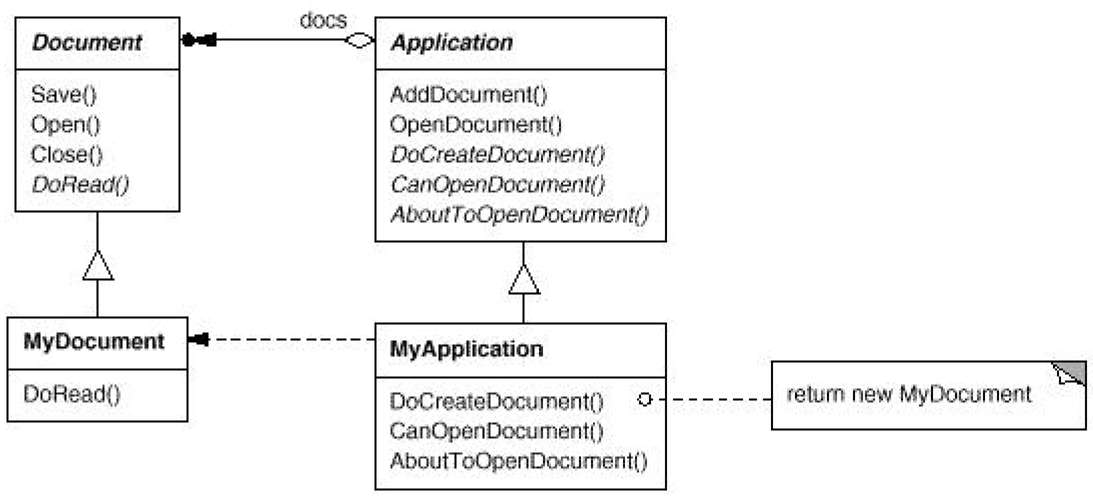
The Template Method Pattern

The Template Method Pattern

- 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
  - ➩ Sometimes you want to specify the order of operations that a method uses, but allow subclasses to provide their own implementations of some of these operations
  - ➩ Consider:



The Template Method Pattern

- Motivation
  - ➪ The OpenDocument() method might look like this:

```
public void OpenDocument (String name) {
  if (!CanOpenDocument(name)) { return; }
  Document doc = DoCreateDocument();
  if (doc != null) {
    docs.AddDocument(doc);
    AboutToOpenDocument(doc);
    doc.Open();
    doc.DoRead();
  }
}
```

  - ➪ The OpenDocument() method is a *Template Method*
  - ➪ The template method fixes the order of operations, but allows Application subclasses to vary those steps as needed

Template Method Pattern Example 1

- Suppose you had a PlainTextDocument class as follows:

```
public class PlainTextDocument {

  ...

  public void printPage (Page page) {

    printPlainTextHeader();   // Unique to PlainTextDocument
    System.out.println(page.body());
    printPlainTextFooter();   // Unique to PlainTextDocument

  }

  ...

}
```

Template Method Pattern Example 1 (Continued)

- And then you wrote an HtmlTextDocument class like this:

```
public class HtmlTextDocument {

  ...

  public void printPage (Page page) {

    printHtmlTextHeader();     // Unique to HtmlTextDocument
    System.out.println(page.body());
    printHtmlTextFooter();     // Unique to HtmlTextDocument

  }

  ...

}
```

Template Method Pattern Example 1 (Continued)

- The printPage() methods in the PlainTextDocument and HtmlTextDocument classes look much alike
- Whenever we see two such similar methods in subclasses, it makes sense to bring the methods together into a superclass method
- We can write a printPage() template method in a superclass that allows for PlainTextDocument and HtmlTextDocument to provide their unique implementations of abstract methods to print the header and footer

Template Method Pattern Example 1 (Continued)

- Here is the TextDocument superclass:

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

Template Method Pattern Example 1 (Continued)

- And here is the new PlainTextDocument class (the new HtmlTextDocument class is similar):

```
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.
  }
  ...
}
```

- Note that all we have to do is provide the proper implementations of the abstract methods in the TextDocument superclass

The Template Method Pattern

- Applicability

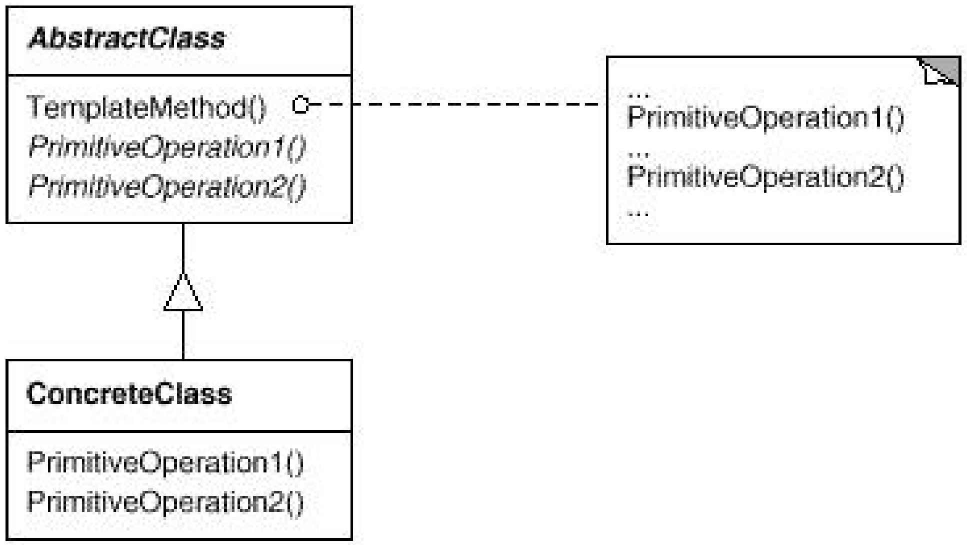
  Use the Template Method pattern:

  - ➪ To implement the invariant parts of an algorithm once and leave it up to subclasses to implement the behavior that can vary
  - ➪ To localize common behavior among subclasses and place it in a common class (in this case, a superclass) to avoid code duplication. This is a classic example of ”code refactoring.”
  - ➪ To control how subclasses extend superclass operations. You can define a template method that calls "hook" operations at specific points, thereby permitting extensions only at those points.

The Template Method is a fundamental technique for code reuse.

The Template Method Pattern

- Structure



The Template Method Pattern

- Implementation Issues
  - ➩ Operations which must be overridden by a subclass should be made abstract
  - ➩ If the template method itself should not be overridden by a subclass, it should be made final
  - ➩ To allow a subclass to insert code at a specific spot in the operation of the algorithm, insert “hook” operations into the template method. These hook operations may do nothing by default.
  - ➩ Try to minimize the number of operations that a subclass must override, otherwise using the template method becomes tedious for the developer
  - ➩ In a template method, the parent class calls the operations of a subclass and not the other way around. This is an inverted control structure that's sometimes referred to as "the Hollywood principle," as in, "Don't call us, we'll call you".

Template Method Pattern Example 2

- Suppose we have a Manufacturing class as follows:

```
public class Manufacturing {
  ...
  // A template method!
  public final void makePart () {
    operation1();
    operation2();
  }

  public void operation1() {
    // Default behavior for Operation 1
  }

  public void operation2() {
    // Default behavior for Operation 2
  }
  ...
}
```

Template Method Pattern Example 2 (Continued)

- And a subclass wants to do some behavior between operation1() and operation2() of makePart(), so it overrides operation2() as follows:

```
public class MyManufacturing {
  ...
  // We want to do behavior between operation1() and
  // operation2() of makePart(), so we override operation2()
  // as follows.  (Note: we could just as easily have
  // overridden operation1().)
  public void operation2() {
    // Put behavior we want to do BEFORE the normal Operation2
    //  here!
    super.operation2();
  }
  ...
}
```

Template Method Pattern Example 2 (Continued)

- If you find that many subclasses want to do this, it is wise to modify the superclass and put in a hook operation:

```
public class Manufacturing {
  ...
  // A template method!
  public final void makePart () {
    operation1();
    hook();  // A hook method
    operation2();
  }
  // Do nothing hook method.
  public void hook() {}
  ...
}
```

- Now subclasses only need to provide an implementation for the hook() method