PADRÕES DE PROJETO
UMA BREVE INTRODUÇÃO

Prof. Dr. Dilermando Piva Jr.
Fatec Indaiatuba

VOCÊ SABE PINTAR?

CURSO DE TÉCNICAS PARA PINTURA...

- Imagine que uma pessoa tenha aprendido diversas **técnicas de pintura**. A partir desse conhecimento, ela saberá como pegar um pincel, como misturar as cores e como trabalhar com diferentes tipos de tinta.
- **Será que somente com esse conhecimento ela conseguirá pintar um quadro?**
- Note que ela sabe tudo o que se precisa para realizar a pintura, porém todo esse conhecimento não é válido se a pessoa não souber **como utilizá-lo**.
- Para realizar essa tarefa, é necessário, além de conhecimento, ter habilidade, que é algo que só se aprende com muita **prática e treino**.
- **Saber as técnicas é apenas o primeiro passo...**

... PROGRAMAÇÃO

- ... acontece um fenômeno similar.
- Quando se aprende uma linguagem orientada a objetos e seus recursos, isso é equivalente a se **aprender a pegar em um pincel.**
- Saber, por exemplo, como utilizar herança e polimorfismo não é o suficiente para distinguir em quais situações eles devem ser empregados de forma apropriada.
- É preciso conhecer os **problemas** que podem aparecer durante a modelagem de um sistema e saber quais as **soluções** que podem ser implementadas para equilibrar requisitos, muitas vezes contraditórios, com os quais se está lidando.

DESIGN PATTERN (PADRÃO DE PROJETO)

- Descrição de um problema recorrente, de forma **genérica**.
- Descrição de uma solução também **genérica**, que deve ser adaptada de acordo com o contexto em que o problema se manifesta.

PADRÕES DE PROJETO

ATENÇÃO!!

- PADRÕES NÃO SÃO NOVAS SOLUÇÕES
- SÃO SOLUÇÕES QUE FORAM IMPLEMENTADAS COM SUCESSO DE FORMA RECORRENTE EM DIFERENTES CONTEXTOS

OBJETIVOS DE PROJETO

- **Reusabilidade, flexibilidade e manutenibilidade**
  - reutilize projetos flexíveis
  - mantenha o código em um nível geral
  - minimize dependência de outras classes
- **Robustez**
  - reutilize projetos seguros
  - reutilize partes robustas
- **Suficiência e correção**
  - modularize o projeto
  - reutilize partes confiáveis

VANTAGENS NO USO DE PP

- Evita a redescoberta de soluções
- Propicia o uso de soluções corretas
- Melhora a qualidade do software
- Melhora a confiabilidade do software
- Provê uma linguagem comum entre desenvolvedores
- Reduz o volume de documentação
- Economiza esforço e tempo de desenvolvimento e manutenção
- Conduz ao bom uso de orientação a objetos

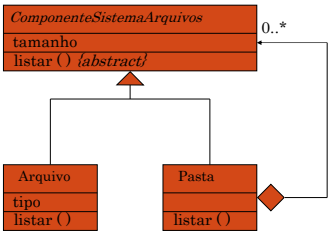
O MINISTÉRIO DA INFORMÁTICA ADVERTE!

O USO DE PADRÕES DE PROJETO PODE **AUMENTAR A COMPLEXIDADE** DOS SISTEMAS!

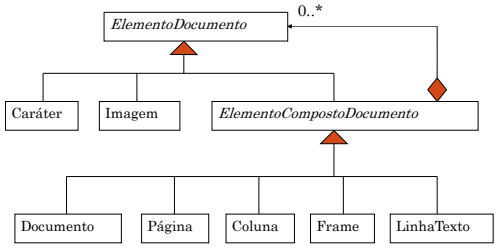
EXEMPLO DE PROBLEMA RECORRENTE : **COMPOSIÇÃO**

- Em um sistema de arquivos, existem arquivos e pastas (diretórios), sendo que todo arquivo está contido em uma pasta e toda pasta pode conter arquivos e também outras pastas.
- Em um documento, existem caracteres e imagens como **elementos básicos**, e páginas, colunas, frames e linhas de texto como **elementos compostos**, sendo que todo elemento básico está contido em um elemento composto e todo elemento composto pode conter elementos básicos e também outros elementos compostos.

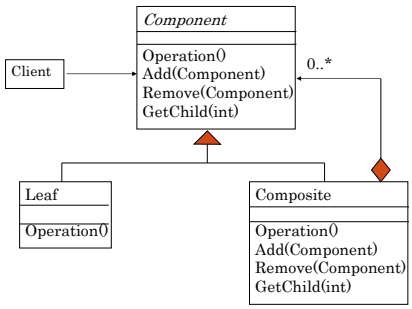
COMPOSIÇÃO EM SISTEMA DE ARQUIVOS



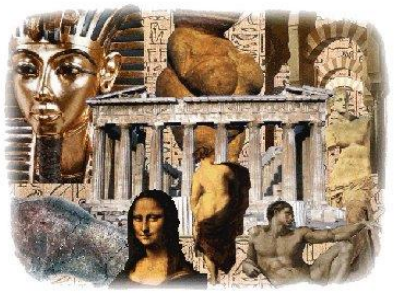
COMPOSIÇÃO EM DOCUMENTO



COMPOSIÇÃO: SOLUÇÃO GENÉRICA (GOF)



UM POUCO DE HISTÓRIA...



A INSPIRAÇÃO

- A ideia 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 implementa-la duas vezes da mesma forma. *(A Pattern Language, 1977)*

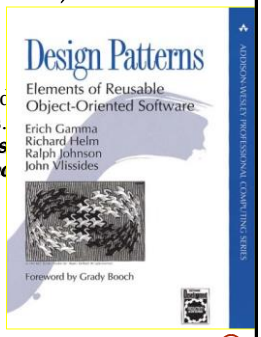
- Livros
  - *The Timeless Way of Building*
  - *A Pattern Language: Towns, Buildings, and Construction*
  - serviram de inspiração para os desenvolvedores de software.

CATÁLOGO DE SOLUÇÕES

- Um padrão encerra o conhecimento de uma pessoa muito experiente em um determinado assunto de uma forma que este 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.**

GANG OF FOUR (GOF)

- E. Gamma and R. Helm and R. Johnson and J. Vlissides. ***Design Patterns - Elements of Reusable Object-Oriented Software***. Addison-Wesley, 1995.



GANG OF FOUR (GOF)

"Descrição de uma solução customizada para resolver um problema genérico de projeto em um contexto específico. [...] Um padrão de projeto dá nome, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum para torná-la reutilizável."

Erich Gamma, et. al

Design Patterns, Elements of Reusable Object-Oriented Software (1994)

GANG OF FOUR (GOF)

- 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**.
- 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 de CC fosse maior. E o impacto foi enorme, o livro vendeu centenas de milhares de cópias.

O FORMATO DE UM PADRÃO

- **Todo padrão inclui**
  - Nome
  - Descrição do problema e contexto para os quais o padrão se aplica
  - Descrição da solução genérica proposta
  - Consequências da aplicação do padrão (custos / benefícios)
- Existem outros tipos de padrões ... Padrões J2EE, GRASP (General Responsability Assignment Software Patters) MVC (model – view – controller), Injeção de Dependências (IoC) entre outros
- vamos nos concentrar no GoF.

O FORMATO DOS PADRÕES NO GOF

- **Nome** (inclui número da página)
  - um bom nome é essencial para que o padrão caia na boca do povo
- **Objetivo / Intenção / Motivação**
  - um cenário mostrando o problema e a necessidade da solução
- **Aplicabilidade**
  - como reconhecer as situações nas quais o padrão é aplicável
- **Estrutura**
  - uma representação gráfica da estrutura de classes do padrão (usando OMT91) em, às vezes, diagramas de interação (Booch 94)
- **Participantes**
  - as classes e objetos que participam e quais são suas responsabilidades
- **Colaborações**
  - como os participantes colaboram para exercer as suas responsabilidades

O FORMATO DOS PADRÕES NO GOF

- **Consequências**
  - vantagens e desvantagens, *trade-offs*
- **Implementação**
  - com quais detalhes devemos nos preocupar quando implementamos o padrão
  - aspectos específicos de cada linguagem
- **Exemplo de Código**
  - no caso do GoF, em C++ (a maioria) ou Smalltalk
- **Usos Conhecidos**
  - exemplos de sistemas reais de domínios diferentes onde o padrão é utilizado
- **Padrões Relacionados**
  - quais outros padrões devem ser usados em conjunto com esse quais padrões são similares a este, quais são as diferenças

CRÍTICAS...

- Segundo alguns especialistas, alguns "padrões de projeto" são apenas evidências de que alguns recursos estão **ausentes em uma determinada linguagem** de programação (Java ou C++ por exemplo).
- Nesta linha, Peter Norvig demonstra que 16 dos 23 padrões do livro 'Design Patterns' são simplificados ou eliminados nas linguagens Lisp ou Dylan, usando os recursos diretos destas linguagens
- Segundo outros, excessos nas tentativas de fazer o código se conformar aos 'Padrões de Projeto' aumentam desnecessariamente a sua complexidade

CATEGORIAS DE PADRÕES (GOF)

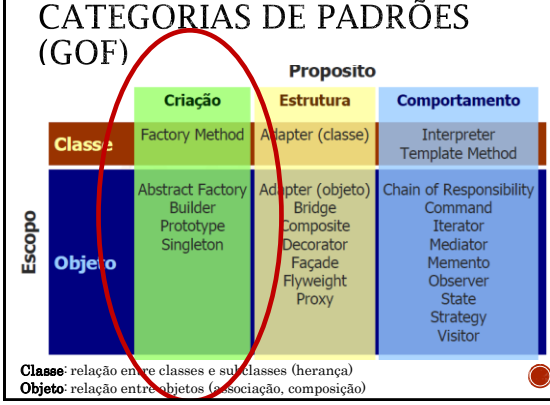
- CRIACIONAIS: criar uma coleção de objetos de maneira flexível ou restrita.
- ESTRUTURAIS: representar uma coleção de objetos relacionados (composição).
- COMPORTAMENTAIS: captar comportamento em uma coleção de objetos (interação e distribuição de responsabilidades).

CATEGORIAS DE PADRÕES (GOF)

Escopo \ Propósito	Criação	Estrutura	Comportamento
Classe	Factory Method	Adapter (classe)	Interpreter Template Method
Objeto	Abstract Factory Builder Prototype Singleton	Adapter (objeto) Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Classe: relação entre classes e subclasses (herança)
Objeto: relação entre objetos (associação, composição)

CATEGORIAS DE PADRÕES (GOF)

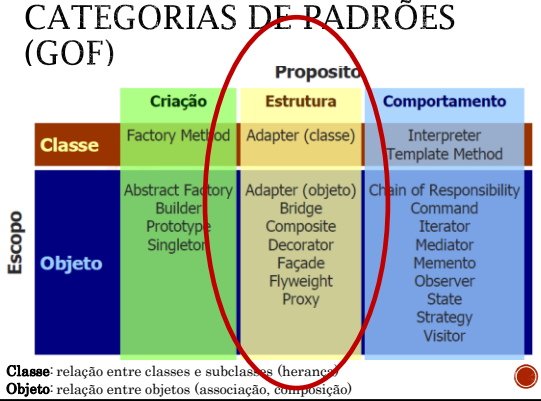


Classe: relação entre classes e subclasses (herança)
Objeto: relação entre objetos (associação, composição)

PADRÕES DE PROJETO CRIACIONAIS

- **Factory Method (Fábrica)**
  - Define uma interface para criação de objetos mas deixa a implementação para as subclasses
- **Abstract Factory (Fábrica Abstrata)**
  - Provê uma interface para a criação de famílias de objetos relacionados sem especificar suas classes concretas
- **Prototype (Protótipo)**
  - Define um protótipo dos objetos a serem usados e cria-os clonando este protótipo
- **Singleton (Objeto Unitário ou Único)**
  - Garante que uma classe possui somente uma instância e provê um ponto de acesso global a ela.
- **Builder (Construtor)**
  - Separa o processo de construção de objetos complexos, desacoplando a criação de instâncias destes objetos

CATEGORIAS DE PADRÕES (GOF)



Classe: relação entre classes e subclasses (herança)
Objeto: relação entre objetos (associação, composição)

PADRÕES DE PROJETO ESTRUTURAIS

- **Composite (Composto)**
  - Agrupa objetos de mesma interface em estrutura de árvore para tratar todos eles como se fossem uma única entidade
- **Adapter (Adaptador)**
  - Converte uma interface de uma classe em outra que objetos clientes esperam utilizar
- **Bridge (Ponte)**
  - Desacopla uma abstração de sua implementação para que ambas possam variar independentemente
- **Facade (Fachada)**
  - Provê uma interface única (ponto central) para diversas outras interfaces do sistema

PADRÕES DE PROJETO ESTRUTURAIS

- **Decorator (Decorador)**
  - Estende a funcionalidade de um objeto dinamicamente anexando objetos que irão executar antes ou depois do objeto "decorado"
- **Flyweight (Peso Mosca)**
  - Forma um pool de objetos imutáveis para serem utilizados em diversas partes do sistema
- **Proxy (Procurador)**
  - Provê um intermediário (procurador) que representa o objeto mas se comporta de forma diferente

CATEGORIAS DE PADRÕES (GOF)

Escopo \ Propósito	Criação	Estrutura	Comportamento
Classe	Factory Method	Adapter (classe)	Interpreter Template Method
Objeto	Abstract Factory Builder Prototype Singleton	Adapter (objeto) Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Classe: relação entre classes e subclasses (herança)
Objeto: relação entre objetos (associação, composição)

PADRÕES DE PROJETO COMPORTAMENTAIS

- **Interpreter (Interpretador)**
  - Criar uma linguagem de representação de operações e construir um interpretador para essa linguagem
- **Template Method (Método Modelo)**
  - Define o esqueleto do algorítmo da operação na superclasse, delegando partes do mesmo para as subclasses
- **Chain of Responsability (Cadeia de Responsabilidade)**
  - Monta uma corrente de objetos que serve a uma requisição, dando a cada um a oportunidade de respondê-la e/ou passá-la adiante.
- **Command (Comando)**
  - Encapsula requisições como objetos, permitindo parametrização, log ou *undo* de funções.

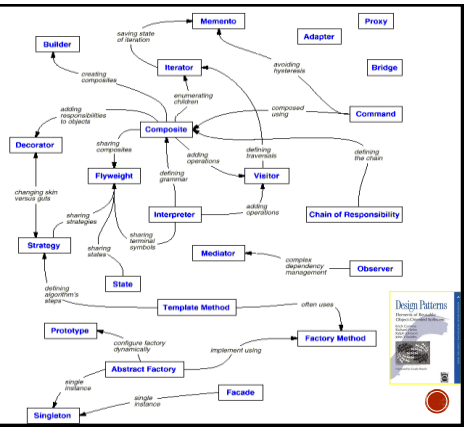
PADRÕES DE PROJETO COMPORTAMENTAIS

- **Iterator (Iterador)**
  - Acessa diversos elementos de um conjunto em sequência sem expor sua representação interna.
- **Mediator (Mediador)**
  - Delega a um objeto a responsabilidade de fazer outros objetos se comunicarem, tirando destes últimos o acoplamento entre si.
- **Memento (Recordação)**
  - Sem violar o encapsulamento, armazena o estado atual do objeto para que possa ser restaurado
- **Observer (Observador)**
  - Define uma dependência um-para-muitos entre objetos tal que quando um objeto muda de estado, todos são notificados.

PADRÕES DE PROJETO COMPORTAMENTAIS

- **State (Estado)**
  - Permite que um objeto altere seu comportamento quando mudar seu estado
- **Strategy (Estratégia)**
  - Define uma família de algoritmos, encapsula-os em objetos e permite que sejam intercambiados
- **Visitor (Visitante)**
  - Representa operações em objetos como outros objetos com uma interface comum, permitindo que sejam criadas novas operações sem alterar o objeto.

RELACIONAMENTO ENTRE OS PADRÕES



DETERMINANDO OS OBJETOS MAIS APROPRIADOS

- Decompor um sistema em objetos é difícil. Várias questões:
  - Encapsulamento, granularidade, dependências, flexibilidade, desempenho, evolução, reusabilidade etc.
- Padrões de projeto podem ajudar:
  - Identificação de abstrações menos óbvias de conceitos não presentes na natureza, como um algorítmo ou um estado.

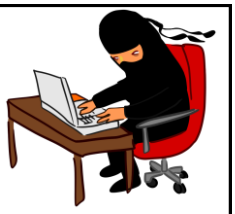
DETERMINANDO A GRANULARIDADE IDEAL

- Um objeto pode representar uma aplicação inteira ou um detalhe minúsculo. Como escolher?
- Padrões de projeto podem ajudar:
  - ***Facade (Fachada)*** explica como representar sistemas inteiros num só objeto;
  - ***Flyweight (Peso Mosca)*** descreve como utilizar muitos objetos de pequena granularidade;
  - ***Abstract Factory, Builder, Visitor*** e ***Command*** descrevem formas específicas de composição de objetos.

ESPECIFICANDO INTERFACES

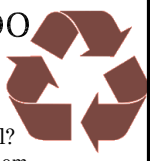
- Interfaces:
  - Conjunto de mensagens que um objeto aceita (contrato, assinaturas);
  - Objetos são conhecidos pelas interfaces, que nada dizem sobre a implementação.
- Padrões de projeto podem ajudar:
  - Identificam os elementos-chave de interfaces;
  - Restringem o que incluir em interfaces;
  - Estabelecem relacionamentos entre elas.

ESPECIFICANDO IMPLEMENTAÇÕES

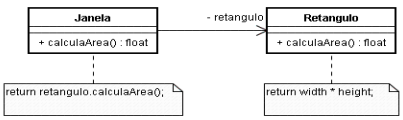


- Boa prática: "programe para interfaces, não para implementações";
- Padrões de projeto podem ajudar:
  - Abstraem o processo de criação de objetos, desacoplando os objetos clientes de implementações, permanecendo somente as interfaces.

TIRANDO PROVEITO DO REUSO



- Como construir software reutilizável?
  - "Dê preferência à composição de objetos em detrimento à herança de classes".
- Padrões de projeto podem ajudar:
  - Delegação dá mais poder à composição.



RELACIONANDO ESTRUTURAS EM TEMPO DE COMPILAÇÃO E EXECUÇÃO

- A estrutura *runtime* é bem diferente do código: dinamismo e interação;
  - Ex.: agregação e associação são codificadas da mesma forma, mas são bem diferentes em *runtime*.
- Padrões de projeto podem ajudar:
  - Se você entende o padrão, passa a entender a distinção, que está explícita na documentação do padrão.

PROJETANDO PARA MUDANÇAS MANUTENABILIDADE

- Sistemas devem ser projetados para facilitarem a manutenção:
  - Riscos de custos imprevistos;
  - Retrabalho.
- Padrões de projeto podem ajudar:
  - Promovem desacoplamento, permitindo maior liberdade dos objetos;
  - Código mais robusto, legível e de maior qualidade.

CAUSAS DE RETRABALHO E PADRÕES (GOF) QUE AS EVITAM

1. Criação de objeto especifica classe explicitamente:
   - O sistema está preso a uma implementação específica;
   - Solução: criar objetos indiretamente com Abstract Factory, Factory Method ou Prototype.
2. Dependência em operações específicas:
   - O sistema só tem uma forma de satisfazer a uma requisição;
   - Solução: evitar requisições "hard-coded" com Chain of Responsibility ou Command.

CAUSAS DE RETRABALHO E PADRÕES (GOF) QUE AS EVITAM

3. Dependência de plataforma:
   - O software utiliza recursos específicos de uma plataforma;
   - Solução: limitar dependências com Abstract Factory ou Bridge.
4. Dependência em representações ou implementações de objetos:
   - Clientes que sabem como um objeto é implementado, representado ou armazenado podem ter que ser alterados se o objeto mudar;
   - Solução: isolar os clientes com Abstract Factory, Bridge, Memento ou Proxy.

CAUSAS DE RETRABALHO E PADRÕES (GOF) QUE AS EVITAM

5. Dependência de algorítmo:
   - Objetos que dependem de algorítmos precisam mudar quando o algorítmo mudar;
   - Solução: isolar os algorítmos com Builder, Iterator, Strategy, Template Method ou Visitor.
6. Forte acoplamento:
   - Classes fortemente acopladas são difíceis de reusar, testar, manter, etc. Sistema monolítico;
   - Solução: enfraquecer o acoplamento com Abstract Factory, Bridge, Command, Chain of Responsibility, Façade, Mediator ou Observer.

CAUSAS DE RETRABALHO E PADRÕES (GOF) QUE AS EVITAM

7. Extensão de funcionalidade por meio de subclasse:
   - Herança é difícil de usar, composição dificulta a compreensão;
   - Solução: usar padrões que implementem bem herança, composição e delegação como Bridge, Chain of Responsibility, Composite, Decorator, Observer ou Strategy.
8. Incapacidade de alterar classes convenientemente:
   - Classes inacessíveis, imcompreensíveis ou difíceis de alterar;
   - Solução: usar Adapter, Decorator ou Visitor.

COMO SELECIONAR UM PADRÃO?

- Entenda como os padrões ajudam a resolver problemas em OO.
- Revise as intenções de cada padrão.
- Estude como os padrões se interrelacionam.
- Estude as similaridades entre os padrões de mesmo propósito.
- Conheça as principais causas de retrabalho.
- Considere o que você pode querer mudar em seu projeto no futuro.

COMO USAR UM PADRÃO

1. Leia o padrão todo uma vez.
2. Estude o código fonte de exemplo.
3. Escolha nomes para os participantes do padrão dentro do seu contexto.
4. Defina as novas classes e modifique classes existentes que são afetadas.
5. Defina nomes para as operações do padrão dentro do seu contexto.
6. Implemente as operações.

O MINISTÉRIO DA INFORMÁTICA ADVERTE!

CUIDADO COM OS **ANTI-PADRÕES** ou **ANTIPATTERNS**!

ANTI-PADRÕES

- Antipatterns são soluções que trazem mais complicação do que ben
- Pode advir de:
  - Uso incorreto de padrões de projeto
  - Criação de um novo padrão sem as devidas reflexões e testes
  - O que era bom ontem pode não ser mais hoje
- É necessário entendê-los para não cair em armadilhas e, se cair, saber se recuperar delas.

PADRÕES DE CRIAÇÃO

Exemplo

PADRÕES DE CRIAÇÃO

Abstraem o processo de instanciação, ajudando a tornar o sistema independente da maneira que os objetos são criados, compostos e representados"

Escopo de Classe	Factory Method
Escopo de Objeto	Abstract Factory
	Builder
	Prototype
	Singleton

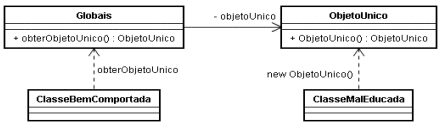
PADRÕES DE CRIAÇÃO

Abstraem o processo de instanciação, ajudando a tornar o sistema independente da maneira que os objetos são criados, compostos e representados"

Escopo de Classe	Factory Method
Escopo de Objeto	Abstract Factory
	Builder
	Prototype
	Singleton

SINGLETON (OBJETO ÚNICO)

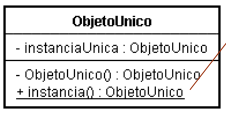
- Intenção:
  - Garantir que uma classe possui somente uma instância e prover um ponto de acesso global a ela



- Algumas vezes você precisa que uma classe só tenha uma instância para todo o sistema
- Prover um ponto de acesso static não é suficiente, pois classes poderão ainda construir outra instância diretamente.

SINGLETON (OBJETO ÚNICO)

- **Solução:**

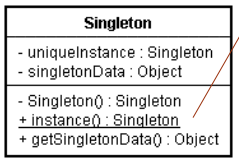



```
public static ObjetoUnico instancia() {
  if (instanciaUnica == null) {
    instanciaUnica = new ObjetoUnico();
  }
  return instanciaUnica;
}
```

- Há um ponto de acesso global (método static)
- Construtor privado ou protected? Depende se as subclasses devem ou não ter acesso
- Bloco de criação poderia ser synchronized para maior segurança em ambientes multithread
- A instância única poderia ser pré-construída.

SINGLETON (OBJETO ÚNICO)

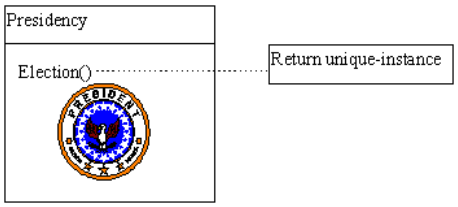
- **Estrutura:**




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

SINGLETON (OBJETO ÚNICO)

- **ANALOGIA:**

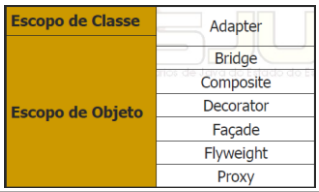


PADRÕES ESTRUTURAIS

Exemplo

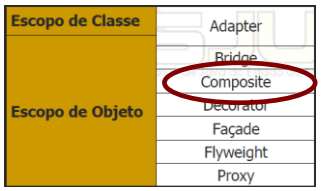
PADRÕES DE ESTRUTURA

Padrões de estrutura com escopo de classe usam herança para compor interfaces ou implementações. Os com escopo de objeto descrevem formas de compor objetos para realizar novas funcionalidades.



PADRÕES DE ESTRUTURA

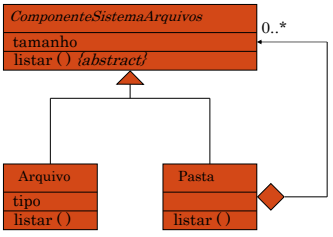
Padrões de estrutura com escopo de classe usam herança para compor interfaces ou implementações. Os com escopo de objeto descrevem formas de compor objetos para realizar novas funcionalidades.



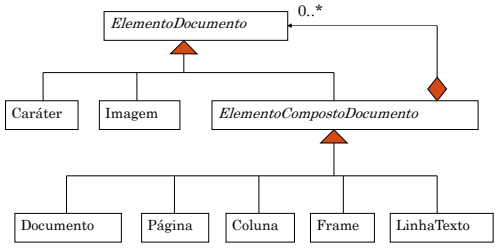
EXEMPLO DE PROBLEMA RECORRENTE : **COMPOSIÇÃO**

- Em um sistema de arquivos, existem arquivos e pastas (diretórios), sendo que todo arquivo está contido em uma pasta e toda pasta pode conter arquivos e também outras pastas.
- Em um documento, existem caracteres e imagens como **elementos básicos**, e páginas, colunas, frames e linhas de texto como **elementos compostos**, sendo que todo elemento básico está contido em um elemento composto e todo elemento composto pode conter elementos básicos e também outros elementos compostos.

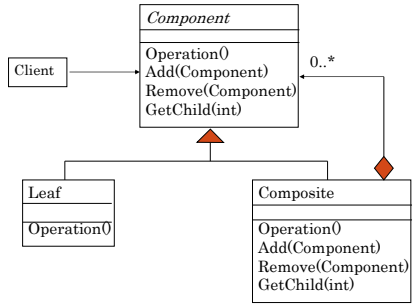
COMPOSIÇÃO EM SISTEMA DE ARQUIVOS



COMPOSIÇÃO EM DOCUMENTO



COMPOSIÇÃO: SOLUÇÃO GENÉRICA (GOF)



PADRÕES COMPORTAMENTAIS

Exemplo

PADRÕES DE COMPORTAMENTO

Preocupam-se com algoritmos e a delegação de responsabilidades entre objetos.

Escopo de Classe	Interpreter
	Template Method
Escopo de Objeto	Chain of Responsibility
	Command
	Iterator
	Mediator
	Memento
	Observer
	State
	Strategy
	Visitor

PADRÕES DE COMPORTAMENTO

Preocupam-se com algoritmos e a delegação de responsabilidades entre objetos.

Escopo de Classe	Interpreter
	Template Method
Escopo de Objeto	Chain of Responsibility
	Command
	Iterator
	Mediator
	Memento
	Observer
	State
	Strategy
	Visitor

OBSERVER (OBSERVADOR)

Vídeo....

Aniversário

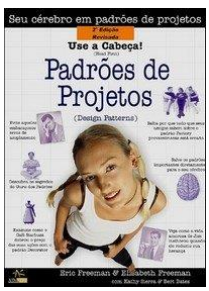
https://www.youtube.com/watch?v=iuavQrvKOO8

MUITO A ESTUDAR...

Dica:

Livro

Padrões de Projeto – Use a Cabeça



REFERÊNCIAS BIBLIOGRÁFICAS

- *Design Patterns: Elements of Reusable Object-Oriented Software*. Erich Gamma e outros (GoF), Addison-Wesley, 1995.
- *Patterns in Java: a catalog of reusable design patterns illustraded in UML*, Volume 1. Mark Grand, John Wiley & Sons, 1998.
- Projeto de Software: da programação à arquitetura *(Software design: from programming to architecture)*. Eric Braude, John Wiley & Sons, 2004.