Design Patterns

Produced by

Eamonn de Leastar

edeleastar@wit.ie

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

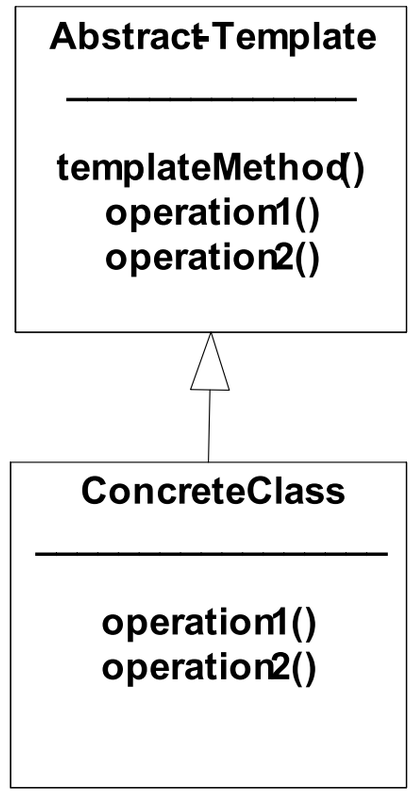
Template Method

Design Pattern

Summary

- Create an abstract class that implements a procedure using abstract methods.

- These abstract methods must be implemented in derived classes to actually perform each step of the procedure.



Intent

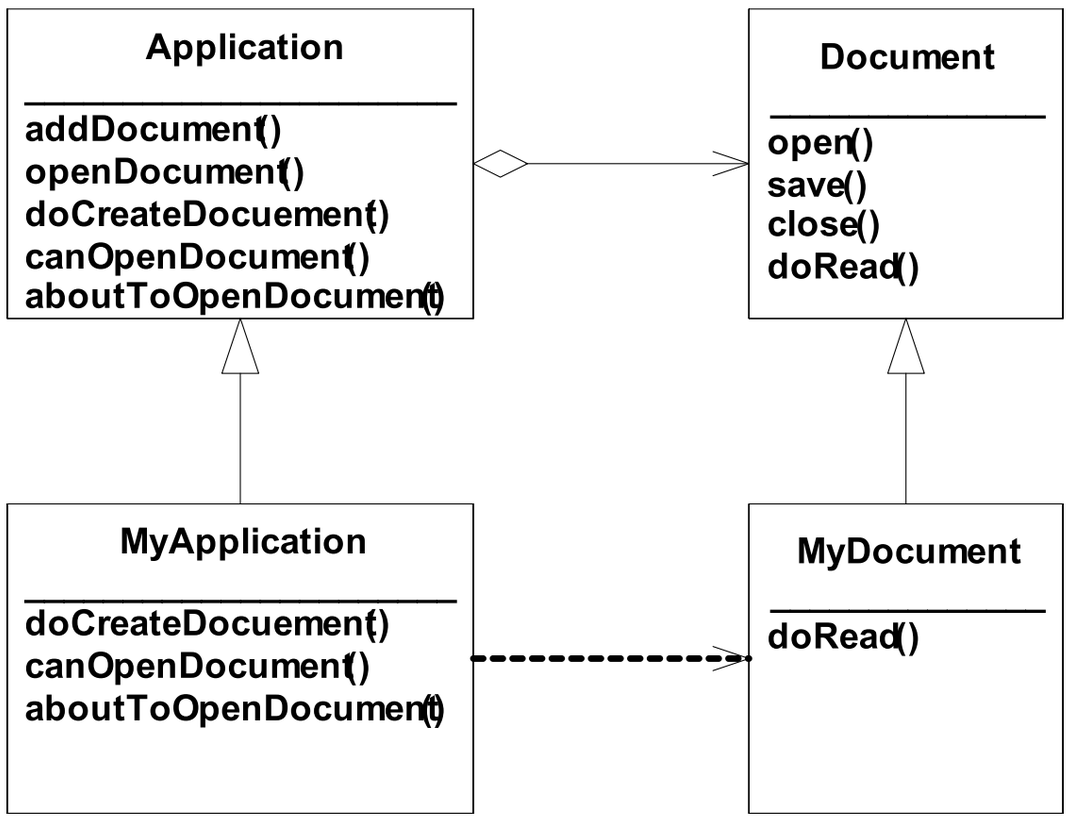
- Define the skeleton of an algorithm in an operation, deferring some steps to subclasses.

- Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

Motivation (example 1)

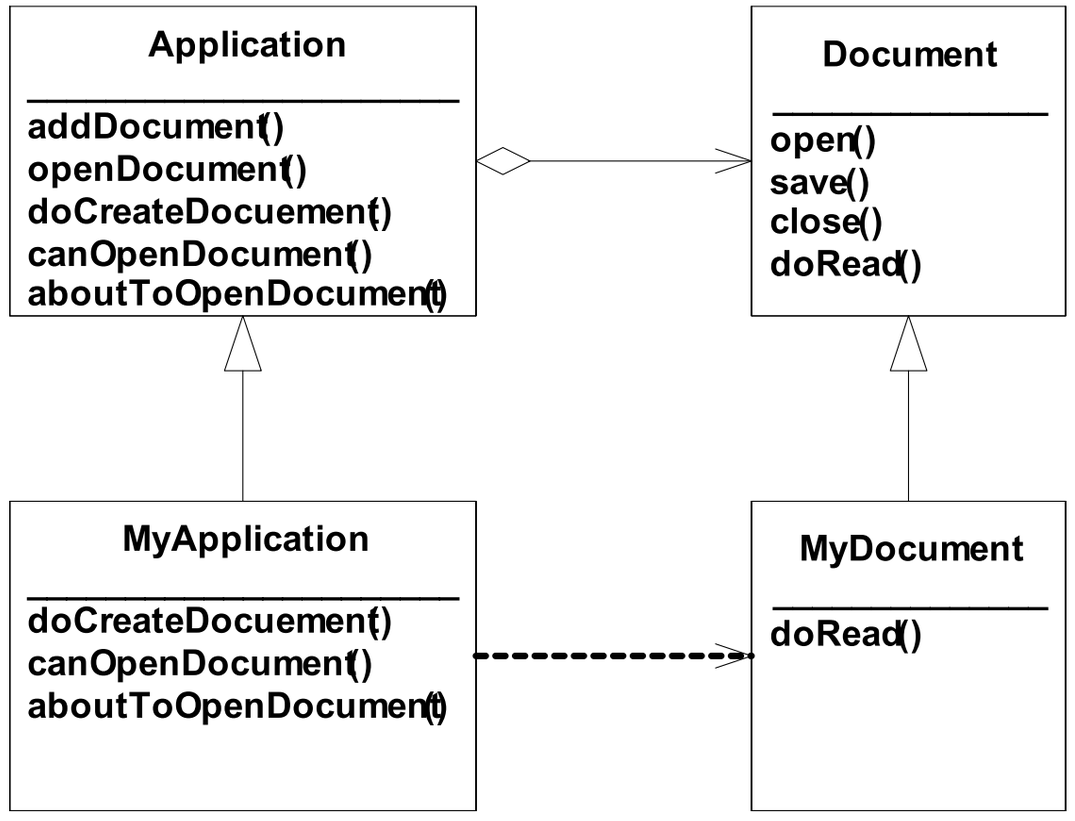
- Consider an application framework that provides Application and Document classes.
  - The Application class is responsible for opening existing documents stored in an external format, such as a file.
  - A Document object represents the information in a document once it's read from the file.
- Applications built with the framework can subclass Application and Document to suit specific needs.
  - For example, a drawing application defines DrawApplication and DrawDocument subclasses;
  - A spreadsheet application defines SpreadsheetApplication and SpreadsheetDocument subclasses.

Motivation (2)

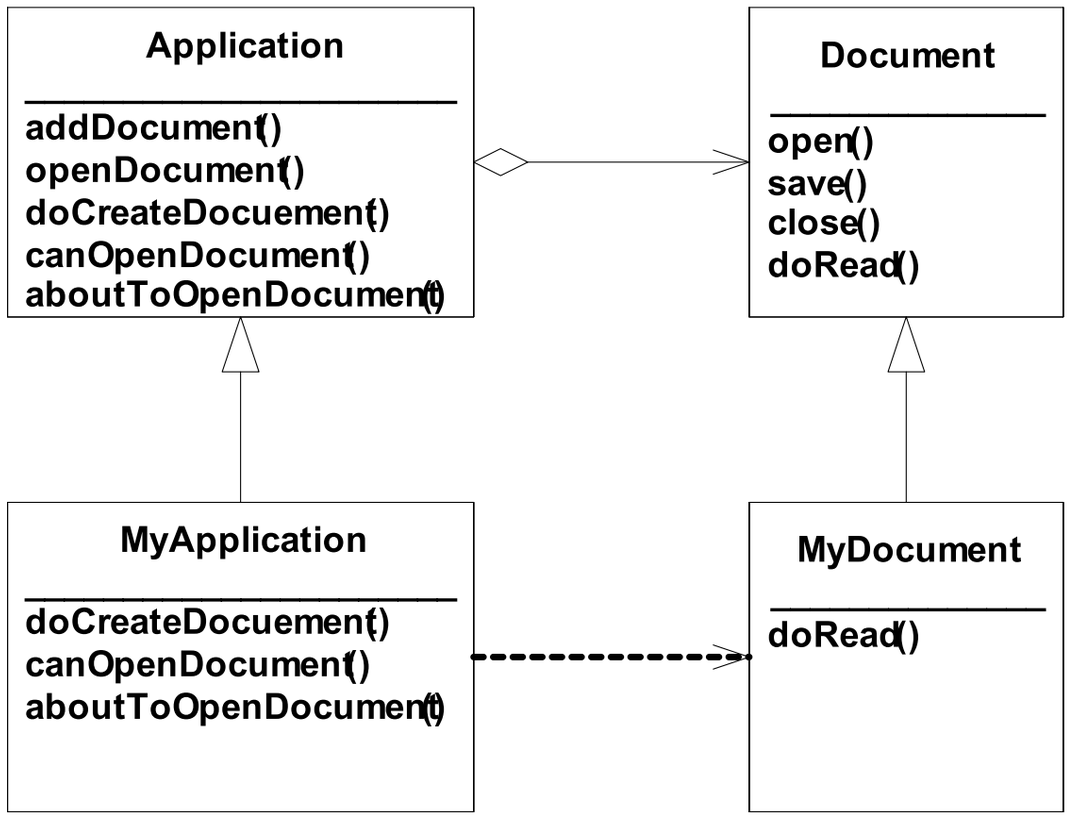


Motivation (4)

- openDocument() is a template method.
- A template method defines an algorithm in terms of abstract operations that subclasses override to provide concrete behavior.
- Application subclasses define the steps of the algorithm that check if the document can be opened (canOpenDocument) and that create the Document (doCreateDocument). Document classes define the step that reads the document (doRead).



Motivation (4)



- The template method also defines an operation that lets Application subclasses know when the document is about to be opened (aboutToOpenDocument), in case they care.
- By defining some of the steps of an algorithm using abstract operations, the template method fixes their ordering, but it lets Application and Document subclasses vary those steps to suit their needs.

openDocument “template method”

```
class Document
{
//...
  void openDocument(String name)
  {
    if (!canOpenDocument(name))
    {
      // cannot handle this document
      return;
    }
    Document doc = doCreateDocument();
    if (doc != null)
    {
      docs.addDocument(doc);
      aboutToOpenDocument(doc);
      doc.open();
      doc.doRead();
    }
  //...
}
```

- The abstract Application class defines the algorithm for opening and reading a document in its openDocument operation

- openDocument defines each step for opening a document. It checks if the document can be opened, creates the application-specific Document object, adds it to its set of documents, and reads the Document from a file.

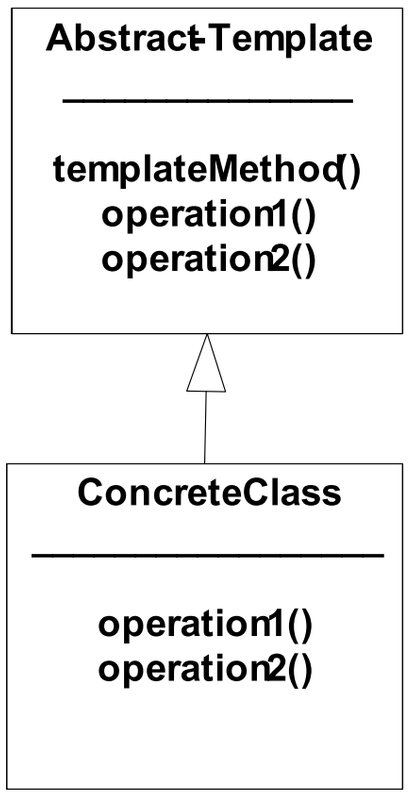
Applicability

- To implement the invariant parts of an algorithm once and leave it up to subclasses to implement the behaviour that can vary.

- When common behaviour among subclasses should be factored and localized in a common class to avoid code duplication.

- To control subclasses extensions. Define a template method that calls "hook" operations at specific points, thereby permitting extensions only at those points.

Structure


AbstractTemplate
templateMethod()
operation1()
operation2()
ConcreteClass
operation1()
operation2()

Participants

- AbstractClass (Application)
    - defines abstract primitive operations that concrete subclasses define to implement steps of an algorithm.
    - implements a template method defining the skeleton of an algorithm. The template method calls primitive operations as well as operations defined in AbstractClass or those of other objects.
- ConcreteClass (MyApplication)
    - implements the primitive operations to carry out subclass-specific steps of the algorithm.

Collaborations

- ConcreteClass relies on AbstractClass to implement the invariant steps of the algorithm.

Consequences (1)

- Template Method is used prominently in frameworks.
    - The framework implements the invariant pieces of a domain's architecture, and defines "placeholders" for all necessary or client customization options.
    - The framework will then call client code, e.g. primitive operations of a template method.
    - This inverted control structure has been labeled "the Hollywood principle" - "don't call us, we'll call you".

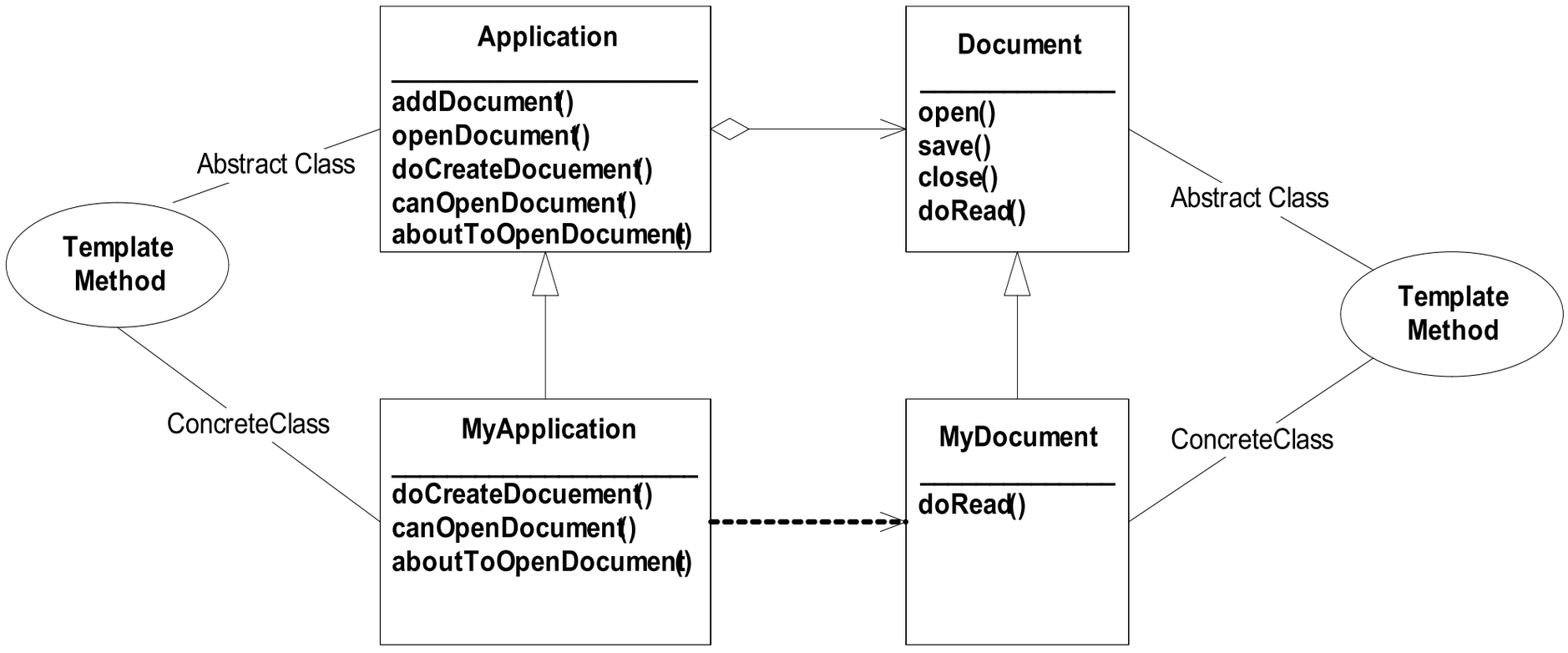
Consequences (2)

- Template methods typically call the following type of methods
  - Concrete AbstractClass operations (that are common to all subclasses)
  - Primitive operations that must be overridden
  - Factory methods (creation of a certain type of object is a variability in the template method, implemented as a Factory.
  - Hook operations which provide default behavior that the subclasses can extend if necessary. Often hook operations do nothing as a default.
- The specialization interface must be described (i.e. inheritance interface). At minimum indicate which methods
  - Must overridden
  - Can be overridden
  - Can not be overridden

Example 2

- A familiar example of a framework that is extended using a Template Method is Java's applet. When you create an applet you're using an application framework:
  - you inherit from JApplet and then override init( ).
- The applet mechanism (which is a Template Method in this case) does the rest by drawing the screen, handling the event loop, resizing, etc.
- An important characteristic of the Template Method is that it is defined in the base class and changing it is prohibited, thus it should be declared final.
- It calls other base-class methods (the ones you override) in order to do its job, but it is usually called only as part of an initialization process (and thus the client programmer isn't necessarily able to call it directly).

UML Pattern Notation



Example 3
PlainTextDocument & HtmlTextDocument

```
public class PlainTextDocument
{
  ...
  public void printPage (Page page)
  {
    printPlainTextHeader();
    System.out.println(page.body());
    printPlainTextFooter();
  }
  ...
}
```

```
public class HtmlTextDocument
{
  ...
  public void printPage (Page page)
  {
    printHtmlTextHeader();
    System.out.println(page.body());
    printHtmlTextFooter();
  }
  ...
}
```

TextDocument superclass

Template Method

```
public abstract class TextDocument
{
  ...
  public final void printPage (Page page)
  {
    printTextHeader();
    printTextBody(page);
    printTextFooter();
  }

  abstract void printTextHeader();

  final void printTextBody(Page page)
  {
    System.out.println(page.body());
  }

  abstract void printTextFooter();
  ...
}
```

PlainTextDocument

```
public class PlainTextDocument extends TextDocument
{
  ...
  public void printTextHeader ()
  {
    // Code for header plain text header here.
  }
  public void printTextFooter ()
  {
    // Code for header plain text footer here.
  }
      ...
}
```

HTMLTextDocument

```
public class HTMLTextDocument extends TextDocument
{
  ...
  public void printTextHeader ()
  {
    // Code for header HTML text header here.
  }
  public void printTextFooter ()
  {
    // Code for header HTML text footer here.
  }
      ...
}
```