

Computação Orientada a Objetos

Padrões de Projeto

Slides baseados em:

- E. Gamma and R. Helm and R. Johnson and J. Vlissides. Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
- Slides Prof. Christian Danniel Paz Trillo
- Slides Profa. Patrícia R. Oliveira

Profa. Karina Valdivia Delgado
EACH-USP

PADRÕES DE PROJETO

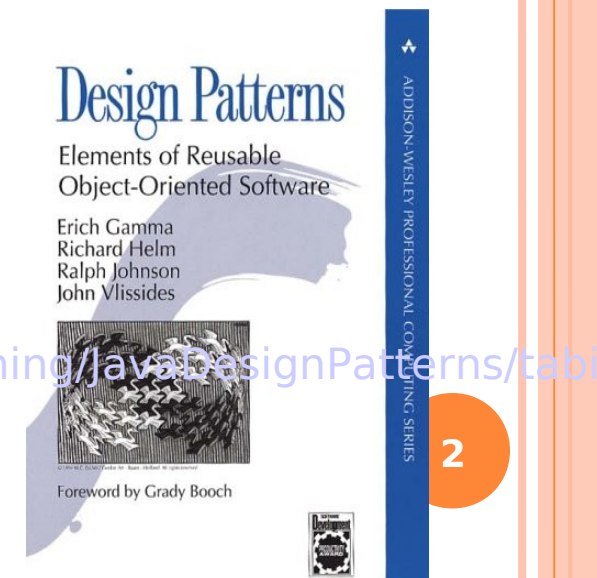
- ▶ E. Gamma and R. Helm and R. Johnson and J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
 - Conhecido como GoF (Gang of Four)
 - Versão em português disponível na biblioteca da EACH. (nome: Padrões de Projeto)

- ▶ Java Design Patterns At a Glance

- <http://www.javacamp.org/designPattern>

- ▶ Java Design Patterns Resource Center

- <http://www.deitel.com/ResourceCenters/Programming/JavaDesignPatterns/tab>



O Problema

- Projetar software reutilizável é difícil porque deve-se procurar por:
 - Uma boa decomposição do problema e a abstração correta.
 - O projeto deve ser **específico** para resolver o problema e **genérico** o suficiente para atender problemas e requisitos futuros
- Projetos frequentemente emergem de um processo iterativo de tentativas e erros.

O Problema

- A boa notícia é que bons projetos existem:
 - na verdade, eles apresentam características recorrentes,
 - mas eles quase nunca são idênticos.
- Perspectiva de engenharia: os projetos podem ser descritos, codificados ou padronizados?
 - isto iria diminuir a fase de tentativa e erro.
 - softwares melhores seriam produzidos mais rapidamente.

PADRÕES DE PROJETO DE SOFTWARE ORIENTADO A OBJETOS

- Também conhecidos como
 - *Padrões de Projeto de Software OO*
 - ou simplesmente como *Padrões*.

PADRÕES DE PROJETO

- A idéia de padrões foi apresentada por Christopher Alexander em 1977 no contexto de **Arquitetura** (de prédios e cidades):

“Cada padrão descreve um **problema** que **ocorre repetidamente** de novo e de novo em nosso ambiente, e então descreve a parte central da solução para aquele problema de uma forma que você pode **usar** esta **solução** um **milhão** de vezes, sem nunca implementá-la duas vezes da mesma forma”.

Quantos padrões de projeto existem?

- Muitos. E muito mais são criados a cada dia.
- Padrões de projeto não são métodos, algoritmos ou componentes.
- Para criar um sistema, podem ser necessários vários padrões.
- Diferentes projetistas podem usar diferentes padrões para resolver o mesmo problema.

Qual a relação entre esses padrões?

- Um padrão pode levar a outro.
- Alguns padrões são similares e alternativos.
- Alguns padrões se ‘encaixam’ naturalmente.
- Padrões podem ser descobertos e documentados.

CATÁLOGO DE SOLUÇÕES

- Um padrão encerra o **conhecimento de uma pessoa muito experiente** em um determinado assunto.
- Esse conhecimento **pode ser transmitido** para outras pessoas menos experientes.
 - Outras ciências (p.ex. química) e engenharias possuem catálogos de soluções.
- Desde **1995**, o desenvolvimento de software passou a ter o seu **primeiro catálogo de soluções** para projeto de software: o livro GoF.

PADRÕES DE PROJETO

- Passamos a ter um **vocabulário comum** para conversar sobre projetos de software.
- Soluções que não tinham nome passam a ter nome.
- Ao invés de discutirmos um sistema em termos de pilhas, filas, árvores e listas ligadas, passamos a **falar de coisas de muito mais alto nível** como:
 - Fábricas
 - Fachadas
 - Observador
 - Estratégia, etc.

PADRÕES DE PROJETO

- A maioria dos autores eram entusiastas de Smalltalk, principalmente o Ralph Johnson.
- Mas acabaram baseando o **livro em C++** para que o impacto junto à comunidade fosse maior.
- O impacto do livro foi enorme, vendeu centenas de milhares de cópias.
- Existem livros destes padrões para Java também:
 - **Patterns in Java:** a catalog of reusable design patterns illustrated with UML. Mark Grand. 2002.

ELEMENTOS ESSENCIAIS DE UM PADRÃO

- Nome
- Problema
 - Quando aplicar o padrão, em que condições?
- Solução
 - Descrição abstrata de como usar os elementos disponíveis (classes e objetos) para solucionar o problema
- Consequências
 - Custos e benefícios de se aplicar o padrão
 - Impacto na flexibilidade, extensibilidade, portabilidade e eficiência do sistema

FORMATO DO PADRÃO GOF

1.Nome

- expressa a própria essência do padrão de forma sucinta

1.Objetivo / Intenção

2.Também conhecido como (AKA)

3.Motivação

- cenário mostrando o problema e a necessidade da solução

1.Aplicabilidade

- situações em que o padrão é aplicável

1.Estrutura

- representação gráfica da estrutura de classes do padrão

FORMATO DO PADRÃO GOF

7.Participantes

- classes e objetos que participam e suas responsabilidades

7.Colaborações

- como os participantes colaboram para exercer suas responsabilidades

7. Consequências

- custos e benefícios da utilização do padrão

7.Implementação

- quais são os detalhes importantes na hora de implementar o padrão
- aspectos específicos de cada linguagem

FORMATO DO PADRÃO GOF

11.Exemplo de Código

- no caso do GoF, em C++ (a maioria) ou Smalltalk

11.Usos Conhecidos

- exemplos de sistemas reais em que o padrão é utilizado

11.Padrões Relacionados

- quais outros padrões devem ser usados em conjunto com esse
- quais padrões são similares a este, quais são as diferenças

TIPOS DE PADRÕES DE PROJETO do GoF

- Padrões de Criação:
 - são utilizados para criar classes e objetos
 - ajudam a tornar um sistema independente de como seus objetos são criados
- Padrões Estruturais:
 - preocupam-se com a forma com que objetos e classes são compostos para formar estruturas mais complexas.
 - utilizam mecanismos de herança e composição.
- Padrões Comportamentais:
 - preocupam-se com algoritmos e a atribuição de responsabilidades entre objetos.

Os 23 Padrões do GoF

Criação

- ▶ Processo de criação de objetos
 - Abstract Factory
 - Builder
 - Factory Method
 - Prototype
 - Singleton

Os 23 Padrões do GoF

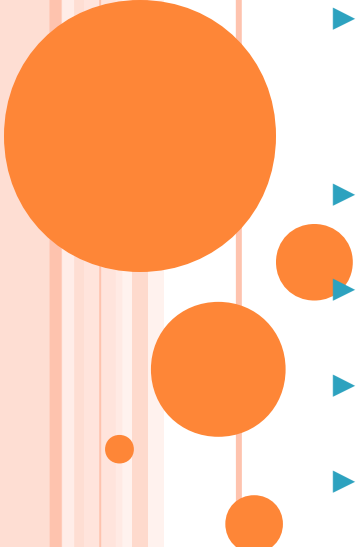
Estruturais

- ▶ Composição de classes ou objetos para formar estruturas complexas
 - Adapter
 - Bridge
 - Composite
 - Decorator
 - Façade
 - Flyweight
 - Proxy

Os 23 Padrões do GoF

Comportamentais

- Forma com que classes ou objetos interagem e distribuem responsabilidades.

- 
- ▶ Chain of Responsibility
 - ▶ Command
 - ▶ Interpreter
 - ▶ Iterator
 - ▶ Mediator
 - ▶ Memento

- ▶ Observer
- ▶ State
- ▶ Strategy
- ▶ Template Method
- ▶ Visitor

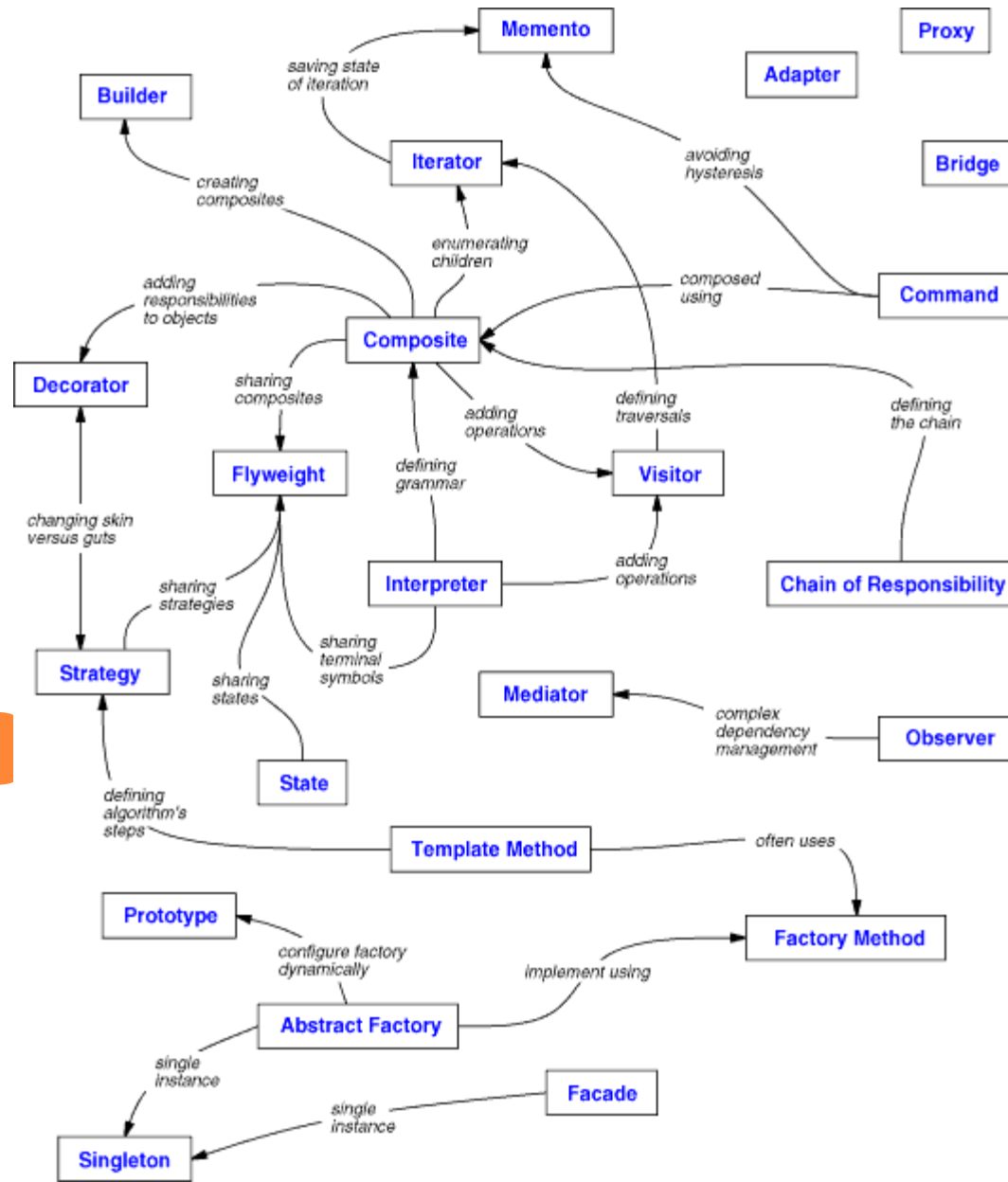
Organizando o catálogo

Escopo		Propósito		
		Criação	Estrutural	Comportamental
	Classe	Factory Method (121)	Adapter (157)	Interpreter (274) Template Method (360)
	Objeto	Abstract Factory (99) Builder (110) Prototype (133) Singleton (144)	Adapter (157) Bridge (171) Composite (183) Decorator (196) Facade (208) Flyweight (218) Proxy (233)	Chain of Responsibility (251) Command (263) Iterator (289) Mediator (305) Memento (316) Observer (326) State (338) Strategy (349) Visitor (366)

- Escopo 'Classe' negociam relacionamentos entre classes e sub-classes. Essas relações são estabelecidas através de herança. Então são estáticas – fixas em tempo de compilação.

- Escopo 'Objeto' negociam com relações de objetos, que podem mudar em tempo de execução.

Relação entre padrões





PADRÃO SINGLETON

PADRÃO SINGLETON

- Padrão de Criação.
- **Objetivo / Intenção**
 - Garantir que uma classe tenha apenas uma única instância, independente do número de requisições que receber para criá-la.
 - Fornecer um ponto global de acesso para essa instância.
 - A instância é criada somente no momento da sua primeira requisição (*lazy instantiation*).

PADRÃO SINGLETON

○ Motivação:

- Em alguns sistemas, é necessário ter apenas uma instância de uma determinada classe e um ponto de acesso global é necessário.
- Por exemplo:
 - Um único banco de dados
 - Um único acesso a um arquivo de log
 - Uma única configuração de jogo
 - Uma única central de controle das eleições.

PADRÃO SINGLETON

○ Aplicabilidade

- Este padrão é utilizado quando é necessário haver apenas uma instância de uma classe e essa instância tiver que dar acesso aos clientes através de um ponto bem conhecido.
- A única instância pode ser extensível através de subclasses, possibilitando aos clientes usar uma instância estendida sem alterar o seu código.

() PADRÃO SINGLETON: Discussão

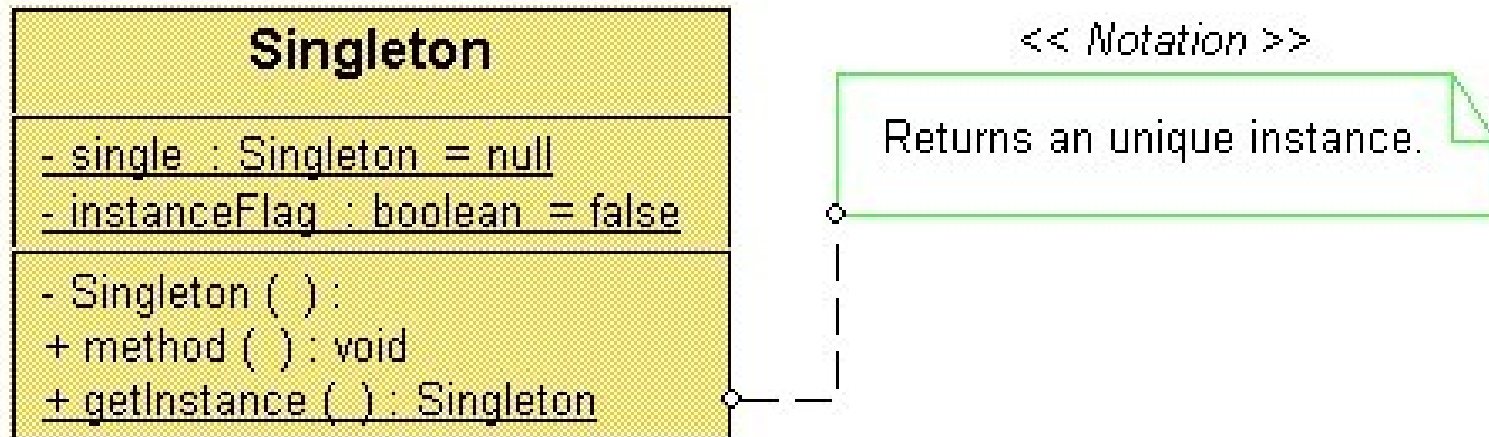
- atributo estático
- método estático

()PADRÃO SINGLETON: Discussão

- Deve-se fazer a própria classe Singleton ser responsável pela criação, inicialização e acesso a sua única instância.
- Deve-se declarar a única instância como um atributo *private static* da classe Singleton.
- A classe Singleton deve fornecer um método *public static* que encapsula todo o código de inicialização e provê acesso à instância.

PADRÃO SINGLETON

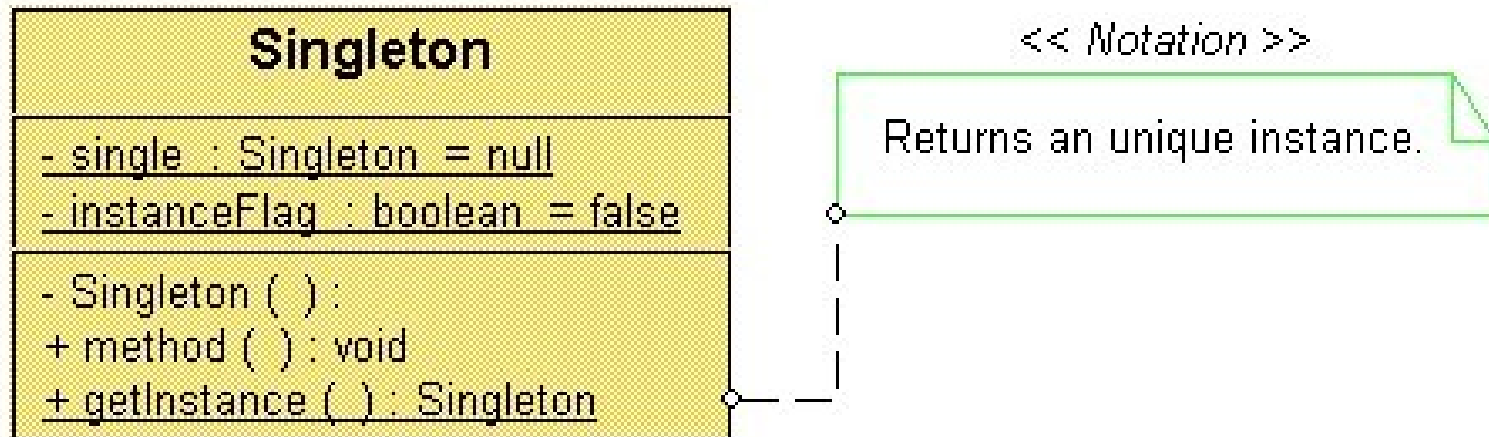
○ Estrutura



- O **construtor** do Singleton deve ser **privado**, para evitar que outras classes possam instanciar a classe diretamente.

PADRÃO SINGLETON

○ Estrutura

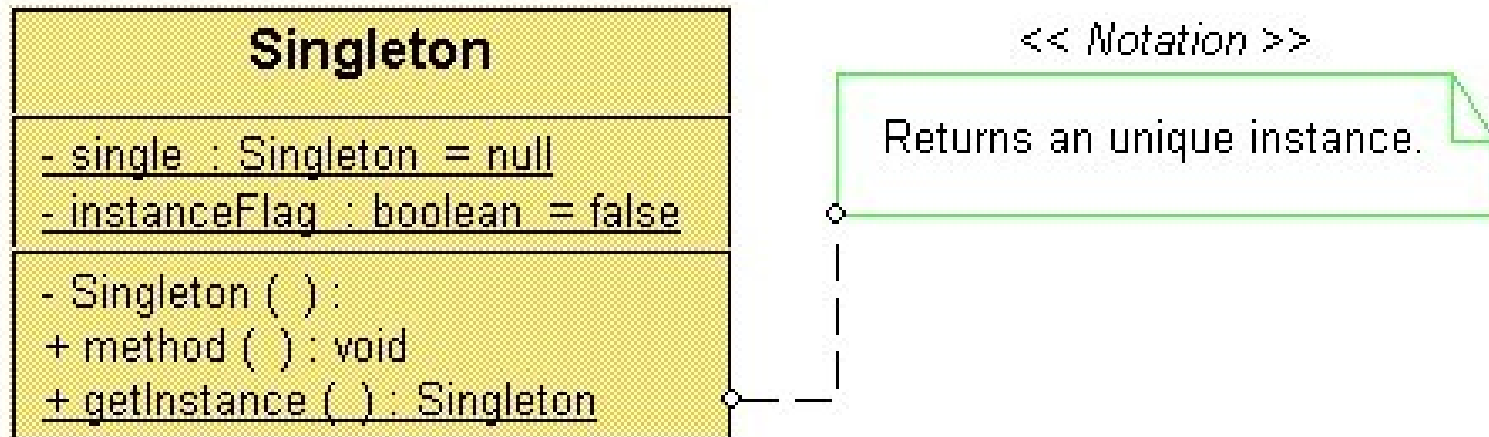


- O **construtor** do Singleton deve ser **privado**, para evitar que outras classes possam instanciar a classe

Para permitir herança o construtor pode ser protegido, mas é preciso ter cuidado com outras classes no mesmo pacote

PADRÃO SINGLETON

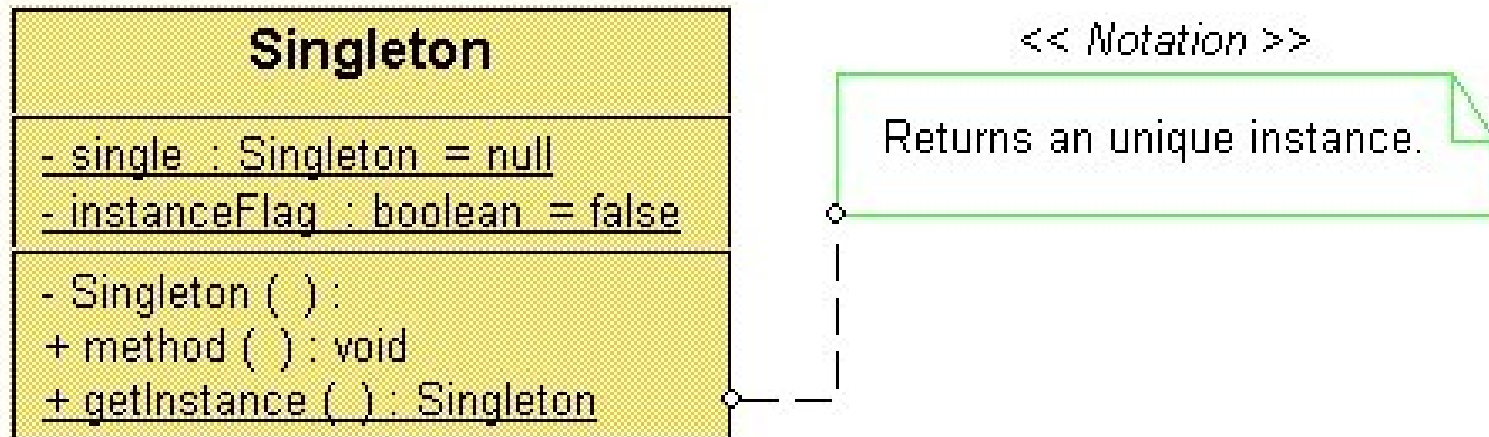
○ Estrutura



- single, instanceFlag e getInstance são **estáticos**.

PADRÃO SINGLETON

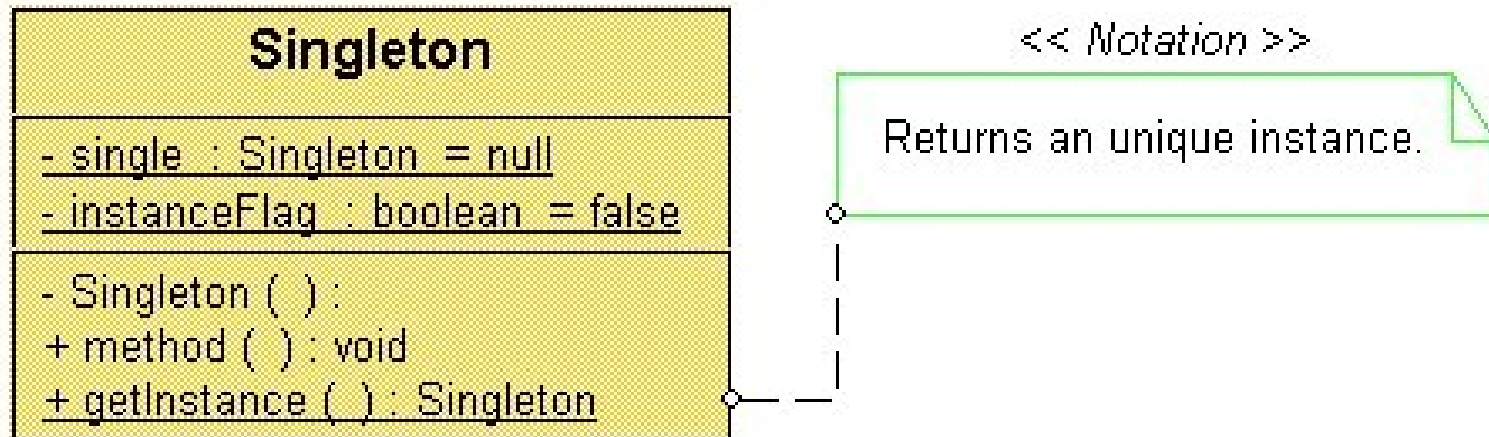
◦ Estrutura



- A única **instância** de Singleton é um **atributo privado**.
- O **método** de acesso à instância (*getInstance*) é um método **público**.

PADRÃO SINGLETON

○ Estrutura



- A classe Cliente chama o método *getInstance* para acessar à única instância de Singleton.

PADRÃO SINGLETON

○ Participantes

● Singleton:

- Atributo estático `instance`.
- Método estático `getInstance` que verifica se o atributo `instance` já foi instanciado.
- Contém todos os outros métodos e atributos não estáticos da instância única.

● Cliente:

- `Acessa a instância` única sempre através do método `getInstance`.

PADRÃO SINGLETON

○ Consequências

- Acesso controlado à instância única.
- Permite um número variável de instâncias: o padrão pode ser facilmente mudado para permitir mais de uma instância da classe Singleton.

Podemos mudar apenas o método `getInstance()`. Assim, podemos criar um vetor de instâncias e a cada chamada devolver a instância mais “livre”.

Ex:

- Entregar a impressora com menos documentos na fila de impressão.

PADRÃO SINGLETON

Implementação (a mais simples)

```
public class Singleton {  
    private static Singleton instance;// instancia unica  
    private Singleton() {}  
    public static Singleton getInstance() {  
        // instanciação tardia  
        if (instance==null){  
            instance = new Singleton();  
        }  
        return instance;  
    }  
}
```

Lazy instantiation: o programa só inicia o recurso quando este for necessário

- controla o uso de recursos.
- economiza espaço de memória.

PADRÃO SINGLETON

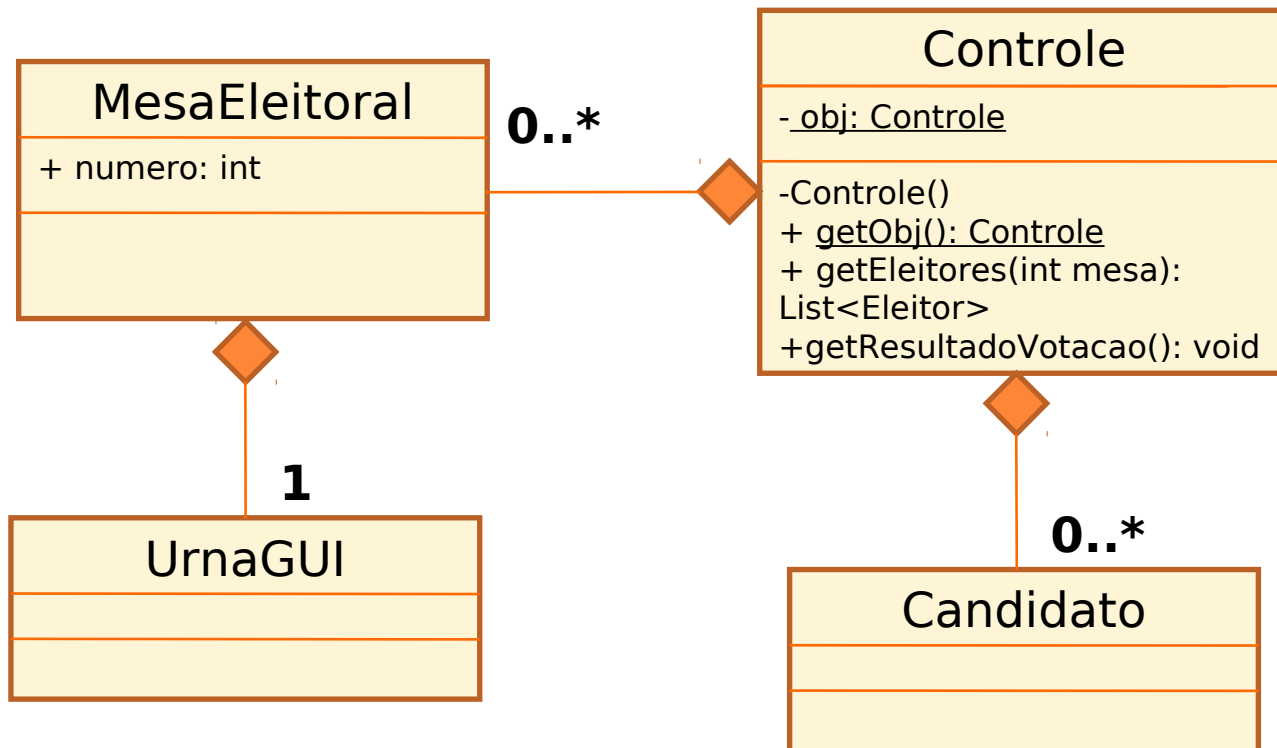
- Padrões relacionados
 - Abstract Factory
 - Builder
 - Prototype

PADRÃO SINGLETON: Noção intuitiva

- O cargo de Presidente do Brasil é um Singleton.
- Deve haver apenas um presidente ativo por vez.
- Não importa qual seja a identidade do presidente ativo, o título “Presidente do Brasil” é um ponto de acesso global que recupera a pessoa que está nesse cargo.
 - Fonte: http://sourcemaking.com/design_patterns/singleton

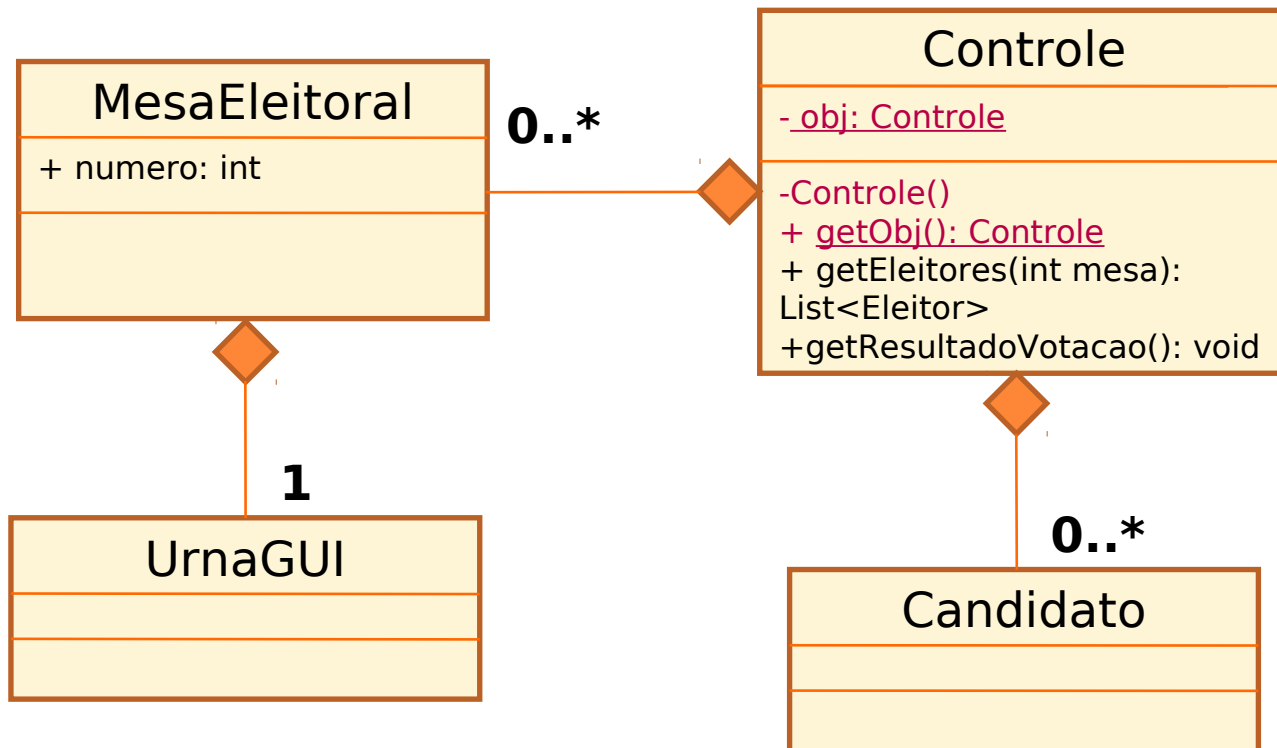
PADRÃO SINGLETON

- foi usado o padrão singleton?



PADRÃO SINGLETON

- foi usado o padrão singleton?



Aqui foi utilizada a instanciação tardia? Por que?

```
public class Highlander {  
    private Highlander() {}  
    private static Highlander instancia = new Highlander();  
    public static synchronized Highlander obterInstancia() {  
        return instancia;  
    }  
}
```

Esta classe implementa o design pattern Singleton

```
public class Fabrica {  
    public static void main(String[] args) {  
        Highlander h1, h2, h3;  
        //h1 = new Highlander(); // nao compila!  
        h2 = Highlander.obterInstancia();  
        h3 = Highlander.obterInstancia();  
        if (h2 == h3) {  
            System.out.println("h2 e h3 são mesmo objeto!");  
        }  
    }  
}
```

Esta classe cria apenas um objeto Highlander

PADRÃO SINGLETON - Check List

1. Defina um atributo *private static* na classe Singleton.
2. Defina um método de acesso *public static* na classe Singleton.
3. Utilize um esquema de inicialização tardia no método de acesso à instância única de Singleton.
4. Defina os construtores de Singleton como sendo *private* ou *protected*.
5. Clientes somente podem manipular a instância de Singleton por meio da sua função de acesso.

PADRÃO SINGLETON

○ Críticas:

- Um dos padrões mais utilizados na época do lançamento dos padrões de projeto pela simplicidade de implementação.
- Hoje é muito questionado por:
 - Muito semelhante a utilização de variáveis e métodos globais.
 - A centralização de responsabilidade pode ser visto como uma característica ruim.

<http://tech.puredanger.com/2007/07/03/pattern-hate-singleton/>

- É **importante conhecer o conceito** pois ainda aparece em muitas aplicações.