AULA 10 – PADRÃO *SINGLETON*

GSI020 – Programação Orientada a Objetos II

Prof. Dr. Murillo G. Carneiro
mgcarneiro@ufu.br

UFU

Material baseado nos slides cedidos pelo Prof. Rafael D. Araújo (FACOM/UFU)

Objetivo da aula

- Entender o funcionamento do padrão de projeto *Singleton*.

Motivação

- Jogo de xadrez
  - *Faz sentido um único tabuleiro*
- Configuração para acesso ao banco de dados
- Sistema gerenciador de janelas
- Infraestrutura de logs
- A fábrica de bebidas criada nas aulas anteriores
- Como assegurar que uma classe possua apenas uma instância e que esta instância compartilhada com toda a aplicação?

Singleton

- É um padrão de projeto de propósito **de criação** com escopo de **objetos**.
- É um padrão simples que evita que uma classe seja instanciada mais de uma vez na aplicação.

Intenção

- O objetivo é garantir uma instância única da classe que implementa este padrão e fornecer um ponto único e global de acesso para a mesma.
- A própria classe é responsável por gerenciar sua única instância.

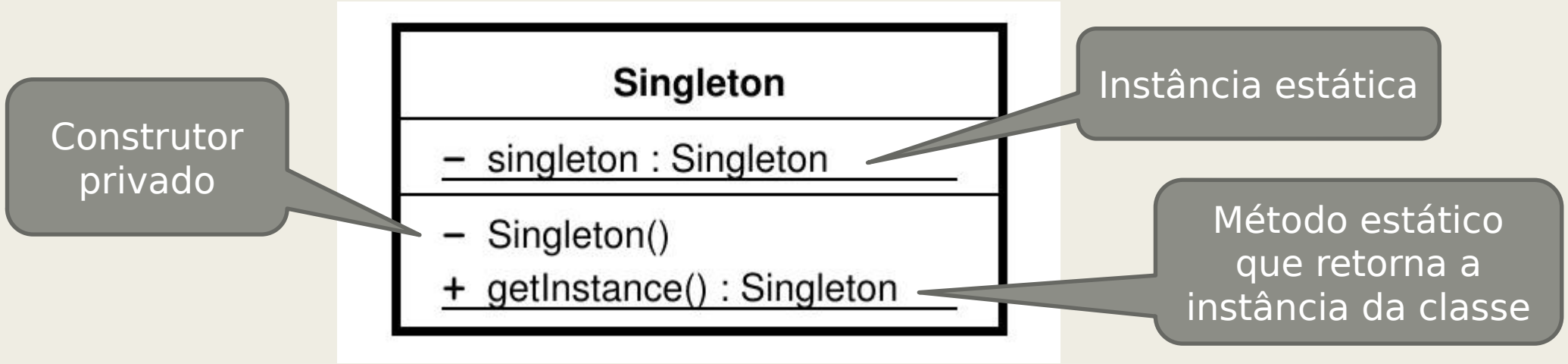
Quando usar

- Quando for necessário haver apenas uma instância de uma classe e essa instância tiver que dar acesso por meio de um ponto único
  - *Como se fosse uma variável global*
- A única instância tiver que ser extensível através de subclasses

Elementos participantes

- Singleton: define uma operação estática getInstance que permite aos clientes acessarem sua única instância.
  - *O singleton pode criar sua instância antecipadamente ou apenas no momento da primeira utilização*

Estrutura


Singleton
- singleton : Singleton
- Singleton()
+ getInstance() : Singleton
Construtor privado
Instância estática
Método estático que retorna a instância da classe

Benefícios

- Acesso controlado à instância única
- Evita poluição do espaço de nomes com variáveis globais que possuem uma única instância
- Permite refinamento de operações e da representação da instância única
- Permite um número variável de instâncias
  - *Apesar da definição ser para uma única instância, é possível controlar e definir um número fixo de instâncias*

Desvantagens

- Acoplamento em uma implementação estática e específica
  - *Código dependente dessa classe*
- Necessita de cuidado com a implementação de concorrência
- Uso indevido => *antipattern*
  - *não deve ser utilizado para poupar número de instâncias*

Padrão comumente usado mas é ineficiente e/ou contra-produtivo em prática (mal uso)

Implementação com instanciação antecipada (*early instantiation*)

```
public class Singleton {
  private static Singleton _INSTANCIA = new Singleton();
  private Singleton () { ... }

  public static Singleton getInstancia() {
   return _INSTANCIA;
  }
}
```

Implementação com instanciação sob demanda (*lazy instantiation*)

```
public class Singleton {
  private static Singleton _INSTANCIA = null;

  private Singleton () { ... }


  public static synchronized Singleton getInstancia() {
   if (_INSTANCIA == null)
   _INSTANCIA = new Singleton();
   return _INSTANCIA;
  }
}
```

Utilização do *Singleton*

```
...
Singleton s = Singleton.getInstancia();
...
```

Problema prático

- Faça com que a máquina de bebidas criada nas aulas anteriores se torne um *singleton*.
  - *Implemente as duas formas de instanciação (early e lazy)*

Referências

- GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1995. Capítulo 3.