Uma introdução a paradigmas de programação

Vitor Della Torre dos Santos
18206049

Fevereiro, 2021

1 Introdução

Para começar, vale a pena descrever o que é um paradigma. De acordo com o dicionário Michaelis [mic], um paradigma é, num sentido amplo, "algo que serve de exemplo ou modelo; padrão."

Pode-se também entender um paradigma como uma forma de pensar ou de elaborar um raciocínio acerca de algo. Estendendo o significado à programação, um *paradigma de programação* nada mais é do que um meio de estruturar palavras de forma que estas estejam organizadas coerentemente de acordo com os mecanismos de verificação e avaliação de uma dada linguagem, sejam esses um compilador e/ou um interpretador.

Nesse sentido, ao empreender tempo criando um raciocínio, este deve atuar em conjunto com as propriedades de uma dada linguagem. Um bom exemplo seria querer realizar o controle manual da memória de um programa em tempo de execução. Linguagens que disponibilizam meios de garantir essa tarefa possuem, em geral, um nível de abstração menor, o que pode vir a exigir uma forma diferente de enxergar um problema por parte do desenvolvedor; dependendo do que o dado verificador e/ou avaliador da linguagem em questão espera, o que é erro programático em um ambiente pode não ser em outro.

Toma-se como exemplo a construção de um arranjo em Python e em C. Nesta última, estruturas de memória dinâmica devem ser tanto alocadas quanto desalocadas manualmente por quem programá-las; em Python, contudo, o *garbage collector* toma conta do trabalho de desalocação da memória, sacrificando controle em prol de segurança.

Dito isso, podemos nos encaminhar ao aprendizado dos diversos paradigmas de programação existentes.

2 Quais são os paradigmas?

De acordo com o web blog *Towards Data Science* [tds], paradigmas de programação se estruturam debaixo de dois grandes grupos, os quais são:

1. Programação Imperativa; e
2. Programação Declarativa.

2.1 Programação Imperativa

Uma das formas mais populares e socialmente difundidas de programação, a Programação Imperativa pode ser observada de relance em filmes e séries que tratam de explorar brechas de segurança digital e *hacking*; consequentemente, é, portanto, a forma que, inicialmente, estudantes pensam ser um estilo único e imortal de se elaborar código programático.

A programação imperativa reina sobre os comandos de um ator a outro; trata-se o primeiro ator como desenvolvedor, o qual envia comandos a uma dada máquina, e, esta, sendo o outro ator em questão, gera saídas de acordo com as entradas que lhe foram estipuladas. Ou seja, o desenvolvedor *impera* sobre o resultado da construção de sua resolução: é ele quem nomeia cada coisa e, portanto, comanda o que cada objeto poderá realizar dentro do ambiente programático.

No entanto, temos subgrupos dentro desse ramo, que são especificações com *flavors* adicionais, vide a Programação Orientada à Objetos e a Programação Procedural [wik].

2.1.1 Programação Orientada a Objetos (P.O.O)

Agrupa instruções e comandos executáveis dentro de **classes**, criando a ilusão, ao desenvolvedor, de ser capaz de criar propriedades para tipos existentes numa dada linguagem, ou até a criar tipos novos. A definição dada é propositalmente vaga, pois, para que entendamos P.O.O., devemos compreender o que o conceito de "objeto" providencia à nossa máquina.

Por exemplo, Python trata todos os seus comandos e tipos como objetos: funções são objetos; variáveis locais, mesmo as com tipos primitivos (inteiros, chars etc), são objetos; valores literais globais, constantes ou não, também, são objetos. O intuito de que tudo seja objeto em Python é garantir que tudo tenha valor hierárquico igual: tudo tem propriedade, tudo pode ser descrito; se algo é colocado dentro de uma outra coisa, pode gerar um outro objeto completamente novo.

No entanto, diferentemente de Python, em Java a criação de objetos não é dada como implícita, pois nem tudo é considerado objeto por padrão; a denotação do que é dito como classe e o que é, então, tratado como objeto torna a linguagem criteriosa acerca da criação destes e, por consequência, soa desconfortável para muitos durante primeiras fases de aprendizado. Usar um valor literal como $letter = a$ em Python nos permite checar e usar funções deste, como $letter.\_\_doc\_\_$, enquanto Java trataria isso como tipo sem propriedade (primitivo) e exigiria um esforço maior do desenvolvedor no caso deste querer um carácter com outras propriedades.

A Orientação a Objetos dá poder ao desenvolvedor para dizer o que é e o que não é pertencente a cada tipo, já existente (*built in*) ou não.

Mais exemplos de linguagens com suporte ao paradigma de orientação à objetos: SmallTalk, C++, Kotlin e Swift.

2.1.2 Programação Procedural

Agrupa instruções e comandos executáveis dentro de **procedimentos**, ou rotinas. Apesar da Orientação a Objetos também envolver criação de funções para execução, faz-se a distinção entre ambos os paradigmas pelas formas como a programação procedural conversa com o hardware das máquinas em que opera graças ao casamento que pode fazer com operações relacionadas ao Registrador de Pilha (*Stack Pointer*). [wik]

Acredito que seja seguro assumir também que faz-se a distinção deste paradigma do Orientado a Objetos por uma perspectiva histórica: a Programação Procedural surgiu em um período em que o usuário deveria ser operador de sua máquina, pois o armazenamento num computador era excessivamente limitado pré-arquitetura de memória de von Neumann e, por consequência, usuário e programador eram uma única entidade, a qual devia se ater à utilização de memória de cada um de seus processos, cuidar manualmente da desalocação de memória dinâmica em "nível de usuário" e se atentar à maior eficiência em questão de endereçamento possível.

Era uma época de diálogos e brigas fortes com o hardware: o nível de abstração era extremamente baixo; muito controle, pouca segurança.

Em suma, a Programação Procedural atém-se ao princípio mais puro de execução ordenada e a conversa com os dispositivos de hardware disponíveis na máquina é um de seus atributos.

Exemplos de linguagens com suporte ao paradigma de Programação Procedural são: C, BASIC, COBOL e Fortran.

2.2 Programação Declarativa

Num outro extremo, sem ser imperativo ao controlar os estados de geração de saídas de uma máquina, temos a Programação Declarativa e seus subgrupos. Numa forma mais lúdica, ao invés de atacar um problema pelo centro, como a Programação Imperativa, a Programação Declarativa deixa a critério da máquina como será solucionado um dado problema; o trabalho do programador se restringe a descrever propriedades do resultado que deseja, sem explicar como efetivamente computá-lo. [tds]

1. Programação Funcional;
2. Programação Lógica;
3. Programação Matemática; e
4. Programação Reativa.

2.2.1 Programação Funcional

Visto como um paradigma auxiliar para algumas linguagens multiparadigmas, o Paradigma Funcional teve suas raízes no sentido matematicamente mais puro quando diz respeito às funções: são mecanismos para mapear uma entrada a uma saída.

Contudo, diferentemente dos paradigmas anteriormente citados, funções são instrumentos de alta hierarquia na Programação Funcional: podem receber nomes e serem tratadas como variáveis, podem retornar outras funções, podem criar funções dentro de si mesmas e, estas últimas, podem fazer o mesmo.

Neste paradigma, faz-se a analogia de que funções são para a Programação Funcional o que objetos seriam para Programação Orientada à Objetos.

Exemplos de linguagens com suporte ao paradigma de Programação Funcional: Haskell, Scheme, Common Lisp e Clojure.

2.2.2 Programação Lógica

Paradigma de Programação Declarativa enraizado na declaração de silogismos lógicos que, ao dialogar com outros silogismos, devem produzir uma saída expressa como Verdadeiro ou Falso. Como em lógica formal, assumir que um dado predicado P é verdadeiro ou falso, e aplicá-lo em conjunto com outros predicados promove a criação de silogismos que resultarão numa saída de acordo com a entrada.

Exemplos de linguagens garantem suporte ao paradigma de Programação Lógica: Prolog, ASP e Datalog. [log]

2.2.3 Programação Matemática

Apegada à otimização de cálculos, a Programação Matemática constitui-se da seleção dos melhores elementos de conjuntos de dados gerados por equações a fim de que se possa encontrar otimizações [mtm], como em problemas em que queremos obter o maior lucro com o menor custo possível em um dado modelo de negócio.

Exemplos de linguagens que dialogam com o paradigma de Programação Matemática: MATLAB e Octave.

2.2.4 Programação Reativa

Paradigma adotado por algumas linguagens de descrição de hardware, a Programação Reativa baseia-se no suporte ao fluxo de dados para garantir, então, o fluxo de execução de um dado programa. Em Verilog, por exemplo, é comum realizar a declaração de um dado sinal elétrico que se torna logicamente ativo ou não de acordo com uma condição a ser preenchida [rea], como o envio de um sinal para indicar início da operação da máquina. Em modelos como este, variáveis declaradas no mesmo nível de hierarquia, correspondentes a outros sinais, produzem saídas de acordo com o nível lógico exprimido por seus antecessores.

Outro exemplo de linguagem com suporte à Programação Reativa: Elm. [rea]

Programação Orientada a Agentes

A Programação Orientada a Agentes tem seu berço no ramo de Inteligência Artificial Simbólica.

Para facilitar a explicação, pode ser vista como uma versão alterada do que seria o paradigma de Programação Orientada à Objetos, em que um Agente é análogo a um Objeto em P.O.O., porém, de acordo com Yoav Shoham [Sho93], detém uma série de condições que, em conjunto, recebem o nome de *estado mental*; são estas crenças, decisões, capacidades e obrigações.

Um Agente que carrega essas quatro variáveis busca um objetivo, o qual tentará atingi-lo da melhor forma possível; isto recebe o nome de *racionalidade*. [aop]

Prover *racionalidade* a um agente lhe dá autonomia para decidir se executará ou não um método, ou se responderá ou não a uma mensagem, por exemplo. No caso, oriunda de uma quantidade autonomia e *racionalidade*, um agente pode elaborar uma solução para um problema que, talvez, a princípio tivesse soado não-ortodoxa ou que, até, nem tenha sido cogitada por seu projetista. [aop]

References

[aop] *O que é programação orientada a agentes?* https://pt.stackoverflow.com/questions/80614/o-que-%C3%A9-programa%C3%A7%C3%A3o-orientada-a-agentes

[log] *Logic Programming.* https://en.wikipedia.org/wiki/Logic_programming

[mic] *Definição de paradigma.* https://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/paradigma

[mtm] *Mathematical optimization.* https://en.wikipedia.org/wiki/Mathematical_optimization

[rea] *Reactive Programming.* https://en.wikipedia.org/wiki/Reactive_programming

[Sho93] SHOHAM, Yoav: Agent-oriented programming. In: *Artificial Intelligence* 60 (1993), Nr. 1, 51-92. http://dx.doi.org/https://doi.org/10.1016/0004-3702(93)90034-9. – DOI https://doi.org/10.1016/0004–3702(93)90034–9. – ISSN 0004–3702

[tds] *Towards Data Science: What Is A Programming Paradigm.* https://towardsdatascience.com/what-is-a-programming-paradigm-1259362673c2

[wik] *Programming paradigm.* https://en.wikipedia.org/wiki/Programming_paradigm