

Princípios de Orientação a Objetos

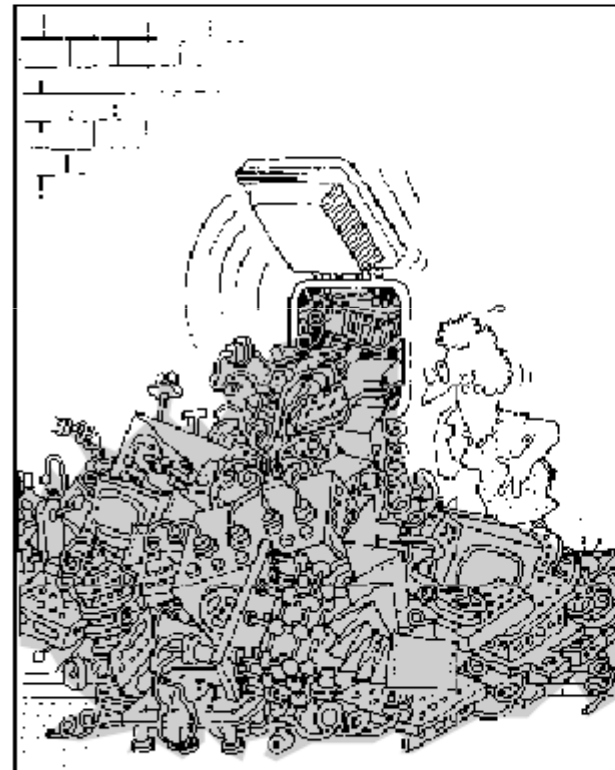
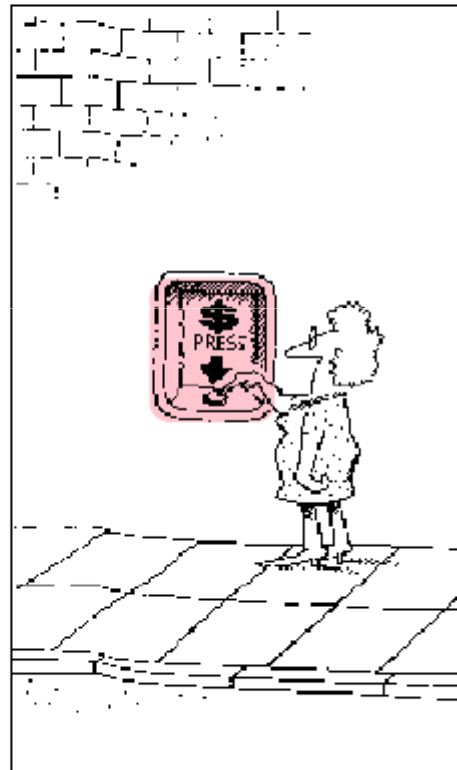
Introdução à Tecnologia de Objetos



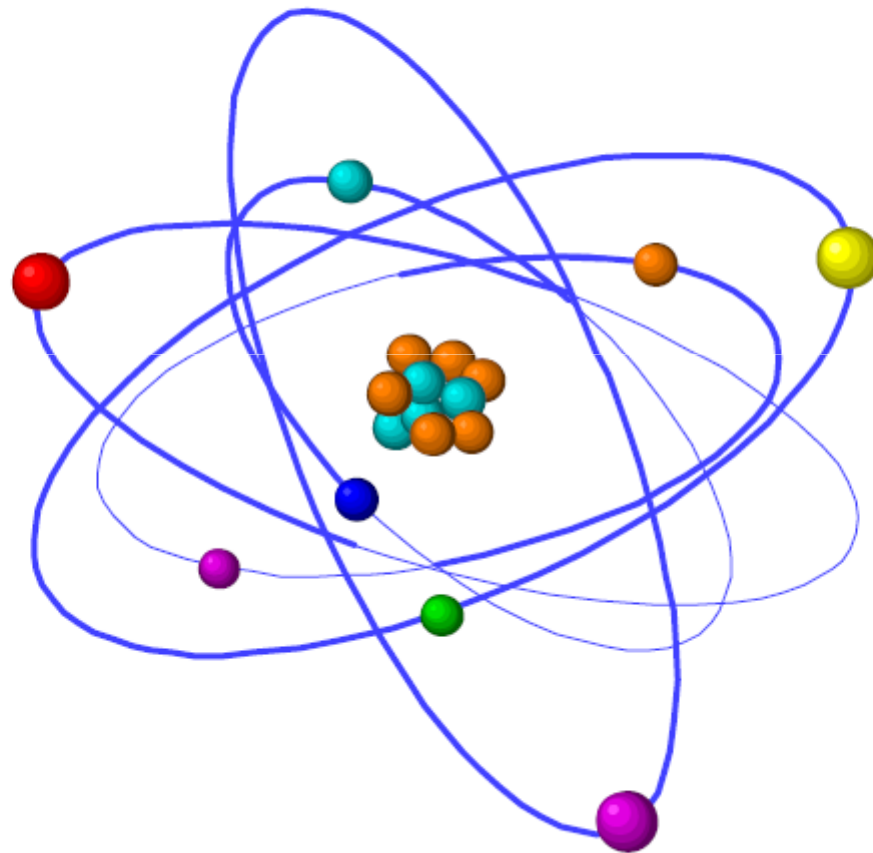
O que é Orientação a Objetos?

- “Técnica de modelagem de software que procura construir sistemas complexos a partir de componentes.”
 - Khoshafian, S. e Abnous, R.
- “Objetos são coisas que podem ser manipuladas.”
 - Martin, R.C.

Sistemas Complexos



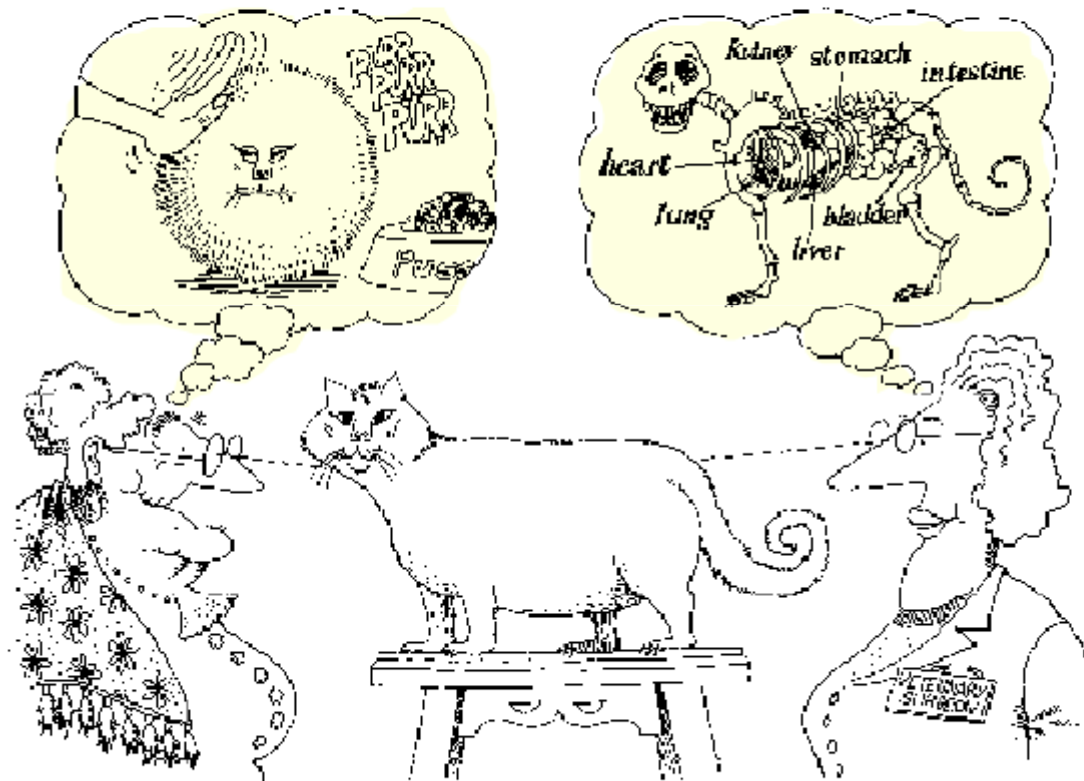
Modelo



O Modelo de Objetos

- Abstração
- Encapsulamento
- Modularidade
- Hierarquia
- Tipagem
- Concorrência
- Persistência

Abstração



Eliminação
do
irrelevante e
amplificação
do essencial

Abstração

- A abstração denota as características essenciais de um objeto que o distingue de outros objetos e oferece uma fronteira conceitual claramente definida, sempre a partir da visão do observador
- Uma abstração deve ser entendida e analisada independentemente do mecanismo que a implementa

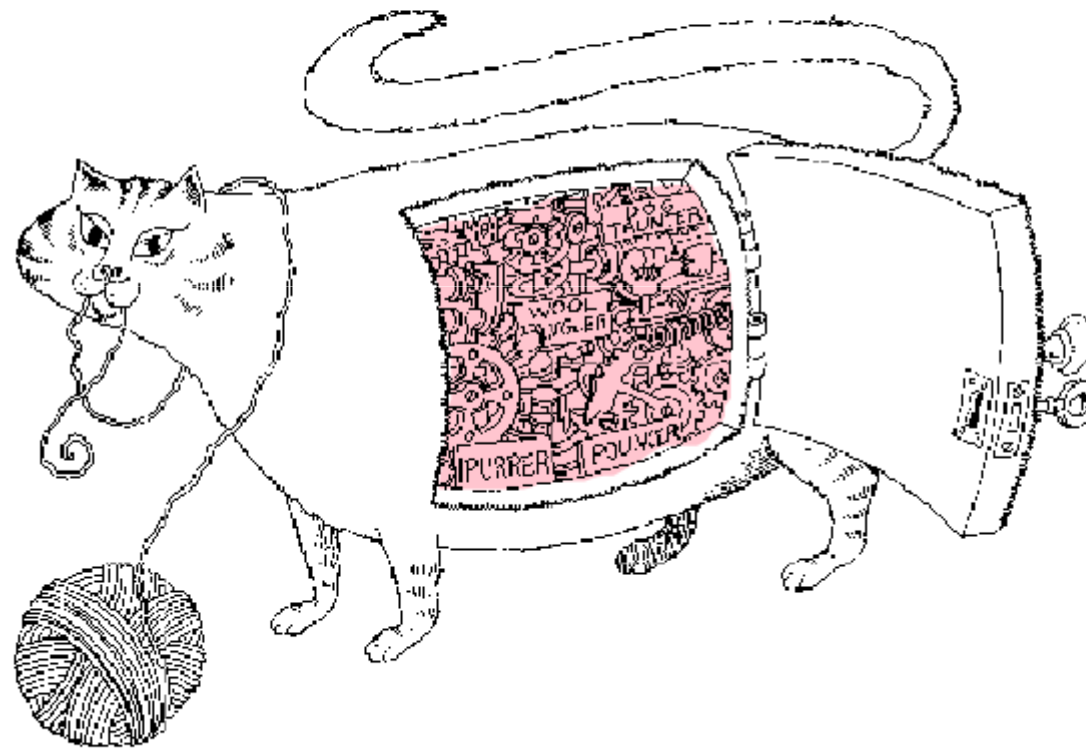
Abstração

No dicionário Aurélio, [abstração](#) significa considerar isoladamente coisas que estão unidas, ou seja, partimos do enfoque global de um determinado problema e procuramos separar os elementos fundamentais e colocá-los de uma forma mais próxima da solução. [A idéia da abstração é identificar os elementos essenciais de um problema e suas propriedades fundamentais, separando ocorrências e atributos acidentais \(isolando detalhes menos importantes\).](#)

Para a análise orientada a objeto, abstração é o processo de [identificação dos objetos e seus relacionamentos](#). Ela permite ao analista concentrar-se no que um objeto é e faz, sem se preocupar como ele o faz.

A abstração se dá em diferentes níveis: inicialmente, abstrai-se o objeto; em seguida, procuramos identificar seus atributos e funcionalidades. De um conjunto de objetos, cria-se um conjunto de classes relacionadas, geralmente uma hierarquia ou um assunto.

Encapsulamento



Nenhuma parte
de um sistema
complexo deve
depende de
detalhes
internos de
qualquer outra
parte

Encapsulamento

- É o processo de compartimentalização dos elementos de uma abstração
- Encapsulamento serve para separar a interface contratual de uma abstração de sua implementação

Encapsulamento

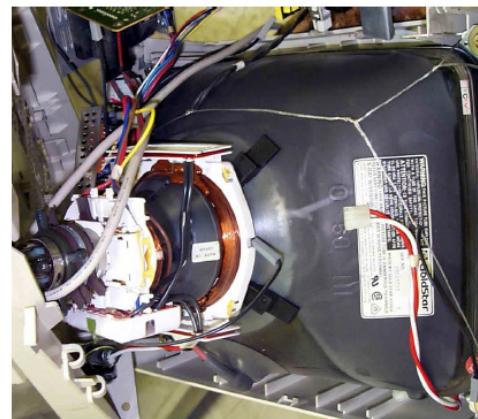
Todos os equipamentos que utilizamos são altamente encapsulados. Tome como exemplo seu monitor de computador. Ele tem um pequeno conjunto de botões que lhe permitem manipular os atributos do objeto monitor que são de seu interesse, como: ligar/desligar, controlar o brilho, o contraste, a resolução. Mas você sabe que o funcionamento do objeto monitor é extremamente complexo e que, ao mudar a resolução, uma série de atributos internos são processados e alterados.

Um monitor CRT tem um tubo de raios catódicos, dezenas de fios, placas internas, resistores, capacitores e uma infinidade de dispositivos elétricos que estão ocultos, encapsulados, escondidos do usuário; ou seja, o encapsulamento separa o que é visível (público) do que é oculto (protegido/privado).

Figura 2: O que você vê?



(a) A visão do usuário.



(b) A visão do engenheiro.

Encapsulamento

Para a análise orientada a objeto, encapsulamento é o ato de esconder do usuário informações que não são de seu interesse. O objeto atua como uma caixa preta, que realiza determinada operação, mas o usuário não sabe, e não precisa saber, exatamente como é feita; ou seja, o encapsulamento envolve a separação dos elementos visíveis de um objeto dos invisíveis. Os elementos visíveis formam a interface de acesso ao objeto.

Segundo Rumbaugh et al. (1994); Blaha and Rumbaugh (2006), "o encapsulamento evita que um programa se torne tão interdependente que uma pequena modificação tenha um efeito de propagação grave".

Pressman (2002) lembra que "os detalhes internos de implementação dos dados e procedimentos são ocultados do mundo exterior (ocultação de informação). Isso reduz a propagação de efeitos colaterais quando ocorrem modificações". Segundo Parga (2006), "como consequência do encapsulamento temos uma diminuição considerável do nível de acoplamento entre as classes que fazem parte do sistema. O nível de acoplamento é avaliado pela interdependência entre os códigos de duas ou mais classes, ou seja, o quanto o funcionamento de uma determinada classe é afetado por alterações em uma outra classe que interaja com ela. O encapsulamento permite a implementação de uma abstração com a separação clara entre a interface com o usuário e a implementação".

Modularidade

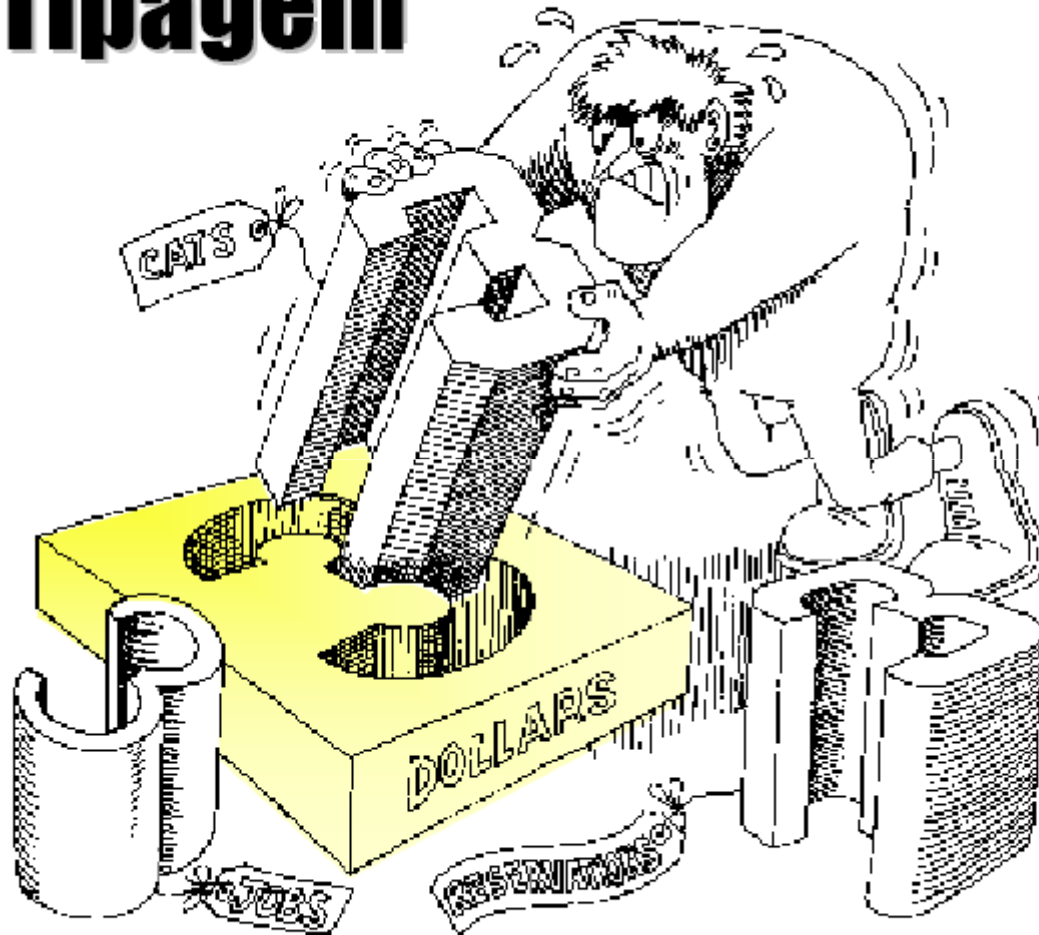
Propriedade de um sistema estar decomposto em partes coesas e fracamente acopladas



Modularidade

- A atividade de particionar um programa em componentes individuais pode reduzir sua complexidade
- Módulos são dependentes uns dos outros, o que leva a uma relação de acoplamento

Tipagem



Objetos diferentes não são intercambiáveis, exceto de modo restrito

Tipagem

- Um tipo se descreve através de uma classe
- Em ambientes fortemente tipados, objetos devem se limitar ao relacionamento estrito dentro de sua casta

Tipos e Classes

Quando falamos de classes, lembramo-nos de classes sociais, de classes de objetos da natureza, de hierarquias (por exemplo, a classe dos animais vertebrados). De um modo geral, uma classe descreve um grupo de objetos com os mesmos atributos e comportamentos, além dos mesmos relacionamentos com outros objetos.

⇒ Cachorros – buldogue, pastor alemão, vira-lata.

⇒ Relógios – de ponteiro, digital, com alarmes, com cronômetros, com calculadoras.

A classe contém a descrição da forma do objeto; é um molde para a criação do objeto, é uma fábrica de objetos. Uma classe também é um tipo definido pelo usuário.

Para a análise orientada a objeto, uma classe é um conjunto de códigos de programação que incluem a definição dos atributos e dos métodos necessários à criação de um ou mais objetos.

Grady Booch Booch (1986), diz que "um objeto é uma entidade concreta que existe no tempo e no espaço, uma classe representa apenas uma abstração, a essência do objeto".

Blaha and Rumbaugh (2006) lembra que "qualquer escolha de classe é arbitrária e depende da aplicação".

Relacionamentos entre Classes

- Associação
- Herança
- Agregação
- Uso
- Instanciação

Atributos / Propriedades

Podemos relacionar alguns atributos (propriedades) a todo objeto.

No exemplo do relógio, podemos relacionar a hora e a data. Uma cadeira pode ser branca ou preta, ter apoio para os braços, ter rodinhas. Um monitor de computador pode ter 15", 17", 19" ou 21" (polegadas) e sua resolução máxima em pixels pode ser 1024x768, 1280x1024, 1600x1280; ou seja, todo objeto tem seus próprios atributos, propriedades.

Os atributos do objeto são utilizados para diferenciar os objetos e para identificar o estado do objeto.

Na programação orientada a objeto, os atributos são definidos na classe e armazenados de forma individual ou coletiva pelos objetos.

⇒ Quando um atributo é individual (ou de instância), ele é armazenado no objeto.

Exemplo: A hora de um relógio.

Cada relógio tem uma hora, que pode ou não estar certa.

⇒ Quando um atributo é compartilhado entre todos os objetos de uma classe, ele é armazenado nela e é conhecido como atributo coletivo ou de classe. Este tipo de atributo define uma característica de toda classe.

Exemplo: Um contador de relógios criados.

Atributos / Propriedades

Podemos relacionar alguns atributos (propriedades) a todo objeto.

No exemplo do relógio, podemos relacionar a hora e a data. Uma cadeira pode ser branca ou preta, ter apoio para os braços, ter rodinhas. Um monitor de computador pode ter 15", 17", 19" ou 21" (polegadas) e sua resolução máxima em pixels pode ser 1024x768, 1280x1024, 1600x1280; ou seja, todo objeto tem seus próprios atributos, propriedades.

Os atributos do objeto são utilizados para diferenciar os objetos e para identificar o estado do objeto.

Na programação orientada a objeto, os atributos são definidos na classe e armazenados de forma individual ou coletiva pelos objetos.

⇒ Quando um atributo é individual (ou de instância), ele é armazenado no objeto.

Exemplo: A hora de um relógio.

Cada relógio tem uma hora, que pode ou não estar certa.

⇒ Quando um atributo é compartilhado entre todos os objetos de uma classe, ele é armazenado nela e é conhecido como atributo coletivo ou de classe. Este tipo de atributo define uma característica de toda classe.

Exemplo: Um contador de relógios criados.

Métodos / Comportamentos

Podemos relacionar determinados comportamentos, funções, operações, ações e reações a um objeto. Veja a seguir alguns exemplos:

- ⇒ Um automóvel tem o comportamento de se locomover.
- ⇒ Um equipamento de medição é utilizado para realizar medidas.
- ⇒ Um computador processa os programas, realizando as operações codificadas.

Na análise orientada a objeto, as funções, operações e os comportamentos dos objetos são descritos pelos métodos, os quais também servem para manipular e alterar os atributos do objeto (alteram o estado do objeto).

Em um programa orientado a objeto os estímulos são representados por eventos. Por exemplo: um evento ocorre quando o usuário clica o mouse sobre o ícone impressora, o qual envia uma mensagem para o objeto impressora solicitando a impressão do arquivo selecionado. O método `Imprimir()` recebe o nome do arquivo a ser impresso, realiza a impressão do mesmo e então retorna um *flag* indicando que a operação de impressão terminou com sucesso ou falhou. Observe que o objeto impressora é representado na tela por seu ícone. Na maioria dos programas, um item de menu é usado para acessar um diálogo com propriedades do objeto impressora.

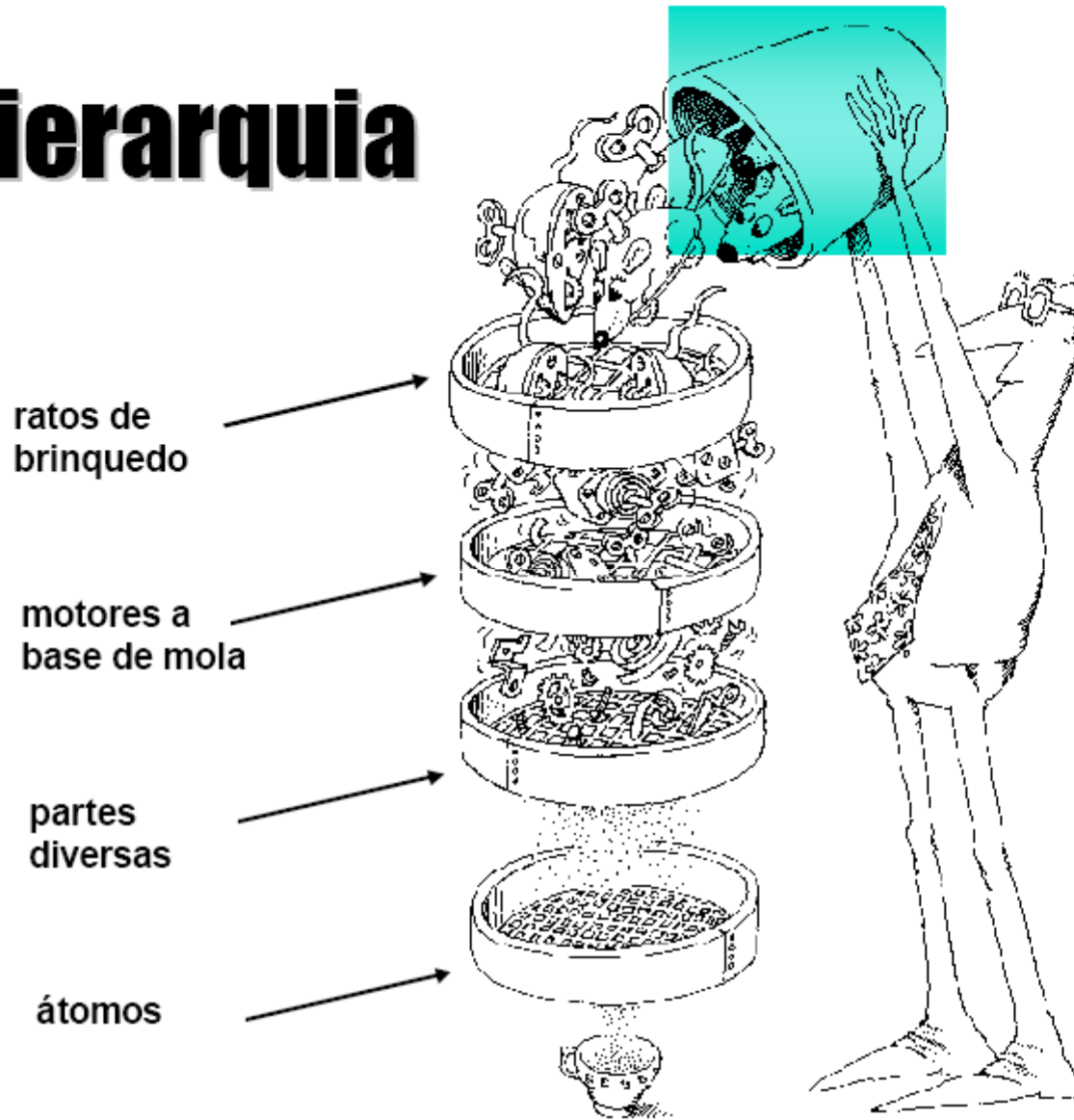
Métodos : Elementos

Podemos dizer que um método é uma operação realizada sobre um objeto e é descrito por uma classe. Um método é a implementação de uma operação.

São elementos de um método:

- ⇒ **Retorno** – todo método retorna um objeto após sua execução.
- ⇒ **Nome** – o nome de um método deve identificar com clareza o que ele faz.
- ⇒ **Parâmetros** – são as variáveis e objetos passados para o método para realização de suas tarefas.

Hierarquia



Classificação
e ordenação
de
abstrações

Hierarquia

- Promove a classificação e organização das abstrações
- Formas de organização:
 - Herança (simples ou múltipla)
 - Agregação

Hierarquia e Herança

A herança está relacionada às hierarquias e às relações entre os objetos. No dia-a-dia, quando se fala de herança, refere-se à transferência de propriedades de um pai aos seus filhos, ou seja, aquilo que é do pai passa a ser do filho.

É comum ainda o dito popular "puxou o pai", que significa que o filho tem as mesmas características do pai. De uma maneira geral, as pessoas sabem que o filho assemelha-se ao pai, mas não são a mesma pessoa. Além disso, o filho apresenta determinadas características diferentes de seu pai. Veja a seguir outros exemplos:

- ⇒ Um processador Pentium IV tem preservadas todas as características do Pentium III, mas acrescentou mais memória cache (a memória cache já existia, mas foi ampliada). Alguns modelos apresentam a tecnologia de *Hyper-Threading* que antes não existia.
- ⇒ Uma placa-mãe nova apresenta a interface USB; trata-se de uma novidade que antes também não existia.

Hierarquia e Herança

Na análise orientada a objeto, herança é o mecanismo em que uma classe-filha herda automaticamente todos os atributos e métodos de sua classe-pai (classe-base).

Herança é a propriedade de podermos criar classes que se ampliam a partir de definições básicas. De classes mais simples e genéricas para classes mais complexas e específicas. A herança permite criar classes-filhas implementando apenas os métodos e atributos que se diferenciam da classe-pai.

A maior vantagem do uso do conceito de herança está associada à compreensão de que os objetos que fazem parte da herança têm o mesmo comportamento, possibilitando um maior reaproveitamento de código com conseqüente aumento da segurança.

- ⇒ **Herança simples** – ocorre quando uma classe herda as propriedades de uma única classe-pai.
- ⇒ **Herança múltipla**¹ – ocorre quando uma classe tem mais de um pai. Exemplo: herança de comportamento; muitas vezes dizemos que um menino herdou o jeito engraçado do tio e é estudioso como o pai, ou seja, o menino herdou características comportamentais de mais de uma pessoa.

¹A herança múltipla é utilizada em C++ (não sendo disponível em Java).

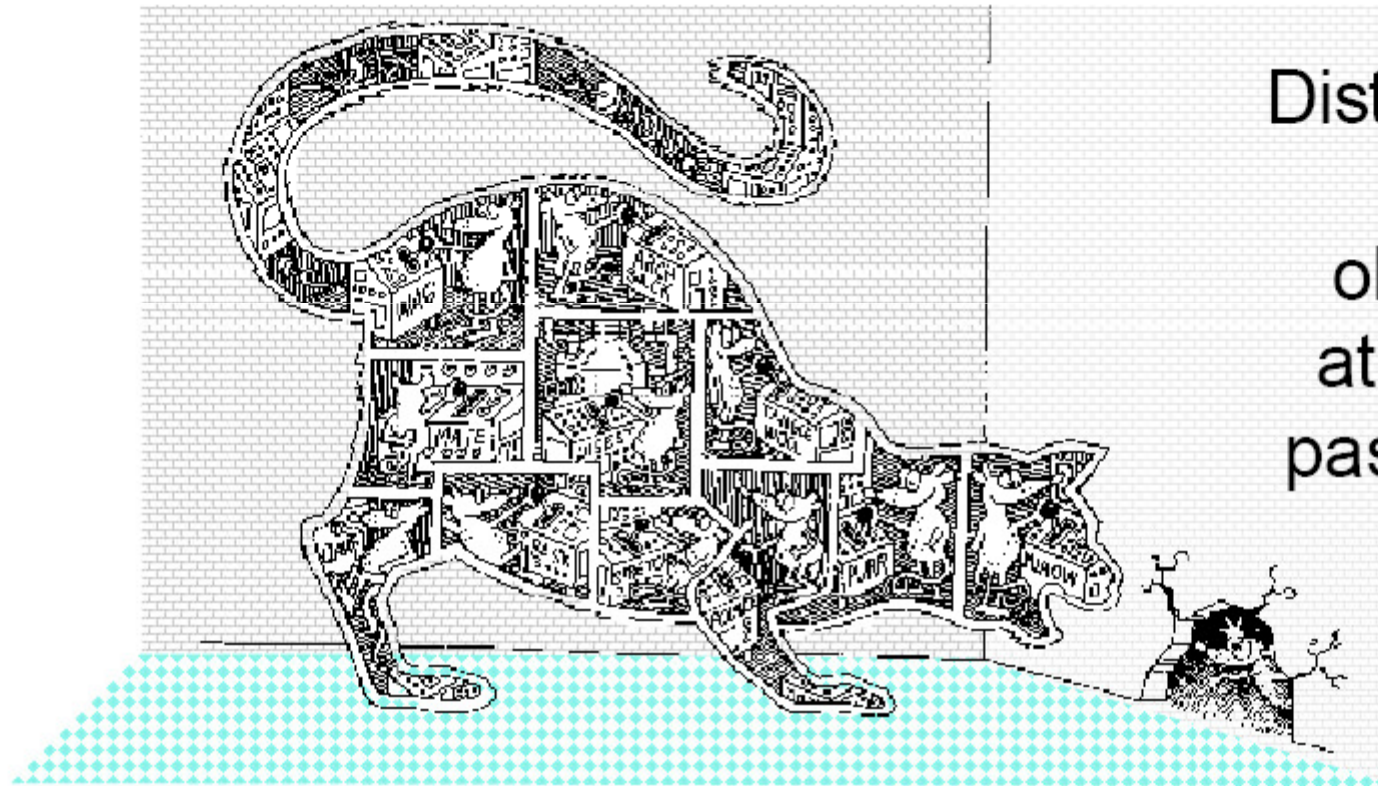
Polimorfismo

A palavra polimorfismo significa muitas formas e representa o fato de uma determinada característica (por exemplo, a potência do motor do veículo) ser diferente para cada filho (tipo de veículo). Quem já andou de Volks e de Mercedes sabe bem a diferença.

Na natureza, o conceito de polimorfismo é inerente ao processo de desenvolvimento; os seres evoluem, modificam-se. Por exemplo: o homo-sapiens é uma evolução de seus ancestrais e tem atributos melhorados e atributos novos. Em suma, estamos partindo de um objeto mais simples e evoluindo, mas os conceitos do objeto-pai continuam a existir nos objetos descendentes, mesmo que tenham sofrido modificações, aperfeiçoamentos e tenham assumido novas formas (polimorfismo).

Para a AOO, polimorfismo é o processo de redefinição dos métodos nas classes-derivadas. Os métodos da classe-base são reescritos para realizarem as mesmas tarefas de uma forma mais especializada.

Concorrência

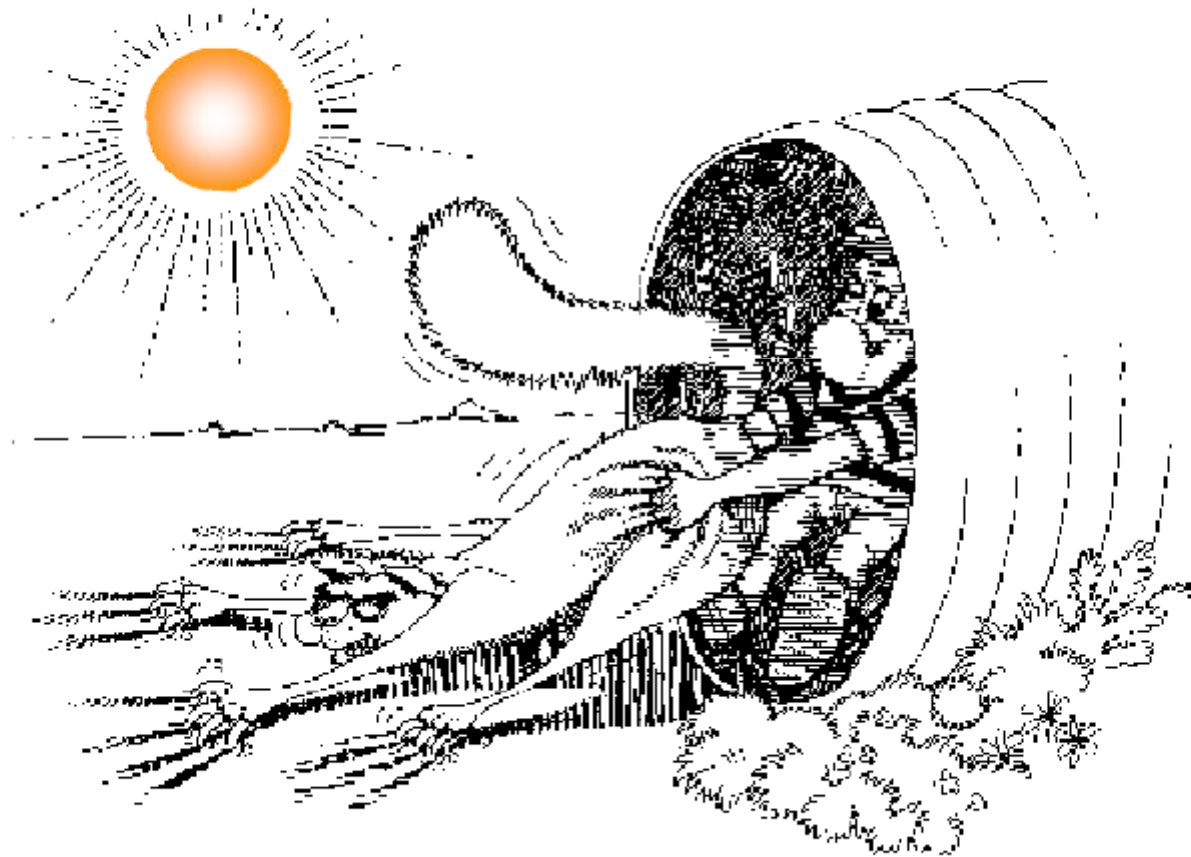


Distinção
entre
objetos
ativos e
passivos

Concorrência

- Pode ser atingida pelo uso de muitos processadores ou por algoritmos de divisão de tempo sobre um único processador
- Possui as mesmas preocupações de projeto, sendo OO ou não:
 - exclusão mútua, sincronismo, *dead lock* ...

Persistência



Tempo
de vida
dos
objetos

Persistência

- Tempos de vida de objetos:
 - temporário (avaliação de expressões)
 - escopo de ativação (locais)
 - durante o processo (memória)
 - além do processo (armazenamento permanente)

Como atingir Orientação a Objetos?

- Cumprindo os princípios fundamentais do modelo de objetos:
 - Abstração
 - Encapsulamento
 - Modularidade
 - Hierarquia

Resumo

Neste capítulo aprendemos alguns [conceitos básicos de OO](#), como a [idéia da abstração](#), que consiste em simplificar os sistemas eliminando detalhes, ou então o conceito de [encapsulamento](#) ou ocultamento da informação cujo objetivo é simplificar o objeto, permitindo ao usuário ver apenas o que será efetivamente utilizado. Vimos ainda [exemplos reais e imaginários de classes e objetos](#), com suas propriedades e funcionalidades. Relembramos o [conceito de herança](#) e tivemos uma idéia do conceito de [polimorfismo](#).

Apresentaremos, no capítulo – Modelagem Orientada a Objeto, o que é um modelo e porque usamos modelos, o que é a UML e quais seus diagramas.

Exercícios

- Na página do grupo da disciplina:
 - Arquivo: [APS - Cap. 2 - APOO - Parte 1 - Principios de OO - Exercícios.pdf](#)

Referências Bibliográficas

- Blaha, M. and Rumbaugh, J. (2006). *Modelagem e Projetos Baseados em Objetos com UML 2*. Campus, Rio de Janeiro.
- Booch, G., editor (1986). *Object-oriented development*, volume 12 of *IEEE Transactions on Software Engineering*.
- Parga, D. F. (2006). Introdução a java. *Revista inform*, pages 10–19.
- Pressman, R. S. (2002). *Engenharia de Software*. McGraw Hill, Rio de Janeiro, 5 edition.
- Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorenzen, W. (1994). *Modelagem e Projetos Baseados em Objetos*. Edit. Campus, Rio de Janeiro.