Centro Estadual de Educação Profissional Professora Amazonina Teixeira de Carvalho

Disciplina:

Programação Estruturada e Orientada a Objetos

Professor: Jedson Diogo Nascimento Silva

O que é POO?

Programação **O**rientada a **O**bjetos

Professor: Jedson Diogo Nascimento Silva

Qual o Objetivo?

Aproximar o Mundo Digital do Mundo Real

Centro Estadual de Educação Profissional Professora Amazonina Teixeira de Carvalho

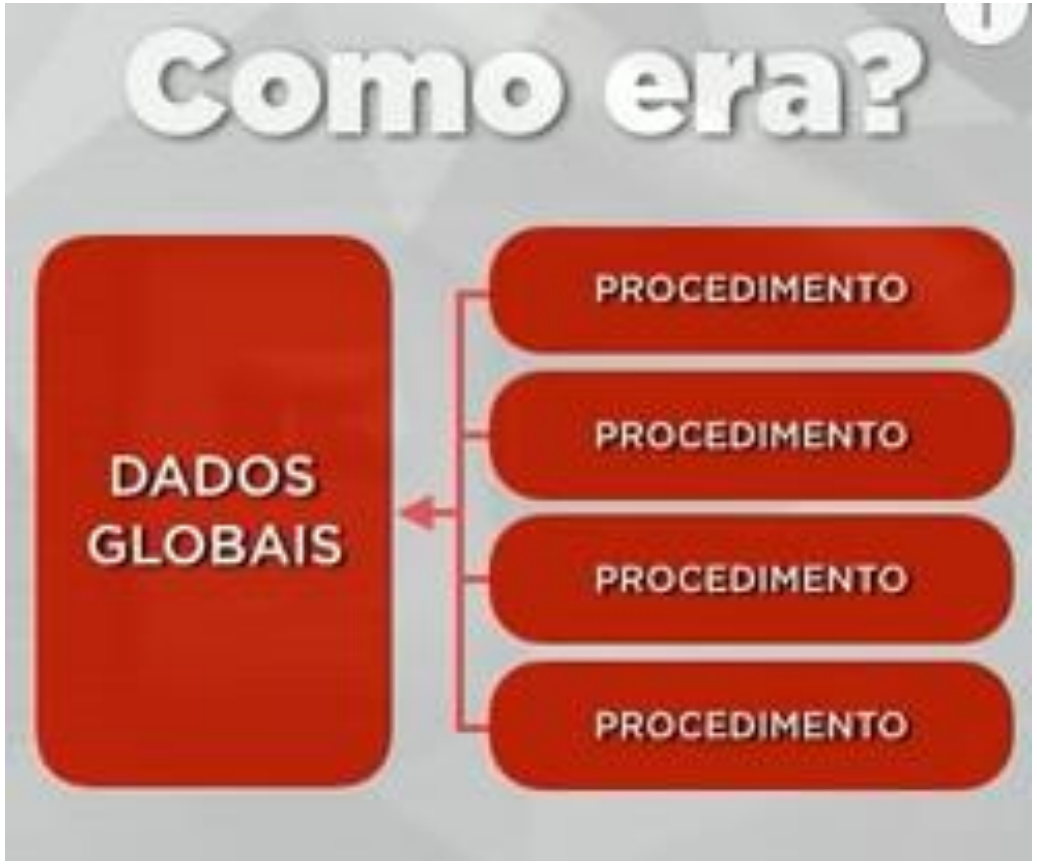
Década		
50	• Programação em Baixo Nível (binários) • Programação Linear	• Muito complicado, ficando restrito somente a engenheiros • Bem parecido com uma lista de compras.
70	• Programação estruturada • Programação modular	Com a evolução da Programação Linear, deu origem a programação estruturada, permitia pedaços de programação linear organizados de maneira que poderia ser executado em qualquer lugar do código. Na Programação Modular, era encapsulado trechos de códigos porem durou pouco tempo.
70	Programação orientada a Objetos	Criada por Alan Kay (Matemático e biólogo). “ O computador ideal deve funcionar como um organismo vivo, isso é, cada célula se relaciona com outras a fim de alcançar um objetivo, mas cada uma funciona de forma autônoma. As células poderiam também reagrupar-se para resolver um outro problema ou desempenhar outras funções.”

Professor: Jedson Diogo Nascimento Silva

“ O computador ideal deve funcionar como um organismo vivo, isso é, cada célula se relaciona com outras a fim de alcançar um objetivo, mas cada uma funciona de forma autônoma. As células poderiam também reagrupar-se para resolver um outro problema ou desempenhar outras funções.”

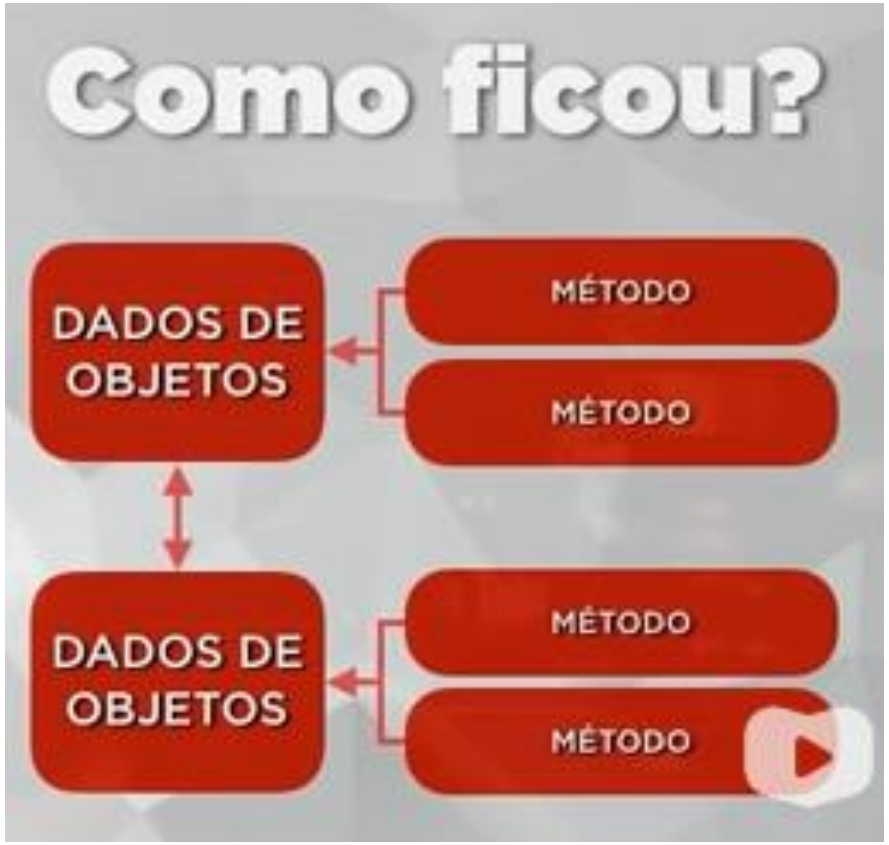
- Alan Kay

Centro Estadual de Educação Profissional Professora Amazonina Teixeira de Carvalho



Professor: Jedson Diogo Nascimento Silva

Centro Estadual de Educação Profissional Professora Amazonina Teixeira de Carvalho



Professor: Jedson Diogo Nascimento Silva

Linguagens que utilizam POO?

- C++
- JAVA
- Python
- Ruby
- PHP
- JavaScript
- .Net

Vantagens:

Professor: Jedson Diogo Nascimento Silva

Confiabilidade: ***todo software que utiliza POO é confiável!***

O Isolamento entre as partes gera softwares seguros. Ao alterar uma parte, nenhuma outra é alterada.

Oportuno: ***todo software que utiliza POO é oportuno!***

Ao dividir tudo em partes, várias delas porem ser desenvolvidas em paralelo.

Manutenível: ***todo software que utiliza POO é de Facil manutenção!***

Atualizar um Software é mais fácil. Uma pequena modificação vai beneficiar todas as partes que usarem o objeto.

Extensível: ***todo software que utiliza POO é Extensível!***

O software não é estático. Ele deve crescer para permanecer útil.

Reutilizável: ***todo software que utiliza POO é reutilizável!***

Podemos usar objetos de um sistema que criamos em outro sistema futuro.

Natural: ***todo software que utiliza POO é Natural!***

Mais fácil de entender. O programador se preocupa mais na funcionalidade do que nos detalhes de implementação.

Centro Estadual de Educação Profissional Professora Amazonina Teixeira de Carvalho

Os 4 pilares da Programação Orientada a Objetos

Professor: Jedson Diogo Nascimento Silva

Abstração

A abstração consiste em um dos pontos mais importantes dentro de qualquer linguagem **Orientada a Objetos**. Como estamos lidando com uma representação de um objeto real (o que dá nome ao paradigma), temos que imaginar o que esse objeto irá realizar dentro de nosso sistema. São três pontos que devem ser levados em consideração nessa abstração.

O primeiro ponto é darmos uma ***identidade*** ao objeto que iremos criar. Essa identidade deve ser única dentro do sistema para que não haja conflito. Na maior parte das linguagens, há o conceito de pacotes (ou namespaces). Nessas linguagens, a identidade do objeto não pode ser repetida dentro do pacote, e não necessariamente no sistema inteiro. Nesses casos, a identidade real de cada objeto se dá por <nome_do_pacote>.<nome_do_objeto>.</nome_do_objeto></nome_do_pacote>

A segunda parte diz respeito a características do objeto. Como sabemos, no mundo real qualquer objeto possui elementos que o definem. Dentro da programação orientada a objetos, essas características são nomeadas ***propriedades***. Por exemplo, as propriedades de um objeto "Cachorro" poderiam ser "Tamanho", "Raça" e "Idade".

Por fim, a terceira parte é definirmos as ações que o objeto irá executar. Essas ações, ou eventos, são chamados ***métodos***. Esses métodos podem ser extremamente variáveis, desde "Acender()" em um objeto lâmpada até "Latir()" em um objeto cachorro.

Encapsulamento

O *encapsulamento* é uma das principais técnicas que define a programação orientada a objetos. Se trata de um dos elementos que adicionam segurança à aplicação em uma programação orientada a objetos pelo fato de esconder as propriedades, criando uma espécie de caixa preta.
A maior parte das linguagens orientadas a objetos implementam o encapsulamento baseado em propriedades privadas, ligadas a métodos especiais chamados *getters* e *setters*, que irão retornar e setar o valor da propriedade, respectivamente. Essa atitude evita o acesso direto a propriedade do objeto, adicionando uma outra camada de segurança à aplicação.
Para fazermos um paralelo com o que vemos no mundo real, temos o encapsulamento em outros elementos. Por exemplo, quando clicamos no botão ligar da televisão, não sabemos o que está acontecendo internamente. Podemos então dizer que os métodos que ligam a televisão estão encapsulados.

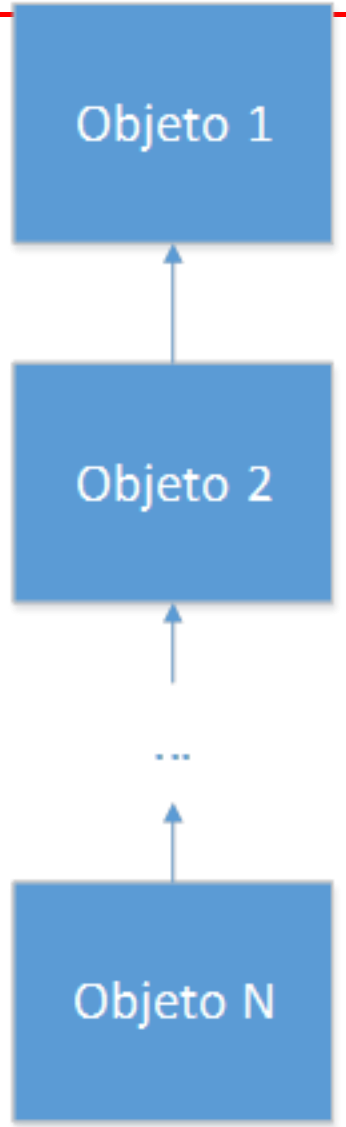
Herança

O reuso de código é uma das grandes vantagens da programação orientada a objetos. Muito disso se dá por uma questão que é conhecida como *herança*. Essa característica otimiza a produção da aplicação em tempo e linhas de código.
Para entendermos essa característica, vamos imaginar uma família: a criança, por exemplo, está herdando características de seus pais. Os pais, por sua vez, herdam algo dos avós, o que faz com que a criança também o faça, e assim sucessivamente. Na orientação a objetos, a questão é exatamente assim, como mostra a **Figura 2**. O objeto abaixo na hierarquia irá herdar características de todos os objetos acima dele, seus "ancestrais". A herança a partir das características do objeto mais acima é considerada herança direta, enquanto as demais são consideradas heranças indiretas. Por exemplo, na família, a criança herda diretamente do pai e indiretamente do avô e do bisavô.

Centro Estadual de Educação Profissional Professora Amazonina Teixeira de Carvalho

A questão da herança varia bastante de linguagem para linguagem. Em algumas delas, como C++, há a questão da herança múltipla. Isso, essencialmente, significa que o objeto pode herdar características de vários "ancestrais" ao mesmo tempo diretamente. Em outras palavras, cada objeto pode possuir quantos pais for necessário. Devido a problemas, essa prática não foi difundida em linguagens mais modernas, que utilizam outras artimanhas para criar uma espécie de herança múltipla.

Outras linguagens orientadas a objetos, como C#, trazem um objeto base para todos os demais. A classe *object* fornece características para todos os objetos em C#, sejam criados pelo usuário ou não.



Professor: Jedson Diogo Nascimento Silva

Polimorfismo

Outro ponto essencial na programação orientada a objetos é o chamado polimorfismo. Na natureza, vemos animais que são capazes de alterar sua forma conforme a necessidade, e é dessa ideia que vem o polimorfismo na orientação a objetos. Como sabemos, os objetos filhos herdam as características e ações de seus “ancestrais”. Entretanto, em alguns casos, é necessário que as ações para um mesmo método seja diferente. Em outras palavras, o *polimorfismo* consiste na alteração do funcionamento interno de um método herdado de um objeto pai.
Como um exemplo, temos um objeto genérico “Eletrodoméstico”. Esse objeto possui um método, ou ação, “Ligar()”. Temos dois objetos, “Televisão” e “Geladeira”, que não irão ser ligados da mesma forma. Assim, precisamos, para cada uma das classes filhas, reescrever o método “Ligar()”..

Com relação ao polimorfismo, valem algumas observações. Como se trata de um assunto que está intimamente conectado à herança, entender os dois juntamente é uma boa ideia. Outro ponto é o fato de que as linguagens de programação implementam o polimorfismo de maneiras diferentes. O C#, por exemplo, faz uso de método virtuais (com a palavra-chave *virtual*) que podem ser reimplementados (com a palavra-chave *override*) nas classes filhas. Já em Java, apenas o atributo "@Override" é necessário.

Esses quatro pilares são essenciais no entendimento de qualquer linguagem orientada a objetos e da orientação a objetos como um todo. Cada linguagem irá implementar esses pilares de uma forma, mas essencialmente é a mesma coisa. Apenas a questão da herança, como comentado, que pode trazer variações mais bruscas, como a presença de herança múltipla. Além disso, o encapsulamento também é feito de maneiras distintas nas diversas linguagens, embora os *getters* e *setters* sejam praticamente onipresentes.