Padrões e Desenho de Software

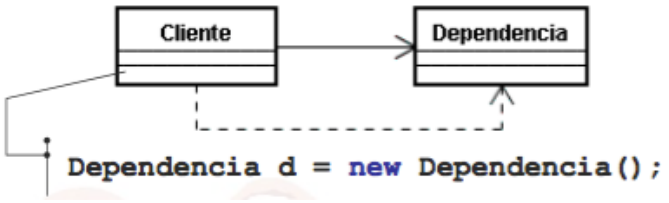
Resumos
2015/2016

João Alegria | 68661

Padrões de Criação

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

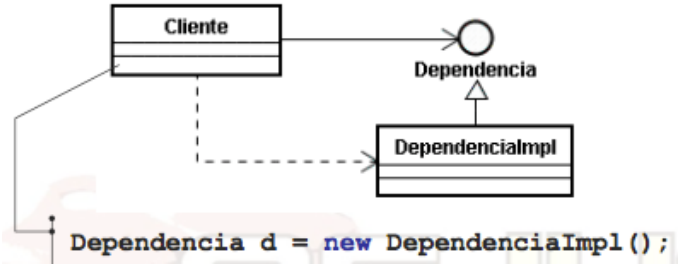
Introdução - Abordagem I



Solução inflexível:

- Cliente refere-se a uma implementação específica da sua dependência;
- Cliente constrói diretamente uma instância específica da sua dependência.

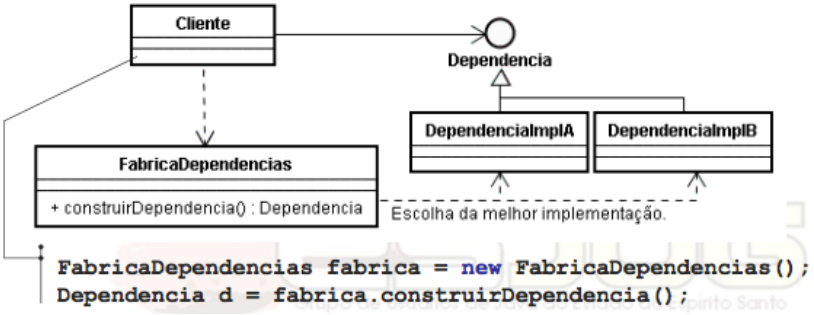
Introdução - Abordagem II



Alguma flexibilidade:

- Cliente já não mais associa-se a uma classe concreta;
- Porém, instancia a mesma diretamente.

Introdução - Abordagem III



Maior flexibilidade:

- A fábrica pode escolher a melhor classe concreta de acordo com um critério:

Maior flexibilização:

- Menor chance de ter que re-projetar;
- Menor desempenho;
- Maior dificuldade de realizar coisas simples;
- Maior dificuldade de compreender o código

Abstract Factory

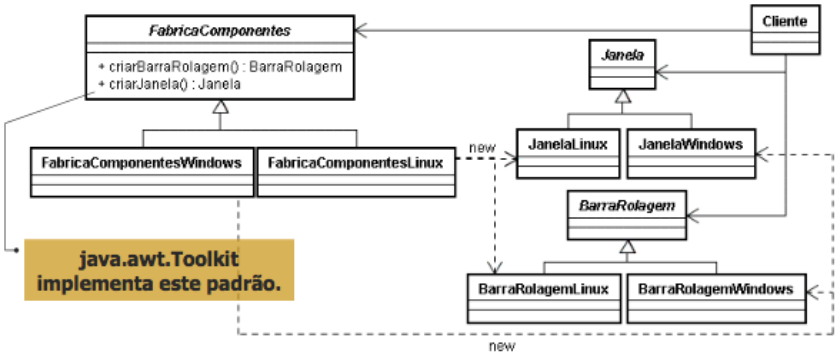
Intenção:

- Fornecer uma interface para criação de famílias de objetos relacionados sem especificar as suas classes concretas.

Problema:

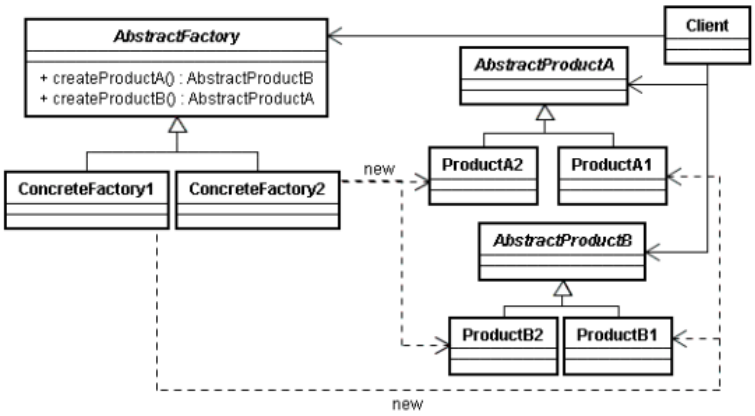
- Uma framework precisa de padronizar um modelo arquitetónico para uma gama de aplicações, mas permitir que aplicações individuais definam os seus próprios objetos e forneçam a sua instanciação.

Solução:



- Um método estático de uma classe retorna um objeto do tipo dessa classe;
- Uma das fábricas de componentes encarregar-se-á da criação dos objetos;
- Cliente não conhece as classes concretas.

Estrutura:



Usar este padrão quando:	Vantagens e Desvantagens:
- O sistema tiver que ser independente de como seus produtos são criados, compostos ou representados; - O sistema tiver que ser configurado com uma ou mais famílias de produtos (classes que devem ser usadas em conjunto); - Você quiser construir uma biblioteca de produtos e quiser revelar apenas suas interfaces e não suas implementações.	- Isola classes concretas: Fábricas cuidam da instanciação, o cliente só conhece interfaces; - Facilita a troca de famílias de classes: Basta trocar de fábrica concreta; - Promove consistência interna: Não dá para usar um produto de uma família com um de outra; - Criar novos produtos é trabalhoso: É necessário alterar as implementações de todas as fábricas para suportar o novo produto.

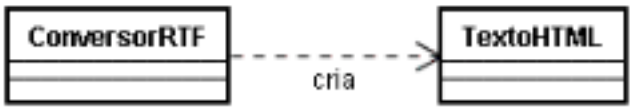
Builder

Intenção:

- Separa o processo de construção de um objeto complexo de sua representação para que o mesmo processo possa criar diferentes representações.

Problema:

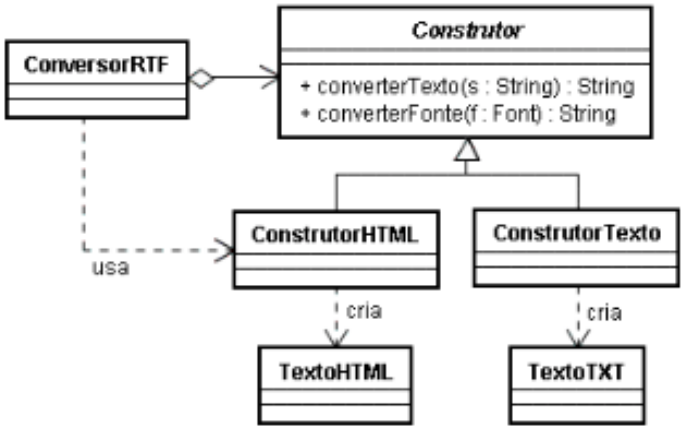
- Uma aplicação precisar criar os elementos de um complexo agregado. A especificação para o agregado existe no armazenamento secundário e uma das muitas representações precisa ser construído no armazenamento primário.



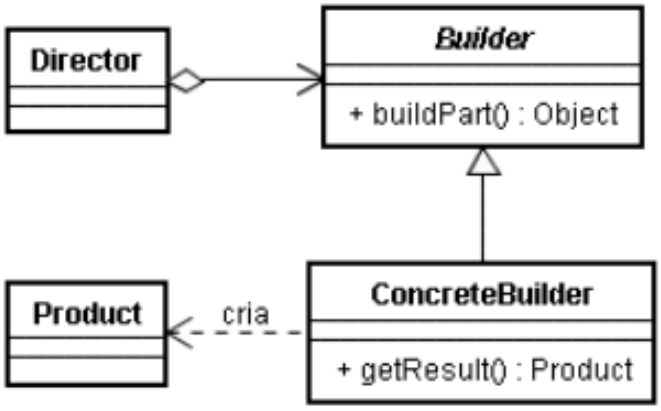
- Um objeto lê um texto em RTF e converte para HTML como produto final de um processamento

- Como fazer para preparar este sistema para uma eventual mudança de formato (TXT, por exemplo)?

Solução:



Estrutura:



- Para um sistema que mostre o resultado em TXT, basta trocar o construtor.

Usar este padrão quando:	Vantagens e Desvantagens:
- O algoritmo para criação de objetos complexos tiver que ser independente das partes que compõem o objeto e como elas são unidas; - O processo de construção tiver que permitir diferentes representações do objeto construído.	- Permite que varie a representação interna de um produto: Basta construir um novo builder - Separa o código da construção: melhora a modularidade, pois o cliente não precisa de saber da representação interna do produto. - Maior controlo do processo de construção: Constrói o produto passo a passo, permitindo o controlo de detalhes do processo de construção.

Factory Method

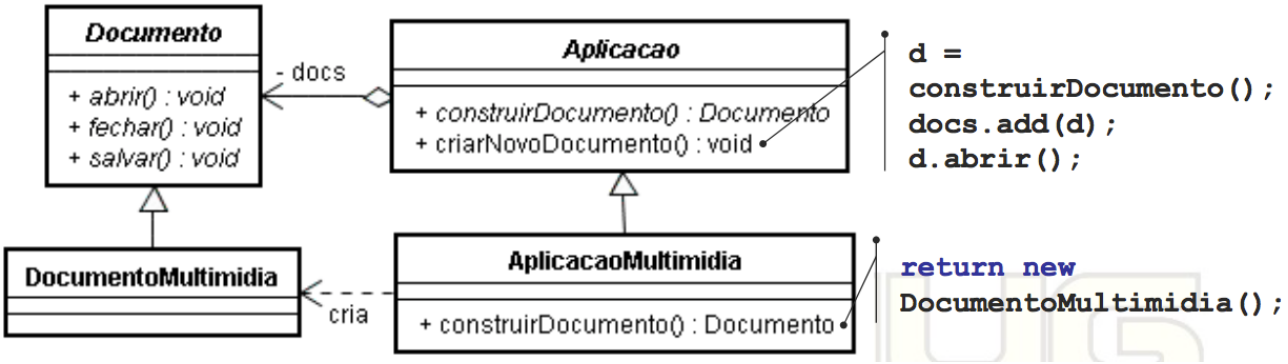
Intenção:

- Definir uma interface para a criação de objetos mas deixar as subclasses decidirem qual classe instanciar. Em outras palavras, delega a instanciação para as subclasses.

Problema:

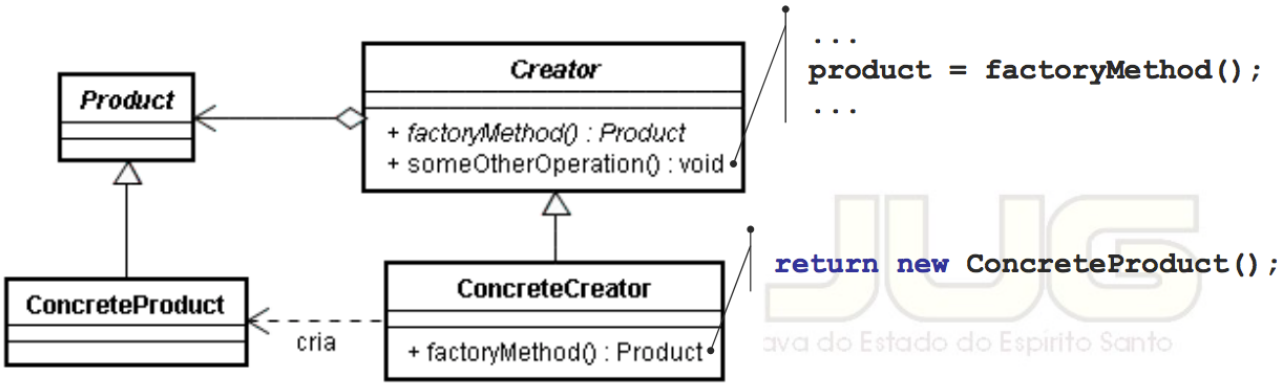
- Framework específico para uma aplicação que manipula documentos multimédia. É possível criar um framework mais genérico, para qualquer aplicação de manipulação de documentos?

Solução:



- Classes abstratas implementam as funções comuns a todo tipo de documento;
- Método fábrica é definido na superclasse e implementado na subclasse.

Estrutura:



Usar este padrão quando:	Vantagens e Desvantagens:
- Uma classe não tem como saber a classe dos objetos que precisará criar; - Uma classe quer que as suas subclasses especifiquem o objeto a ser criado.	- Melhor extensibilidade: Não é necessário saber a classe concreta do objeto para criá-lo. - Obrigatoriedade da subclasse fábrica: Não é possível criar somente um produto novo sem fábrica (excepto no caso do parametrizado)

Prototype

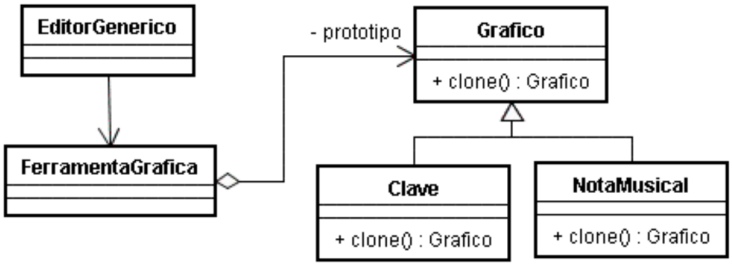
Intenção:

- Especificar o tipo de objeto a ser criado utilizando uma instância como protótipo e criar novos objetos copiando este protótipo.

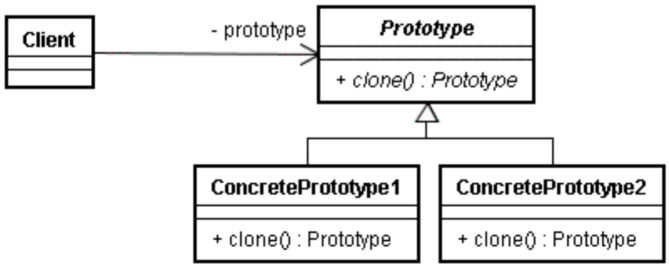
Problema:

- Existe um framework para edição de documentos genéricos. A ferramenta gráfica para um editor musical precisa de conhecer os gráficos específicos, o que anula o benefício conseguido anteriormente.

Solução:



Estrutura:



- A ferramenta gráfica é configurada com protótipos dos objetos gráficos.
- Quando precisar de criar algum deles, chama o método clone(), que retorna uma cópia de si mesmo.

Usar este padrão quando:	Vantagens e Desvantagens:
- O sistema deve ser independente de como seus produtos são criados, compostos e representados e… - As classes que devem ser ciadas são especificadas em tempo de execução; - Ou você não quer construir uma fábrica para cada hierarquia de produtos; - Ou as instâncias da classe clonável só tem alguns (poucos) estados possíveis, e é melhor clonar do que criar objetos	- Esconde a implementação do produto; - Permite adicionar e remover produtos em tempo de execução (configuração dinâmica da aplicação); - Não necessita de uma fábrica para cada hierarquia de objetos; - Implementar clone() pode ser complicado.

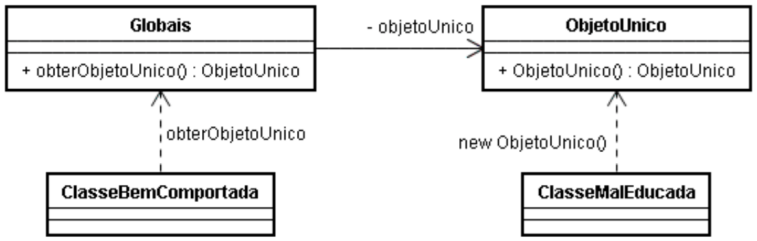
Singleton

Intenção:

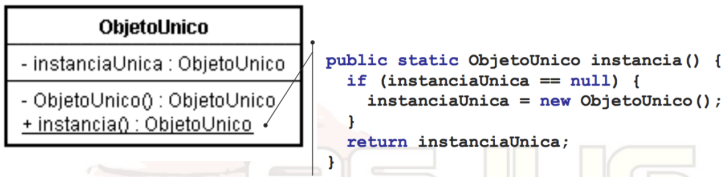
- Garantir que uma classe possui somente uma instância e fornecer um ponto de acesso global a ela.
- Algumas vezes precisa-se que uma classe só tenha uma instância para todo o sistema.
- Providenciar um ponto de acesso *static* não é suficiente, pois classes poderão ainda construir outra instância diretamente.

Problema:

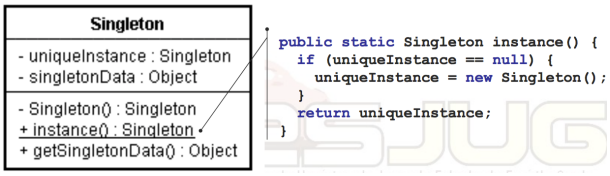
- A aplicação precisa de uma, e apenas uma, instância do objeto.
- Garantir que uma classe possui somente uma instância e fornecer um ponto de acesso global a ela.
- Algumas vezes precisa-se que uma classe só tenha uma instância para todo o sistema.
- Providenciar um ponto de acesso *static* não é suficiente, pois classes poderão ainda construir outra instância diretamente.



Solução:



Estrutura:



- Definir um ponto de acesso global (método static);
- Construtor privado ou protected, dependendo se as subclasses devem ter acesso;
- Bloco de criação poderia ser *synchronized* para maior seguranças em ambientes multithread;
- A instância única poderia ser pré-construída.

Usar este padrão quando:	Vantagens e Desvantagens:
- Tiver que haver exatamente uma instância de uma classe e ela tiver que estar acessível a todos num local bem definido; - Quiser permitir ainda que esta classe tenha subclasses (construtor protected).	- Acesso controlado à instância: A própria classe controla a sua instância única. - Não há necessidade de variáveis globais: Variável globais poluem o espaço dos nomes. - Permite extensão e refinamento: A classe Singleton pode ter subclasses. - Permite número variado de instâncias: Podemos controlar este número. - Mais flexível do que operações de classe: Usar membro static perde flexibilidade.

Object Pool

Intenção:

- Objet Pool pode oferecer um aumento de desempenho significativo. É mais eficaz em situações onde:
    - O custo de inicializar uma instância da classe é alta;
    - A taxa de instanciação de uma classe é alta;
    - O número de instâncias em uso a qualquer momento é baixa.

Problema:

- Os objetos são usados para gerir a cache do objeto. Um cliente com acesso a uma pool de objetos pode evitar a criação de novos objetos simplesmente perguntando à pool por um que já tenha sido instanciado.
- É desejável manter todos os objetos reutilizáveis que não estão atualmente em uso no mesmo pool de objetos para que eles possam ser geridos por uma política coerente.

Resumo:

- É utilizado para a reutilização de objetos e não para os eliminar.
- Verifica se o objeto está na pool e se estiver retorna-o em vez de criar um novo

RESUMIDAMENTE

Abstract Factory: Cria uma instância de várias famílias de classes;
Builder: Separa a construção do objeto da sua representação;
Factory Method: Cria uma instância de várias classes derivadas;
Prototype: Instância totalmente inicializada a serem copiados ou clonados;
Singleton: Uma classe do qual só um exemplo único pode existir
Object Pool: Evita a extensa aquisição e libertação de recursos através da reciclagem de objetos que não estão em uso.