Template Method Design Pattern in Java

APRIL 2, 2018 BY PANKAJ — 15 COMMENTS

Template Method is a **behavioral design pattern**. Template Method design pattern is used to create a method stub and deferring some of the steps of implementation to the subclasses.

Table of Contents [hide]

Template Method Design Pattern

Template method defines the steps to execute an algorithm and it can provide default implementation that might be common for all or some of the subclasses.

Let's understand this pattern with an example, suppose we want to provide an algorithm to build a house. The steps need to be performed to build a house are – building foundation, building pillars, building walls

and windows. The important point is that the we can't change the order of execution because we can't build windows before building the foundation. So in this case we can create a template method that will use different methods to build the house.

Now building the foundation for a house is same for all type of houses, whether its a wooden house or a glass house. So we can provide base implementation for this, if subclasses want to override this method, they can but mostly it's common for all the types of houses.

To make sure that subclasses don't override the template method, we should make it final.

Template Method Abstract Class

Since we want some of the methods to be implemented by subclasses, we have to make our base class as abstract class.

`HouseTemplate.java`

```
package com.journaldev.design.template;

public abstract class HouseTemplate {

	//template method, final so subclasses can't override
	public final void buildHouse(){
		buildFoundation();
		buildPillars();
		buildWalls();
		buildWindows();
		System.out.println("House is built.");
	}

	//default implementation
	private void buildWindows() {
		System.out.println("Building Glass Windows");
	}

	//methods to be implemented by subclasses
	public abstract void buildWalls();
	public abstract void buildPillars();
```

`buildHouse()` is the template method and defines the order of execution for performing several steps.

Template Method Concrete Classes

We can have different type of houses, such as Wooden House and Glass House.

WoodenHouse.java

```
package com.journaldev.design.template;

public class WoodenHouse extends HouseTemplate {

	@Override
	public void buildWalls() {
		System.out.println("Building Wooden Walls");
	}

	@Override
	public void buildPillars() {
		System.out.println("Building Pillars with Wood coating");
	}

}
```

We could have overridden other methods also, but for simplicity I am not doing that.

GlassHouse.java

```
package com.journaldev.design.template;

public class GlassHouse extends HouseTemplate {

	@Override
	public void buildWalls() {
		System.out.println("Building Glass Walls");
	}

	@Override
	public void buildPillars() {
		System.out.println("Building Pillars with glass coating");
	}

}
```

Template Method Design Pattern Client

Let's test our template method pattern example with a test program.

HousingClient.java

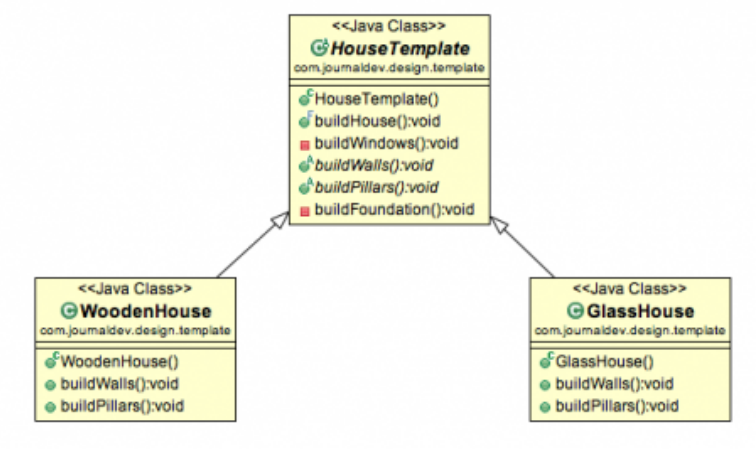
```
package com.journaldev.design.template;

public class HousingClient {

	public static void main(String[] args) {

		HouseTemplate houseType = new WoodenHouse();

		//using template method
		houseType.buildHouse();
		System.out.println("************");

		houseType = new GlassHouse();

		houseType.buildHouse();
	}

}
```

Notice that client is invoking the template method of base class and depending of implementation of different steps, it's using some of the methods from base class and some of them from subclass.

Output of the above program is:

```
Building foundation with cement,iron rods and sand
Building Pillars with Wood coating
Building Wooden Walls
Building Glass Windows
House is built.
************
Building foundation with cement,iron rods and sand
Building Pillars with glass coating
Building Glass Walls
Building Glass Windows
House is built.
```

Template Method Class Diagram



Template Method Design Pattern in JDK

- All non-abstract methods of java.io.InputStream, java.io.OutputStream, java.io.Reader and java.io.Writer.
- All non-abstract methods of java.util.AbstractList, java.util.AbstractSet and java.util.AbstractMap.

Template Method Design Pattern Important Points

1. Template method should consists of certain steps whose order is fixed and for some of the methods, implementation differs from base class to subclass. Template method should be final.
2. Most of the times, subclasses calls methods from super class but in template pattern, superclass template method calls methods from subclasses, this is known as Hollywood Principle – "don't call us, we'll call you.".
3. Methods in base class with default implementation are referred as **Hooks** and they are intended to be overridden by subclasses, if you want some of the methods to be not overridden, you can make them final, for example in our case we can make buildFoundation() method final because if we don't want subclasses to override it.

Thats all for template method design pattern in java, I hope you liked it.

« PREVIOUS
Strategy Design Pattern in Java – Example Tutorial

NEXT »
Visitor Design Pattern in Java

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: DESIGN PATTERNS

Comments

Mustafa says
JULY 26, 2018 AT 8:42 AM
Excellent thank you

Reply

Larsen says
JULY 8, 2018 AT 10:05 PM
That's a fantastic explanation.
Reply

shobhit khandelwal says
JULY 30, 2017 AT 3:52 AM
explanation is superb.
Reply

Prosenjit says
SEPTEMBER 25, 2016 AT 4:37 AM
really helpful. Thanks a lot. □
Reply

test says
DECEMBER 16, 2015 AT 6:14 AM
What if I do it like below..the purpose of template method is lost.
HouseTemplate houseType = new WoodenHouse();
houseType .buildWindows();
houseType .buildWalls();
houseType .buildPillars();
How to restrict this.
Reply

Raghav says
AUGUST 24, 2016 AT 7:07 PM
make the buildWindows, buildwalls and buildpillars as protected so that they are not seen from HouseTemplate.
Usually, only the buildHouse method should be exposed (via interface) and all other methods be hidden from the user/other modules that use the HouseTemplate.
Anyways, the example above give us an idea about the template pattern only. Well Explained.
Reply

Nilesh Suryavanshi says
APRIL 8, 2015 AT 11:11 PM
Real time approach & Real time professional … Great
Reply

Himansu Nayak says
DECEMBER 16, 2014 AT 1:47 PM
Hi Pankaj,
The jdk classes(Reader, Writer, …) mentioned by you don't have any sort of
template_method (){
m1();
m2(); …
}
so how have they implemented the fixed order of calling method.
Thanks,
Himansu
Reply

perminder singh says
OCTOBER 3, 2014 AT 5:39 AM
nice one. really awesome…..
Reply

Sajal Saha says
JUNE 12, 2014 AT 12:04 AM
Nice Tutorial.
Reply

Subu says
NOVEMBER 18, 2013 AT 4:39 PM
Excellent !! Thanks a lot for all your articles in Design Patterns ! Really helpful !

Reply

Ravi says

SEPTEMBER 20, 2013 AT 8:22 AM

Hello Pankaj,

Your design pattern tutorials are excellent !!! Thanks a lot !!!

Wishing you all the best !!!

Thanks,

Ravi

Reply

Sreeni says

AUGUST 3, 2013 AT 7:24 PM

The buildWindows() and buildFoundation() methods are private, so we cannot overrride them in sub classes. Basically the statement "We could have overridden other methods also, but for simplicity I am not doing that." is incorrect, unless the two methods mentioned here are either abstract or public.

Reply

Pankaj says

AUGUST 4, 2013 AT 12:22 AM

Thats the point to note here, if you don't want subclasses to override any method then make it private like I have done in this example but if you want to provide flexibility for subclasses to override them, keep them public.

The designing of system based on the requirements and you can keep them public or private.

Reply

You Know says

DECEMBER 23, 2014 AT 11:53 AM

You're wrong again. You could also make the method final. So making them private is not the only way to guarantee they won't be overriden.

As a matter of fact, there's not a single definition of Template method that defines that the actual templateMethod() must be final. In some cases, you might want to let it be overriden.

Stop making stuff up, mate, you're not meant for that. Stick to the text books, and stop copying stuff from Joe!

You're pissing me off.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

DESIGN PATTERNS TUTORIAL

Java Design Patterns

Creational Design Patterns

- Singleton
- Factory
- Abstract Factory
- Builder
- Prototype

Structural Design Patterns

- Adapter
- Composite
- Proxy
- Flyweight
- Facade
- Bridge
- Decorator

Behavioral Design Patterns

- Template Method
- Mediator
- Chain of Responsibility
- Observer
- Strategy
- Command
- State
- Visitor
- Interpreter
- Iterator
- Memento

Miscellaneous Design Patterns

- Dependency Injection
- Thread Safety in Java Singleton

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial