Orientação a Objetos

Introdução à Orientação a Objetos

- Programação orientada a objetos é um método de implementação no qual programas são organizados como coleções cooperativas de objetos, sendo que cada um desses objetos representa uma instância de uma classe. Essas classes são membros de uma hierarquia de classes unidas via relações de herança.

- Partes importantes da definição anterior:
  - Programação orientada a objetos usa objetos, não algoritmos, como peças fundamentais de construção;
  - Cada objeto é uma instância de alguma classe;
  - Classes podem estar relacionadas entre si através de relacionamentos de herança.

- Se em algum programa, falta qualquer um desses elementos, o mesmo não pode ser orientado a objetos.
  - Exemplo: programação sem herança não é programação orientada a objetos, mas somente programação com tipos abstratos de dados.

- Projeto orientado a objetos é um método de projeto que engloba o processo de decomposição orientada a objetos e uma notação descritiva tanto lógica quanto física bem como modelos estáticos e dinâmicos do sistema em desenvolvimento.

- Partes importantes da definição anterior:
  - Projeto orientado a objetos leva a uma decomposição orientada a objetos;
  - Utiliza diferentes notações para expressar modelos diferentes da lógica de criação de um sistema, em adição aos aspectos estáticos e dinâmicos do sistema.
    - Modelos lógicos:
      - Estruturas de classes;
      - Estruturas de objetos.
    - Modelos físicos:
      - Arquitetura de modelos;
      - Arquitetura de processos.

- O suporte para a decomposição orientada a objetos é o que faz o projeto orientado a objetos diferente do projeto estruturado.

- Para estruturar sistemas logicamente:
  - o projeto orientado a objetos utiliza abstrações de classes e objetos.
  - o projeto estruturado utiliza abstrações algorítmicas.

- A análise orientada a objetos é um método de análise que examina requisitos pela perspectiva de classes e objetos encontrados no vocabulário do domínio do problema.

- Os produtos da análise orientada a objetos servem como modelos pelos quais é possível iniciar o projeto orientado a objetos;

- Os produtos do projeto orientado a objetos servem como plano para a implementação completa de um sistema utilizando a programação orientada a objetos.

- A programação orientada a objetos se baseia na composição e interação entre diversas unidades de software chamadas de objetos.

- Nós humanos estamos sempre identificando objetos ao nosso redor. Nós damos nomes para esses objetos e, de acordo com as suas características, classificamos esses objetos em grupos, ou seja, classes.

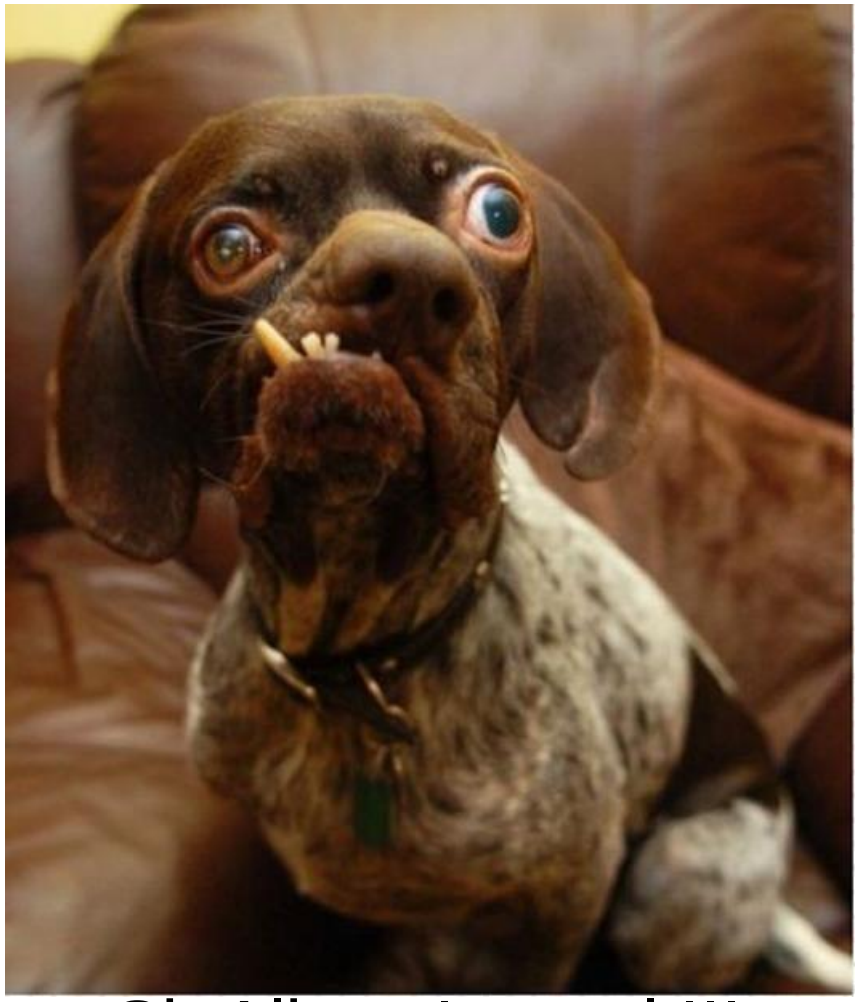
O que é isso?

Bob

O que é isso?

Pipoca

O que é isso?



Sir Albert Leonel III

- Você respondeu "Um cachorro" para as três fotos. Cachorro é uma classe. Porém, foram apresentados três cachorros diferentes. Cada cachorro apresentado é um objeto. Todo objeto é uma instância de uma classe. Dessa forma, cada cachorro apresentado é uma instância da classe Cachorro.

- A classe Cachorro define atributos e comportamentos. Como atributos da classe Cachorro podemos considerar, por exemplo: nome, cor do pelo e raça. Como comportamento podemos considerar: latir e correr.

- Para não confundirmos o real significado da palavra comportamento, vamos evitá-la nesse primeiro momento. Na orientação a objetos estamos mais interessados em saber o estado (os atributos) de um objeto e enviar mensagens para esses objetos. Então vamos pensar que a classe cachorro define atributos e mensagens.

- Logo, para facilitar o entendimento, vamos pensar em latir e correr como ordens e não como comportamento. Ao término dessa aula, vamos notar que mensagens não são somente ordens, mas para entendermos melhor os conceitos apresentados até este ponto, vamos considerar que essas mensagens são ordens dadas ao cachorro. Dessa forma, latir e correr são ordens.

- Podemos melhorar o exemplo? Sim! Vamos continuar considerando os atributos nome, raça e cor do pelo e, como mensagens, vamos considerar sentar, deitar e fingir de morto.

- Um objeto é capaz de armazenar estados, reagir a mensagens enviadas a ele mesmo e enviar mensagens a outros objetos. Um objeto armazena estados utilizando, para isso, os seus atributos.

- Agora saindo do mundo natural e pensando em nossos programas, poderíamos criar uma classe Aluno para representar um aluno em um sistema. Essa classe poderia ter uma série de atributos, dentre os quais destacaremos nome, data de nascimento e matrícula.

- Além de aluno, nós poderíamos criar uma classe Turma, que possuiria os atributos período e alunos. Nós poderíamos matricular um aluno em uma Turma, logo podemos ter a mensagem MatricularAluno na classe Turma. Ao enviar a mensagem MatricularAluno, uma instância da classe Aluno deve ser passada.

- Como podemos fazer isso?

- Exemplos de definição de classes em C#:

```
class Aluno
{

}

class Turma
{

}
```

- Para criar uma instância de uma classe, ou seja, para criarmos um objeto, utilizamos a seguinte sintaxe: Classe identificadorObjeto = new Classe();

- Dessa forma, para criarmos um objeto da classe Aluno, utilizaremos o código: Aluno a = new Aluno();

- Agora vamos utilizar variáveis para definirmos os atributos de nossas classes:

```
class Aluno
{
      string nome;
      int matricula;
      DateTime dataNascimento;
}

class Turma
{
      int periodo;
      Aluno[] alunos = new Aluno[100];
}
```

- Precisamos utilizar um modificador de acesso para os nossos atributos. Nesse momento, utilizaremos o modificador public para que nossos atributos sejam públicos, ou seja, para que possam ser acessados de fora da classe em que estão contidos. Isso não é uma boa prática, como veremos posteriormente.

```
class Aluno
{
      public string nome;
      public int matricula;
      public DateTime dataNascimento;
}

class Turma
{
      public int periodo;
      public Aluno[] alunos = new Aluno[100];
}
```

- As mensagens podem ser implementadas através de procedimentos e/ou funções:

```
class Aluno {
        public string nome;
        public int matricula;
        public DateTime dataNascimento;
}

class Turma {
        public int periodo;
        public Aluno[] alunos = new Aluno[100];

        public void MatricularAluno(Aluno a)
        {
                alunos[?] = a;
        }
}
```

```
class Turma
{
	public int periodo;
	public Aluno[] alunos = new Aluno[100];
	private int numeroAlunos = 0;

	public void MatricularAluno(Aluno a)
	{
		alunos[numeroAlunos] = a;
		numeroAlunos++;
	}
}
```

- Ao deixar que os valores dos nossos atributos sejam setados fora da classe Aluno, um programador pode esquecer-se de fazer uma validação do dado que está sendo inserido. Por exemplo: o usuário poderia informar como data de nascimento uma data futura, o que é impossível. Então o programador poderia criar um método para checar se a data é válida ou não, mas qual o melhor lugar para se colocar esse método? Um segundo programador se lembraria de utilizá-lo?

- Na orientação a objetos existe o conceito de encapsulamento. Logo, os atributos de suas classes não devem ser públicos.

- Para que alguém consiga utilizá-los, você deve criar métodos públicos. As validações dos dados podem ser feitas nesses métodos.

```
class Aluno {
  private string nome;
  private int matricula;
  private DateTime dataNascimento;

  public string getNome()
  {
    return nome;
  }

  public void setNome(string novoNome)
  {
    nome = novoNome;
  }

  public int getMatricula()
  {
    return matricula;
  }

  public void setMatricula(int novaMat)
  {
    matricula = novaMat;
  }

  public DateTime getDataNascimento()
  {
    return dataNascimento;
  }

  public void setDataNascimento(DateTime novaDt)
  {
    dataNascimento = novaDt;
  }
}
```

```
class Turma
{
  public int periodo;
  public Aluno[] alunos = new Aluno[100];
  private int numeroAlunos = 0;

  public int getPeriodo()
  {
    return periodo;
  }

  public void setPeriodo(int novoPer)
  {
    periodo = novoPer;
  }

  public Alunos[] getAlunos()
  {
    return alunos;
  }
```

```
  public void MatricularAluno(Aluno a)
  {
    if (numeroAlunos < 100)
    {
      alunos[numeroAlunos] = a;
      numeroAlunos++;
    }
    else
    {
      LancarErro("O número máximo de alunos foi " +
                 "alcançado!");
    }
  }
}
```

- Você deve criar uma função get para cada atributo que você desejar permitir que seus valores sejam lidos de fora da classe aluno.

- Você deve criar um procedimento set para cada atributo que você desejar permitir que seus valores sejam setados de fora da classe aluno.

```
class Turma
{
  public int periodo;
  public Aluno[] alunos = new Aluno[100];
  private int numeroAlunos = 0;

  public int getPeriodo()
  {
    return periodo;
  }

  public void setPeriodo(int novoPer)
  {
    periodo = novoPer;
  }

  public Alunos[] getAlunos()
  {
    return alunos;
  }
```

```
 public void MatricularAluno(Aluno a)
  {
    if (numeroAlunos < 100)
    {
      alunos[numeroAlunos] = a;
      numeroAlunos++;
    }
    else
    {
      LancarErro("O número máximo de alunos foi " +
                 "alcançado!");
    }
  }
}
```

- Agora, já podemos instanciar alguns alunos e matriculá-los em alguma turma instanciada.

```
Turma turmaA = new Turma();
turmaA.setPeriodo(2);

Aluno a = new Aluno();
a.setNome("João")
a.setMatricula(2015003);
a.setDataNascimento(novo Data(1995, 1, 1));

Aluno b = new Aluno();
b.setNome("Maria")
b.setMatricula(2015002);
b.setDataNascimento(new DateTime(1996, 12, 31));

turmaA.MatricularAluno(a);
turmaA.MatricularAluno(b);

Console.Write("Número de alunos na turma do {0}", turmaA.getPeriodo());
Console.Write("o período: {0}", turmaA.getNumeroAlunos());
```

- Pronto! Note que com a orientação a objetos, os atributos dos objetos ficam encapsulados e os seus acessos só são permitidos através de funções e procedimentos. Essas funções e/ou procedimentos impedem que o programador que estiver utilizando a classe insira sem perceber um valor que deixaria o objeto em um estado inconsistente.

- Note também o envio de mensagens que ocorre tanto nos métodos get e set quanto no método MatricularAluno. Entenda que o melhor lugar para o método MatricularAluno é dentro da classe Turma, pois somente um objeto da classe Turma sabe quantos alunos matriculados existem e não é possível matricular mais de 100 alunos em uma classe.

- Foi dito que a classe Turma possui um atributo alunos. Note que o tipo de dados desse atributo é um vetor de Aluno. Como Aluno é um tipo de dado criado por você mesmo, é bem provável que você queira dar um nome diferente para esse atributo para diferenciá-lo dos atributos que apenas utilizam tipos de dados contidos em sua linguagem de programação. O nome para esse relacionamento é associação. Para diferenciar melhor, vamos dar o exemplo de uma classe venda, apresentado no próximo slide.

```
class Venda
{
          private int numero;

          private Cliente cliente;

          public void setNumero(int novoNumero)
          {
               numero = novoNumero;
          }

          public int getNumero()
          {
               return numero;
          }

          public void setCliente(Cliente novoCliente)
          {
               cliente = novoCliente;
          }

          public Cliente getCliente()
          {
               return cliente;
          }
}
```

- Para a classe Venda apresentada, numero é um atributo enquanto cliente é uma associação. Note que a forma como programamos um atributo ou uma associação é idêntica.

- Geralmente, atributos são utilizados para coisas pequenas, como datas, valores booleanos, números etc.

- Associações geralmente são usadas para classes mais significativas, como clientes e pedidos.

- Essa escolha está muito mais relacionada à ênfase do que a qualquer significado subjacente.