The ***Template Pattern*** defines the steps of an algorithm and allows subclasses to provide the implementation for one or more steps.

The ***Template Method Pattern*** defines the skeleton of an algorithm in a method, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm’s structure.

***New, hip CaffeineBeverage powered by Template Method***

* The CaffeineBeverage class runs the show; It has the algorithm, and protects it.
* The CaffeineBeverage class maximizes reuse among the subclasses.
* The algorithm lives in one place and code changes only need to be made there.
* The Template method version provides a framework that other caffeine beverages can be plugged into. New caffeine beverages only need to implement a couple of methods.
* The CaffeineBeverage class concentrates knowledge about the algorithm and relies on subclasses to provide complete implementations.

**AbstractClass**

templateMethod()

primitiveOperation1()

primitiveOperation2()

primitiveOperation1()

primitiveOperation2()

**ConcreteClass**

primitiveOperation1()

primitiveOperation2()

**abstract** **class** AbstractClass{

**final** **void** templateMethod() {

primitiveOperation1();

primitiveOperation2();

concreteOperation();

}

**abstract** **void** primitiveOperation1();

**abstract** **void** primitiveOperation2();

**void** concreteOperation() {

//implementation here

}

}

**abstract** **class** AbstractClass{

**final** **void** templateMethod() {

primitiveOperation1();

primitiveOperation2();

concreteOperation();

hook();

}

**abstract** **void** primitiveOperation1();

**abstract** **void** primitiveOperation2();

**void** concreteOperation() {

//implementation here

}

**void** hook() {} // A concrete method but it does nothing

}

**Note:** We can also have concrete methods that do nothing by default, we call these hooks. Subclasses are free to override these but don’t have to. A hook is a method that is declared in the abstract class, but only given an empty or default implementation. This gives subclasses the ability to “hook into” the algorithm at various points, if they wish; a subclass is also free to ignore the hook;

Q: How do I know when to use abstract method and hooks?

A: Use abstract methods when your subclass MUST provide an implementation of the method or step in the algorithm. Use hooks when that part of that algorithm is optional. With hooks, a subclass may choose to implement that hook, but it doesn’t have to.

* A “template method” defines the steps of an algorithm, deferring to subclasses for the implementation of those steps.
* The Template Method Pattern gives us an important technique for code reuse.
* The template method’s abstract class may define concrete methods, abstract methods, and hooks.
* Abstract methods are implemented by subclasses.
* Hooks are methods that do nothing or default behavior in the abstract class, but may be overridden in the subclass.
* To prevent subclasses from changing the algorithm in the template method. Declare the template method as final.
* You will see lots of uses of the Template Method Pattern in real world code, but don’t expect it all (like any pattern) to be designed “by the book”.
* The Strategy and Template Method Patterns both encapsulate algorithms, one by inheritance and one by composition.
* The Factory Method is a specialization of Template Method.