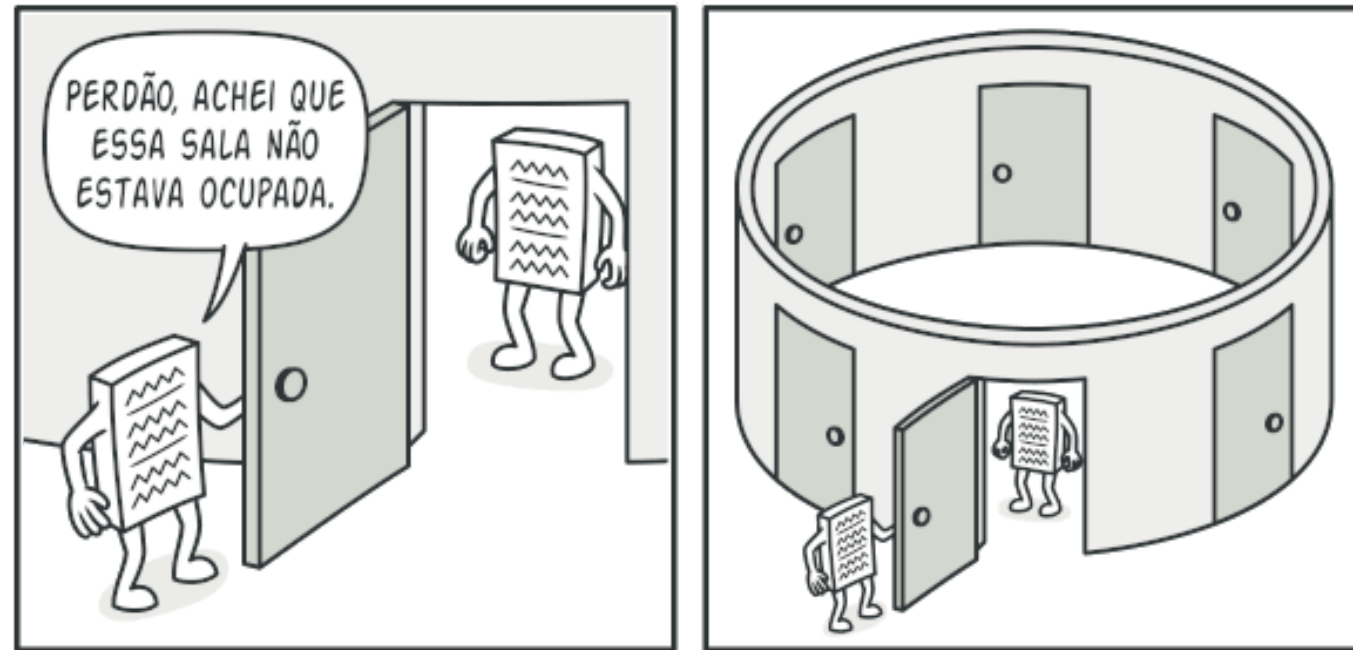


TÓPICO 04: SINGLETON

Design Patterns - Professor Ramon Venson - SATC 2026.2

Definição

Singleton é um padrão de projeto que garante que uma classe tenha **uma única instância** e fornece um **ponto de acesso global** a essa instância.



Problema

Alguns recursos devem ser **únicos** na aplicação, como configurações, loggers e gerenciadores de conexões. Se cada módulo fizer `new` para criar sua própria instância, isso pode levar a:

- configurações diferentes espalhadas
- consumo excessivo de recursos
- difícil coordenar quem manda no estado

Neste exemplo, temos dois objetos diferentes que instanciam a mesma classe:

```
public class FaturaService {  
    private ConfiguracaoSistema config = new ConfiguracaoSistema();  
  
    public void processarFatura() {  
        System.out.println("Configuração: " + config.getValor());  
    }  
}
```

```
public class RelatorioService {  
    private ConfiguracaoSistema config = new ConfiguracaoSistema();  
  
    public void gerarRelatorio() {  
        System.out.println("Configuração: " + config.getValor());  
    }  
}
```

Uma das soluções sugere utilizar uma classe com **métodos estáticos**:

```
public class ConfiguracaoSistema {  
    private static String valor = "configuração padrão";  
  
    public static String getValor() {  
        return valor;  
    }  
  
    public static void setValor(String novoValor) {  
        valor = novoValor;  
    }  
}
```

Porém isso não é o ideal, pois a classe não pode implementar interfaces, não pode ser herdada e não pode ser instanciada. Também não é possível controlar a criação da instância, o que pode levar a problemas de concorrência.

Solução

Vamos criar uma solução que garanta **uma única instância** e forneça um **ponto de acesso global**, além de permitir a implementação de interfaces e herança e o controle da criação da instância.

- **Construtor privado**: apenas a própria classe pode instanciar a si mesma
- **Campo estático**: guarda a instância única
- **Método estático** (`getInstancia`): devolve sempre o mesmo objeto

Implementação

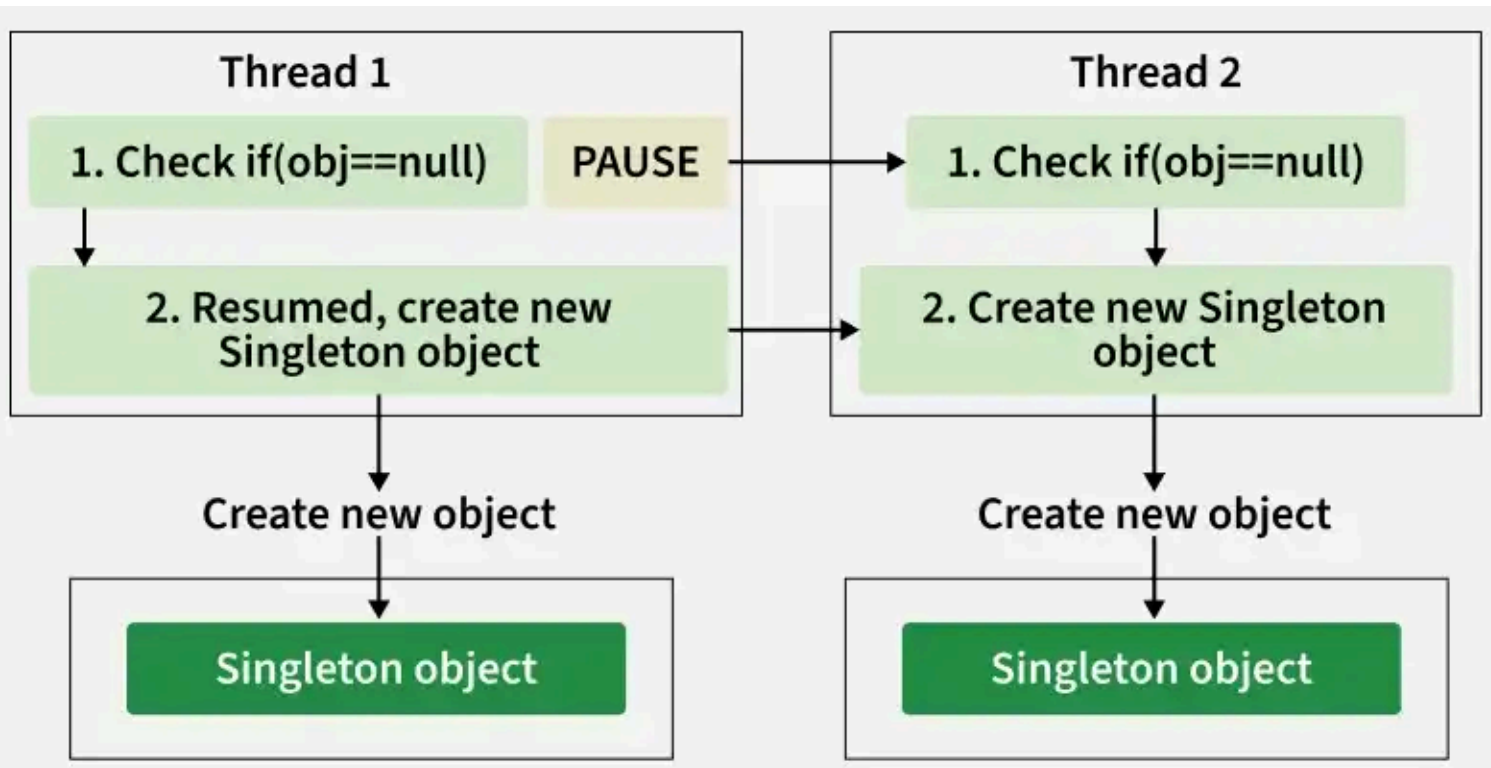
```
public class ConfiguracaoSistema {  
    private static ConfiguracaoSistema instancia;  
  
    // Construtor privado impede instanciamento externo  
    private ConfiguracaoSistema() {  
    }  
  
    // Retorna a instância única, criando-a se necessário  
    public static ConfiguracaoSistema getInstancia() {  
        if (instancia == null) {  
            instancia = new ConfiguracaoSistema();  
        }  
        return instancia;  
    }  
}
```

Utilização

O método estático garante que sempre será retornado o mesmo objeto, independente de quantas vezes seja chamado:

```
ConfiguracaoSistema c1 = ConfiguracaoSistema.getInstancia();  
ConfiguracaoSistema c2 = ConfiguracaoSistema.getInstancia();  
System.out.println(c1 == c2); // true
```

A primeira chamada é responsável por criar a instância, e as próximas chamadas apenas retornam a mesma instância.



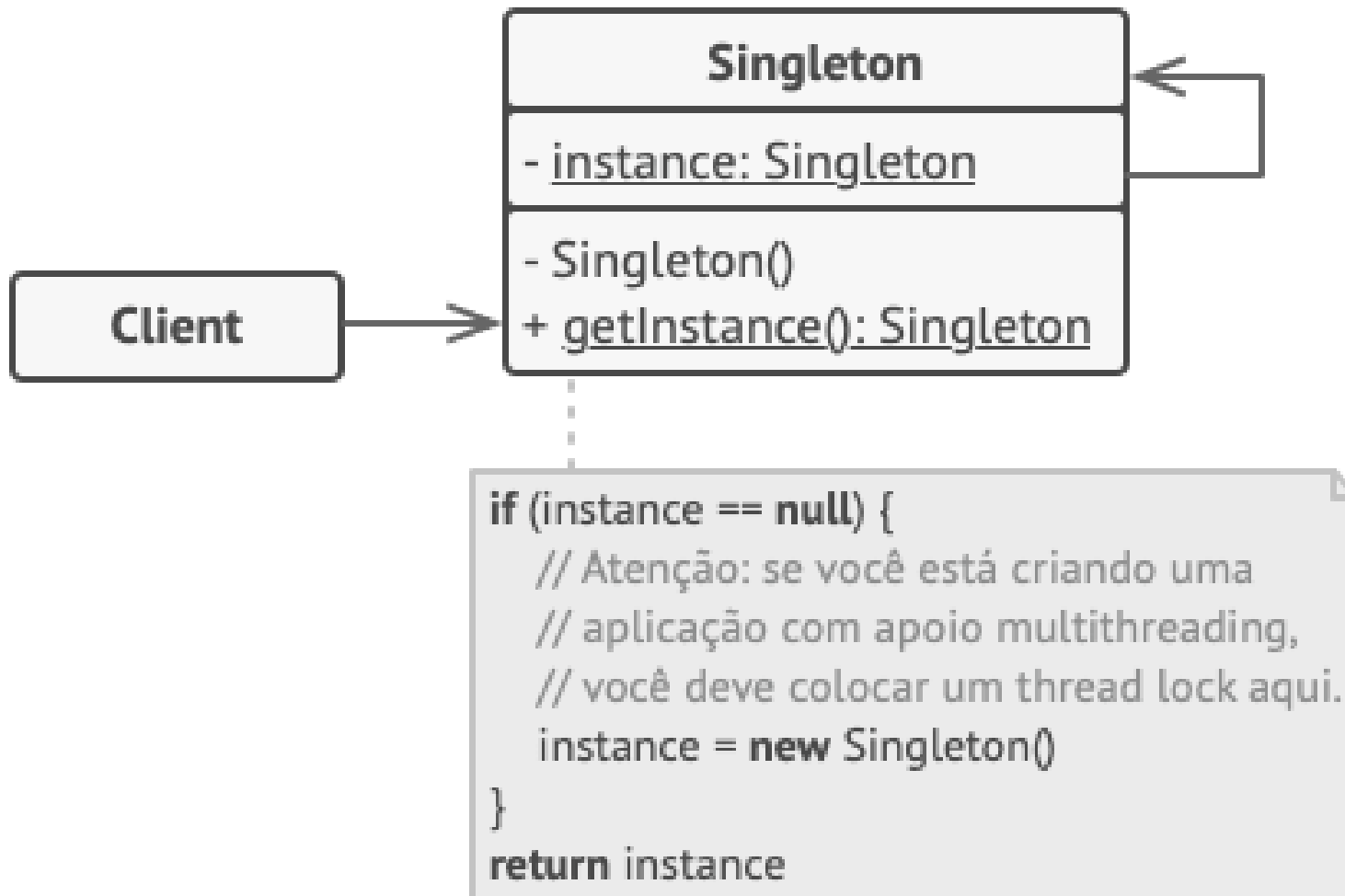
Thread-safe

Essa solução, no entanto, não é **thread-safe**. Em um ambiente multi-thread, duas threads podem chamar `getInstancia()` ao mesmo tempo e criar duas instâncias diferentes.

Para resolver isso, no java, podemos usar **sincronização**:

```
public static synchronized ConfiguracaoSistema getInstancia() {  
    if (instancia == null) {  
        instancia = new ConfiguracaoSistema();  
    }  
    return instancia;  
}
```

Isso garante que apenas uma thread por vez possa executar o método, evitando a criação de múltiplas instâncias.



Diagramas

A classe declara um **construtor privado** e um **método estático** que retorna a instância única.

Cliente é qualquer classe que precise acessar a instância única da classe Singleton.

Analogias

- **Central de emergências** (tipo 190/192/193):
 - Vários telefones, uma mesma central
 - Vários módulos do sistema, uma mesma instância
- Não faz sentido cada telefone ter sua própria “mini central”
- Singleton funciona como esse **serviço único** na aplicação

Aplicações

- Classes que só precisam de **uma única instância**:
 - configuração global
 - logger
 - gerenciador de conexões
 - cache em memória

Prós

- **Instância única:** garante que só exista uma instância da classe
- **Acesso simples:** pode ser acessada por meio do método estático
`MinhaClasse.getInstance()`
- **Lazy loading:** a instância é criada apenas quando necessária
- **Estado global:** permite compartilhar estado entre diferentes partes do sistema

Contras

- **Acoplamento global:** a classe está fortemente acoplada ao sistema, o que dificulta a manutenção e a reutilização
- **Dificuldade de testes unitários:** requer mocks para testar, pois a instância é global
- **Violação de SRP:** a classe pode assumir responsabilidades que não são suas, como gerenciar a própria instância

Resumo

- Singleton: **uma instância + acesso global controlado**
- Útil para recursos realmente **únicos**
- Use com cuidado: fácil virar anti-padrão
- Considere alternativas:
 - injeção de dependências
 - gerenciamento por contêiner (ex.: Spring singleton scope)