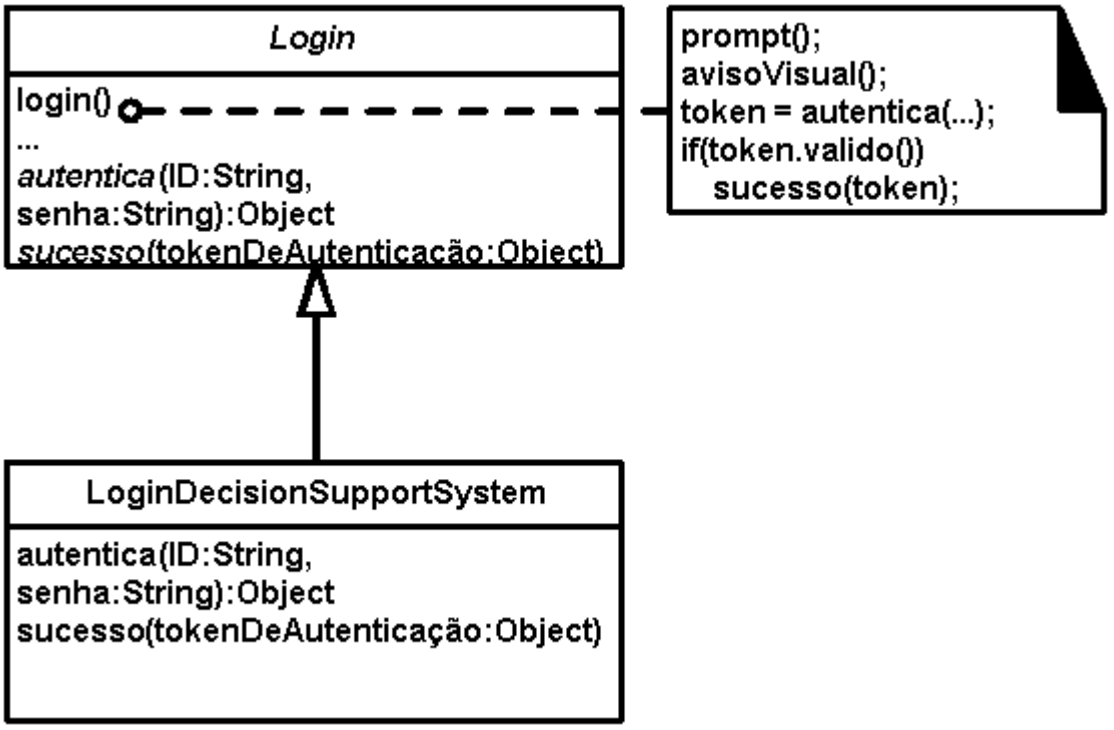
Template Method

Um problema a resolver: login

- Sua tarefa é de escrever uma classe para controlar o login de usuários numa aplicação
- Tem dois caminhos básicos para resolver o problema:
  - Escrever uma classe específica para a aplicação sob consideração
  - Escrever uma solução genérica para qualquer aplicação
    - Para promover o reuso
- A solução genérica seria um framework de login
  - Um esqueleto de login que seja comum a qualquer tarefa de login
  - Ganchos para permitir que cada aplicação *customize* o processo de login
- Quais são os passos genéricos de login?
  - **Prompt** para o usuário fornecer sua identificação (ID) e senha
  - **Autenticação** da ID e senha
    - O resultado da autenticação deve ser um objeto
    - Este objeto pode encapsular qualquer informação que possa ser usada adiante pela aplicação para comprovar a autenticação
  - **Aviso visual** de progresso indicando que a autenticação está sendo realizada
  - Um **aviso de sucesso** ao resto da aplicação que o login foi realizado e para disponibilizar o objeto produzido pela autenticação
- A lógica das etapas 1 e 3 não muda
- A lógica das etapas 2 e 4 pode variar muito entre aplicações
  - Cada aplicação deverá prover seu próprio código
- Solução
  - Criar uma classe abstrata Login contendo:

- Um método principal (chamado Template Method) que capture a lógica comum de login
- Métodos abstratos para representar as etapas do algoritmo que devem ser fornecidas pela aplicação particular

- Estender a classe Login para prover implementações dos métodos abstratos para uma aplicação particular (digamos para uma aplicação de DecisionSupportSystem)



- Chama-se o método login() de Template Method

O padrão Template Method

Objetivo

- Define o esqueleto de um algoritmo numa operação, deixando que subclasses completem algumas das etapas
- O padrão Template Method permite que subclasses redefinem determinadas etapas de um algoritmo sem alterar a estrutura do algoritmo

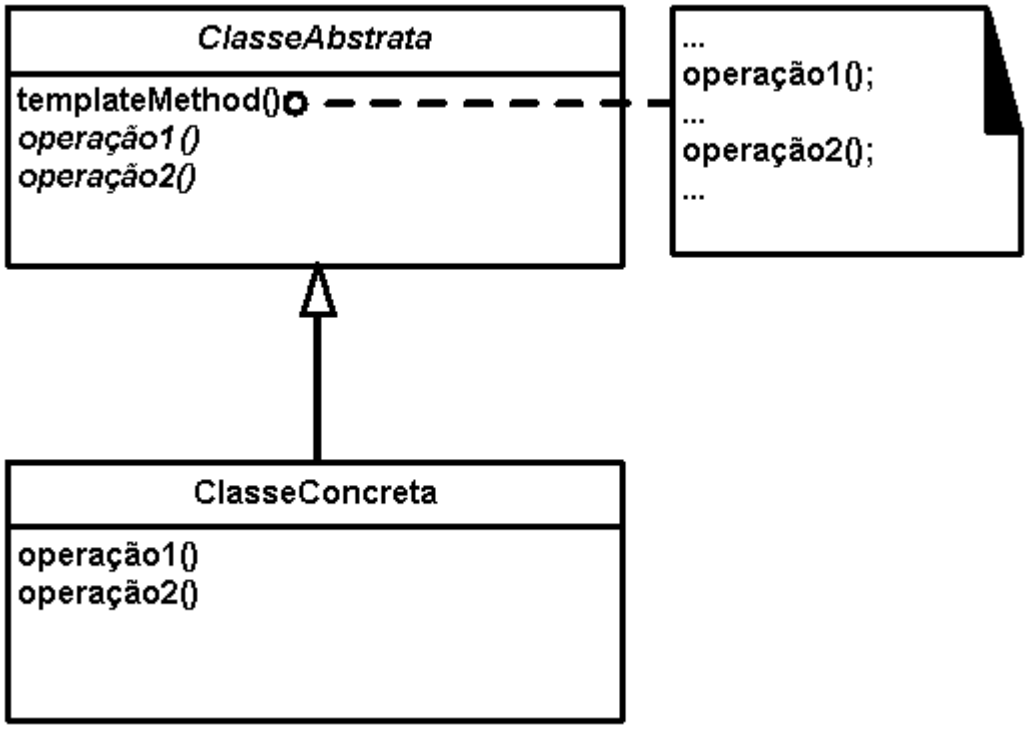
Resumo

- Um Template Method define um algoritmo usando operações abstratas
- Subclasses fazem override das operações para prover um comportamento concreto
- Este padrão é a base para a construção de *frameworks*

Quando usar o padrão Template Method?

- Para implementar partes invariantes de um algoritmo uma única vez e deixar subclasses implementarem o comportamento variável
- Quando comportamento comum entre subclasses deveria ser fatorado e localizado numa classe comum para evitar duplicação
  - É um passo freqüente de "refactoring" de código
    - Primeiro identifique as diferenças
    - Coloque as diferenças em novos métodos
    - Substitua o código das diferenças por uma chamada a um dos novos métodos
- Para controlar extensões de subclasses
  - Você pode definir um Template Method que chame operações-gancho (hook) e pontos específicos, permitindo extensões apenas nestes pontos
    - Faça com que apenas os métodos-gancho possam sofrer override, usando adjetivos de visibilidade
      - "public final" para o Template Method
      - "protected" para métodos que devem/podem sofrer override

Estrutura genérica



Participantes

- **ClasseAbstrata** (Login)
  - Define operações abstratas que subclasses concretas definem para implementar certas etapas do algoritmo
  - Implementa um Template Method definindo o esqueleto de um algoritmo
    - O Template Method chama várias operações, entre as quais as operações abstratas da classe
- **ClasseConcreta** (LoginDecisionSupportSystem)
  - Implementa as operações abstratas para desempenhar as etapas do algoritmo que tenham comportamento específico a esta subclasse

Colaborações

- ClasseConcreta depende de ClasseAbstrata para implementar as partes invariantes do algoritmo

Conseqüências do padrão Template Method

- Template Methods constituem uma das técnicas básicas de reuso de código
  - São particularmente importantes em frameworks e bibliotecas de classes para o fatoramento de comportamento comum
- Template Methods levam a uma inversão de controle
  - O código particular de uma aplicação é chamado pelo resto do código
  - Normalmente, escrevemos o código "de cima" e chamamos partes comuns "em baixo"
    - Aqui, é o contrário
  - Também chamado de "Hollywood Principle"
    - "Don't call us, we'll call you"
- O Template Method pode chamar vários tipos de operações
  - Operações concretas (da ClasseConcreta ou de outras classes)
  - Operações concretas de ClasseAbstrata (operações comuns úteis às subclasses)
  - Operações abstratas (onde o comportamento varia)
    - *Devem* sofrer override
  - Factory Methods
  - Operações-gancho
    - *Podem* sofrer override
    - Uma operação-gancho tem implementação nula (fazendo nada) na ClasseAbstrata
    - A subclasse pode fazer override para inserir algo neste ponto (daí, "gancho")
    - É uma forma limpa de estender o comportamento de uma classe de forma controlada
      - Isto é, apenas nos pontos onde há ganchos

Considerações de implementação

- É importante minimizar o número de operações

abstratas que devem sofrer override para completar o algoritmo
    - Motivo: simplificação para não chatear o programador
- Convenções de nome
    - Métodos abstratos que *devem* sofrer override deveriam ter algo de comum no nome
        - Exemplo: doXpto() // começa com "do"
    - Métodos-gancho que *podem* sofrer override deveriam ter algo de comum no nome
        - Exemplo: logHook() // termina com "Hook"

Pergunta final para discussão

- O Template Method funciona com herança. Seria possível obter a mesma funcionalidade da Template Method usando a composição de objetos? Quais seriam alguns dos trade-offs?

programa