# Gerador de Classes

*Yohanês da Silva Zanghelini*[1]; *Rodrigo Curvello*[2]

[1] Estudante de Graduação em Ciência da Computação, IFC - *Campus* Rio do Sul. E-mail: yzanghelini@gmail.com
[2] Orientador, Professor EBTT, IFC - *Campus* Rio do Sul. E-mail: rodrigo.curvello@ifc.edu.br

## RESUMO

O propósito do presente trabalho foi desenvolver um gerador de classes em diferentes linguagens de programação. A princípio, a ideia inicial do projeto era criar um tradutor de classes entre linguagens, mas optou-se por uma abordagem de geração de código a partir de um conjunto de especificações. A estrutura principal inclui uma classe "Class" que armazena elementos como nome, modificador, atributos e métodos, utilizando listas para organizá-los. O processo de geração é gerido pela classe "Generator", que seleciona a linguagem alvo e utiliza classes específicas para cada linguagem, como "javaClass" e "pythonClass", que implementam a interface "classInterface". O gerador facilita a criação de classes ao permitir a concatenação de métodos e centraliza as operações através do padrão Facade.

**Palavras-chave**: Gerador de classes; Padrões de projeto; Modularidade; Flexibilidade; Linguagens de programação.

## INTRODUÇÃO

A geração automática de código é uma prática que pode aumentar significativamente a produtividade e a consistência no desenvolvimento de software. Tradicionalmente, um tradutor de classes seria utilizado para converter código de uma linguagem para outra. No entanto, devido à complexibilidade de traduzir uma classe já existente, optou-se por desenvolver um gerador de classes, capaz de criar novas classes conforme a linguagem de programação selecionada. Como afirma Parnas (1972), "a modularidade no design de software permite que sistemas complexos sejam divididos em partes menores e mais gerenciáveis, o que facilita a manutenção e evolução do software." Com o objetivo de aprimorar a manutenibilidade e modularização do código, o gerador foi desenvolvido utilizando padrões de projeto como Builder, Facade e Factory Method. Este artigo descreve a estrutura e implementação do gerador, que oferece uma solução prática para desenvolvedores que trabalham com múltiplas linguagens.

## PROCEDIMENTOS METODOLÓGICOS

O gerador de classes foi desenvolvido utilizando Java como linguagem base. A estrutura principal envolve a criação de uma classe "Class" para armazenar os elementos que há em uma classe padrão. Em seguida, uma classe "Generator" gerencia a criação desses elementos e sua organização. A classe "Class", armazena nome, modificador, os atributos e os métodos de uma classe. Os métodos e atributos são armazenados em ArrayLists do tipo "Method" e "Attribute", respectivamente. Ambas as classes contêm um tipo, modificador e nome, que são atributos do tipo String. A classe "Class" é utilizada na classe "Generator", que é responsável por iniciar o processo de armazenamento dos elementos da classe e pela geração do código da classe na linguagem selecionada. Utilizando o padrão Factory Method, implementamos a interface "classInterface" nas classes "Generator", "javaClass" e "pythonClass". Essa interface define os


ISSN 2318-2342

métodos necessários para a geração de uma classe. Cada classe específica, como "javaClass", implementa essa interface e fornece a lógica específica para gerar os elementos de uma classe na respectiva linguagem. Como mencionado anteriormente, utilizamos a classe "Class" para armazenar todos os elementos de uma classe, e essa mesma estrutura é utilizada tanto na "javaClass" quanto na "pythonClass" para manipulação dos dados e para gerar um StringBuilder contendo o código da classe. Ao instanciar um novo objeto da classe "Generator", é passado como parâmetro um valor do enum "optionLanguage". Se o parâmetro selecionado for "JAVA", uma nova instância da classe "javaClass" é atribuída ao objeto cInterface; se for "PYTHON", uma instância de "pythonClass" é atribuída a cInterface.

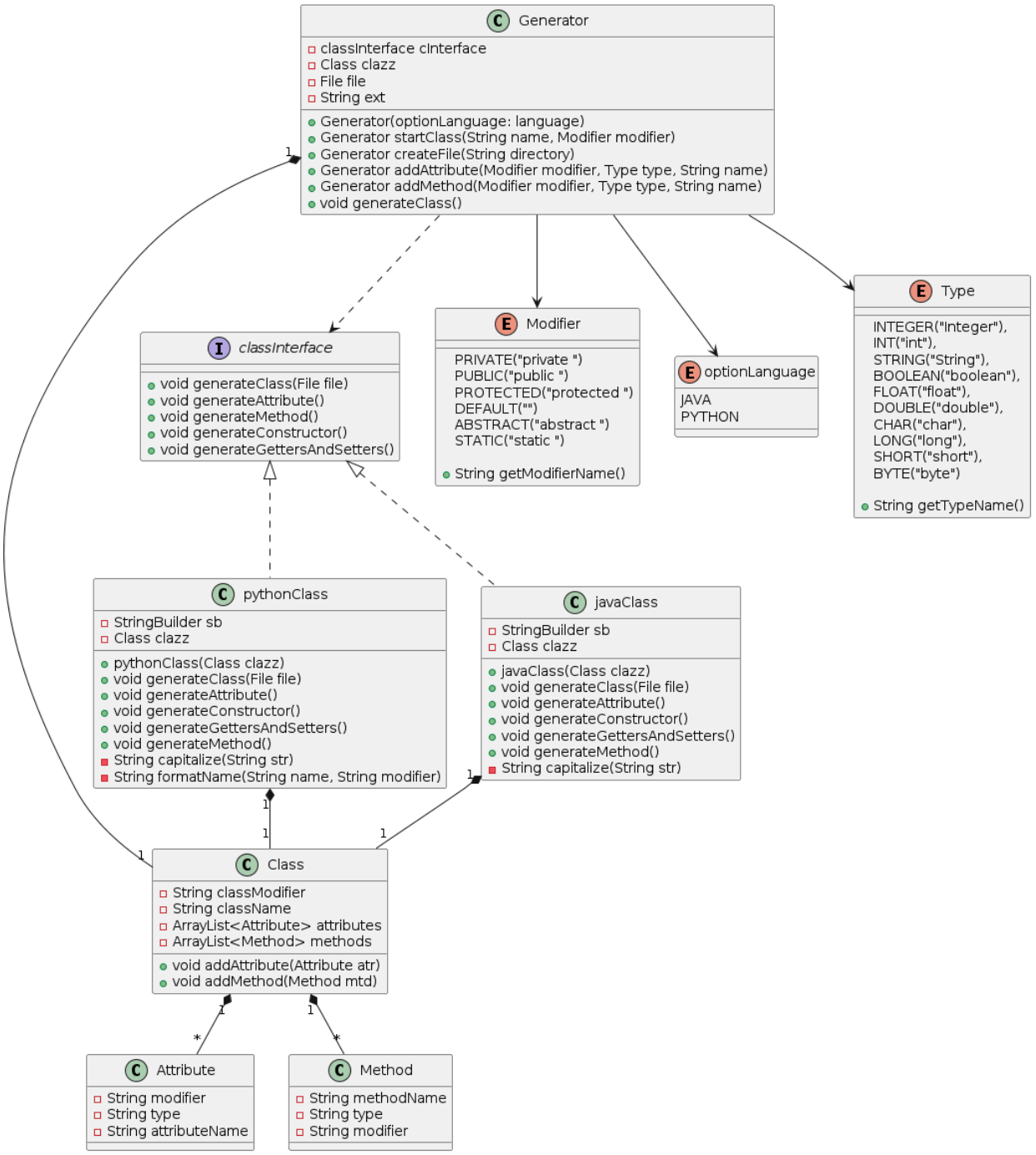
A classe "Generator" também é responsável por criar o arquivo com a extensão da linguagem selecionada, definir nome, modificador, adicionar atributos e métodos ao objeto da classe "Class" e principalmente gerar a classe utilizando o cInterface. Os métodos da classe Generator retorna uma instância dela mesma (this), permitindo a concatenação de métodos como adicionar atributos, facilitando a criação da classe, utilizando o padrão de projeto Builder. Para facilitar o uso, o projeto faz o uso também do padrão de projeto facade. A classe contém todos os métodos necessários para realizar o processo de criação da classe. Centralizando todos os métodos que devem ser utilizados.

## RESULTADOS E DISCUSSÃO

O gerador desenvolvido demonstrou eficácia na criação de classes em diferentes linguagens, permitindo que o código seja modular e facilmente extensível para novas linguagens de programação. O uso dos padrões de projeto garantiu que a solução fosse flexível, facilitando a adição de novos elementos ou a modificação de elementos existentes sem afetar a estrutura geral. A aplicação prática deste gerador pode reduzir significativamente o tempo de desenvolvimento em projetos que requerem a criação de classes em múltiplas linguagens, além de melhorar a consistência do código gerado. A discussão sobre a aplicação dos padrões de projeto no contexto deste gerador destacou a importância de utilizar abordagens que permitissem a modularidade e a flexibilidade do código, permitindo assim a adição de novas opções de linguagens de programação sem interferir nas classes já existentes.


ISSN 2318-2342

**Figura 01:** Diagrama de classes UML





ISSN 2318-2342

**Figura 02:** Exemplo de utilização da classe "Generator" para criar uma classe "Carro".

```
public static void main(String[] args) {

        Generator gn = new Generator(PYTHON);

        gn.startClass("Carro", PUBLIC)
        .addAttribute(PRIVATE, STRING, "modelo").addAttribute(PRIVATE, INTEGER, "ano")
        .addAttribute(PRIVATE, STRING, "cor")
        .addAttribute(PRIVATE, DOUBLE, "preco")
        .addMethod(PUBLIC, BOOLEAN, "iniciarMotor")
        .addMethod(PUBLIC, BOOLEAN, "acelerar")
        .addMethod(PUBLIC, BOOLEAN, "frear")
        .createFile("demo/src/main/java/com/yohanesz")
        .generateClass();

    }
```


ISSN 2318-2342

**Figura 03:** Classe "Carro" gerada utilizando a classe "Generator".

```python
class Carro:

    def __init__(self):
        pass

    def __init__(self, __modelo, __ano, __cor, __preco):
        self.__modelo = __modelo
        self.__ano = __ano
        self.__cor = __cor
        self.__preco = __preco

    def get_Modelo(self):
        return self.__modelo

    def set_Modelo(self, value):
        self.__modelo = value

    def get_Ano(self):
        return self.__ano

    def set_Ano(self, value):
        self.__ano = value

    def get_Cor(self):
        return self.__cor

    def set_Cor(self, value):
        self.__cor = value

    def get_Preco(self):
        return self.__preco

    def set_Preco(self, value):
        self.__preco = value

    def iniciarMotor(self):
        pass

    def acelerar(self):
        pass

    def frear(self):
        pass

```


ISSN 2318-2342

## CONSIDERAÇÕES FINAIS

O desenvolvimento de um gerador de classes que utiliza padrões de projeto como Builder, Facade e Factory Method demonstrou ser uma abordagem eficaz para aumentar a modularidade, flexibilidade e manutenibilidade no desenvolvimento de software. A escolha de criar um gerador, em vez de um tradutor de classes, mostrou-se vantajosa, pois permitiu a construção de classes totalmente novas, adaptadas às especificidades de cada linguagem de programação selecionada.

Este trabalho evidencia a importância da modularidade e do uso adequado de padrões de projeto na criação de ferramentas que visam simplificar e agilizar o processo de desenvolvimento, especialmente em contextos onde múltiplas linguagens de programação são utilizadas. Além disso, o gerador desenvolvido oferece aos desenvolvedores uma solução prática e escalável, que pode ser expandida para suportar outras linguagens e requisitos específicos.

Futuras melhorias poderiam incluir a extensão do gerador para suportar uma maior variedade de linguagens de programação e a adição de funcionalidades que permitam maior personalização na geração do código, atendendo a necessidades específicas de diferentes projetos e arquiteturas.

## REFERÊNCIAS

**Parnas, D. L. (1972).** "On the criteria to be used in decomposing systems into modules." *Communications of the ACM, 15*(12), 1053-1058.

**Factory Method em Java / Padrões de Projeto.** Disponível em: https://refactoring.guru/pt-br/design-patterns/factory-method/java/example. Acesso em: 15 ago. 2024.

**Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994).** *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

**Fowler, M. (1999).** *Refactoring: Improving the Design of Existing Code.* Addison-Wesley.


ISSN 2318-2342