

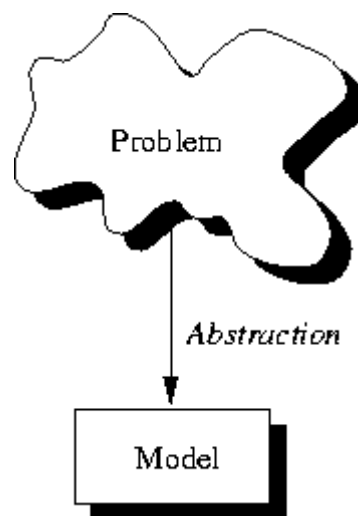
# Programação Orientada a Objetos em C++

## 2. Introdução à Programação e Análise Orientada a Objetos

### 2.1. Introdução aos objetos

#### 2.1.1. Abstração

- Abstrair consiste no processo de retirar do domínio do problema os detalhes relevantes e representá-los não mais na linguagem do domínio, e sim na linguagem da solução.



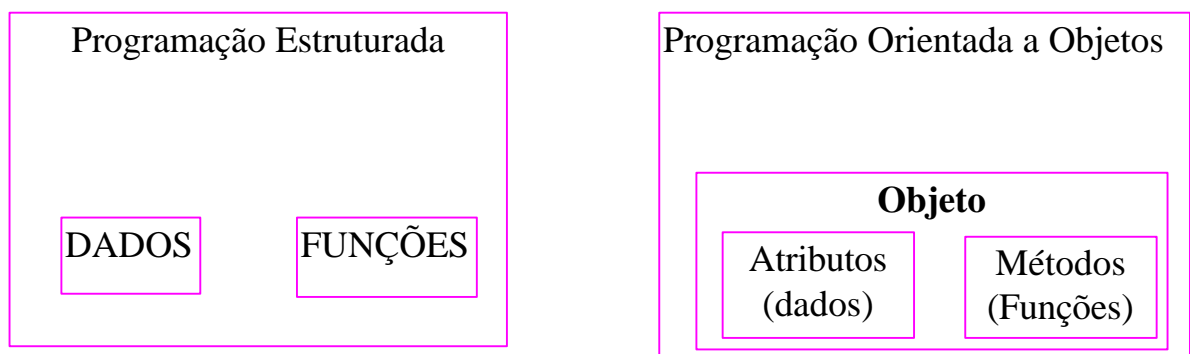
- Todas as linguagens de programação fornecem abstrações. A linguagem Assembly é uma pequena abstração da máquina na qual os programas são executados; As linguagens imperativas (FORTRAN, BASIC e C) são abstrações da linguagem Assembly.
- Nestas linguagens, é necessário pensar em termos da estrutura do computador, ao invés de se pensar em termos da estrutura do problema a ser resolvido!
- O programador deve estabelecer uma associação entre o modelo da máquina (no espaço da solução) e o modelo do problema que está sendo resolvido (no espaço do problema). O esforço para fazer este mapeamento pode ser gigantesco ...
- A programação orientada a objetos tenta trazer o espaço da solução para o espaço do problema: ambos são representados como objetos!!
- O programa permite se adaptar ao problema, adicionando novos tipos ao espaço da solução que mapeiam os tipos existentes no espaço do problema → Descreve-se o problema em termos do problema e não em termos da solução!!

### **2.1.2. Mas o que são objetos?**

- Um objeto é uma variável ... Ele armazena dados. Então uma struct é um objeto?
- Uma struct é um objeto, mas um objeto pode ser mais que uma struct: você pode pedir que determinadas operações sejam feitas sobre os objetos.
- Um objeto possui então atributos (dados) e comportamentos (métodos, procedimentos, funções, que atuam sobre ele).
- Exemplos de objetos: cachorros, carros, videocassetes, edifícios, funcionários, indústrias ...
- Um programa é um conjunto de objetos dizendo uns para os outros o que fazer através do envio de mensagens.
- Concretamente, pode-se pensar nas mensagens como sendo chamadas a funções que pertencem a um objeto em particular.
- Cada objeto tem a sua própria região de memória, que pode ser composta por outros objetos, também. Exemplo: o objeto carro pode ser composto pelos objetos lataria, rodas, motor, etc.

### 2.1.3. Comparação com outros enfoques

- Análise Estruturada e funcional
  - Programa é dividido em um conjunto de funções e dados
  - As funções agem sobre os dados
  - Cada função tem de 10 a 100 linhas
  - Decomposição funcional: força o programador a fixar atenção muito mais nos procedimentos que nos dados
- Enfoque estruturado x Enfoque Orientado a Objetos:
  - Estruturado: Ênfase nos procedimentos, implementados em blocos estruturados, com comunicação entre procedimentos por passagem de dados;
  - Enfoque OO: Dados e procedimentos fazem parte de um só elemento básico (objeto). Os elementos básicos comunicam-se entre si, caracterizando a execução do programa → Dados e procedimentos ENCAPSULADOS em um só elemento.



## 2.2. Mecanismos Básicos da Orientação a Objetos

### 2.2.1. Objeto:

- Entidade que formaliza o modo pelo qual compreendemos algo no domínio de um problema;
- Em um objeto estão *encapsulados* os **dados** (atributos) e os **procedimentos** (serviços ou métodos) exclusivos dele. Os procedimentos são aplicáveis aos dados residentes no objeto.

Exemplo 1: Objeto funcionário “Abravanel dos Santos”.

Atributos: nome, endereço, data de admissão, identidade, CPF ...

Procedimentos: Consulta endereço, aumenta salário, paga comissão, ...

Exemplo 2: Objeto Matriz A

Atributos: Linhas, colunas, elementos da matriz.

Serviços: inicialização, leitura, multiplicação por um escalar, inversão, multiplicação por um vetor, etc ...

Exemplo 3: ...

- Um objeto possui estado, comportamento e identidade
  - Estado: valores de atributos
  - Comportamento: Definido pelos métodos (como o objeto age e reage);
  - Identidade: Aquilo que diferencia um objeto de outro

### 2.2.2. Classes:

- Descrevem um conjunto de objetos semelhantes.

Exemplo: Abravanel pertence à classe dos funcionários: todos funcionários possuem nome, CPF, ..., todos têm comportamento definido pelos métodos que compartilham, etc.

- Qual a diferença entre classes e objetos?
  - Objeto é uma entidade concreta e a classe é uma *abstração*.
  - Ex: Números inteiros, matrizes, Números complexos, automóveis, árvores, casas, cidades, países, etc ....
- Programação: definir uma classe significa formalizar um tipo de dados e todas as operações associadas a este tipo. Declarar objetos significa criar variáveis do tipo definido.
- Os objetos são também denominados *instâncias* de uma classe; Exemplo: O meu Uno-Mille é uma instância da classe automóvel.

- Uma classe possui dados que definem suas propriedades (atributos) e os procedimentos (serviços) que devem ser executados sobre estes dados. A eles dá-se o nome de *membros* de uma classe.
- Os atributos são os dados ou informações do objeto ou da classe. Existem atributos de objetos e atributos de classe.
- Exemplos de atributos de objetos: Funcionário: nome, endereço, telefone, CPF, ....; Carro: nome, marca, ano, cor, ...; Livro: autor, editora, ano, ....
- Atributos de classe: São menos freqüentes que os atributos de objetos. Exemplo: Funcionário: número de funcionários da loja.
- Serviços: Implantados como funções colocados no nível do objeto ao qual se relacionam.
- Mensagens: São os pedidos enviados a um objeto, afim de que ele desempenhe algum serviço. A comunicação entre objetos se faz através destas mensagens.
- Protocolo: é o conjunto de mensagens que um objeto suporta. Define a maneira como um objeto interage com o mundo. Define a **interface** do objeto. Exemplo: Lâmpada.

Lâmpada
<pre>liga(); desliga() aumenta_luminosidade(); diminui_luminosidade();</pre>



- Outros exemplos: videocassete, automóvel, telefone, etc.

### 2.2.3. Encapsulamento

- A idéia por trás do Encapsulamento é a de que a utilização dos objetos não deve depender de sua implementação interna, e sim de sua interface
- A abstração define a *Interface* dos objetos, ou seja, a forma como eles se relacionarão com as demais entidades do sistema.
- A *Implementação* definirá como a abstração será representada (o código). Este não deve ser visível às entidades que usarão a classe de objetos, por ser exclusivamente do interesse da própria classe.
- A interface não apresenta necessariamente todos os métodos de um objeto, mas somente aqueles que podem ser acessados pelo público em geral, os chamados métodos públicos. Existem métodos internos aos objetos: os métodos privados.

Videocassete
<b>público:</b> Play(); FastForward(); Rewind(); Record(); ... <b>privado:</b> PosicionaCabeçaGravação(posição); GiraMotor(velocidade); ...

### 2.2.4. Reutilizando por Composição

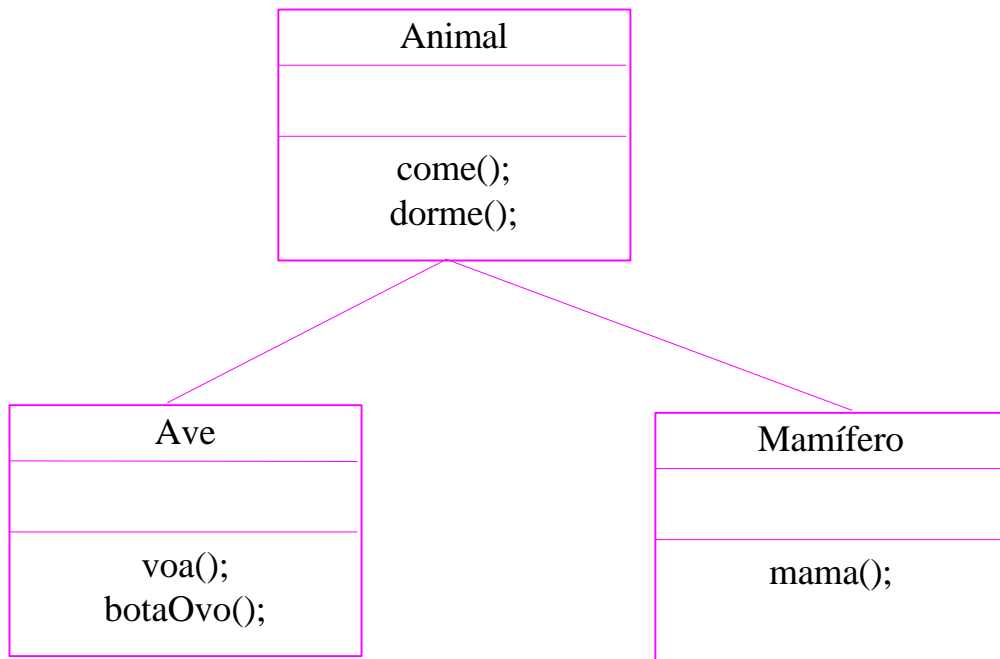
- Uma vez que uma classe foi criada e testada, ela pode ser utilizada por outras classes para auxiliar sua implementação. Esta é uma das principais vantagens da Programação Orientada a Objetos.

- A forma mais simples de reutilização é usar um objeto daquela classe diretamente. Exemplo: janela no Windows; Matriz em um programa de cálculo, etc ...
- Porém, pode-se utilizar um objeto de uma classe dentro de uma nova classe: sua classe pode ser composta pelo número e tipo de objetos que se fizerem necessários, o que é chamado de Composição.
- Exemplos:
  - Classes roda, motor, lataria, ... e a classe carro;
  - Classes motor, controle do motor, cabeça de leitura, cabeça de gravação, ... e a classe videocassete.
- Identifica-se a possibilidade de composição através dos seguintes verbos típicos: **conter**, **possuir**. Ex: Um carro contém 5 rodas, motor, lataria, ...
- Outra possibilidade de reutilização é a utilização de herança.

### 2.2.5. Herança

- Permite modelar uma hierarquia entre classes: classes mais especializadas (classes filhas) herdam propriedades da classe mais geral (classe pai);
- Pode-se compartilhar automaticamente métodos e dados entre diferentes classes, subclasses e objetos. Criar uma nova classe programando somente as diferenças desta para a classe pai.
- A classe filha herda a interface da classe pai, podendo substituí-la quando se espera um objeto da classe pai.
- Identifica-se a possibilidade de herança através da seguinte expressão típica: **é um tipo de**.

**Exemplo:**



- Ave é um tipo de Animal; Mamífero é um tipo de Animal;
- Ave come, dorme, voa e botaOvo; Mamífero come, dorme e mama.
- Uma Ave (ou um Mamífero) podem substituir Animal pois é um tipo de Animal (tem a mesma interface!).
- Herança múltipla: quando a classe derivada possui características herdadas de duas ou mais classes base.
- Exemplo: sistema com interface gráfica associada à manipulação de uma base de dados. Biblioteca gráfica + biblioteca de manipulação de bancos de dados, baseadas em classes: classes do nosso programa construídas herdando as características de interesse das classes de ambas bibliotecas.
- A herança é uma das responsáveis pela facilidade de reaproveitamento de código da POO. Precisamos fazer uma implementação semelhante a uma anterior: derivamos uma classe, programando as diferenças e reaproveitando o código útil da classe base.

### **2.2.6. Polimorfismo**

- “O que possui várias formas”. Propriedade de se usar o mesmo nome para métodos diferentes, implementados em diferentes níveis de uma hierarquia de classes. Para cada classe, tem-se um comportamento específico para o método.
- Exemplo: Método draw para hierarquia de objetos gráficos.
- O polimorfismo é um dos responsáveis pela facilidade de extensão de um programa orientado a objetos: para se efetivar uma extensão de um programa que utiliza polimorfismo, basta derivar novas subclasses de uma classe herdada, programando as modificações na nova classe.
- Não se preocupem demais com o conceito, no momento ...

### **2.2.7. Modularidade**

- Programas grandes e complexos: separação de conjuntos de módulos, cada um contendo classes com independência de funcionamento, podendo ser compilados independentemente.
- Separação entre interface (.h) e implementação (.cpp ou .cc)
- Modularizar consiste em decidir onde empacotar fisicamente classes e objetos presentes na estrutura lógica do programa.

### **2.2.8. Persistência e tipificação:**

- Objetos persistentes são aqueles que permanecem existindo mesmo após o término da execução do programa. Associados à persistência estão o gerenciamento dinâmico da memória e o armazenamento de objetos em bases de dados.
- Característica desejável em um modelo OO é a capacidade de distinguir os diferentes tipos de classe. Uso de um objeto de uma classe onde previa-se o uso de objeto de outra só permitido em condições controladas (conversões explícitas).