Alexandre de Oliveira

POO

História e Conceitos

Programação Estruturada

O princípio básico da programação estruturada é que um programa pode ser divido em três partes que se interligam: sequência, seleção e iteração.

Programação Estruturada

Sequência

Na sequência são implementados os passos de processamento necessários para descrever determinada funcionalidade.

Um exemplo básico seria um fluxograma, onde primeiro é executado a Etapa 1 e após a sua conclusão a Etapa 2 é executada, e assim por diante.

Programação Estruturada

Seleção

Na seleção o fluxo a ser percorrido depende de uma escolha. Existem duas formas básicas para essa escolha.

- A primeira é através do condicional “Se”, onde se uma determinada condição for satisfatória o fluxo a ser corrido é um e, caso contrário, o fluxo passa a ser outro. Ou seja, se o fluxo só percorre apenas um caminho, apenas uma ação é processada.
- A outra forma de escolha é onde o número de condições se estende a serem avaliadas. Por exemplo, se a Condição 1 for verdade faça Processamento 1, caso contrário, se a Condição 2 for verdade faça Processamento 2, caso contrário, se a Condição 3 for verdade faça Processamento 3, e assim por diante.

Programação Estruturada

Iteração

Na iteração é permitido a execução de instruções de forma repetida, onde ao fim de cada execução a condição é reavaliada e enquanto seja verdadeira a execução de parte do programa continua.

Programação Estruturada

Modularização

A medida que o sistema vai tomando proporções maiores, é mais viável que o mesmo comece a ser divido em partes menores, onde é possível simplificar uma parte do código deixando a compreensão mais clara e simplificada. Essa técnica ficou conhecida como Subprogramação ou Modularização. No desenvolvimento utilizamos essa técnica através de procedimentos, funções, métodos, rotinas e uma série de outras estruturas. Com essa divisão do programa em partes podemos extrair algumas vantagens, como:

Cada divisão possui um código mais simplificado;
Facilita o entendimento, pois as divisões passam a ser independentes;
Códigos menores são mais fáceis de serem modificados;
Desenvolvimento do sistema através de uma equipe de programadores;
Reutilização de trechos de códigos.

- **Alan Curtis Kay** (Springfield, 17 de maio de 1940) é um informático estadunidense.

É conhecido por ter sido um dos inventores da linguagem de programação Smalltalk, e um dos pais do conceito de programação orientada a objetos, que lhe valeu o Prêmio Turing em 2003. Concebeu o laptop e a arquitetura das modernas interfaces gráficas dos computadores (GUI).

https://pt.wikipedia.org/wiki/Alan_Kay

Smalltalk

- Tudo é representado como objetos. (De longe, a regra mais importante em Smalltalk).
- Toda computação é disparada pelo envio de mensagens. Uma mensagem é enviada para um objeto fazer alguma coisa.
- Quase todas as expressões são da forma <recebedor> <mensagem>.
- Mensagens fazem com que métodos sejam executados, sendo que o mapeamento de mensagens para métodos é determinado pelo objeto recebedor. Os métodos são as unidades de código em Smalltalk, equivalente a funções ou procedimentos em outras linguagens.
- Todo objeto é uma instância de alguma classe. 12 é uma instância da classe SmallInteger. 'abc' é uma instância da classe String. A classe determina o comportamento e os dados de suas instâncias.
- Toda classe tem uma classe mãe, exceto a classe Object. A classe mãe define os dados e comportamento que são herdados por suas classes filhas. A classe mãe é chamada de superclasse e suas filhas, subclasses

Orientação à Objetos: Introdução

Platão e Aristóteles

- **Introdução a POO**
  - Antes...
  - As primeiras linguagens de programação para computadores trabalhavam sobre um paradigma estruturado.
  - Esse paradigma era pouco funcional e metodológico.
  - Dificulta a organização do código.
  - Quanto maior o software, mais difícil a manutenção.

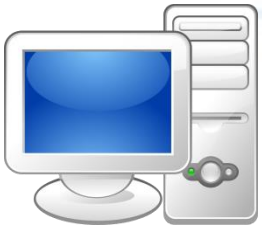
- **Introdução a POO**
  - **Programação Orientada à Objetos**
  - Um paradigma de programação de sistemas focado em reusabilidade.
  - Tenta aproximar o mundo real do mundo virtual (através da utilização de objetos).
  - Os objetos podem ser concretos e abstratos.
  - Os objetos definidos dentro de um software podem “conversar” (interagir) entre si.

- **Introdução a POO**

Objetos do Mundo real

Pessoa

Computador



Cachorro

Objetos do Software

Venda

Formulario

Funcionário

- **Introdução a POO**
  - Os objetos possuem propriedades (atributos)

Pessoa

Nome

Altura

Peso

Venda

Data

Valor

Status

Computador

Marca

Tipo

Potencia

Funcionário

Nome

Cargo

Salário

- **Introdução a POO**
  - Os objetos possuem comportamentos (métodos)

Pessoa

Andar

Falar

Comer

Computador

Ligar

Desligar

Abrir Programa

Venda

Concluir Venda

Cancelar Venda

Calcular Valor Total

Funcionário

Salvar Funcionário

Editar Funcionário

Excluir Funcionário

- **Introdução à POO**
  - **Classes**
    - É uma abstração que define o molde de uma classe de objetos.
    - Agrupam uma classe de objetos que compartilham de uma mesma série de atributos e métodos.
    - Os objetos de uma classe respeitam suas implementações.

- **Introdução à POO**
  - **Classe “Cadeira”**
    - Objetos da classe “Cadeira”

- **Introdução à POO**
  - **Classes:** Atributos
    - Os atributos são as propriedades que os objetos compartilham.
    - Possuem tipos de dados (String, int, double, boolean, etc.)
    - Funcionam como variáveis.
  - **Classes:** Métodos
    - Os métodos são os comportamentos que os objetos executam.
    - Podem ou não retornar alguma informação

- **Introdução à POO**
  - Imagine a classe **Pessoa**.

- **Atributos**
  - Nome
  - Peso
  - Altura
- **Métodos**
  - Falar
  - Andar
  - Comer

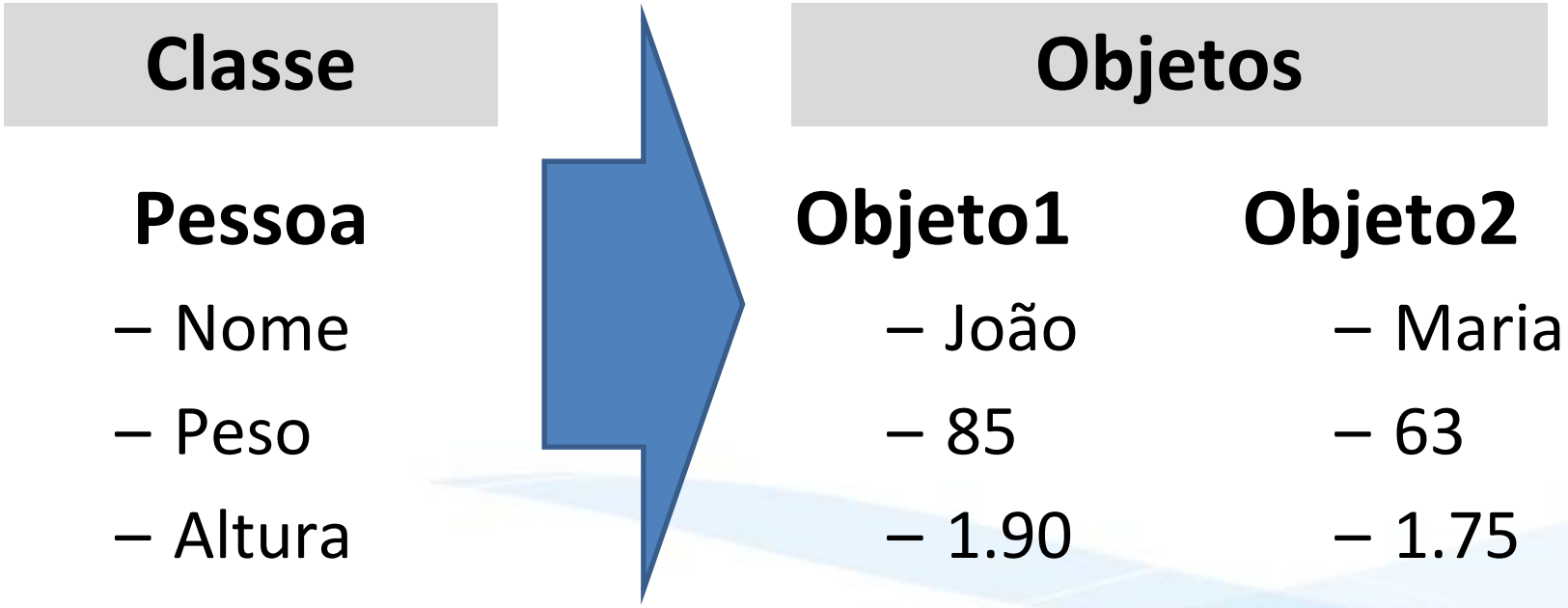
- **Introdução à POO**
  - **Exercício 1:**

    - Defina pelo menos 5 atributos para as classes à seguir:

    1. Cliente
    2. Venda
    3. Produto
    4. Aluno
    5. Curso

- **Introdução à POO**
  - **Objetos**
    - Objetos são considerados **instâncias** das classes.
    - Enquanto as classes são generalizadas, os objetos são algo específico, mas que respeitam a estrutura de uma classe.
    - Podem existir vários objetos/instâncias de uma mesma classe, mas cada um é independente.

- **Introdução à POO**
  - **Objetos:** Para entender melhor...



Implementação

- **Implementação**
  - Em um projeto Java, cada classe deve corresponder à um arquivo de códigos-fonte.
  - **Para declarar uma classe:**

```
public class Pessoa {

}
```

As chaves delimitam o início e o fim da classe.

- **Implementação**
  - **Atributos**: Para se criar os atributos de uma classe, deve-se respeitar a estrutura: *tipo de dado => nome do atributo.*

```
public class Pessoa {
    String nome;
    double peso;
    double altura;
}
```

Modificadores de Visibilidade

- Modificadores de Visibilidade
  - Os atributos e métodos de uma classe possuem modificadores de visibilidade quanto ao seus acessos.
  - Isso possibilita a restrição de acesso às características ou comportamentos.
  - Os modificadores possibilitam a utilização de um dos principais conceitos de OO. O **encapsulamento.**

- Modificadores de Visibilidade
  - Existem **3** modificadores de visibilidade:

➢ + **public**: pode ser acessado dentro e fora da classe.

➢ # **protected**: pode ser acessado por classes da mesma família

➢ - **private**: pode ser acessado apenas dentro da classe.

- Modificadores de Visibilidade
  - O modificador deve ser informado antes da declaração do atributo ou método.

```
public class Pessoa {
    public String nome;
    public double peso;
    public double altura;
}
```

- Modificadores de Visibilidade
  - Na prática!!!

Instâncias (Objetos)

- Instâncias (Objetos)
  - Como já abordado, os objetos são as instâncias de uma classe.
  - Eles são representações físicas das classes e que obedecem suas implementações.
  - Os objetos podem acessar os atributos e executar os métodos públicos da classe a qual pertencem.
  - Os objetos funcionam como “variáveis” e portanto devem possuir nome.

- Instâncias (Objetos)

```
public class Pessoa {
    public String nome;
    public double peso;
    public double altura;
}
```

Classe

```
Pessoa p1 = new Pessoa();
p1.nome = "João";
p1.peso = 85;
p1.altura = 1.9;
```

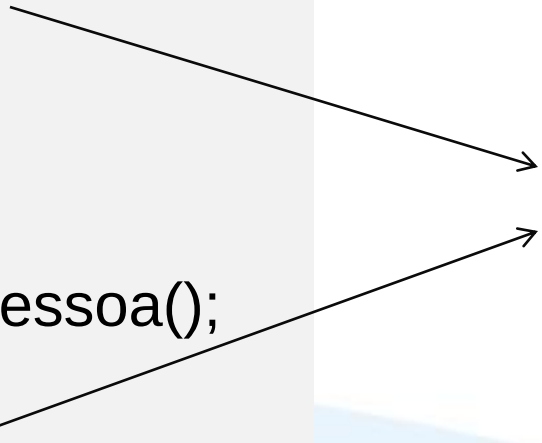
p1 é um objeto do tipo "Pessoa".

Para acessar uma propriedade/método da classe, utilize o sinal ponto (.)

- Instâncias (Objetos)

```
Pessoa p1 = new Pessoa();
p1.nome = “João”;
p1.peso = 85;
p1.altura = 1.9;

Pessoa p2 = new Pessoa();
p2.nome = “Ana”;
p2.peso = 60;
p2.altura = 1.75;
```



p1 e p2 são instâncias de uma mesma classe (Pessoa), mas ambos são independentes entre si.

- Instâncias (Objetos)
  - Na prática!!!

- Instâncias (Objetos)
  - Exercício:

1. Crie a classe Carro com os atributos públicos marca, modelo, combustível, portas e capacidade_tanque. Depois crie 5 instâncias da classe Carro e defina valores para cada um dos atributos.

Implementação de Métodos

- Implementação de Métodos
  - Os métodos são os comportamentos das classes.
  - São blocos de códigos independentes que podem ser executados sempre que “chamados”.
  - Métodos podem alterar alguma propriedade da classe ou mesmo executar outros métodos da própria classe.
  - São considerados as **funções/procedimentos** da OO.

- Implementação de Métodos

  – Algumas características dos métodos:

  - Possuem modificadores de acesso
  - Podem receber parâmetros (**dados de entrada**)
  - Podem ou não retornar alguma informação (**dados de saída**)

- Implementação de Métodos

```
public class Pessoa {

    public String nome;
    public double peso;
    public double altura;

    public void falar(){
        System.out.println("Olá");
    }
}
```

void: indica que o método não retorna nada.

- Implementação de Métodos

  – Os métodos são executados também através do sinal ponto (.)

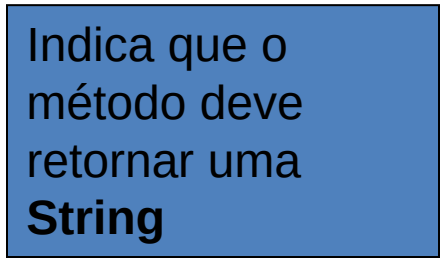
```
Pessoa p1 = new Pessoa();
p1.falar();  // Imprime “Olá”
```

- Implementação de Métodos

```
public class Pessoa {

    public String nome;
    public double peso;
    public double altura;

    public String falar(){
        String mensagem = "Olá";
        return mensagem;
    }
}
```



- Implementação de Métodos

  – Como nesse caso o método retorna uma **String**, é necessário uma variável para recebê-la.

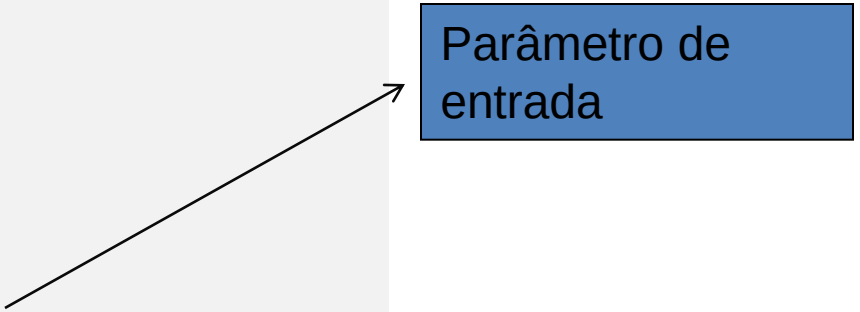
```
Pessoa p1 = new Pessoa();
String retorno;
retorno = p1.falar();  // Atribui “Olá” à variável retorno
```

- Implementação de Métodos

```
public class Pessoa {

    public String nome;
    public double peso;
    public double altura;

    public void falar(String mensagem){
            System.out.println(mensagem);
    }
}
```



- Implementação de Métodos

  – Quando um ou mais parâmetros de entrada são definidos em um método, eles deverão ser informados quando o método for executado.

```
Pessoa p1 = new Pessoa();
p1.falar("Uma mensagem qualquer");

String msg = "Outra mensagem";
p1.falar(msg);
```

- Implementação de Métodos

Indica que o método deve retornar um **int**

```
public class Calcular{

    public int somar(int num1, int num2){
        return num1 + num2;
    }
}
```

```
Calcular c = new Calcular();
int resultado;
resultado = c.somar(12, 13);
```

- Implementação de Métodos

  – Exercício:

2. Crie uma classe chamada “Matematica”. Nessa classe, implemente os métodos “somar”, “subtrair”, “multiplicar” e “dividir”. Cada um desses métodos deverá receber 2 inteiros como parâmetro e imprimir o respectivo resultado.

- Implementação de Métodos

  – Exercício:

3. Crie uma classe chamada “CalculaArea”. Dentro dessa classe, crie os atributos altura e largura do tipo inteiro. Crie um método chamado **calcularArea** que deverá retornar a área de um local. (Área = largura * altura). Crie uma instância da classe e faça simulações para testar o cálculo de uma área.

- Implementação de Métodos

  – Exercício:

4. Crie uma classe chamada Aluno. Essa classe deverá conter os atributos nomeAluno, notaExercicio, notaTrabalho e notaProva. A classe ainda deverá ter o método calculaMedia, que deverá receber como parâmetro os pesos para exercício, trabalho e prova (**pe**, **pt** e **pp**). A soma dos pesos deverá ser igual a 1, caso contrário o método imprime uma mensagem de erro. Recebendo esses parâmetros o método deverá calcular e imprimir a média final do aluno.

Polimorfismo

- Polimorfismo
  - O polimorfismo costuma ser chamado de o terceiro pilar da programação orientada a objetos, depois do encapsulamento e a herança. O polimorfismo é uma palavra grega que significa "de muitas formas" e tem dois aspectos distintos:
    - Em tempo de execução, os objetos de uma classe derivada podem ser tratados como objetos de uma classe base, em locais como parâmetros de método, coleções e matrizes. Quando esse polimorfismo ocorre, o tipo declarado do objeto não é mais idêntico ao seu tipo de tempo de execução.
    - As classes base podem definir e implementar métodos virtuais e as classes derivadas podem substituí-los, o que significa que elas fornecem sua própria definição e implementação. Em tempo de execução, quando o código do cliente chama o método, o CLR procura o tipo de tempo de execução do objeto e invoca a substituição do método virtual. Em seu código-fonte, você pode chamar um método em uma classe base e fazer com que a versão do método de uma classe derivada seja executada.

Fonte: https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/polymorphism

- Polimorfismo Exemplo

Os métodos virtuais permitem que você trabalhe com grupos de objetos relacionados de maneira uniforme. Por exemplo, suponha que você tem um aplicativo de desenho que permite que um usuário crie vários tipos de formas sobre uma superfície de desenho. Você não sabe em tempo de compilação que tipos específicos de formas que o usuário criará. No entanto, o aplicativo precisa manter controle de todos os diferentes tipos de formas que são criados e atualizá-los em resposta às ações do mouse do usuário. Você pode usar o polimorfismo para resolver esse problema em duas etapas básicas:

1. Crie uma hierarquia de classes em que cada classe de forma específica derive de uma classe base comum.
2. Use um método virtual para invocar o método adequado em qualquer classe derivada por meio de uma única chamada para o método da classe base.
3. Primeiro, crie uma classe base chamada Shape e as classes derivadas como *Rectangle*, *Circle* e *Triangle*. Atribua à classe Shape um método virtual chamado Draw e substitua-o em cada classe derivada para desenhar a forma especial que a classe representa. Crie List<Shape> um objeto Circlee Triangleadicione Rectangle um , e a ele.

Fonte: https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/polymorphism

- Polimorfismo Exemplo

```
public class Shape
{
    // A few example members
    public int X { get; private set; }
    public int Y { get; private set; }
    public int Height { get; set; }
    public int Width { get; set; }

    // Virtual method
    public virtual void Draw()
    {
        Console.WriteLine("Performing base class drawing tasks");
    }
}

public class Circle : Shape
{
    public override void Draw()
    {
        // Code to draw a circle...
        Console.WriteLine("Drawing a circle");
        base.Draw();
    }
}
public class Rectangle : Shape
{
    public override void Draw()
    {
        // Code to draw a rectangle...
        Console.WriteLine("Drawing a rectangle");
        base.Draw();
    }
}
public class Triangle : Shape
{
    public override void Draw()
    {
        // Code to draw a triangle...
        Console.WriteLine("Drawing a triangle");
        base.Draw();
    }
}
```

Fonte: https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/polymorphism

- Polimorfismo Exercício

1. Crie uma classe abstract Figura;
2. Crie dois métodos abstract CalcularArea() e CalcularPerimetro() que (nas derivadas) retornarão os valores da área e do perímetro;
3. Crie as classes herdadas de Figura e reescreva CalcularArea() e CalcularPerimetro() para cada uma delas:
    1. Quadrado
        a) Perímetro:
            i. $P = b \times 4$
            ii. onde: P = Perímetro; b = base.
        b) Área:
            i. $A = b \times b$;
            ii. onde: A = Área; b = base.
        c) Precisa de uma propriedade chamada b;
    2. Circulo
        a) Perímetro:
            i. $P = 2 \times \pi \times r$
            ii. onde: P = Perímetro; r = raio do círculo;
        b) Área:
            i. $A = \pi r^2$
            ii. onde: A = área; r = raio do círculo;
        c) Precisa de uma propriedade chamada r;
4. Para as demais classes, pesquise os cálculos de área e perímetro e deduza as propriedades necesárias
    1. Pentagono
    2. Hexagono
    3. Heptagono
    4. Octogono.

- Leitura Complementar:
  - https://www.caelum.com.br/apostila-csharp-orientacao-objetos/#null

Bibliografia Básica

- CARVALHO, Adelaide. Praticas De C# Programação Orientada Por Objetos. Barueri: FCA, 2011.

- SIERRA, Kathy; BATES, Bert. Use a Cabeça! Java. 2. ed. Rio de Janeiro: Alta Books, 2005. 496 p.

- HORSTMAN, Cay S.; CORNELL, Gary. Core Java: Volume 1. 8. ed. São Paulo: Pearson, 2010. 400 p.

- DEITEL, Paul; DEITEL, Harvey. Java: Como Programar. 8. ed. São Paulo: Pearson, 2010. 1176 p.

Obrigado

QUESTÕES

Alexandre de Oliveira (Montanha)

alexmontanha@hotmail.com