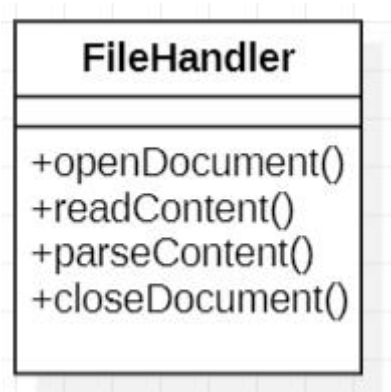
Template Method

A behavioral pattern

Learning goals

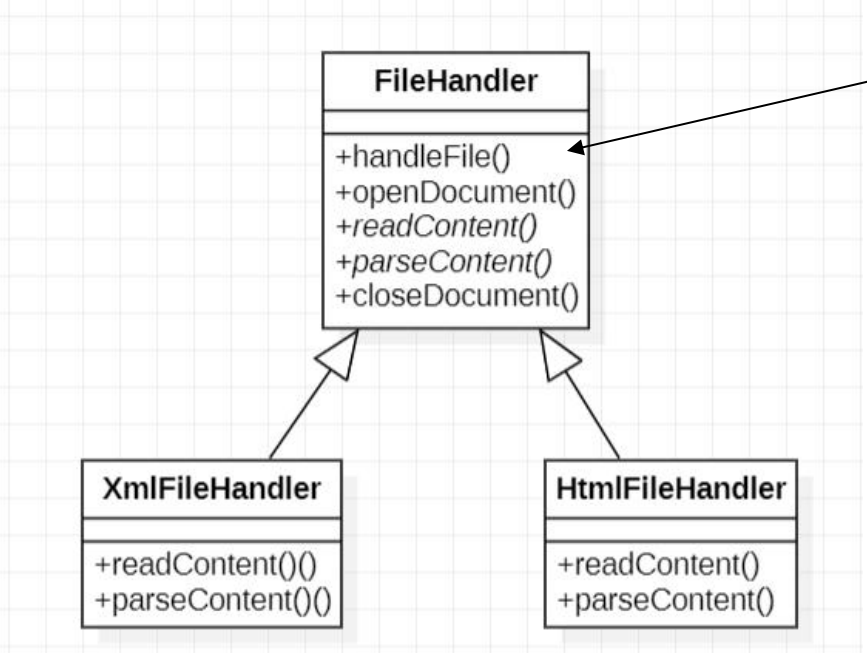
1. Learn the idea, structure, and Java implementation of the Template Method pattern.
2. Learn to apply the Template Method DP in your own programming.

Background: Template Method



- Frequently, an large operation contains a **sequence of operations**, i.e. an algorithm.
  - E.g. handling a file consists of opening, reading, parsing, and closing a document.
- The class above shows the primitive operations for file handling.
- What happens if there will be different types of files to handle?
  - E.g. XML files, HTML files, ...
  - Obviously, a subclass can be created for each specific type: XMLFileHandler, HtmlFileHandler, …
  - This may result in duplicate code, as some steps may be the same for some file types.
- Also, we would like to code the sequence for carrying out the primitive operations only once.

Idea of Template Method




```
public UsefulData handleFile(File filePath) {
    file = openDocument(filePath);
    rawContent = readContent(file);
    usefulData = parseContent(rawContent);
    closeDocument(file);
    return usefulData;
}
```

- The **handleFile()** is a **template method** that specifies the algorithm.

Idea of Template Method

- In the Template Method DP, we create a template method in the superclass.
- It specifies the algorithm.
  - The template method calls the primitive operations which are also declared in the superclass.
- The subclasses may override some or all of the superclass implementations of the primitive operations.
- Some of the primitive operations in the superclass may be declared abstract.
  - This enforces coding the implementation in the subclass, as there is no default one.
- The template method itself is declared final as it is not designed for overriding.

General structure

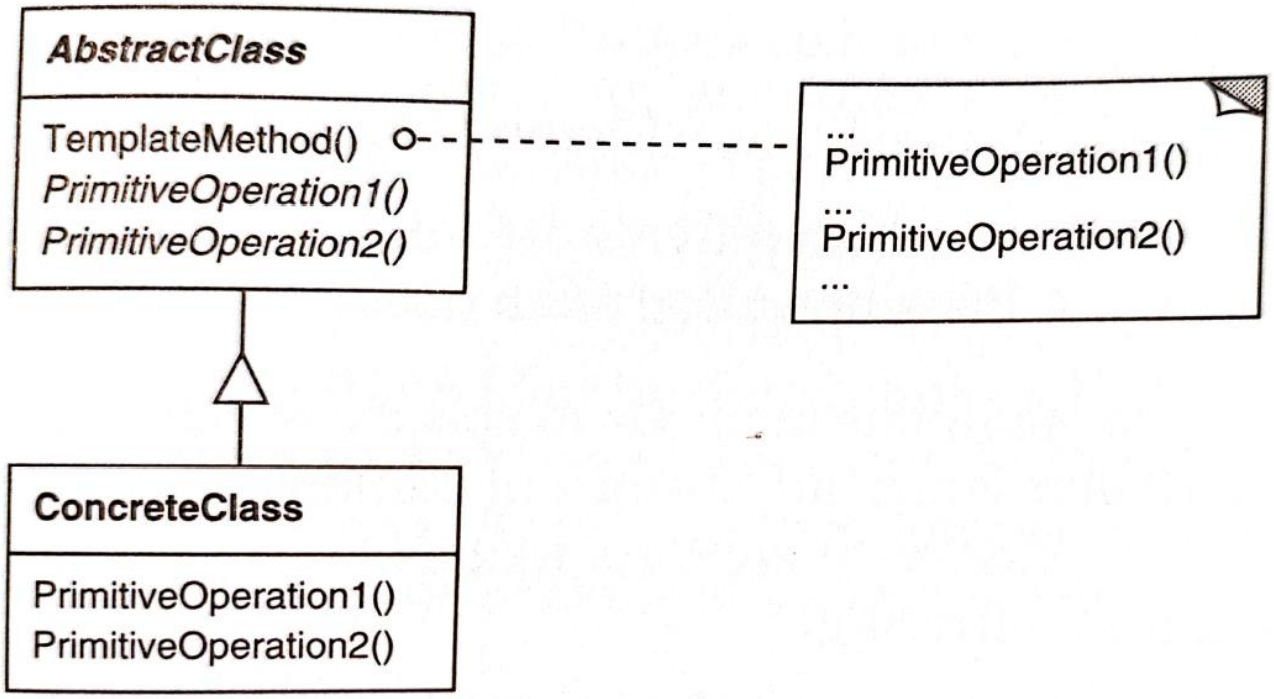


Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 327

Roles

- **Abstract Class**: Defines the abstract or default primitive operations that implement the algorithm. Also, implements the template method.
- **Concrete Class**: provides subclass-specific implementations to the primitive operations where necessary.

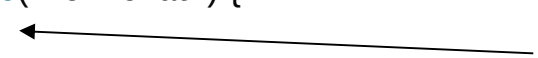
Inversion of Control

- In the Template Method DP, the template method in the superclass is responsible for calling the subclass methods, not vice versa.
- This design approach is referred as **Inversion of Control** which is common in framework setups.
  - An object that depends on other object should not call it but wait to be called.
  - Also called as the Hollywood Principle (”Don’t call us, we’ll call you!)

Hooks

```
public UsefulData handleFile(File filePath) {
  prepareForFileRead();
  file = openDocument(filePath);
  rawContent = readContent(file);
  usefulData = parseContent(rawContent);
  closeDocument(file);
  return usefulData;
}

public void prepareForFileRead() {
}
```



hook

- Sometimes it is feasible to add places for expansion into the template method.
- These **hooks** are primitive operations that have an empty, concrete implementation in the superclass.
- If needed, they can be overridden in a subclass, thus acting as starting-points for additional operations.

Practical issues

- The Template Method DP make it possible to code the invariant parts of an algorithm only once.
- Use hooks to control subclass extension. They specify the permitted places for additional subclass functionality.
- Try to minimize abstract primitive operations.
  - They make coding the subclasses tedious.
- The Template Method DP is applied very frequently, as the very idea of abstract classes encourages its use.