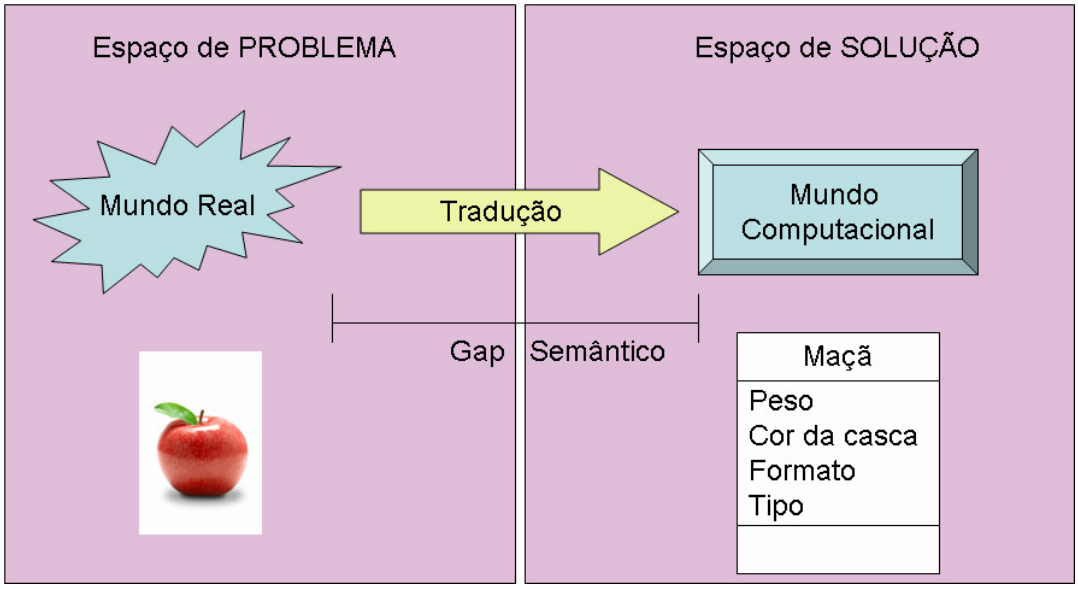
1 INTRODUÇÃO À ORIENTAÇÃO A OBJETOS

1.1 O PROPÓSITO DA ORIENTAÇÃO A OBJETOS

A construção de uma solução computadorizada consiste no mapeamento do problema a ser resolvido no mundo real (o **Espaço do Problema**) em um modelo de solução no **Espaço de Soluções**, isto é, o meio computacional. A modelagem envolve, então, a identificação de objetos e operações relevantes no mundo real e o mapeamento desses em representações abstratas no mundo computacional.

À distância existente entre o problema no mundo real e o modelo abstrato construído, convencionou-se chamar **gap semântico** e, obviamente, quanto menor ele for, mais direto será o mapeamento e, portanto, mais rapidamente serão construídas soluções para o problema. Sob essa ótica, é fácil perceber que o *gap* semântico representa a área de atuação da Engenharia de Software. Diversas técnicas e métodos têm sido propostos para as diferentes fases do processo de desenvolvimento, buscando minimizá-lo. A **Orientação a Objetos** é um dos paradigmas existentes para apoiar o desenvolvimento de sistemas, que busca fornecer meios para se diminuir o *gap* semântico. (Origem: Prof. Ricardo de Almeida Falbo, Apostila de Análise de Sistemas - Notas de Aula, 2002, UFES)



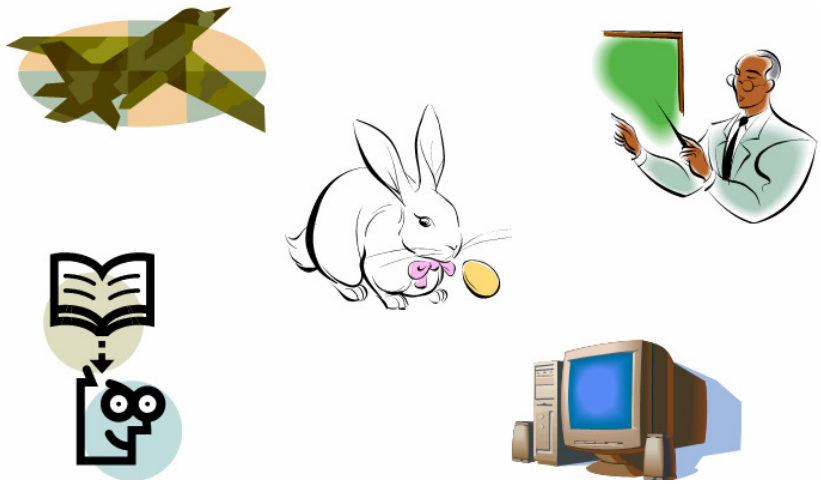
A computação atualmente é parte do dia-a-dia da Sociedade. Em termos de hardware e software a evolução é constante e tem atingido altos níveis. Pode-se dizer que para boa parte dos problemas do mundo real que precisam ser mapeados em sistemas computacionais, mais precisamente em softwares, já possuem as tecnologias de hardware e software necessários. Assim o problema neste caso deixa de ser onde o sistema executar e passa a ser como resolver o problema de maneira mais eficiente, ou seja, como representar o problema do mundo real para o mundo computacional capturando todas as características necessárias e que execute a melhor solução possível. E é quando começa-se a falar em metodologias, tecnologias e técnicas de desenvolvimento de software. O que está dominando atualmente o mercado é justamente a **Orientação a Objetos (OO)**.

A técnica da orientação a objeto está entre as melhores maneiras conhecidas para se desenvolver **Sistemas de Informação Automatizados** grandes ou complexos.

Aprender a racionar orientado a objetos, bem como a utilizar corretamente as suas técnicas não é fácil, mas pode-se afirmar que é essencial para qualquer profissional da área atualmente.

A Orientação a Objeto é um paradigma de análise, projeto e programação de sistemas de informação automatizados baseado na composição e interação entre diversas unidades de software chamadas objetos.

Mas o que é Objeto? De acordo com o dicionário: - Objeto: "1. Tudo que se oferece aos nossos sentidos ou à nossa alma. 2. Coisa material: Havia na estante vários objetos. 3. Tudo que constitui a matéria de ciências ou artes. 4. Assunto, matéria. 5. Fim a que se mira ou que se tem em vista".



A figura acima destaca uma série de objetos. Objetos podem ser não só coisas concretas como também coisas inanimadas, como por exemplo uma matrícula, as disciplinas de um curso, os horários de aula.

Na programação orientada a objetos, implementa-se um conjunto de **classes** que definem os **objetos** presentes no *software*. Cada classe possui um **comportamento** (definidos pelos **métodos**) e **estados** possíveis (valores dos **atributos**) de seus objetos, assim como o **relacionamento** com outros objetos.

Simula (a 1ª. Linguagem OO conhecida), Smalltalk (a 1ª. Linguagem OO a ganhar destaque), Perl, Python, PHP, Java, C++ e C# são algumas das linguagens de programação mais importantes com suporte a orientação a objetos.

1.1.1 A PROPOSTA DA ORIENTAÇÃO A OBJETOS

A proposta da Orientação e Objetos é representar o mais fielmente possível as situações do mundo real nos sistemas computacionais. Nós entendemos o mundo como um todo composto por vários objetos que interagem uns com os outros. Da mesma maneira, a orientação a objetos consiste em considerar os sistemas computacionais não como uma coleção estruturada de processos, mas sim como uma coleção de objetos que interagem entre si (Origem: http://72.14.209.104/search?q=cache:Ba1uGSi5Tz0J:ftp.faculdadesnda.com.br/lcarvalho/20112%2520-%2520Linguagem%2520de%2520Programa%25E7%25E3o%2520III/APOSTILAS/Livro%2520AOO-Pascal.pdf+compara%C3%A7%C3%A3o+abordagem+diferen%C3%A7as+estruturada+an%C3%A1lise+prop%C3%B3sito+%22orienta%C3%A7%C3%A3o+a+objetos%22&hl=pt-BR&gl=br&ct=clnk&cd=46)

1.1.2 OS PRINCÍPIOS DA ORIENTAÇÃO A OBJETOS

Origem: Artigo Paradigma da Orientação a Objetos
Autor: Alessandro F L (lecadf)

Local: http://www.forumweb.com.br/artigos
Data: Sexta-feira, 14 de Abril de 2006

[...]

Há alguns anos, Alan Kay, um dos pais do paradigma da orientação a objetos, formulou a chamada "analogia biológica". Nessa analogia ele imaginou como seria um sistema de software que funcionasse como um ser vivo, no qual cada "célula" interagiria com outras células através do envio de mensagens para realização do objetivo comum.

De uma forma mais geral, Kay pensou em como construir um sistema de software a partir de agentes autônomos que interagem entre si. Ele então estabeleceu os seguintes princípios da orientação a objetos:[1]

(a) Qualquer coisa é um objeto;

(b) Objetos realizam tarefas através da requisição de serviços a outros objetos;

(c) Cada objeto pertence a uma determinada classe. Uma classe agrupa objetos similares;

(d) A classe é um repositório para comportamento associado ao objeto;

(e) Classes são organizadas em hierarquias.

[...]

1.1.3 VANTAGENS / BENEFÍCIOS

A técnica da orientação a objetos apresenta uma série de vantagens, cada vez mais necessárias no atual desenvolvimento de sistemas computacionais. Entre tais vantagens podem ser destacadas as seguintes:

- Gap Semântico reduzido – o mundo real é composto por objetos, e o mesmo ocorre com o mundo computacional, facilitando assim a compreensão;
- Centralização de dados e funções em uma única unidade, o objeto;
- Reutilização – o que permite um ganho significativo de tempo e custo. Aqui há destaque para o conceito de herança. No desenvolvimento de *softwares* o uso de tal conceito reduz custo e tempo;
- Modularização – não só no que se refere aos pacotes, mas principalmente ao tratamento de classes;
- Compatibilidade – os modelos desenvolvidos (análise, projeto e programação) são claramente relacionados e complementares. Fica destacada aqui a importância crescente da documentação de um *software*;
- Manutenibilidade – as classes centralizam os dados e funções daquilo que tratam, ficando mais fácil localizar os pontos necessários em uma manutenção, evitando parcialmente o verdadeiro efeito "bola de neve" que uma manutenção mal planejada pode causar. E principalmente, a possibilidade de trabalhar com herança, que centraliza ainda mais as características comuns que as classes compartilharem;
- Ocultamento de Informação – uma classe enxerga apenas a interface de outra classe, o que evita vários erros acidentais.

Origem: Livro: Análise Orientada a Objetos, Prof. Dr. Malcon A. Tafner, Prof. Carlos H. Correa

OS BENEFÍCIOS DA ORIENTAÇÃO A OBJETOS

A técnica de análise e programação orientada a objetos está se tornando cada vez mais popular entre os desenvolvedores de sistema. Essa popularidade não é fruto do acaso, ou da moda, e sim das vantagens que os desenvolvedores passam a usufruir quando adotam essa metodologia.

A principal vantagem da orientação a objetos consiste em reunir em uma mesma estrutura os dados e os processos que são executados sobre esses dados, permitindo assim um maior grau de organização e simplicidade do programa.

A partir do momento em que um desenvolvedor cria um objeto para realizar uma tarefa complexa, os outros desenvolvedores só precisam acessar esse objeto para realizar essa mesma tarefa, sem precisar saber como elas são realizadas. Se alguém muda a maneira como esse objeto realiza internamente essa tarefa, os outros desenvolvedores não precisam alterar os seus programas para continuar acessando o novo comportamento, e isso representa um ganho real de produtividade.

O procedimento de manutenção restrito ao objeto reduz custo, tempo e assegura a certeza de uma manutenção eficaz, visto que as variáveis controladas estavam restritas aos objetos modificados, e mais em nenhuma outra parte do sistema.

Uma estrutura eficaz de objetos, uma vez criada, permitirá incluir módulos adicionais no sistema que reutilizem as funcionalidades já desenvolvidas sem necessidade de reprogramação. Reutilizar não é duplicar código e alterar. Significa que o mesmo código pode ser usado por sistemas diferentes, reduzindo o custo de desenvolvimento e o risco de cometer erros de programação, ao mesmo tempo que não será necessário alterar trechos, o que poderia introduzir novos erros no código.

Por exemplo, uma empresa desenvolve um sistema educacional que tenha sua funcionalidade de cobrança baseada em objetos. A mesma empresa pode desenvolver um sistema imobiliário e reutilizar objetos de cobrança , incluindo-os no novo sistema. Se o autor do sistema educacional fizer aperfeiçoamentos ou correções nos objetos de cobrança, essas alterações terão sido realizadas de uma vez em todos os sistemas da empresa.

Dentro de uma visão orientada a objetos, quando precisamos criar um sistema de informações, precisamos, antes de tudo, identificar quais são os objetos necessários para operacionalizar esse sistema. Uma vez definidos os objetos necessários, e antes mesmo de partir para a criação destes, podemos partir para a reutilização de objetos já existentes, ou seja, verificar em outros sistemas se já não existem objetos que realizam as tarefas desejadas. Se existirem objetos que podem ser reutilizados , estes podem ser incluídos no novo sistema sem a necessidade e o custo de criação. Outra opção também pode ser a transformação de objetos semelhantes, ou seja, objetos cujas características são bastante semelhantes podem, com pequenas modificações, serem úteis ao novo sistema.

Para a mudança de um sistema, é válida a mesma proposta de flexibilidade, podendo-se adaptar, excluir, ou mesmo incluir novos objetos rapidamente, com a mesma garantia de ter atingido todo o universo necessário dentro do sistema para assegurar a eficácia da modificação.

O desenvolvimento orientado a objetos é um método de desenvolvimento cujo objetivo é proporcionar o máximo possível de reutilização. É também um método que simplifica a montagem de novos sistemas, uma vez que esse método trata do manuseio de objetos pré-prontos.

1.2 COMPARAÇÃO ENTRE AS TÉCNICAS ESTRUTURADA E ORIENTADA A OBJETOS

1.2.1 HISTÓRICO – SURGIMENTO DAS TÉCNICAS ESTRUTURADA E ORIENTADA A OBJETOS

Origem: Manual de Referência – Técnicas e Projeto de Sistemas, Prof. M.Sc. Gerson Pesente Fockin, Escola Técnica federal de Palmas, Curso Técnico em Informática (lab.etfto.gov.br/~focking/manual_referencia.doc)

...

Até o início da década de 70, o computador era utilizado somente por grandes empresas. Neste período, com a queda do preço dos computadores e a consequente proliferação do uso destes, cresceu a demanda por software.

As técnicas de desenvolvimento de software utilizadas até então não eram suficientes para contornar os problemas existentes no desenvolvimento de sistemas, principalmente quando desenvolvidos em grande escala, como então se exigia. Na verdade, pouco se possuía de técnicas que estivessem realmente sendo aplicadas.

Foi neste contexto que surgiu a **programação estruturada**, seguida pelo conceito de **desenvolvimento estruturado de sistemas**. Esta modelagem tentava oferecer soluções para os problemas ligados ao desenvolvimento de sistemas, ao pregar a aplicação dos seguintes princípios:

1 - Princípio da Abstração*:* para resolver um problema, o analista deveria analisá-lo separadamente dos demais, ou seja, abstrair os detalhes;

2 - Princípio da Formalidade*:* o analista deveria seguir um caminho rigoroso e metódico para solucionar um problema;

3 - Princípio de "dividir para conquistar*":* o analista deveria dividir o problema em partes menores, independentes e com possibilidade de serem mais simples de entender e solucionar;

4 - Princípio da Disposição Hierárquica*:* o analista deveria organizar os componentes da solução do problema na forma de uma árvore com estrutura hierárquica. O sistema seria entendido e construído nível a nível, onde cada novo nível acrescentaria mais detalhes.

Estas técnicas tiveram uma rápida disseminação, sendo hoje conhecidas pela grande maioria dos que trabalham na área.

Estamos agora numa situação semelhante àquela que se apresentava na década de 70. A disseminação do uso do computador se tornou muito grande e a todo instante surgem novas tecnologias de hardware, que impulsionam os preços para baixo e nos abrem horizontes para novas aplicações, assim como novas perspectivas para aplicações atuais.

As novas aplicações se caracterizam por:

- Grande interação com o usuário;
- Uso de interfaces gráficas (GUI) como o Windows;
- Necessidade permanente de alteração e expansão, dada a velocidade de mudanças na tecnologia de hardware;
- Interação com outros sistemas, possibilitando a troca de dados entre estes;
- Portabilidade para diversas plataformas e sistemas operacionais.

As técnicas oferecidas pela programação estruturada superavam com certa dificuldade as complexidades envolvidas no desenvolvimento deste tipo de sistemas. Demandava-se novas técnicas.

Foi neste contexto que retornou-se a programação orientada a objetos. Esta metodologia não é recente, mas somente agora vislumbrou-se sua potencialidade. Isto porque a orientação a objetos, devido as suas características, reduz as dificuldades de desenvolvimento:

- A linguagem do analista e do usuário passam a ser semelhantes, por se referirem a objetos do mundo real;
- Pode-se utilizar um mesmo objeto em diferentes sistemas, aumentando a produtividade do processo de desenvolvimento;
- Se a biblioteca de objetos for independente do sistema operacional utilizado, ao se mudar a plataforma ou o sistema operacional, basta recompilar o programa.

A metodologia de objetos vem oferecer uma solução alternativa para o desenvolvimento de sistemas, e significa uma grande evolução desde a programação estruturada.

É uma evolução porque apesar da metodologia de orientação a objetos também utilizar os mesmos princípios da programação estruturada (abstração, hierarquização, decomposição) acrescenta novos princípios, como objetos, classes, herança, etc.

[...]

1.2.2 ABORDAGEM ESTRUTURADA X ABORDAGEM ORIENTADA A OBJETOS

Origem: Prof. Ricardo de Almeida Falbo, Apostila de Análise de Sistemas - Notas de Aula, 2002, UFES

Uma vez que, atualmente, a Orientação a Objetos tem tomado o espaço antes ocupado pelo paradigma estruturado no desenvolvimento de sistemas, é interessante fazer uma comparação entre os paradigmas que fundamentam estas abordagens:

- **Estruturado:** adota uma visão de desenvolvimento baseada em um modelo entrada-processamento-saída. No paradigma estruturado, os dados são considerados separadamente das funções que os transformam e a decomposição funcional é usada intensamente.
- **Orientado a Objetos:** pressupõe que o mundo é composto por objetos, onde um objeto é uma entidade que combina estrutura de dados e comportamento funcional. No paradigma orientado a objetos, os sistemas são estruturados a partir dos objetos que existem no domínio do problema, isto é, os sistemas são modelados como um número de objetos que interagem.

Em função do paradigma que os rege, métodos de análise e projeto de sistemas são classificados em métodos estruturados e métodos orientados a objetos.

Métodos Estruturados

Fazem clara distinção entre funções e dados. Funções, a princípio, são ativas e têm comportamento, enquanto dados são repositórios passivos de informação, afetados por funções. Esta divisão tem origem na arquitetura de hardware de von Neumann, onde a separação entre programa e dados é fortemente enfatizada.

Os **métodos orientados a funções** conduzem o desenvolvimento de software estruturando as aplicações segundo a ótica das funções (ações) que o sistema deverá realizar. O sistema é decomposto em funções, e os dados são transportados entre elas. Esta é a filosofia da proposta original da Análise Estruturada [DeMarco78] [Gane79], cuja ferramenta básica de modelagem são os diagramas de fluxo de dados (DFDs).

Os **métodos orientados a dados**, por sua vez, enfatizam a identificação e estruturação dos dados, subjulgando a análise das funções para um segundo plano. Esses métodos têm origem no projeto de bancos de dados e, geralmente, têm no modelo de Entidades e Relacionamentos (ER) [Chen79] sua principal ferramenta.

A ênfase nas funções, geralmente, leva a sistemas com muita redundância e, conseqüentemente, inconsistentes e difíceis de serem integrados. Por outro lado, a ênfase nos dados está fundamentada em dois fatores significativos:

- Dados possuem existência própria nas organizações independentemente dos processos que os manipulam.
- Dados são muito mais estáveis que as funções em uma organização. A menos que haja grandes mudanças nos negócios de uma empresa, os dados tendem a se manter estáveis.

Assim, é possível desenvolver modelos de dados sem redundância, sem inconsistência e fáceis de integrar. Entretanto, uma vez que o modelo de dados deve representar a realidade, e o conhecimento da realidade, muitas vezes, passa pelo conhecimento das funções, ele deve ser construído de forma iterativa, não podendo ser considerado um produto acabado.

A Análise Essencial [Pompilho95] procurou conciliar as abordagens orientadas a dados e a funções em um único método, utilizando modelos para dados, funções e controles (DFDs e Modelo ER e Diagramas de Transição de Estados, respectivamente) como ferramentas para a modelagem de sistemas.

Um sistema desenvolvido usando um método estruturado, frequentemente, é difícil de ser mantido. A princípio, o problema principal advém do fato de todas as funções terem de conhecer como os dados estão armazenados, isto é, a estrutura dos dados. Além disso, mudanças na

estrutura dos dados quase sempre acarretam modificações em todas as funções relacionadas a essa estrutura. Em suma, a interpretação dos dados é apenas implícita, provida pelos programas que lêem ou escrevem dados. Diferentes programas podem dar diferentes interpretações aos dados e, portanto, é necessário conhecer como eles foram projetados para poder interpretá-los corretamente [Snyder93].

Métodos Orientados a Objetos

Os métodos orientados a objetos partem de um ponto de vista distinto e intermediário, onde se pressupõe que o mundo real é povoado por objetos, onde um objeto é uma entidade que combina estrutura de dados e comportamento funcional. Métodos orientados a objetos estruturam os sistemas a partir dos objetos que existem no domínio do problema.

A orientação a objetos oferece um número de conceitos bastante apropriados para a modelagem de sistemas. Utilizando a orientação a objetos como base, os sistemas são modelados como um número de objetos que interagem. Os modelos baseados em objetos são úteis para a compreensão de problemas, para a comunicação com os especialistas e usuários das aplicações, e para a realização das tarefas ao longo do ciclo de desenvolvimento de software. Os principais objetivos da orientação a objetos são:

- Diminuir a distância conceitual entre o mundo real (domínio do problema) e o modelo abstrato de solução (domínio da solução);
- Trabalhar com noções intuitivas (objetos e ações) durante todo o ciclo de vida, atrasando, ao máximo, a introdução de conceitos de implementação.

Normalmente, esta é uma maneira mais natural para descrever sistemas, já que os objetos são geralmente bastante estáveis. Alterações que por ventura venham a ocorrer, geralmente, afetam um ou alguns poucos objetos [Jacobson92].

Eduard Yourdon [Yourdon94] dá um bom resumo do que pode ser considerado um produto orientado a objeto: “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.”

Métodos orientados a objetos utilizam uma perspectiva mais humana de observação da realidade, incluindo objetos, classificação e compreensão hierárquica. São benefícios esperados com o uso da orientação a objetos:

1. Capacidade de enfrentar novos domínios de aplicação;
2. Melhoria da interação entre analistas e especialistas;
3. Aumento da consistência interna dos resultados da análise;
4. Uso de uma representação básica consistente para a análise e projeto;
5. Alterabilidade, legibilidade e extensibilidade;
6. Possibilidade de ciclos de desenvolvimento variados;
7. Apoio à reutilização.

É importante enfatizar, no entanto, que a orientação a objetos não é mágica, isto é, ela não é uma nova “tábua de salvação” para eliminar os problemas de produtividade e qualidade que têm atormentado a indústria de software ao longo das últimas décadas. Se praticada cuidadosamente, combinada com várias outras técnicas de Engenharia de Software – tais como, uso de métricas, reutilização, testes, e garantia da qualidade - a orientação a objetos pode ajudar a levar a melhorias substanciais no desempenho de uma organização de software [Yourdon94]. Portanto, é imprescindível, para grandes projetos, a definição de um processo de desenvolvimento que garanta o uso consistente dessas técnicas e que seja apoiado por ferramentas computacionais, tais como ferramentas CASE e Ambientes de Desenvolvimento de Software.

1.2.3 PROGRAMAÇÃO ESTRUTURADA X PROGRAMAÇÃO ORIENTADA A OBJETOS

Origem: Notas de Aula, Introdução à Programação Orientada a Objetos, prof. Ulysses de Oliveira, 2002
Com breves adaptações.

Na tabela abaixo está apresentada uma comparação de algumas características que diferenciam as programações algorítmicas (procedimental ou estruturada) e a orientada a objeto.

Programação Algorítmica (Procedimental)	**Programação OO**
• Ênfase: construção de algoritmos; • Utiliza abordagem de refinamentos sucessivos; • Subproblemas são codificados como unidades denominadas procedimentos, subrotinas ou funções; • Unidades de programa podem ser agrupadas em módulos; • Programa resultante consiste de uma coleção de unidades que se comunicam entre si; • Visão *Top-down*; • Forte uso de Decomposição Funcional; • O sistema é composto por dados e funções, tratados separadamente, mas que podem interagir. Problemas: • Programas muito grandes tornam-se muito complexos e difíceis de entender e manter; • Dificuldade em modelar muitos problemas da vida real com enfoque em algoritmos; • É mais fácil simular o funcionamento de sistemas complexos com enfoque em suas partes constituintes do que em termos dos algoritmos utilizados pelo sistema. Exemplo: um automóvel é melhor entendido em termos de direção, freios, etc. do que em termos dos algoritmos que o fazem funcionar; • Linguagens procedimentais não oferecem facilidades para criação de novos tipos de dados que funcionem como os tipos de dados primitivos.	• Metodologia desenvolvida objetivando suprir deficiências encontradas em programação algorítmica; • Encapsulamento: combinação de dados (atributos) e funções (métodos) numa única entidade de programa; • Objeto: entidade de encapsulamento; • Um programa no paradigma OO consiste de vários objetos que se comunicam (isto é, trocam mensagens) utilizando seus métodos constituintes; • Visão de Objetos cooperativos; • As características de comportamento e informações são modeladas de maneira fortemente relacionadas; • O sistema é composto por objetos, que contém dados e funções (isto é, estão reunidos em um só elemento). Ocultação de Informação: • Métodos que fazem parte de um objeto provêem (usualmente) a única forma de acesso aos seus dados; • Campos de um objeto não podem ser acessados diretamente, apenas indiretamente por meio de seus métodos constituintes; • Previne alterações acidentais de dados e facilita a manutenção e depuração dos programas.

1.3 DICAS DE LEITURA

- Livro Modelagem de Objetos através da UML, Autor: José Davi Furlan, capítulo 1 – Rumo à Orientação a Objetos, tópico 1 – O meio empresarial em perspectiva;
- Artigo - Do diagrama de Fluxo de Dados ao Use case. Autores: Alessandro Botelho Bovo, Renato Balancieri, Sandra Ferrari;
- Texto – Programação Orientada a Objetos: uma Introdução. Autor: Marcio Frayze David. Site: http://www.guiadohardware.net/artigos/programacao-orientada-objetos/;
- Texto – O que é a programação Orientada a Objetos. Autor: Miguel Angel Alvarez. Site: http://www.criarweb.com/artigos/215.php.

2 CONCEITOS DA ORIENTAÇÃO A OBJETOS

Origem: (1) Apostila de Análise de Sistemas, prof. Ricardo Falbo (UFES), capítulo 04 – Introdução à Orientação a Objetos, tópico 4.2 – Conceitos da Orientação a Objetos; (2) Notas de Aula, prof. Denise Franzoti Togneri (FAESA), Princípios e Conceitos da OO.
Com breves adaptações.

O mundo real é extremamente complexo. Tal complexidade pode ser percebida a medida que analisamos os detalhes de tal mundo, mesmo que nos concentremos em algum objeto em particular. Essa característica é transferida para o mundo computacional, quando da criação de uma solução computadorizada para um problema do mundo real. Alguns dos elementos fundamentais na geração desta complexidade são:

- A complexidade do próprio domínio do problema;
- A dificuldade no gerenciamento do processo de desenvolvimento do sistema necessário;
- O leque de variadas soluções que podem ser encontradas no projeto do sistema.

O paradigma orientado a objeto procura administrar essa complexidade através de uma série de conceitos, tais como abstração, encapsulamento, modularização e hierarquia.

Os principais conceitos associados à Orientação a Objetos são abaixo definidos.

2.1 ABSTRAÇÃO

Uma das principais formas do ser humano lidar com a complexidade é através do uso de abstrações. As pessoas tipicamente tentam compreender o mundo, construindo modelos mentais de partes dele. Tais modelos são uma visão simplificada de algo, onde apenas elementos relevantes são considerados. Modelos mentais, portanto, são mais simples do que os complexos sistemas que eles modelam.

Consideremos, por exemplo, um mapa como um modelo do território que ele representa. Um mapa é útil porque abstrai apenas aquelas características do território que se deseja modelar. Se um mapa incluísse todos os detalhes do território, provavelmente teria o mesmo tamanho do território e, portanto, não serviria a seu propósito.

Da mesma forma que um mapa precisa ser significativamente menor que o território que mapeia, incluindo apenas informações cuidadosamente selecionadas, um modelo mental abstrai apenas as características relevantes de um sistema para seu entendimento. Assim, podemos definir abstração como sendo o princípio de ignorar aspectos não relevantes de um assunto, segundo a perspectiva de um observador, tornando possível uma concentração maior nos aspectos principais do mesmo. De fato, a abstração consiste na seleção que um observador faz de alguns aspectos de um assunto, em detrimento de outros que não demonstram ser relevantes para o propósito em questão, isto é, a abstração é aplicada de acordo com o interesse do observador, e por isso de um mesmo objeto pode-se ter diferentes visões, como demonstra a figura 1 abaixo.

Só devem ser mapeados os objetos que são relevantes ao problema, bem como as características (propriedades e comportamento) desses objetos que forem necessários.

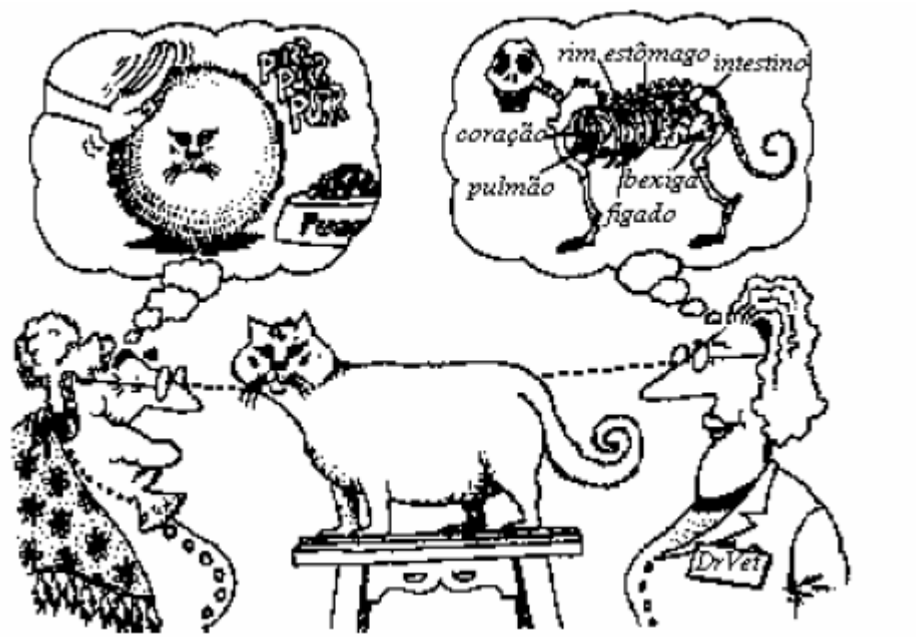


Figura 4.1 - A abstração enfoca as características essenciais de um objeto [Booch94].

No que tange ao desenvolvimento de software, duas formas adicionais de abstração têm grande importância: a abstração de dados e a abstração de procedimentos.

Abstração de Dados

Consiste em definir um tipo de dado conforme as operações aplicáveis aos objetos deste tipo. Os objetos só podem ser modificados e observados através dessas operações [Coad92].

Exemplo: Um tipo de dado **`pilha`** pode ser definido através das operações `empilhar`, isto é, colocar um elemento no topo da pilha, e `desempilhar`, ou seja, retirar o elemento que está no topo da pilha. Um objeto do tipo `pilha` só pode ser modificado e observado através dessas duas operações.

Abstração de Procedimentos

Segundo esse princípio, uma operação com um efeito bem definido pode ser tratada por seus usuários como uma entidade única, mesmo que a operação seja realmente conseguida através de alguma sequência de operações de nível mais baixo.

Exemplo: Seja a operação `calcular-salário-líquido` de um objeto do tipo `funcionário`. Essa operação pode ser tratada por seus usuários como uma entidade única, mesmo que ela seja, na realidade, construída como uma sequência de operações de nível mais baixo, tais como: `calcular-INSS`, `calcular-IR`, `calcular-anuênio`, etc.

2.2 ENCAPSULAMENTO

No mundo real, um objeto pode interagir com outro sem conhecer seu funcionamento interno. Uma pessoa, por exemplo, geralmente utiliza uma televisão sem saber efetivamente qual a sua estrutura interna ou como seus mecanismos internos são ativados. Para utilizá-la, basta saber realizar algumas operações básicas, tais como ligar/desligar a TV, mudar de um canal para outro, regular volume, cor, etc. Como estas operações produzem seus resultados, mostrando um programa na tela, não interessa ao telespectador.

O encapsulamento consiste na separação dos aspectos externos de um objeto, acessíveis por outros objetos, de seus detalhes internos de implementação, que ficam ocultos dos demais objetos [Rumbaugh94]. A interface de comunicação de um objeto deve ser definida de forma a revelar o menos possível sobre o seu funcionamento interno.

Os usuários tem conhecimento apenas das operações que podem ser realizadas e precisam estar cientes apenas do QUE as operações fazem, e não COMO elas estão implementadas. [Booch94]

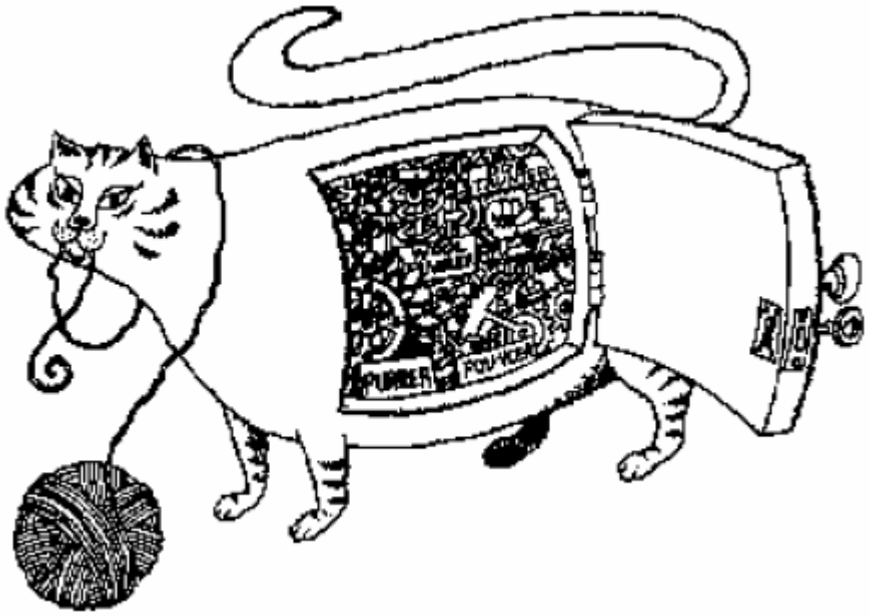


Figura 4.2 - O encapsulamento oculta os detalhes de implementação de um objeto [Booch94].

Abstração x Encapsulamento

Abstração e Encapsulamento são conceitos complementares: enquanto a abstração enfoca o comportamento observável de um objeto (o que se deve mapear), o encapsulamento enfoca a implementação que origina esse comportamento (como realizar a abstração). Encapsulamento é frequentemente conseguido através do **ocultamento de informação**, isto é, escondendo detalhes que não contribuem para suas características essenciais. Tipicamente, em um sistema orientado a objetos, a estrutura de um objeto, e a implementação de seus métodos, são encapsuladas [Booch94].

Por exemplo, para usar um carro, uma pessoa não precisa conhecer sua estrutura interna (motor, caixa de marcha, etc...), nem tampouco como se dá a implementação de seus métodos. Sabe-se que é necessário ligar o carro, mas não é preciso saber como esta operação é implementada. Assim, sobre carros, um motorista precisa conhecer apenas as operações que permitem utilizá-lo, a que chamamos de **interface do objeto**, o que inclui a ativação de operações, tais como ligar, mudar as marchas, acelerar, frear, etc..., e não como essas operações são de fato implementadas.

Encapsulamento serve para separar a interface contratual de uma abstração e sua implementação. Os usuários têm conhecimento apenas das operações que podem ser requisitadas e precisam estar cientes apenas do **que** as operações realizam e não **como** elas estão implementadas.

A principal motivação para o encapsulamento é facilitar a reutilização de objetos e garantir estabilidade aos sistemas. Um encapsulamento bem feito pode servir de base para a localização de decisões de projeto que necessitam ser alteradas. Uma operação pode ter sido implementada de maneira ineficiente e, portanto, pode ser necessário escolher um novo algoritmo. Se a operação está encapsulada, apenas o objeto que a define precisa ser modificado, garantindo estabilidade ao sistema.

2.3 MODULARIDADE / DECOMPOSIÇÃO

Muitos métodos de construção de software buscam obter sistemas modulares, isto é, construídos a partir de elementos que sejam autônomos, conectados por uma estrutura simples e coerente. Modularidade é crucial para se obter manutenibilidade, reusabilidade e extensibilidade, com consequente redução de custos.

Modularidade é uma propriedade de sistemas decompostos em um conjunto de módulos coesos (elementos internos logicamente relacionados) e fracamente acoplados (dependência entre os módulos). Assim, abstração, encapsulamento e modularidade são princípios sinergéticos. Um objeto provê uma fronteira clara em torno de uma abstração e o encapsulamento e a modularidade provêem barreiras em torno dessa abstração [Booch94].

2.4 HIERARQUIA

O conceito de Herança na Orientação a Objetos lembra o conceito de herança genética, na qual um elemento descendente herda as características de um elemento ancestral.

Abstração é um princípio importantíssimo, mas em todas as aplicações, exceto aquelas mais triviais, deparamo-nos com um número de abstrações maior do que conseguimos compreender em um dado momento. O encapsulamento ajuda a gerenciar esta complexidade através do ocultamento da visão interna de nossas abstrações. Modularidade auxilia também, dando-nos um meio de agrupar logicamente abstrações relacionadas. Entretanto, isto ainda não é o bastante. Um conjunto de abstrações freqüentemente forma uma hierarquia e, pela identificação dessas hierarquias, é possível simplificar significativamente o entendimento sobre um problema [Booch94]. Em suma, hierarquia é uma forma de arrumar as abstrações.

2.5 OBJETO

O mundo real é povoado por elementos que interagem entre si, onde cada um deles desempenha um papel específico. A esses elementos, chamamos **objetos**. Objetos podem ser coisas concretas ou abstratas, tais como um carro, uma reserva de passagem aérea, uma organização, uma planta de engenharia, um componente de uma planta de engenharia, etc.

Os objetos podem ser (Pressman, 2002):

- **Entidades externas**: (isto é, outros sistemas, dispositivos, pessoas) que produzem ou consomem informação para ser usada através de um sistema baseado em computador;
- **Coisas**: (isto é, relatórios, displays, sinais, letras) que são partes do domínio da informação para o problema;
- **Ocorrências ou eventos** (isto é, uma transferência de propriedade ou o aspecto dos movimentos de robôs) que ocorrem dentro do contexto da operação do sistema;
- **Papéis** (isto é, gerente, engenheiro, vendedor) executados por pessoas que interagem com o sistema;
- **Unidades organizacionais** (isto é, divisões, grupos, times) que são relevantes para a aplicação;
- **Locais** (isto é, chão de fábrica, locais de carga) que estabelecem o contexto do problema e as funções gerais do sistema;
- **Estruturas** (isto é, sensores, computadores, veículos) que definem uma classe de objetos ou, no extremo, classes relacionadas de objetos.

Do ponto de vista da modelagem de sistemas, um objeto é uma entidade que incorpora uma abstração relevante no contexto de uma aplicação. Um objeto possui um **estado** (informação),

exibe um **comportamento** bem definido, expresso por um número de operações para examinar ou alterar seu estado, e tem **identidade** única. Ou seja, um objeto é algo que precisamos representar computacionalmente e que tem propriedades próprias, um comportamento específico e uma identidade única.

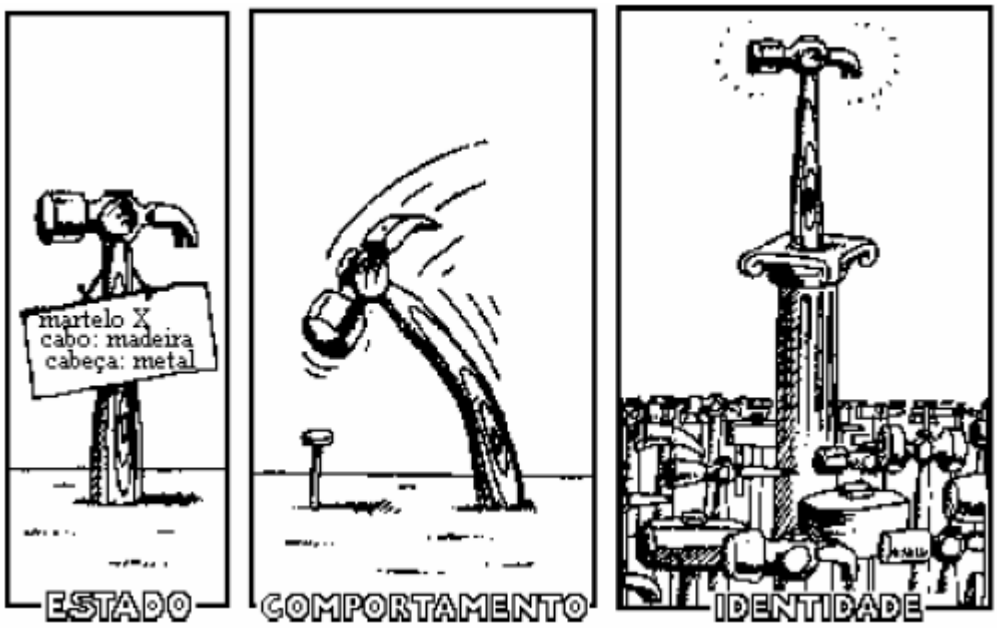


Figura 4.3 - Um objeto possui estado, exibe algum comportamento bem definido e possui identidade própria [Booch94].

Estado - O estado de um objeto compreende o conjunto de suas propriedades, associadas a seus valores correntes. Propriedades de objetos são geralmente referenciadas como atributos e, portanto, o estado de um objeto diz respeito aos seus atributos e aos valores a eles associados.

Comportamento - A abstração incorporada por um objeto é caracterizada por um conjunto de serviços ou operações, que outros objetos, ditos clientes, podem requisitar. Operações são usadas para recuperar ou manipular a informação de estado de um objeto e referem-se apenas às estruturas de dados do próprio objeto, não devendo acessar diretamente estruturas de outros objetos.

A comunicação entre objetos dá-se por meio de **troca de mensagens**. Para acessar a informação de estado de um objeto, é necessário enviar uma mensagem para ele. Uma mensagem consiste do nome de uma operação e os argumentos requeridos. Assim, o comportamento de um objeto representa como este objeto reage às mensagens a ele enviadas.

Em outras palavras, o conjunto de mensagens a que um objeto pode responder representa o seu comportamento.

Um objeto é, pois, uma entidade que tem seu estado representado por um conjunto de atributos (uma estrutura de informação) e seu comportamento representado por um conjunto de operações.

Identidade - Cada objeto tem uma identidade própria, que lhe é inerente. Todos os objetos têm existência própria, ou seja, dois objetos são distintos mesmo se seu estado e comportamento forem iguais. A identidade de um objeto transcende os valores correntes de suas variáveis de estado (atributos). Identificar um objeto diretamente é geralmente mais eficiente que designá-lo pela sua descrição [Snyder93].

No mundo real, um objeto limita-se a existir, mas, no que se refere ao mundo computacional, cada objeto dispõe de um identificador único pelo qual pode ser referenciado inequivocamente [Rumbaugh94].

2.6 CLASSE E INSTÂNCIA

É bastante comum encontrarmos no mundo real, diferentes objetos desempenhando um mesmo papel. Consideremos, por exemplo, duas cadeiras. Apesar de serem objetos diferentes, elas compartilham uma mesma estrutura e um mesmo comportamento. Entretanto, não há necessidade de se despender tempo modelando as duas cadeiras, ou várias delas. Basta definir, em um único lugar, um modelo descrevendo a estrutura e o comportamento desses objetos. A esse modelo damos o nome de **classe**.

Uma classe descreve um conjunto de objetos com as mesmas propriedades (atributos), o mesmo comportamento (operações), os mesmos relacionamentos com outros objetos e a mesma semântica. Objetos que se comportam da maneira especificada pela classe são ditos **instâncias** dessa classe.

Todo objeto pertence a uma classe, ou seja, é instância de uma classe. De fato, a orientação a objetos norteia o processo de desenvolvimento através da **classificação de objetos**, isto é, objetos são agrupados em classes, em função de exibirem facetas similares, sem, no entanto, perda de sua individualidade. Assim, a modelagem orientada a objetos consiste, basicamente, na definição de classes. O comportamento e a estrutura de informação de uma instância são definidos pela sua classe.

Objetos com propriedades e comportamento idênticos são descritos como instâncias de uma mesma classe, de modo que a descrição de suas propriedades possa ser feita uma única vez, de forma concisa, independentemente do número de objetos que tenham tais propriedades em comum. Deste modo, uma classe captura a semântica das características comuns a todas as suas instâncias.

Enquanto um objeto individual é uma entidade real, que executa algum papel no sistema como um todo, uma classe captura a estrutura e comportamento comum a todos os objetos que ela descreve. Assim, uma classe serve como uma espécie de contrato que deve ser estabelecido entre uma abstração e todos os seus clientes.

Resumindo: Classe é um conjunto de elementos com as mesmas características (atributos, operações, relacionamentos). Objeto é um elemento desse conjunto. Quando se considera diferentes objetos de uma mesma classe podem ocorrer de encontrarmos diferentes: valores, identificação, o resultado de um comportamento.

A identificação de classes e objetos envolve:

- **Descoberta**. Através da descoberta, é possível reconhecer as abstrações-chave e os mecanismos que formam o vocabulário do domínio do problema. A descoberta ocorre à medida que se compreende o problema;
- **Invenção**. Através da invenção, imaginam-se as abstrações generalizadas, bem como os novos mecanismos que especificam como os objetos colaboram entre si. A invenção ocorre à medida que se soluciona o problema.

O problema da classificação é um problema relacionado com encontrar semelhanças. A classificação ajuda a identificar generalizações, especializações e hierarquia de agregações entre as classes.

Classificação é um processo interativo e incremental, difícil de ser executado porque um determinado conjunto de objetos pode ser classificado igualmente, de várias maneiras.

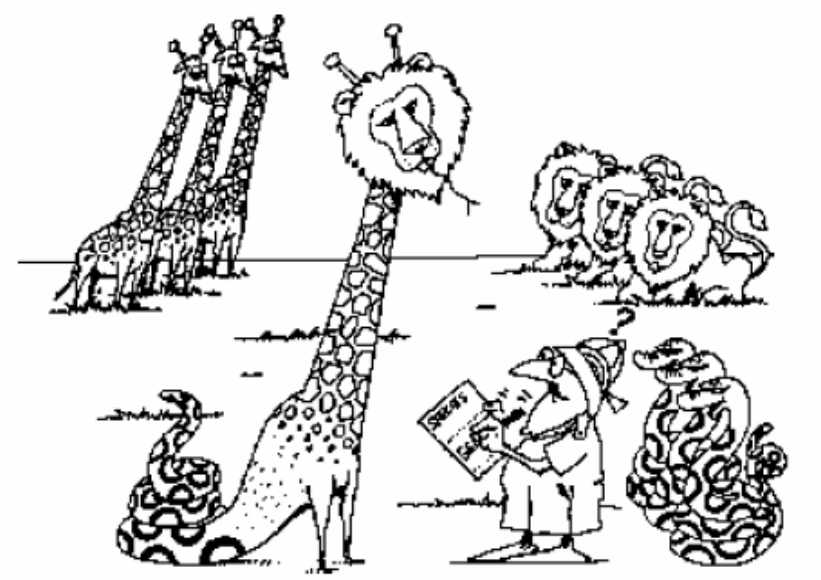


Figura 4.4 - Classificação é o meio pelo qual ordenamos conhecimento [Booch94].

Alguns autores utilizam os conceitos de classe e tipo indistintamente. Entretanto, um tipo e uma classe não são a mesma coisa. Um tipo é definido por um conjunto de operações, isto é, pelas manipulações que podemos fazer com o tipo. Uma classe é mais do que isso.

Podemos também olhar para dentro de uma classe, por exemplo, para ver sua estrutura de informação. Assim, uma classe é melhor conceituada como uma implementação específica de um tipo [Jacobson92].

2.7 MENSAGENS E MÉTODOS

A abstração incorporada por um objeto é caracterizada por um conjunto de operações que podem ser requisitadas por outros objetos, ditos clientes. Métodos são implementações reais de operações. Para que um objeto realize alguma tarefa, é necessário enviar a ele uma mensagem, solicitando a execução de um método específico. Um cliente só pode acessar um objeto através da emissão de mensagens, isto é, ele não pode acessar ou manipular diretamente os dados associados ao objeto. Os objetos podem ser complexos e o cliente não precisa tomar conhecimento de sua complexidade interna. O cliente precisa saber apenas como se comunicar com o objeto e como ele reage.

Embora existam diferentes tipos de operações, elas podem ser divididas em três categorias [Pressman02]:

- Operações que manipulam dados;
- Operações que executam cálculos computacionais;
- Operações que monitoram um objeto para ocorrência de um evento de controle.

As mensagens são o meio de comunicação entre objetos e são responsáveis pela ativação de todo e qualquer processamento. Dessa forma, é possível garantir que clientes não serão afetados por alterações nas implementações de um objeto que não alterem o comportamento esperado de seus serviços.

Uma mensagem estimula a ocorrência de algum comportamento no objeto receptor da mesma. O comportamento é estabelecido quando uma operação é executada.

O formato normalmente considerado na emissão de uma mensagem é:

Mensagem: [destino, método, parâmetros], onde:

Destino: o objeto receptor

Método: a operação que deve ser realizada

Parâmetros: As informações necessárias à execução do método

Resumindo:

- Mensagem: Mecanismo para disparar um método de uma classe diferente da corrente, isto é, são o meio através do qual os objetos interagem;
- Método: Uma ação a ser realizada.

2.8 MECANISMOS DE RELACIONAMENTO ENTRE CLASSES

Em qualquer sistema os objetos se comunicam, trocando mensagens. Em sistemas não triviais geralmente nos deparamos com uma quantidade razoável de objetos e classes e há necessidade de gerenciar tais elementos. Assim, é preciso estruturar as classes. Vários mecanismos tem sido propostos nesse sentido.

2.8.1 HIERARQUIA / HERANÇA / GENERALIZAÇÃO / ESPECIALIZAÇÃO

Muitas vezes, um conceito geral pode ser especializado, adicionando-se novas características. Tomemos, como exemplo, o conceito que temos de `estudantes`. De modo geral, há características que são intrínsecas a quaisquer estudantes. Entretanto, é possível especializar este conceito para mostrar especificidades de subtipos de estudantes, tais como `estudantes de 1º. grau`, `estudantes de 2º. grau`, `estudantes de graduação` e `estudantes de pós-graduação`, entre outros.

Da maneira inversa, pode-se extrair de um conjunto de conceitos, características comuns que, quando generalizadas, formam um conceito geral. Por exemplo, ao avaliarmos os conceitos que temos de `carros`, `motos`, `caminhões` e `ônibus`, podemos notar que esses têm características comuns que podem ser generalizadas em um supertipo `veículos automotores terrestres`.

As abstrações de especialização e generalização são muito úteis para a estruturação de sistemas. Com elas, é possível construir **hierarquias de classes**, subclasses, subsubclasses, e assim por diante.

A **herança** é um mecanismo para modelar similaridades entre classes, representando as abstrações de generalização e especialização. Através da herança, é possível tornar explícitos atributos e serviços comuns em uma hierarquia de classes. O mecanismo de herança possibilita reutilização, captura explícita de características comuns, definição incremental de classes, além de evitar a duplicação de características, e consequentemente, de código.

Frequentemente, promove-se a herança como a idéia central para a reutilização na indústria de software. Entretanto, apesar da herança, quando adequadamente usada, ser um mecanismo muito útil em diversos contextos, incluindo reuso, ela não é um pré-requisito para a reutilização, e nem mesmo para o desenvolvimento de um sistema orientado a objetos.

Uma das principais vantagens da herança é facilitar a modificação de modelos. A herança nos permite conceber uma nova classe como um refinamento de outras classes. A nova classe pode herdar as similaridades e definir apenas a funcionalidade nova. O desenvolvimento orientado a objetos é fortemente baseado na identificação de objetos e na construção de hierarquias de classes, utilizando o mecanismo de herança.

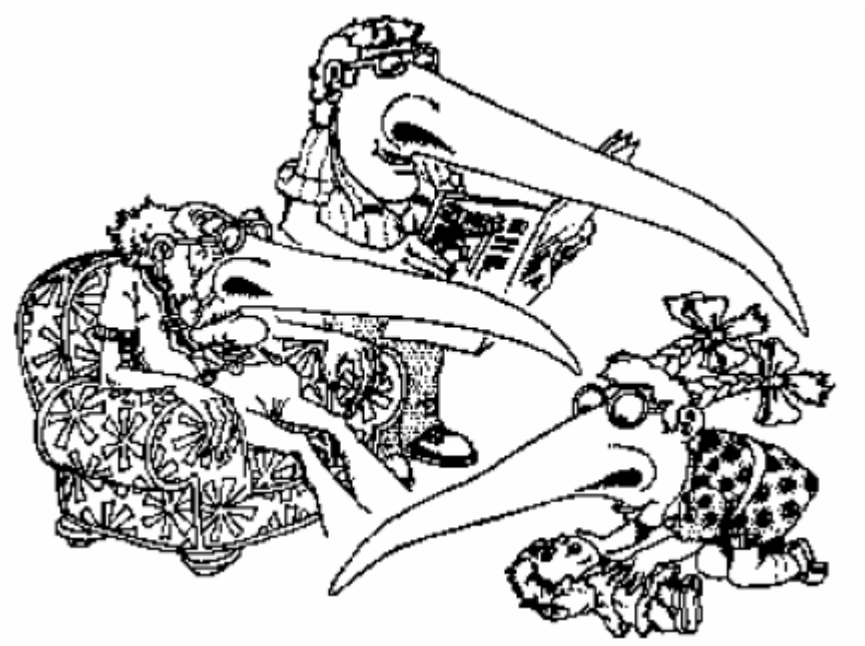


Figura 4.5 - Uma subclasse herda estrutura e comportamento de suas superclasses [Booch94].

Muitas vezes alguns objetos não se enquadram satisfatoriamente às propriedades descritas em uma classe, tipicamente porque:

- Além das propriedades descritas na classe, esses objetos possuem outras ainda não descritas;
- Algumas das propriedades descritas para a classe não são adequadas aos novos objetos, sendo necessário, portanto, redefini-las ou mesmo cancelá-las. Vale ressaltar que o cancelamento de propriedades, contudo, é um indicador de que a hierarquia não está modelada adequadamente.

Através do mecanismo de herança, tais problemas podem ser contornados. A herança define o relacionamento entre classes, no qual uma classe compartilha a estrutura, o comportamento e o relacionamento definidos em uma ou mais outras classes. A classe que herda características é chamada **subclasse** e a que fornece as características, **superclasse**. Desta forma, a herança representa uma hierarquia de abstrações na qual uma subclasse herda de uma ou mais superclasses.

Tipicamente, uma subclasse aumenta ou redefine características das superclasses. Assim, se uma classe `B` herda de uma classe `A`, todas as características descritas em `A` tornam-se automaticamente parte de `B`, que ainda é livre para acrescentar novas características para seus propósitos específicos.

A **generalização** permite abstrair, a partir de um conjunto de classes, uma classe mais geral contendo todas as características comuns. A **especialização** é a operação inversa e portanto, permite especializar uma classe em um número de subclasses, explicitando as diferenças entre as novas subclasses. Deste modo é possível compor a hierarquia de classes. Esses tipos de relacionamento são conhecidos também como relacionamentos "**é um tipo de**", onde um objeto da subclasse também "é um tipo de" objeto da superclasse. Neste caso uma instância da subclasse é dita uma **instância indireta** da superclasse.

Quando uma subclasse herda características de uma única superclasse, tem-se **herança simples**. Quando uma classe é definida a partir de duas ou mais superclasses, tem-se **herança múltipla**. É importante observar, no entanto, que na herança múltipla podem ocorrer dois problemas: colisão de nomes herdados a partir de diferentes superclasses e a possibilidade de herança repetida. A figura 4.6 ilustra estes dois casos.

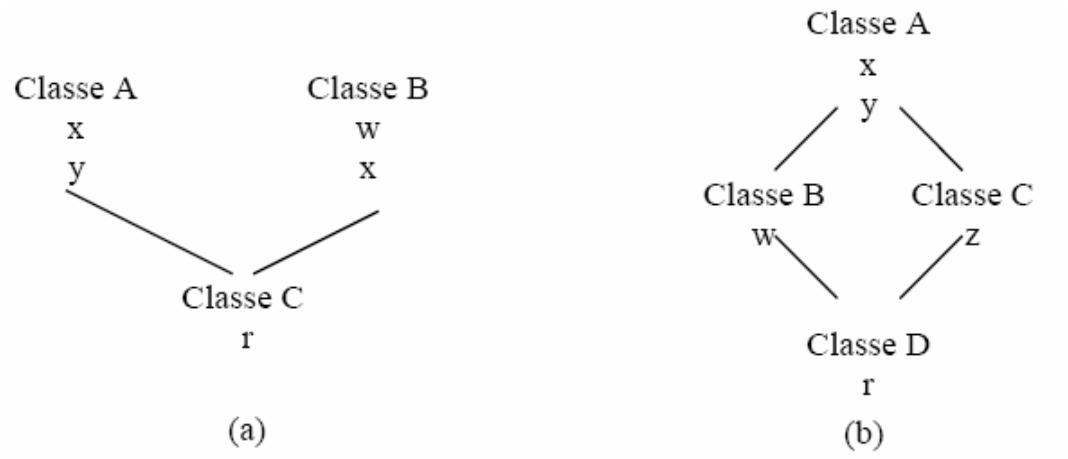


Figura 4.6 - (a) Colisão de nomes. (b) Herança repetida.

No primeiro caso, a classe `C` herda das classes `A` e `B`. Entretanto ambas possuem uma característica com nome `x`. Assim, como será a característica `x` em `C`, igual à definida na classe `A` ou igual à da classe `B`?

No segundo caso, a classe `D` herda das classes `B` e `C`, que por sua vez herdam da classe `A`. Assim, temos um caso de herança repetida, já que, indiretamente, a classe `D` herda duas vezes da classe `A`.

O resultado líquido da herança é que desenvolvedores podem evitar a codificação de redundâncias através da localização de cada operação no nível apropriado na hierarquia de classes.

Resumindo os conceitos complementares de hierarquia, herança, generalização e especialização:

- Hierarquia: Mecanismo para organização de classes que se relacionam de maneira estruturada e ordenada, em níveis;
- Herança: Mecanismo que permite que uma subclasse herde características de sua superclasse;
- Generalização: Mecanismo que permite extrair conceitos comuns de várias classes, posicionando-os em um local comum e compartilhado, a superclasse;
- Especialização: Mecanismo que permite especializar um conceito comum, refinando características, através de adição e/ou modificação de itens, criando diferentes subclasses.

2.8.2 COMPOSIÇÃO / AGREGAÇÃO

Uma forma especial de relacionamento entre objetos é a **composição** ou **agregação**. Parte do poder dos softwares orientados a objetos advém de sua habilidade de manipular objetos complexos, como sendo compostos de vários outros objetos mais simples. Um carro, por exemplo, é composto por motor, rodas, carroceria, etc... Um motor, por sua vez, é composto de bloco, válvulas, pistões, e assim por diante.

Composição é o relacionamento **todo-parte/uma-parte-de** onde os objetos representando os componentes de alguma coisa são associados a um objeto representando o todo. Em outras palavras, a composição é um tipo forte de associação, onde um objeto agregado é composto de vários objetos componentes [Rumbaugh94].

Diferenciando composição e agregação:

- A composição é um tipo ainda mais forte de agregação;
- Na agregação um objeto-parte pode continuar existindo mesmo com a exclusão do objeto-todo. Exemplo: carro e motor.

  Ao se excluir o objeto-todo, basta excluir o(s) relacionamento(s) dele com o(s) objeto(s)-parte;
- Na composição um objeto-parte não continua existindo se o objeto-todo for excluído. Exemplo: nota fiscal e item de nota fiscal.

  Ao se excluir o objeto-todo, deve-se excluir o(s) objetos(s)-parte a ele associado(s).

2.8.3 LIGAÇÕES E ASSOCIAÇÕES

As ligações/associações são os relacionamentos cotidianos que ocorrem entre objetos/classes, quando há necessidade de troca de mensagem. Já os relacionamentos anteriormente listados são tipos especiais de associações.

Objetos relacionam-se com outros objetos. Por exemplo, em "o empregado João trabalha no departamento de Pessoal", temos um relacionamento entre o objeto `empregado João` e o objeto `departamento Pessoal`.

Ligações e associações são meios de se representar relacionamentos entre objetos e entre classes, respectivamente. Uma ligação é uma conexão entre objetos. No exemplo anterior, há uma ligação entre os objetos `João` e `Pessoal`. Uma associação, por sua vez, descreve um conjunto de ligações com estrutura e semântica comuns. No exemplo anterior, há uma associação entre as classes `empregado` e `departamento`. Todas as ligações de uma associação interligam objetos das mesmas classes, e assim, uma associação descreve um conjunto de potenciais ligações da mesma maneira que uma classe descreve um conjunto de potenciais objetos [Rumbaugh94].

2.9 CLASSES E OPERAÇÕES – ABSTRATAS, CONCRETAS, GENÉRICAS

Nem todas as classes são projetadas para instanciar objetos diretamente. Algumas são usadas simplesmente para organizar características comuns a diversas classes ou para encapsular classes que participam de uma mesma associação ou composição. Tais classes são ditas **classes abstratas**. Uma classe abstrata é desenvolvida basicamente para ser herdada por outras classes. Ela existe meramente para que um comportamento comum a um conjunto de classes possa ser fatorado em uma localização comum e definido uma única vez. Assim, uma classe abstrata não possui instâncias diretas, mas suas **classes descendentes concretas**, sim. Uma classe concreta é uma classe instanciável, isto é, que pode ter instâncias diretas.

Uma classe abstrata pode ter subclasses também abstratas, mas as classes-folhas na árvore de herança devem ser classes concretas.

Classes abstratas podem ser projetadas de duas maneiras distintas. Primeiro, elas podem prover implementações completamente funcionais do comportamento que pretendem capturar. Alternativamente, elas podem prover apenas definição de um protocolo para uma operação sem apresentar um método correspondente. Tal operação é dita uma **operação genérica ou abstrata**. Neste caso, a classe abstrata não é completamente implementada e todas as suas subclasses concretas são obrigadas a prover uma implementação para suas operações abstratas, obtendo assim as chamadas **operações concretas**. Assim, diz-se que uma operação abstrata é uma operação com múltiplas implementações. Ela define apenas a **assinatura** a ser usada nas implementações que as subclasses deverão prover, garantindo, assim, uma interface consistente. Métodos que implementam uma operação genérica têm a mesma semântica. Uma vez que a mesma operação é definida em várias classes de uma mesma hierarquia, é importante que os métodos que a implementam conservem sua interface com o exterior (assinatura), isto é, o nome, o número e o tipo dos argumentos, e os resultados da operação.

Uma classe concreta não pode conter operações abstratas porque senão seus objetos teriam operações indefinidas. Analogamente, toda classe que possuir uma operação genérica não pode ter instâncias diretas e, portanto, obrigatoriamente é uma classe abstrata.

OBSERVAÇÃO:

Apesar de se encontrar classes/operações abstratas como sinônimos de classes/operações genéricas há uma diferença a ser considerada: Classes/operações genéricas referem-se a habilidade de parametrizar uma classe/operação com tipos de dados diferentes.

2.10 INTERFACES

Origem: Orientação a objetos – Parte I. Introdução da linguagem Java, Classes, Interfaces, Objetos (Herança, Encapsulamento e Polimorfismo). Autor: Rafael Cardoso Pereira. Artigo em site da DevMedia (http://www.devmedia.com.br)

As interfaces são estruturas muito parecidas com as classes, porém com um propósito diferente. Uma interface não possui atributos e sim somente operações.

O fato aqui é que as operações não são como nas classes, ou seja, em uma interface estas operações não possuem implementação.

Uma interface será utilizada para definir comportamentos que outros elementos, as classes, podem desejar implementar. Imagine então que queremos definir um tipo de comportamento que poderá ser implementado por mais de uma classe em nosso sistema, então este comportamento colocaremos em uma interface. A classe obrigatoriamente deverá possuir estas operações da interface implementadas nela, sendo que a implementação ficará na classe e não na interface.

2.11 SOBRECARGA

Em sistemas orientados a objetos, operações distintas de classes distintas, ou até de uma mesma classe, podem ter o mesmo nome. Neste caso, temos um nome de operação sobrecarregado.

No caso de métodos com o mesmo nome dentro de uma mesma classe as assinaturas dos métodos devem ser diferentes.

2.12 POLIMORFISMO

O polimorfismo está ligado a seguinte afirmação: “uma interface, múltiplos métodos”.

O polimorfismo é uma poderosa ferramenta para o desenvolvimento de sistemas flexíveis. Polimorfismo significa a habilidade de tomar várias formas. No contexto da orientação a objetos, o polimorfismo está intrinsecamente ligado à comunicação entre objetos. De fato, polimorfismo pode ser melhor caracterizado, neste contexto, como o fato de um objeto emissor de uma mensagem não precisar conhecer a classe do objeto receptor. Assim, uma mensagem pode ser interpretada de diferentes maneiras, dependendo da classe do objeto receptor, ou seja, é o objeto receptor que determina a interpretação da mensagem, e não o objeto emissor. O emissor precisa saber apenas que o receptor pode realizar certo comportamento, mas não a que classe ele pertence e, portanto, que operação é efetivamente executada. Um objeto sabe qual é a sua classe, e, portanto, a correta implementação da operação requisitada. A mensagem é associada ao método a ser realmente executado, através da identificação da operação e da classe do objeto receptor.

Frequentemente, o polimorfismo é caracterizado como o fato de uma operação poder ser implementada de diferentes maneiras em diferentes classes. Todavia, isto é apenas uma consequência do que foi dito anteriormente e não polimorfismo em si.

A maioria dos autores não diferencia sobrecarga e polimorfismo, tratando ambos os casos como polimorfismo. Neste texto, entretanto, preferimos utilizar polimorfismo com uma semântica mais rígida: polimorfismo limitado a uma hierarquia de classes. Uma operação será dita polimórfica se ela existir, com a mesma assinatura, em uma cadeia de superclasses-subclasses. Uma operação polimórfica tem uma única semântica e os métodos que a implementam conservam essa semântica e, por conseguinte, a sua interface (assinatura). Na sobrecarga de operador, as operações não têm necessariamente a mesma semântica e não há necessidade de se preservar a assinatura. Ao contrário, se as operações sobrecarregadas forem da mesma classe, elas deverão ter assinaturas diferentes. Na prática as operações sobrecarregadas são, normalmente, uma coincidência na escolha de nomes de operação.

De um outro ponto de vista, o polimorfismo pode ser considerado uma característica dos objetos, refletindo a capacidade deles mudarem de classe em tempo de execução. O polimorfismo é extremamente útil quando deseja-se executar alguma operação em um objeto, mas não se sabe, a priori, qual será exatamente a sua classe em tempo de execução.

2.13 LIGAÇÃO ESTÁTICA E LIGAÇÃO DINÂMICA

Quando uma mensagem é enviada a um objeto, deve ser estabelecida uma ligação entre a mensagem enviada e o método a ser executado. A seleção do código para executar uma operação, como dito anteriormente, é baseada nos objetos identificados nas mensagens.

Quando a classe do objeto que recebe a mensagem é conhecida em tempo de compilação, a ligação pode ser feita nesta etapa ou durante a link-edição. Neste caso, tem-se **ligação estática**. No entanto, muitas vezes, a classe de um objeto só pode ser identificada quando a mensagem é realmente emitida e, portanto, o código só pode ser selecionado neste momento. Neste caso, tem-se **ligação dinâmica** ou **tardia (*late binding*).**

Muitas vezes, polimorfismo e ligação dinâmica são confundidos. Porém, é importante frisar que ligação dinâmica significa apenas que a mensagem só é associada ao método específico da classe do objeto receptor no momento em que é enviada, ou seja, quando ocorre a execução.

2.14 RESUMO

Origem: Introdução à Programação Orientada a Objetos, prof. Daniel Merli Lamosa, maio de 2002, slides

Algumas Características conceituais:

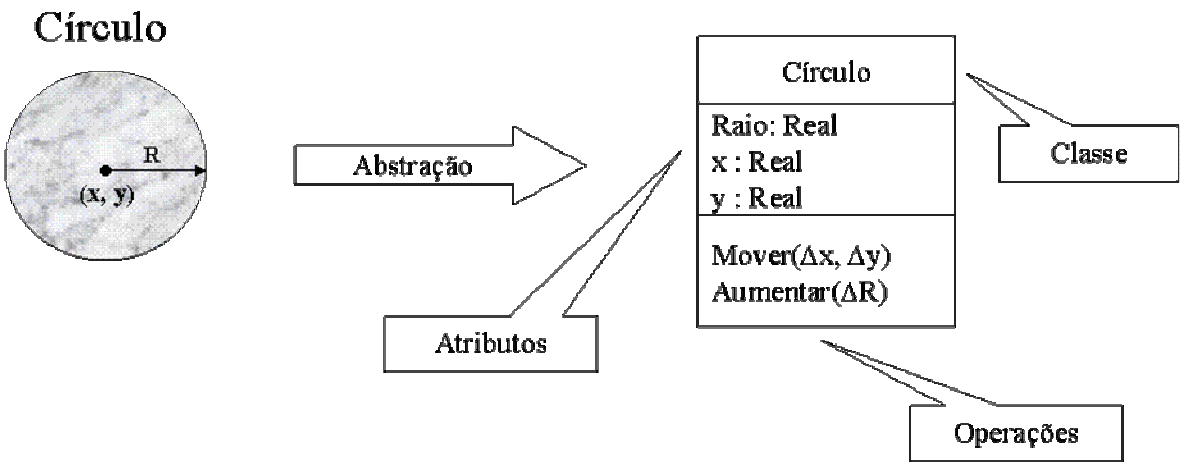
- **Identidade**: 2 objetos são distintos mesmo que todos os valores de seus atributos sejam iguais;
- **Classificação**: Objetos com a mesma estrutura de dados (*atributos*) e mesmo comportamento (*operações*) são agrupados em uma classe;
- **Polimorfismo**: A mesma operação pode atuar de modos diversos em classes diferentes;
- **Herança**: Compartilhamento de atributos e operações entre classes com base em um relacionamento hierárquico.

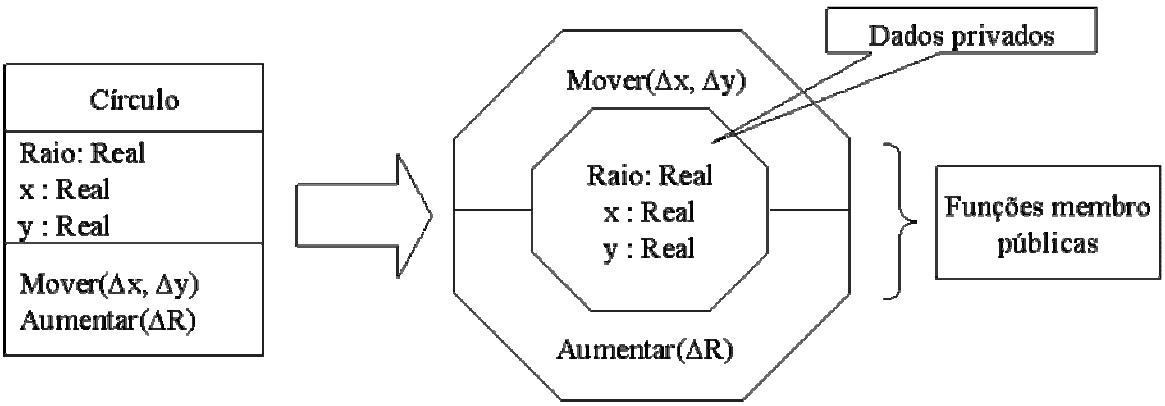
Algumas Características da tecnologia orientada a objetos:

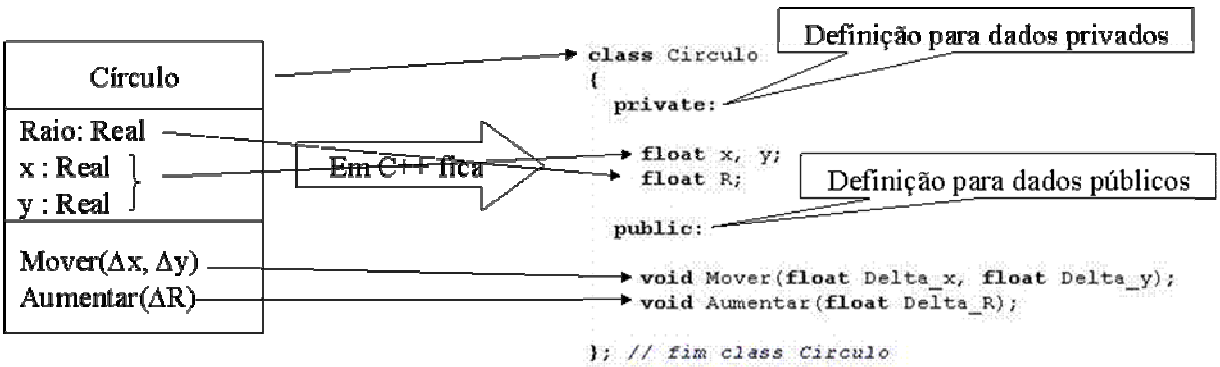
- **Abstração**: Concentração nos aspectos essenciais, próprios, de uma entidade e em ignorar suas propriedades acidentais;

- **Encapsulamento**: Separação dos aspectos externos de um objeto, acessíveis por outros, dos detalhes internos da implementação que ficam ocultos dos demais;
- **Combinação de Dados e Comportamento**: Polimorfismo dos objetos;
- **Compartilhamento**: Compartilhar a estrutura comum (classes) por diversas subclasses sem redundâncias (Herança);
- **Ênfase na Estrutura de Objetos**: Especificar o objeto e não como ele é utilizado;
- **Sinergia**: Seguir todas as características simultaneamente.

Representando a Orientação a Objetos:







2.15 DICAS DE LEITURA

- Livro: Modelagem de Objetos através da UML. Autor: José Davi Furlan, capítulo 1 – Rumo à Orientação a Objetos, tópico 2 – As bases da Orientação a Objetos;
- Livro: Java – como programar. Autor: Deitel & Deitel. Capítulo 1 – Introdução aos computadores, à internet e à world wide web, estudo de caso de engenharia de software: introdução à tecnologia de objetos e UML;
- Artigo - Paradigma da Orientação a Objetos. Autor: Alessandro F L (lecadf);
- Texto: O que significa Orientação a Objetos. Link: http://www.macoratti.net/oo_conc2.htm;
- Texto: Orientação a Objetos: Conceitos Básicos. Link: http://www.macoratti.net/net_oocb.htm;
- Texto: Conceitos da OO. Link: www.inf.ucp.br/profs/guilherme/**Conceitos**%20de%20OO.ppt;
- Origem: Orientação a objetos – Partes I e II. Introdução da linguagem Java, Classes, Interfaces, Objetos(Herança, Encapsulamento e Polimorfismo). Autor: Rafael Cardoso Pereira. Artigo em site da DevMedia (http://www.devmedia.com.br).