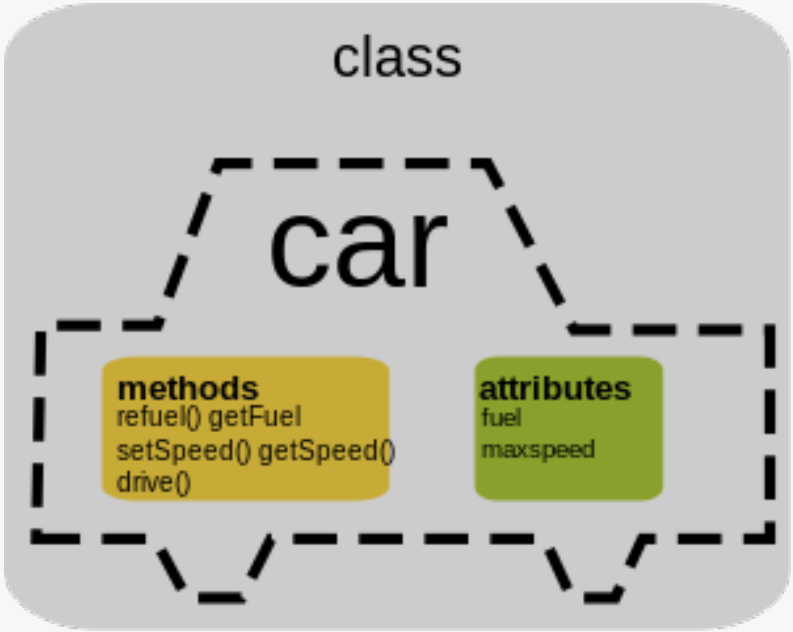


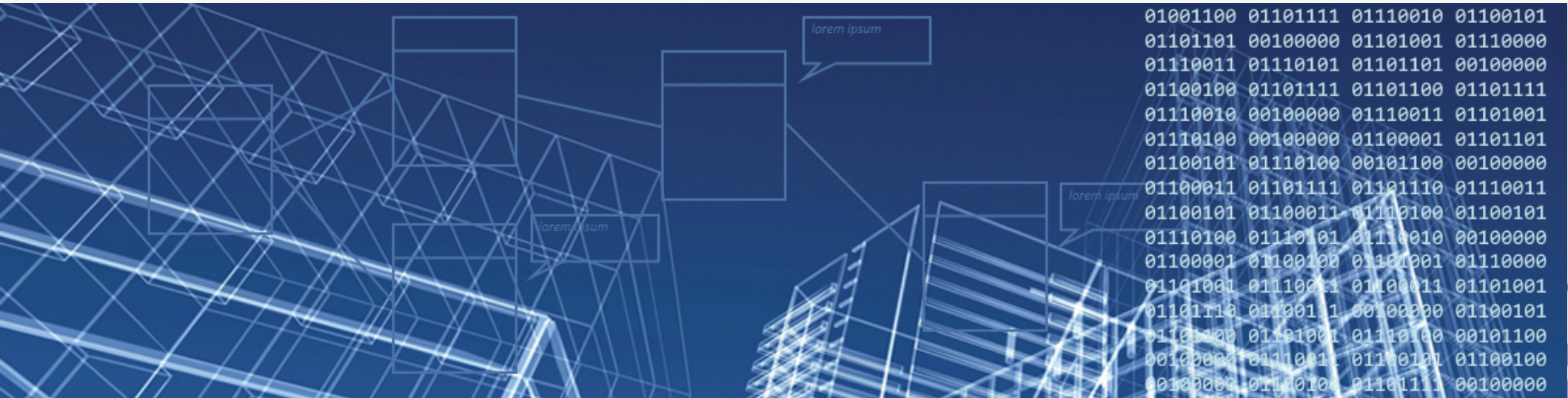
Orientação a Objetos

Conceitos | Objeto

O objeto é a instância de uma classe. Um objeto é composto por:

- Propriedades ou Atributos: são as informações, estruturas de dados que representam o estado interno do objeto. Em geral, não são acessíveis aos demais objetos.
- Comportamento: conjunto de operações, chamados de métodos, que agem sobre as propriedades.
- Identidade: é uma propriedade que diferencia um objeto de outro. É o seu nome.





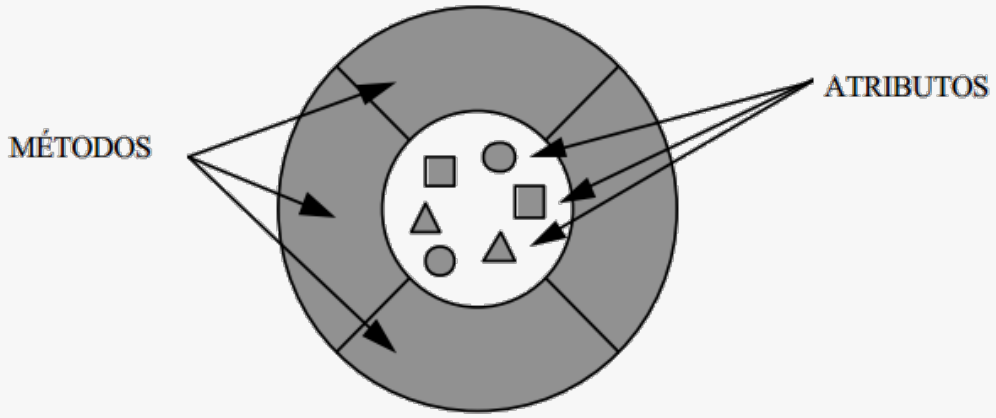
Orientação a Objetos

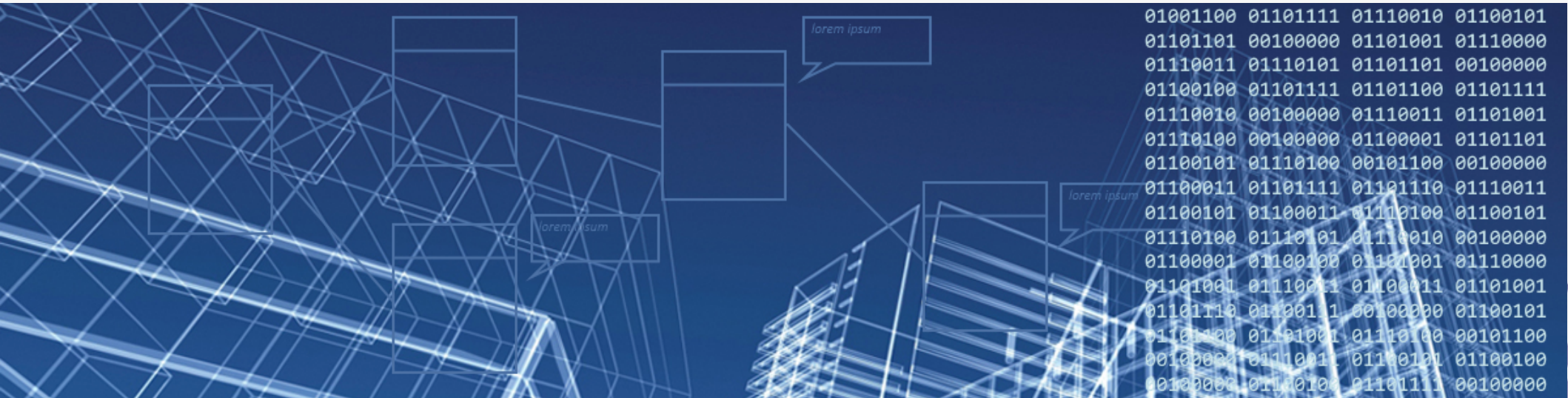
Conceitos | Objeto

Dessa forma, percebe-se que um objeto possui uma identidade, que é o seu nome e, além disso, é formado por métodos (comportamento) e atributos (propriedades).

É fácil identificar os objetos no mundo real. Por exemplo, numa empresa podemos encontrar o objeto FUNCIONÁRIO ou COLABORADOR. Quais seriam o estado (atributos, propriedades, variáveis) e comportamento (métodos) desse objeto?

Vejamos.





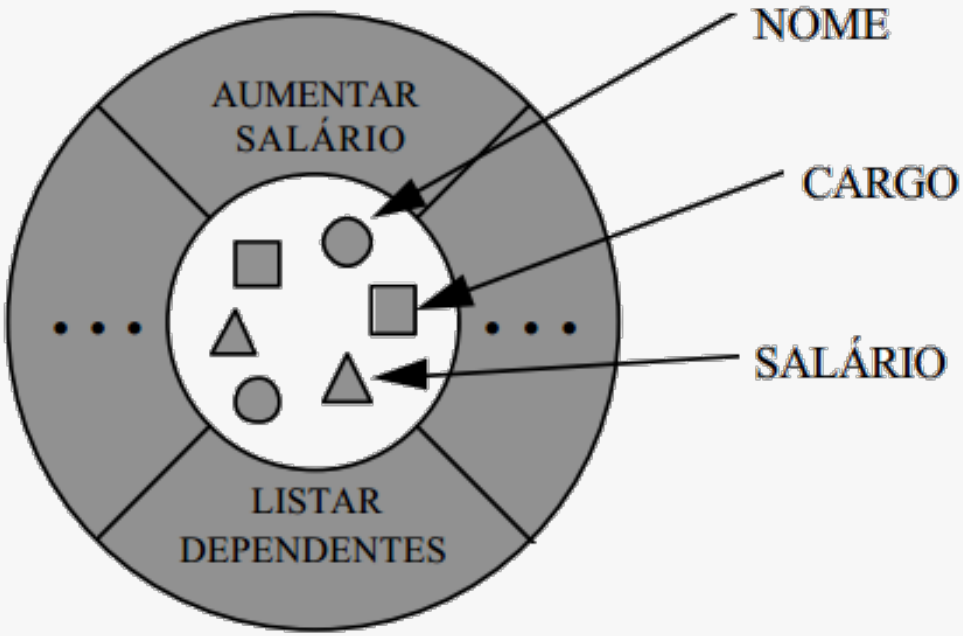
Orientação a Objetos

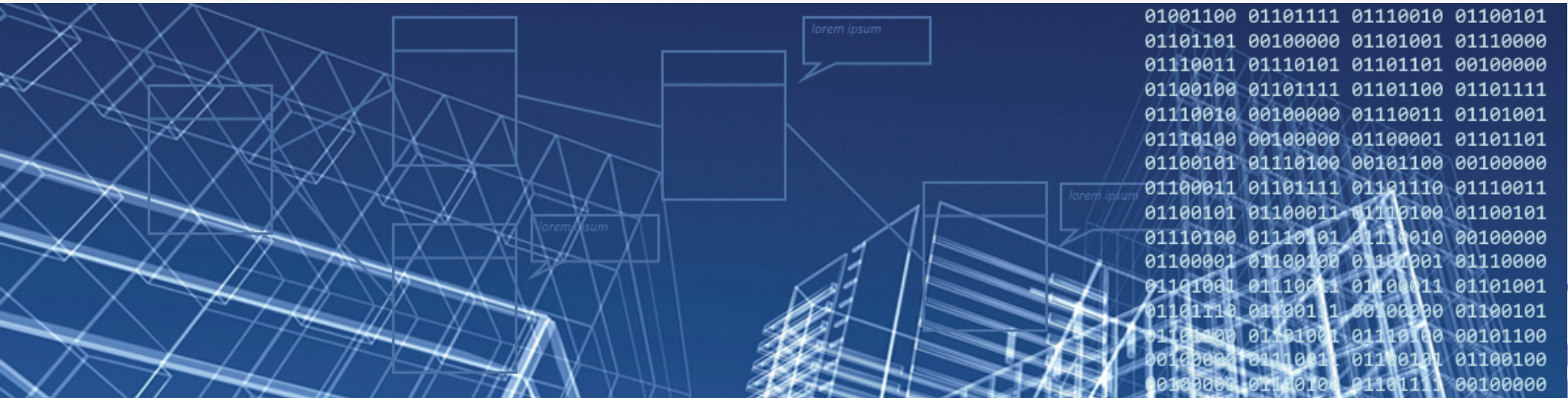
Conceitos | Objeto

Dentre os muitos atributos possíveis para um colaborador, podemos citar: NOME, CARGO e SALÁRIO.

Dentre os possíveis métodos estão: AUMENTAR SALÁRIO e LISTAR DEPENDENTES.

E, dessa maneira, o analista vai encontrando todos os objetos de um domínio para a modelagem prévia de um sistema que será desenvolvido.





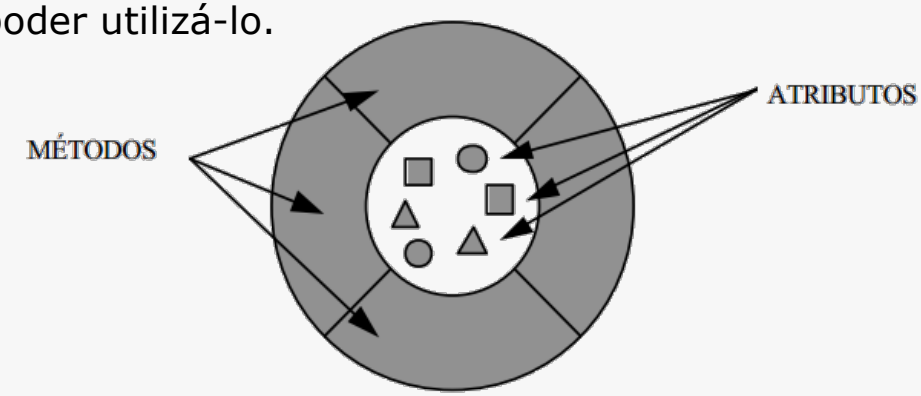
Orientação a Objetos

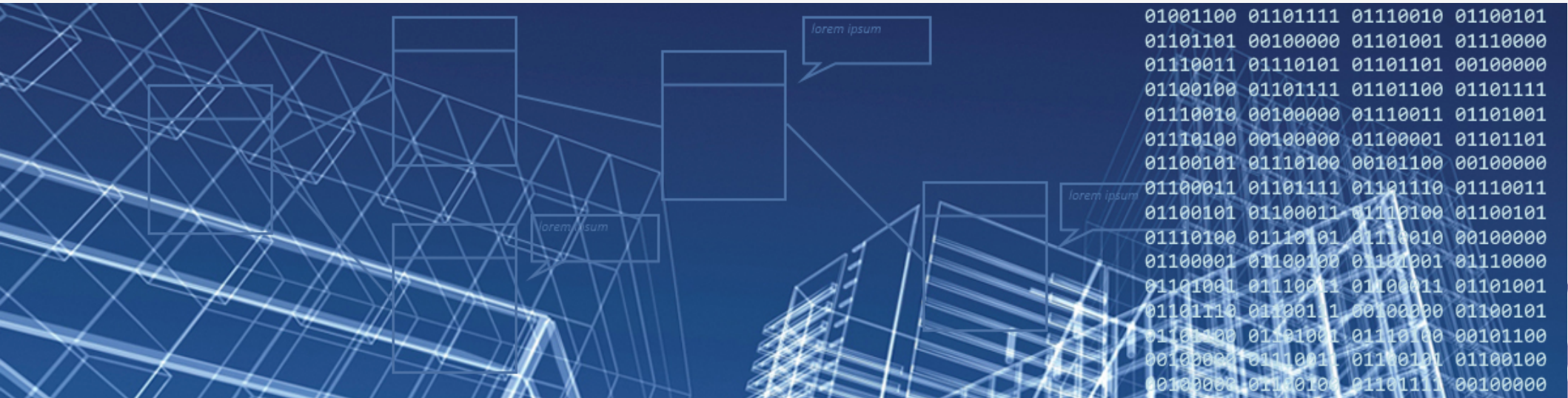
Conceitos | Objeto

Utilizar objetos traz alguns benefícios. Já que será utilizado o princípio da abstração de dados e cada objeto saberá exatamente o que fazer, temos:

Modularidade: o código fonte para um objeto pode ser escrito e mantido independente do código fonte de outros objetos. Além disso, um objeto pode ser facilmente migrado para outros sistemas.
Ocultamento de Informações: um objeto tem uma interface pública que os outros objetos podem utilizar para estabelecer comunicação com ele.

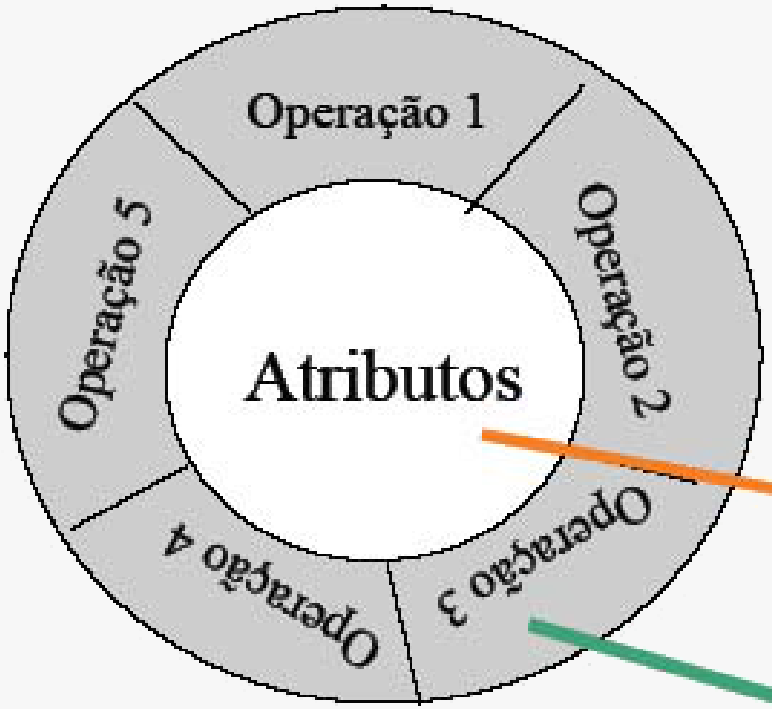
Mas, o objeto mantém informações e métodos privados que podem ser alterados a qualquer hora sem afetar os outros objetos que dependem dele. Ou seja, não é necessário saber como o objeto é implementado para poder utilizá-lo.





Orientação a Objetos

Conceitos | Objeto



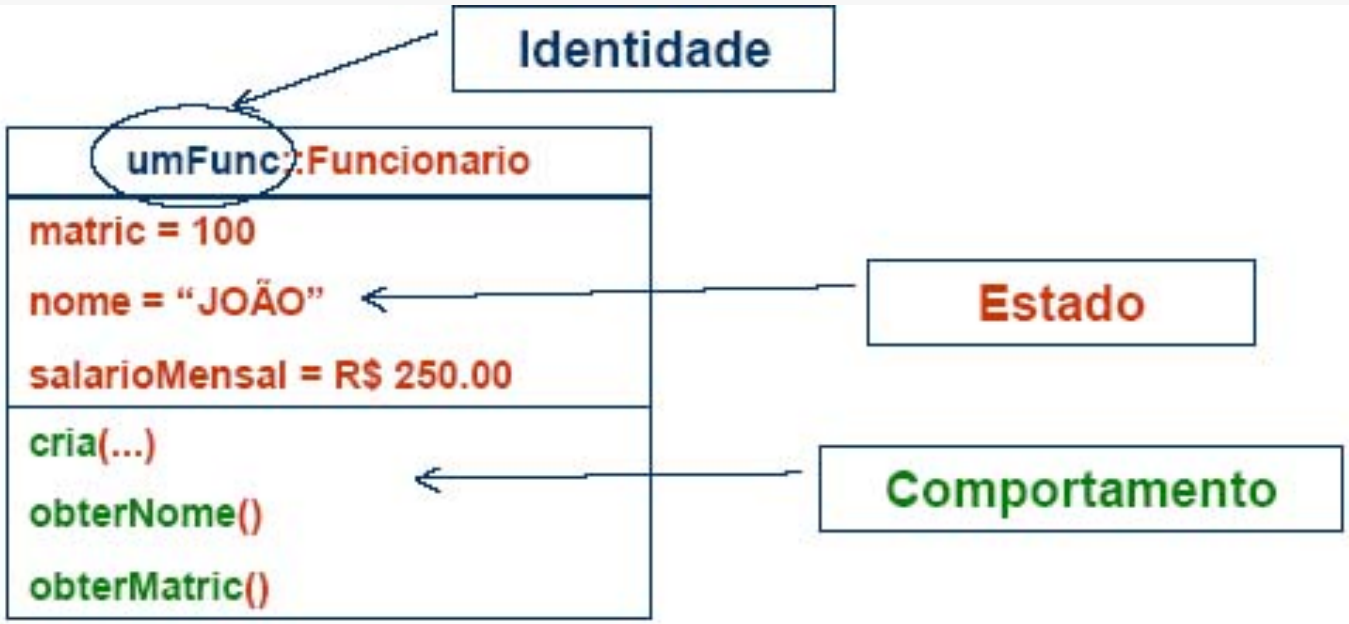
Objeto geométrico

cor:
posição:

selecionar(p: Ponto): boolean
girar(Ângulo: real)
mover(delta: coord)

Orientação a Objetos

Conceitos | Objeto



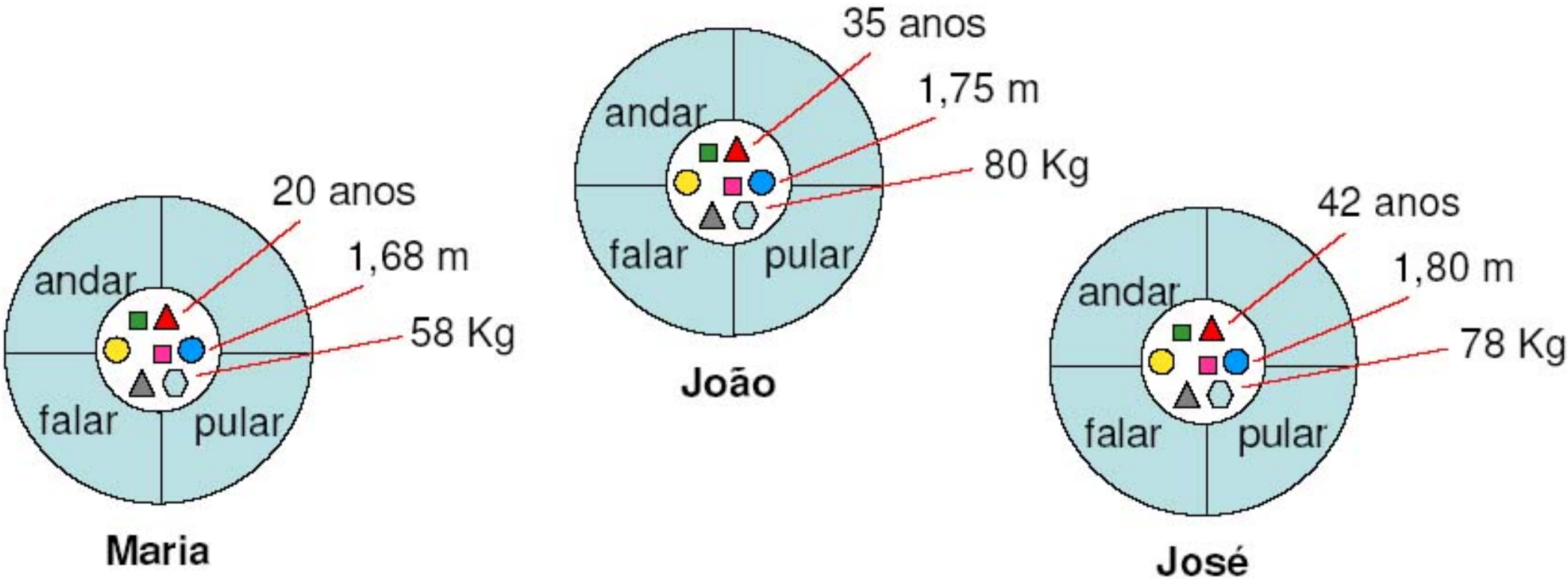

```
procedure TForm1.Button1Click(Sender: TObject);
var   umFunc: Funcionario;
begin
  umFunc := Funcionario.criar('JOÃO', 100, 250.00);
  Label4.Caption := umFunc.obterNome();
  Label2.Caption := IntToStr(umFunc.obterMatric());
end;
```

método para criar o objeto

Orientação a Objetos

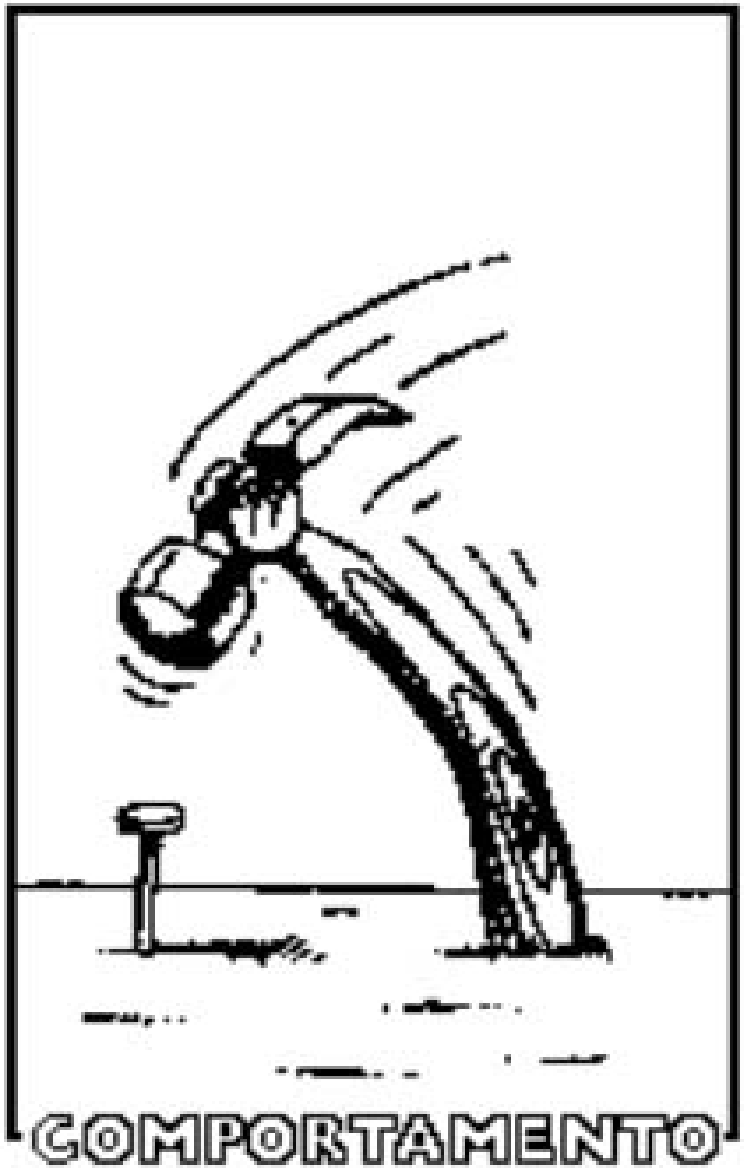
Conceitos | Objeto

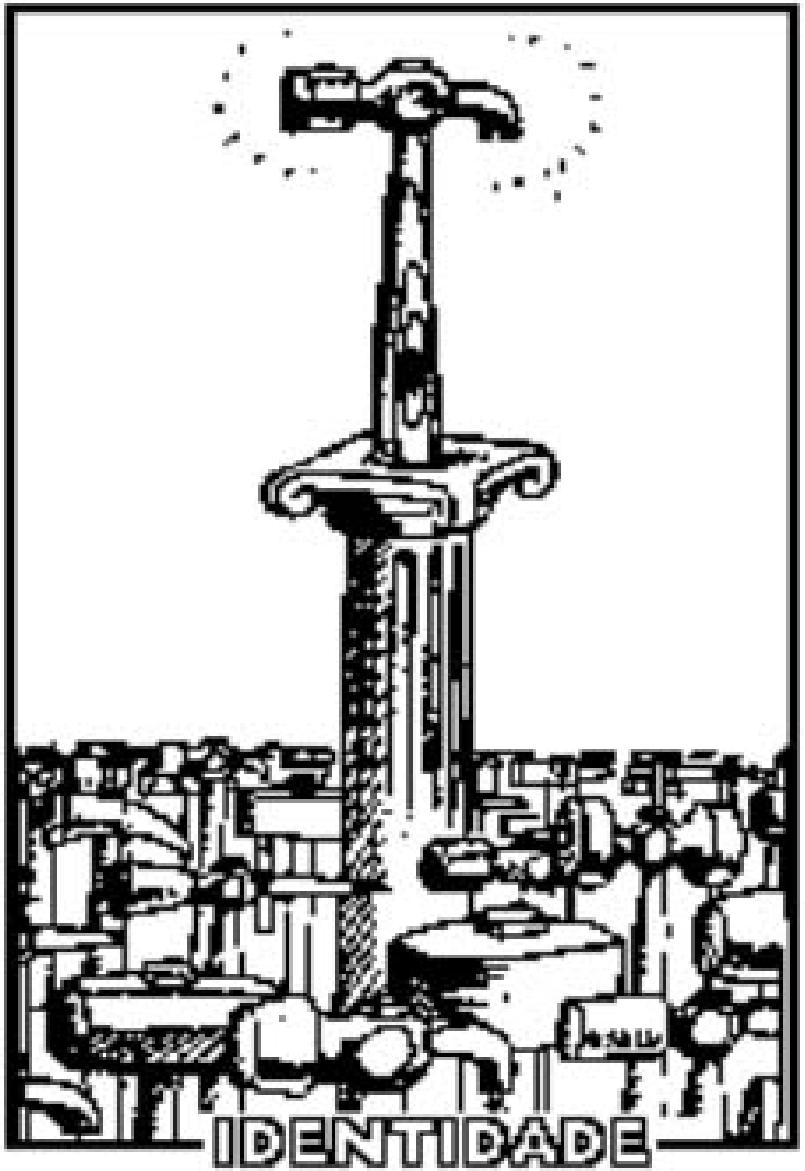
Um sistema pode conter um ou inúmeros objetos ativos. Cada objeto ativo no sistema em particular também é chamado de instância. As diferentes instâncias possuem seu próprio estado. O exemplo abaixo mostra várias instâncias de Pessoa. Cada instância, além do estado (atributos), também possui seus métodos (comportamento) que operam sobre o próprio estado. Em outras palavras, para pular, cada pessoa vai fazer uma determinada força dependendo da sua idade, altura e peso, por exemplo. A ideia é que cada objeto seja responsável por seus dados (estado, atributos) e seja capaz de realizar as próprias operações que lhe foram atribuídas (comportamento).



Orientação a Objetos

Conceitos | Objeto





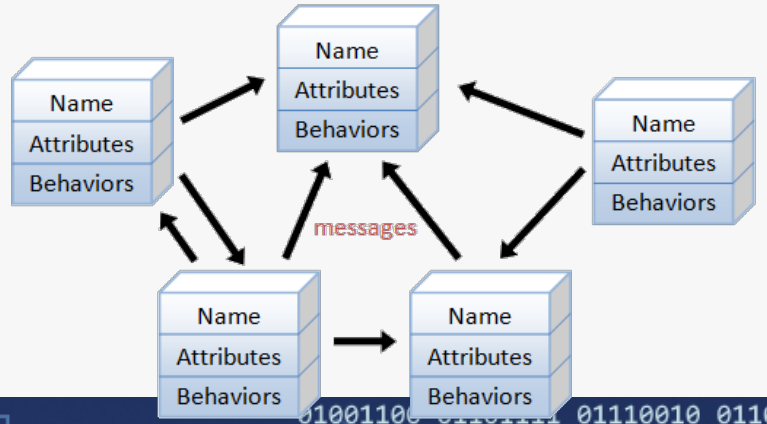
Orientação a Objetos

Conceitos | Mensagem

Para que um objeto realize alguma tarefa, deve haver um estímulo enviado a este objeto. Faz sentido dizer que tal objeto pode responder a estímulos a ele enviados, assim como faz sentido dizer que seres vivos reagem a estímulos que eles recebem, pois os objetos representam algo do mundo real.

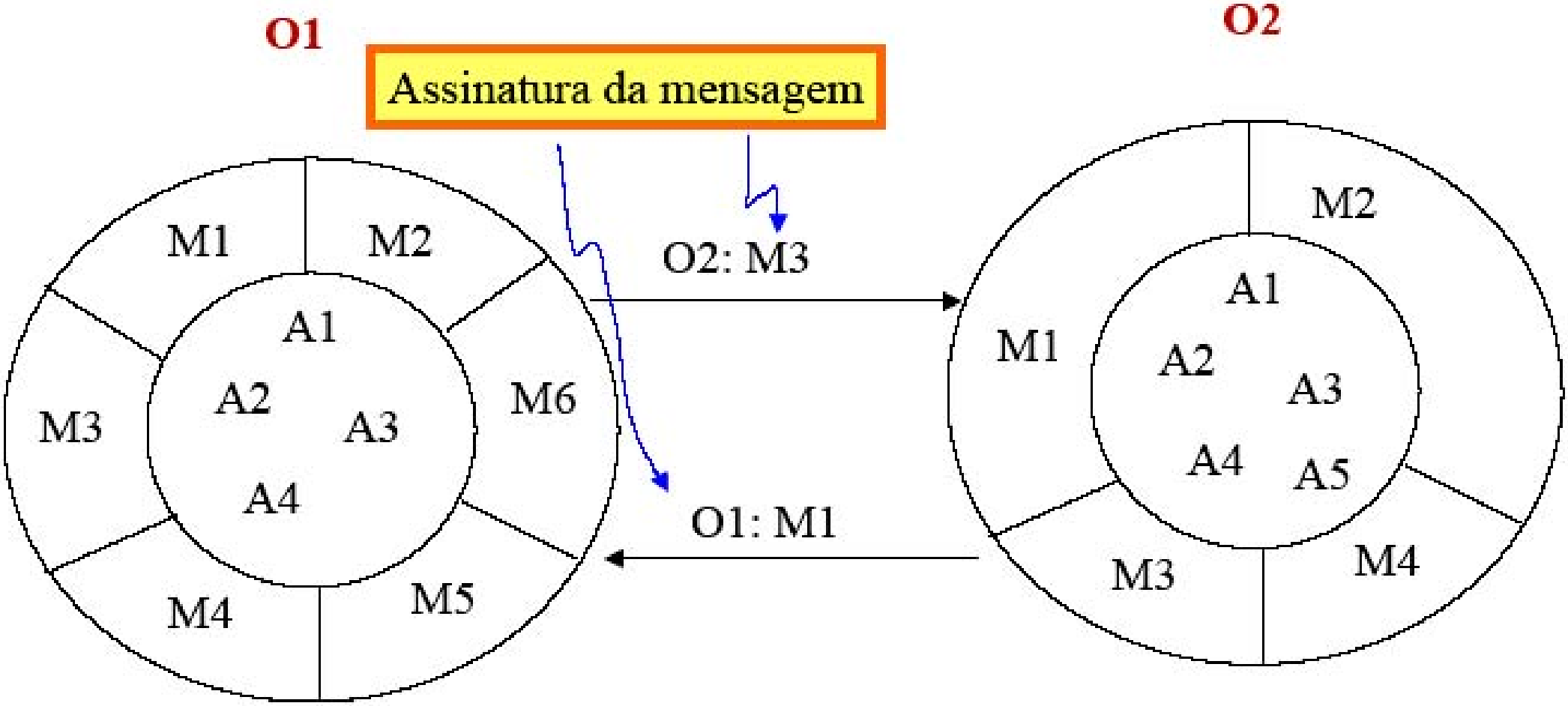
Independentemente da origem do estímulo, quando ele ocorre, diz-se que o objeto em questão está recebendo uma mensagem. Uma mensagem é uma requisição enviada de um objeto a outro para que este último realize alguma operação.

Objetos de um sistema trocam mensagens. Isto significa que estes objetos estão enviando mensagens uns aos outros com o objetivo de realizar alguma tarefa dentro do sistema no qual eles estão inseridos.



Conceitos | Mensagem

Objetos "conversam" uns com outros por meio de mensagens, solicitando serviços



Orientação a Objetos

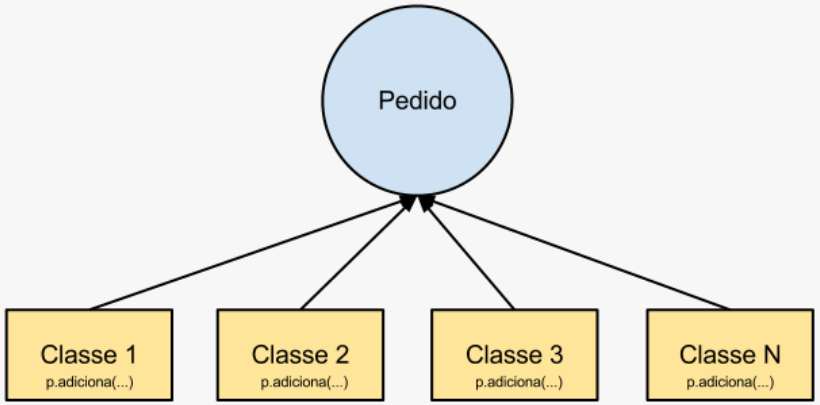
Princípios | Encapsulamento

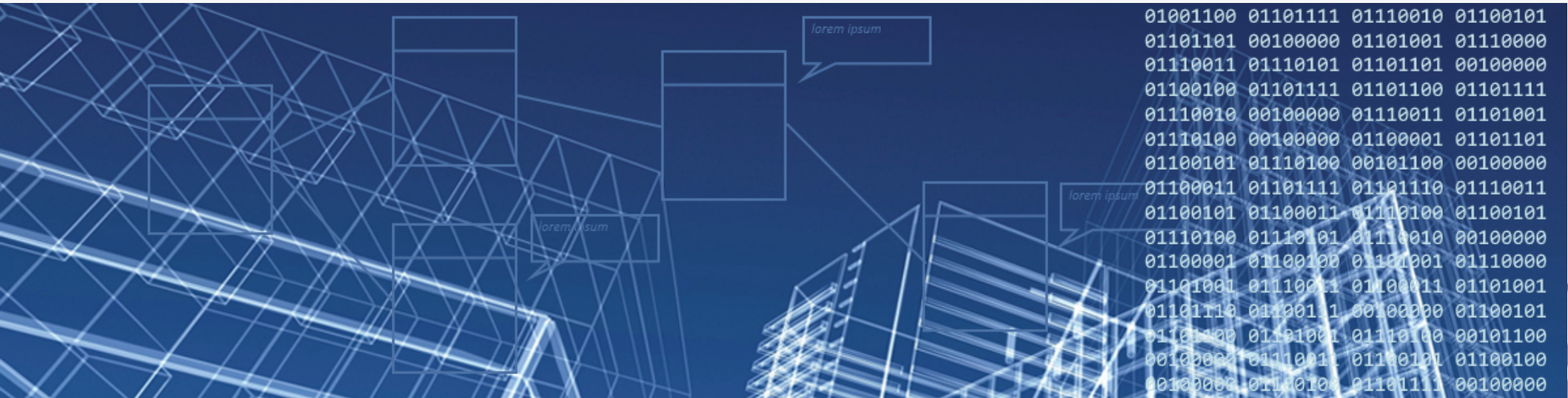
Já vimos que os objetos possuem comportamento. O termo comportamento diz respeito a que operações são realizadas por um objeto e também de que modo estas operações são executadas.

De acordo com o encapsulamento, os objetos devem "ocultar" a sua complexidade. Esse princípio aumenta qualidade dos sistemas, em termos de:

– Legibilidade
– Clareza
– Reuso

Pelo princípio do encapsulamento, a implementação utilizada por um objeto receptor de uma mensagem não importa para um objeto remetente da mesma.



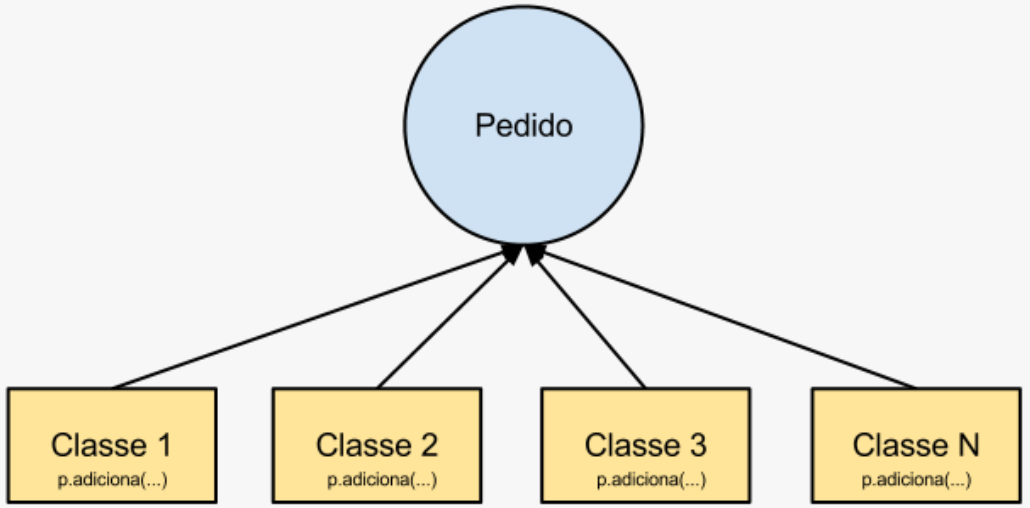


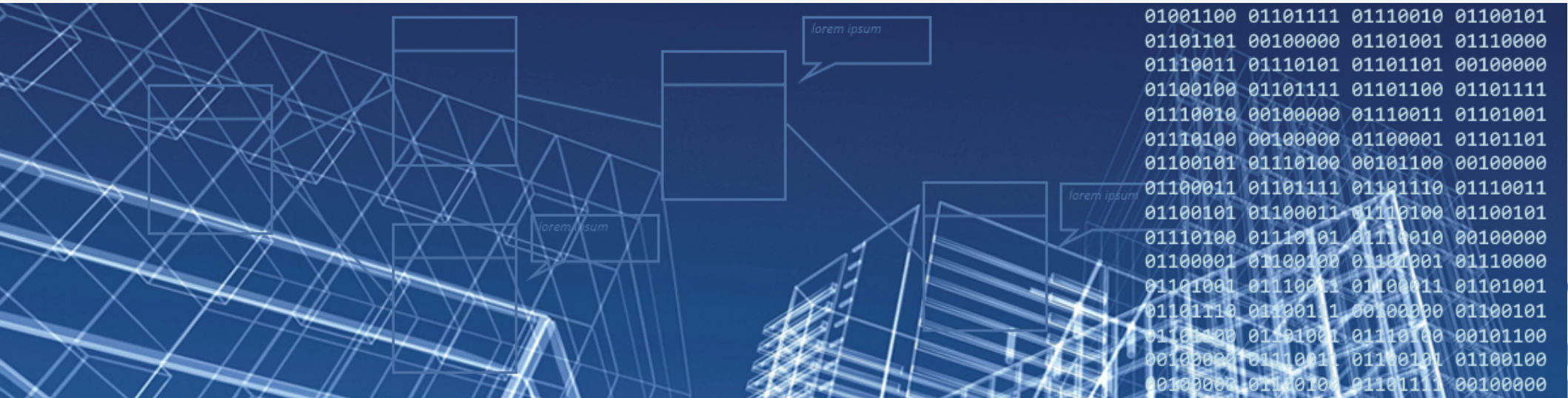
Orientação a Objetos

Princípios | Encapsulamento

Observe na imagem que as classes “Classe 1”, “Classe 2”, “Classe 3” e “Classe N” serão instanciadas, ou seja, terão seus respectivos objetos colocados na memória e enviarão uma mensagem para o objeto “Pedido”.

Cada uma essas classes instancia Pedido através de uma variável “p”. Elas chamam o método “adiciona()” do objeto Pedido, da seguinte forma: “p.adiciona(...)”. As classes não sabem como o método “adiciona” está implementado, pois o mesmo está encapsulado.



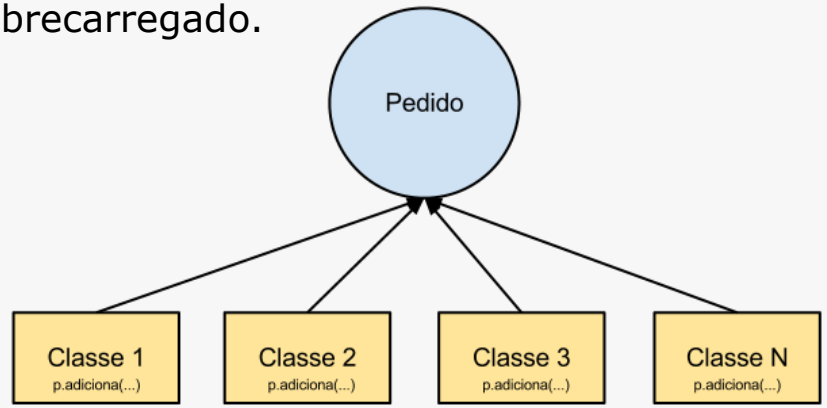


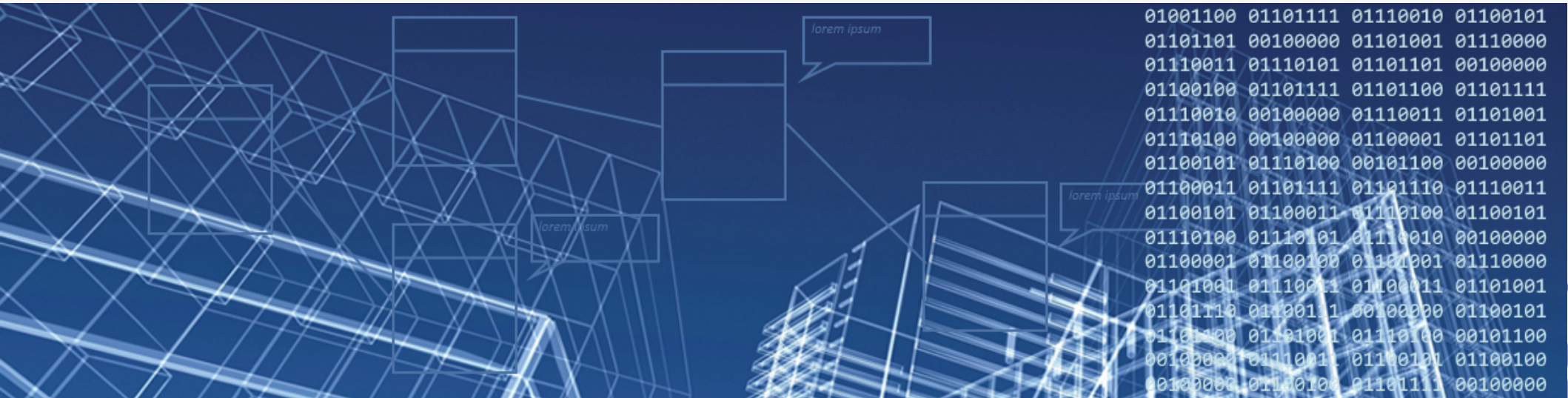
Orientação a Objetos

Princípios | Polimorfismo

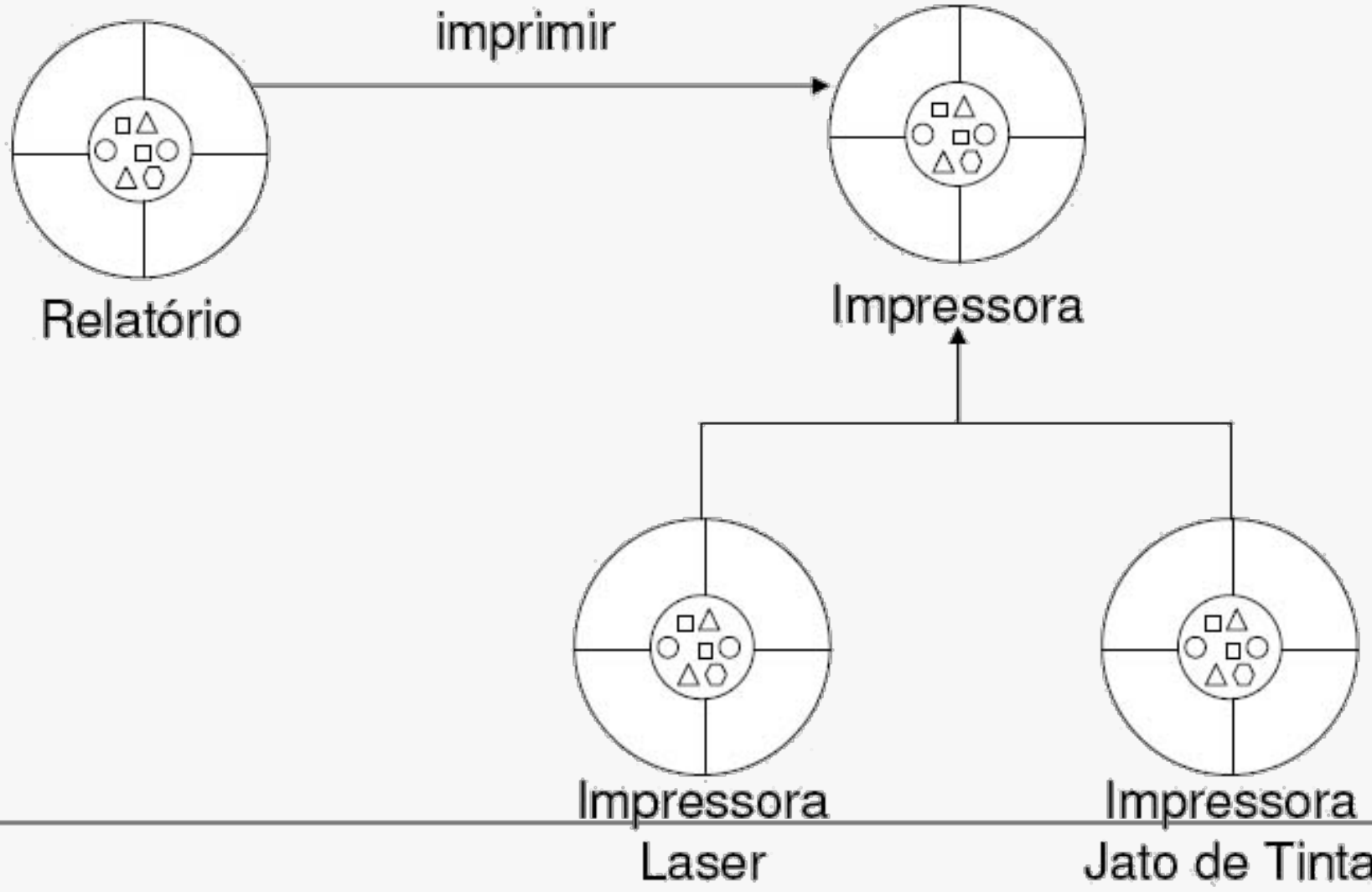
Polimorfismo refere-se à capacidade de dois ou mais objetos responderem à mesma mensagem, cada um a seu próprio modo. A utilização da herança torna-se fácil com o polimorfismo. Desde que não é necessário escrever um método com nome diferente para responder a cada mensagem, o código é mais fácil de entender. No exemplo do Pedido, podemos pensar na utilização do polimorfismo paramétrico. Ele ocorre quando existem dois ou mais métodos com o mesmo nome, mas com parâmetros diferentes. Essa característica é conhecida como sobrecarga.

Imagine que a "Classe 1" precisasse passar uma data para pedido ao passo que a "Classe 2" precisasse passar a data e um valor. Pedido teria duas implementações de "adiciona()", ou seja, o método "adiciona()" seria sobrecarregado.





Princípios | Polimorfismo



Orientação a Objetos

Princípios | Generalização (Herança)

O mecanismo de herança permite a reutilização das propriedades de uma classe na definição de outra. A classe mais generalizada é chamada superclasse e a mais especializada, subclasse.

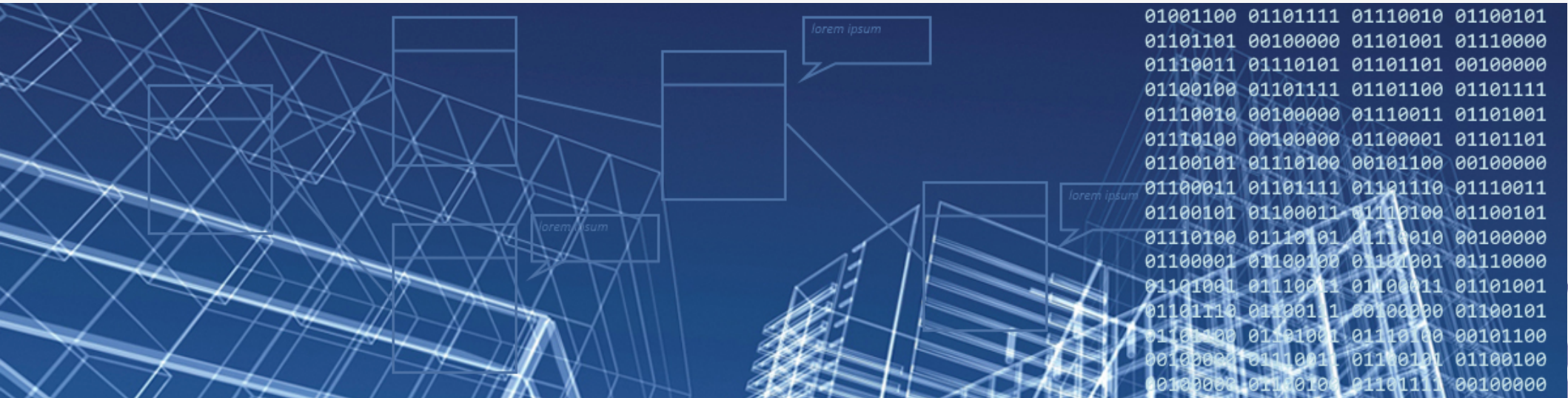
Assim como no mundo real, o objeto descendente não tem nenhum trabalho para receber a herança. Ele a recebe simplesmente porque é um objeto descendente.

A herança facilita o compartilhamento de comportamento entre classes semelhantes.

Na herança, classes semelhantes são agrupadas em hierarquias.

– Cada nível de uma hierarquia pode ser visto como um nível de abstração.
– Cada classe em um nível da hierarquia herda as características das classes nos níveis acima.

As diferenças ou variações de uma classe em particular podem ser organizadas de forma mais clara.

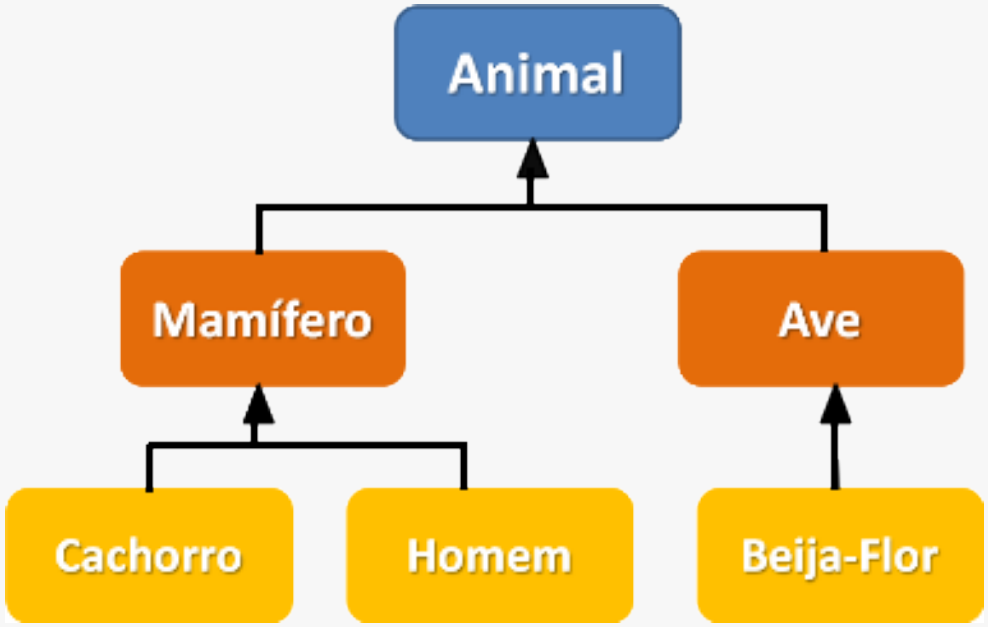


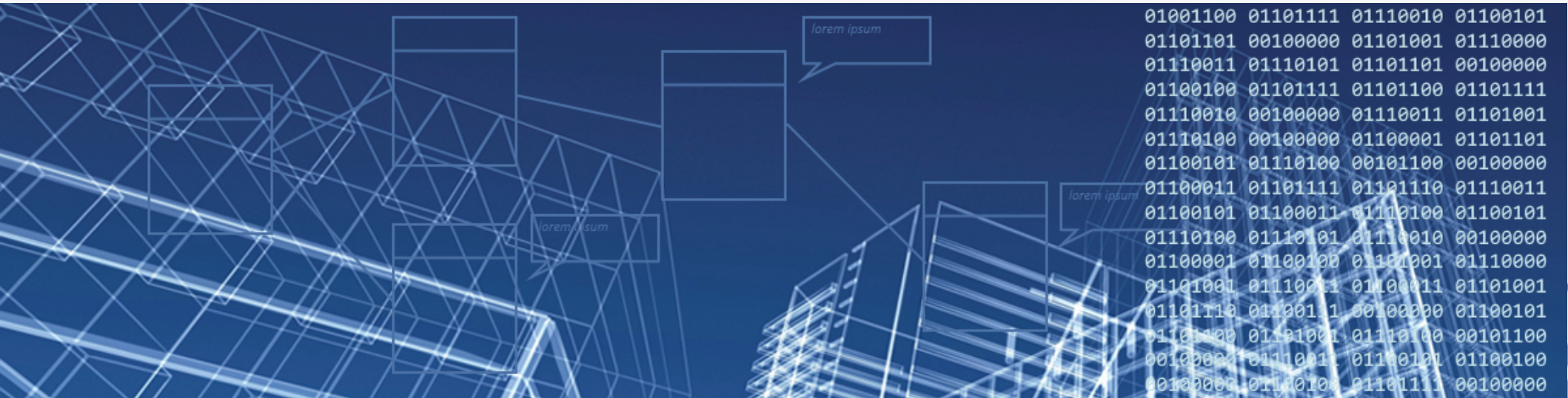
Orientação a Objetos

Princípios | Generalização (Herança)

A herança não é limitada a apenas um nível. A hierarquia de classes pode ser tão profunda quanto for preciso. Métodos e variáveis internas são herdados por todos os objetos dos níveis abaixo. Quanto mais para baixo na hierarquia uma classe estiver, mais especializado o seu comportamento.

Várias subclasses descendentes podem herdar as características de uma superclasse ancestral, assim como vários seres humanos herdam as características genéticas de um antepassado.



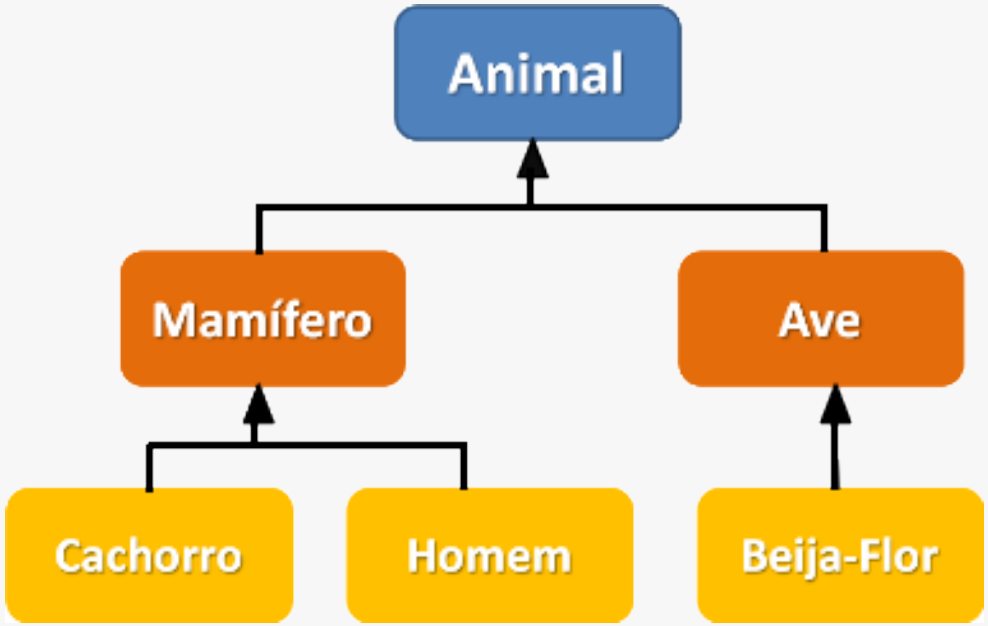


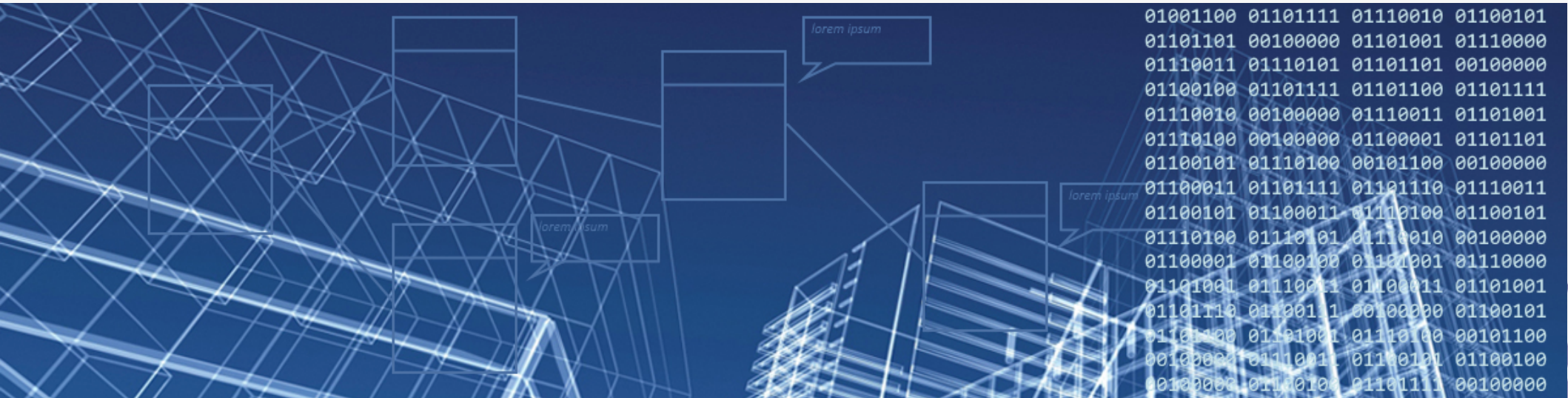
Orientação a Objetos

Princípios | Generalização (Herança)

A herança é interpretada da seguinte maneira: se Mamífero é subclasse de Animal então:

- Os objetos da classe Mamífero suportam todas as operações suportadas pelos objetos da classe Animal, exceto aquelas redefinidas.
- As variáveis de instância de Mamífero incluem todas as variáveis de instância de Animal.



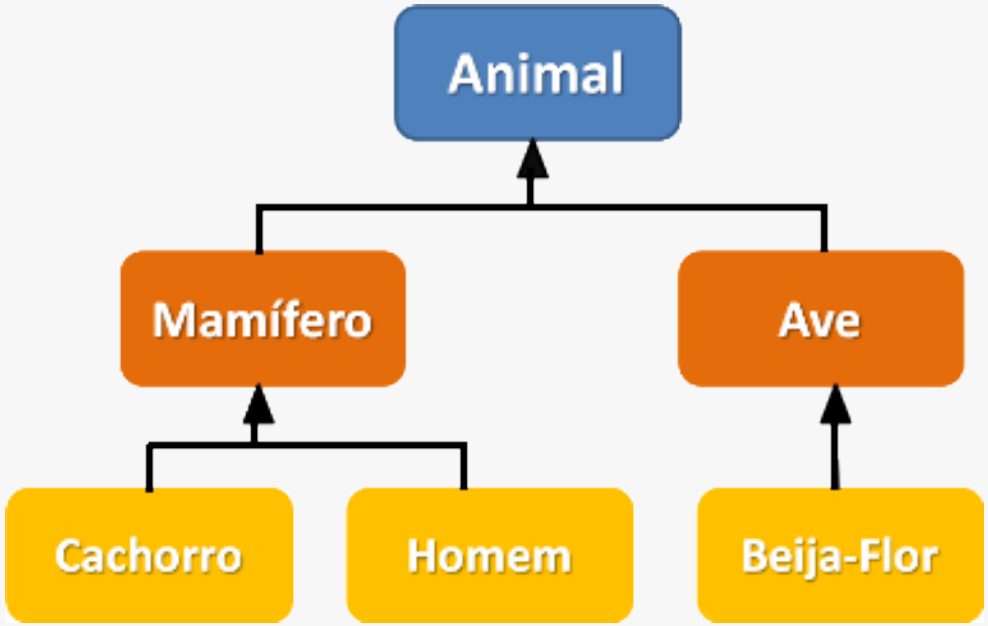


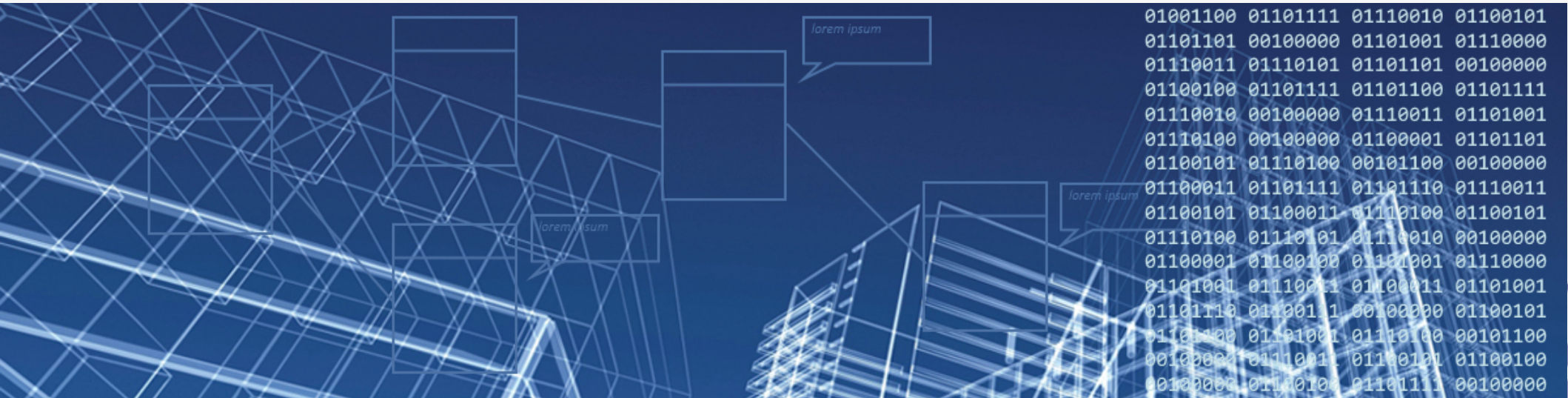
Orientação a Objetos

Princípios | Generalização (Herança)

Sendo assim, notamos que uma subclasse compartilha o comportamento e o estado de sua superclasse, podendo redefini-lo para seu próprio uso.

Além disso, é possível acrescentar novas características que identificam um comportamento próprio. A maior parte das linguagens orientadas a objetos utiliza herança como uma técnica para prover suporte à especialização, e não permitem que uma classe exclua uma operação herdada.





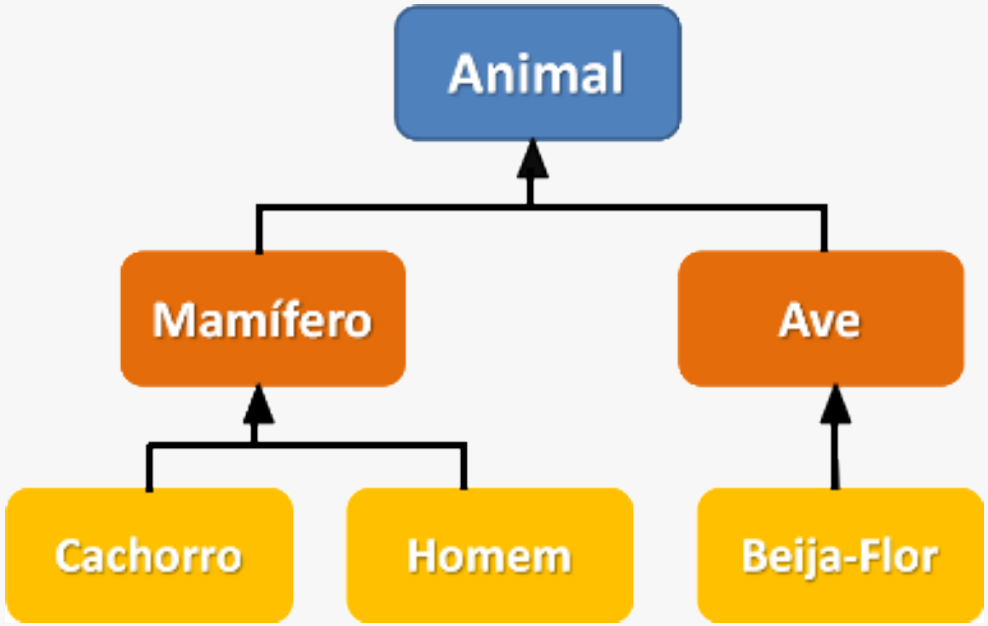
Orientação a Objetos

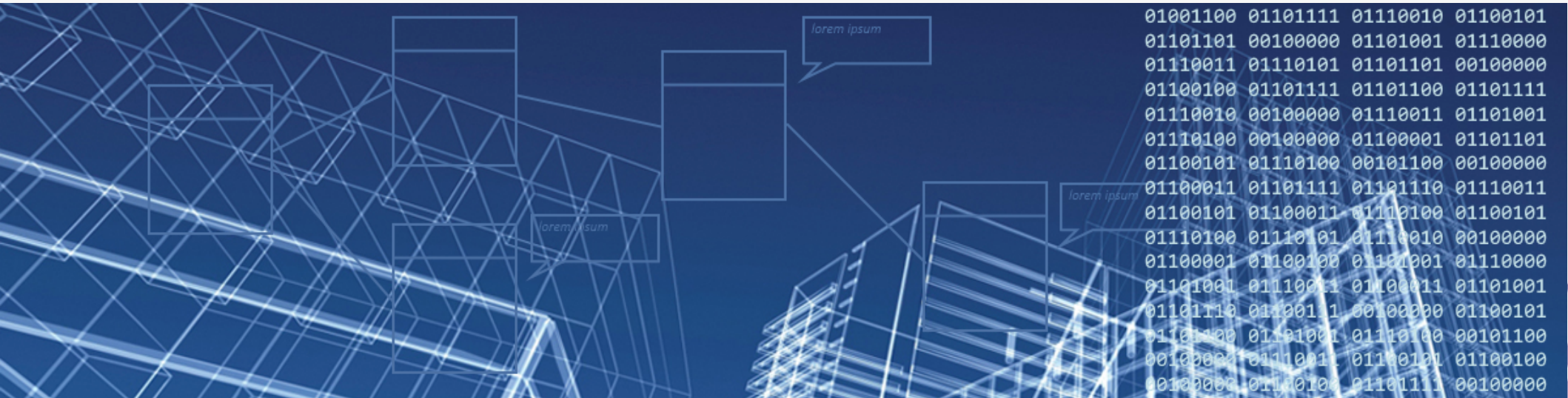
Princípios | Generalização (Herança)

Lembra quando mencionamos a sobrecarga lá no tópico do Polimorfismo? A utilização da herança torna-se fácil com o polimorfismo. Desde que não é necessário escrever um método com nome diferente para responder a cada mensagem, o código é mais fácil de entender.

Por exemplo, teríamos:

```
Animal.respirar();
Mamífero.respirar();
Cachorro.respirar();
```



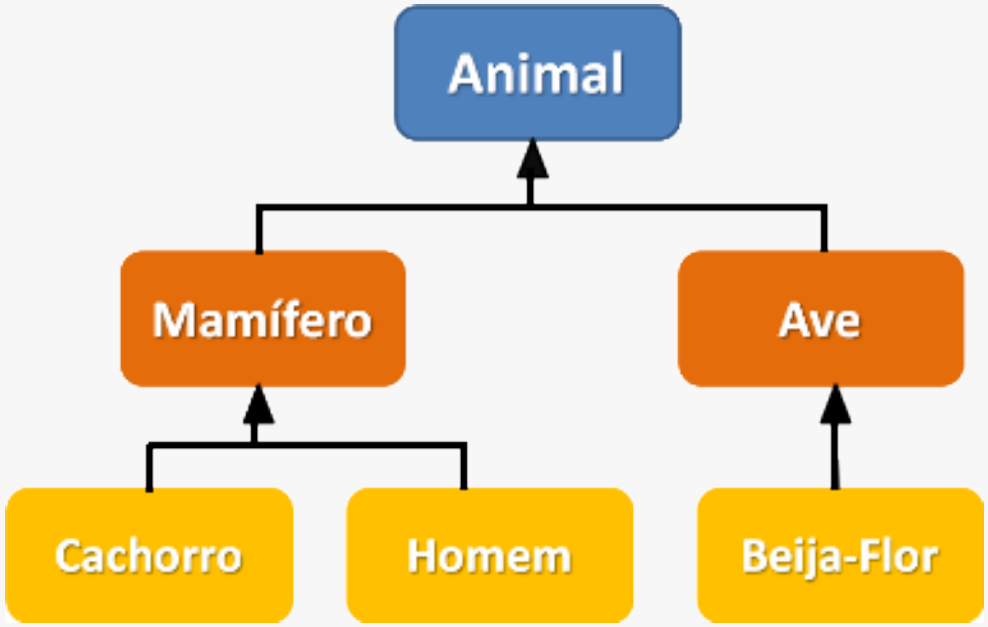


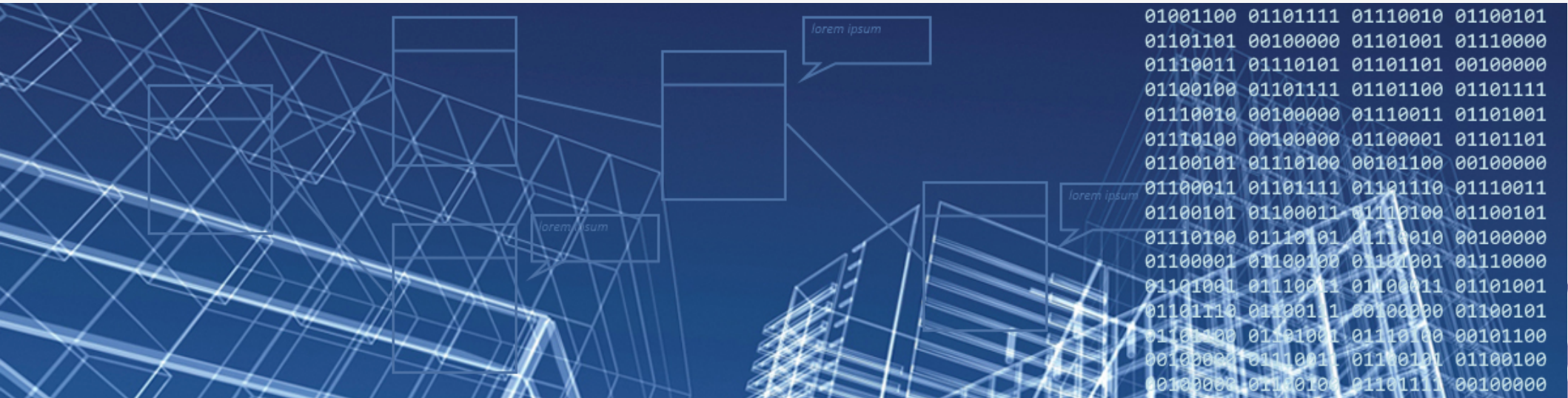
Orientação a Objetos

Princípios | Generalização (Herança)

A implementação do método "respirar()" já estaria na classe "Animal". Mas, digamos que "Cachorro" tenha um aspecto diferente no método "respirar()". O que seria feito? O método "respirar()" seria implementado na classe "Cachorro" com as especificidades necessárias. Essa técnica é conhecida como sobrescrita. Resumindo:

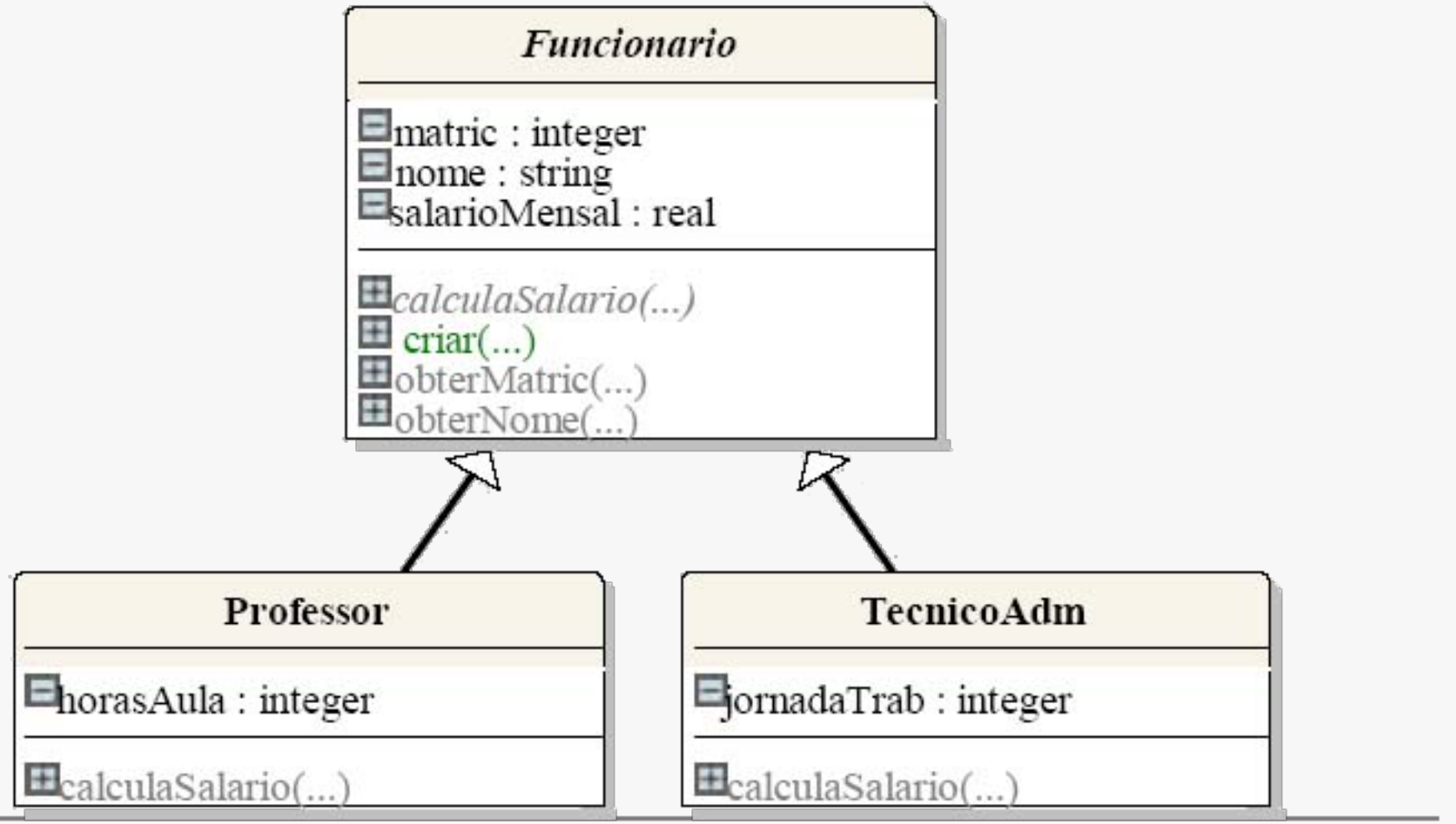
Sobrecarga: métodos com mesmo nome na mesma classe.
Sobrescrita: métodos com mesmo nome em classes descendentes.





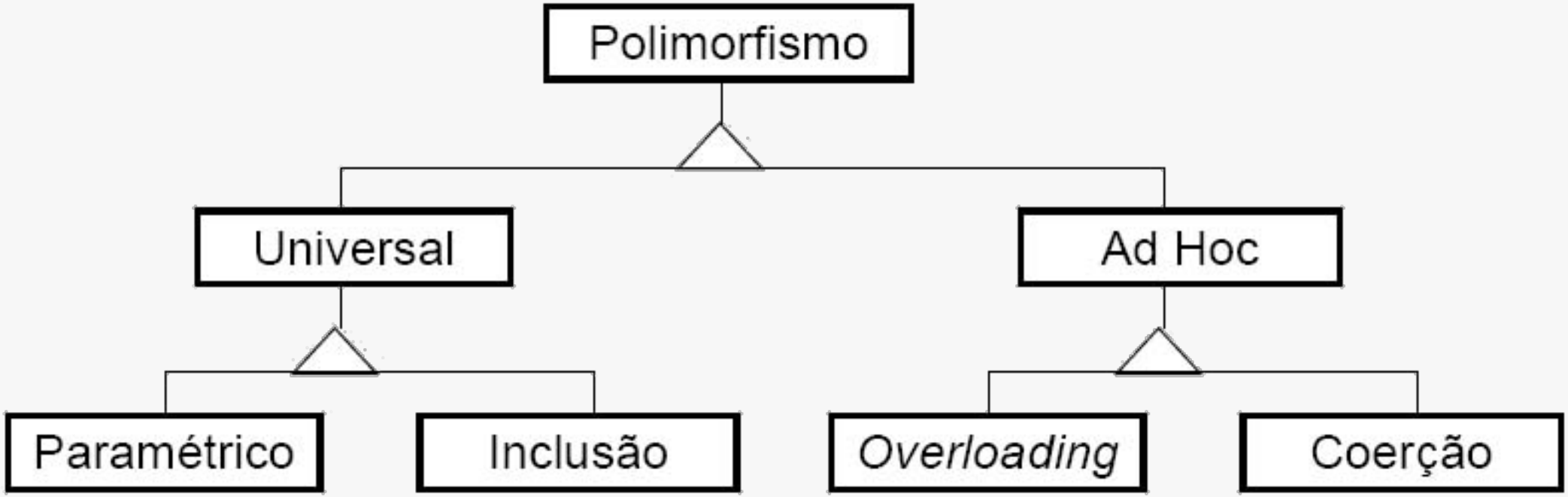
Orientação a Objetos

Princípios | Generalização (Herança)



Princípios | Polimorfismo

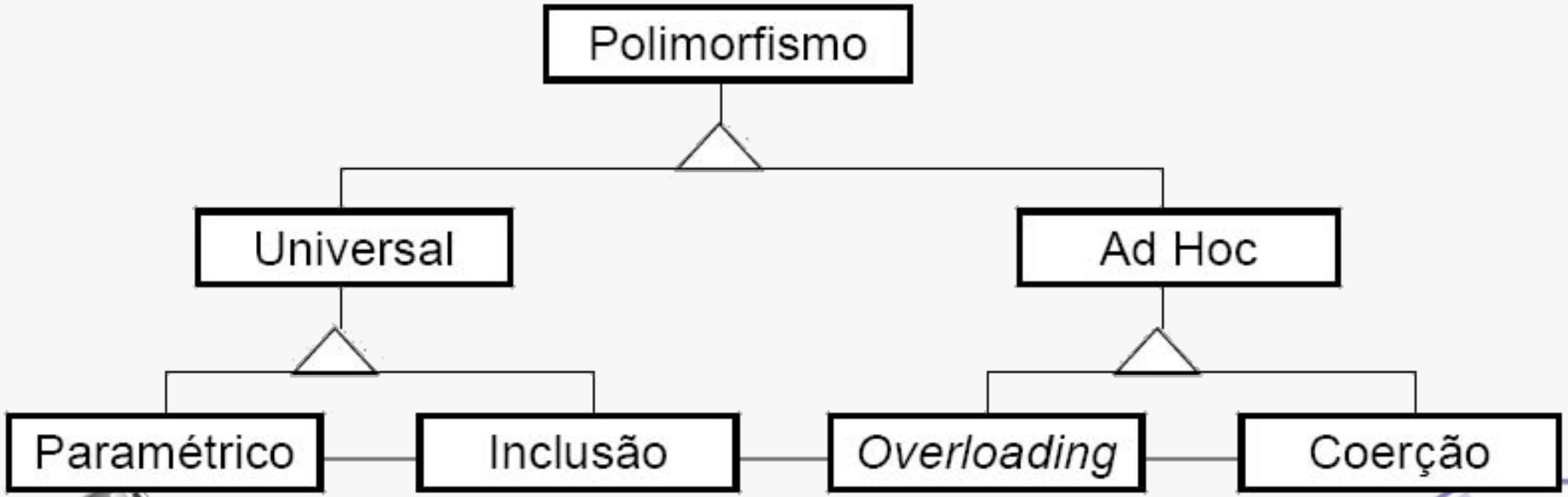
O assunto Polimorfismo costuma causar estranheza e até mesmo confusão, principalmente para quem vem de uma linguagem estruturada. Portanto, retomo o tema para mais esclarecimentos. Observe a imagem com a classificação de Cardelli & Wegner (1985).



Princípios | Polimorfismo

Coerção: a linguagem de programação tem um mapeamento interno entre tipos. Exemplo: se o operador + é definido para somar dois números reais e um inteiro é passado como parâmetro então o inteiro é "coagido" para real.

Overloading (sobrecarga): permite que um "nome de método" seja usado mais de uma vez com diferentes tipos de parâmetros. O compilador automaticamente chama o método "correto" que será executado. Exemplos: o operador + que pode ter 2 parâmetros inteiros, 2 parâmetros reais, 2 cadeias de caracteres (concatenação), etc.

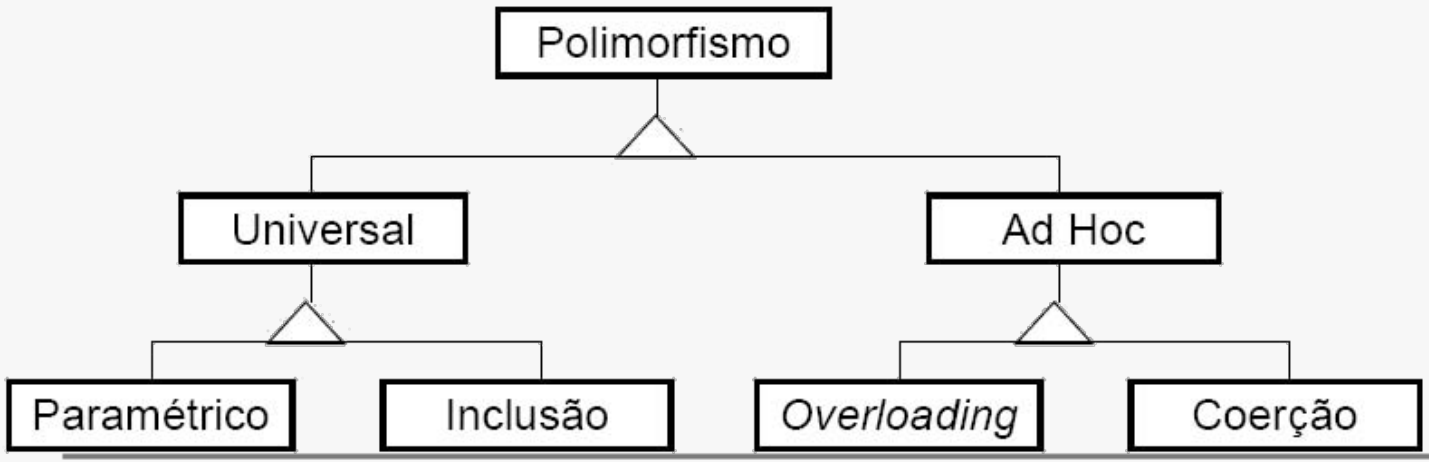


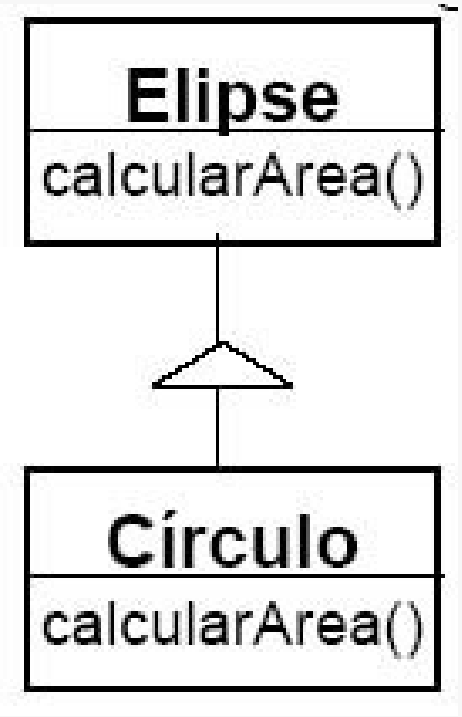
Princípios | Polimorfismo

Polimorfismo de Inclusão: tipo de polimorfismo encontrado em linguagens orientadas a objetos. Todo objeto de uma subclasse pode ser usado no contexto de um superclasse. Exemplo: todo objeto do tipo círculo pode ser usado no lugar de um objeto do tipo elipse.

```
/* Código polimórfico */
void main( ) {
  Elipse e;
  Circulo c;
  imprimir(e);
  imprimir(c);
}

void imprimir(Elipse eli) {.... }
```





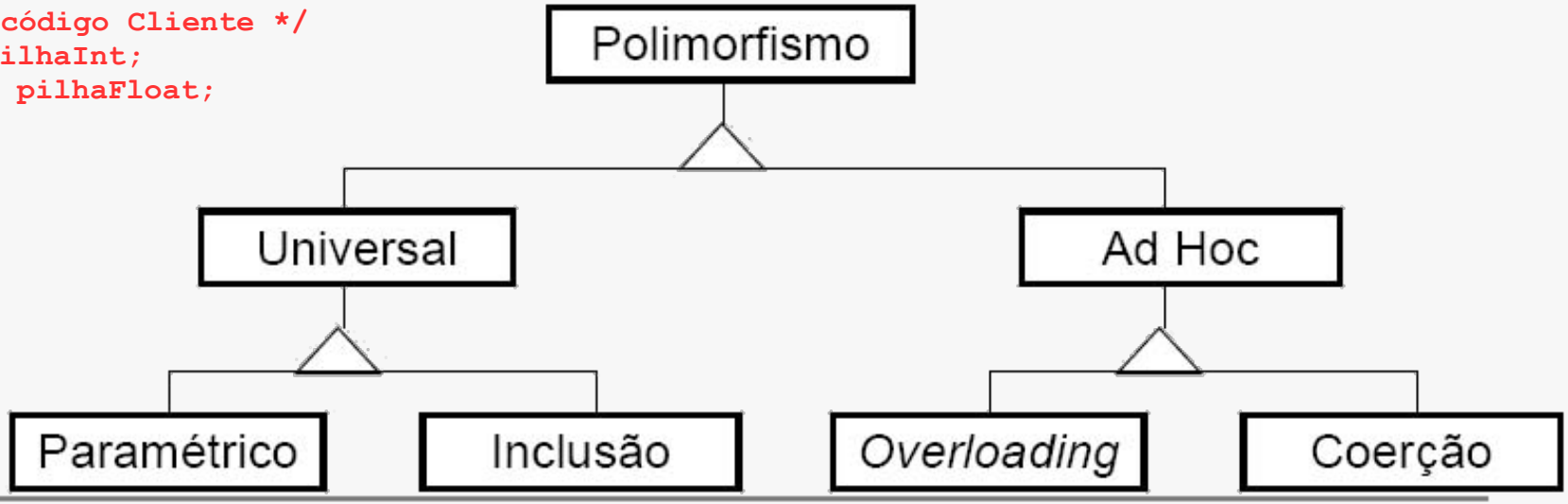
Princípios | Polimorfismo

Polimorfismo paramétrico ou parametrização: a partir de uma única definição de um método, ele pode trabalhar uniformemente.

```
Exemplo C++:

/* Definição da Classe */
template class pilha <Tipo T> {
          /* Estrutura de dados ... */
          void empilhar( T );
          T desempilhar( );
}

/* Utilização no código Cliente */
     pilha<int> pilhaInt;
     pilha<float> pilhaFloat;
```



Orientação a Objetos

Princípios | Agregação (Composição)

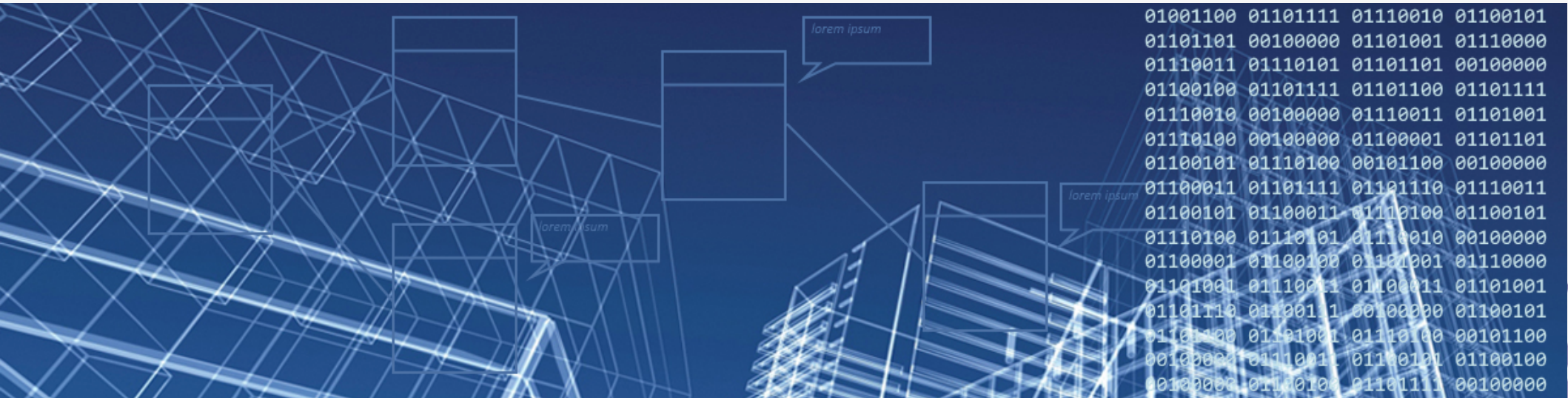
A relação de composição permite que objetos sejam compostos pela agregação de outros objetos ou componentes.

Enquanto a herança é um tipo de relação "É Um", a composição é um tipo de relação "Todo-Parte", sendo que a parte não faz sentido sem o todo.

Sendo assim, um cachorro É UM mamífero. Um mamífero É UM animal. Qual o tipo de relação? Herança.

Agora imagine a classe "Veículo". Caminhão É UM veículo (herança). Ônibus É UM veículo (herança). Pneu É UM veículo? Não. Embreagem É UM veículo? Não.

Agora imagine uma embreagem sozinha. Ela serve para alguma coisa? Não. A embreagem e o pneu são componentes que farão parte de um todo: o veículo, seja ele caminhão ou ônibus.

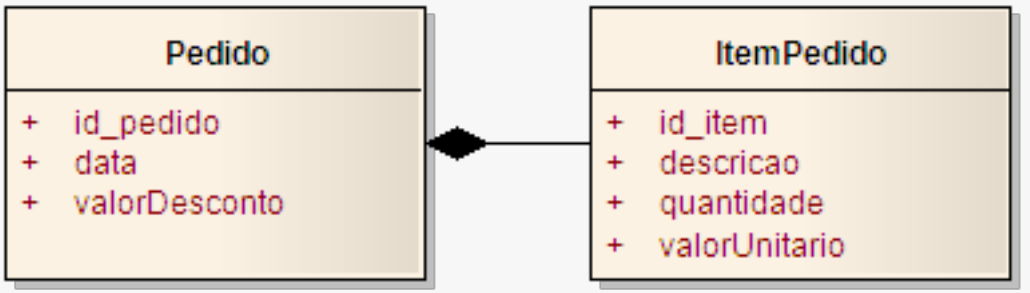


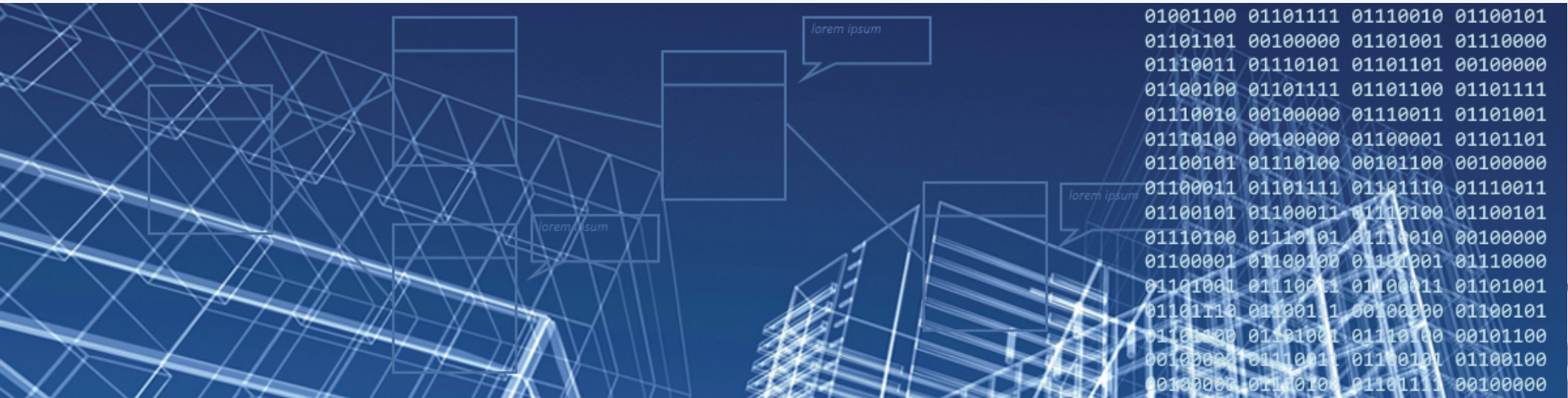
Orientação a Objetos

Princípios | Agregação (Composição)

Para ficar mais claro, vamos citar o exemplo de um pedido, onde temos os dados do pedido em si e os seus itens. Um pedido pode ter vários itens. Mas um item de pedido não pode existir sem um pedido, logo, uma relação de "Todo-Parte".

Você então criaria a classe Pedido e a classe ItemPedido. No momento em que o objeto Pedido fosse instanciado ele teria uma instância da classe ItemPedido. Para ser mais preciso, seria instanciada uma Lista de ItemPedido no objeto Pedido.





Modelagem de Sistemas

UML

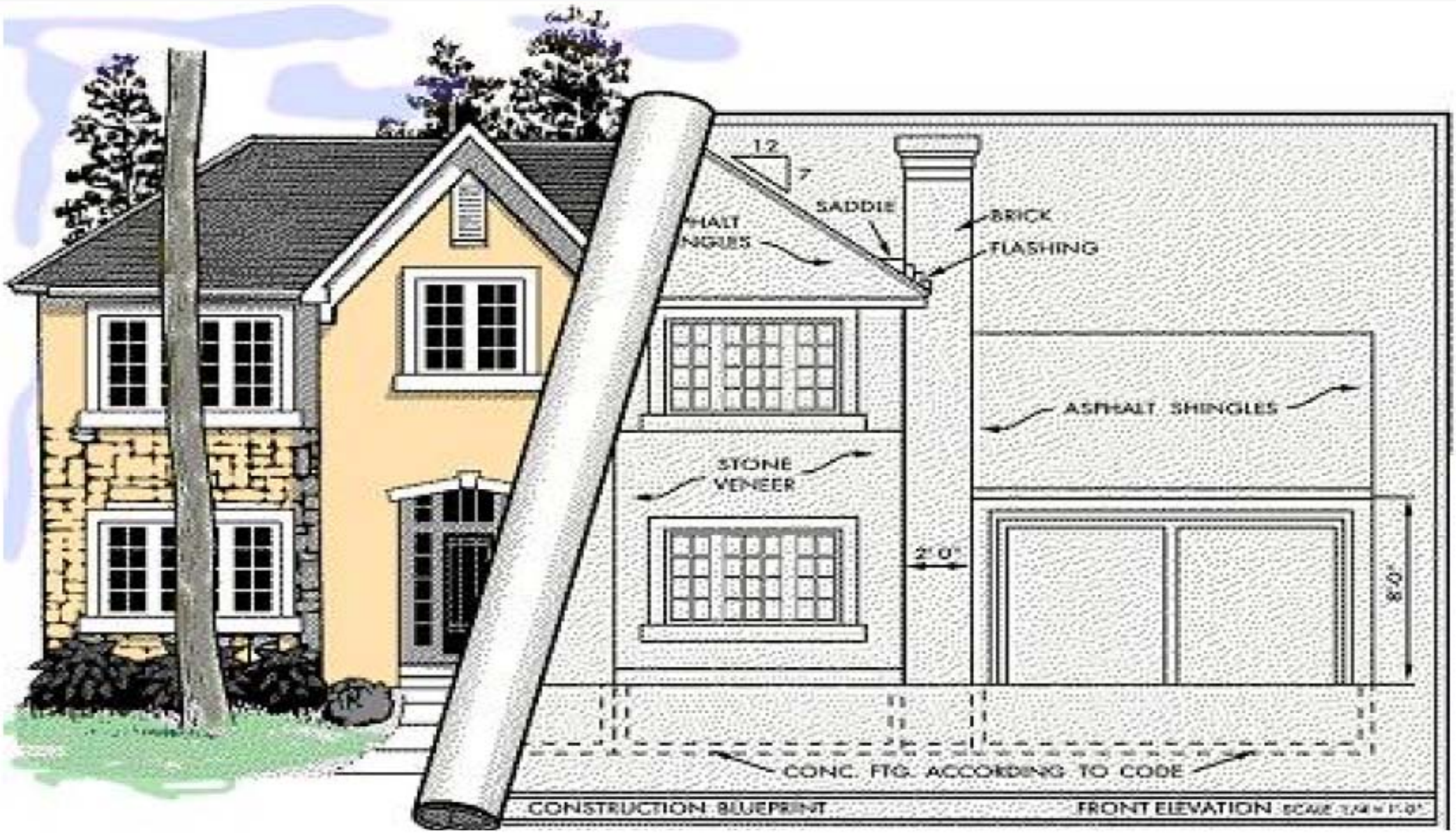
E depois de ter a visão geral sobre o paradigma OO, voltamos ao tema: Modelagem de Sistemas. Nesse ínterim, chegamos na UML. O que é a UML? Deixemos que a Wikipédia nos responda:

Na área de Engenharia de Software, a Linguagem de Modelagem Unificada (do inglês, UML - Unified Modeling Language) é uma linguagem de modelagem que permite representar um sistema de forma padronizada.

A UML não é uma metodologia de desenvolvimento, o que significa que ela não diz para você o que fazer primeiro e em seguida ou como projetar seu sistema, mas ela lhe auxilia a visualizar seu desenho e a comunicação entre os objetos. Basicamente, a UML permite que desenvolvedores visualizem os produtos de seus trabalhos em diagramas padronizados. Junto com uma notação gráfica, a UML também especifica significados, isto é, semântica. É uma notação independente de processos, embora o RUP (Rational Unified Process) tenha sido especificamente desenvolvido utilizando a UML.

Modelagem de Sistemas

UML



UML – História

Durante a década de 1980, o paradigma da orientação a objetos se libertou dos laboratórios de pesquisa e invadiu o mercado das linguagens de programação visual.

Seguindo o exemplo de Smalltalk e C++, diversas linguagens de programação passaram a incorporar os novos recursos de programação orientada a objetos (OOP).

A maioria delas passaram a incorporar também recursos de interface gráfica (GUI), como foi o caso do Turbo Pascal da Borland, que passou a se chamar Delphi.

Essa revolução ocorrida nas linguagens de programação exigia dos metodólogos uma postura atualizada sobre como deveria ser a Análise e o Projeto Orientados a Objetos (OOAD - Object-oriented analysis and design).

A partir de 1989, intensificou-se a elaboração de propostas metodológicas para OOAD. Os métodos apresentados eram muito similares pois compartilhavam os mesmos conceitos básicos, porém adotavam notações distintas que acabavam confundindo os projetistas.

UML – História

Os primeiros esforços em direção a padronização dos métodos, alavancados por membros da OMG (Object Management Group), foram frustrados pelos frequentes protestos dos metodólogos mais relevantes.

A fusão de métodos iniciou-se em 1994, quando Jim Rumbaugh (autor do método OMT - Object Modeling Technique) deixou a General Eletric para se associar à Grady Booch (autor do método Booch) na Rational Software.

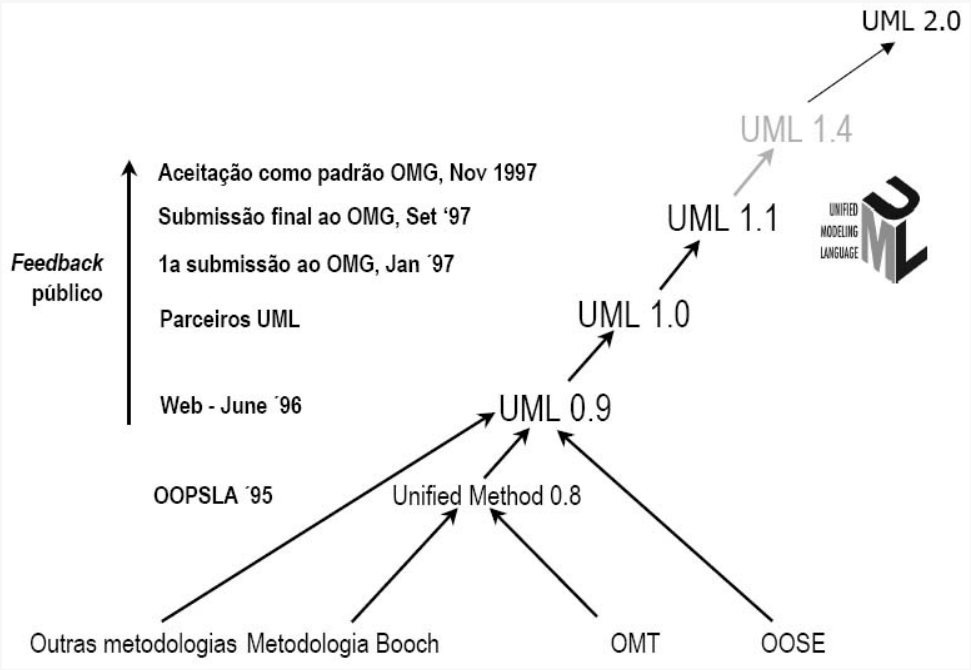
Em 1995, a dupla apresentou a versão 0.8 da documentação sobre o Unified Method (Método Unificado) e anunciou a adesão de mais um membro: Ivar Jacobson (autor do método OOSE - Object-Oriented Software Engineering). Em janeiro de 1997, os "três amigos" lançaram a versão 1.0 da documentação sobre UML como proposta de padronização de métodos a fim de facilitar o intercâmbio entre diferentes modelos. Esta e outras propostas, submetidas à avaliação da OMG, sugeriam um metamodelo oficial e uma notação opcional.

UML – História

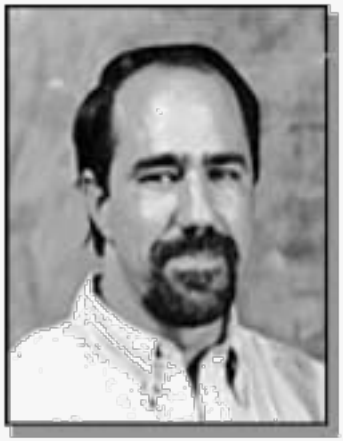
A UML 1.0 havia recebido contribuições de parceiros como Digital Equipment Corp., HP, i-Logix, IntelliCorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI e Unisys.

A versão revisada UML 1.1 foi produzida pela sinergia dos parceiros UML e submetida à apreciação da OMG.

A UML vem evoluindo desde então e já estamos na versão 2.2 que, segundo a OMG, possui 14 tipos de diagramas.



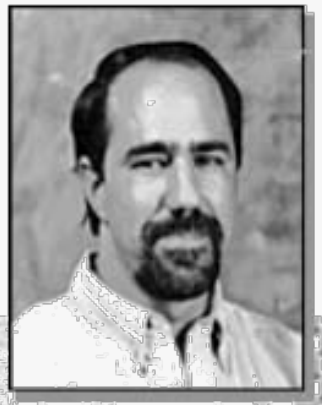
UML – História

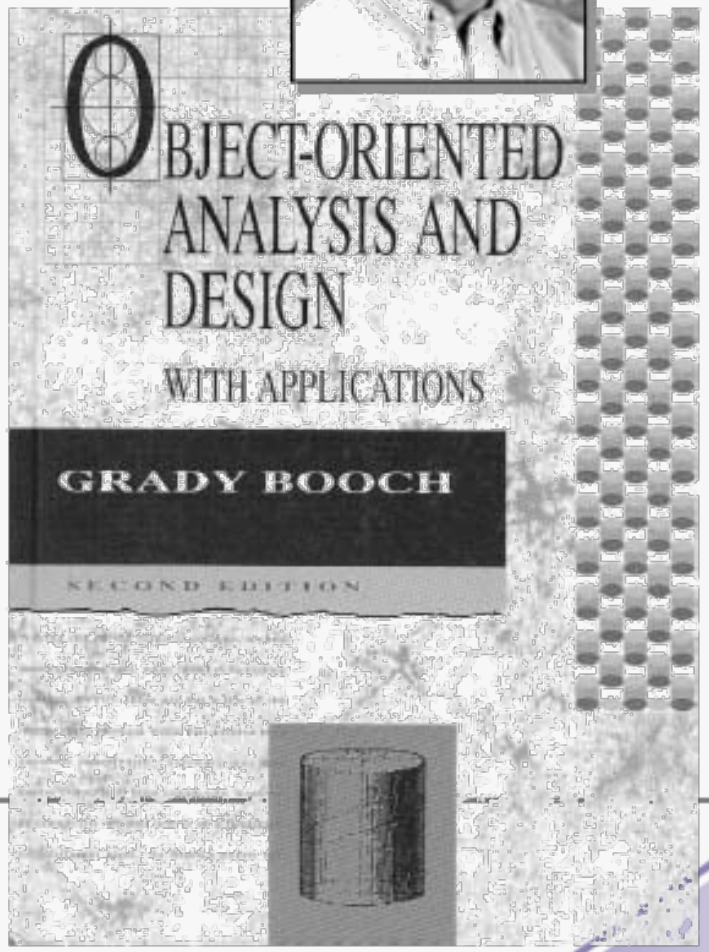


- Grady Booch
  - Um dos pioneiros da OO
  - 1980: ênfase em técnicas de projeto para Ada
  - 1992-1994: livros
    - *Object-Oriented Design with Applications*
      - projeto de programas em C++ e Ada

UML – História

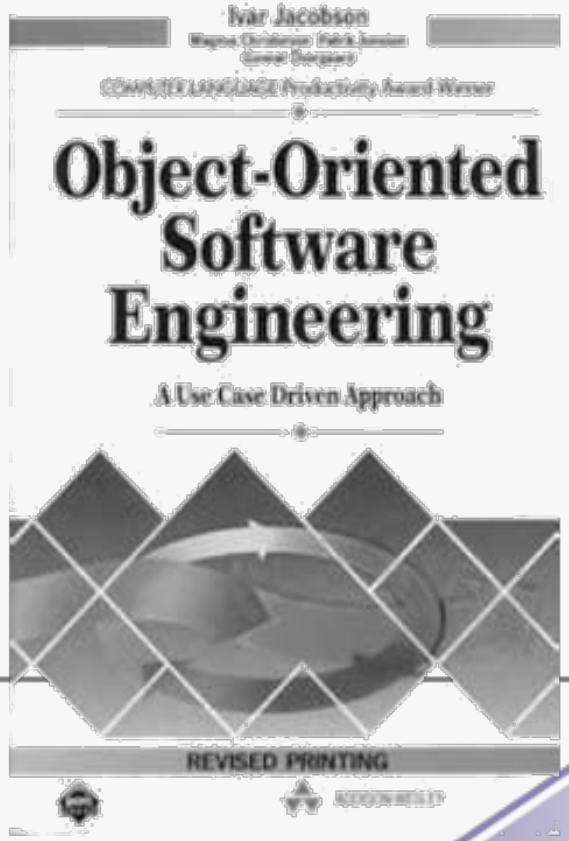
- 1994: *Object-Oriented Analysis and Design with Applications*
  - texto sobre conceitos de OO e modelagem de objetos
  - projeto de várias aplicações-exemplo com diferentes linguagens da época
- 1998: Fundação da Rational





UML – História

- Ivar Jacobson
  - Modelagem OO baseado em Casos de Uso
  - *Objectory*



UML – História

- James Rumbaugh
  - Object Modeling Technique (OMT)
  - Desenvolvida na GE
  - Metodologia baseada em notações

    pré-existentes (ER, DTE, DFD)
  - Clara distinção entre as três visões do problema

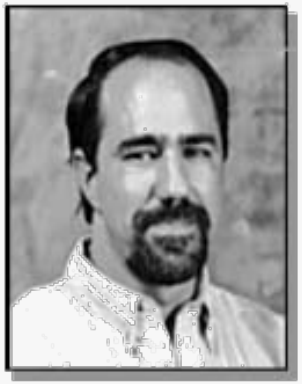
UML – História

- James Rumbaugh

UML – História

RATIONAL SOFTWARE

UML



Booch

Jacobson

OOSE

Rumbaugh

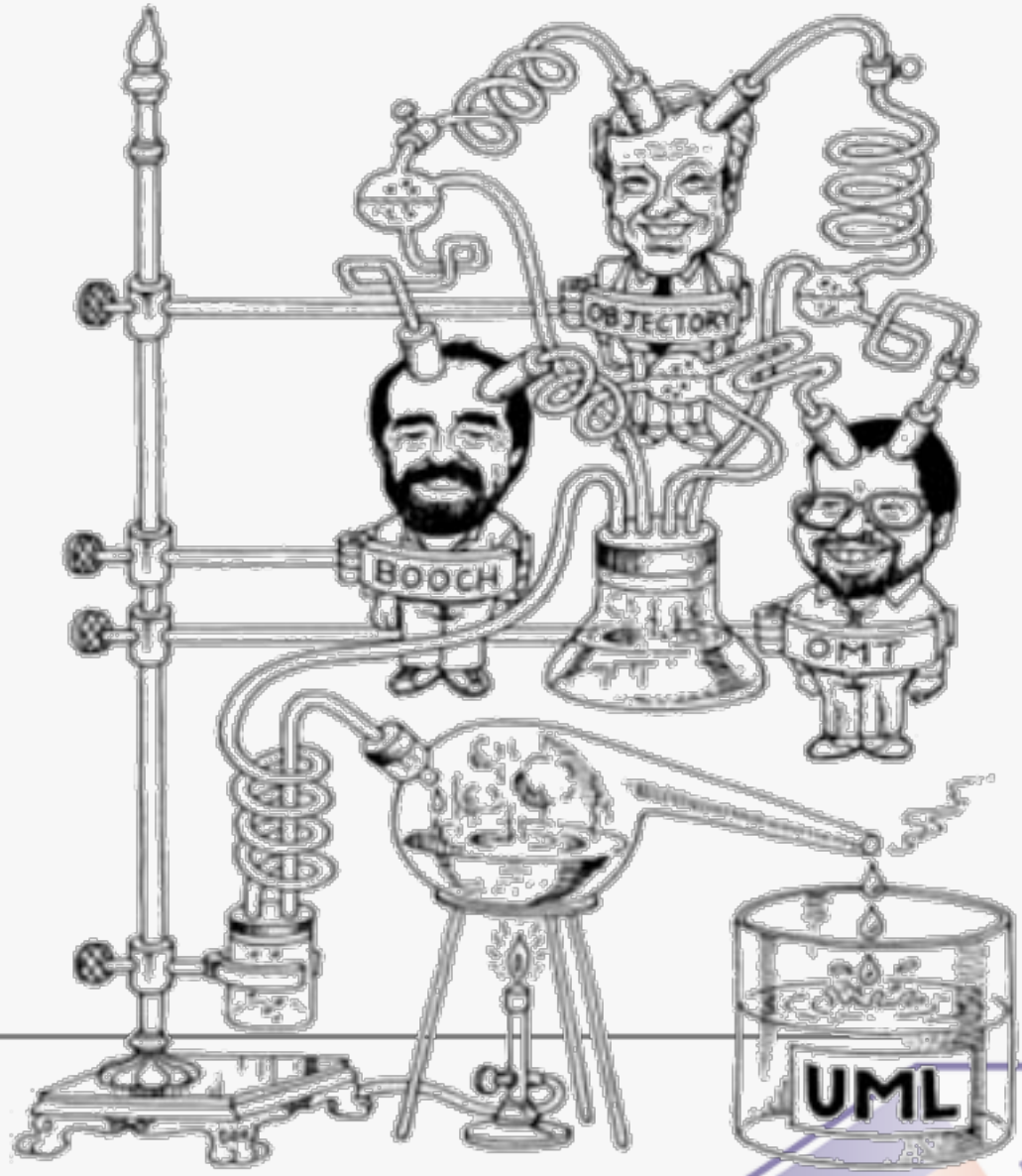
OMT

UML – História – Os Três Amigos

E foi assim, com a união dos chamados "três amigos" que tivemos a formação da UML, que é a padronização das metodologias de desenvolvimento de sistemas baseados na orientação a objetos. O desenvolvimento de um sistema em UML divide-se em 5 fases: análise de requisitos, análise, design (projeto), implementação (programação) e testes.

A UML se propõe a ser a linguagem definitiva para modelagem de sistemas orientados a objetos por ser unificada e facilitar que grupos de desenvolvimento de software interpretem de uma maneira correta e sem ambiguidades modelos gerados por outros analistas ou grupos de desenvolvimento.

É possível criar um sistema utilizando todos os conceitos da UML, adotando o RUP, por exemplo. Também é possível utilizar alguns artefatos da UML, em detrimento de outros, de acordo com a necessidade e filosofia de desenvolvimento da organização.



Modelagem de Sistemas

UML – Fases de Desenvolvimento

Análise de Requisitos: Fase que captura as intenções e necessidades dos usuários do sistema, através das funções desejadas no sistema, chamadas de Casos de Uso.

Análise: Onde se criam as primeiras abstrações e mecanismos presentes no domínio do problema.

Design (Projeto): O resultado da análise é expandido em soluções técnicas. As classes do domínio do problema são mescladas com classes de infraestrutura. É o detalhamento para a fase de programação.

Implementação (Programação): Os modelos criados são convertidos em códigos de linguagem.

Testes: Testes unitários, testes de integração e testes de aceitação.

UNIFIED MODELING LANGUAGE™

Modelagem de Sistemas

UML – Fases de Desenvolvimento | Análise de Requisitos

Esta fase captura as intenções e necessidades dos usuários do sistema a ser desenvolvido através do uso de funções chamadas "use-cases" (casos de uso). Através do desenvolvimento de "use-case", as entidades externas ao sistema (em UML chamados de "atores externos") que interagem e possuem interesse no sistema são modelados entre as funções que eles requerem.

Os atores externos e os "use-cases" (UC) são modelados com relacionamentos que possuem comunicação associativa entre eles ou são desmembrados em hierarquia.

Cada "use-case" modelado é descrito através de um texto, e este especifica os requerimentos do ator externo que utilizará este "use-case".

O diagrama de "use-cases" mostrará o que os atores externos, ou seja, os usuários do futuro sistema, deverão esperar do aplicativo, conhecendo toda sua funcionalidade sem importar como esta será implementada. A análise de requisitos também pode ser desenvolvida baseada em processos de negócios, e não apenas para sistemas de software.

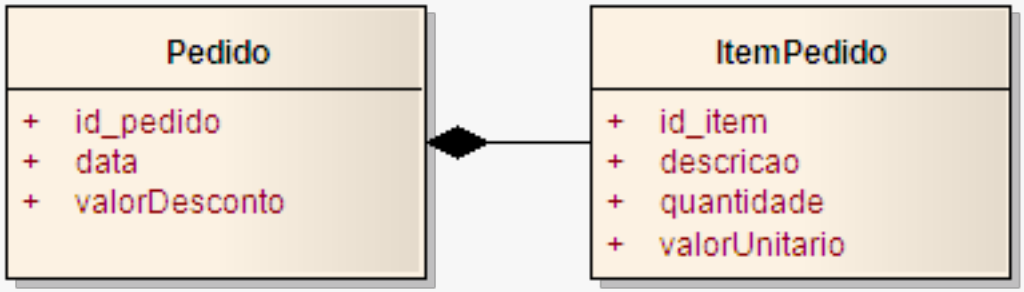
Modelagem de Sistemas

UML – Fases de Desenvolvimento | Análise

A fase de análise está preocupada com as primeiras abstrações (classes e objetos) e mecanismos que estarão presentes no domínio do problema. As classes são modeladas e ligadas através de relacionamentos com outras classes, e são descritas no Diagrama de Classe.

As colaborações entre classes também são mostradas neste diagrama para desenvolver os "use-cases" modelados anteriormente, estas colaborações são criadas através de modelos dinâmicos em UML.

Na análise, só serão modeladas classes que pertençam ao domínio principal do problema do software, ou seja, classes técnicas que gerenciem banco de dados, interface, comunicação, concorrência e outros não estarão presentes neste diagrama.

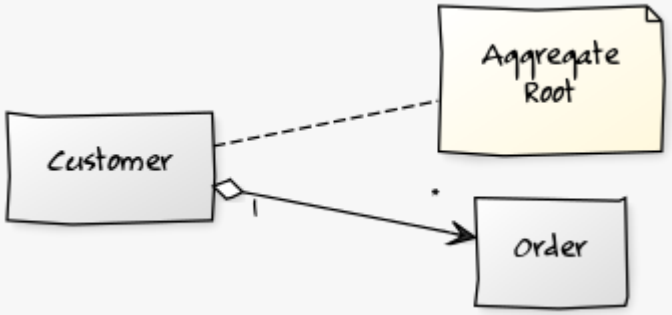


UML – Fases de Desenvolvimento | Design (Projeto)

Na fase de projeto, o resultado da análise é expandido em soluções técnicas. Novas classes serão adicionadas para prover uma infraestrutura técnica: a interface do usuário e de periféricos, gerenciamento de banco de dados, comunicação com outros sistemas, dentre outros.

As classes do domínio do problema modeladas na fase de análise são mescladas nessa nova infraestrutura técnica tornando possível alterar tanto o domínio do problema quanto a infraestrutura.

O projeto resulta no detalhamento das especificações para a fase de programação do sistema.



Modelagem de Sistemas

UML – Fases de Desenvolvimento | Programação

Na fase de programação, as classes provenientes do design são convertidas para o código da linguagem orientada a objetos escolhida (a utilização de linguagens procedurais é extremamente não recomendada).

Dependendo da capacidade da linguagem usada, essa conversão pode ser uma tarefa fácil ou muito complicada. No momento da criação de modelos de análise e design em UML, é melhor evitar traduzi-los mentalmente em código.

Nas fases anteriores, os modelos criados são o significado do entendimento e da estrutura do sistema.

A programação é uma fase separada e distinta onde os modelos criados são convertidos em código.

Modelagem de Sistemas

UML – Fases de Desenvolvimento | Testes

Um sistema normalmente é rodado em testes de unidade, integração e aceitação.

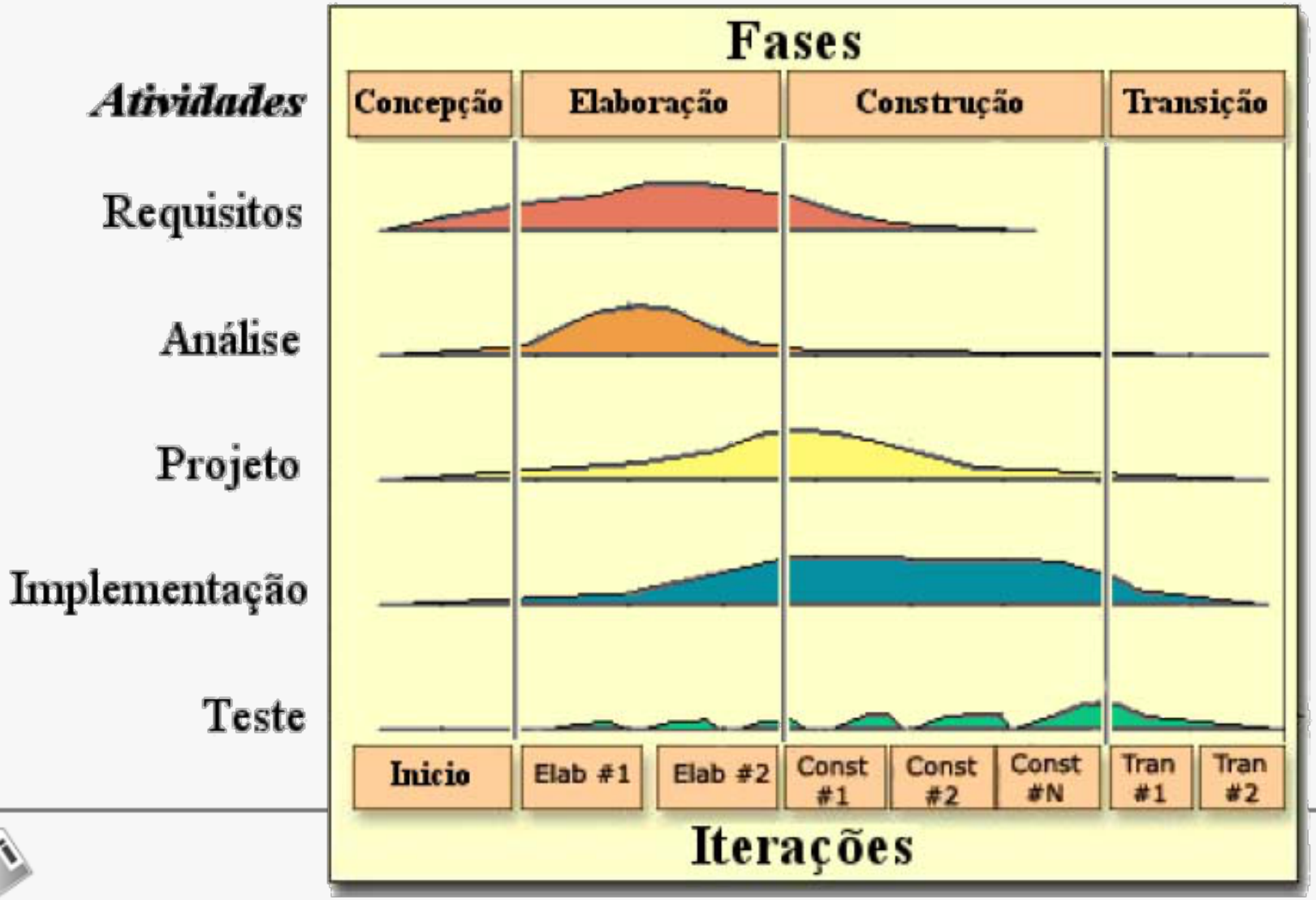
Os testes de unidade são para classes individuais ou grupos de classes e são geralmente testados pelo programador.

Os testes de integração são aplicados já usando as classes e componentes integrados para se confirmar se as classes estão cooperando umas com as outras como especificado nos modelos.

Os testes de aceitação observam o sistema como uma "caixa preta" e verificam se o sistema está funcionando como o especificado nos primeiros diagramas de "use-cases".

O sistema será testado pelo usuário final e será verificado se os resultados mostrados estão realmente de acordo com as intenções pretendidas.

UML – Fases de Desenvolvimento



Modelagem de Sistemas

UML – Notação

A UML é dividida em algumas partes, como segue:

Visões: mostram os diferentes aspectos do sistema, dando enfoque a ângulos e níveis de abstrações diferentes, construindo uma visão completa do sistema a ser construído.

Modelos de Elementos: são os conceitos utilizados nos diagramas. Representam definições comuns da OO.

Mecanismos Gerais: fornecem os comentários suplementares, informações ou semântica sobre os elementos dos modelos.

Diagramas: são gráficos que descrevem o conteúdo em uma visão. A UML possui vários tipos de diagramas que, combinados, formam todas as visões do sistema.

UNIFIED MODELING LANGUAGE™

UML – Visões


Visão de
PROJETO
Visão de
IMPLEMENTAÇÃO
Visão de
CASO DE USO
Visão de
PROCESSO
Visão de
IMPLANTAÇÃO

Modelagem de Sistemas

UML – Visões

Os sistemas são, geralmente, compostos por diferentes níveis de visões.

Cada visão é descrita por um número de diagramas que contém informações que dão ênfase aos aspectos particulares do sistema.

Os diagramas podem fazer parte de mais de uma visão do sistema.

Vamos observar agora a descrição de cada uma dessas visões e seus respectivos diagramas.

Visão	Descrição

Caso de Uso	• focaliza os comportamentos de um sistema
	• deve ser transparente para todos os envolvidos na construção do sistema
	• diagramas utilizados
	  • diagramas de casos de uso
	  • diagramas de interação
	  • diagramas de atividades
	  • diagramas de gráficos de estado

Modelagem de Sistemas

UML – Visões

Visão	Descrição

Projeto	• focaliza a estrutura do sistema
	• mostra as classes, colaborações e as interfaces do sistema
	• diagramas utilizados:
	  • diagrama de classes
	  • diagramas de objetos
	  • diagramas de interação
	  • diagramas de atividades
	  • diagramas de gráficos de estado

Visão	Descrição

Processo	• focaliza as questões de:
	  • desempenho
	  • escalabilidade
	  • mecanismos de concorrência
	  • mecanismos de sincronização
	• diagramas utilizados:
	  • diagrama de classes
	  • diagramas de objetos
	  • diagramas de interação
	  • diagramas de atividades
	  • diagramas de gráficos de estado

Modelagem de Sistemas

UML – Visões

Visão	Descrição

Implementação	• focaliza os artefatos físicos para efetiva montagem do sistema
	• são abordados os componentes e outros arquivos que servem para montagem do sistema
	• diagramas utilizados:
	  • diagramas de componentes
	  • diagramas de interação
	  • diagramas de atividades
	  • diagramas de gráficos de estado

Visão	Descrição

Implantação	• focaliza os nós que formam a topologia de hardware em que o sistema será executado
	• diagramas utilizados:
	  • diagramas de implantação
	  • diagramas de interação
	  • diagramas de atividades
	  • diagramas de gráficos de estado

Modelagem de Sistemas

UML – Modelos de Elementos

Os elementos da UML são blocos de construção para os modelos dos diagramas e são essenciais para o entendimento da UML.

Cada elemento tem um propósito diferente, diferentes regras e notações.

Alguns elementos podem ser usados em diferentes diagramas.

Existe um grande conjunto de elementos disponíveis na especificação.

Vamos conhecer alguns destes elementos especificados pela UML.

UNIFIED MODELING LANGUAGE™

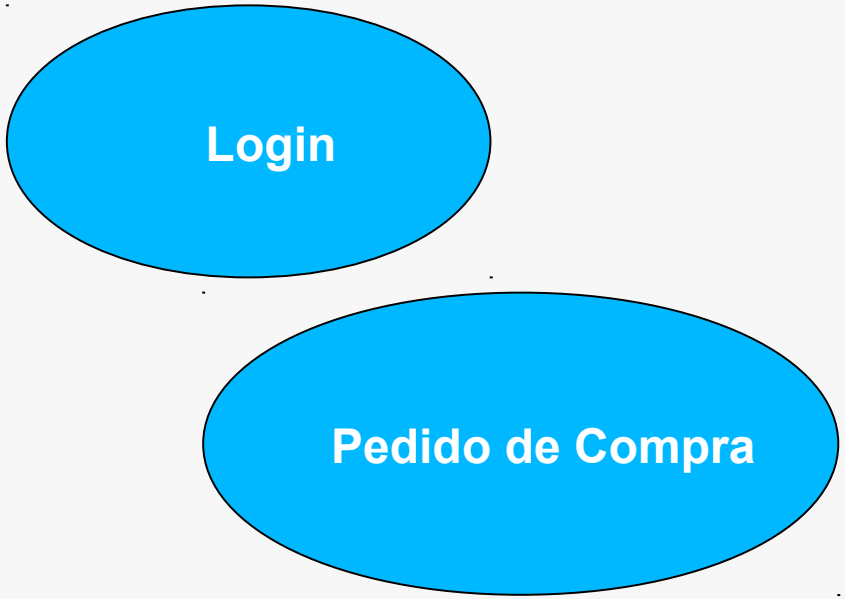
Modelagem de Sistemas

UML – Modelos de Elementos | Caso de Uso

É a descrição de um conjunto de sequências de ações realizadas pelo sistema, que proporciona resultados observáveis de valor para um determinado ator.

Um caso de uso é realizado por uma colaboração.

Graficamente é representado por uma elipse de linhas contínuas, incluindo somente seu nome.



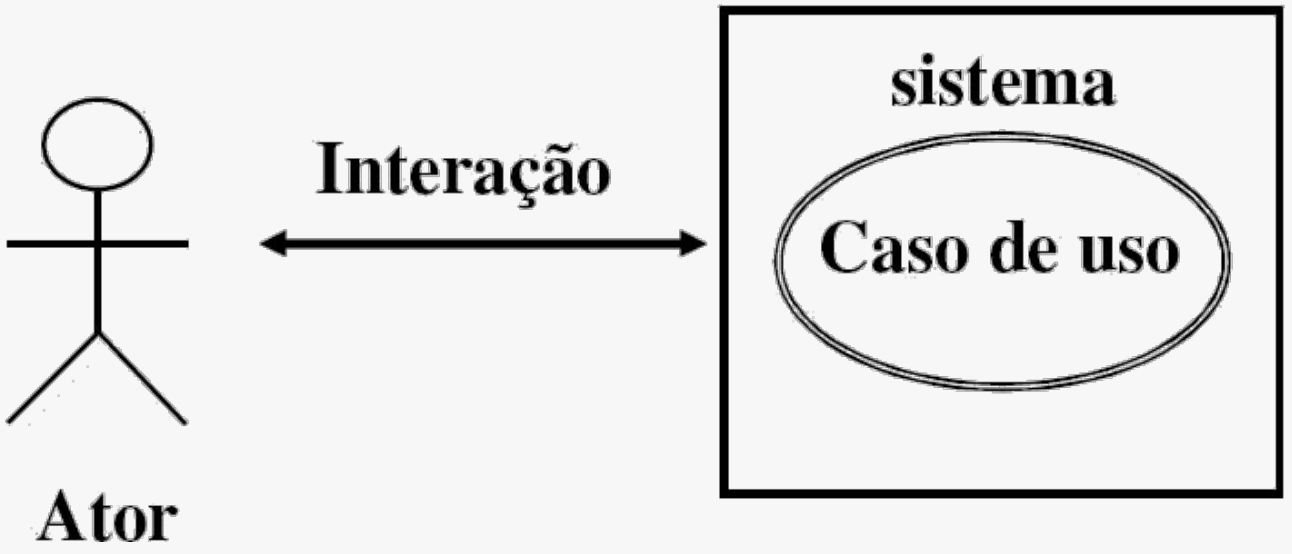
Modelagem de Sistemas

UML – Modelos de Elementos | Ator

O Ator é alguém ou algo externo ao sistema, mas que vai interagir com o sistema.

Pode ser uma pessoa do mundo real ou até mesmo um outro sistema.

Atores são representados como bonecos.

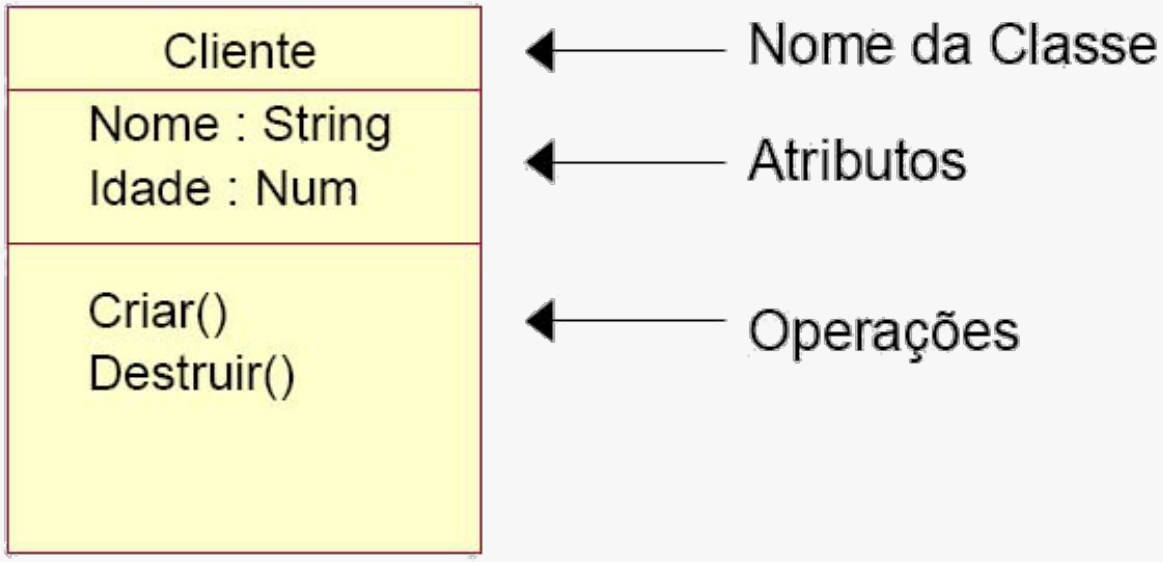


Modelagem de Sistemas

UML – Modelos de Elementos | Classe

É a descrição de conjunto de objetos que compartilham os mesmos atributos e relacionamentos (estado), operações e semântica (comportamento).

As classes podem implementar uma ou mais interfaces e, graficamente, são representadas por retângulos, com três divisões: Nome da Classe, Conjunto de Atributos e Conjunto de Métodos.

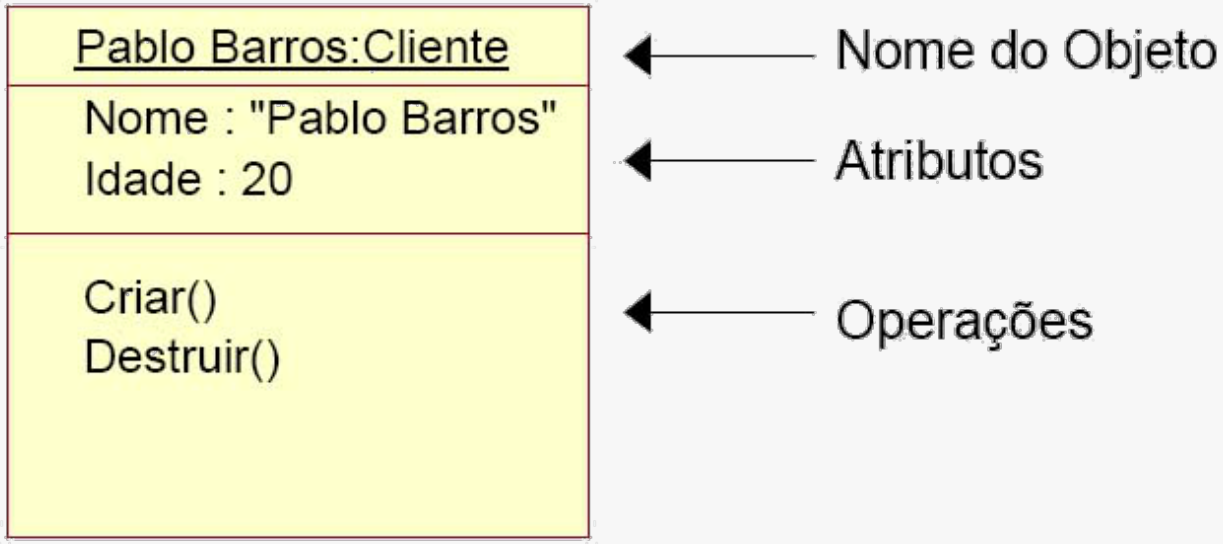


Modelagem de Sistemas

UML – Modelos de Elementos | Objeto

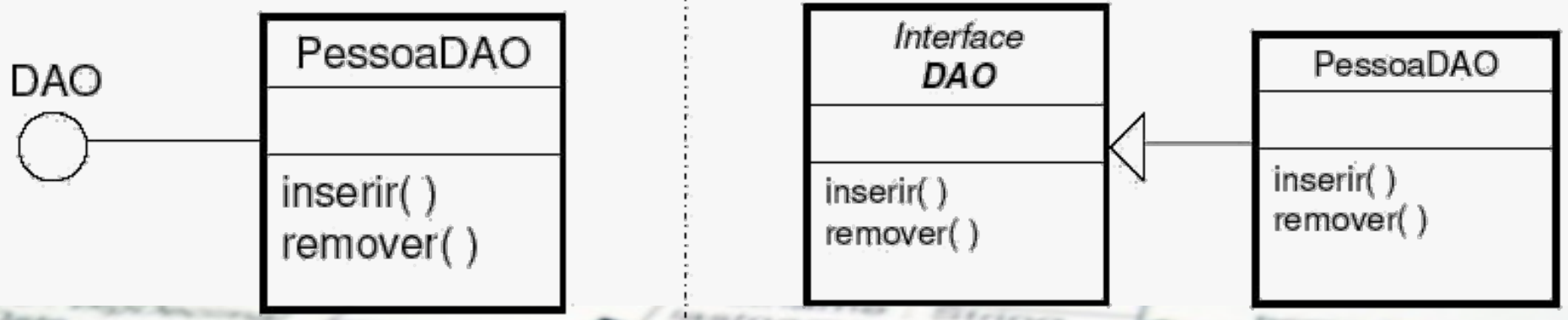
Um objeto é uma instância de uma classe, em tempo de execução.

Os objetos são graficamente representados como uma classe só que seu nome é sublinhado e pode ser mostrado opcionalmente precedido do nome da classe.



UML – Modelos de Elementos | Interface

É um elemento que define uma coleção de operações que especificam serviços de uma classe ou componente. Uma interface representa todo o comportamento externamente visível do elemento. Pode representar todo o comportamento, ou apenas parte dele. A interface define um conjunto de especificações de operações, mas nunca de um conjunto de implementações de operações. É representada graficamente por um círculo e o respectivo nome. Uma interface raramente aparece sozinha.



Modelagem de Sistemas

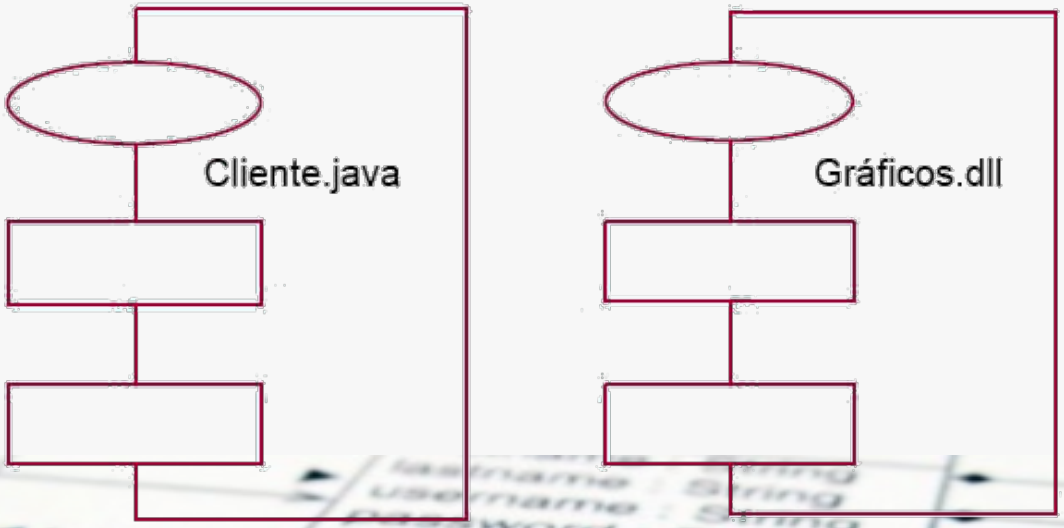
UML – Modelos de Elementos | Estado

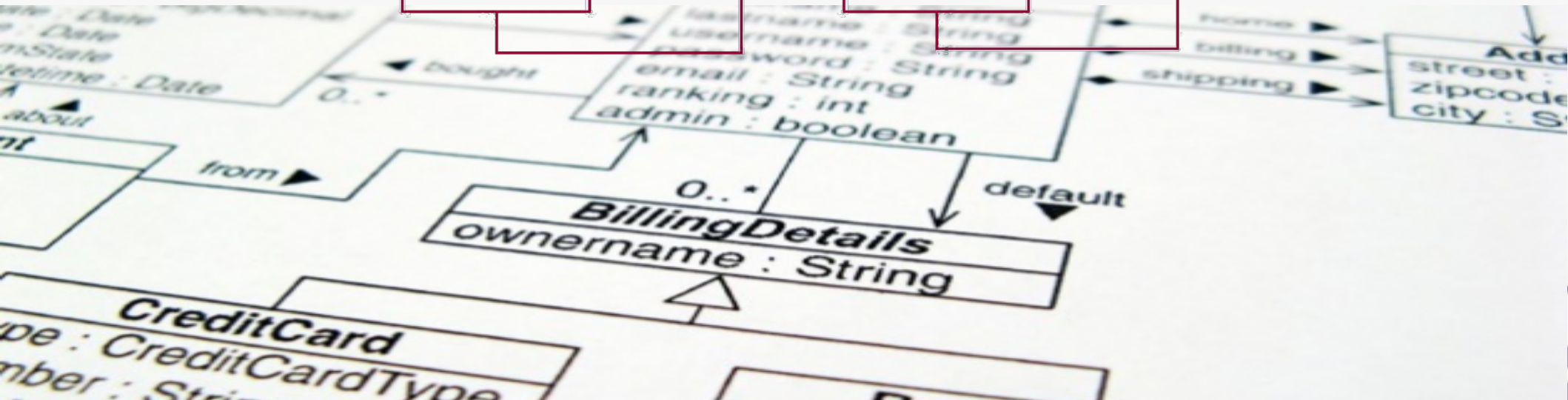
Todo os objetos possuem um estado, que é o resultado das atividades executadas por ele.

O estado é representado por um retângulo com os cantos arredondados e o nome do evento dentro do desenho.

UML – Modelos de Elementos | Componente

É a parte modular de um sistema, cujo comportamento é definido pelas suas interfaces. O trabalho interno dos componentes deve ser invisível, e o seu uso ser independente de plataforma. Graficamente é representado por um retângulo com duas abas na lateral esquerda.

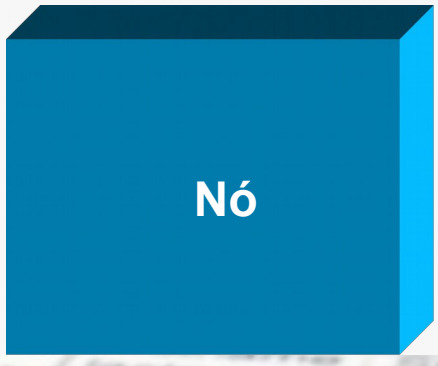




Modelagem de Sistemas

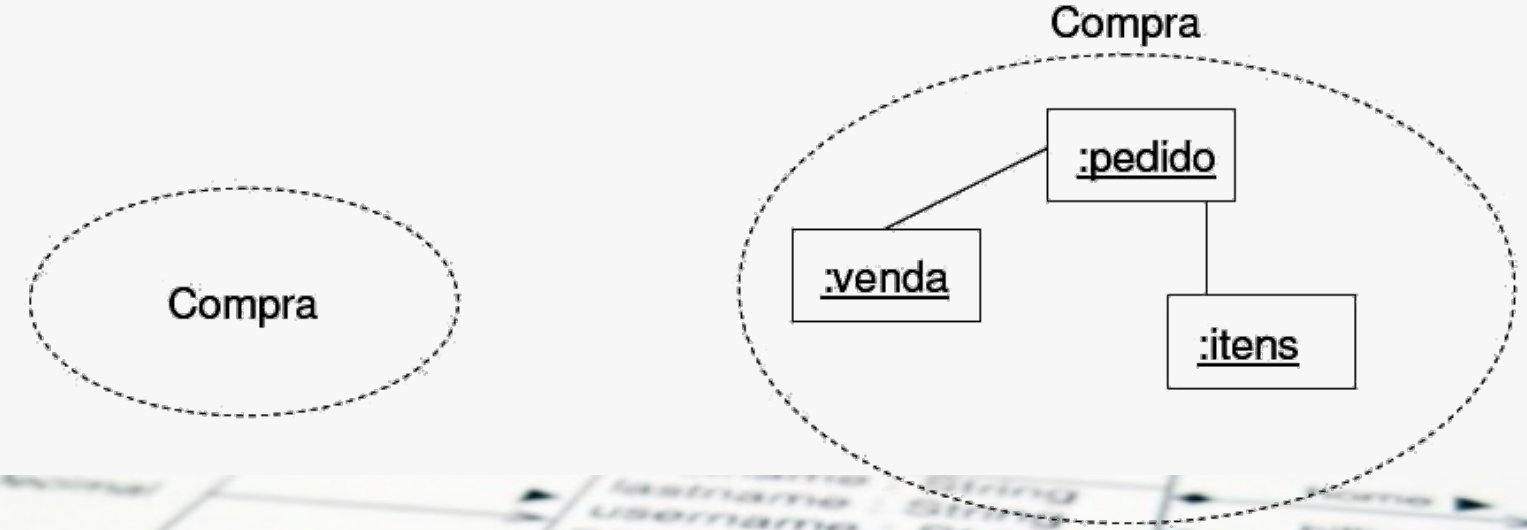
UML – Modelos de Elementos | Nó

Um nó é uma peça física de equipamento, na qual o sistema será disponibilizado, por exemplo uma estação de trabalho ou um servidor. O nó geralmente contém os componentes e outras partes executáveis do código, que podem ser ligados a processos em particular ou espaços de execução. O nós são usados nos diagramas de deployment, para modelar o deploy de um sistema, e para ilustrar a alocação física dos artefatos implementados. Graficamente é representado como um cubo.



UML – Modelos de Elementos | Colaboração

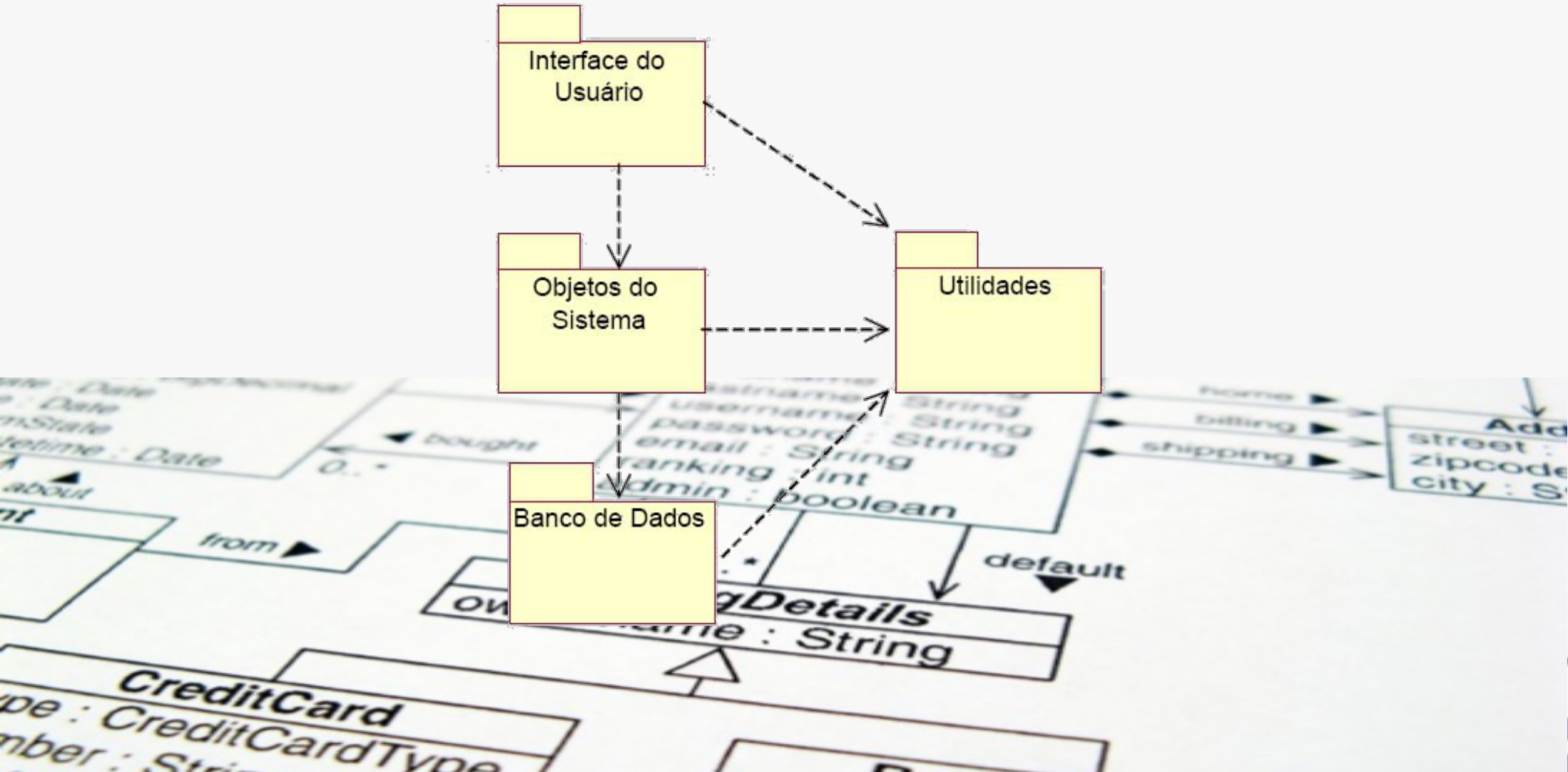
Define as iterações e o comportamento cooperativo, resultado da soma das funções dos elementos. Assim, as colaborações contêm dimensões estruturais e comportamentais. Graficamente são representadas como elipses tracejadas com o seu nome.



Modelagem de Sistemas

UML – Modelos de Elementos | Pacote

É um mecanismo de propósito geral para a organização de elementos em grupo. Graficamente é representado como um retângulo com uma guia.



Modelagem de Sistemas

UML – Modelos de Elementos | Relacionamentos

Os elementos dos modelos UML estão ligados uns aos outros, especificando o que cada elemento significa ao outro e qual o grau de ligação deles, ou seja, qual a relação lógica entre os elementos.

A estas ligações, damos o nome de relacionamento.

Existem diferentes tipos e graus de relacionamentos.

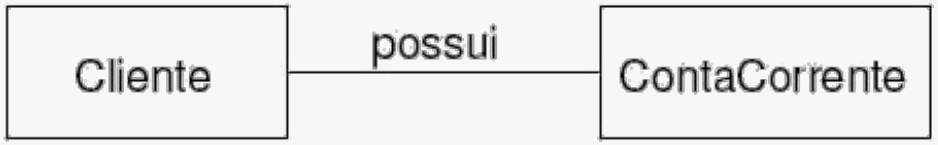
São eles:

* Associação
* Generalização
* Dependência
* Refinamento

UML – Modelos de Elementos | Relacionamentos

Associações

A associação representa uma ligação entre dois elementos. Geralmente são expressas como uma linha sólida, de um elemento ao outro, e com um verbo (ou substantivo) que qualifique a associação.

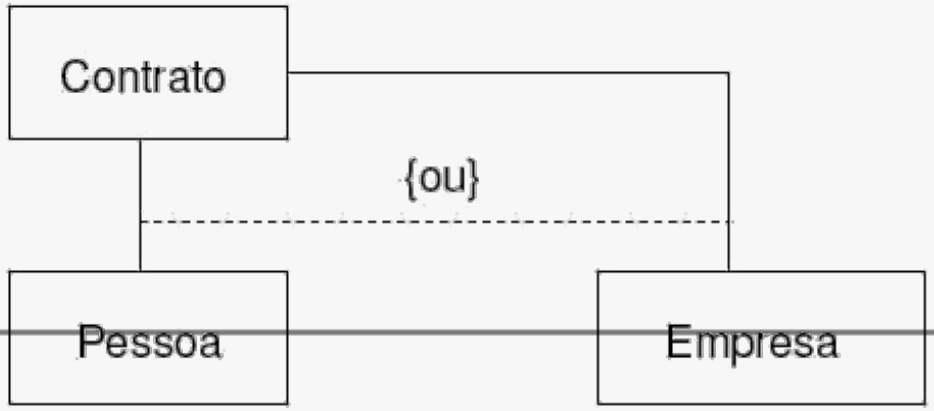


As associações ainda podem expressar a cardinalidade e a navegação (sentido) da associação. A cardinalidade (ou multiplicidade) indica quantos elementos são possíveis de cada lado da associação e pode ser expressada como um número ou um intervalo.

UML – Modelos de Elementos | Relacionamentos

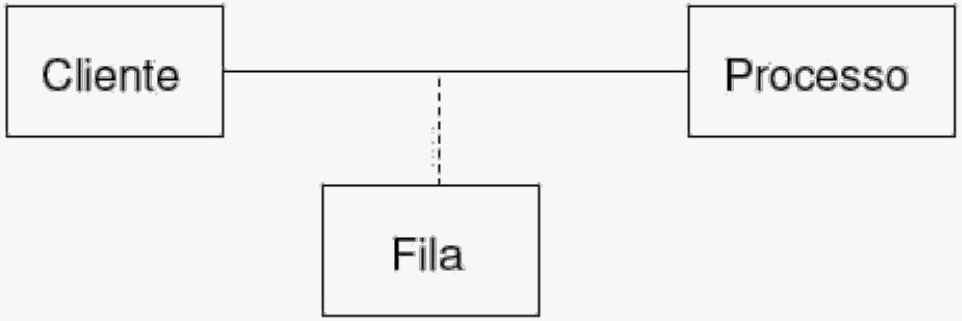
Associação Recursiva: Acontece quando um elemento se conecta a ele mesmo, e associação tem alguma semântica no modelo.

Associação Exclusiva: Quando algumas combinações de associações não são possíveis no domínio do problema. É uma restrição entre duas ou mais associações.

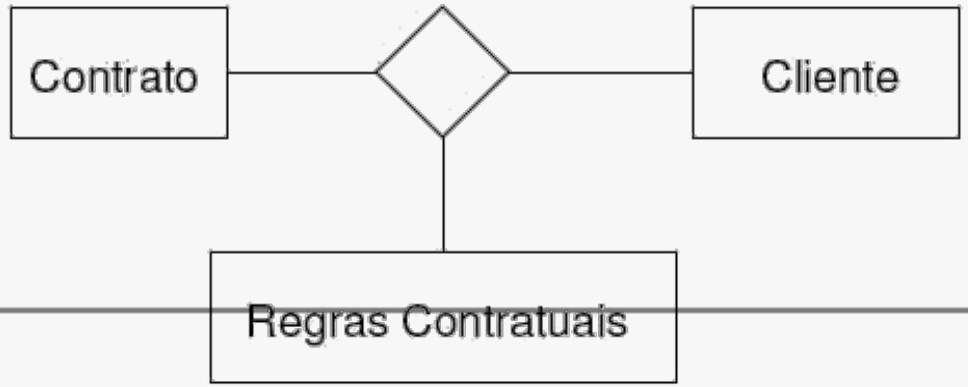


UML – Modelos de Elementos | Relacionamentos

Associação de Classe: Uma classe pode ser associada a uma associação. Serve para adicionar informações extras à associação existente.



Associação Ternária: Usada quando mais de duas classes podem se associar entre si.



UML – Modelos de Elementos | Relacionamentos

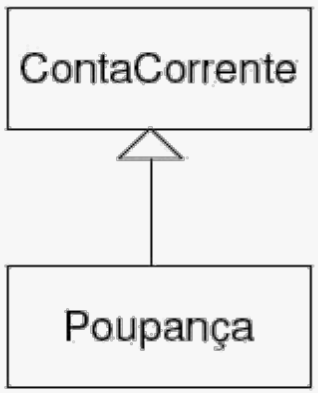
Agregação: Este é um caso particular de associação. Indica que um elemento é parte ou está contida em outra classe. Representa uma relação do tipo parte/todo.

Agregação de Composição: É uma relacionamento onde um elemento está contido em outro, ou seja a vida de um depende do outro, e o seus tempos de vida são os mesmos.

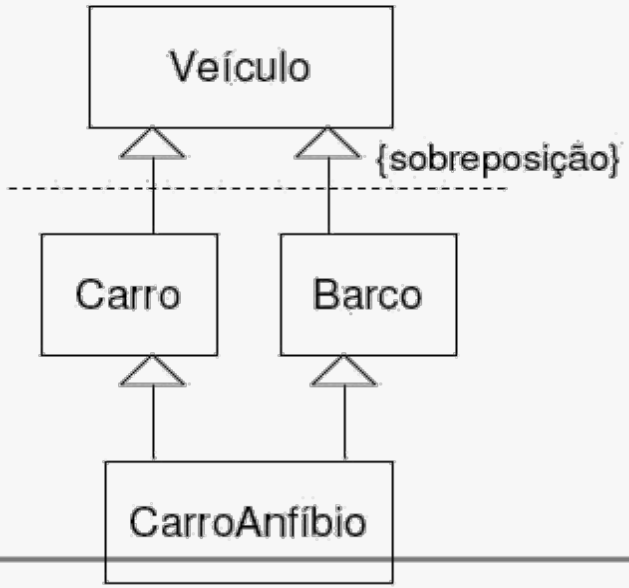
UML – Modelos de Elementos | Relacionamentos

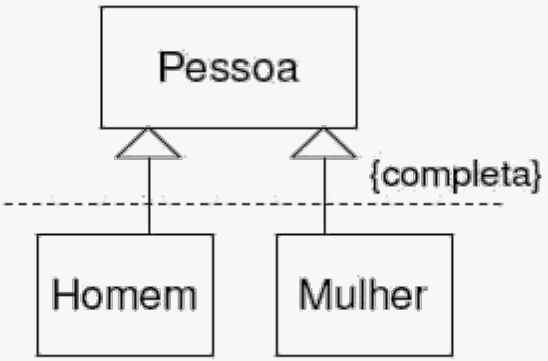
Generalizações

A generalização é um relacionamento entre um elemento mais geral e um mais específico. O elementos mais específico possui todas as características do seu elemento mais geral, como as propriedades e seu comportamento, além de poder adicionar mais características a ele mesmo. As generalizações podem ser normal e restrita. As restritas se dividem em sobreposição, disjuntiva, completa e incompleta.



Normal

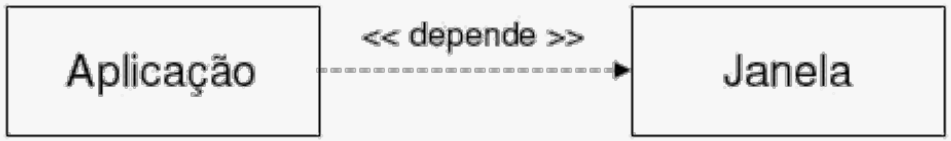




UML – Modelos de Elementos | Relacionamentos

Dependência

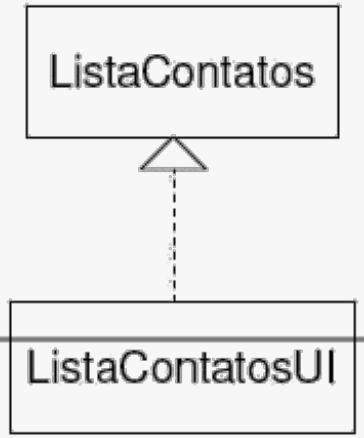
A dependência é uma conexão semântica entre dois elementos, um independente e outro dependente. Qualquer alteração no elemento independente pode afetar o elemento dependente. Em classes, a dependência indica que o elemento apenas instancia e/ou usa o elemento independente, sem manter uma relação duradoura com o elemento.



UML – Modelos de Elementos | Relacionamentos

Refinamento

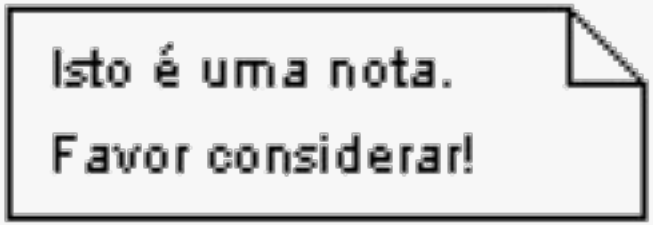
O relacionamento de refinamento ocorre entre dois elementos parecidos, em diferentes níveis de abstração. Um refinamento pode acontecer entre um tipo e uma classe que o realiza, nesse caso é chamado de realização. Refinamento também pode ocorrer entre uma classe de análise e uma classe de design gráfico, ou entre um alto e um baixo nível de descrição. Refinamento também é usado para modelar diagramas de implementações diferentes da mesma coisa, uma simples, e outra mais complexa. O Refinamento é usado no modelo de coordenação. Em grandes projetos, todos os modelos devem ser coordenados.



UML – Mecanismos Gerais | Notas

A nota é apenas um símbolo para representar restrições e comentários anexados a um elemento. Geralmente usa-se a nota para aprimorar os diagramas.

Graficamente é representada por um retângulo com um dos cantos com uma dobra na página.



UML – Mecanismos Gerais | Ornamentos

Ornamentos gráficos são incluídos nos modelos de elementos em diagramas e adicionam semânticas ao elemento. Um exemplo de um ornamento é o da técnica de separar um tipo de uma instância.

Quando um elemento representa um tipo, seu nome é mostrado em negrito. Quando o mesmo elemento representa a instância de um tipo, seu nome é escrito sublinhado e pode significar tanto o nome da instância quanto o nome do tipo.

Outros ornamentos são os de especificação de multiplicidade de relacionamentos, onde a multiplicidade é um número ou um intervalo que indica quantas instâncias de um tipo conectado pode estar envolvido na relação.

UML – Diagramas

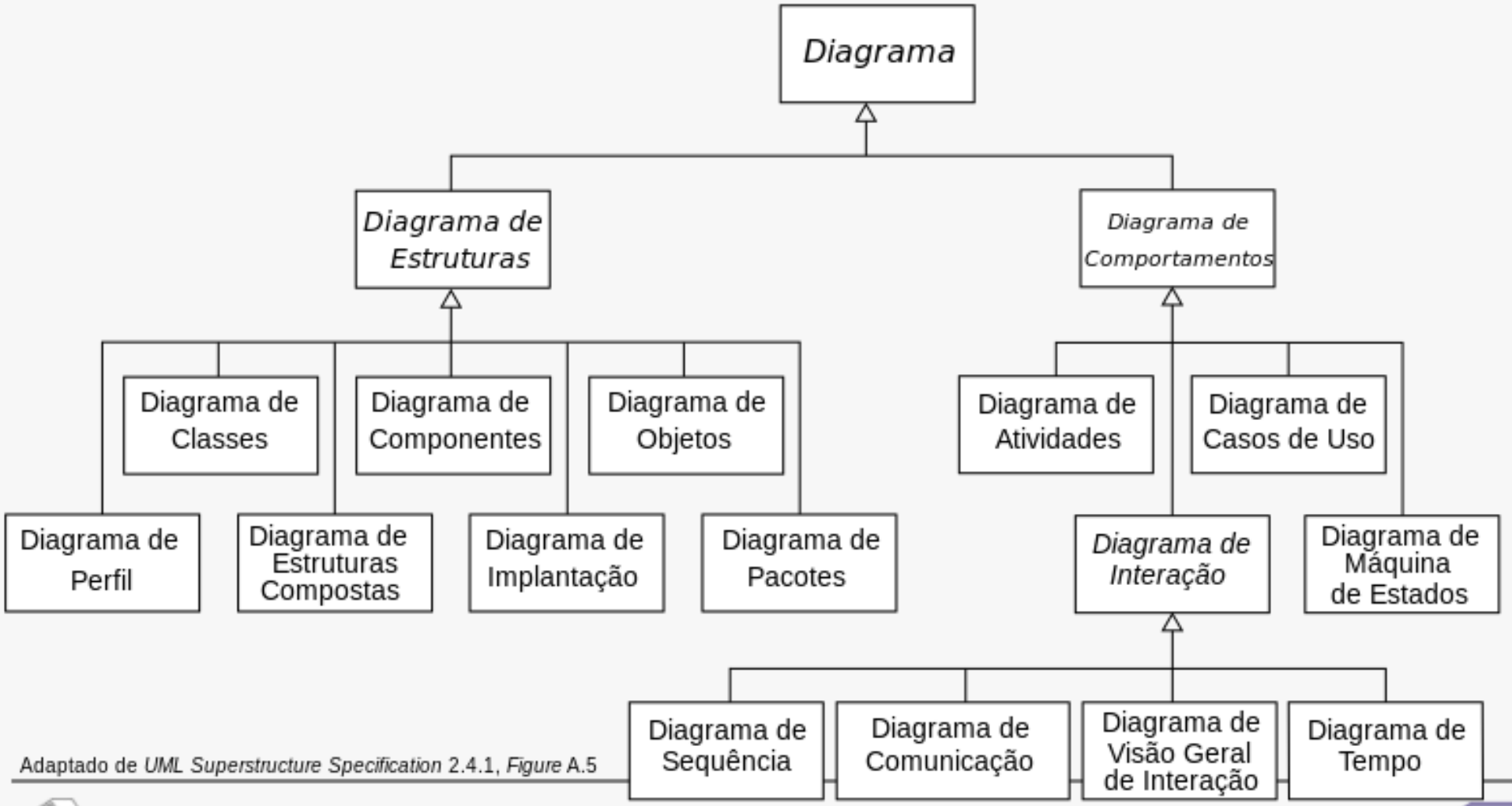
A UML 2.2, conforme a OMG, possui 14 tipos de diagramas, divididos em duas grandes categorias: Estruturais e Comportamentais.

Sete tipos de diagramas representam informações estruturais, e os outros sete representam tipos gerais de comportamento, incluindo quatro em uma subcategoria que representam diferentes aspectos de interação.

Estes diagramas podem ser visualizados de forma hierárquica, como apresentado no padrão de diagrama de classes na página seguinte.

Modelagem de Sistemas

UML – Diagramas



Adaptado de *UML Superstructure Specification* 2.4.1, *Figure* A.5

UML – Diagramas

Para compreender como cada diagrama funciona é preciso estudar a UML. O estudo de cada diagrama depende do grau de interesse de cada leitor. Não adianta ler e conhecer a teoria de cada um deles apenas para dizer que leu. Fica a cargo de cada leitor pesquisar e se aprofundar sobre cada diagrama.

No entanto, conhecermos parte dos diagramas nesse momento, usaremos um estudo de caso simples.

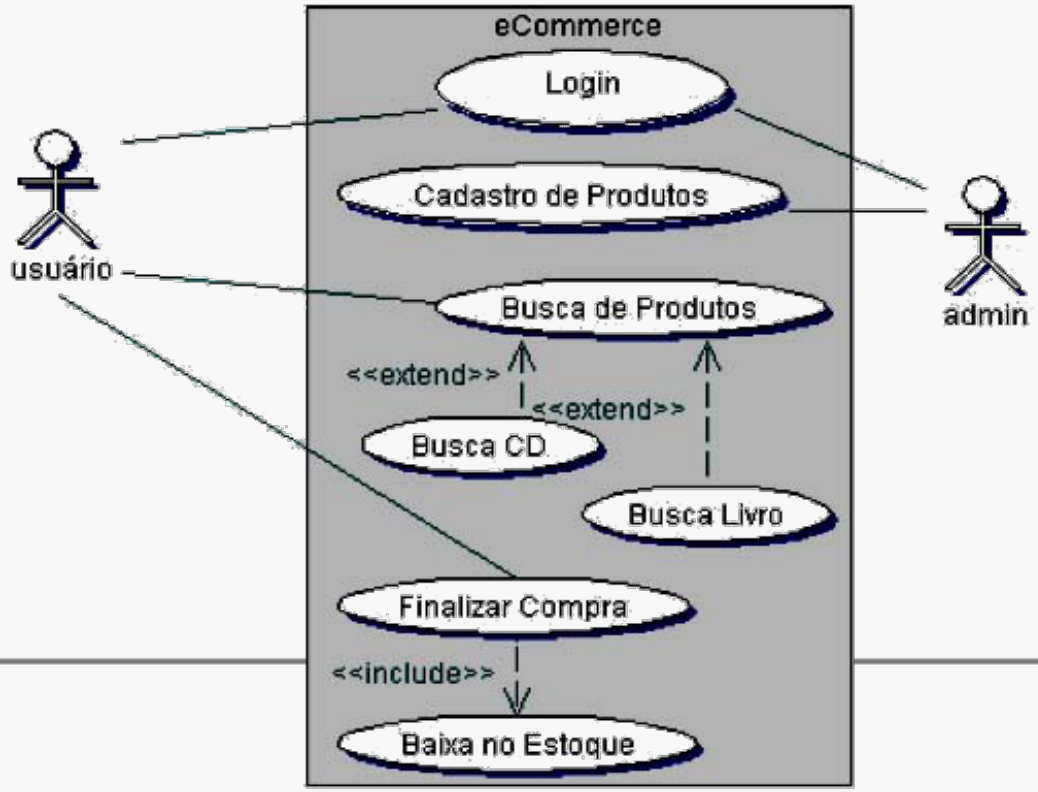
Imagine um loja, que vende seus produtos na internet.

Os usuário poderão fazer o login no site, escolher os produtos que desejam adquirir e consolidar o seu pedido, efetuando a compra.

UML – Diagramas

Diagrama de Casos de Uso

O Diagrama de Casos de Uso serve para visualizar os relacionamentos entre os atores e os casos de uso do sistema (cenários), numa visão geral. Serve para levantar os requisitos funcionais do sistema.



UML – Diagramas

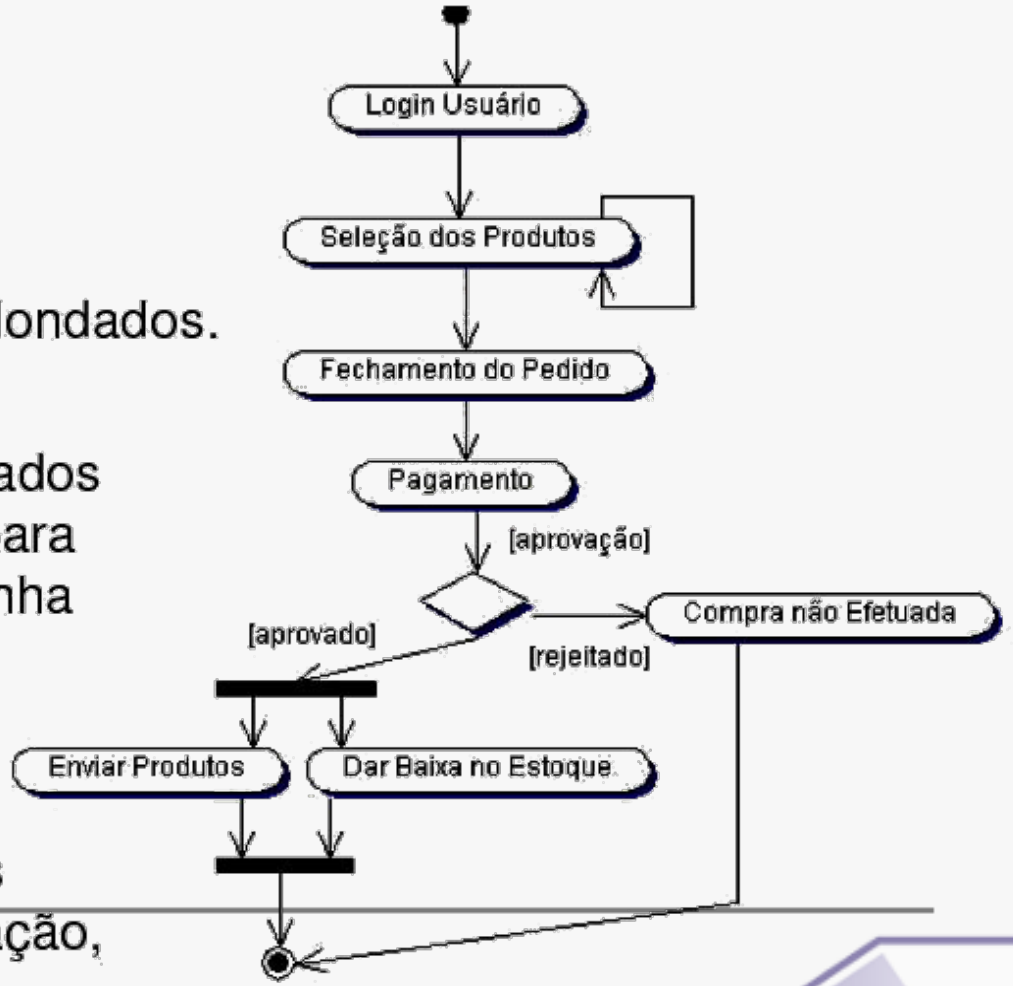
Diagrama de Atividades

O Diagrama de Atividades mostra o fluxo de controle.

As atividades são representadas como retângulos com cantos arredondados.

Tipicamente as atividades são estados de ação – estados que transitam para outro estado, assim que a ação tenha sido completada.

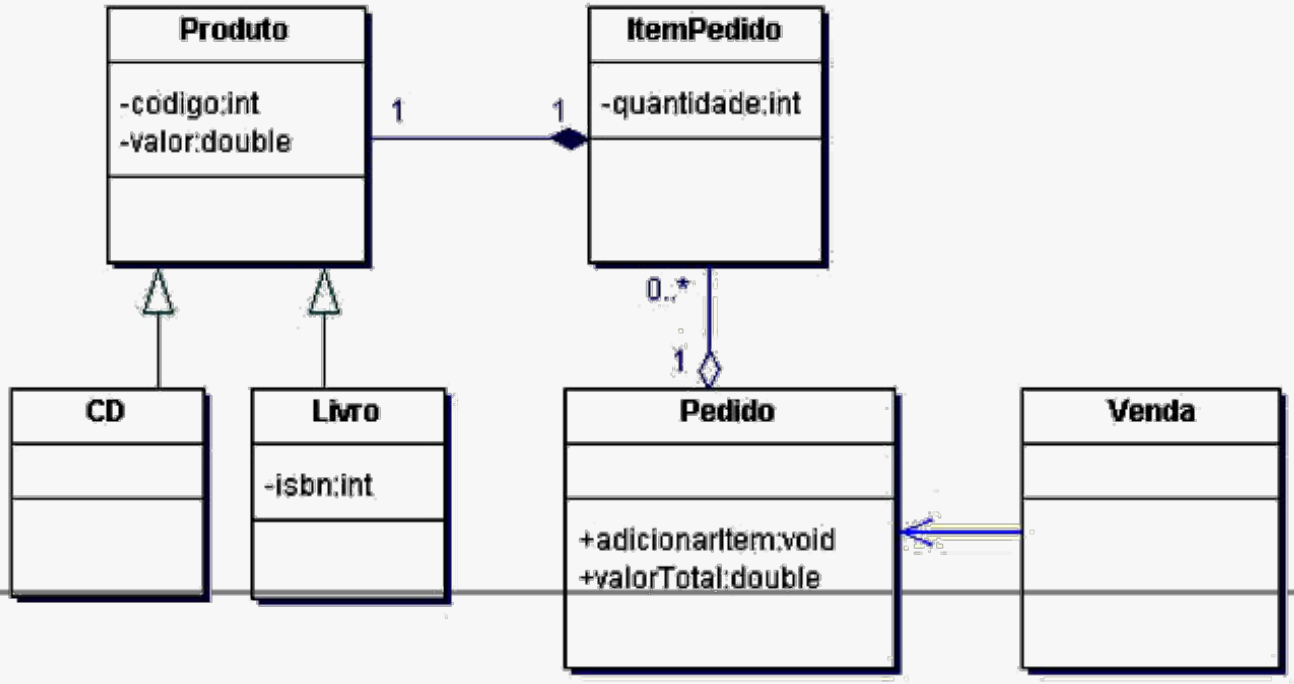
Este diagrama pode ser usado em qualquer nível: fluxo dos casos de uso, fluxo no nível de programação, fluxo das regras de negócio, etc.



UML – Diagramas

Diagrama de Classes

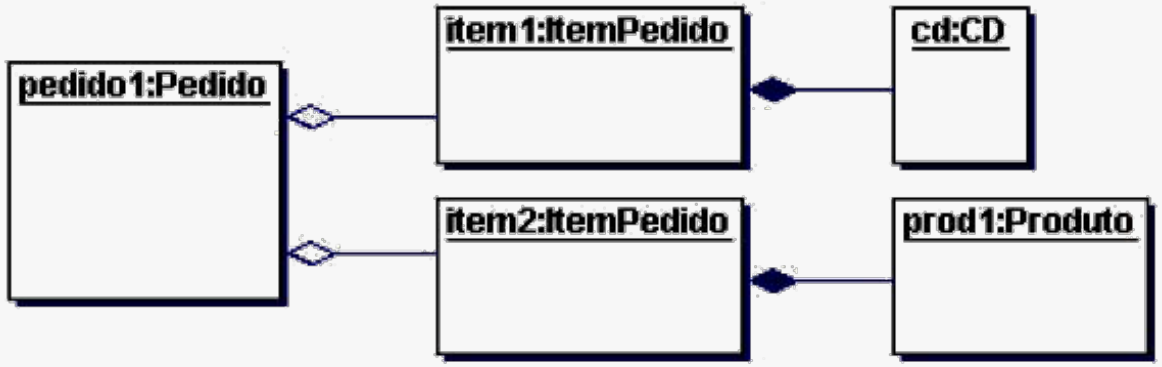
O Diagrama de Classes mostra a estrutura estática do modelo da aplicação. Este diagrama exibe as classes do sistema e o grau do relacionamentos entre elas.



UML – Diagramas

Diagrama de Objetos

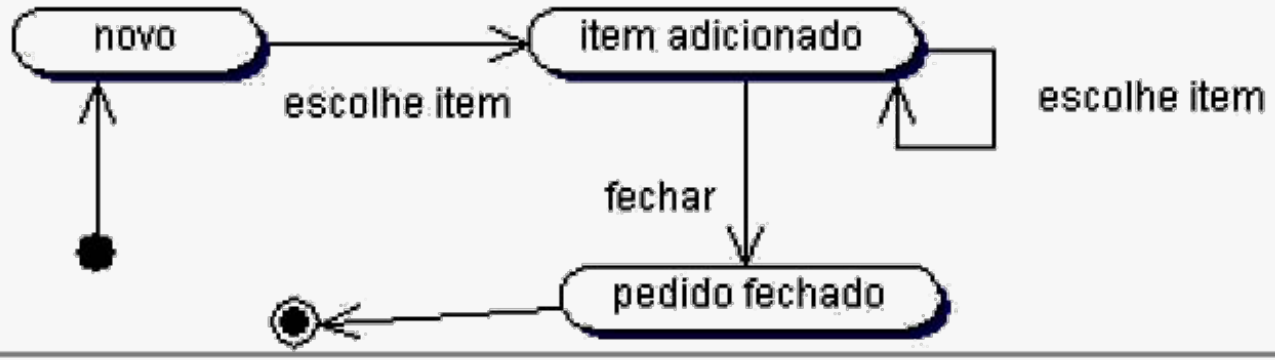
O Diagrama de Objetos é muito similar ao Diagrama de Classes e utiliza quase a mesma notação. Este diagrama mostra uma "fotografia" dos objetos existentes em um determinado momento na execução do sistema. São muito úteis para exemplificar relacionamentos complexos entre objetos em determinado momento. Este diagrama também é usado no Diagrama de Colaboração.



UML – Diagramas

Diagrama de Estados

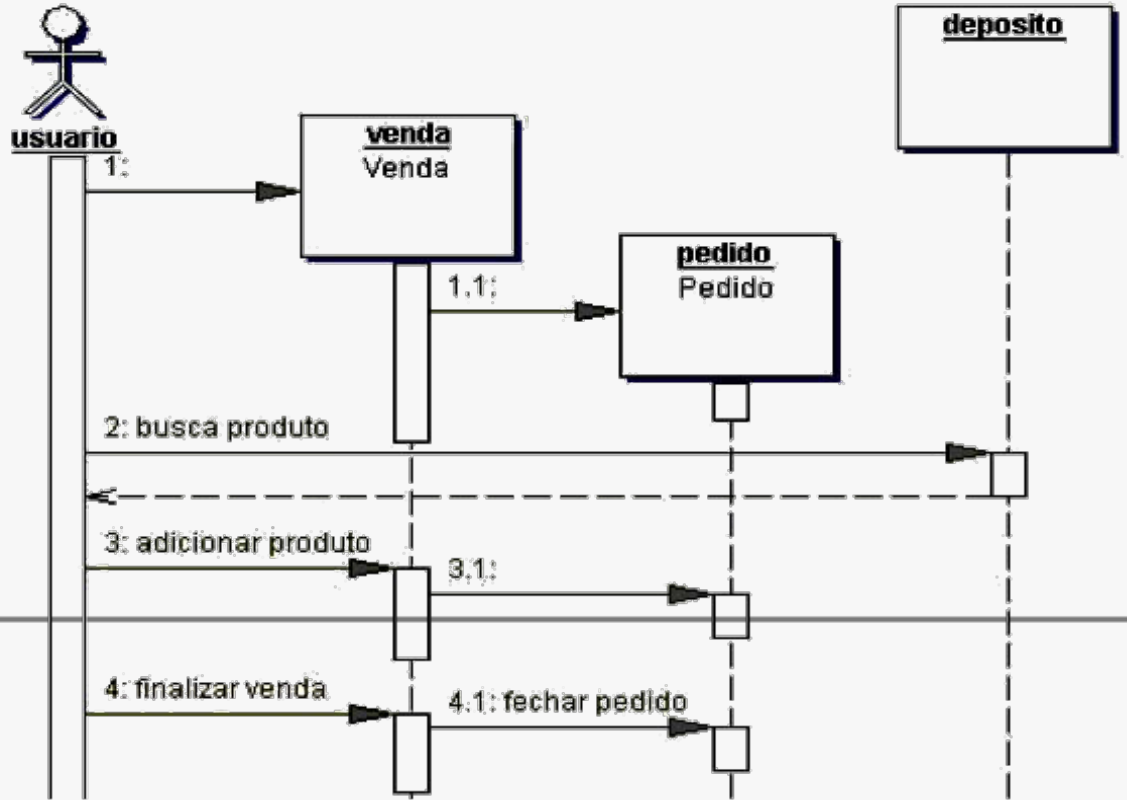
O Diagrama de Estados serve para mostrar todos os estados possíveis dos objetos de um classe do modelo, e que eventos do sistema causam essas mudanças de estado. Não há a necessidade de representar os estados dos objetos de todas as classes.



UML – Diagramas

Diagrama de Seqüência

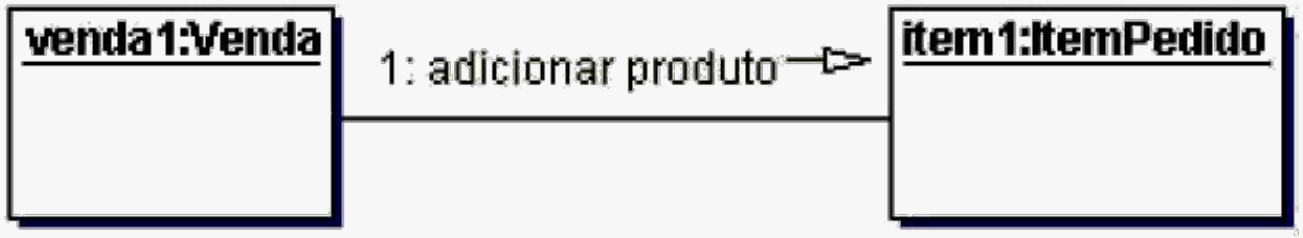
O Diagrama de Seqüência mostra a interação entre os objetos da aplicação arranjados numa linha do tempo. São utilizados para descrever a seqüência de um fluxo ou caso de uso da aplicação. É muito útil para se levantar quais são os envolvidos no fluxo e definir a interface de alguns objetos.



UML – Diagramas

Diagrama de Colaboração

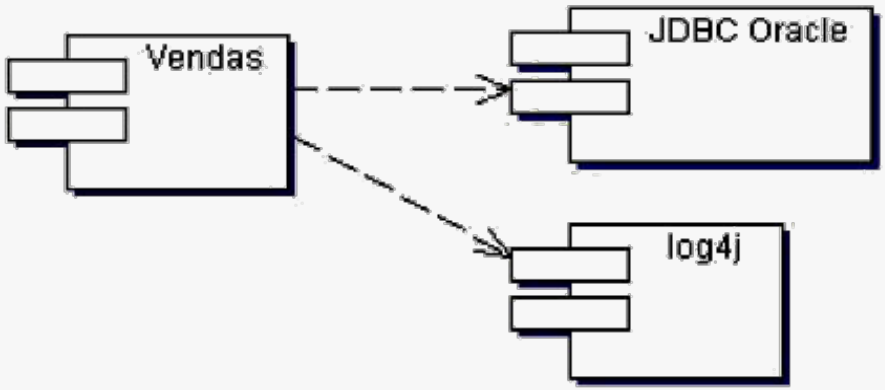
O Diagrama de Colaboração é semelhante ao Diagrama de Seqüência, mostrando a colaboração dinâmica entre os objetos, sem levar em conta a linha do tempo. Neste diagrama, além da troca de mensagens, pode-se perceber o relacionamento entre os objetos.



UML – Diagramas

Diagrama de Componentes

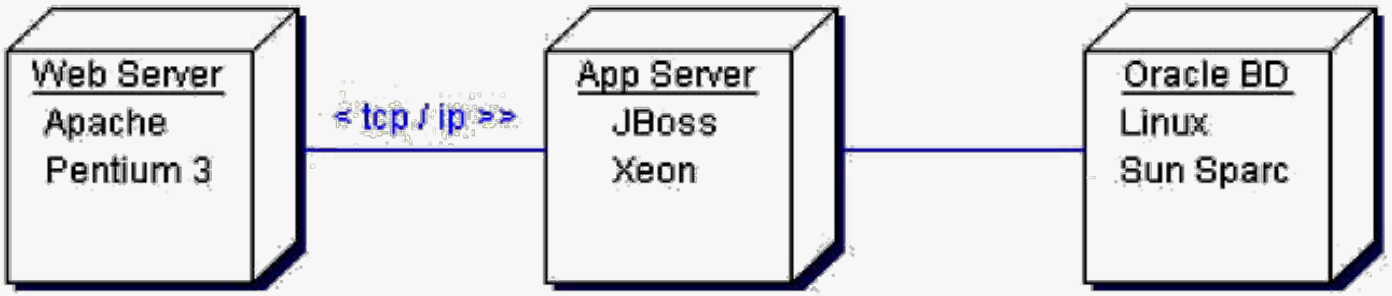
O Diagrama de Componentes mostra o lado funcional, expondo a relação entre seus componentes e suas dependências.



UML – Diagramas

Diagrama de Execução

O Diagrama de Execução mostra o lado funcional, exibindo a arquitetura física do hardware e do software do sistema.



Referências

BOOCH, G; RUMBAUGH, J e JACOBSON, I: UML, Guia do Usuário: tradução; Fábio Freitas da Silva, Rio de Janeiro, Campus, 2000.

CARDELLI, L.; WEGNER, P. On Understanding Types, Data Abstraction, and Polymorphism. ACM Computing Surveys (CSUR). vol.17. 1985.

FURLAN, J. D. Modelagem de Objetos Através da UML: São Paulo, Brasil, Makron Books, 1998.

LARMAN, G. Utilizando UML e padrões: Uma introdução à análise e ao projeto orientados a objetos; Tradução Luiz A Meirelles Salgado. Bookman Porto Alegre, 2000.

WIKIPEDIA, Acessos em Fev/2016.

W3C, Acesso em Fev/2016.