
Buscar

[comentários](#) [favorito \(36\)](#) [marcar como lido](#) [para impressão](#) [anotar](#)

Os 4 pilares da Programação Orientada a Objetos

Conheça nesse artigo os principais pilares, bem como as diferenças para programação estruturada e as principais vantagens da POO.



8

Curtir

178

**Gostei (31)**

(0)

O desenvolvimento de software é extremamente amplo. Nesse mercado, existem

diversas linguagens de programação, que seguem diferentes paradigmas. Um desses paradigmas é a **Orientação a Objetos**, que atualmente é o mais difundido entre todos. Isso acontece porque se trata de um padrão que tem evoluído muito, principalmente em questões voltadas para segurança e reaproveitamento de código, o que é muito importante no desenvolvimento de qualquer aplicação moderna.

A **Programação Orientada a Objetos (POO)** diz respeito a um padrão de desenvolvimento que é seguido por muitas linguagens, como C# e Java. A seguir, iremos entender as diferenças entre a POO e a **Programação Estruturada**, que era muito utilizada há alguns anos, principalmente com a linguagem C. Esse padrão se baseia em quatro pilares que veremos ao longo desse artigo. Além disso, a POO possui diversas vantagens em sua utilização, que também serão vistas e explicadas.

Programação Estruturada vs Programação Orientada a Objetos

Na **Figura 1** vemos uma comparação muito clara entre a programação estruturada e a programação orientada a objetos no que diz respeito aos dados. Repare que, no paradigma estruturado, temos procedimentos (ou funções) que são aplicados globalmente em nossa aplicação. No caso da orientação a objetos, temos métodos que são aplicados aos dados de cada objeto. Essencialmente, os procedimentos e métodos são iguais, sendo diferenciados apenas pelo seu escopo.

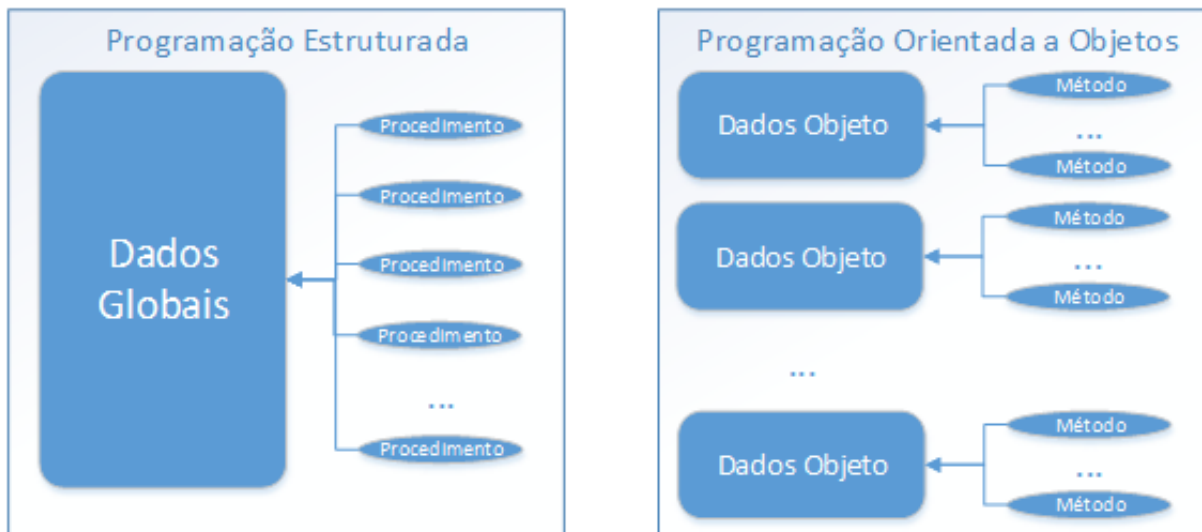


Figura 1. Estruturada x Orientação a Objetos

A linguagem C é a principal representante da programação estruturada. Se trata de uma linguagem considerada de baixo nível, que atualmente não é utilizada para projetos muito grandes. A sua principal utilização, devido ao baixo nível, é em programação para sistemas embarcados ou outros em que o conhecimento do hardware se faz necessário para um bom programa.

Essa colocação nos traz a um detalhe importante: a programação estruturada, quando bem feita, possui um desempenho superior ao que vemos na programação orientada a objetos. Isso ocorre pelo fato de ser um paradigma sequencial, em que cada linha de código é executada após a outra, sem muitos desvios, como vemos na **POO**. Além disso, o paradigma estruturado costuma permitir mais liberdades com o hardware, o que acaba auxiliando na questão desempenho.

Entretanto, a programação orientada a objetos traz outros pontos que acabam sendo mais interessantes no contexto de aplicações modernas. Como o desempenho das aplicações não é uma das grandes preocupações na maioria das aplicações (devido ao poder de processamento dos computadores atuais), a programação orientada a objetos se tornou muito difundida. Essa difusão se dá muito pela questão da reutilização de código e pela capacidade de representação do sistema muito mais perto do que

veríamos no mundo real.

Veremos em detalhes esses e outros pontos que dizem respeito a **programação orientada a objetos**. Como desenvolvedores, é nossa missão entender quais são as vantagens e desvantagens de cada um dos paradigmas de programação e escolhermos o melhor para nossa aplicação. A escolha da linguagem também deve estar presente nessa escolha.

Os 4 pilares da Programação Orientada a Objetos

Para entendermos exatamente do que se trata a orientação a objetos, vamos entender quais são os requerimentos de uma linguagem para ser considerada nesse paradigma. Para isso, a linguagem precisa atender a quatro tópicos bastante importantes:

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 ..

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.

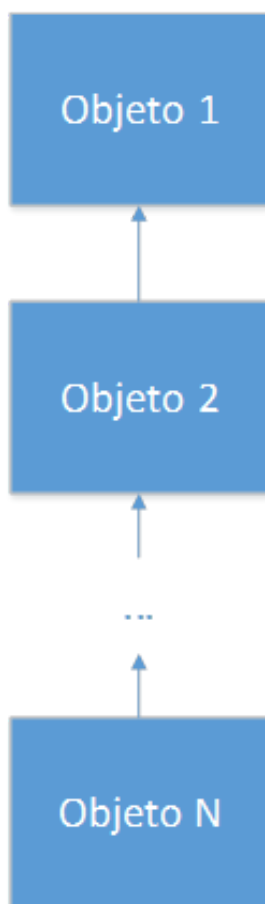


Figura 2. Herança na orientação a objetos

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.

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.

Principais vantagens da POO

A programação orientada a objetos traz uma ideia muito interessante: a representação de cada elemento em termos de um objeto, ou classe. Esse tipo de representação procura aproximar o sistema que está sendo criado ao que é observado no mundo real, e um objeto contém características e ações, assim como vemos na realidade. Esse tipo de representação traz algumas vantagens muito interessantes para os desenvolvedores e também para o usuário da aplicação. Veremos algumas delas a seguir.

A reutilização de código é um dos principais requisitos no desenvolvimento de software atual. Com a complexidade dos sistemas cada vez maior, o tempo de desenvolvimento iria aumentar exponencialmente caso não fosse possível a reutilização. A orientação a objetos permite que haja uma reutilização do código criado, diminuindo o tempo de desenvolvimento, bem como o número de linhas de código. Isso é possível devido ao fato de que as linguagens de programação orientada a objetos trazem representações

muito claras de cada um dos elementos, e esses elementos normalmente não são interdependentes. Essa independência entre as partes do software é o que permite que esse código seja reutilizado em outros sistemas no futuro.

Outra grande vantagem que o desenvolvimento orientado a objetos traz diz respeito a leitura e manutenção de código. Como a representação do sistema se aproxima muito do que vemos na vida real, o entendimento do sistema como um todo e de cada parte individualmente fica muito mais simples. Isso permite que a equipe de desenvolvimento não fique dependente de uma pessoa apenas, como acontecia com frequência em linguagens estruturadas como o C, por exemplo.

A criação de bibliotecas é outro ponto que é muito mais simples com a orientação a objetos. No caso das linguagens estruturadas, como o C, temos que as bibliotecas são coleções de procedimentos (ou funções) que podem ser reutilizadas. No caso da POO, entretanto, as bibliotecas trazem representações de classes, que são muito mais claras para permitirem a reutilização.

Entretanto, nem tudo é perfeição na programação orientada a objetos. A execução de uma aplicação orientada a objetos é mais lenta do que o que vemos na programação estruturada, por exemplo. Isso acontece devido à complexidade do modelo, que traz representações na forma de classes. Essas representações irão fazer com que a execução do programa tenha muitos desvios, diferente da execução sequencial da programação estruturada. Esse é o grande motivo por trás da preferência pela linguagem C em hardware limitado, como sistemas embarcados. Também é o motivo pelo qual a programação para sistemas móveis como o Google Android, embora em Java (linguagem orientada a objetos), seja feita o menos orientada a objetos possível.

No momento atual em que estamos, tecnologicamente essa execução mais lenta não é sentida. Isso significa que, em termos de desenvolvimento de sistemas modernos, a

programação orientada a objetos é a mais recomendada devido as vantagens que foram apresentadas. Essas vantagens são derivadas do modelo de programação, que busca uma representação baseada no que vemos no mundo real.

Exemplos de Linguagens Orientadas a Objetos

Há uma grande quantidade de linguagens de programação orientada a objetos no mercado atualmente. Nesse artigo, iremos apresentar 3 das mais utilizadas no momento: Java, C# e C++. Cada uma delas possui uma abordagem diferente do problema que as torna muito boas para alguns tipos de aplicações e não tão boas para outros.

Java

O Java é, muito provavelmente, a linguagem de programação mais utilizada no mercado atual. Auxiliado pela presença do JRE (*Java Runtime Environment*), ou variações dele, em quase todos os dispositivos eletrônicos do momento, a linguagem Java é um grande sucesso entre os desenvolvedores. O sucesso da linguagem aumentou ainda mais com o Google Android, que escolheu o Java como linguagem preferencial de desenvolvimento de aplicações.

O Java implementa os quatro pilares de forma bastante intuitiva, o que facilita o entendimento por parte do desenvolvedor. A abstração, o primeiro pilar, é implementado através de classes, que contém propriedades e métodos, de forma bastante simples. Já o encapsulamento é realizado através de propriedades privadas, auxiliadas por métodos especiais *getters* e *setters*, como mostra a **Listagem 1**. Vale ressaltar a palavra-chave “this” mostrada no método `setId()`. Essa palavra-chave funciona como um representante da classe atual, uma auto-referência ao próprio

objeto.

Listagem 1. Encapsulamento em Java

```
private int id;

public int GetId()
{
    return id;
}

public void SetId(int id)
{
    this.id = id;
}
```

As questões de herança e polimorfismo no Java são um pouco mais complexas. O Java possui herança simples, o que significa que cada classe pode herdar de apenas uma outra. Entretanto, o Java possui as chamadas Interfaces, que possuem propriedades e assinaturas de métodos. Essas interfaces precisam ser implementadas para funcionar, o que significa que uma classe pode implementar várias interfaces e herdar de apenas uma classe. Na questão de polimorfismo, o atributo `@Override` é responsável por informar ao Java que o método em questão está sendo reescrito.

C#

O C#, por sua vez, é outra das linguagens mais utilizadas no mercado. Como os computadores pessoais no mundo, em sua maioria, possuem o sistema operacional Windows, da Microsoft, o C# se popularizou. Isso porque o Windows implementa o Framework .NET, ao qual o C# está associado. O C# é uma linguagem de uso geral e especialmente criada para utilização com a orientação a objetos. Vale ressaltar que, em C#, tudo é um objeto (herda da classe *object*).

A abstração é muito simples, e segue o modelo do Java. A questão de encapsulamento é um pouco diferente devido a implementação dos métodos *getter* e *setter*. A nomenclatura também é um pouco diferente. A variável que realmente guarda o valor do dado é chamada atributo, enquanto a propriedade é o elemento que realmente acessa aquele dado do mundo externo. Isso está mostrado na **Listagem 2**. Além disso, o C# faz uso de duas palavras-chave especiais: *get* e *set*.

Listagem 2. Encapsulamento em C#

```
// Atributo
private int id;

// Propriedade
public int Id
{
    get;
    set;
}
```

A questão da herança em C# também segue o modelo do Java: herança simples e a possibilidade de utilização de interfaces. A importância das interfaces é muito grande, uma vez que elas podem dar o tipo dos dados, que somente posteriormente serão associados a um tipo real, como mostra a **Listagem 3**. Isso também é válido para o Java. Por padrão, as identidades das interfaces começam com a letra “I”. O polimorfismo, por sua vez, é baseado em métodos virtuais (com a palavra-chave *virtual*) na classe pai e reescritos com a palavra-chave *override* na classe filha.

Listagem 3. Interfaces em C#

```
IExemploInterface exemplo;
exemplo = new ImplementacaoIExemploInterface();
```

C++

O C++, por sua vez, é uma linguagem um pouco mais primitiva, e permite muito mais liberdades com o hardware. Como ele foi derivado imediatamente do C, o C++ permite a utilização de ponteiros, por exemplo, que irão trabalhar diretamente com a memória. Além disso, o C++ pode utilizar todas as bibliotecas C que existem diretamente.

Em termos de abstração, o C++ implementa classes, assim como qualquer linguagem orientada a objetos. Ele também possui o sentido de privado e público, que é utilizado para encapsulamento. Esse encapsulamento é realizado através de métodos *getter* e *setter*, muito similar ao visto em Java, como mostra a **Listagem 4**. Repare que a listagem mostra somente a assinatura dos métodos especiais, sendo que sua implementação é a mesma que em Java. Esse tipo de adaptação é muito comum em C++, onde a classe é guardada em um arquivo `.h` e sua implementação em um arquivo `.cpp`.

Listagem 4. Encapsulamento em C++

```
private:
    int id;
public:
    int GetId() const;
    void SetId(int const id);
```

A questão da herança no C++ é um pouco diferente. A linguagem permite a herança múltipla, o que significa que cada classe pode herdar de quantas classes desejar. Isso pode causar problemas de métodos que possuem o mesmo nome, portanto o desenvolvedor precisa estar atento. O polimorfismo é baseado em métodos virtuais, da mesma forma como o C#. A complexidade, entretanto, é maior, uma vez que temos que cuidar de detalhes de mais baixo nível, como acesso a memória.

Além dessas exemplificadas, existem outras linguagens que merecem ser citadas. Entre elas, podemos elencar: [Python](#), linguagem de script orientada a objetos que é muito utilizada em pesquisas científicas devido a sua velocidade; [Object Pascal](#) (também conhecida como **Delphi**, devido ao nome de sua IDE), que está caindo em desuso, apesar do grande número de sistemas mais antigos que a utilizam; [Objective-C](#), que é a linguagem de preferência para desenvolvimento de aplicações para os sistemas da Apple, como iPhone e iPad; Ruby, voltada para o desenvolvimento web; e Visual Basic [.NET](#), muito utilizada até pouco tempo, mas também caindo em desuso, principalmente devido ao avanço do C# em popularidade.



Ao longo desse artigo, procuramos elencar os elementos que fazem da programação orientada a objetos um sucesso no momento. Vimos os quatro pilares desse paradigma e entendemos como eles são implementados em algumas das linguagens mais utilizadas no mercado de desenvolvimento. Além disso, entendemos algumas das vantagens que tornaram a programação orientada a objetos um grande sucesso para o desenvolvimento de sistemas modernos.



Henrique Machado Gasparotto

Estudante de Engenharia de Computação na Universidade Federal de Santa Maria – UFSM e Técnico em Informática pelo SENAC Santa Maria. Experiência em programação C# .NET e Java, além de aplicações mobile (Android e Windows Phone) e [...]

O que você achou deste post?

 [Gostei \(31\)](#)  (0)

[+ Mais conteúdo sobre .net](#)

Todos os comentarios (15)

[Postar dúvida / Comentário](#)

Meus comentarios



Douglas Ricardo Lucio MVP

Ola Amigo, parabéns pelo post, apenas gostaria de entender o ponto que você mencionou sobre a linguagem Object Pascal ou a IDE Delphi, você citou em seu artigo que a linguagem esta em desuso, eu discordo, pois ela esteve sim com sérios problemas ate algumas versões anteriores depois da ultima versão disponibilizada pela Borland (Delphi 7) que dominava muito o mercado, depois foi vendida para a embarcadero, ela teve sim, alguns problemas de bugs na ferramenta nas versões posteriores, mas a ultima versão lançada, a Xe7, esta muito bom, principalmente com os recursos de compilação de projetos multi plataforma, o qual você consegue através de uma mesma IDE, desenvolver sistemas para windows, android, Mac e Ios. Isso é muito legal, apesar de ser muito bom também. Acredito que ela tenha perdido, neste meio tempo, muitos programadores que migraram de plataforma, por conta de problemas que estavam sendo apresentados nas versões, mas na minha opinião ela continua sim sendo uma excelente plataforma e acredito que com esses recursos de compilação multi plataforma, ela recupere sim o seu espaço dentre as ferramentas mais utilizadas em desenvolvimento de software.

Grande Abraço

[há +1 mês] - Responder



Rodrigo Mendes Lopes MVP

Caro amigo Douglas, só para complementar seu comentário totalmente pertinente, após a versão 7 (ano de 2002) do Delphi a linguagem passou a se chamar "Delphi" e não mais Object Pascal. Então a citação "...Object Pascal (também conhecida como Delphi, devido ao nome de sua IDE) ..." de nosso amigo Henrique não está totalmente correto. Sou desenvolvedor Delphi e também C#, e ambas implementam o paradigma da Orientação a Objetos, dadas as devidas considerações e particularidades de cada uma, e é nisso que esse artigo deve e/ou deveria se manter. De qualquer forma, parableno nosso amigo Henrique pelo artigo simples, porém muito esclarecedor sobre o assunto proposto. E endosso a importância do estudo desse paradigma, pois em minha opinião, as vantagens são enormes.

[há +1 mês] - Responder



Adiel França MVP

Olá Henrique, quero parabenizá-lo pelo artigo, muito útil. Porém, discordo também sobre o Delphi estar em desuso.

Acredito que os investimentos da Embarcadero permitiu que a IDE torna-se uma ferramenta poderosa, e ao contrário do desuso está em muito uso porque as empresas estão conhecendo o recurso de Multi-Device que sem dúvida é o melhor reaproveitamento de código que existe.

Abcs

[há +1 mês] - Responder



Douglas Claudio

Olá pessoal, concordo plenamente com vocês. Eu tive a oportunidade de participar da Conference Embarcadero Delphi no ano passado e o Marco Cantú deixou claro que o Delphi não morreu. No entanto, ele admitiu o problema da ferramenta demorar muito para evoluir, mas consequência disto foram crises internas na diretoria. Com isso a perda de espaço no mercado para as novas tecnologias foi bem notável mas não o suficiente para derrubar o nosso bom e velho Delphi, porém a Embarcadero está revertendo esse quadro com excelentes melhoras na ferramenta.

[há +1 mês] - Responder



Marcelo Zerbinati E Zerbinati Ltda Me MVP

Realmente, o artigo está ótimo, muito bem explicado, parabéns. Só que como os amigos disseram acima, Delphi é uma linguagem de programação (Delphi Language) e dá suporte a POO.

[há +1 mês] - Responder



Danimar Veriato

Amigo, qual a de postagem deste artigo?

[há 14 dias] - Responder



Douglas Claudio

Olá Danimar, tudo bem?

Foi publicado no dia 30/10/2014.

Um abraço!

[há 11 dias] - Responder



[autor] Henrique Machado Gasparotto

Olá Pessoal, agradeço os comentários e os elogios. Peço desculpas na questão da nomeação da linguagem, acabei não explicando muito bem esse ponto. Quanto ao fato de que Delphi está morrendo, é uma questão muito relativa. É claro que a Embarcadero vai falar que não, e as ferramentas realmente melhoraram muito ultimamente. Mas a questão é que não se procuram desenvolvedores Delphi como antigamente, C# e Java estão tomando conta e a tendência é que continue esse padrão, principalmente com o .NET para Linux e Mac OS X. Um abraço e qualquer dúvida estamos a disposição.

[há +1 mês] - Responder



Jose Hugo Baia De Lima

olá Henrique!

Prazer, sou o Hugo Lima, Aluno de Engenharia de Computação, como você! gostaria de saber, qual o conselho que você me dá, em relação ao curso de Engenharia de Computação?

Se tiver um contato, pode me mandar para trocarmos informações sobre o curso.

[há 11 dias] - Responder



Douglas Claudio

Olá Jose, obrigado pelo seu comentário.

Enviamos sua solicitação ao Henrique e estamos no aguardo de um feedback do mesmo.

Um abraço.

[há 10 dias] - Responder



[autor] Henrique Machado Gasparotto

Olá Hugo, tudo bem? Os cursos de Engenharia de Computação são muito diferentes entre si, normalmente. Qual universidade você estuda? No geral, eu gosto muito do curso, e o mercado de trabalho é bastante promissor nessa área no Brasil. Então, meu conselho seria: vai fundo, sempre buscando conhecimento extra, que você pode ir longe. Tenho um contato sim, podemos conversar melhor: é hmgasparotto@hotmail.com. Um abraço e qualquer dúvida estamos a disposição.

[há 8 dias] - Responder



Mário Moreira

Olá Henrique.

Eu gostaria de aprender a programação orientada a objectos, concretamente o Visual Basic.NET, pois já mexe com o Visual Basi.

Gostaria que me indicasses algum material ou algum curso que eu possa fazer para poder migrar para esse paradigma de programação.

Posso entrar em contacto consigo a partir do teu e-mail? Assim poderia fazer-te mais questões.

Gostei da matéria que escreveste, deu para ter uma visão melhor do que é POO.

[há 6 dias] - Responder



[autor] Henrique Machado Gasparotto

Olá Mário, tudo bem? Se puder escolher, aconselho que foque em aprender C#. Há muito mais material e é uma linguagem mais requisitada no mercado, atualmente.

Quanto a cursos, sugiro que foque, inicialmente, em entender o .NET

(<http://www.devmedia.com.br/curso/curso-de-introducao-ao-net-framework/373>). No mais, há alguns bons cursos de programação aqui na DevMedia, como esse: <http://www.devmedia.com.br/curso/programacao-orientada-a-objetos-em-net/302>.

Sinta-se à vontade para me contatar por email ou através dessa página de comentários.

Um abraço e qualquer dúvida estamos a disposição.

[há 5 dias] - Responder



Mário Moreira

Obrigado Henrique!!!

Farei o que me estas a dizer. Irei entrar em contacto consigo por e-mail. Mais uma duvida, o que é o MVP? Pois para ver alguns videos do vosso site é necessário ter uma dessas assinatura dessas.

Quais são as vantagens dessas assinaturas?

Abraço!

[há 5 dias] - Responder



[autor] Henrique Machado Gasparotto

Olá Mário, tudo bem? A assinatura MVP te dá acesso à todas as sessões do site, incluindo cursos, revistas, pocket vídeos, entre outros. Repassei a sua dúvida para a equipe responsável para que eles possam te explicar melhor. Um abraço e qualquer dúvida estamos a disposição.

[há 4 dias] - Responder

Publicidade



Mais posts

Pocket Video

Integrando o TinyMCE editor html no ASP.NET MVC.

Video aula

Adicionando mais produtos no carrinho de compras - Curso de ASP.NET MVC 5 - Quiron Loja Virtual - Aula 51

Video aula

Attribute Routing - Curso de ASP.NET MVC 5 - Quiron Loja Virtual - Aula 50

Pocket Video

Consumindo serviço ASP.NET Web API com AngularJS

Video aula

Conexões Persistentes em threads diferentes II - Curso de SignalR - Aula 24

Video aula

Conexões Persistentes em threads diferentes I - Curso de SignalR - Aula 23

Video aula

Hubs – Exemplo prático parte 2 - Curso de SignalR - Aula 22

Video aula

Hubs – Exemplo prático parte 1 - Curso de SignalR - Aula 21

Video aula

Hubs - Cliente: implementando sem utilizar proxy dinâmico - Curso de SignalR - Aula 20

Listar mais conteúdo



Anuncie | Loja | Publique | Assine | Fale conosco



DevMedia

Curtir Página

67 mil curtidas

Seja o primeiro de seus amigos a curtir isso.



Hospedagem web por Porta 80 Web Hosting