**Template Method Design Pattern**

**Overview**

This design pattern defines barebone skeleton of algorithm in a method defined in core parent class that defers slightly from subclasses in some steps. Subclasses then inherit parent class methods and redefine certain steps differently without changing algorithms structure. Parent class defines mandatory steps ( methods that cannot be overridden ) some optional methods that can be overridden by subclasses inheriting parent classes. We can make common sense of template in code where we pick a template and then make some changes leaving some part intact.

A simple example would be making of tea and coffee where processes are similar such as boiling of water, mixing milk, and then there are steps such as coffee versus tea that is added in mixture which shows some activities carried over from parent, and some activities not shared at all.

This design pattern is about inheritance data structure where a parent class defines a method and locks few method inside that and child classes inherit parent class and carry behaviours without modification, by keeping same method signature, and then in parent class some methods are left open to be overridden by child classes in some cases otherwise some methods are supposed to be inherited and used in child classes. This process is good for controlling multiple objects through one main object. With regular inheritance child classes would have freedom to override method whereas template method pattern would provide a template of same behaviour and then allow some freedom in child classes level.

**Benefits**

* It can lock number of steps that objects can do in shared situation and can do its own activity as well without duplicating process ( method duplication) by inheriting characters of parent.
* It provides flexibility to some of activities which are allowed to overridden.
* One object or one template means there is control in what can be done in subclasses and what cannot be done; it is clearly defined by parent.

**Problem/Scenario**

As an example of this would be two processes that are similar yet different in some way. There are lot of them where I have chosen cooking of pasta and cooking of noodle. Both of these processes involve following processes which are similar.

* Heating the pan
* Adding oil on pan
* Adding vegetable
* Adding pasta/noodle and stirring

Process that is not similar

In noodle cooking, you would add water and make soup but with pasta, you would eat it dry so adding water is not necessary.

**Before Refactor**

Before refactor Pasta, and Noodle would use simple inheritance to carry out abovementioned processes but it will give them freedom to override method freely, we wanted to lock down some compulsory processes.

**After Refactor**

After refactor, few methods are grouped into a superclass and sealed and cannot be overridden on child classes. But, some abstract methods can be overridden by child classes CookPasta, and CookNoodle would be able to add more activities in their own ways such as adding barbecue sauce instead of tomato sauce, adding salt, or just pepper according to taste. Locking down processes such as heating pan, adding oil, adding noodle/pasta are mandatory for cooking these food.

After inheritance, child classes would have freedom in some method levels to show different behaviour than defined in parent. So, with this implementation Cooking pasta and cooking noodle should both have steps 1, 2, and 3 but in step 4 where it says add sauce, it could add any sauce , also add any condiments such as salt, pepper or just sult. So, it gave more control on steps that are mandatory from Recipe to pasta and noodle.

Before/after link

https://github.com/sppanday/S120-PRT583-Group-A/tree/feat/design-pattern/s260598-PandaySurendra/Sprint-3-Deliverables/Task017\_Iterator\_Pattern/PizzaHouseIteratorPattern/PizzaHouseIteratorPattern

**UML Diagram**

Please find it inside relevant project UML in pattern folder

/TemplateMethodPattern.png

**Reference**

Freeman Eric., Freeman Elisa. 2004. 1st ed. O’Reily Media Inc. 1995, Sebasttopo, CA