





**🧩 What is the Template Method Pattern?**

The **Template Method Pattern** is a behavioral design pattern that defines the **skeleton of an algorithm** in a base class, but **defers some steps to subclasses**.

It allows subclasses to **override certain steps** of an algorithm without changing its structure.

**✅ Use Cases of Template Method Pattern**

| **Use Case** | **Example** |
| --- | --- |
| ✅ Data processing pipelines | File parsing: base class defines read → process → write flow |
| ✅ Report generation | Common format steps, subclass defines data section |
| ✅ Authentication mechanisms | Common steps for login, subclass defines validation method |
| ✅ Game development (turn-based games) | Skeleton of game loop; subclasses define player actions |
| ✅ UI rendering | Framework defines lifecycle; subclasses define rendering logic |
| ✅ Testing frameworks | @Before, @After, @Test follow a template |

**Structure of Template Method Pattern**

**👇 Participants**

| **Component** | **Description** |
| --- | --- |
| **AbstractClass** | Defines the **template method** (final) and abstract methods |
| **ConcreteClass** | Implements the variable steps of the algorithm |
| **✅ Advantages**   | **Advantage** | **Description** | | --- | --- | | ✅ **Reusability** | Common logic lives in the abstract base class | | ✅ **Inversion of control** | Superclass calls the subclass implementation | | ✅ **Avoids code duplication** | Shared steps only written once | | ✅ **Easy to enforce sequence** | Template method defines execution order | | ✅ **Open/Closed Principle** | Add new steps via subclass without modifying base |   **❌ Disadvantages**   | **Disadvantage** | **Description** | | --- | --- | | ❌ **Inheritance tightly couples classes** | Cannot be used with final classes | | ❌ **Less flexible than composition** | Hard to vary behavior dynamically | | ❌ **Limited extensibility at runtime** | Runtime changes require subclassing | | ❌ **Subclass explosion** | Many subclasses needed for slight variations |   **✅ When to Use Template Method**   | **Use If...** | **Example** | | --- | --- | | ✅ You have invariant logic across multiple processes | Report generation | | ✅ You want to enforce a fixed flow | Authentication, data processing | | ✅ You want to let subclasses override some steps | File readers, games |   **❌ When NOT to Use**   | **Avoid If...** | **Reason** | | --- | --- | | ❌ You want composition over inheritance | Strategy pattern may be better | | ❌ Behavior needs to vary at runtime | Template is compile-time polymorphism | | ❌ Shared logic is minimal | Subclassing adds unnecessary complexity |   **🔁 Template Method vs Strategy Pattern**   | **Feature** | **Template Method** | **Strategy Pattern** | | --- | --- | --- | | Based on | Inheritance | Composition | | Flexibility | Compile-time (fixed via subclassing) | Runtime (change behavior dynamically) | | Shared behavior | In abstract class | In context using interchangeable strategy | | Use when | Behavior mostly common with few variations | Behavior varies entirely |   **✅ Summary Table**   | **Aspect** | **Template Method Pattern** | | --- | --- | | Type | Behavioral | | Core Idea | Skeleton of algorithm in superclass | | Main Use | Enforce structure; allow variation | | Extends | Abstract base class | | Real-world Uses | Report generation, parsing, UI hooks | |  |

**1. Abstract Class**

java

CopyEdit

public abstract class ReportTemplate {

// Template method (defines algorithm skeleton)

public final void generateReport() {

fetchData();

formatReport();

exportReport();

}

protected abstract void fetchData(); // Step to be implemented by subclasses

protected void formatReport() {

System.out.println("Formatting report...");

}

protected void exportReport() {

System.out.println("Exporting report to PDF...");

}

}

**2. Concrete Class 1 – Sales Report**

java

CopyEdit

public class SalesReport extends ReportTemplate {

@Override

protected void fetchData() {

System.out.println("Fetching sales data...");

}

}

**3. Concrete Class 2 – Inventory Report**

java

CopyEdit

public class InventoryReport extends ReportTemplate {

@Override

protected void fetchData() {

System.out.println("Fetching inventory data...");

}

}

**4. Client Code**

java

CopyEdit

public class TemplateMethodDemo {

public static void main(String[] args) {

ReportTemplate report1 = new SalesReport();

report1.generateReport();

ReportTemplate report2 = new InventoryReport();

report2.generateReport();

}

}