# GCC178 - Práticas de Programação Orientada a Objetos

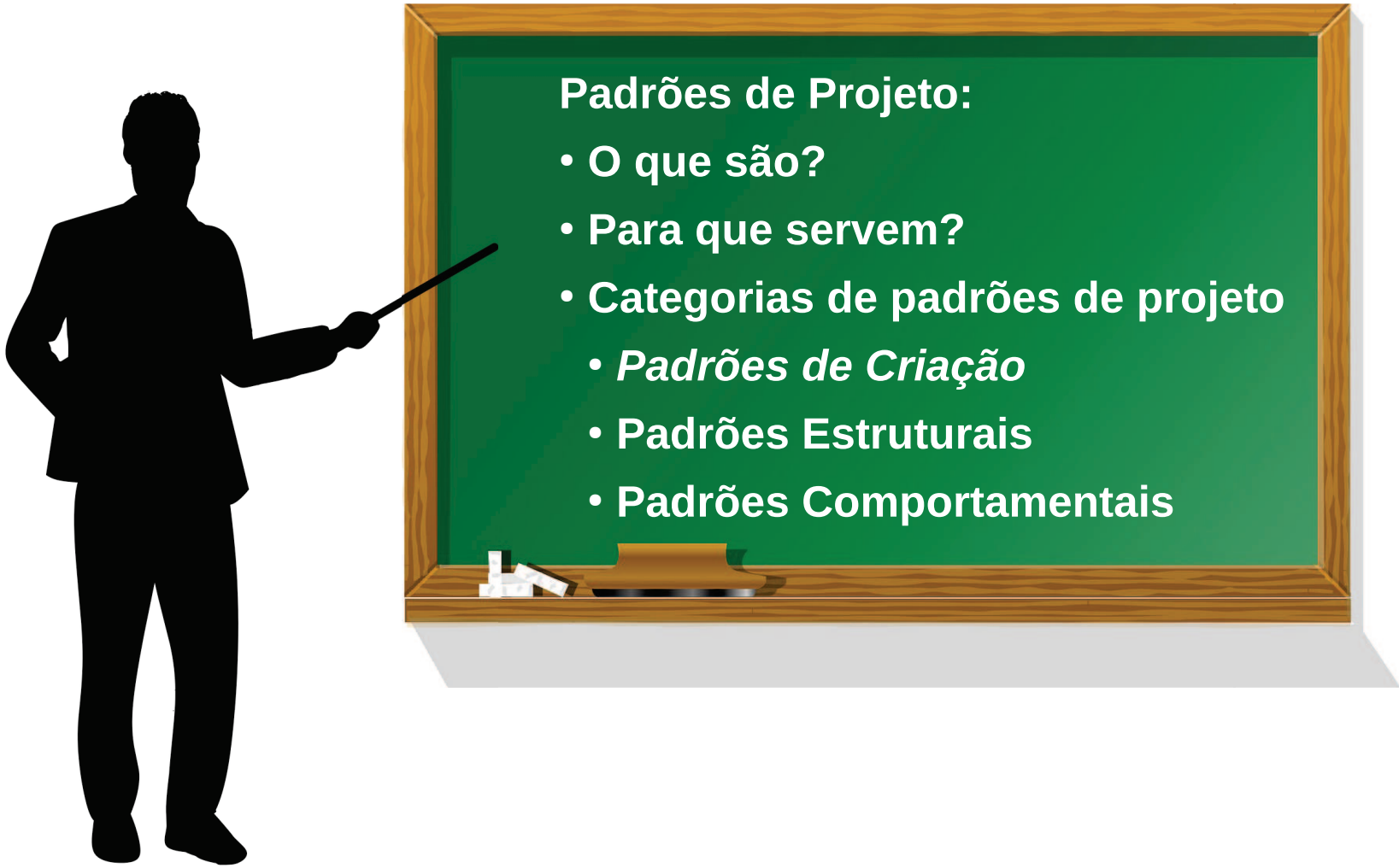
## Padrões de Projeto

Luiz Henrique de Campos Merschmann
Departamento de Ciência da Computação
Universidade Federal de Lavras

luiz.hcm@ufla.br

UFLA
UNIVERSIDADE FEDERAL DE LAVRAS

# Na Aula de Hoje



# Reutilização

O projeto...

- Projetar software orientado a objetos pode não ser muito fácil.
  - Mas projetar um software orientado a objetos **reutilizável** é ainda mais difícil.

O seu projeto deve ser **específico** para o **problema a ser resolvido**, mas também **genérico** o suficiente para **atender problemas e requisitos futuros**.

- Como fazer isso?
  - Projetistas experientes sabem que não devem resolver cada problema a partir de princípios elementares ou do zero.
  - Ao invés disso, eles **reutilizam soluções** que funcionaram no passado.

# Padrões de Projeto

## Como surgiram e para que servem?

- Quando projetistas encontram uma boa solução, eles a utilizam repetidamente.
- Consequentemente, surgem **padrões**, de classes e de comunicação entre objetos, que reaparecem com frequência em sistemas orientados a objetos.
- Esses padrões resolvem problemas específicos de projetos tornando-os mais flexíveis e, consequentemente, reutilizáveis.
- Esses padrões nos ajudam a reutilizar ideias (estratégias de solução) que já foram bastante testadas e bem-sucedidas.

UFLA
UNIVERSIDADE FEDERAL DE LAVRAS

# Padrões de Projeto

## O que são?

- Segundo Christopher Alexander: "Cada padrão descreve um problema no nosso ambiente e o cerne da sua solução, de tal forma que você possa usar essa solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira".
- Ou seja:
  - Um padrão de projeto descreve um problema comum que ocorre regularmente no desenvolvimento de software...
  - ... e descreve então uma solução geral para esse problema que pode ser usada em muitos contextos diferentes.

# Padrões de Projeto

De forma geral um padrão de projeto contém a descrição de algumas classes e a forma como elas se relacionam.

## Estrutura de um padrão:

- **Nome**
  - Expressa a sua essência de forma sucinta.
  - Facilita a comunicação das equipes de desenvolvimento.
- **Descrição do problema**
  - Explica o problema e seu contexto.
  - Descreve em qual situação a solução deve ser utilizada.
- **Descrição da solução**
  - Descreve os elementos que compõem o padrão de projeto, seus relacionamentos, responsabilidades e colaborações.
  - Essa solução pode ser vista como um gabarito (*template*) que pode ser aplicado em muitas situações diferentes.
- **Consequências**
  - São os resultados e análises das vantagens e desvantagens da aplicação do padrão.

UFLA UNIVERSIDADE FEDERAL DE LAVRAS

# Categorização

De acordo com a sua **finalidade**, os padrões de projetos podem ser classificados em:

- **Padrões de Criação:** se preocupam com o processo de criação de objetos.
  - Ex.: *Singleton*, *Factory Method*, *Builder*, *Prototype* etc.
- **Padrões Estruturais:** lidam com a composição de classes ou de objetos.
  - Ex.: *Decorator*, *Adapter*, *Composite*, *Façade* etc.
- **Padrões Comportamentais:** caracterizam as maneiras pelas quais classes ou objetos interagem e distribuem responsabilidades.
  - Ex.: *Strategy*, *Template Method*, *Observer*, *Iterator* etc.

# Singleton

Prof. Luiz Merschmann

# Padrão de Projeto *Singleton*

## Descrição do problema

- O padrão *Singleton* garante que uma classe tenha somente uma instância e fornece um ponto de acesso global para a mesma.

## Motivação

- É importante para algumas classes terem uma única instância.
- Exemplos: classes que cuidam de configurações, logs etc.

# Padrão de Projeto *Singleton*

Como garantir via código que uma classe tenha somente uma instância e que essa instância seja facilmente acessível?

Hummm... Acho que é impossível, pois qualquer um pode usar um **new** para criar um objeto de uma classe a qualquer momento.

Já sei...É só utilizarmos uma variável global! Fazemos isso declarando um atributo estático público em alguma classe. Desse modo, a variável global torna o objeto facilmente acessível.

Hummm... Acho que não isso não resolve o problema, pois a utilização de variável global não impede a instanciação de múltiplos objetos.

Portanto, o padrão de projeto **Singleton** é uma solução simples e amplamente testada para esse tipo de problema!

# Padrão de Projeto *Singleton*

Como criamos um objeto?

new MinhaClasse( );

Outras classes podem criar objetos *MinhaClasse*?

Com certeza!

Portanto, podemos dizer que uma classe pode ser instanciada várias vezes?

Podemos, mas para isso o construtor deve ser público.

E se declararmos o construtor como privado?

Humm...será que isso funciona?

O que significa declarar o construtor como privado?

Que a classe não pode ser instanciada!

Então nenhuma outra classe poderia usar o construtor?

Bom, a própria classe pode usar algo privado. Mas isso não faz sentido!

Por que?

Como eu criaria objetos *MinhaClasse* a partir de outras classes?

Mas e se utilizarmos o método a seguir em *MinhaClasse*?

```
public static MinhaClasse getInstance(){
   return new MinhaClasse( );
}
```

Que interessante! Agora temos uma nova maneira de instanciar objetos de uma classe.

UFLA
UNIVERSIDADE FEDERAL DE LAVRAS

# Exemplo com *Singleton*

Veja como fica a implementação de *MinhaClasse* a partir da ideia anterior.

```
public class MinhaClasse {
    //Atributos...

    private MinhaClasse() {
        //Inicializações necessárias...
    }

    public static MinhaClasse getInstance() {
        return new MinhaClasse();
    }
}
```

Como complementar esse código para garantir que apenas uma instância dessa classe será criada?

# Exemplo com *Singleton*

```
public class MinhaClasse {
    //Atributos...
    private static MinhaClasse instanciaUnica;

    private MinhaClasse() {
        //Inicializações necessárias...
    }

    public static MinhaClasse getInstance() {
        if(instanciaUnica == null) {
            instanciaUnica = new MinhaClasse();
        }
        return instanciaUnica;
    }
}
```

Usamos um atributo estático para referenciar nossa instância única da classe.

Construtor declarado como privado garante que apenas a própria classe pode criar instâncias.

Esse método nos permite instanciar um único objeto da classe e retorná-lo.

O método cria a instância apenas na primeira vez que é chamado. Nas próximas vezes, ele retorna a instância que já havia sido criada antes.

```
public class TesteMinhaClasse {

    public static void main(String[] args) {
        MinhaClasse mc;

        for (int i = 0; i < 1000; i++) {
            mc = MinhaClasse.getInstance();
        }
    }
}
```

Ainda que estejamos chamando 1000 vezes o método que cria uma instância de *MinhaClasse*, somente uma instância é criada.