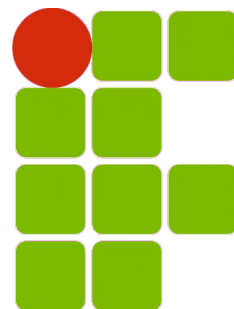


# INF011 – Padrões de Projeto

## 25 – *Template Method*

**Sandro Santos Andrade**  
sandroandrade@ifba.edu.br

**Instituto Federal de Educação, Ciência e Tecnologia da Bahia**  
**Departamento de Tecnologia Eletro-Eletrônica**  
**Graduação Tecnológica em Análise e Desenvolvimento de Sistemas**



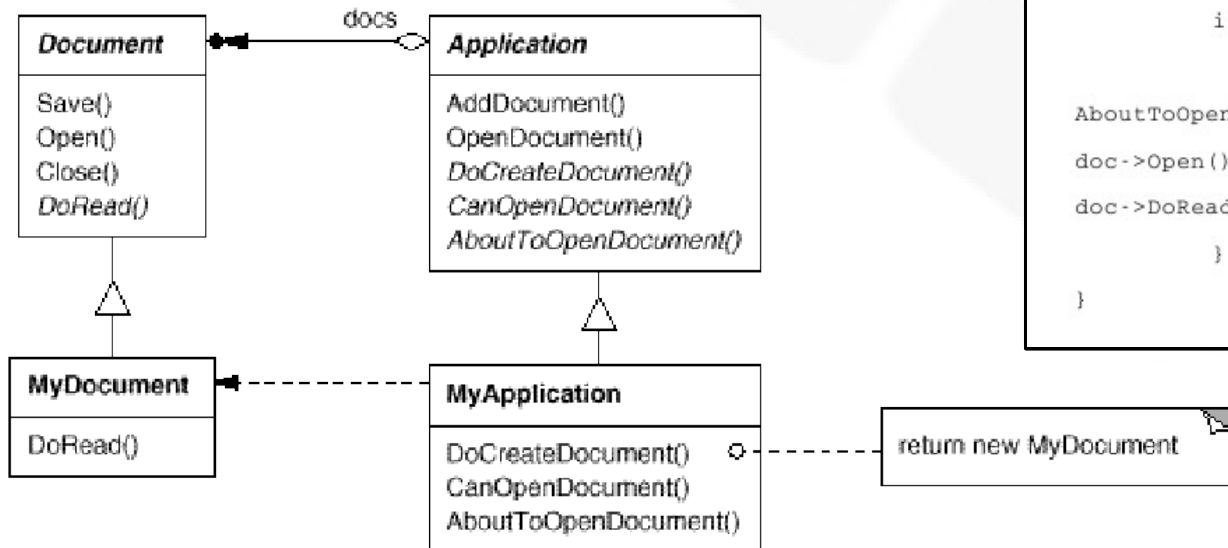
**INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA**

# Template Method

- Propósito:
  - Definir um arcabouço de um algoritmo sob a forma de um método, deixando que alguns passos sejam implementados por sub-classes
  - As sub-classes podem redefinir certos passos de um algoritmo sem modificar a sua estrutura geral

# Template Method

- Motivação:
  - *Application Framework* para editores de documentos



```
void Application::OpenDocument (const char* name) {
    if (!CanOpenDocument(name)) {
        // cannot handle this document
        return;
    }

    Document* doc = DoCreateDocument();
    if (doc) {
        _docs->AddDocument(doc);

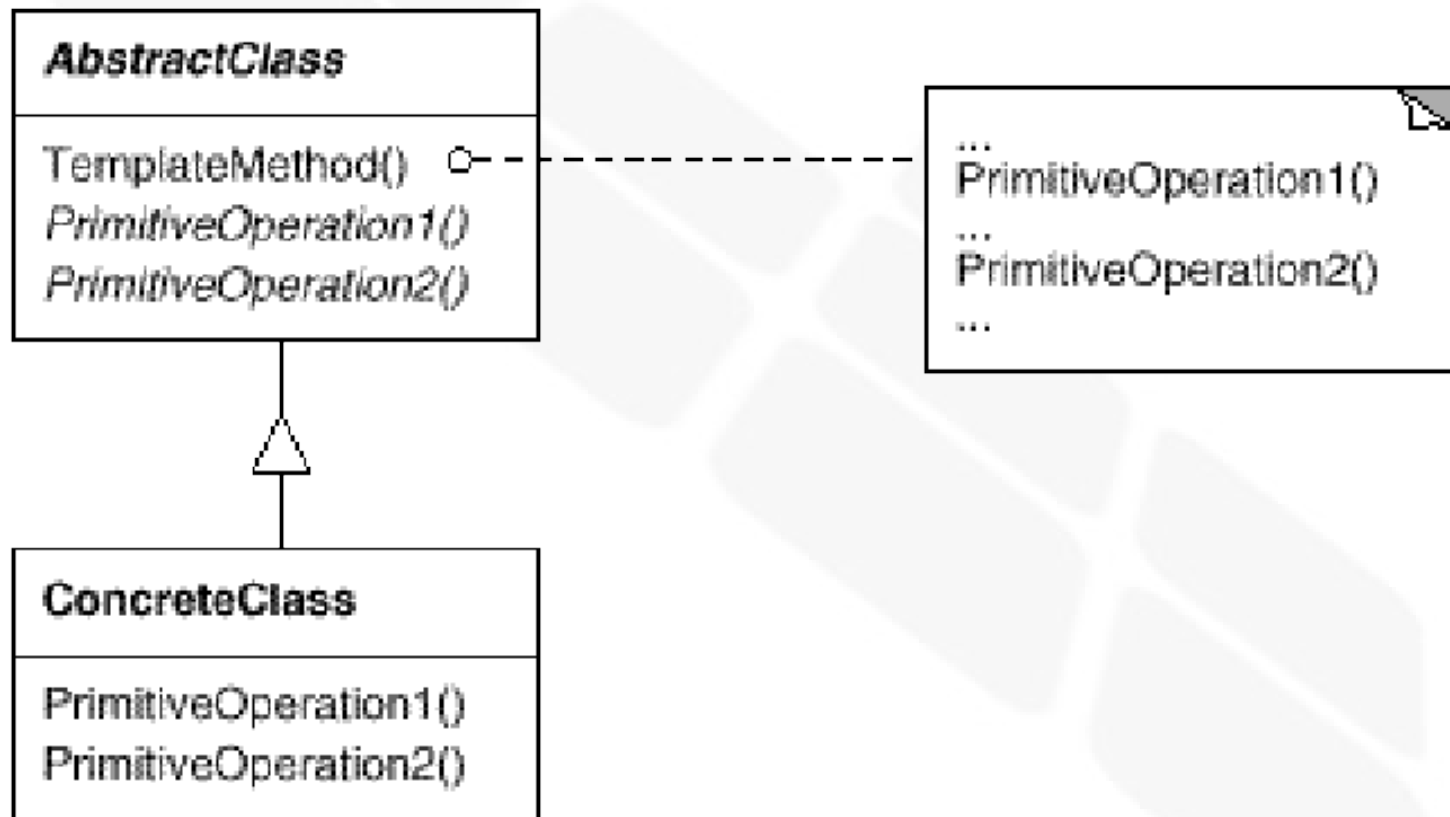
        AboutToOpenDocument(doc);
        doc->Open();
        doc->DoRead();
    }
}
```

# Template Method

- Aplicabilidade:
  - Para implementar uma única vez as partes invariantes de um algoritmo e deixar que sub-classes implementem o comportamento que varia
  - Quando um comportamento comum entre sub-classes deve ser fatorado e localizado em uma classe comum, com o objetivo de evitar repetição de código
  - Para controlar as extensões realizadas pelas sub-classes. O *Template Method* pode definir operações *hook* em pontos específicos e, portanto, permitindo variações somente nestes locais

# Template Method

- Estrutura:



# Template Method

- Participantes:
  - *AbstractClass* (Application):
    - Declara operações primitivas abstratas a serem definidas por sub-classes ao implementar os passos do algoritmo
    - Implementa o *Template Method* definindo o arcabouço do algoritmo. Este *Template Method* invoca tanto as operações primitivas quanto operações concretas de *AbstractClass* ou de outros objetos
  - *ConcreteClass* (MyApplication):
    - Implementa as operações primitivas, executando os passos do algoritmo específicos desta sub-classe

# Template Method

- Colaborações:
  - *ConcreteClasses* esperam que *AbstractClass* implemente os passos invariantes do algoritmo

# Template Method

- Consequências:
  - *Template Methods* são uma técnica fundamental para viabilizar a reutilização de código
  - Conduzem a uma estrutura de Inversão de Controle
  - *Template Methods* podem invocar os seguintes tipos de métodos:
    - Operações concretas (da sua própria classe ou de outros objetos)
    - Operações primitivas (métodos abstratos) – Devem ser Sobrepostas
    - *Factory Methods*
    - Métodos *hook* (métodos de *AbstractClass* com implementações *default* mas que podem ser sobrepostos pelas sub-classes) – Podem ser sobrepostas



# Template Method

- Consequências:
  - Aplicação dos métodos *hook* (ANTES)

```
void DerivedClass::Operation () {  
    // DerivedClass extended behavior  
    ParentClass::Operation();  
}
```

# Template Method

- Consequências:
  - Aplicação dos métodos *hook* (DEPOIS)

```
void ParentClass::Operation () {  
    // ParentClass behavior  
    HookOperation();  
}
```

HookOperation does nothing in ParentClass:

```
void ParentClass::HookOperation () { }
```

Subclasses override HookOperation to extend itsbehavior:

```
void DerivedClass::HookOperation () {  
    // derived class extension  
}
```

# Template Method

- Implementação:
  - Utilizando o controle de acesso do C++
    - As operações primitivas devem ser declaradas como métodos *protected*
    - As operações primitivas devem ser virtuais puras
    - O *Template Method* não deve poder ser sobreposto. Para isso, ele deve ser declarado como não-virtual
  - Minimizando operações primitivas
    - É desejável minimizar o número de operações primitivas que sub-classes devem ter de implementar
    - Quanto mais métodos precisem ser sobrepostos maior o trabalho para os usuários do *Template Method*

# Template Method

- Implementação:
  - Padronização para nomes
    - Pode-se padronizar as operações a serem redefinidas utilizando-se um determinado prefixo, por exemplo, *do\**

# Template Method

- Código exemplo:

```
void View::Display () {  
    SetFocus();  
    DoDisplay();  
    ResetFocus();  
}
```

# Template Method

- Código exemplo:

```
void View::DoDisplay () { }
```

Subclasses override it to add their specific drawing behavior:

```
void MyView::DoDisplay () {  
    // render the view's contents  
}
```

# Template Method

- Usos conhecidos:
  - Na maioria das classes abstratas
  - *Application Frameworks*

# Template Method

- Padrões relacionados:
  - *Factory Methods* são frequentemente chamados por *Template Methods*
  - *Template Methods* usam herança para variar partes de um algoritmo. *Strategies* usam delegação para variar todo o algoritmo

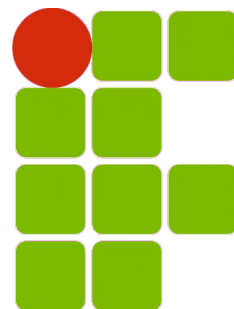


# INF011 – Padrões de Projeto

## 25 – *Template Method*

**Sandro Santos Andrade**  
sandroandrade@ifba.edu.br

**Instituto Federal de Educação, Ciência e Tecnologia da Bahia**  
**Departamento de Tecnologia Eletro-Eletrônica**  
**Graduação Tecnológica em Análise e Desenvolvimento de Sistemas**



**INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA**