Strategy

Introdução

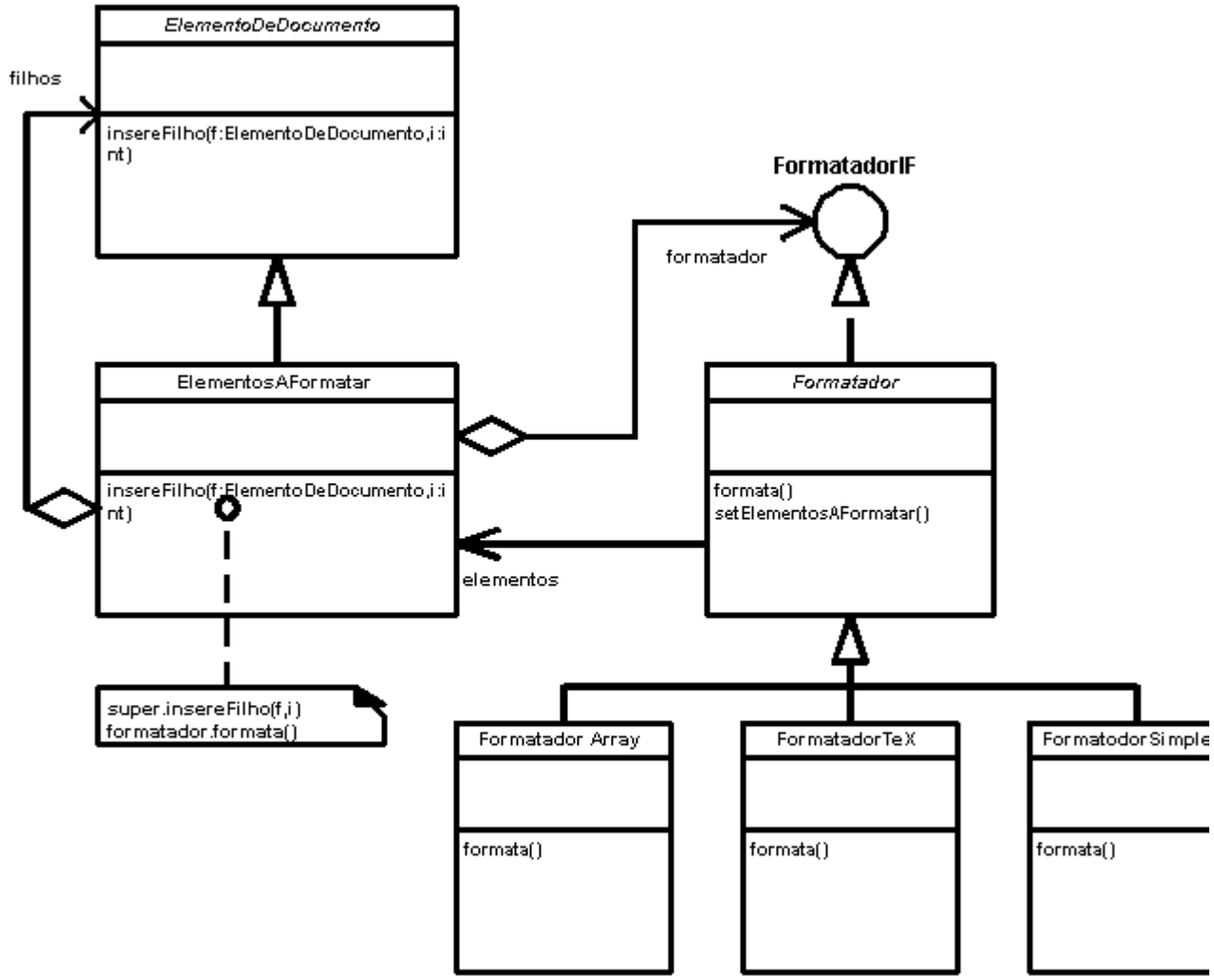
- Vamos considerar o problema de formatação no editor de documentos *WYSIWYG*
- Já sabemos como é a representação da informação
- Mas ainda não sabemos como gerar uma estrutura particular de linhas, colunas e objetos simples para um documento particular
  - Isso é resultado da formatação do documento
- Para simplificar, a palavra "formatação" significa apenas quebrar em linhas
  - O resto da formatação pode ser tratado de forma análoga
- Automatizar a formatação não é simples
  - Há um trade-off entre velocidade de formatação e a qualidade resultante
  - Muitas coisas devem ser consideradas tais como a "cor" de um documento (o espalhamento uniforme de espaços em branco, sem criar "rios")
  - Muitos algoritmos têm sido propostos e podem ser usados no editor
    - O algoritmo pode até mudar em tempo de execução
      - Considere a formatação em Word com "layout normal" (simples) e "layout da página" (mais demorado porém mais *WYSIWYG*)
- Ponto importante: queremos manter o algoritmo de formatação isolado da estrutura do documento
  - Podemos adicionar elementos gráficos sem afetar o algoritmo de formatação
  - Podemos mudar o algoritmo de formatação sem afetar o tratamento de elementos de documento
- Isolaremos o algoritmo de formatação através de sua encapsulação num objeto
- Usaremos uma hierarquia de classes para objetos que

encapsulam algoritmos de formatação

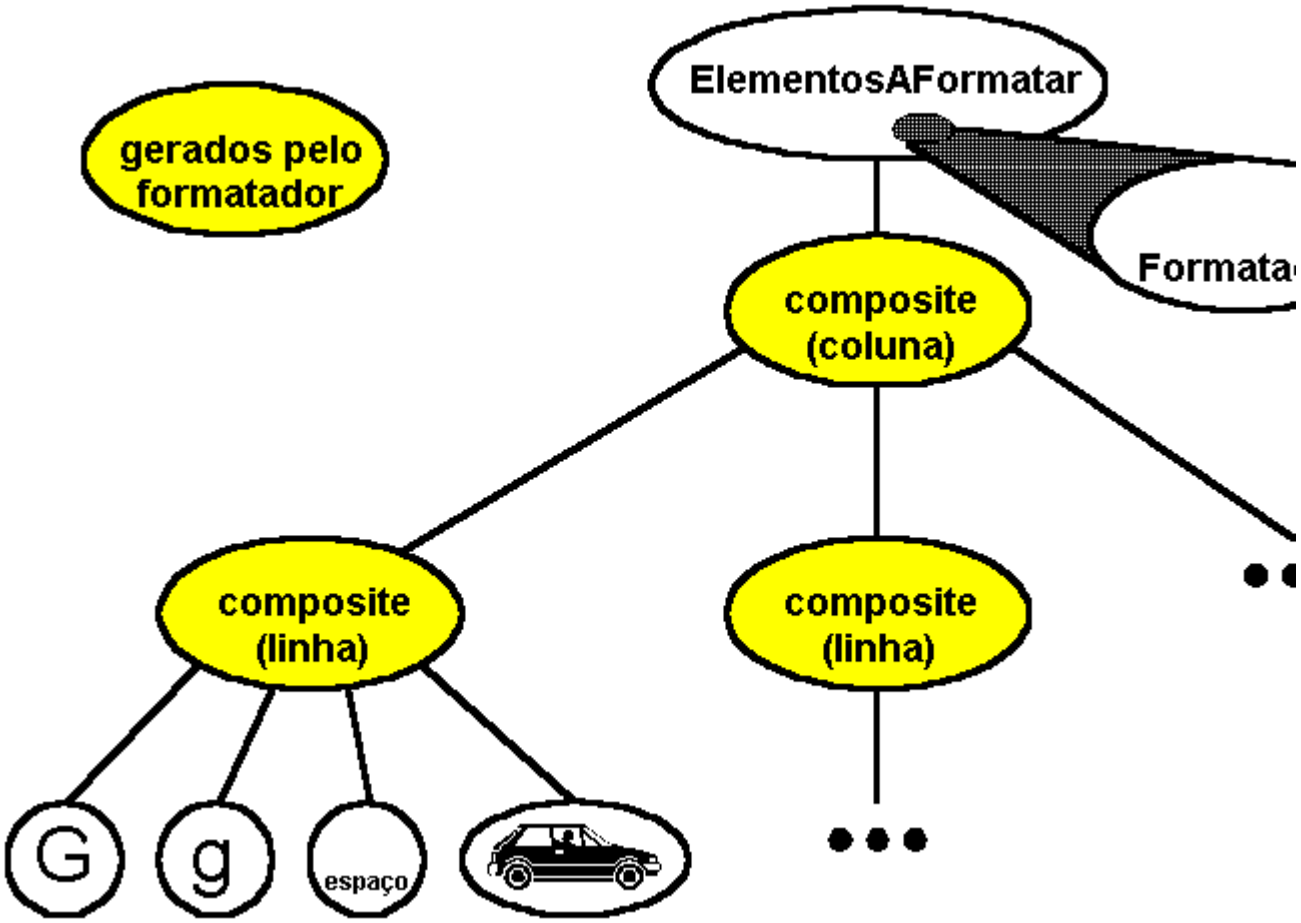
- A raiz da hierarquia será uma interface suficientemente genérica para suportar uma larga gama de algoritmos
- Cada subclasse implementa a interface para um algoritmo particular

As classes Formatador e ElementosAFormatar

- A classe Formatador é usada para objetos que encapsulam um algoritmo de formatação
- As subclasses de Formatador implementam algoritmos de formatação específicos
- Os elementos a formatar são filhos de uma subclasse especial de ElementoDeDocumento (ElementosAFormatar)
  - Tem um único objeto da classe ElementosAFormatar
  - ElementosAFormatar existe por dois motivos:
    - Para ser pai dos elementos básicos quando não tem formatação feita ainda
      - Já que o pai é normalmente um ElementoDeDocumentoIF, ElementosAFormatar também o é
    - Para manter referência ao formatador
- Quando um objeto ElementosAFormatar é criado, ele cria uma instância de uma subclasse de Formatador, especializada em formatar de alguma forma
- O objeto ElementosAFormatar pede ao formatador para formatar seus elementos quando necessário
  - Por exemplo, quando há uma mudança ao documento
- Ver a estrutura de classes abaixo



- Um objeto ElementosAFormatar não formatado contém apenas os elementos básicos visíveis (sem Composites linha, coluna, etc.)
- Quando os ElementosAFormatar precisam de formatação, o objeto chama o formatador
  - O formatador varre os filhos de ElementosAFormatar e insere os elementos linha, coluna, etc. de acordo com o algoritmo de formatação
  - Ver resultado dos objetos abaixo



- Formatadores diferentes implementam algoritmos diferentes
  - FormatadorSimples é o mais simples e rápido (não tem cuidados especiais com "cor") e trata apenas uma linha de cada vez
  - FormatadorTeX é mais complexo pois considera o parágrafo inteiro para distribuir os espaços em branco e evitar "rios"
  - FormatadorArray coloca um número igual de elementos em cada linha (para alinhar imagens, por exemplo)
- Resultado: a separação Formatador-ElementosAFormatar assegura uma separação forte entre o código que suporta a estrutura do documento e o código de formatação
  - Podemos adicionar novos formatadores sem tocar nas classes de elementos e vice-versa
  - Podemos até mudar de formatador em tempo de execução com uma função setFormatador na classe ElementosAFormatar

- O padrão que usamos chama-se Strategy

O padrão Strategy

Objetivo

- Definir uma família de algoritmos, encapsular cada um, e torná-los intercambiáveis
- Strategy permite mudar os algoritmos independentemente dos clientes que os usam

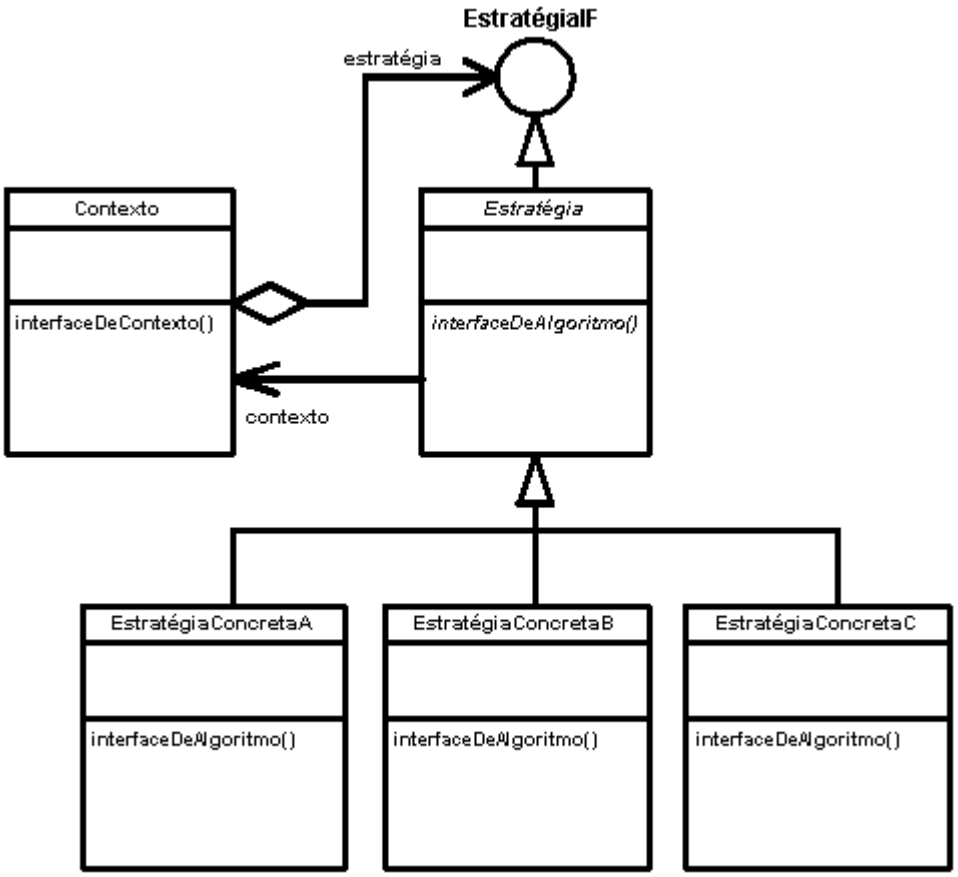
Também conhecido como

- Policy

Quando usar o padrão Strategy?

- Várias classes relacionadas têm apenas comportamento diferente
- Você precisa de variantes de um algoritmo
  - Exemplo: com trade-offs diferentes de espaço-tempo
- Para esconder dos clientes os dados complexos que um algoritmo usa
- Uma classe tem vários comportamentos escolhidos com muitas decisões
  - Em vez de usar decisões, mova os trechos apropriados para sua própria classe Strategy

Estrutura genérica



Participantes

- **EstratégiaIF** (exemplo: FormatadorIF)
  - Declara a interface comum a todos os algoritmos
  - O contexto usa essa interface para chamar o algoritmo definido em EstratégiaConcreta
- **Estratégia** (exemplo: Formatador)
  - Possível classe abstrata para fatorar código comum entre os algoritmos
- **EstratégiaConcreta** (exemplo: FormatadorSimples)
  - Implementa o algoritmo
- **Contexto** (exemplo: ElementosAFormatar)
  - É configurado com um objeto EstratégiaConcreta
  - Mantém uma referência para um objeto EstratégiaIF
  - Pode definir uma interface para que a estratégia acesse seus dados

Colaborações entre objetos

- O Contexto pode chamar a Estratégia passando a si mesmo para que a estratégia acesse os dados
- O Contexto encaminha pedidos dos seus clientes para a Estratégia
- Clientes normalmente criam uma EstratégiaConcreta e a passam para o contexto
  - A partir daí, os clientes interagem apenas com o Contexto

Conseqüências do uso do padrão Strategy

- Uma alternativa à herança
  - Poder-se-ia usar herança (subclasses de Contexto) para fazer a mesma coisa
    - Teríamos uma hierarquia de ElementosAFormatar, cada um com um *método* formatar() diferente
    - Em tempo de execução, instanciaríamos alguma subclasse de ElementoAFormatar
      - Qual das subclasses instanciar depende do algoritmo específico de formatação que se deseja
    - O problema: como mudar o algoritmo formatar() em tempo de execução??
      - Teria que fazer "mutação" do objeto ElementoAFormatar1 para um novo tipo ElementoAFormatar2
      - Mas a "mutação de tipo" é sinal de que o código fede!
        - As linguagens não dão suporte para isso
    - Solução melhor: retirar o método formatar e criar um objeto à parte só para este método
  - Portanto, isso seria um mau exemplo do uso de herança
    - Acopla o Contexto com os algoritmos (parafusa o

comportamento no Contexto)
    - Deixa o Contexto mais difícil de entender, manter e estender
    - O algoritmo não poderia variar dinamicamente
  - Exemplo clássico de herança versus composição
- Estratégias eliminam statements condicionais
  - Quando vários comportamentos estão agrupados na mesma classe
  - Código que tem muitas condições assim é candidato para o padrão Strategy
- Ponto negativo: transparência incompleta dos 2 objetos
  - Depois que o contexto é criado, os clientes enxergam apenas um objeto (o contexto) e não dois (contexto e estratégia)
  - Porém, no momento da criação do contexto, alguma estratégia deve ser escolhida
    - Os clientes devem *portanto conhecer* as estratégias antes de escolher uma
    - Isso expõe os clientes a considerações de implementação
    - Só usa Strategy quando a diferença de comportamento for relevante aos clientes
- Ponto negativo: mais objetos no projeto
  - O padrão flyweight mostra como lidar com isso

Exemplo na API Java

- Os Layout Managers de AWT são exemplos de classes que representam uma estratégia de layout para containers
- A EstratégiaIF é LayoutManager
  - Também tem LayoutManager2 que é uma extensão de LayoutManager
- Não tem classe abstrata Estratégia
- Tem muitas classes concretas (GridLayout, FlowLayout, BorderLayout, ...)
- O Contexto pode ser qualquer Container (Panel,

ScrollPane, Window, ...)

Perguntas finais para discussão

- O que ocorre quando um sistema tem uma explosão de objetos de estratégia? Tem uma forma melhor de gerenciar essas estratégias?
- Em Gamma, os autores descrevem duas formas pelas quais um objeto de estratégia pode obter a informação de que precisa para fazer seu trabalho. Uma forma descreve como um objeto de estratégia poderia receber uma referência a um objeto de contexto, permitindo assim o acesso aos dados de contexto. Mas não poderia ocorrer que os dados necessários à estratégia não estivessem disponíveis através da interface do objeto de contexto? Como remediar este possível problema?

Ver também

- http://www.javaworld.com/javaworld/jw-04-2002/jw-0426-designpatterns.html

programa