

Paradigmas de Programação

Mateus Barbosa, Matheus de Oliveira Rocha

Ciências da Computação – Universidade do Vale do Itajaí (Univali)
Itajaí, SC – Brasil

{mateus.babosa, matheus.rocha}@edu.univali.br

Abstract. *This document makes a simple overview of the concepts about Programming Paradigms, that defines pattern for coding structure, mainly the Imperative, Declarative, and theirs derivatives, like OOP (Object Oriented Programming), Functional, Structured, and so on. The core concepts of the first paradigms (Declarative and Imperative), are basically “what the program should do” and “how the program should do it”, respectively. The Declarative may be more simple to read, but comes with challenges in implementation and it’s not so suitable for big projects, while the Imperative offers a more reliable way to manage big projects, so most languages will adopt this concept.*

Resumo. *Esse documento é uma visão geral dos conceitos de Paradigmas de Programação, que definem padrões de estrutura de códigos, principalmente o Imperativo, Declarativo e seus derivados, como POO (Programação Orientada à Objeto), Funcional, Estruturado, entre outros. O conceito base dos primeiros paradigmas (Declarativo e Imperativo), são basicamente “o que o programa deve fazer” e “como o programa deve fazer”, respectivamente. O Declarativo pode ser mais simples de ler, mas possui desafios na implementação e não é recomendado para projetos grandes, enquanto o Imperativo é mais adaptado para projetos grandes, com isso muitas linguagens utilizam esse conceito.*

1. O que são os Paradigmas de Programação?

Primeiramente, o termo "paradigma" é usado em diversas áreas científicas, como por exemplo na filosofia usa-se para descrever um conjunto de crenças, valores e métodos compartilhados por uma comunidade de pensadores ou cientistas. É uma espécie de "modelo mental" que orienta a maneira como essas pessoas pensam sobre o mundo, formulam perguntas, coletam dados e interpretam resultados. E assim que a computação criou modelos para que programadores consigam ler códigos e facilitar a manutenção sem ter que aprender o código novamente, ou se forçar a comentar cada linha.

“Qualquer um pode escrever um código que o computador entenda. Bons programadores escrevem códigos que os humanos entendam.” – Martin Fowler

Os paradigmas de programação são um meio de padronizar como se é estruturado um projeto, a fim de facilitar a manutenção, escalabilidade e leitura do código. Ou seja, informam a melhor maneira de solucionar problemas usando uma linguagem em específico. Com isso, a linguagem se torna mais uma ferramenta, que possui utilidade em um certo cenário, enquanto o paradigma é uma forma eficaz de usar aquela ferramenta.

Por mais que seja um conceito de “programação”, os paradigmas datam desde os primórdios dos computadores, na década de 1950 à 1960, mas ainda não possuía esse conceito, eram apenas “meios” e formas de abordar a programação, algo que era muito complexo naquela época, já que as máquinas possuíam pouca capacidade de processamento e tinha uma complexidade mais alta, tendo em mente que os computadores levavam dias para retornar algo e não havia as diversas abstrações das linguagens de hoje.

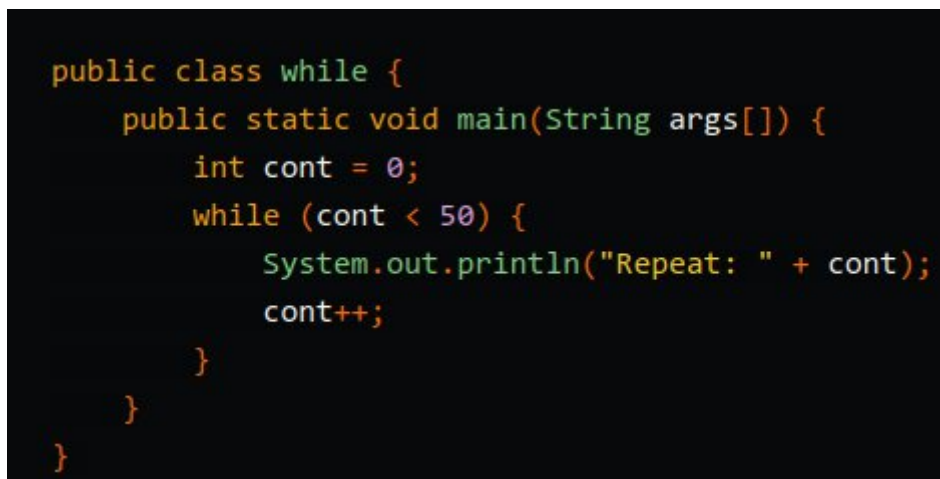
Assim foi implementado os paradigmas: Imperativo e Declarativo. Eles se tornaram mais complexos e definiram muitos outros métodos de programação, como o OOP (Object Oriented Programming), Funcional, entre outros.

1.1. Paradigma Imperativo

Usada na linguagem de programação C. O desenvolvedor informa o computador exatamente como devem ser executados os processos, isso realizado de forma sequencial e dizendo o passo-a-passo das instruções. De certa forma, o programador se preocupa mais em “como” a resolução do problema deve ser feita, ou seja, não se importando em criar um código muito abstrato.

Algumas das vantagens seria que é o modelo mais aproximado do que se vê no mundo real, muitas empresas utilizam essa metodologia para projetos grandes. E na maioria dos casos é mais eficiente que o paradigma Declarativo, fazendo com que a maioria das linguagens adotem esse paradigma como seu padrão.

Ainda assim, ela possui desvantagens, como é usando em projetos grandes com diversos programadores, sua legibilidade e manutenibilidade é afetada, gerando processos confusos e programas mais sujeitos a *bugs*.

A imagem mostra um trecho de código Java em um editor de texto com fundo escuro. O código define uma classe chamada 'while' com um método estático 'main' que recebe um array de strings. Dentro do método 'main', há uma variável 'cont' inicializada com 0, seguida por um loop 'while' que continua enquanto 'cont' for menor que 50. No corpo do loop, é impresso no console 'Repeat: ' seguido do valor de 'cont', e depois 'cont' é incrementado em 1. O código é encerrado com fechamento de chaves para o método, a classe e o pacote 'public'.

```
public class while {  
    public static void main(String args[]) {  
        int cont = 0;  
        while (cont < 50) {  
            System.out.println("Repeat: " + cont);  
            cont++;  
        }  
    }  
}
```

Figura 1. Exemplo de código feito em Java com Paradigma Imperativo. No código é gerado um output no Console (Terminal) que contará do 0 ao 49.

1.2. Paradigma Declarativo

A Declarativa seria o oposto da Imperativa, nesse caso ao invés de pensar em “como” deve ser feito a solução do problema, deve-se pensar em “o que” vai ser resolvido. Ou seja, você terá uma abstração muito maior, se preocupando em descrever a sequência lógica e o resultado esperado, e não como esse resultado será computado pela máquina. As declarações iniciais da lógica são imutáveis, reforçando o fato de que ela sempre

retornará o mesmo valor, em contrapartida o Imperativo pode retornar valores diferentes para a mesma computação. As linguagens que usam esse modelo são: HTML, CSS e SQL.

Vale ressaltar que o código dos programas Declarativos são muito menores que os do Imperativo, já que a lógica terá em mente a abstração dos processos. Outra analogia é que *Estrutura de Dados*, é o método Declarativo para gerar lógica com a finalidade de aceitar “quaisquer” (apenas para ênfase de sua capacidade), mas retornando o mesmo valor.

Com isso podemos ver suas vantagens de legibilidade, redução de *bugs*, redução de efeitos colaterais (em que o programa altera algo que não deva). Mas com isso vem os desafios de adaptar o código, e extrema complexidade de implementação.

```
let languages = ["python", "java", "go", "ruby"];
// Imperative

for(let i = 0; i < languages.length; i++){
  languages[i] = "I love " + languages[i];
}
// Return: ["I love python", "I love java", "I love go", "I love ruby"]

// Declarative

languages.map(language => "I love" + language);

// Return: ["I love python", "I love java", "I love go", "I love ruby"]
```

Figura 2. Exemplo de código feito em JavaScript comparando os Paradigmas Imperativo e Declarativo, respectivamente. No código, o Array da variável “languages” terá cada um de seus elementos iterados para adicionar a frase “I love ” antes do seu valor. Como pode ser visualizado, os 2 resultados computam o mesmo item, porém com o Declarativo é mais “direto”, você não se preocupa em fazer o Loop.

2. Principais Paradigmas de Programação da Atualidade

Agora que já se tem uma ideia melhor sobre os Paradigmas *parent*, vamos falar um pouco sobre os *child* que foram criados a partir deles.

Existem muitos outros Paradigmas, citando apenas alguns:

- OOP (Programação Orientada à Objeto)
- Orientado a Eventos
- Lógico
- Funcional
- Matemático
- Distribuído
- Reativo
- Orientado a Aspectos
- Procedural

A lista continua, mas iremos focar em apenas 3, sendo eles: OOP, Funcional e Lógico.

Como ponto de observação, a maioria dos paradigmas informados são teoremas antigos, porém como informado anteriormente, cada um possui sua utilidade, mesmo que apenas em cenários muito específicos.

2.1. OOP

OOP (Object-Oriented Programming ou Programação Orientada a Objetos) é um paradigma de programação derivado da Imperativa que se baseia no conceito de objetos, foi criada em meados da década de 70, criada com a ajuda de Alan Kay, um dos criadores da linguagem Smalltalk que é considerada a primeira linguagem a usar o paradigma em questão. Atualmente é usada pela maioria das linguagens, como por exemplo Java, C++, C#, Ruby, Python, PHP, e muitas outras.

Algumas vantagens iniciais do OOP são: o projeto é enorme e necessita uma grande quantidade de compartilhamento de código, possui uma equipe grande assim os desenvolvedores não tem que saber todo o código, fácil escalabilidade.

Mas nem tudo é 100%, caso o indivíduo possua dúvidas sobre os conceitos básicos de OOP, vários problemas podem surgir, tornando o paradigma contra si mesmo. Como, não utilização correta do conceito DRY (Don't Repeat Yourself), uso errôneo de herança, pouco uso de Classes e Interfaces abstratas, objetos que não condizem com modelo do projeto.

Outro ponto que influenciou a utilização do OOP foi a popularização da Linguagem Java, uma linguagem que começou a ser criada em 1991 pela Sun Microsystems com a ideia principal de rodar em aparelhos domésticos, além dos próprios computadores. E com o avanço da internet, foi-se adaptado para se tornar o que conhecemos da linguagem hoje em dia, com seu lançamento oficial em 1995.

Uma das principais características do Java é sua capacidade de rodar em uma máquina virtual, tornando o código independente da plataforma em que é executado. O Java também possui um sistema de coleta de lixo automático, que gerencia a alocação e liberação de memória.

A linguagem Java é usada em uma variedade de aplicações, desde sistemas corporativos até dispositivos móveis e jogos. A sua popularidade é atribuída à sua portabilidade, segurança e vasta biblioteca de classes e APIs disponíveis. Alguns exemplos de aplicações Java incluem o Android OS, o Minecraft e o Eclipse IDE.

Em OOP, um objeto é uma instância de uma classe, que é uma estrutura que define um conjunto de propriedades e métodos que o objeto pode ter, ou seja, com uma classe você pode ter N número de objetos contendo as mesmas propriedades daquela classe, mas com valores diferentes.

```

public class Carro {
    // Atributos da classe
    private String marca;
    private String modelo;
    private int ano;
    private int velocidade;

    // Construtor da classe
    public Carro(String marca, String modelo, int ano) {
        this.marca = marca;
        this.modelo = modelo;
        this.ano = ano;
        this.velocidade = 0;
    }

    // Métodos da classe
    public void acelerar(int valor) {
        this.velocidade += valor;
    }

    public void frear(int valor) {
        this.velocidade -= valor;
    }

    public void exibirInformacoes() {
        System.out.println("Marca: " + this.marca);
        System.out.println("Modelo: " + this.modelo);
        System.out.println("Ano: " + this.ano);
        System.out.println("Velocidade: " + this.velocidade + " km/h");
    }
}

```

Figura 3. Exemplo de uma Classe feito em Java

O objetivo da programação orientada a objetos é modelar o mundo real em termos de objetos, permitindo que o software seja mais modular, reutilizável e fácil de manter. Isso é alcançado por meio de quatro conceitos fundamentais da OOP: encapsulamento, herança, polimorfismo e abstração.

Encapsulamento refere-se à ideia de que as propriedades e métodos de um objeto devem ser encapsulados, ou seja, protegidos de acesso externo e acessíveis apenas por meio de métodos públicos definidos na classe.

Herança é o conceito em que uma classe pode herdar propriedades e métodos de outra classe, permitindo que as classes derivadas compartilhem comportamentos comuns.

Polimorfismo refere-se à ideia de que as classes derivadas podem ser tratadas como se fossem da classe base, permitindo que os objetos sejam mais facilmente intercambiáveis e reutilizáveis.

Abstração é o conceito em que as características essenciais de um objeto são destacadas e simplificadas, permitindo que a complexidade seja reduzida e que as implementações sejam reutilizáveis.

Em resumo, a OOP é uma abordagem poderosa para a programação que se concentra em modelar objetos do mundo real em um software. Ela fornece uma estrutura para criar programas modulares e escaláveis, permitindo que o código seja reutilizado, mantido e modificado com mais facilidade.

2.2. Funcional

É baseado do Paradigma Declarativo e usa funções matemáticas para realizar os processos. Nos programas utilizando esse modelo, é criado funções curtas e que nunca irão modificar os valores fora do escopo da própria função. Ou seja, a função receberá

o(s) dado(s) de entrada e retornará o valor desejado. Reiterando o que se diz no Paradigma Declarativo, em que se informa “o que” se deseja fazer, e não “como” irá fazer.

As linguagens mais comuns para esse estilo de programação são: Haskell, Lisp, ML, Scala, entre outras. Ou seja, esse estilo de programar está associado a códigos que precisam de matemática envolvida diretamente.

```
1 -- Definindo uma função para verificar se um número é par
2 ehPar :: Int -> Bool
3 ehPar n = n `mod` 2 == 0
4
5 -- Definindo uma função para filtrar números pares de uma lista de inteiros
6 filtrarPares :: [Int] -> [Int]
7 filtrarPares lista = filter ehPar lista
8
9 -- Utilizando a função 'filtrarPares' em uma lista de números
10 main :: IO ()
11 main = do
12     let listaNumeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
13     let listaPares = filtrarPares listaNumeros
14     putStrLn ("Os números pares de " ++ show listaNumeros ++ " são: " ++ show listaPares)
```

Figura 4. Exemplo de código usando Haskell e o Paradigma Funcional

Como se pode ver na figura 4, o problema é separado em pequenas funções que realizam o cálculo matemático, sem alterar quaisquer valores fora do seu escopo. Isso torna o código muito mais legível, já que a lógica não “sairá” daquela função, porém os desenvolvedores têm que ter isso em mente quando utilizam esse método.

2.3. Lógico

Um paradigma mais diferenciado que os demais, mesmo que tenha sido baseado no Declarativo, ele não é baseado por instruções e sim em fatos, usando todos esses fatos para definir um cenário em que sejam verdadeiros levando-o a um final. Assim podemos perceber que esse paradigma é mais focado em projetos de Inteligência Artificial, mas pode ser usado na criação de programas especialistas, que visam simular um profissional em uma área de conhecimento, ou comprovação de teoremas.

```
% Definindo os fatos e regras
mulher(maria).
mulher(julia).
homem(jose).
homem(joao).
pai(jose, maria).
pai(jose, julia).
mae(maria, joao).
mae(julia, joao).
irma(X, Y) :- mulher(X), pai(Z, X), pai(Z, Y), X \= Y.

% Consultando as informações
?- irma(maria, julia).
true.

?- pai(jose, X), mae(Y, X).
X = maria,
Y = joao ;
X = julia,
Y = joao.
```

Figura 4. Exemplo de código usando Prolog e o Paradigma Lógico

As linguagens que utilizam esse Paradigma são: Prolog (Programming in Logic), Alice, Ciao, Absys, Mercury, etc. Esse modelo é mais especializado, e muito difícil de ser usado amplamente em vários projetos de diversos contextos.

4. Conclusão

Como podemos ver, existem diversos modelos de como programar, o desenvolvedor tem que escolher qual paradigma vai resolver o problema com maior eficiência, legibilidade e facilidade de manutenção, claro que isso sempre levando em conta o problema em questão e a linguagem de programação que será usada.

O OOP é a opção mais comum e utilizada para o mercado de trabalho, mas não devemos esquecer das outras alternativas, como o Paradigma Lógico e Funcional, que em algum momento um projeto poderá usa-lós. Assim como o Paradigma é um manual que instrui como eficientemente utilizar uma determinada linguagem, desde que a linguagem permita isso.

References

- Yuri, P. (2023) “História da Programação”, <https://www.infoescola.com/informatica/historia-da-programacao/>, Março.
- Isaac, F. de S. (2021) “Paradigmas de Programação”, <https://guia.dev/pt/pillars/languages-and-tools/programming-paradigms.html>, Março.
- Eduardo, S. (2022) “Quais são os paradigmas de programação mais importantes?”, https://blog.geekhunter.com.br/quais-sao-os-paradigmas-de-programacao/#Paradigmas_de_programacao_Declarativos, Março.
- Administrador (2021) “As vantagens de estabelecer processos bem definidos no desenvolvimento de softwares”, <https://blog.cronapp.io/as-vantagens-de-estabelecer-processos-bem-definidos-no-desenvolvimento-de-softwares/>, Março.
- Cairo, N. (2020) “Paradigmas de programação: o que são e quais os principais?”, <https://blog.betrybe.com/tecnologia/paradigmas-de-programacao/>, Março.
- Aryclenio, B. (2020) “Programação imperativa e declarativa”, <https://dev.to/aryclenio/programacao-imperativa-e-declarativa-1ooi>, Março.
- Ivan, L. M. R. (2003) “Estruturas de dados”, <https://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node10.html>, Março.
- Yuri, P. (2023) “História do Java”, <https://www.infoescola.com/informatica/historia-do-java/>, Março.