CONCEITOS BÁSICOS DE PROGRAMAÇÃO ORIENTADA A OBJETOS

Article · July 2007

CITATIONS
0

READS
106

1 author:

Fernanda Farinelli
Federal University of Minas Gerais
12 PUBLICATIONS **1** CITATION

SEE PROFILE

CONCEITOS BÁSICOS DE PROGRAMAÇÃO ORIENTADA A OBJETOS

Autor: Fernanda Farinelli

2007

ÍNDICE:

PROGRAMAÇÃO ORIENTADA A OBJETOS.

1. Histórico:

A “Engenharia de software” surgiu no final da década de 60 para tentar solucionar os problemas gerados pela “Crise do software”, no entanto, várias técnicas que foram desenvolvidas nos anos 70 e 80 não conseguiram resolver os problemas de produtividade e qualidade em softwares.

Nos anos 90, um grande número de empresas, de diversos portes, já estava com boa parte de seus sistemas com um considerável nível de informatização, além disso, o uso da internet como meio de comunicação e busca maciça de informação foi amplamente divulgado e finalmente o computador passou a ser uma ferramenta de trabalho para muitas pessoas.

Com tudo isso, surge a necessidade de se produzir softwares mais atraentes, dinâmicos e com alto poder de troca de informação, além disso, passou-se a exigir dos softwares uma maior produtividade e melhor qualidade.

Os softwares atuais se caracterizam por ter grande interação com o usuário, pelo uso de interfaces gráficas, pela necessidade permanente de alteração e expansão (dada a velocidade de mudanças de hardware) e apresentam interação com outros sistemas possibilitando a troca de dados entres eles, portabilidade para diversas plataforma e sistemas operacionais.

Ao longo de várias pesquisas e testes, verificou-se que a reutilização era a peça chave para o aumento da produtividade e melhoria da qualidade dos softwares. A reutilização significa usar novamente algo que foi feito. Seu objetivo é permitir uma ampla utilização de todos os tipos de informação encontradas na situação de desenvolvimento. Podemos citar como benefício da reutilização uma redução das etapas de desenvolvimentos de software.

As técnicas oferecidas pela programação estruturada não eram suficientes para atender, com satisfação desejada, a elaboração destes tipos de aplicações. Seria necessário partir para uma nova técnica de programação. A técnica que começou a ser adotada por parte dos programadores foi a Programação Orientada a Objetos.

2. Fundamentos da Programação Orientada a Objetos:

A Orientação a Objetos é uma tecnologia que enxerga os sistemas como sendo coleção de objetos integrantes. Ela permite melhorar a reusabilidade e extensibilidade dos softwares.

A tecnologia orientada a objetos é fundamentada no que, coletivamente, chamamos de modelo de objetos, que engloba os princípios da abstração, hierarquização, encapsulamento, classificação, modularização, relacionamento, simultaneidade e persistência.

2.1. Paradigma Orientado por Objetos:

A proposta da orientação a 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.

Os programas Orientados a objetos são programas estruturados em módulos que agrupam um estado e operações sobre este estado. Apresentam ênfase em reutilização de código.

Um dos grandes diferenciais da programação orientada a objetos em relação a outros paradigmas de programação que também permitem a definição de estruturas e operações sobre essas estruturas está no conceito de herança, mecanismo através do qual definições existentes podem ser facilmente estendidas. Juntamente com a herança deve ser enfatizada a importância do polimorfismo, que permite selecionar funcionalidades que um programa irá utilizar de forma dinâmica, durante sua execução.

"Orientação a Objetos consiste em considerar os sistemas computacionais como uma coleção de objetos que interagem de maneira organizada."

2.2. Principais conceitos de POO:

2.2.1. Objetos:

Usamos o termo objeto para representar um determinado elemento do mundo real. Mas somente analisaremos os objetos que tem relevância para a solução de

um determinado problema. Portanto, o objeto é uma entidade do mundo real que merece representação para o ambiente estudado.

Objetos são instâncias de classes, que determinam qual informação um objeto contém e como ele pode manipulá-la. É uma entidade capaz de reter um estado (informação) e que oferece uma série de operações (comportamento) ou para examinar ou para afetar este estado. É através deles que praticamente todo o processamento ocorre em sistemas implementados com linguagens de programação orientadas a objetos.

Como exemplos de objetos, podemos citar os objetos físicos (um livro, uma mercadoria), funções de pessoas para os sistemas (cliente, vendedor), eventos (uma compra, um telefonema), interações entre outros objetos (um item de uma nota fiscal é uma interação entre uma compra e um produto do estoque) e lugares (loja matriz, revenda norte).

Por exemplo, vamos considerar um cachorro como nosso “objeto” de estudo:

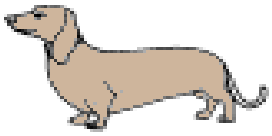


Figura 1 - Cachorro

Analisando este objeto, o cachorro, podemos deduzir que o mesmo possui algumas características que pertencem apenas a ele próprio. Por exemplo, um cachorro possui:

- *Um nome,*
- *Uma idade,*
- *Um comprimento de pêlos,*
- *Uma cor dos pelos,*
- *Uma cor dos olhos,*
- *Um peso,*
- *...*

“As características que descrevem um objeto são chamadas de atributos”.

Além do conjunto de características que descrevem o cachorro, podemos também identificar um conjunto de ações que ele é capaz de executar:

- · *Latir,*
- · *Babar,*

- · *Correr em círculos,*
- · *Pegar a bola,*
- · *Sentar,*
- · *Comer,*
- · *Dormir,*
- · *...*

"As ações que um objeto pode executar são chamadas de métodos ou serviços".

A única maneira de interagir com os objetos é através dos métodos que ele disponibiliza. Para interagir com o cachorro, utilizamos os métodos que relacionamos acima: para alimentá-lo, utilizamos o método "comer", para brincar com ele, utilizamos o método "pegar a bola", etc.

"Chamamos de interface ao conjunto de métodos disponíveis em um objeto."

- **Atributos e métodos:**

Vimos então que os objetos são compostos de atributos e métodos, mas afinal, o que são atributos e o que são métodos?

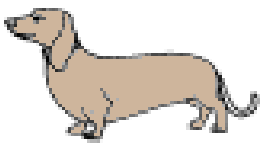
Atributos:

Os objetos do mundo real possuem propriedades que possuem valores. Estes valores definem o estado do objeto. As propriedades recebem o nome de atributos em OO.

Podemos dizer que os atributos dos objetos são "variáveis" ou "campos" que armazenam os diferentes valores que as características dos objetos podem conter.

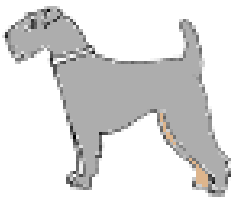
O estado de um objeto é o conjunto de valores de seus atributos em um determinado instante. O comportamento de um objeto é como ele age e reage em termos de suas mudanças de estado e troca de mensagens com outros objetos.

O cachorro do nosso exemplo poderia ser representado pelos seguintes atributos:



Cachorro	
Nome:	Pluto
Idade:	2 anos
Comprimento dos pêlos:	Curtos
Cor dos pêlos:	Bege
Cor dos olhos	Castanhos
Peso:	5 Kg

Um outro objeto “cachorro” apresentaria valores diferentes para os mesmos atributos, por exemplo:



Cachorro	
Nome:	Snoopy
Idade:	4 anos
Comprimento dos pêlos:	Compridos
Cor dos pêlos:	Cinza
Cor dos olhos	Pretos
Peso:	8 Kg

Os atributos de um objeto somente mudam de valor através de estímulos externos ou internos. A única forma de modificar os atributos dos objetos é disparando eventos que provocam a transição desses estados no objeto.

Métodos:

Os métodos são procedimentos ou funções que realizam as ações próprias do objeto. Assim, os métodos são as ações que o objeto pode realizar. Tudo que o objeto faz é realizado através de seus métodos, pois é através dos seus métodos que um objeto se manifesta, e através deles que o objeto interage com os outros objetos.

Um objeto exibe algum comportamento (executa uma operação) quando recebe um estímulo de outro objeto. Um objeto requisita a ação de algum outro objeto, enviando uma mensagem para ele. Esta mensagem é uma solicitação a um objeto para que seja executada as rotinas que chamamos de Método da classe. Os métodos são responsáveis por acessar ou alterar os atributos de um objeto.

Imaginando os métodos do nosso objeto de estudo, o cachorro, enumeramos métodos (ações) como Latir, babar, comer sentar, etc.

- **Exercício 1:**

Para atender as necessidades de informação de uma biblioteca universitária foi proposto um sistema que deve atender as seguintes características:

- O cadastro dos usuários da biblioteca com endereço completo. Os usuários podem ser classificados em três grupos: Professores, Alunos e Funcionários.
- O cadastro das obras da biblioteca, que podem ser classificadas em: Livros científicos, periódicos científicos, periódicos informativos, periódicos diversos, entretenimento, etc.
- A língua em que se encontra o exemplar da obra.

- A mídia onde se encontra o exemplar da obra.
- Os autores da obra com o controle da nacionalidade do autor.
- As editoras dos exemplares com o ano de edição de cada exemplar.

Identifique os possíveis objetos com seus respectivos atributos e métodos.

2.2.2. Classes:

Uma classe representa um conjunto de objetos que possuem características e comportamentos comuns e de agora em diante, diremos que um objeto é uma instância de uma determinada classe, ou seja, criaremos nossos objetos baseados nas características definidas nas classes.

A ênfase da metodologia orientada a objetos é dada na criação das classes, e não dos objetos, como se poderia pensar pelo nome.

Olhando os dois cães do exemplo anterior, vemos que os dois possuem exatamente o mesmo conjunto de atributos. Isso acontece porque se trata de dois objetos da mesma classe, ou categoria. Isso significa que os dois possuem exatamente o mesmo conjunto de atributos e métodos, embora cada objeto possa ter valores diferentes para os seus atributos.

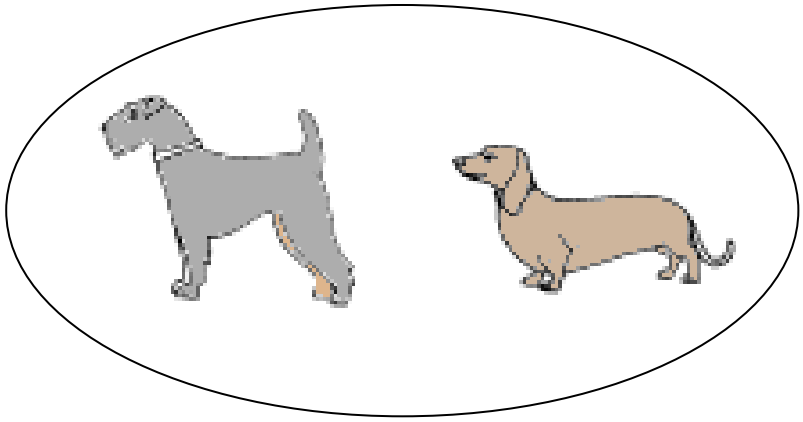


Figura 2 - Classe Cães

Objetos de mesma classe possuem a mesma definição tanto para métodos quanto para atributos.

Tomemos uma classe gatos formada de objetos “gato”. Estes objetos possuem as seguintes características: nome, idade, peso, cor de pêlos, cor de olhos e comprimento de pêlos. Além disso, o objeto possui as seguintes ações: miar, comer, dormir, subir na árvore.

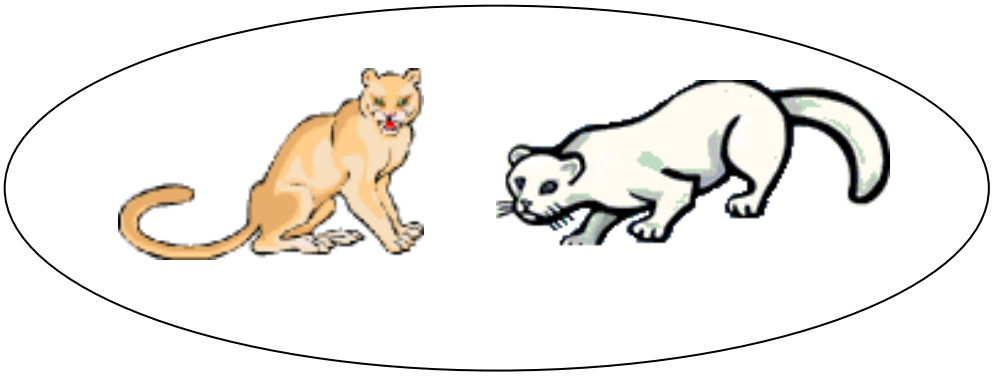


Figura 3 - Classe Gatos

Muitos objetos do mundo real possuem características comuns e podem ser agrupados de acordo com elas. Uma classe representa um gabarito para muitos objetos e descreve como estes objetos estão estruturados internamente.

As classes Cães e Gatos possuem características e métodos comuns, por exemplo: Características comuns (nome, idade, peso, cor de pêlos, cor de olhos e comprimento de pêlos), Métodos comuns (pegar a bola, comer, dormir). Surge então o conceito de subclasse e superclasse. Podemos ter uma superclasse de Mamíferos:

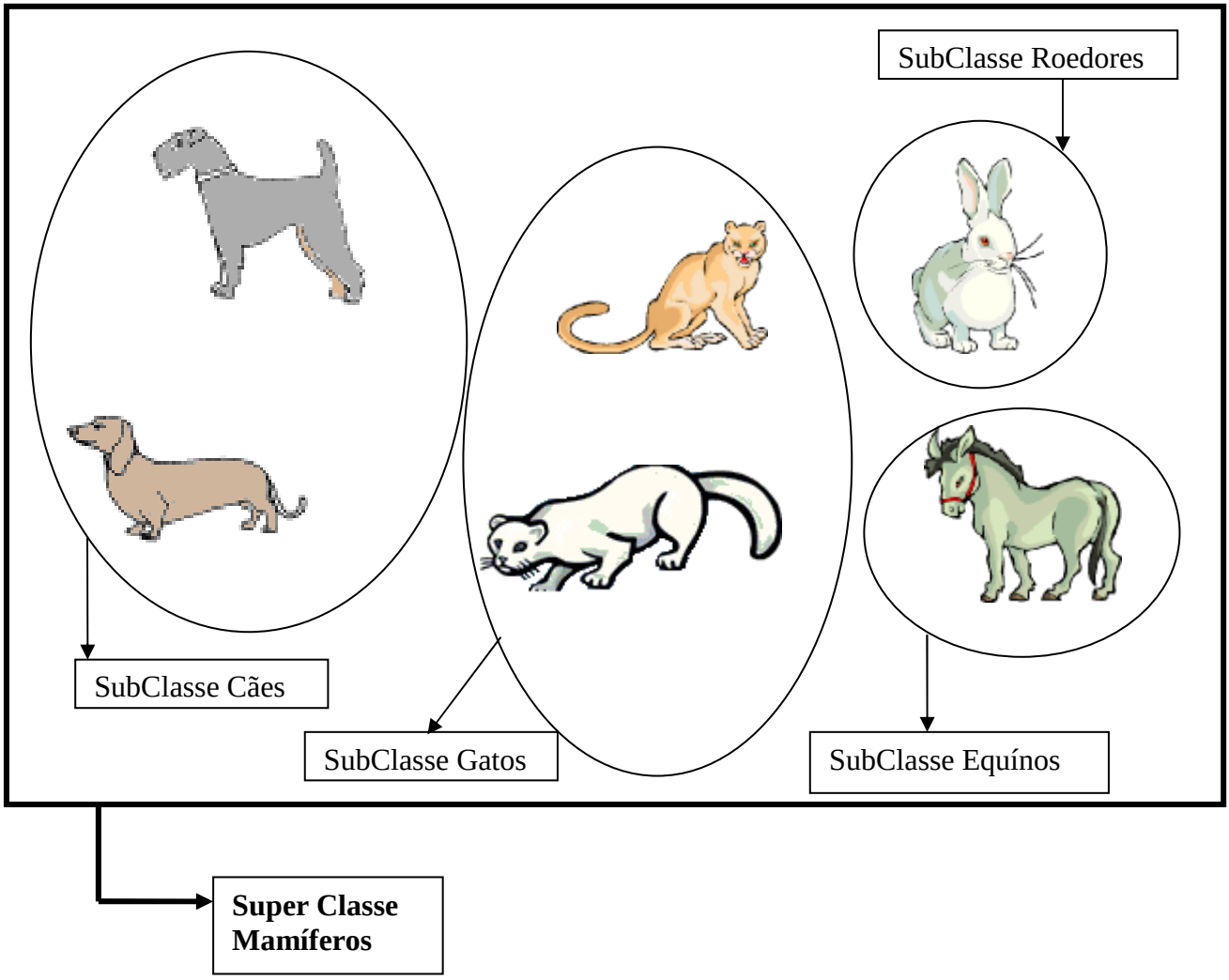


Figura 4 - Super Classe Mamíferos

Dentro da SuperClasse Mamíferos temos pelo menos 4 Subclasses. Podemos dizer que elas apresentam as seguintes características em comum: nome, idade, peso, cor dos olhos e cor dos pêlos. Além disso, estas subclasses possuem os seguintes métodos em comum: comer, sentar e dormir.

Observando a figura 4, podemos distinguir quatro categorias diferentes de animais: “cães”, “gatos”, “eqüinos” e “roedores”. Verificamos que existem seis objetos na super classe mamíferos, sendo distribuídos em quatro subclasses (Cães, gatos, roedores e eqüinos). Essa é a diferença entre classe e objeto: a classe é um modelo e todos os objetos daquela classe possuem atributos comuns, mas esses atributos possuem valores distintos, e os métodos também são comuns.

Utilizando a hierarquia de classe, podemos omitir da declaração de um objeto ou de uma classe inferior tudo aquilo que já foi definido na(s) classe(s) superiores.

Só serão definidos no objeto os atributos e métodos particulares desse objeto que não são atribuídos aos outros objetos da mesma classe.

Chamamos de “ancestrais” às classes das quais as outras dependem e de “descendentes” as classes originadas a partir de outra. No exemplo acima, a classe “mamíferos” tem um ancestral (animais) e dois descendentes (cães e gatos). Em cada descendente só é preciso descrever os atributos e métodos que apresentam alteração em relação à descrição do ancestral, conforme a figura 5.

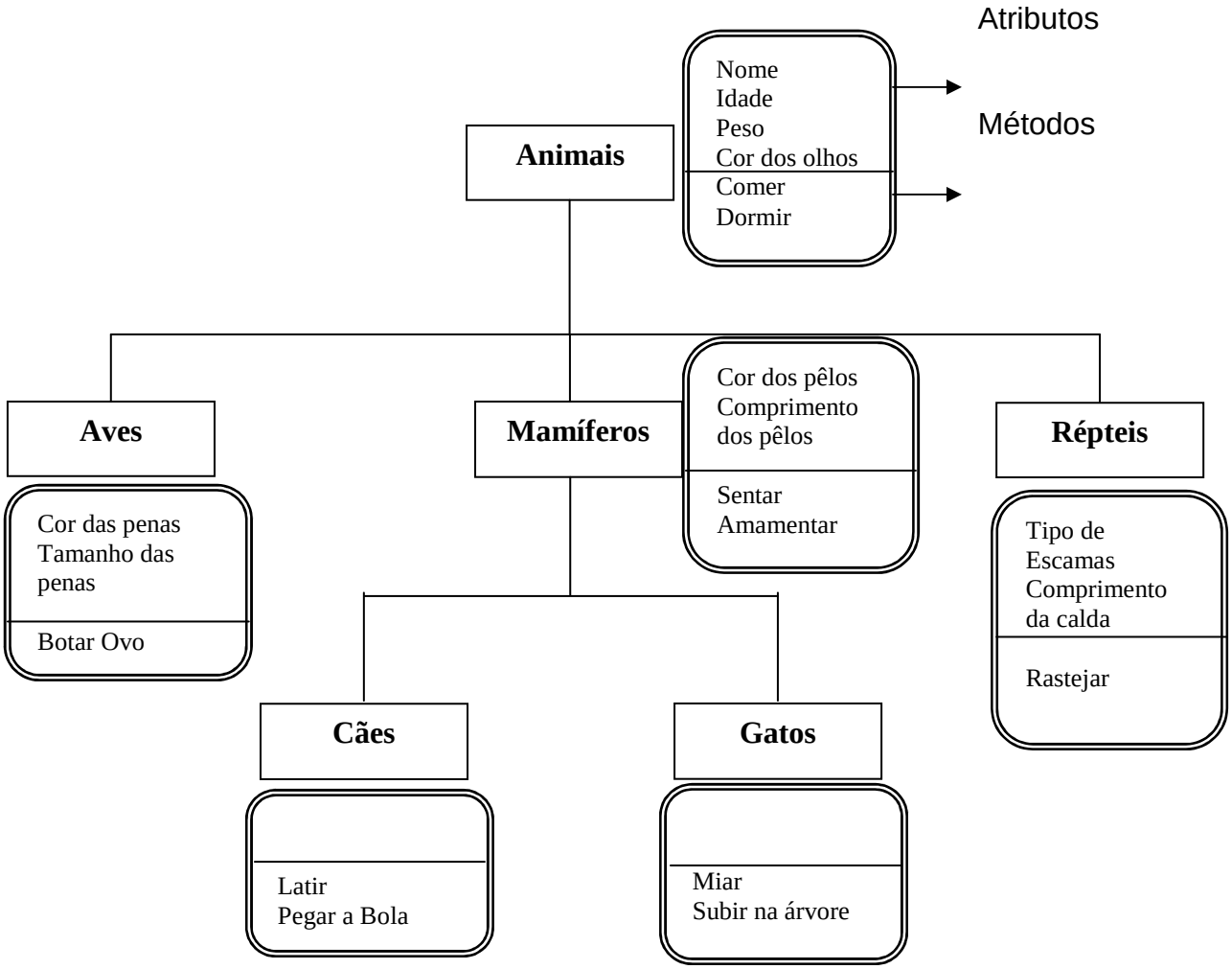


Figura 5 – Hierarquia de classes

As classes são definições de como os objetos devem ser e não existem na realidade. Somente os objetos têm existência. Usando o exemplo dos animais, quando vamos mostrar nosso cachorro a alguém, não dizemos “esse é um cão”, e sim “esse é o pluto”, ou “snoopy”. O que se pode ver não é uma classe de seres, mas um cachorro específico, um objeto.

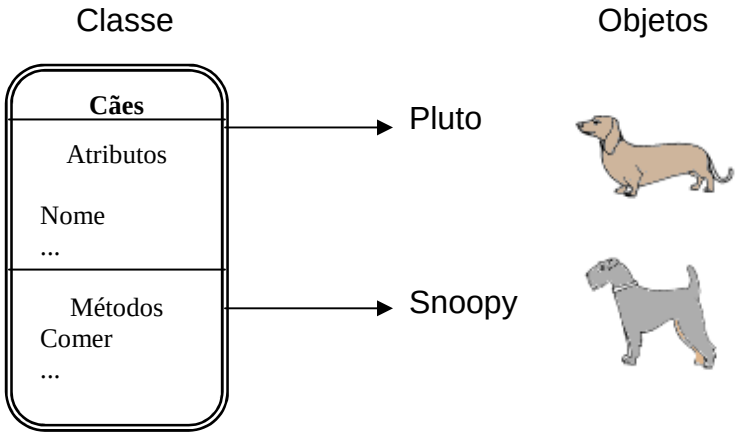


Figura 6 – Exemplo de Objetos

- **Instanciação:**

A instanciação é quando a classe produz um objeto, como se ela fosse uma espécie de modelo ou gabarito para a criação de objetos. Conforme a teoria da orientação a objetos, dizemos que um objeto é, nada mais nada menos, que a instância de uma classe.

Pelo exemplo que estamos estudando, cada cachorro que for armazenado é um novo objeto, uma nova instância da classe “Cães”. A classe serve de modelo para a criação de novos objetos.

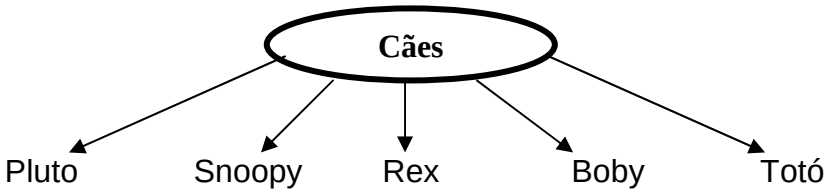


Figura7 - Instanciação

- **Classes puras ou abstratas:**

Classes puras são classes das quais os objetos nunca são instanciados diretamente, mas sempre por uma classe descendente dela. Essas classes são criadas para facilitar o processo de estruturação. Um exemplo clássico é criar uma classe Pessoa, que contém os atributos (nome, endereço, telefone, etc.) e métodos (alteração de endereço, imprimir ficha, etc.) necessários para manusear dados de pessoas em um sistema de informação. A partir dessa classe genérica, criam-se classes descendentes específicas para manusear funcionário, gerente, etc.

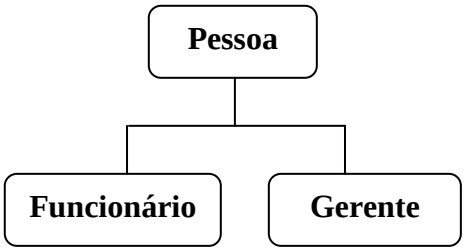


Figura 8 – Exemplo de classe pura ou abstrata

A classe Pessoa nunca terá um objeto a ela instanciado, ela só existe para unificar todos os atributos e métodos comuns as classes Gerente e Funcionário evitando assim a redundância. Pelo exemplo acima, “Pessoa” é uma classe pura ou abstrata.

Existem notações que sugerem que uma classe pura seja chamada também de classe-&-classe e as classes que podem ser instanciadas de classe-&-objeto.

- **Exercício 2:**

Para atender as necessidades de informação de uma biblioteca universitária foi proposto um sistema que deve atender as seguintes características:

- O cadastro dos usuários da biblioteca com endereço completo. Os usuários podem ser classificados em três grupos: Professores, Alunos e Funcionários. Para os Alunos é necessário conhecer o curso ao qual pertencem. Dos Professores e funcionários, é necessário conhecer o Departamento ao qual estão sempre vinculados.
- O cadastro das obras da biblioteca, que podem ser classificadas em: Livros científicos, periódicos científicos, periódicos informativos, periódicos diversos, entretenimento, etc.
- A língua em que se encontra o exemplar da obra.
- A mídia onde se encontra o exemplar da obra.
- Os autores da obra com o controle da nacionalidade do autor.
- As editoras dos exemplares com o ano de edição de cada exemplar.
- O histórico dos empréstimos.

Identifique as possíveis classes e objetos com seus respectivos atributos e métodos.

2.3. Notação gráfica das classes e dos objetos:

A UML (Unified Modeling Language) é o sucessor de um conjunto de métodos de análise e projeto orientados a objeto (OOA&D). A UML está, atualmente, em processo de padronização pela OMG (Object Management Group).

A UML é um modelo de linguagem, não um método. Um método pressupõe um modelo de linguagem e um processo. O modelo de linguagem é a notação que o método usa para descrever o projeto. Os processos são os passos que devem ser seguidos para se construir o projeto.

Na UML, a representação para uma classe no diagrama de classes é tipicamente expressa na forma gráfica, como mostrado na Figura 9.

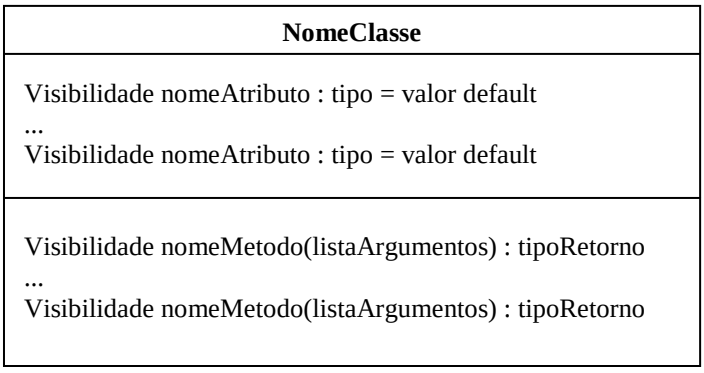


Figura 9 – Representação de Classe em UML

Como se observa nessa figura, a especificação de uma classe é composta por três regiões: o nome da classe, o conjunto de atributos da classe e o conjunto de métodos da classe.

Outra notação gráfica muito utilizada é a que Coad e Yourdon adotaram conforme ilustrado na figura 10. Como na UML, utilizamos um retângulo dividido em 3 seções, onde a primeira seção é o nome que identifica a classe ou o objeto, a

segunda seção é composta pelos atributos da classe ou o objeto e a terceira e última seção é composta pelos métodos que compõem a classe ou o objeto respectivo.

A diferença gráfica da notação UML para a de Coad e Yourdon é que a última representa classe e objetos de forma diferente conforme a figura 10.

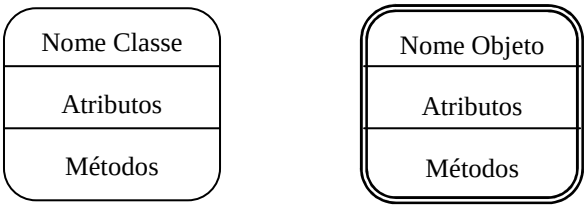


Figura 10 – Representação de classe e objeto segundo Coad e Yordon

2.4. Definição de classes e objetos em Java:

- **Introdução a JAVA:**

Java é uma linguagem de programação Orientada a Objetos muito conveniente para o desenvolvimento de software que funcione em conjunto com a Internet. A linguagem Java foi desenvolvida na Sun Microsystems em 1991, uma grande vantagem que ela apresenta diante das outras linguagens de programação é o fato de seus códigos fonte e objetos serem portáveis para diversas arquiteturas e sistemas operacionais.

A definição de uma classe em Java é:

```
public class NomeDaClasse {
   CorpoDaClasse
}
```

O corpo de uma classe pode conter (atributos, métodos, construtores/inicializadores e outras classes).

No Exemplo 1, temos um código de um aplicativo mínimo geralmente exibe a string “Alo Mundo” na tela. O código Java abaixo mostra um exemplo de um aplicativo mínimo Java.

Exemplo 1:

```
public class AloMundo
{
   static public void main(String[ ] args)
   {
        System.out.println("Alo Mundo");
   }
}
```

A seguir é dada uma breve descrição desses elementos de programa:

- public - O método main() pode ser acessado por qualquer classe.
- static - Uma palavra-chave que informa ao compilador que main() não requer a chamada de uma instância dessa classe.
- void - Indica que nada é retornado por main(). Essa informação é importante pois Java realiza uma verificação de tipo cuidadosa, que inclui a verificação dos métodos chamados, observando se eles realmente retornam os tipos com os quais foram declarados.
- String args[] - A declaração de uma matriz do tipo String. Estes são os argumentos digitados na linha de comando após o nome da classe:

Exemplo 2:

```
public class MDC    {                    // Definição de classe
   private int X=0;                      // Declaração de atributos
   private int Y=0;

   public int Calcula (int X, int Y)     // Declaração de método
   {
      if ( Y == 0 )
         return X;
      else if ( X == 0 )
         return Y;
      else
         return Calcula(Y,X%Y);
   }

   public static void main (String[] args)
   {
      MDC a1 = new MDC();                // Intanciação de objeto

      int R1 = a1.Calcula(10,5);         //Execução do Método da classe
      System.out.println("MDC(10,5) = "+ R1+'\n');

      System.exit(0);
   }

}
```

2.5. Princípios Básicos da orientação a objetos:

5.2.1. Abstração:

Procurando a palavra “abstração” em um dicionário da língua portuguesa, encontramos, como definição, “ato de separar mentalmente um ou mais elementos de uma totalidade complexa (coisa, representação, fato), os quais só mentalmente podem subsistir fora dessa totalidade”.

Também conhecida como Ocultamento da informação, a abstração é o princípio pelo qual cada componente deve manter oculta sob sua guarda uma decisão de projeto única. Para a utilização desse componente, apenas o mínimo necessário para sua operação deve ser revelado (tornado público).

Possuímos o recurso da abstração como forma de entender problemas tidos como complexos. Assim, diante de um problema complexo, procuramos dividi-lo em problemas menores, e feito isso, resolvemos cada um deles até encontrar a solução do problema inteiro.

O termo “abstração”, aplicado ao desenvolvimento de sistemas, significa a “grosso modo” só deve ser representado aquilo que vai ser usado.

Pelo princípio da abstração nós isolamos os objetos que queremos representar do ambiente complexo em que se situam, e nesses objetos representamos só as características que são relevantes para o problema em questão. Por exemplo: toda pessoa tem um atributo para “cor dos olhos”, mas em um sistema de folha de pagamento, essa informação não é relevante, portanto, ela não será incluída na relação de características de pessoas que queremos armazenar em nosso sistema. Da mesma maneira, apesar de uma boa parte das pessoas ter animais de estimação, é completamente irrelevante para um sistema de folha de pagamento implementar um objeto para armazenar animais de estimação, e relacioná-los com as pessoas.

O uso da abstração permite também que determinadas partes do problema ganhem maior ou menor peso, e dessa maneira, detalhes podem ser desconsiderados em determinados momentos para que outros sejam ressaltados.

5.2.2. Encapsulamento:

Encapsulamento é o princípio de projeto pelo qual cada componente de um programa deve agregar toda a informação relevante para sua manipulação como uma unidade (uma *cápsula*). Aliado ao conceito de ocultamento de informação, é um poderoso mecanismo da programação orientada a objetos. Com este mecanismo podemos ocultar detalhes de uma estrutura complexa, que poderiam interferir durante o processo de análise.

O modo de utilização dos atributos e métodos (dados e operações realizadas com esses dados) é diferente do modo como a programação tradicional utiliza. Na orientação a objetos os dados e os processos que tratam esses dados estão “encapsulados” numa única entidade. A única maneira de conhecer ou alterar os atributos de um objeto é através de seus métodos.

O encapsulamento é um dos grandes trunfos da programação orientada a objetos em relação à programação tradicional (estruturada). A vantagem é que o *encapsulamento* disponibiliza o objeto com toda a sua funcionalidade sem que você precise saber como ele funciona internamente, nem como armazena internamente os dados que você recupera. Podemos exemplificar da seguinte

forma, para telefonar para alguém você simplesmente pega um telefone e disca o número destino. Isto acontece sem que você saiba quais foram os procedimentos que ocorreram na empresa telefônica que conectou você ao destino que você informou.

Outro ponto importante é permitir que você faça modificações internas em um objeto, acrescentando métodos sem afetar os outros componentes do sistema que utilizam esse mesmo objeto. Para exemplificar esta vantagem, podemos retornar ao nosso exemplo do telefone, onde sabemos que a companhia telefônica pode efetuar diversas alterações no tratamento telefônico, como é o caso de converter as linhas analógicas para digitais, ou aumentar a quantidade de serviços oferecidos, e isso tudo sem modificar o método que você utiliza para telefonar. Isto pode ser realizado porque a companhia telefônica encapsulou a maneira de realizar uma chamada telefônica. Assim, a empresa está livre para efetuar todas as alterações que desejar, sem afetar a maneira como você acessa os serviços dessa companhia.
O que importa para poder haver interação entre dois objetos é que um conheça o conjunto de operações disponíveis do outro (interface) para que então envie e receba informação, ou mesmo ordene a realização de procedimentos.

5.2.3. Herança:

A palavra herança, conforme um dicionário da língua portuguesa, significa:

1. *Aquilo que se herda;*
2. *Aquilo que se transmite por hereditariedade.*

Um exemplo simples de explicar a herança é a própria genética. Ou seja, um filho herda características genéticas dos pais, e por sua vez, repassa essas características aos seus filhos.
Na programação Orientada a Objetos, Herança é o mecanismo pelo qual uma classe obtém as características e métodos de outra para expandi-la ou especializá-la de alguma forma, ou seja, uma classe pode “herdar” características, métodos e atributos de outras classes. Da mesma maneira uma classe transmite suas características para outras classes, tornando aquelas que recebem suas características suas herdeiras.

Sob o ponto de vista prático da orientação a objetos, a herança constitui um mecanismo muito inteligente de aproveitar código. É através da herança que os objetos podem compartilhar métodos e atributos. Assim, podemos criar uma nova classe fazendo com que esta herde os métodos e atributos de uma outra classe, tornando-a uma classe “filho” da classe que a gerou. A grande vantagem neste caso é que reutilizaremos todo o código já implementado na classe pai, restando apenas implementar os métodos e atributos que a diferenciem da classe pai.

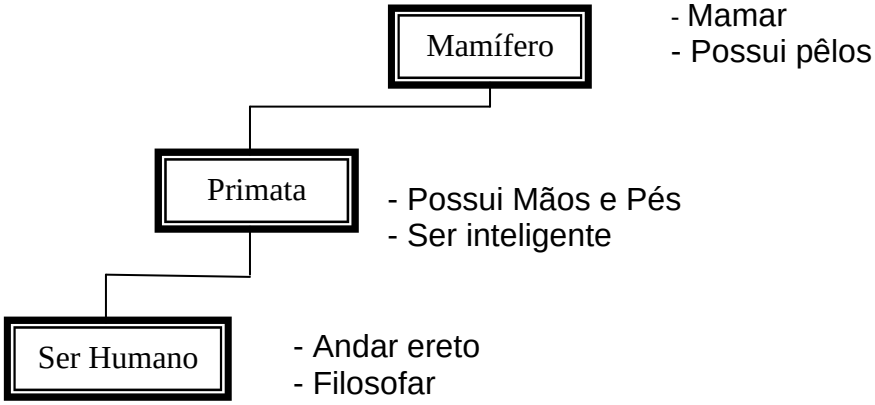


Figura 11 – Exemplo de Herança

De acordo com o exemplo da figura acima, podemos dizer que a classe “Primata” é subclasse (descendente ou filha) da classe “Mamífero” e superclasse (ancestral) da classe “Ser Humano”.

“Herança significa que todos os atributos e métodos programados no ancestral já estarão automaticamente presentes em seus descendentes sem necessidade de reescrevê-los.”

A respeito dos métodos relacionados no exemplo acima possui podemos dizer que: os métodos Andar ereto e Filosofar são pertencem exclusivamente a classe “Ser humano”, Possui Mãos e Pés e Ser inteligente pertencem a classe “Primata” e são herdados pela classe “Ser Humano”, e os métodos Mamar e Possui pêlos pertencem a classe “Mamífero” e são herdados pelas classes “Primata” e “Ser Humano”. O mesmo acontecerá com os atributos que forem definidos para essas classes.

Mais do que mera economia de código, a herança significa maior integridade, pois quando se altera um comportamento em uma classe, todos as classes descendentes desta também estarão utilizando o método atualizado sem necessidade de reprogramação.

5.2.4. Formas de herança

- **Herança Simples:**

A herança é denominada simples quando uma classe herda características de apenas uma superclasse. Por exemplo, podemos ter como superclasse uma classe chamada Pessoa, e dela derivar uma subclasse chamada Funcionário. Esta nova classe, Funcionário, herda todas as características da sua superclasse Pessoa, e de mais nenhuma outra. Como diferencial esta nova classe Funcionário possui atributos e métodos que a classe Pessoa não possui, uma vez que

atributos como Salário e Função não pertencem ao contexto da classe Pessoa, mas sim ao contexto da classe Funcionário.

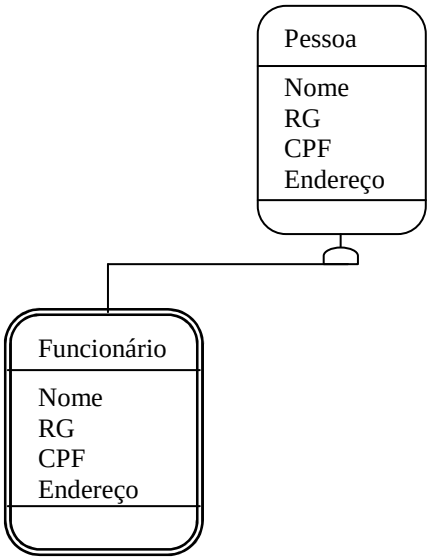


Figura 12 – Herança Simples

Nada impede, entretanto, que a mesma superclasse gere mais de uma subclasse. Uma superclasse Pessoa pode gerar uma subclasse Cliente da mesma forma como pode gerar uma subclasse Funcionário.

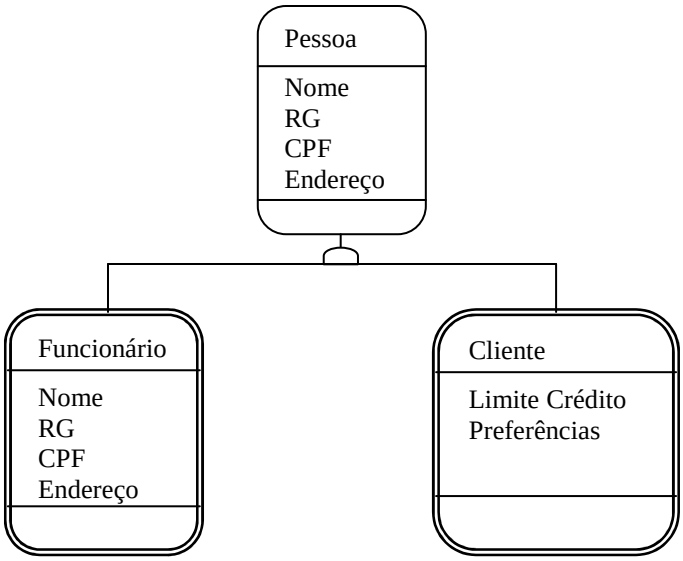


Figura 13 – Herança Simples com dois descendentes

- **Herança Múltipla:**

A herança é denominada múltipla quando uma classe herda características de duas ou mais superclasses. Por exemplo, usando o exemplo do Funcionário e do Cliente, devemos estar atentos ao fato de que um funcionário pode vir a ser um cliente também. Assim, além de possuir atributos e métodos que o qualificam

como funcionário, deve também possuir possui atributos e métodos que o qualificam como cliente.

Para tanto, o que pode ser feito é criar uma nova subclasse, chamada Funcionário-Cliente, que tem como superclasse as classes Funcionário e Cliente. Esta nova classe, Funcionário-Cliente, herda todas as características da superclasse Pessoa, Funcionário e Cliente. O seu diferencial pode ser exatamente o fato de possuir os atributos e métodos da classe Funcionário e da classe Cliente juntas. Porém nada impede que tenha atributos próprios como, por exemplo, um desconto especial para clientes de uma empresa que são também funcionários.

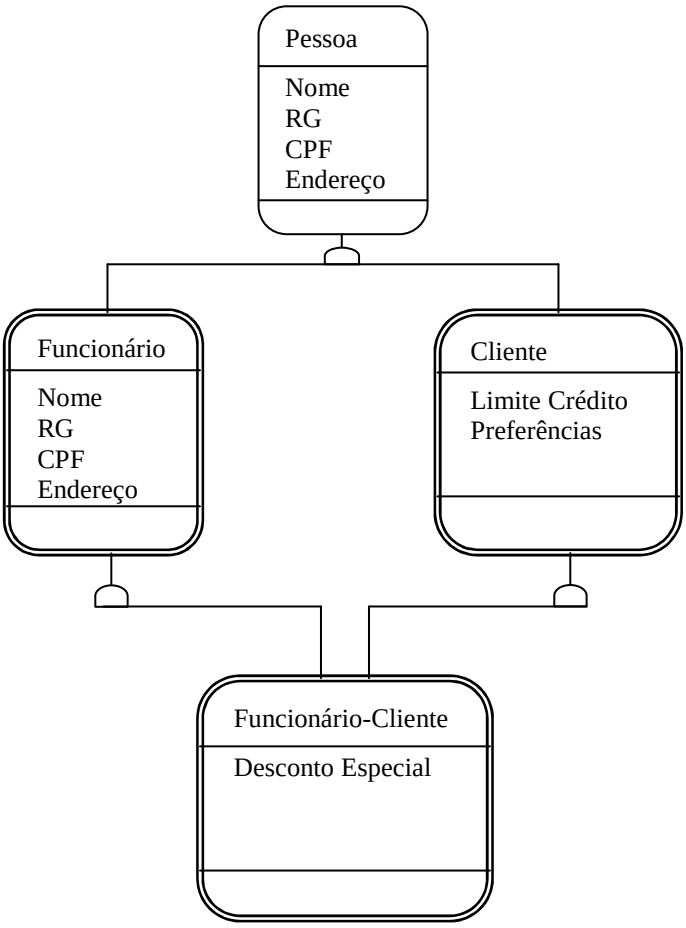
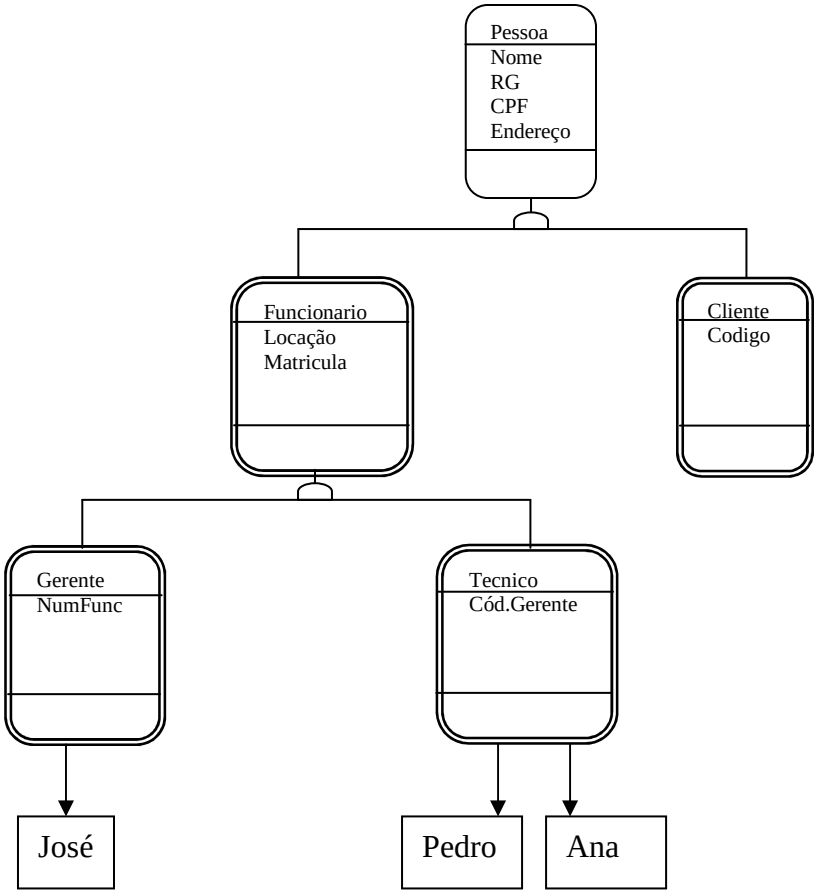


Figura 14 – Herança Múltipla

- **Exercício 3:**



Segundo o diagrama marque verdadeiro (V) ou falso (F) para as afirmações abaixo.

( ) O digrama acima ilustra cinco classes e três objetos.
( ) A classe Funcionário apresenta três descendentes e um ancestral.
( ) Podemos dizer que José é um objeto instanciado da classe gerente.
( ) A classe Técnico herda todos os atributos de Funcionário, Pessoa e Cliente.
( ) A classe Funcionário é subclasse de Pessoa e superclasse de Gerente e Técnico.
( ) O objeto Pedro poderia herdar os métodos de Pessoa caso existissem.
( ) Os atributos “Locação” e”Matricula” pertencem exclusivamente a classe Funcionário.
( ) Se for incluído um novo atributo na classe Pessoa todas as outras classes descendentes passaram a ter este atributo também.
( ) O tipo de herança existente entre Cliente e Pessoa é herança simples, já entre Funcionário, Técnico e Gerente é do tipo herança múltipla.

5.2.5. Polimorfismo:

O Polimorfismo é a capacidade de uma variável se referir em tempo de execução a objetos de diversas classes. Podemos defini-lo também como sendo o nome dado à capacidade que objetos diferentes têm de responder a uma mesma mensagem. Uma mesma mensagem pode apresentar formas de execução diferentes, próprias de cada objeto. Com esta característica o usuário pode enviar uma mensagem genérica e abandonar detalhes sobre a exata implementação sobre o objeto receptor.

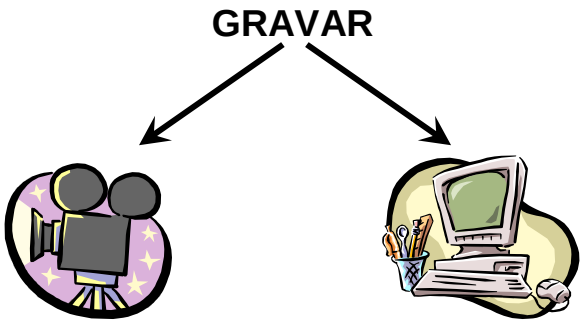


Figura 15 – Exemplo de Polimorfismo.

Pelo exemplo da figura 15, verificamos que o computador e a filmadora possuem uma mesma função que é gravar, mas esta função é realizada de forma diferente. O polimorfismo é uma característica importante da orientação a objetos que está diretamente ligada à hereditariedade de classes.

"O polimorfismo ocorre quando um método que já foi definido no ancestral é redefinido no descendente com um comportamento diferente."

➢ **Exercício 4:**

Dos objetos abaixo, separe em grupos aqueles que possuem “métodos” em comum, mas que são executados de forma diferente, ou seja, apresentam polimorfismo entre eles.


(1)

(2)

(3)

(4)

(5)

(6)

(7)

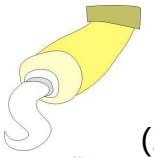
(8)

(9)

(10)

(11)

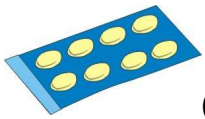
(12)

(13)

(14)

(15)

(16)

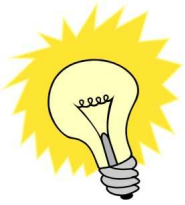
(17)

(18)

(19)

(20)

(21)

(22)

(23)

(24)

(25)

2.6. Estruturas:

As estruturas ajudam os analistas a arranjar os objetos de forma que se possa visualizar melhor o domínio e a complexidade do problema em estudo. Na análise orientada a objetos, existem dois tipos básicos de estrutura: Generalização–Especialização e Todo–Parte.

6.2.1. Estrutura de Generalização – Especialização:

Para mostrar um exemplo da aplicação dessa estrutura, utilizaremos o mesmo exemplo apresentado no item Herança simples, aquele que trata da classe Pessoa, Funcionário e Cliente. Neste caso, temos a classe Pessoa como a mais alta classe generalizada, sendo então a superclasse da estrutura. Abaixo dela, encontramos as suas duas subclasses, Funcionário e Cliente, que nada mais são que especializações da classe Pessoa. Ou, simplesmente, podemos ver a classe Pessoa como uma generalização da classe Funcionário e da classe Cliente.

Esta estrutura é formada por uma classe genérica no topo da estrutura e suas classes descendentes especializadas dispostas abaixo. O uso do semicírculo, conforme mostra a ilustração da Figura 16, identifica a classe genérica e as especializações. A linha saindo do ponto central do semicírculo aponta para a generalização.

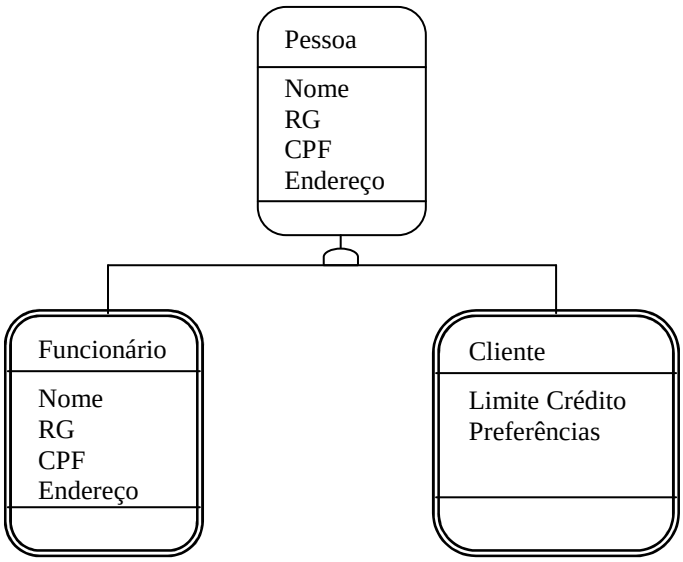


Figura 16 – Exemplo de uma estrutura de Generalização-Especialização

Esta estrutura é, basicamente, uma estrutura hierárquica onde temos superclasses e suas respectivas subclasses. A estrutura Generalização – Especialização é, de fato, o processo de herança discutido na seção anterior. Assim, desejando-se realizar uma herança deve-se utilizar esta estrutura.

6.2.2. Estrutura Todo - Parte:

Este tipo de estrutura é bastante característico, uma vez que trata de agregação ou decomposição de objetos. Essa estratégia é bastante útil na identificação dos objetos e dos seus componentes diante de um determinado problema em estudo.

É nesta estrutura que descrevemos, quando for o caso, quais subclasses formam uma superclasse. Isto significa que um objeto, ou mesmo uma classe pode conter, dentro de si, outras estruturas. Assim, o nosso entendimento sobre um determinado problema reduz-se ao estado do problema em questão, tornando a análise mais clara e também objetiva.

A notação gráfica é bastante simples. Usa-se um triângulo, partindo das subclasses apontando a superclasse, dessa forma sabemos quem é composto por quem.

Um bom exemplo para ilustrar esse tipo de estrutura pode ser a relação entre um carro e seus componentes. O carro tratado como um objeto pode ser decomposto em diversos outros objetos, como o motor, as rodas, o painel, o banco, as portas etc. Podemos seguir com a decomposição e avançar sobre o motor, que pode ser dividido em distribuidor, carburador e outros. Portanto, percebemos que um objeto mais complexo, muitas vezes, nada mais é do que um agregado de diversos outros objetos simples.

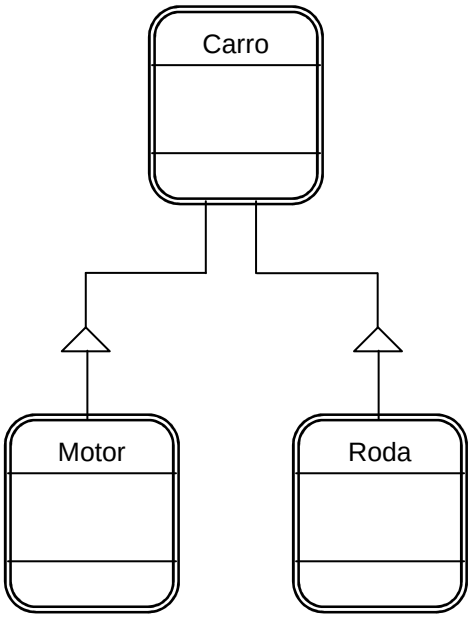


Figura 17 – Estrutura Todo-Parte

A estrutura Todo – Parte também possui outro componente, chamado de cardinalidade. Essa palavra é um termo sofisticado que significa “quanto”. Por exemplo, um carro possui 4 rodas (5 para o caso do estepe), que sob o ponto de

vista da análise, são todas iguais. Ou seja, cada roda é igual à outra. Assim, vemos a roda como um objeto que se repete 4 a 5 vezes dentro do objeto carro.

Estendendo a cardinalidade também ao motor do nosso carro, temos uma relação de 0,1 do motor para com o carro, e de 1,1 do carro para com o motor. Ou seja, significa que o motor pode estar em nenhum ou um carro, e que o carro deve ter no mínimo 1 motor e no máximo também 1. Não temos visto até o momento um carro com dois motores, mas certamente já vimos motores sem carro (em oficinas mecânicas por exemplo).

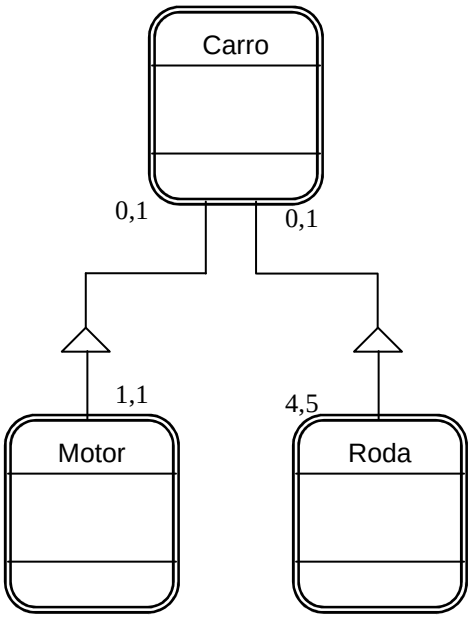


Figura 18 – Cardinalidade

3. Modelagem de um Sistema Orientado a objetos:

A modelagem baseada em objetos é um meio para se descrever os sistemas do mundo real. Um objeto representa uma abstração de uma entidade do mundo real, combinando, em um mesmo elemento informação e comportamento. Os paradigmas anteriores do desenvolvimento de software ligavam apenas fracamente a informação e o comportamento. A orientação a objetos conduz a uma união mais bem definida destes dois conceitos.

É perfeitamente possível modelar uma solução pensando totalmente orientado a objetos desde a fase de análise, passando pelo projeto do software e chegando à implementação em uma linguagem de programação orientada a objetos.

Uma grande vantagem de se pensar totalmente orientado a objeto é o fato de que um mesmo objeto, concebido em fase de análise, passa com as mesmas características desde o usuário até o programador que será responsável pela codificação final.

Apesar das múltiplas técnicas de modelagem de sistemas baseadas em objetos e filosofias diferentes difundidas atualmente, uma parte considerável dos autores (talvez todos) converge quando se trata de um ponto em particular, o processo de análise inicia com a identificação das classes e dos objetos. A idéia é que, uma vez definidas as classes e os objetos, o analista, expandindo e ajustando essas classes e objetos também acabe melhorando o modelo. Esse processo é natural e esperado.

Os passos básicos a serem observados para o desenvolvimento de um modelo orientado a objetos, normalmente compreendidos, são os seguintes:

1. Identificar as Classes e os Objetos;
2. Identificar as Estruturas e os Relacionamentos entre os Objetos.
3. Identificar os Atributos e os Métodos Importantes;

Ao contrário de outras técnicas de modelagem de sistemas, o foco principal da análise orientada a objetos não está centrado na estrutura ou nos dados do sistema, mas sim na composição das classes e objetos do sistema. É muito importante que todas as classes e objetos estejam bem definidas antes de começar qualquer linha de código, pois as classes e os objetos estão amarrados de tal modo que mesmo uma alteração pequena pode acarretar uma mudança que tende a se propagar por todo o sistema modelado.

- **Exercício 5:**

Para ilustrar a aplicação dos passos descritos, utilizaremos um exemplo de modelagem de uma escola que oferece cursos para a comunidade. A relação das premissas que deverão ser consideradas neste sistema é a seguinte:

- Um curso pode ser formado por uma ou muitas disciplinas diferentes;
- Uma disciplina poderá fazer parte de nenhum ou até muitos cursos;
- Cada disciplina deverá ser ministrada por apenas um professor, podendo o professor ministrar uma ou muitas disciplinas diferentes;
- Um curso não poderá ter mais de 40 alunos, nem menos de 20 alunos matriculados;
- Para cada turma de alunos deverá haver uma sala de aula;
- Um aluno poderá se matricular em nenhum ou até muitos cursos;

1. Identificar as Classes e os Objetos;

2. Identificar as Estruturas e os Relacionamentos entre os Objetos.

3. Identificar os Atributos e os Métodos Importantes;

4. Programação Orientada a Objetos:

A programação orientada a objetos consiste em utilizar estruturas de dados que simulem o comportamento dos objetos, facilitando a programação pelo uso de uma metodologia unificada para a análise e a programação. Os mesmos objetos que o analista identifica no seu diagrama podem ser implementados exatamente como foram projetados, sem necessidade de fazer uma "tradução" dos diagramas da análise para estruturas de dados e de programação como acontece na análise estruturada.

É possível, entretanto, fazer uma análise orientada a objetos e implementar em uma linguagem tradicional, bem como é possível também fazer implementações em linguagens orientadas a objetos de sistemas analisados com outras metodologias.

"Programação Orientada a Objetos consiste em utilizar objetos computacionais para implementar as funcionalidades de um sistema."

As técnicas de programação orientada a objetos recomendam que a estrutura de um objeto e a implementação de seus métodos devem ser tão privativos como possível. Normalmente, os atributos de um objeto não devem ser visíveis externamente. Da mesma forma, de um método deve ser suficiente conhecer apenas sua especificação, sem necessidade de saber detalhes de como a funcionalidade que ele executa é implementada.

5. Vantagens e Benefícios da programação OO:

A Orientação a Objetos consiste em conceber um sistema computacional como um todo orgânico formado por objetos que se relacionam entre si. Esse enfoque pode ser aplicado tanto à análise de sistemas quanto à programação, e essa é uma das principais vantagens da orientação a objetos: a mesma metodologia serve tanto para a definição lógica do sistema quanto para a sua implementação.

- **Unificação entre dados e processos:**

Uma antiga questão da análise de sistemas é a consistência entre a definição dos dados e dos processos. Algumas metodologias recomendam definir a estrutura de dados primeiro, e em seguida definir os processos que os utilizam. Outras recomendam o oposto: procurar definir os procedimentos que o sistema automatiza e posteriormente identificar os dados que esses processos necessitam.

Entretanto, na prática dificilmente essas duas visões são coerentes entre si. Partindo dos dados, chega-se a processos diferentes do que aqueles identificados pelo enfoque nos processos, e vice-versa. Em projetos maiores, quando há pessoas diferentes identificado dados e processos, essas diferenças levam a problemas que vão complicar a implementação, pois os procedimentos podem não estar completamente compatíveis com a base de dados.

Na orientação a objetos, esse problema inexiste: dados e processos são apenas componentes, e o enfoque está em identificar quais os objetos que interagem entre si no sistema. Os dados são identificados procurando os atributos que definem os objetos, e os procedimentos pelas operações que estes objetos realizam. A interação entre os objetos é definida pelas estruturas e relacionamentos que são identificados. O resultado é que em um modelo orientado a objetos, existe total coerência entre os dados e os processos, mesmo quando há muitas pessoas trabalhando no mesmo sistema.

- **Consistência entre análise e desenvolvimento:**

A análise de sistemas convencional procura mapear separadamente as estruturas de dados e os processos que manipulam esses dados. Normalmente isso é feito utilizando-se como ferramentas o Modelo de Entidades e Relacionamento (MER) e o Diagrama de Fluxo de Dados (DFD). Essa modelagem do sistema é apenas conceitual, e não existe correlação direta entre as entidades conceituais levantadas na análise e as entidades físicas a serem implementadas, sejam estruturas de dados ou programas.

Por exemplo, uma entidade em um diagrama de fluxo de dados que representa um processo pode tanto ser implementado em vários programas como várias entidades serem implementadas em um único programa. Tanto que é quase impossível ler um programa qualquer e identificar qual dos processos do DFD que ele está implementando.

Utilizando-se orientação a objetos tanto na análise quanto no desenvolvimento, esse problema não existe. As classes e objetos definidos na análise são exatamente os mesmos que serão implementados. Dependendo do ambiente e da linguagem escolhidos irão fazer com que seja necessário acrescentar objetos a mais na implementação, normalmente para o controle da interface, reduzindo o número de problemas oriundos de erros de tradução entre análise e implementação.

- **Reutilização e aumento da produtividade:**

A produtividade do desenvolvimento de sistemas orientados a objetos é visível e bastante perceptível principalmente quando fazemos uso da reutilização. A reutilização, na prática da orientação a objetos, é muito mais do que simplesmente copiar funções, ou mesmo utilizar bibliotecas de funções como se faz ainda em programas escritos usando o paradigma da programação estruturada. A reutilização de objetos é, de fato, a utilização de objetos já utilizados em sistemas anteriores em novos sistemas sem modificar suas estruturas internas, ou mesmo sem a necessidade de fazer modificações no sistema para acomodar esses novos códigos.

A reutilização de objetos já encapsulados pode somar força quando vamos projetar um novo sistema de informações. Podemos, por exemplo, durante um novo projeto, depois de identificar quais serão os objetos necessários para operacionalizar o novo sistema, buscar a reutilização de objetos já implementados em sistemas anteriores. Assim, antes mesmo de partir para a implementação dos novos objetos, podemos buscar objetos já implementados em outros sistemas e utiliza-los sem alteração de estrutura. Se já existirem esses objetos, 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 já implementados cujas características são bastante semelhantes aos objetos necessários, podem, com pequenas modificações, serem úteis ao novo sistema. Outra forma de reutilização pode surgir utilizando a propriedade da herança entre objetos, ou seja, caso surja necessidade de um novo objeto cuja idéia esteja baseada em um objeto já existente, podemos criar novos objetos herdando as características dos objetos já desenvolvidos. Essa propriedade diminui sensivelmente o custo de desenvolvimento de novos sistemas.

- **Multidesenvolvimento:**

O multidesenvolvimento de um sistema, quando utilizada a técnica de desenvolvimento estruturado de sistemas, é, certamente, muito delicado. A técnica em si parece não ajudar muito, já que existe, reconhecidamente, um verdadeiro abismo entre as fases de análise, projeto e programação. Para esse tipo de empreendimento, é necessário não somente uma extensa e detalhada documentação sobre o sistema, mas também um grande entrosamento entre as diversas equipes que compõem o grupo de desenvolvimento.

Dentro do contexto da orientação a objetos, o multidesenvolvimento também é uma realidade facilitada, devido ao fato de que o funcionamento interno dos

objetos é irrelevante para sua utilização. Isso significa que qualquer desenvolvedor pode utilizar os objetos desenvolvidos por qualquer outro, bastando para isso conhecer o conjunto de métodos necessários para utilizá-los.

Desde que a interface não mude, o funcionamento interno pode mudar sem afetar outras partes do sistema. É certo que um procedimento parecido deverá ocorrer se tratando de um sistema estruturado, porém, lembramos que temos facilidades de transposição de código que o método estruturado não possui, como por exemplo, a facilidade de manusear objetos implementados. Facilidade essa existente graças ao fato de os objetos estarem encapsulados. Essa facilidade também é verificada durante toda a implementação, pois a atenção está voltada ao objeto e não ao sistema. Basta que os objetos sejam construídos conforme o projeto para que funcionem em harmonia com os outros objetos.

- **Facilidades em manutenção:**

Quando falamos em manutenção de sistemas orientados a objetos, falamos em manutenção de objetos. Novamente a atenção é centrada no objeto, e não em um sistema com suas interligações. Assim, todas as modificações necessárias para fazer funcionar as novas tarefas do sistema em manutenção ficam restritas às modificações no comportamento dos objetos envolvidos no contexto da nova necessidade.

Na programação convencional a alteração de uma funcionalidade que é utilizada em vários lugares do sistema pode se tornar um verdadeiro pesadelo. O programador que efetuar a manutenção pode não conseguir localizar todos os trechos de código que acessam a funcionalidade, e algumas vezes ao realizar a alteração, são incluídos erros em alguns dos locais alterados. E devemos contar também a quantidade de horas de trabalho que serão gastas nesse procedimento.

Utilizando a orientação a objetos, as alterações são efetuadas em um único ponto, dentro do código do objeto que implementa o comportamento a ser alterado. Não existem dúvidas quanto ao local da alteração: é só localizar o programa onde o objeto em questão está sendo implementado.

O procedimento de manutenção restrito ao objeto reduz custo, tempo e assegura a certeza de uma manutenção eficaz. Isso é possível por que as variáveis envolvidas estavam restritas aos objetos modificados (atributos e métodos), e mais em nenhuma outra parte do sistema. Os sistemas que dependem dos objetos alterados não necessitarão de qualquer alteração. Essa flexibilidade também permite que novos objetos sejam acrescidos ou excluídos com iguais facilidades, e da mesma forma não teremos rastros de código sem vida e sem função dentro do sistema.

6. Exercício de revisão:

Para as afirmações abaixo, numere a segunda coluna de acordo com a primeira:

Coluna 1	Coluna 2
(1) Orientação a Objetos	(___) É o mecanismo pelo qual uma classe obtém as características e métodos de outra para expandi-la ou especializá-la de alguma forma, ou seja, uma classe pode “herdar” características, métodos e atributos de outras classes. Da mesma maneira uma classe transmite suas características para outras classes, tornando aquelas que recebem suas características suas herdeiras.
(2) Objeto	
(3) Atributos	
(4) Métodos	
(5) Classes	
(6) Herança	
(7) Polimorfismo	
(8) Herança simples	
(9) Herança múltipla	(___) Representa um gabarito para muitos objetos e descreve como estes objetos estão estruturados internamente.
(10) Generalização–Especialização	
(11) Todo–Parte	
(12) Instanciação	(___) Quando uma classe herda características de duas ou mais superclasses.
(13) Hierarquia de classe	
(14) Classes puras	(___) Esta estrutura é formada por uma classe genérica no topo da estrutura e suas classes descendentes especializadas dispostas abaixo.

(___) Propiciam a interação com os objetos.
(___) Sua utilização nos permite omitir da declaração de um objeto ou de uma classe inferior tudo aquilo que já foi definido na(s) classe(s) superiores.
(___) É quando a classe produz um objeto, como se ela fosse uma espécie de modelo ou gabarito para a criação de objetos.
(___) São classes das quais os objetos nunca são instanciados diretamente, mas sempre por uma classe descendente dela.
(___) Entidade do mundo real que merece representação para o sistema em analise.
(___) Significa que todos os atributos e métodos programados no ancestral já estarão automaticamente presentes em seus descendentes sem necessidade de reescrevê-los.
(___) Quando uma classe herda características de apenas uma superclasse.
(___) Quando disparados, podem provocar modificações nos atributos dos objetos.
(___) Representa um conjunto de objetos que possuem características e comportamentos comuns.
(___) Ocorre quando um método que já foi definido no ancestral é redefinido no descendente com um comportamento diferente.
(___) Esta estrutura é, basicamente, uma estrutura hierárquica onde temos superclasses e suas respectivas subclasses.
(___) É a instância de uma classe.
(___) Usado para representar um determinado elemento do mundo real.
(___) São as características que descrevem um objeto.
(___) São as ações que um objeto pode executar.
(___) Este tipo de estrutura é bastante característico, uma vez que trata de agregação ou decomposição de objetos.
(___) Seus valores definem o estado do objeto.
(___) Consiste em considerar os sistemas computacionais como uma coleção de objetos que interagem de maneira organizada.

7. Bibliografia:

COAD, Peter e YOURDON, Edward. **Análise Baseada em Objetos**. Rio de Janeiro : Campus, 1992.

TAFNER, Malcon A. e CORREA, Carlos H. .**Manual de Análise Orientada a Objetos.** Blumenau – SC.

RICARTE, Ivan Luiz Marques. **Programação Orientada a Objetos: Uma Abordagem com Java.** Campinas, 2001.