

A decorative background featuring a large, stylized number '2' in blue. Surrounding the number are numerous 3D blocks in various colors (purple, blue, yellow, green, orange, pink, grey) arranged in a scattered, overlapping manner. Some blocks are solid, while others are hollow. Faint, thin lines in blue, green, and purple are also visible, weaving through the blocks.

2

Conceitos de orientação a objetos

- ✓ Objetos;
- ✓ Classes;
- ✓ Herança;
- ✓ Persistência;
- ✓ Abstração;
- ✓ Encapsulamento;
- ✓ Polimorfismo;
- ✓ Compartilhamento.

2.1. Introdução

A abordagem de organização proposta pelo termo orientação a objetos é muito diferente da abordagem presente no desenvolvimento tradicional de um software, em que as rotinas e as estruturas de dados são desenvolvidas de maneira linear e sem relacionamento direto entre as ações e dados.

Alguns conceitos considerados fundamentais no desenvolvimento de bons programas, como a abstração e a encapsulação, são favorecidos pela abordagem de orientação a objetos. Esses conceitos não são, porém, exclusivos dessa abordagem, mas apenas melhor suportados nesta do que em outras metodologias.

2.2. Objetos

Para entendermos melhor o conceito de objeto, podemos considerá-lo como uma representação do mundo real. Em termos gerais, entendemos por objetos quaisquer elementos da natureza aos quais é possível atribuir comportamentos e características. No entanto, em termos de computação, entendemos por objetos os elementos capazes de representar uma entidade que esteja no domínio de interesse do problema analisado. As entidades representadas por objetos podem ser concretas ou abstratas.

As linguagens de programação que são orientadas a objetos possuem mecanismos que, a partir do conceito de classes, permitem determinar os tipos de objetos que serão destinados a cada aplicação. Os objetos que apresentam semelhanças entre si são agrupados em classes.



Os objetos, quando são criados, ocupam um espaço na memória que é utilizado para o armazenamento de seu estado e de um grupo de operações, as quais podemos aplicar ao objeto. Mais especificamente, o estado de um objeto refere-se aos valores de seu conjunto de atributos, e seu grupo de operações refere-se ao conjunto de métodos. A classe é responsável por definir ambos, atributos e métodos.

Tendo em vista que os objetos são instâncias de classe, para que eles realizem suas tarefas é preciso que as instâncias sejam criadas, uma vez que, por meio de sua manipulação, as tarefas dos objetos podem ser efetuadas. Uma vez realizadas essas tarefas, podemos excluir os objetos.

Ainda em relação aos objetos, é importante notar alguns aspectos:

- Todos os objetos são distinguíveis e possuem uma identidade própria;
- Os objetos possuem duas finalidades: promover o entendimento do mundo real e dar suporte a uma base prática para uma implementação computacional;
- A decomposição de um problema em objetos depende da natureza do problema e do julgamento do projetista, ou seja, não existe uma maneira correta ou única de fazer esta decomposição;
- Um objeto é constituído por atributos e métodos.

Podemos citar como exemplos de objetos: Nota Fiscal, Cliente, Produto, Boleto, Orçamento.

2.2.1. Atributos

Os valores de dados assumidos pelos objetos de uma classe são chamados de atributos. Para cada instância do objeto, um atributo possui um valor, que pode ser o mesmo para instâncias diferentes. Para exemplificar, o objeto **Pessoa** pode possuir os atributos **Nome** e **Altura**, e o objeto **Produto** pode possuir os atributos **Nome**, **Preço** e **EstoqueAtual**, ou seja, os atributos podem ser entendidos como variáveis ou campos utilizados para armazenar os diferentes valores que as características dos objetos podem conter.

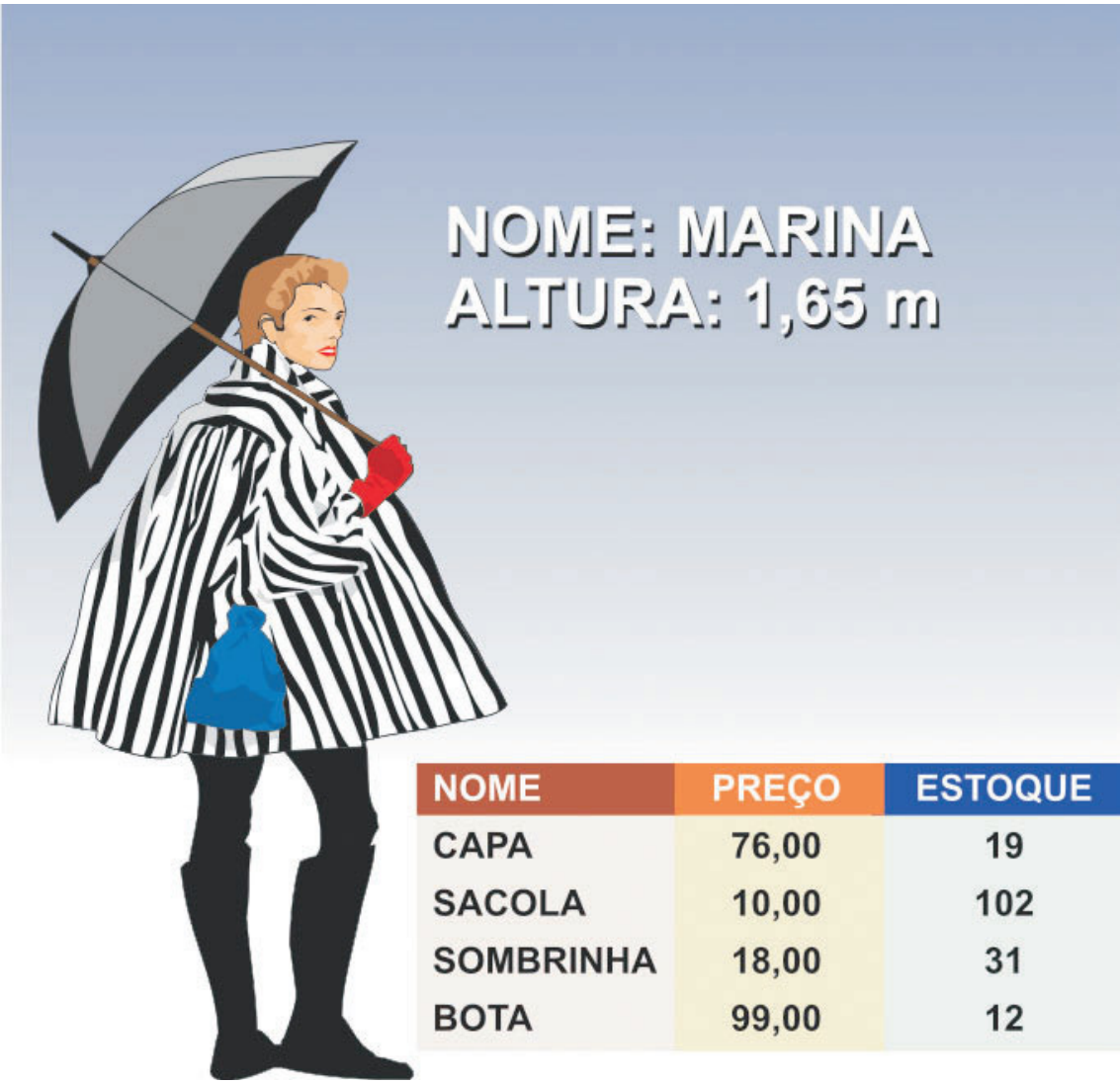


ILUSTRAÇÃO: EDUARDO JOSÉ DE SOUZA ENGELMANN

Dentro de uma classe, os nomes de cada atributo devem ser exclusivos, porém, podemos ter atributos com o mesmo nome em classes diferentes.

Os atributos dos objetos só podem ter seus valores alterados por meio de estímulos internos ou externos. Para modificar esses valores, é necessário disparar eventos que provoquem a transição de estados no objeto.

Os dois itens a seguir garantem a independência completa de qualquer objeto em relação aos demais, além da possibilidade de alteração dos atributos e do código interno dos métodos do objeto, sem o risco de afetar outros objetos internamente:

- Cada objeto é responsável pela mudança de seus próprios atributos;
- Salvo por meio da solicitação de serviços, nenhum objeto possui a capacidade de interferir nos atributos de outro objeto.

2.2.2. Operações e métodos

Vejamos o que são as operações e os métodos:

• Operações

Operação é uma transformação, ou função, que pode ser aplicada a objetos de uma classe ou por esses objetos. Dentro de uma classe, as mesmas operações são compartilhadas por todos os objetos.

Toda operação possui, como argumento implícito, um objeto-alvo, cuja classe determinará o comportamento da operação. O objeto conhece a classe à qual pertence e por isso é possível escolher a implementação correta da operação.



ILUSTRAÇÃO: EDUARDO JOSÉ DE SOUZA ENGELMANN

Operações que se aplicam a diversas classes diferentes são chamadas de polimórficas, pois podem assumir formas diferentes em cada uma das classes.

- **Métodos**

Os métodos permitem que um objeto se manifeste e interaja com outros objetos. Método é uma implementação de uma operação para uma classe específica.

Um método possui uma assinatura constituída pelo nome de uma operação, o número, tipo e ordem de seus argumentos e pelo valor de retorno. Como estratégia de desenvolvimento, é recomendável manter, além de um comportamento consistente entre as implementações, assinaturas que sejam coerentes para os métodos que implementem uma determinada operação.

Como exemplo, podemos citar que o objeto **Conta** pode conter os métodos **Retirar** e **Depositar**.

2.2.3. Mensagens

A mensagem, ou seja, o meio de comunicação entre os objetos, é uma chamada feita a um objeto com o objetivo de invocar um de seus métodos, ativando um comportamento descrito por sua classe. A mensagem é uma requisição de ação junto com argumentos necessários para a execução da tarefa solicitada.



Vejamos um exemplo:

MinhaConta.Depositar(100)

O objeto **MinhaConta** está enviando a mensagem **Depositar** e passando o parâmetro **100**. Esse valor será usado para executar a ação corretamente (depositar 100 reais na conta representada pelo objeto (**MinhaConta**)).

2.3. Classes

As classes criam representações computacionais a partir de entidades do mundo real. São elementos fundamentais no desenvolvimento de softwares orientados a objetos e podem ser definidas como descrições coletivas ou genéricas do mundo real. Assim, em um sistema, a definição das classes deve procurar inspiração nas entidades mundanas.

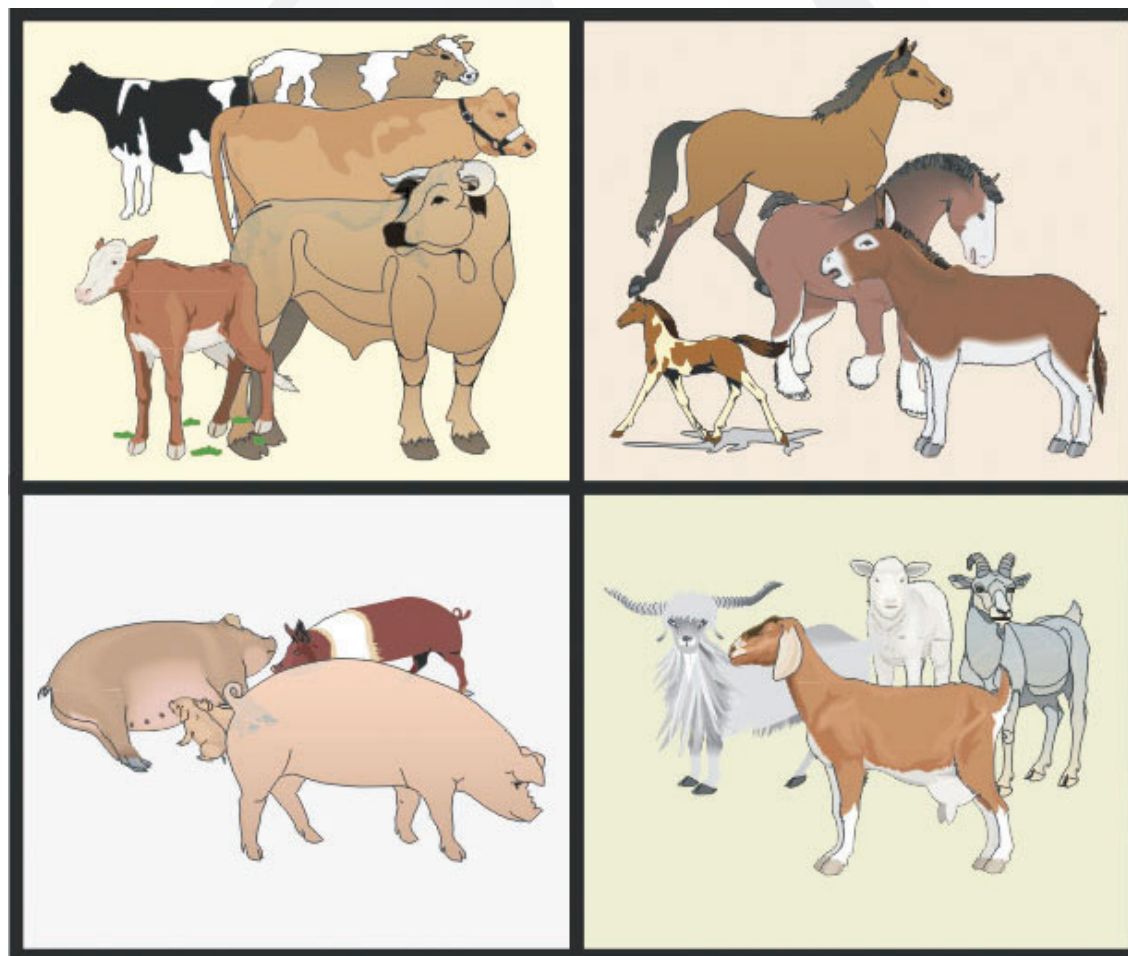


ILUSTRAÇÃO: EDUARDO JOSÉ DE SOUZA ENGELMANN

Computacionalmente, podemos definir classe como uma abstração de um conjunto de objetos, os quais são agrupados por possuírem similaridades em termos de comportamento (operações) e características (atributos). Sendo assim, as propriedades ou os atributos de um objeto são descritos a partir da definição de uma classe.

É importante notar que os objetos são criados pela classe, ou seja, não criamos os objetos, mas sim definimos, na classe, os atributos e métodos necessários para essa criação.

O ato de criar um objeto é chamado de **Instanciação**. Instanciar um objeto é criar uma cópia de uma classe na memória, para uso no programa.

2.3.1. Instanciação

Na teoria de orientação a objetos, um objeto é definido como uma instância de uma classe. A instanciação é a criação de um objeto por uma classe, em que esta funciona como um gabarito, ou modelo, para essa criação.

2.4. Herança

A herança possibilita que as classes compartilhem seus atributos, métodos e outros membros da classe entre si. Para a ligação entre as classes, a herança adota um relacionamento esquematizado hierarquicamente.

O conceito relacionado ao mecanismo de herança é um dos maiores diferenciais entre a programação orientada a objetos e os outros tipos de programação. Por meio da herança é possível estender as definições existentes. Quando temos uma hierarquia de classes planejada de forma adequada, temos a base para que um código possa ser utilizado novamente, o que permite poupar esforço e tempo, na medida em que o desenvolvimento de um código exige ambos de forma considerável.

Na herança, temos dois tipos principais de classe:

- **Classe base:** A classe que concede as características a uma outra classe;
- **Classe derivada:** A classe que herda as características da classe base.

O fato de as classes derivadas herdarem atributos das classes base assegura que programas orientados a objetos cresçam de forma linear, e não geométrica, em complexidade.

Cada nova classe derivada não possui interações imprevisíveis em relação ao restante do código do sistema.

Com o uso da herança, uma classe derivada geralmente é uma implementação específica de um caso mais geral. A classe derivada deve apenas definir as características que a tornam única.

Por exemplo, uma classe base, que serviria como um modelo genérico, poderia ser a classe **Produto** com os campos **Nome** e **Preço**. Já uma classe derivada poderia ser a **Livro**, com os campos **Nome** e **Preço** herdados de **Produto** e acrescida do campo **Numero de Páginas**.

De maneira natural, as pessoas visualizam o mundo como sendo formado de objetos relacionados entre si hierarquicamente. Vejamos a seguir a relação entre animais, mamíferos e cachorros.

Os animais, sob uma descrição abstrata, apresentam atributos, tais como tamanho, inteligência e estrutura óssea, além de aspectos comportamentais, como mover-se, dormir, respirar, alimentar-se, entre outros. Os atributos e aspectos comportamentais descritos definem a classe dos animais.

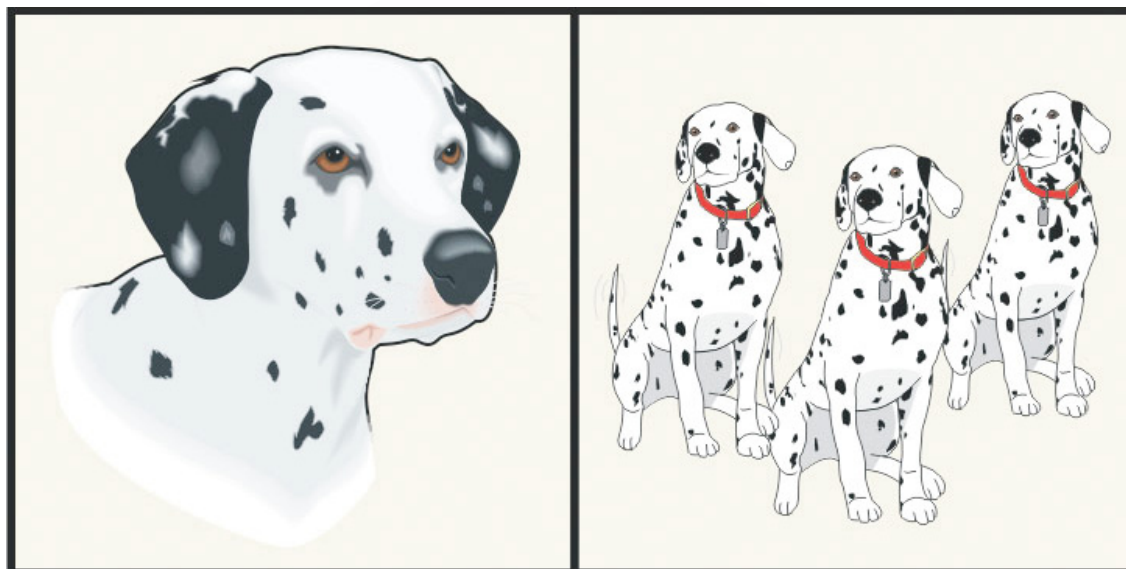


ILUSTRAÇÃO: EDUARDO JOSÉ DE SOUZA ENGELMANN

Se analisarmos os mamíferos, que estão dentro da classe de animais, notaremos atributos mais particulares, como tipo de dente, pelos e glândulas mamárias. Os mamíferos são classificados como uma classe derivada dos animais, que por sua vez são uma classe base de mamíferos.

Pela chamada hierárquica de classes, a classe derivada mamíferos recebe todos os atributos de animais, partindo do princípio que uma classe derivada recebe por herança todos os atributos de seus ancestrais.

No exemplo seguinte, temos a hierarquia de classes do cachorro:

Reino	Animal
Filo	Cordata (cão, lobo, raposa, gato, cavalo, cobra, sapo, peixe)
Classe	Mammalia (cão, lobo, raposa, gato, cavalo)
Ordem	Carnívora (cão, lobo, raposa, gato)
Família	Canídea (cão, lobo, raposa)
Gênero	Canis (cão, lobo)
Espécie	Canis familiares (cão): dalmata, pastor, dobermann etc.

Adotemos como exemplo um cão da raça **dalmata**, pertencente à espécie **Canis familiares**, como visto no exemplo anterior. Por herança, o dalmata herda as características do gênero **Canis**, que por sua vez herda as características da família **Canídea**, que por sua vez herda as características da ordem **Carnívora**, e assim sucessivamente.


2.4.1. Herança simples

A herança simples determina que uma classe herdará características de apenas uma superclasse.

2.4.2. Herança múltipla

A herança múltipla determina que uma classe herdará características de duas ou mais superclasses, como é o caso das classes derivadas, quando herdarem duas ou mais classes base.

Quando utilizamos a herança múltipla, é importante observarmos a manipulação de nomes de membros duplicados nas classes base. Para especificar a qual membro a classe derivada se refere, utilizamos o mecanismo chamado qualificação. A qualificação consiste na prefixação do nome do membro com o nome da classe base a que ele faz referência. Os nomes do membro (atributo ou método) e da classe base são separados por um duplo dois-pontos (::).



Existem muitas discussões sobre a questão da utilização da herança múltipla e suas vantagens. Um ponto importante a ressaltar é que os exemplos para ilustrar a utilidade da herança múltipla, em muitos casos, são pouco naturais. Contudo, não se coloca em questão o mecanismo de herança múltipla em si, mas sim a sua validade como ferramenta de modelagem de aplicações. Sua utilização acaba sendo uma questão de escolha pessoal, visto que em diversas linguagens o mecanismo está presente.

2.4.3. Classes abstratas

Entendemos por classes abstratas as classes a partir das quais não é possível realizar qualquer tipo de instância. São classes feitas especialmente para serem modelos para suas classes derivadas. As classes derivadas, via de regra, deverão sobrescrever os métodos para realizar a implementação dos mesmos. As classes derivadas das classes abstratas são conhecidas como **Classes Concretas**.

Como medidas de segurança, as classes abstratas podem ser somente estendidas e a criação de um objeto a partir da mesma é um procedimento evitado. Além disso, caso um ou mais métodos abstratos estejam presentes nessa classe abstrata, a classe filha será, então, forçada a definir tais métodos, pois, caso contrário, a classe filha também se tornará abstrata.

A funcionalidade dos métodos abstratos que são herdados pelas classes filha normalmente é atribuída de acordo com o objetivo ou o propósito dessas classes. É possível, porém, não atribuirmos uma funcionalidade a esses métodos abstratos. Nesse caso, faz-se necessário, pelo menos, declará-los.

Os métodos abstratos estão presentes somente em classes abstratas, e são aqueles que não possuem implementação.

2.5. Persistência

A persistência de um objeto é o tempo que este permanece armazenado na memória RAM (principal) ou auxiliar (um meio magnético, por exemplo).

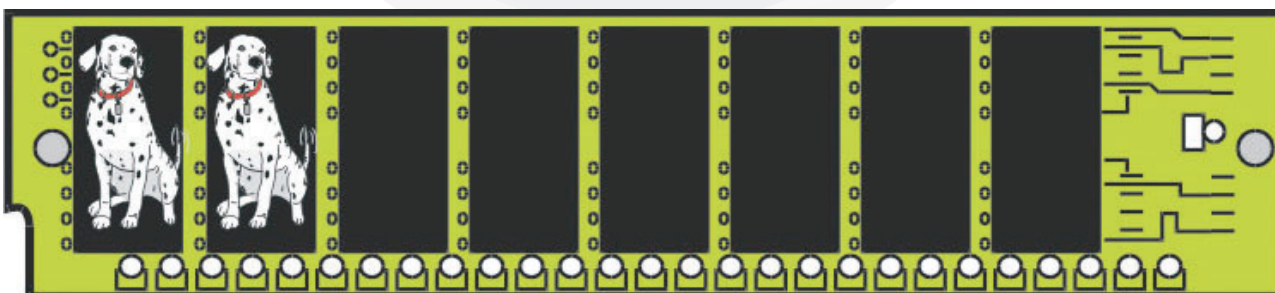


ILUSTRAÇÃO: EDUARDO JOSÉ DE SOUZA ENGELMANN

Um objeto torna-se persistente a partir do momento em que os valores de seus atributos e as informações necessárias para a manutenção de suas conexões com outros objetos são armazenados em uma mídia qualquer.

A persistência refere-se mais a um conceito do que a uma técnica de análise ou de desenvolvimento.

Os objetos criados e destruídos sem que sejam salvos de alguma maneira são chamados **objetos não persistentes**. Esses objetos são temporários, já que os valores de seus atributos são transientes e, por isso, não existe a necessidade ou o interesse de que sejam gravados.

2.6. Abstração

A abstração ocorre quando nos focamos nos aspectos essenciais de uma entidade e ignoramos propriedades secundárias. No desenvolvimento de sistemas, a abstração consiste em nos concentrarmos no que o objeto é e executa, para somente depois tomar decisões sobre sua implementação.

Por meio da abstração, isolamos o objeto que desejamos representar de um ambiente complexo e representamos nele apenas as características relevantes, de acordo com o problema atual.



ILUSTRAÇÃO: EDUARDO JOSÉ DE SOUZA ENGELMANN

Assim, ao utilizarmos os conceitos de abstração, as decisões relacionadas ao desenvolvimento e à implementação de objetos podem ser tomadas quando entendemos melhor o problema a ser resolvido. Portanto, a abstração é um dos elementos mais importantes da programação orientada a objetos.

Podemos dizer que a abstração é um processo por meio do qual separamos os fatos relevantes dos detalhes que não têm grande importância. É um conceito bastante adequado em situações de grande complexidade.

Utilizando a abstração, podemos desconsiderar alguns detalhes para ressaltar outros, visto que determinadas partes do problema ganham maior ou menor peso.

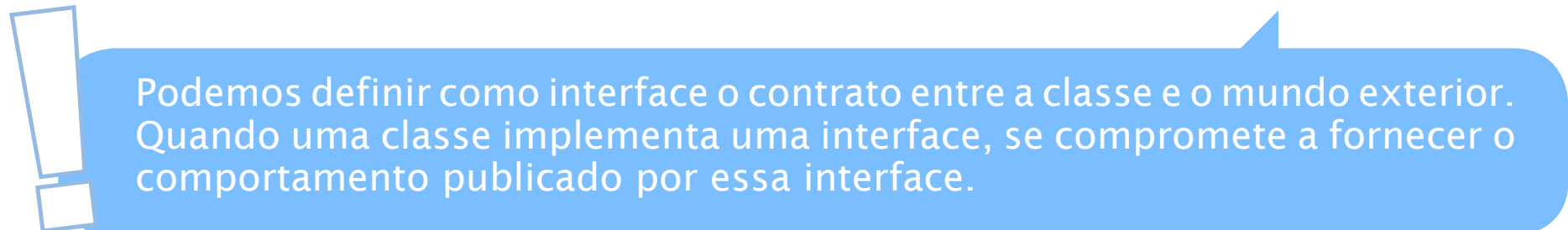
A utilização apropriada da abstração permite utilizar em todas as fases de desenvolvimento do sistema (desde a sua análise até a sua documentação) um mesmo modelo conceitual (orientação a objetos).

2.7. Encapsulamento

Diferentemente do que ocorre na programação tradicional, na programação orientada a objetos, no modo de utilização dos atributos e métodos, os dados e as operações realizadas com esses dados são encapsulados em uma única entidade. Assim, a única forma de conhecer ou alterar os atributos de um objeto é por meio de seus métodos. A lista a seguir descreve as vantagens do encapsulamento:

- O objeto é disponibilizado com toda a sua funcionalidade, sem a necessidade de conhecermos seu funcionamento ou armazenamento interno;
- É possível modificar um objeto internamente, acrescentando métodos, sem que isso afete os outros componentes do sistema que utilizam o objeto modificado;
- O processo de desenvolvimento de sistemas é acelerado e simplificado, já que os usuários dos objetos não precisam necessariamente saber como eles são constituídos internamente;
- A implementação de um comportamento pode ser modificada radicalmente sem que haja impacto no resto do programa. Isso é possível porque o código que utiliza o objeto não depende da maneira como ele é implementado.

Para que dois objetos possam interagir, é necessário que um conheça o conjunto de operações disponíveis no outro (interface), e vice-versa, para que possam enviar e receber informações, além de ordenarem a realização de procedimentos.

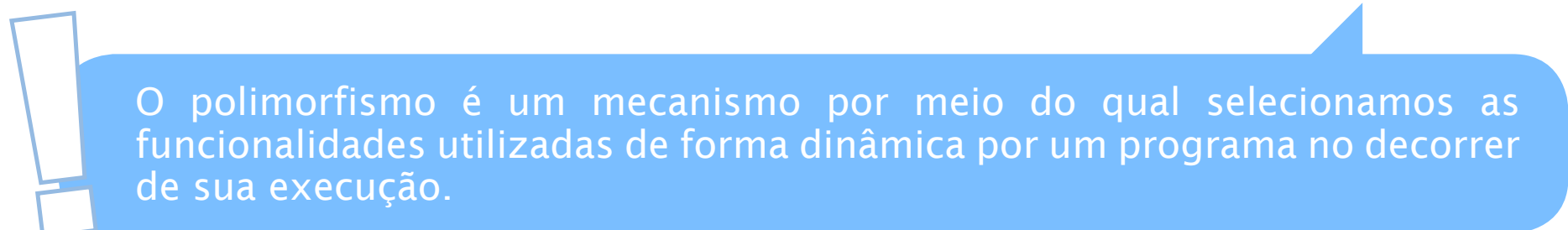


Podemos definir como interface o contrato entre a classe e o mundo exterior. Quando uma classe implementa uma interface, se compromete a fornecer o comportamento publicado por essa interface.

2.8. Polimorfismo

Definimos polimorfismo como um princípio a partir do qual as classes derivadas de uma única classe base são capazes de invocar os métodos que, embora apresentem a mesma assinatura, comportam-se de forma específica para cada uma das classes derivadas.

Com o polimorfismo, os mesmos atributos e objetos podem ser utilizados em objetos distintos, porém, com implementações lógicas diferentes. Por exemplo, podemos assumir que um triângulo e um retângulo são tipos de polígonos e que o cálculo da área de cada um é realizado de forma diferente.



O polimorfismo é um mecanismo por meio do qual selecionamos as funcionalidades utilizadas de forma dinâmica por um programa no decorrer de sua execução.

Para exemplificar o polimorfismo, podemos dizer que uma classe chamada **Cliente** e outra chamada **Funcionario** podem ter como base uma classe chamada **Pessoa**, com um método chamado **Enviar Email**. Se esse método (definido na classe base) se comportar de maneira diferente para as chamadas feitas a partir de uma instância de **Cliente** e para as chamadas feitas a partir de uma instância de **Funcionario**, ele será considerado um método polimórfico, ou seja, um método de várias formas.

2.9. Compartilhamento

Na orientação a objetos, existem técnicas que possibilitam o compartilhamento em vários níveis. Comportamento e herança de estrutura de dados permitem o compartilhamento de estruturas comuns entre classes similares diversas, sem redundância. A utilização da herança no compartilhamento do código traz duas vantagens:

- Economia de código;
- Redução dos casos distintos para análise, já que há uma maior clareza conceitual a partir do reconhecimento de que operações diferentes estão relacionadas ao mesmo elemento.

O desenvolvimento orientado a objetos permite, além do compartilhamento de informação dentro de um projeto, o reaproveitamento de códigos (e até mesmo de projetos) em projetos futuros. A metodologia disponibiliza ferramentas que possibilitam o compartilhamento, como a abstração, a encapsulação e a herança. A reutilização entre projetos pode utilizar como estratégia a criação de bibliotecas de elementos reutilizáveis, além do pensamento do desenvolvedor voltado para termos genéricos, ou seja, não focado apenas na aplicação atual.