1. PARADIGMA DE ORIENTAÇÃO A OBJETOS

"Paradigma é a representação de um padrão a ser seguido. É um pressuposto filosófico, matriz, ou seja, uma teoria, um conhecimento que origina o estudo de um campo científico; uma realização científica com métodos e valores que são concebidos como modelo; uma referência inicial como base de modelo para estudos e pesquisas."

http://pt.wikipedia.org/wiki/Paradigma

Até há alguns anos atrás, o desenvolvimento de software baseou-se no chamado modelo procedural. Mas, depois de muitos estudos, a comunidade de Tecnologia da Informação desenvolveu uma técnica mais interessante, simples e natural para o processo de análise de problemas e desenvolvimentos de aplicações: a Orientação a Objetos (OOP - Object Oriented Programming). Para compreender mais o significado dessa mudança, e o porquê de uma massiva migração de profissionais do modelo procedural, para a orientação a objetos. Vamos primeiramente fazer uma comparação das duas formas de programar.

1.1. Modelo procedural

Na programação estruturada ou procedural, temos algumas premissas básicas de entendimento, isso significa que:

- O programa completo tem um desenho e desenvolvimento modular;
- Os módulos se integram com uma metodologia descendente, embora possa-se fazer no sentido contrário também, ascendente;
- Codifica-se cada módulo usando as três estruturas de controle básicas: Sequenciais (comandos normais da linguagem), seletivas (desvios condicionais) e repetitivas (laços de repetição);
- Ausência completa de desvios imperativos, uso de comandos como o *goto;*
- Estruturação e modularidade são conceitos complementares;

As técnicas de programação tradicionais, como por exemplo a “decomposição funcional”, leva o desenvolvedor a decompor o sistema em partes menores (funções), criando um emaranhado de inúmeras funções que chamam umas às outras. Geralmente não há separação de conceitos e responsabilidades, causando dependências enormes no sistema, dificultando futuras manutenções no código do programa. Não existe muito reaproveitamento de código, ao contrário, muitas vezes se tem muito código duplicado

Resumidamente podemos dizer que, na programação estruturada ou procedural, existem as seguintes características:

- Entrada , processamento, saída;
- Dados separados das funções de acesso à eles;

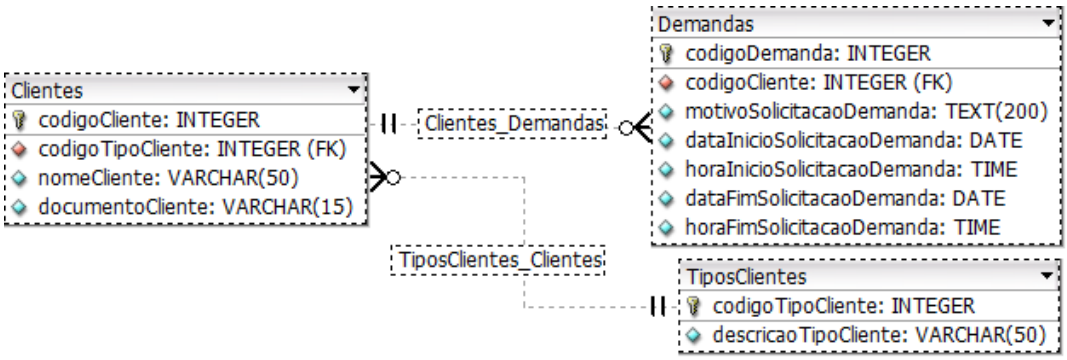
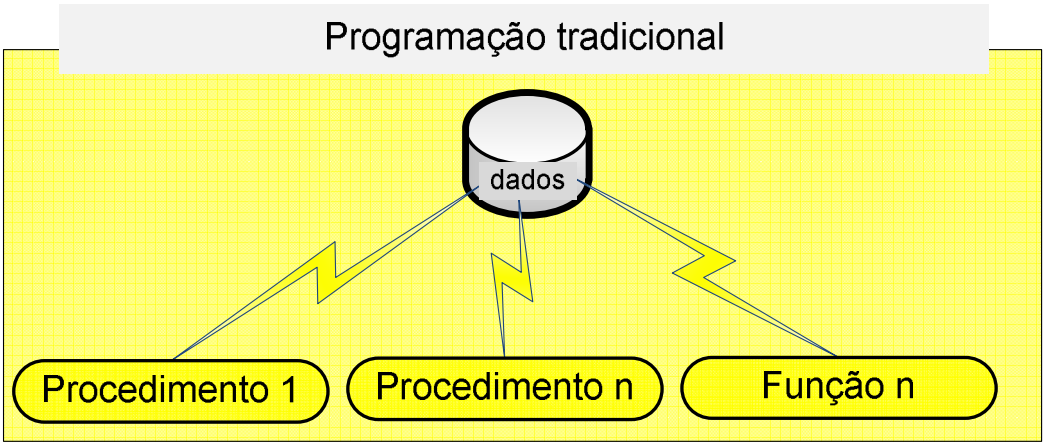


Diagrama ER - Modelo procedural



Forma de acesso aos dados - programação tradicional

1.2. Modelo orientado a objetos

> *"Um sistema construído usando um método orientado a objetos é aquele cujos componentes são partes encapsuladas de dados e funções, que podem herdar atributos e comportamento de outros componentes da mesma natureza, e cujos componentes comunicam-se entre si por meio de mensagens". Eduard Yourdon*

A programação orientada a objetos aponta uma nova visão às definições que regra a programação estruturada quando os problemas a se resolver são complexos. Inverso à programação procedural, a POO (Programação Orientada a Objetos) foca nos dados (*ATRIBUTOS*). O intuito da POO é ajustar a linguagem ao problema. A ideia básica é desenhar formatos de dados que correspondam as características fundamentai de um problema. As linguagens de programação que implementam orientação a objetos combinam, em uma única unidade, o módulo, tanto os dados como as funções e operações que interagem sobre eles. A unidade se chama *OBJETO*. Caso seja necessário alterar os dados de um objeto, essa operação deve ser feita

através de um membro do próprio objeto, seus *MÉTODOS*. Nenhuma outra função pode acessar os dados do objeto. Com isso, a depuração e manutenção do programa fica privilegiada.

Resumidamente podemos dizer que, na POO, existem as seguintes características:

- O mundo é composto por objetos;
- Objetos combinam dados e funções;
- Conceitos do problema são modelados como objetos que são associados e interagem entre si.

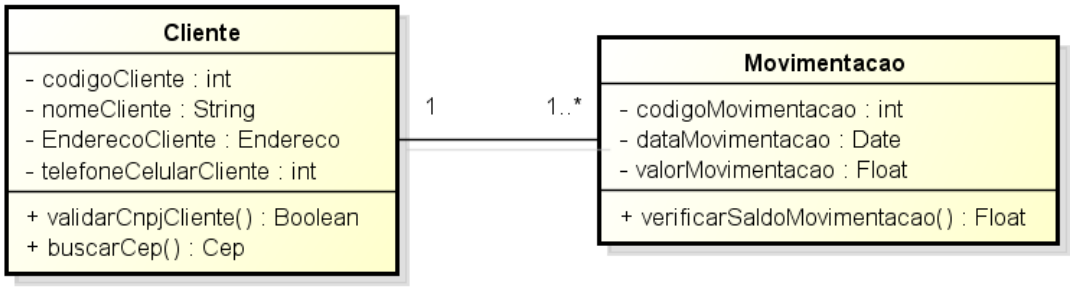
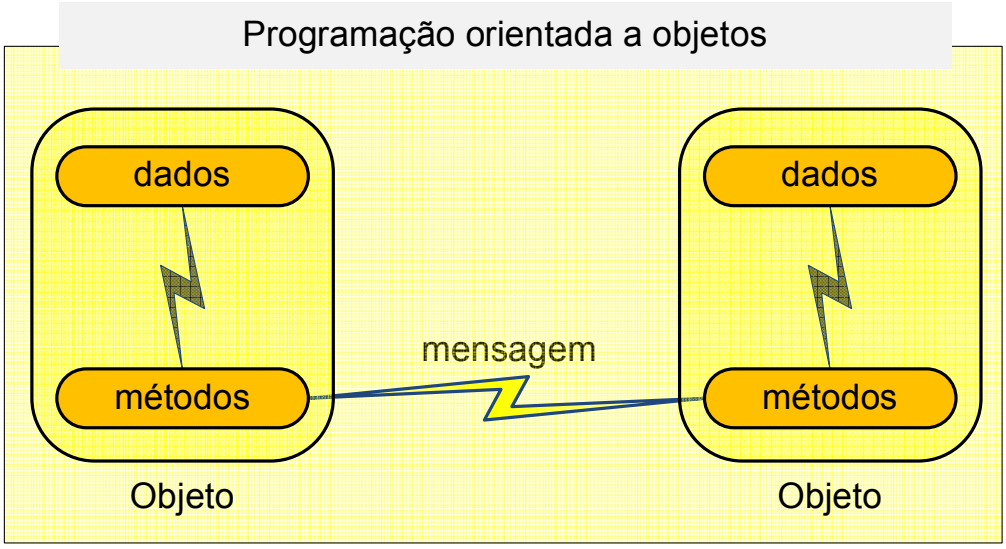


Diagrama de classes

Orientação a objetos não é mágica e nem a "tábua de salvação" do desenvolvimento. É preciso aplicá-la com disciplina e em conjunto com outras técnicas da Engenharia de Software.



Forma de acesso aos dados - programação OO

A adoção do paradigma da OO nos traz alguns benefícios, como:

- Capacidade de enfrentar novos domínios;
- Melhoria da interação analistas x especialistas;
- Aumento da consistência interna da análise;
- Alterabilidade, legibilidade e extensibilidade;
- Apoio à reutilização.

1.3. Linguagens orientadas a objetos

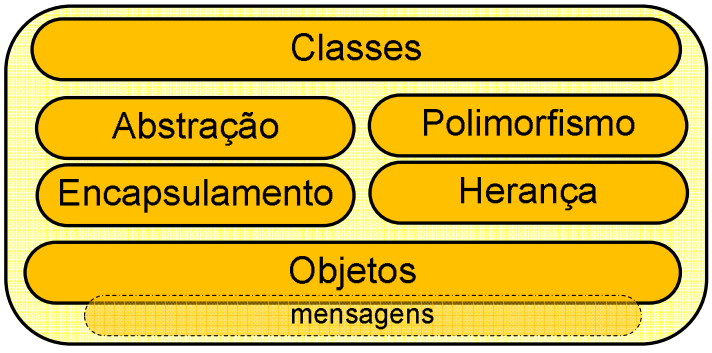
Uma das primeiras linguagens com o conceito de orientação a objetos foi a linguagem Simula, baseada em outra linguagem - Algol60. Seu projeto data no ano de 1962 e teve o seu desenvolvimento em três fases - Simula0 (62-63), simula 1 (63-65) e simula 67 (66-67). Essa última influenciou a primeira linguagem totalmente orientada a objetos, a SmalTalk, que não é mais utilizada, desenvolvida por Alan Kay no início da década de 70.
No site da Tiobe Software, constantemente são elencadas as linguagens e a seu registro de utilização, conforme mostra abaixo, a visualização das dez primeiras posições nesse ranking:

Mar 2014	Mar 2013	Change	Programming Language	Ratings	Change
1	2	^	C	17.535%	+0.39%
2	1	v	Java	16.406%	-1.75%
3	3		Objective-C	12.143%	+1.91%
4	4		C++	6.313%	-2.80%
5	5		C#	5.572%	-1.02%
6	6		PHP	3.698%	-1.11%
7	7		(Visual) Basic	2.955%	-1.65%
8	8		Python	2.021%	-2.37%
9	11	^	JavaScript	1.899%	+0.53%
10	16	^^	Visual Basic .NET	1.862%	+0.97%

Ranking de linguagens - Tiobe Software[1]

1.4. Princípios fundamentais da orientação a objetos

Alinhado ao contexto do "mundo de objetos", a OO introduz e enfatiza os princípios (conceitos) que norteiam essa ideia:



Princípios conceitual de programação OO

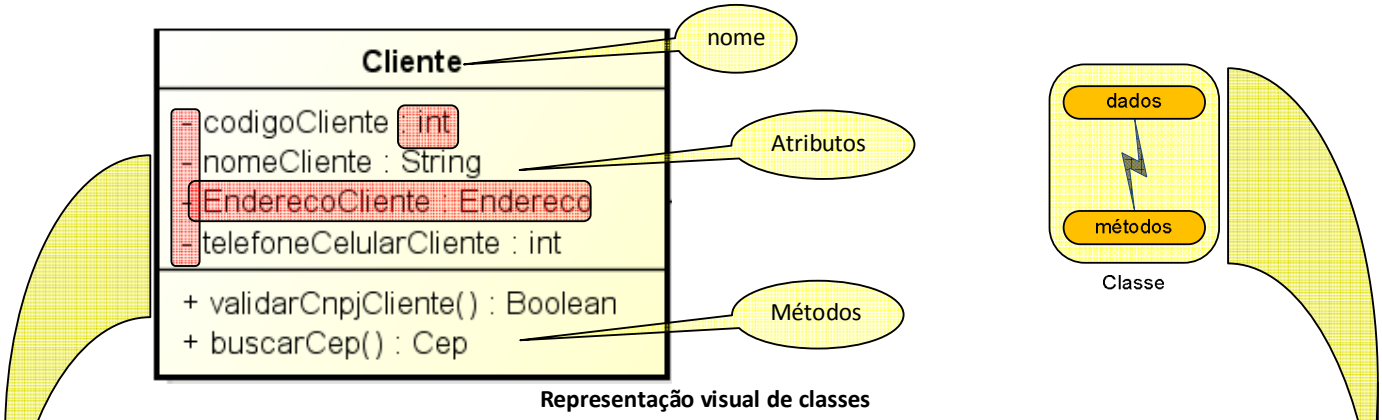
[1] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

1.5. Classes

A identificação de similaridades em características (atributos/dados) e operações (comportamento/métodos) em uma classificação em objetos distintos sugere-nos a criação de uma classe específica, mapeada do mundo real para a modelagem.
Dessa forma, uma classe nada mais é do que a representação de um conjunto de objetos que compartilha estruturas de atributos e métodos bem como seus relacionamentos. Uma classe pode ter um número indeterminado de atributos ou ainda nenhum, da mesma forma, tem a possibilidade de aceitar qualquer número de métodos, inclusive nenhum. Sinteticamente podemos afirmar que uma classe é um modelo para criação de objetos à sua imagem.

Como então podemos identificar uma classe ?

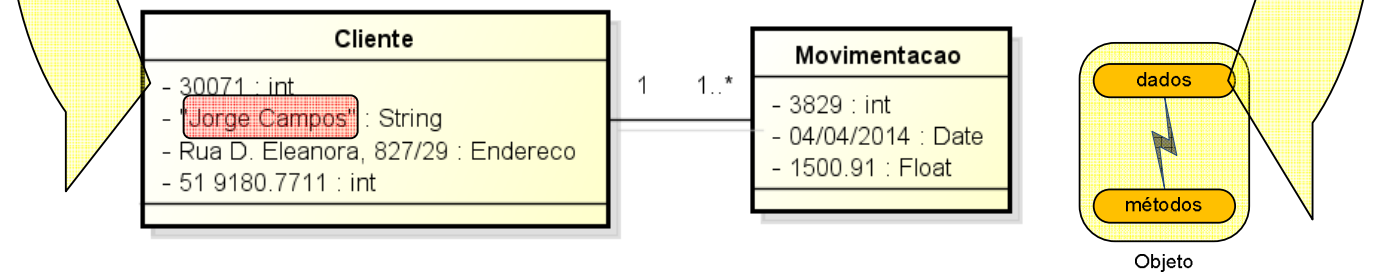
Assim como em banco de dados, onde os substantivos do nosso modelo conceitual pode ser interpretado como uma provável tabela no nosso banco, na POO ou OOP, essa premissa também é válida. Geralmente esses substantivos tendem a levar-nos a objetos físicos (carro, cadeira, livro, etc.) ou ainda conceituais (reserva, norma, etc.). Após a identificação devemos selecionar de acordo com a relevância para o desenvolvimento do sistema, regrando-se isso obteremos uma síntese do que pode ser uma classe e, em um novo levantamento mais apurado, o que será apenas um atributo de uma. Com a experiência, esse processo torna-se mais fácil.



Representação visual de classes

1.6. Objetos

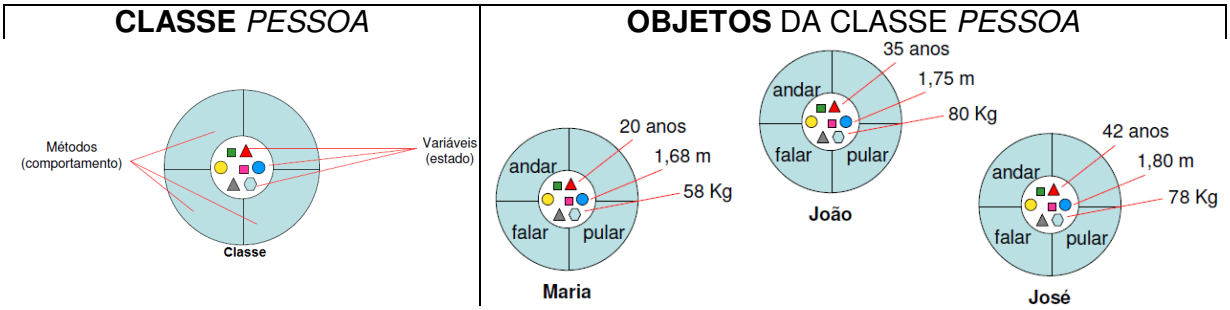
Na concepção do mundo real, objetos são quaisquer coisas reais que possuam características, seus atributos e representa a abstração do tipo de dados ou os estados que o objeto da classe podem possuir. Essa "transformação" de classe para atributo se dá através de um processo de **instanciação** da classe, ou seja, a criação do objeto a partir do modelo definido por ela.



As características que podemos destacar dos objetos, assim com na classe são:

Estado (*estrutura*):	conjunto de suas propriedades e seus valores correntes;
Comportamento (*operações*):	conjunto de serviços que o objeto provê;
Identidade (*Identificador*):	*Identificação* única que diferencia cada objeto, mesmo que tenham o mesmo estado e comportamento.

A diferença primordial entre eles (classes e objetos) é que o objeto possui valoração dos atributos, um estado ou contexto já definido.



Representação visual de classes, instanciação e objetos

1.7. Visibilidade - Modificadores de acesso

Na orientação a objetos é prática, quase que obrigatória, proteger seus atributos contra acessos não permitidos. Com isso, cada classe é responsável por controlar seus atributos, portanto ela deve julgar se aquele novo valor é válido ou não.
Esta validação NÃO deve ser controlada por quem está usando a classe e sim por ela mesma que tem a responsabilidade de centralizar e facilitar mudanças futuras no sistema. Muitas outras vezes nem mesmo queremos que outras classes saibam da existência de determinado atributo, escondendo-o por completo, já que ele diz respeito ao funcionamento interno do objeto.

ESCOPO	MODIFICADOR DE ACESSO			
	private	*default*	*protected*	*public*
mesma classe	***sim***	***sim***	***sim***	***sim***
mesmo pacote	***não***	***sim***	***sim***	***sim***
pacotes diferentes (subclasses)	***não***	***não***	***sim***	***sim***
pacotes diferentes (sem subclasses)	***não***	***não***	***não***	***sim***

É muito comum, e faz todo sentido, que seus atributos sejam private e quase todos seus métodos sejam public embora não seja uma regra. Desta forma, toda conversa de um objeto com outro é feita por troca de mensagens, isto é, acessando seus

métodos. Algo muito mais educado que mexer diretamente em um atributo que não é seu !

1.8. Métodos de leitura e atualização de valores - Getters e setters

O modificador de acesso tem a característica de resguardar as informações do atributo contra leituras e/ou mudanças não validadas. Todo e qualquer acesso aos atributos deve, mas não é regra, ser feito por um método da classe. A prática mais comum para acesso, de maneira controlada, é a criação de dois métodos específicos para esse fim; Um que retorna o valor do dado e outro que o altere.
Esses métodos que possuem a missão de manutenir os dados dos objetos são chamados de métodos **setters**. Como o método deve ser especializado, outro método para leitura dos dados deve ser construído. Esses métodos de resgate de informações do nosso objeto são chamados de **getters**.

A convenção para esses métodos é de colocar a palavra get ou set antes do nome do atributo.
É uma má prática criar uma classe e logo em seguida criar getters e setters pros seus atributos. Você só deve criar um getter ou setter se tiver a real necessidade.

1.9. Abstração

Abstração é a habilidade e a capacidade de se modelar conceitos, entidades, elementos, problemas e características do mundo real, de um domínio do problema em questão, levando-se em conta apenas os detalhes importantes para a resolução do problema e desprezando coisas que não têm importância no contexto.
Se pensarmos no conceito de "*conta corrente*" bancária e abstrairmos este conceito, podemos identificar detalhes comuns, como o número da conta, número da agência e saldo; e operações como débito em conta, depósito e extrato da conta. Basicamente essas são as características de conta corrente para todos os bancos, apesar de um ou outro banco ter uma política de descontos de taxas e outras peculiaridades distintas.

*Assim sendo, **abstrair** consiste em ignorar aspectos irrelevantes e concentrar nos principais.*

1.10. Encapsulamento

Na OO, encapsulamento é o mecanismo utilizado para disponibilizar métodos que operam sobre os dados e que protegem o acesso direto indevido aos atributos de uma instância fora da classe onde estes foram declarados. Esta proteção consiste em se usar modificadores de acesso mais restritivos sobre os atributos definidos na classe e fornecendo métodos que alteram os valores destes atributos de alguma forma. O encapsulamento ajuda a prevenir o problema de interferência externa indevida sobre os dados de um objeto, como objetos que possam alterar os dados de outros objetos indevidamente.

*Então, resumidamente, **encapsulamento** consiste em separar os aspectos externos (o que faz) dos aspectos internos (como faz).*

1.11. Herança

Herança é um mecanismo da POO que permite criar novas classes a partir de classes já existentes aproveitando-se das suas características. O grande reuso e reaproveitamento de código existente é um dos fatores que mais se destacam dentro dessa prática na OO. A herança permite a criação de classes derivadas (subclasses) a partir de classes bases (superclasses). As subclasses são mais especializadas do que as suas superclasses, mais genéricas. As subclasses herdam todas as características de suas superclasses, como seus atributos e comportamento.

*Se a subclasse precisa cancelar características da superclasse, há algo errado ! A resposta positiva para a pergunta "Todos tem ?" é um forte indício de utilização correta da **herança**.*

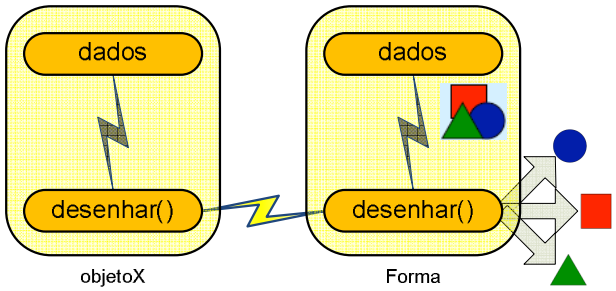
1.12. Generalização e especialização

De acordo com o levantamento e análise, podemos identificar características comuns em várias classes modeladas. O processo de agrupamento de informações comuns sugere a generalização da classe, ou seja: a estruturação de uma classe genérica servirá como base para a criação, ou especialização de outras classe que contenham características similares, porém com especificidades próprias identificadas na modelagem.

***Generalização**: Agrupamento em uma classe, de forma genérica, de semelhanças de atributos/métodos;*
***Especialização**: Refinamento da classe com a adição de novas características e comportamento.*

1.13. Polimorfismo

O objeto emissor não precisa saber quem é o objeto receptor, contanto que saiba que ele responde a uma certa mensagem. O emissor só conhece uma interface. O receptor sabe a implementação certa. É uma forma de sobrescrita, mas todas as operações mantém a mesma semântica em toda a hierarquia.
São diferentes implementações de uma mesma função podem ser usadas conforme as condições atuais (ou conforme o tipo de dados que você fornece como argumento). Exemplo: nos carros que usam o controle de tração, método acelerar(), o mesmo método pode se comportar de maneira diferente conforme o tipo ou as condições do terreno que podem ser passadas por parâmetro.

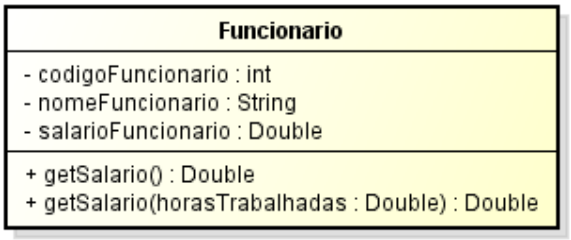


Representação visual de comunicação - polimorfismo

Alguns linguagens promovem o polimorfismo principalmente através do uso de classes abstratas e interfaces, como é o caso da tecnologia Java.

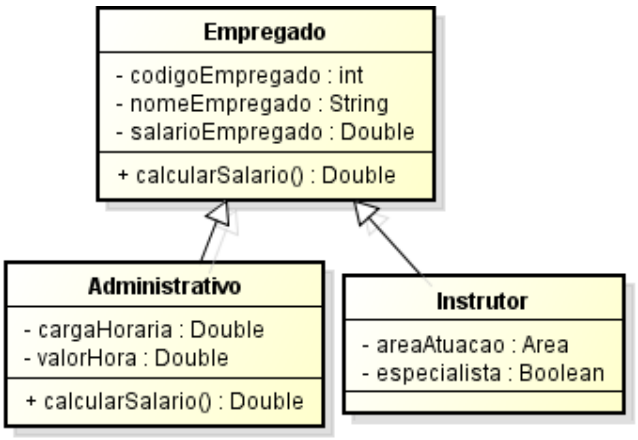
Polimorfismo *é a habilidade do objeto de tomar várias formas.*
Classes abstratas *são classes que não podem gerar instâncias de objetos e que possuem um ou mais métodos sem implementação, deixando para suas subclasses a tarefa de implementar estes métodos abstratos.* ***Interfaces*** *são um tipo de contrato que algumas classes têm de seguir, ou seja, as interfaces apenas definem métodos abstratos que as classes que implementam esta interface têm de implementar.*

Sobrecarga (Overloading): O polimorfismo permite a criação de métodos com o mesmo nome, porém com a distinção entre eles através dos parâmetros passados para o método. Conforme a assinatura desse método e os parâmetros que estão sendo informados, a execução é realizada pelo método que a representa. Isso é chamado de **polimorfismo paramétrico** ou **sobrecarga de operadores**.



Polimorfismo - Overloading

Sobrescrita (overriding): Esse tipo de polimorfismo ocorre entre uma superclasse e uma subclasse quando na classe filha existe um método com mesma assinatura que um já existente na classe pai. Nesse caso, a implementação realizada na subclasse é que passa a ser válida e não mais ao código implementado no método da superclasse.



Polimorfismo - Overriding

1.14. Mensagens

Um objeto por si só não significa muito em um sistema. Para ter algum sentido e valor esses objetos precisam interagir e comunicar-se entre si. Os objetos se comunicam por meio de mensagens. Quando um objeto A quer se comunicar com um objeto B é enviada uma mensagem de A para B. Então, se A envia uma mensagem para B, podemos entender que o objeto A está executando um método do objeto B.

As mensagens são compostas por três partes:

- Objeto a quem a mensagem é endereçada;
- Nome do método a ser chamado;
- Parâmetros que o método recebe.

*Enviar uma **mensagem** significa executar um método de um objeto.*