Programação Orientada a Objetos

Matheus Vieira Gonçalves[1]

[1]Bacharelado em Engenharia de Computação – Centro Universitário de Anápolis (UniEVANGÉLICA) – Anápolis – GO

[1]matheusvieira1900@hotmail.com

Resumo. *A fim de acompanhar as constantes evoluções tecnológicas, os softwares aumentam a sua complexidade e consequentemente o tamanho do código necessário para suprir tal evolução. Portanto, com o objetivo de melhorar o desenvolvimento e reduzir o número de falhas geradas pelo aumento de código, surgiu a Programação Orientada a Objetos (POO). A POO, refere-se a um modelo de desenvolvimento de software que visa propor uma nova forma de organização do código, separando-o em várias partes a fim de facilitar a sua manutenção e reutilização, reduzindo dessa forma o número de falhas e erros.*

Palavras-chave: *POO; Organização de Código; Programação.*

1. Introdução

Atualmente no mercado de trabalho, o desenvolvimento de software apresenta uma vasta variedade de princípios e conceitos que são aplicados, e dentre os quais a programação orientada a objeto ganhou destaque, tornando-se a principal forma de desenvolvimento aplicada (GASPAROTTO, 2014). Linguagens como Java, C#, Python e C++ são orientadas a objeto e mesmo possuindo formas diferente de implementação, todas seguem os mesmo princípios e conceitos (EDUARDO, 2005).

O presente documento tem por objetivo explicar de forma simples e clara o paradigma da orientação a objeto, através de uma apresentação objetiva envolvendo os principais conceitos e pilares que constituem este modelo, buscando desta forma solucionar dúvidas existentes na implementação dessa prática de desenvolvimento, a qual se tornou uma das principais na construção de softwares de qualidade, graças aos benefícios que ela proporciona ao possibilitar uma melhor organização e reutilização de código, e consequentemente gerando uma redução no número de problemas e gastos.

2. Desenvolvimento

2.1. Contextualização

O termo “orientação a objeto” é mais antigo do que muitas pessoas imaginam. Ele foi criado por Alan key, um dos criadores da linguagem de programação Smalltalk em 1969 e se tornou pública apenas em 1980, quando sua versão estável foi finalizada. Esta foi uma das primeiras linguagens a incorporar o paradigma da orientação a objeto, onde todos os dados como: números, métodos e blocos de código, eram considerados objetos. Entretanto, após algumas pesquisas, foi relatado que este paradigma já havia aparecido anteriormente, incorporado em 1967 na linguagem de programação “Simula 67”, desenvolvida em 1962 por Ole-Johan Dahl e Kristen Nygaard (MONQUEIRO, 2007).

Dessa forma, o modelo de programação orientada a objeto pode ser considerado como antigo e, entretanto sua aplicação no mercado só tomou maiores proporções recentemente, com a expansão de linguagens no mercado de trabalho, como: Java, Delphi, C++, C#, Ruby, Python, entre outras (MONQUEIRO, 2007).

2.2. Ideia Básica da POO

O principal objetivo da criação da Programação Orientada a Objeto (POO), baseou-se numa aproximação do mundo virtual com o mundo real (MONQUEIRO, 2007). Como meio para se alcançar tal objetivo, longas linhas de códigos foram separadas e transformadas em vários objetos, possibilitando que o desenvolvedor possa pensar em um objeto por vez e posteriormente conectá-lo aos demais, ao invés de pensar em uma única lógica que envolvesse todo o código, a qual era utilizada anteriormente na programação estruturada.

Dessa forma, torna-se mais fácil desenvolver o software, trabalhando separadamente em cada objeto, possibilitando a reutilização de código, consequentemente reduzindo o número de erros e facilitando a sua manutenção.

2.3. Os 4 Pilares da Programação Orientada a Objetos

Para aprofundar os conhecimentos sobre orientação a objeto, primeiramente, torna-se necessário conhecer os quatro pilares fundamentais para o funcionamento desse modelo, os quais são eles: Abstração, Encapsulamento, Herança e Polimorfismo. Cada pilar apresenta suas características, capazes de revelar sua total eficiência quando trabalhadas em conjunto com os demais pilares.

2.3.1. Abstração

A abstração é o principal conceito abordado na orientação a objeto, sendo indispensável para o seu funcionamento (MONQUEIRO, 2007). De forma simples, este conceito revela a capacidade de recriar os objetos existentes no mundo real no mundo digital e utilizá-los em um programa, contudo, levando em consideração somente as características do objeto que são relevantes para o funcionamento do código (PEREIRA, 2010). Para alcançar seu objetivo, três pontos devem ser levados em consideração dentro da abstração, sendo eles: a identidade, as propriedades e os métodos.

O primeiro ponto é a identidade e que consiste em atribuir um nome ao objeto de criação. No entanto, o nome deve ser único e sem ambiguidade evitando dessa forma conflitos dentro do sistema, por exemplo atribuir o nome “Carro” a um objeto ao invés de "Automóvel", já que a segunda opção pode englobar carros, motocicletas, ônibus, entre outros. O segundo ponto são as propriedades, definidas basicamente como as características que o objeto apresentaria, e da mesma forma que os objetos do mundo real, tais como: modelo, tamanho, cor, entre outras. Por fim, o terceiro ponto denominado de métodos, o qual vai definir as ações que são executadas pelos objetos, como por exemplo, “acelerar” e “frear”.

2.3.2. Encapsulamento

O encapsulamento é o segundo pilar aplicado na POO, sendo o responsável pela segurança da aplicação. Ele possibilita ocultar ideias, propriedades e métodos, gerando uma maior eficiência na realização de manutenções, reduzindo o risco de comprometer toda a aplicação.

Grande parte das linguagens que aplicam o encapsulamento utilizam propriedades privadas, que são ligadas a métodos especiais denominados de getters e setters, responsáveis por retornar e setar o valor da propriedade, respetivamente. Graças a isso, o acesso direto a propriedade do objeto é bloqueada, criando dessa forma uma nova camada de segurança (GASPAROTTO, 2014).

Um exemplo de encapsulamento é dado por Thiago Vinícius em seu artigo sobre os quatro pilares da orientação a objeto:

> Como um exemplo mais técnico podemos descrever o que acontece em um sistema de vendas, onde temos cadastros de funcionários, usuários, gerentes, clientes, produtos entre

> outros. Se por acaso acontecer um problema na parte do usuário é somente nesse setor que será realizada a manutenção não afetando os demais (PALMEIRA, 2012).

Portanto, a utilização do encapsulamento possibilita diversas vantagens durante o desenvolvimento do software, facilitando não apenas a leitura e manutenção do código, como também gerando uma maior segurança para a aplicação, tornando-se dessa forma essencial para a orientação a objeto.

2.3.3. Herança

Herança é o terceiro pilar da orientação a objeto, sendo responsável por possibilitar que os atributos e métodos pertencentes a uma classe, sejam atribuídos a outra. Dessa forma o trabalho é otimizado, reduzindo a quantidade de código que seria duplicado e, consequentemente, o tempo para a sua produção.

A herança é definida por autores como Júlio César Monqueiro como a capacidade que uma classe possui de:

> […] estender todas as características de outra e adicionar algumas coisas a mais. Desta forma, a classe SerHumano será uma especialização (ou subclasse) da classe Animal. A classe Animal seria a classe pai da SerHumano, e logicamente, a classe SerHumano seria a classe filha da Animal (MONQUEIRO, 2007).

Apesar de o conceito ser o mesmo para todas as linguagens que aplicam a herança em seus códigos, a forma de sua aplicação pode sofrer as variações. Um exemplo prático seria na linguagem C++, a qual utiliza heranças múltiplas, possibilitando que a "classe filha" possam herdar características de várias "classes pais". Já linguagens como C#, assim como afirma Gasparotto (2014), "trazem um objeto base para todos os demais", ou seja, é criado apenas uma "classe pai" que fornecerá características para todas as demais "classes filhas".

2.3.4. Polimorfismo

Polimorfismo é o quarto e último pilar da orientação a objeto, entretanto apresenta tanta importância quanto os demais. É através da sua utilização que objetos criados através da herança entre as classes pai e filha, tornam-se capazes de modificar suas propriedades e métodos, adaptando-se conforme o necessário.

Um exemplo mais prático é abordado por autores como Gasparotto, como:

> [..] 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 ()' (GASPAROTTO, 2014).

É válido ressaltar que, assim como na herança, o polimorfismo apresenta o mesmo conceito para todas as linguagens que o utilizam, entretanto, a sua forma de implementação sofrer variações de linguagem para linguagem. Portanto, independente da linguagem utilizada, é por meio do polimorfismo que os objetos que utilizam a herança, tornam-se aptos a modificar seus atributos e métodos conforme o necessário.

2.4. Principais Conceitos

Após analisar todos os principais conceitos aplicados no quadro pilares de orientação a objeto, tornasse possível estudar os demais conceitos essenciais, presentes para o funcionamento deste modelo de desenvolvimento de software, sendo eles: Classes, objetos, interfaces e classes abstratas.

2.4.1. Classes

Uma classe é uma abstração que pode ser descrita como um modelo com todas as características que os objetos criados através dessa classe devem ter, ou seja, todos os atributos e métodos que ele deverá incorporar.

De forma análoga, podemos comparar uma classe com uma receita de bolo. A receita de bolo apresenta todos os atributos que um bolo deverá ter, entretanto ela é apenas um modelo a ser seguido. A classe é justamente este modelo, ela servirá como base para a criação de objetos, dessa forma, cada objeto distinto possui sua modelo de criação ou classe. Portanto, de forma mais simples podemos definir classes como um modelo para a criação de um ou mais objetos.

2.4.2. Objetos

Objetos são definidos como instâncias de classes, as quais determinam quais informações o objeto deverá conter, ou seja, eles podem ser descritos como um exemplar da classe. Cada objeto possui todos os atributos que são passados na classe, entretanto cada um apresenta atributos diferentes do objeto anterior.

Uma forma mais simples de entender os objetos seria através da analogia feita anteriormente com a classe e a receita de bolo. No entanto, dessa vez a comparação será entre o objeto e o bolo. A classe como citada anteriormente seria a receita do bolo e o objeto seria o bolo propriamente dito. Cada bolo possui todos os ingredientes (atributos) e forma de preparação (método) presentes em sua classe, entretanto, por mais que um bolo seja parecido com o anterior é impossível que todas as suas características sejam iguais, isto porque para a sua criação deverá ser utilizado novos ingredientes e o seu modo de preparação pode sofrer pequenas variações. Portanto, podemos definir objetos como algo sólido construído através de uma classe.

2.4.3. Interface

Interface é definida por alguns autores como Camargo (2010) como um “contrato entre a classe e o mundo exterior” (p. ?). Em outras palavras ela pode ser definida como um conjunto de métodos e atributos que ao implementada por outra classe, todos os atributos e métodos presentes na interface obrigatoriamente deverão ser utilizados na classe que a implementou.

Dessa forma, as interfaces são basicamente um auxiliador na criação de classes que deverão apresentar atributos e métodos iguais, evitando assim que exista variações entre as classes, tais como: atributos e métodos declarados de forma diferente e que posteriormente possam ocasionar conflitos. É válido ressaltar que uma interface não pode ser instanciada, ou seja, ela não deve ser utilizada na criação de objetos.

2.4.4. Classes Abstratas

As classes abstratas são definidas por Camargo (2010) como “classes feitas especialmente para serem modelos para suas classes derivadas”. Graças a isso, como medida de segurança a criação de instâncias através das classes abstratas é evitada.

Outro fator de relevância é que as classes derivadas da classe abstrata, necessariamente deverão sobrescrever os métodos para realizar a implementação dos mesmos, isso pois na classe abstrata todos os métodos são declarados de forma genérica, desta forma ao implementá-lo em uma classe em específico torna-se necessário a sua modificação de genérico para específico, possibilitando a sua utilização.

3. Considerações Finais

Por intermédio da pesquisa realizada anteriormente, torna-se visível que a programação orientada a objeto se tornou um dos principais modelos adotados no mercado de trabalho, inclusive dentro da Fábrica de Tecnologias Turing, graças a sua eficiência na melhora do desenvolvimento de software, proporcionando não apenas um código melhor estruturado como reduzindo também o número de erros e falhas.

Diversas linguagens que apresentam grande influência no mercado, tais como Java (atualmente utilizada dentro da Fábrica) e C# já utilizam de tal modelo, tornando-se necessário o seu aprendizado a fim de acompanhar as evoluções que ocorrem no mundo da tecnologia. Portanto, após os estudos sobre os principais conceitos abordados na programação orientada a objeto presente neste trabalho, recomenda-se o estudo mais aprofundado sobre uma linguagem em específico, a qual faz uso da orientação a objeto em sua aplicação, complementando desta forma os conhecimentos adquiridos no presente documento.

Referências

CAMARGO, W. B. de. *Interfaces: Programação Orientada a Objetos*. 2010. Disponível em: <https://www.devmedia.com.br/interfaces-programacao-orientada-a-objetos/18695>. Acesso em: 16 de nov. de 2018.

GASPAROTTO, H. M. *Os 4 pilares da Programação Orientada a Objetos*. 2014. Disponível em: <https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264>. Acesso em: 15 de nov. de 2018.

MONQUEIRO, J. C. B. *Programação Orientada a Objetos: uma introdução*. 2007. Disponível em:<https://www.hardware.com.br/artigos/programacao-orientada-objetos/>. Acesso em: 16 de nov. de 2018.

PALMEIRA, T. V. V. *Abstração, Encapsulamento e Herança: Pilares da POO em Java*. 2012. Disponível em: <https://www.devmedia.com.br/abstracao-encapsulamento-e-heranca-pilares-da-poo-em-java/26366>. Acesso em: 15 de nov. de 2018.

PALMEIRA, T. V. V. *Polimorfismo, Classes abstratas e Interfaces: Fundamentos da POO em java*. 2012. Disponível em: <https://www.devmedia.com.br/polimorfismo-classes-abstratas-e-interfaces-fundamentos-da-poo-em-java/26387>. Acesso em: 15 de nov. de 2018.

PAULI, G. *Introdução à POO - Partes 1 e 2*. 2010. Disponível em: <https://www.devmedia.com.br/introducao-a-poo-partes-1-e-2/17282>. Acesso em: 20 de nov. de 2018.

SANTANA, E. F. Z. *Principais conceitos da Programação Orientada a Objetos*. 2015. Disponível em: <https://www.devmedia.com.br/principais-conceitos-da-programacao-orientada-a-objetos/32285>. Acesso em: 20 de nov. de 2018.