

Padrões de Projeto

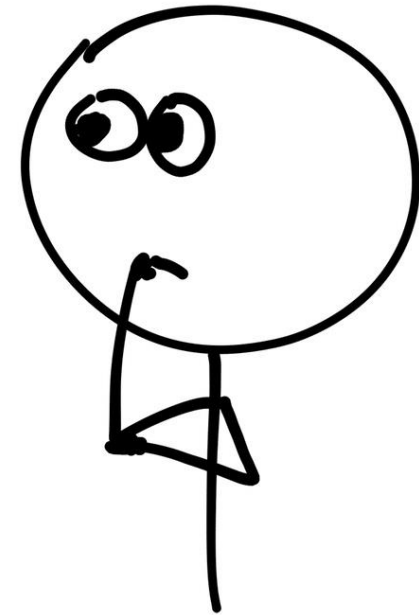
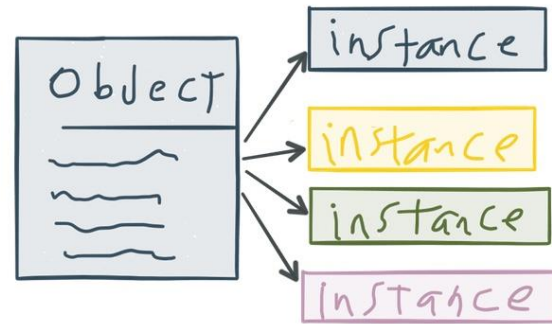
Prof. Adilson Vahldick

Departamento de Engenharia de Software

Udesc Ibirama

Objetivos da aula

- Conhecer e aplicar o padrão
 - Singleton



Problema (1)

- Nosso sistema precisa de um recurso para calcular os impostos.
- **Solução:** criar uma classe para calcular os impostos.

Problema (2)

```
package meusistema.fiscal;

public class CalcImposto {
    private float aliquota;
    public float calc(float valor) {
        // calculo do imposto
    }
}
```

```
CalcImposto imp1 = new CalcImposto();
imp1.setAliquota(10f);
float icms1 = imp1.calc (100f);

...

CalcImposto imp2 = new CalcImposto();
imp2.setAliquota(10f);
float icms2 = imp2.calc (150f);
```

É realmente necessário criar duas instâncias da classe ?

Eu poderia usar sempre a mesma instância.

Problema (3)

- Como resolver ?
- Primeira solução: uma classe sem estado (Stateless)

```
package meusistema.fiscal;  
  
public class CalcImposto {  
  
    private static float aliquota;  
  
    public static float calc(float valor) {  
        // calculo do imposto  
    }  
}
```

```
float icms1 = CalcImposto.calc(100);  
  
...  
  
float icms2 = CalcImposto.calc(150);
```

Problema (3)

- Como resolver ?
- Primeira solução: uma classe com estado (Stateless)

Não temos mais instâncias. Menos consumo de memória

```
package com.example.project;\n\npublic class CalcImposto {\n\n    private static float ...\n\n    public static float ...\n    // calcula ...\n}\n\nns2 = CalcImposto.calc(150);
```

E se quisermos serializar ?

E se tivesse que implementar uma interface ?

E se precisarmos definir uma estrutura de herança ?

Problema (4)

- **Singleton:** Garantir que uma classe tenha somente uma instância e fornecer um ponto global de acesso a mesma

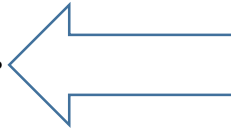
```
public class Singleton {
```

```
    private Singleton() { }
```



Construtor privado

```
    private static Singleton instance;
```



Atributo estático para controlar a instância

```
    public synchronized static Singleton getInstance() {  
        if (instance == null)  
            instance = new Singleton();  
  
        return instance;  
    }
```

Um ponto único no sistema para fornecer a instância

```
}
```

Problema (5)

- **Singleton:** uso do padrão (***singleton1***)

```
Singleton s1 = Singleton.getInstance();  
s1.facaAlgo();
```


Problema (6)

- Nosso sistema precisa de um recurso para calcular os impostos.
- **Solução:**
 - 1-criar uma classe para calcular os impostos.
 - 2-aplicar o padrão Singleton nessa classe

RESOLVAM !!!

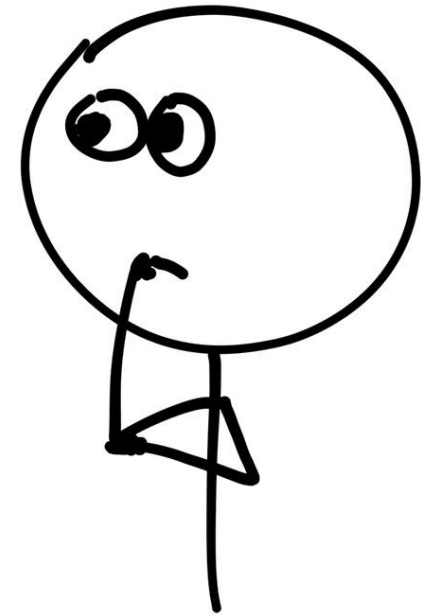
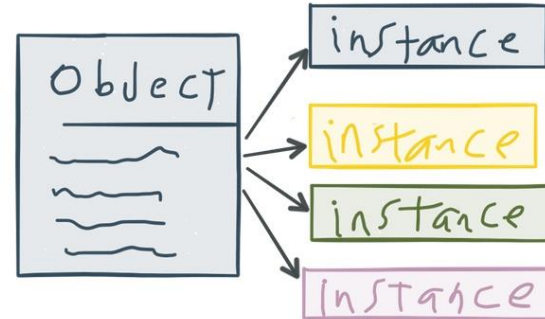
Baixar e refatorar *singleton2*

```
package meusistema.fiscal;

public class CalcImposto {
    private float aliquota;
    public float calc(float valor) {
        // calculo do imposto
    }
}
```

Singleton

- Aplicações
 - Log
 - Cache
 - Configurações
 - Acesso a recursos externos
 - Banco de dados



Exercício 2 (*singleton3*)

- Crie uma classe para incrementar o valor e que consiga fornecer esse valor. Aplicar o padrão Singleton.
- Use a classe StartWindows para testar Incrementar. Remova os comentários dessa classe.

```
public class Incrementar {  
  
    private int conta = 0;  
  
    public void inc() { conta++; }  
  
    public int getConta() { return conta; }  
  
}
```

Singleton X Métodos estáticos

```
public class Incrementar {  
    private int conta = 0;  
    public void inc() { conta++; }  
    public int getConta() { return conta; }  
}
```

```
public class Incrementar {  
    private static int conta = 0;  
    public static void inc() { conta++; }  
    public static int getConta() { return conta; }  
}
```

Singleton X Métodos estáticos

- Uma classe sem estado hoje pode vir a ser com estado no futuro
 - Requisitos mudam 😞
- Herdar de uma outra classe
- Ser herdada (Como fazer isso em Java ?)
- Implementar uma interface
- Ser serializada
- Ser passada para outras classes
- Ser testada de forma mais fácil

Exercício 3 (1)

- Log é um processo de registro de eventos relevantes. Com base na interface abaixo, crie duas classes para log: uma para armazenar os registros em arquivo texto (use PrintWriter) e outra para imprimir na console. Essas duas classes devem ser Singleton.

```
public interface Log {  
  
    void atencao(String mensagem) ;  
    void erro(String mensagem) ;  
    void info(String mensagem) ;  
  
}
```

Exercício 3 (2)

- Crie uma classe Logger que
 1. Realize a interface Log
 2. Aplique Singleton
 3. No constructor leia de um arquivo de configuração o tipo de log a ser usado. Use a classe `java.util.Properties` para ler a configuração.

```
Properties props = new Properties();  
props.load(new InputStreamReader(new FileInputStream(new File("conf.properties"))));  
String tipo = props.getProperty("tipo");
```

tipo=console

tipo=arquivo
nome=teste.txt

Exercício 3 (3)

- Experimente usar esse sistema de Log em alguma das aplicações MVC que já tenhas feito.

Singleton

- Pelo último exemplo podemos concluir que Singleton é uma maneira de termos uma **Variável Global** no sistema !!!

REFERÊNCIAS

- GAMMA, Erich et al. **Padrões de projeto:** soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.
- SHALLOWAY, Alan; TROTT, James. **Explicando padrões de projeto:** uma nova perspectiva em projeto orientado a objeto. Porto Alegre: Bookman, 2004.