What is the difference between the template method and the strategy patterns?

148

64

Can someone please explain to me what is the difference between the template method pattern and the strategy pattern is?

As far as I can tell they are 99% the same - the only difference being that the template method pattern has an abstract class as the base class whereas the strategy class uses an interface that is implemented by each concrete strategy class.

However, as far as the *client* is concerned they are consumed in exactly the same way - is this correct?

design-patterns strategy-pattern template-method-pattern

edited Aug 3 '18 at 12:22
Yassin Hajaj
15.1k 8 30 62

asked Mar 21 '09 at 13:06
Calanus
11.4k 23 71 109

2 This post in SO has a better answer for the same question: stackoverflow.com/questions/464524/… – Gob00st Apr 9 '12 at 15:51

10 The question gob00st linked to is the difference between strategy and bridge. It's not the answer to this question at all. – bluekeys Jan 30 '15 at 9:46

16 Answers

118

The main difference between the two is when the concrete algorithm is chosen.

With the **Template method pattern** this happens at *compile-time* by *subclassing* the template. Each subclass provides a different concrete algorithm by implementing the template's abstract methods. When a client invokes methods of the template's external interface the template calls its abstract methods (its internal interface) as required to invoke the algorithm.

```
class ConcreteAlgorithm : AbstractTemplate
{
    void DoAlgorithm(int datum) {...}
```

```
{
    void run(int datum) { DoAlgorithm(datum); }

    virtual void DoAlgorithm() = 0; // abstract
}
```

In contrast, the **Strategy pattern** allows an algorithm to be chosen at *runtime* by *containment*. The concrete algorithms are implemented by separate classes or functions which are passed to the strategy as a parameter to its constructor or to a setter method. Which algorithm is chosen for this parameter can vary dynamically based on the program's state or inputs.

```
class ConcreteAlgorithm : IAlgorithm
{
    void DoAlgorithm(int datum) {...}
}

class Strategy
{
    Strategy(IAlgorithm algo) {...}

    void run(int datum) { this->algo.DoAlgorithm(datum); }
}
```

In summary:

- Template method pattern: **compile-time** algorithm selection by **subclassing**
- Strategy pattern: **run-time algorithm** selection by **containment**

edited May 7 '11 at 20:25

Jeff Axelrod
15.9k 25 127 225

answered Mar 21 '09 at 14:15

thehouse
4,376 4 23 32

42 Both patterns support runtime selection of the algorithm used (for Template Method, you would do something like `if (config.useAlgoA) impl = new AlgoA() else impl = new AlgoB()`) so this answer is incorrect. – Borek Bernard Jun 14 '10 at 8:15

13 Sure you could do that but then you're not using the Template Pattern. In fact, that's almost exactly what the code creating the Strategy instance will look like! – thehouse Jun 14 '10 at 13:23

1 Could I ask you, where did you get this information from? I'd say it's bogus. – Mykolas Simutis Oct 26 '11 at 12:24

Zam. That cleared up so much.....after years of actually one or the other from time to time. Just made it very clear. – granadaCoder May 1 '17 at 17:18

116

The template pattern is used when a particular operation has some invariant behavior(s) that can be defined in terms of other varying primitive behaviors. The abstract class defines the invariant behavior(s), while the implementing classes defined the dependent methods.

In a strategy, the behavior implementations are independent -- each implementing class defines the behavior and there is no code shared between them. Both are behavioral patterns and, as such, are consumed in much the same way by clients. Typically strategies have a single public method -- the `execute()` method, whereas templates may define a set of public methods as well as a set of supporting private primitives that subclasses must implement.

The two patterns could easily be used together. You might have a strategy pattern where several implementations belong to a family of strategies implemented using a template pattern.

edited Aug 3 '18 at 12:29

Yassin Hajaj

15.1k 8 30 62

answered Mar 21 '09 at 13:33

tvanfosson

436k 83 654 758

This sounds right to me, however why does WikiPedia mentions that "strategy pattern is for an algorithm's behavior to be selected at runtime"? It could also be used for selecting algorithm's behavior at compile time, just like the template method? Am I missing something? – BornToCode Jul 31 '15 at 16:01

2 @BornToCode I would assume what they are talking about is choosing a particular strategy at run time. For example, there are are several ways of numerically finding the roots of an equation. Depending on the problem domain or the data you might choose Newton-Raphson, Euler, or some other strategy for solving the equation. Each one of those is a strategy. The larger algorithm, of which solving the equation is one part, chooses the strategy to employ based on some quality of the problem. – tvanfosson Jul 31 '15 at 16:07

Yes, but it's not like strategy pattern should be used ONLY for those cases? I mean if I only need to select algorithm's behavior at compile time should I still use strategy pattern, or it wasn't meant to be used that way? – BornToCode Jul 31 '15 at 16:18

1 @BornToCode I would say that a strategy is most useful when the choice is dynamic. Template is basically a way of building up different, related behaviors for known. You'd use some strategy (though not the strategy pattern necessarily) for choosing which templated behavior to employ. For example, product inheritance - you'd create a base product, add features for different products. Choosing which product type (class) to instantiate might depend on which tables/views it's loaded from. Strategy pattern doesn't really come into play there. – tvanfosson Jul 31 '15 at 16:22

2 @BornToCode it's not an either/or thing, it's yes-and. Apply the pattern where it's appropriate, combine patterns where it's useful. – tvanfosson Jul 31 '15 at 16:34

22

when you have a "template" algorithm having defined steps where subclasses override these steps to change some details. In case of strategy, you need to create an interface, and instead of inheritance you are using delegation. I would say it is a bit more powerful pattern and maybe better in accordance to DIP - dependency inversion principles. It is more powerful because you clearly define a new abstraction of strategy - a way of doing something, which does not apply to template method. So, if this abstraction makes sense - use it. However, using template method may give you simpler designs in simple cases, which is also important. Consider which words fit better: do you have a template algorithm? Or is the key thing here that you have an abstraction of strategy - new way of doing something

Example of a template method:

```
Application.main()
{
Init();
Run();
Done();
}
```

Here you inherit from application and substitute what exactly will be done on init, run and done.

Example of a strategy:

```
array.sort (IComparer<T> comparer)
```

Here, when writing a comparer, you do not inherit from an array. Array delegates the comparison algorithm to a comparer.

answered Mar 21 '09 at 13:38

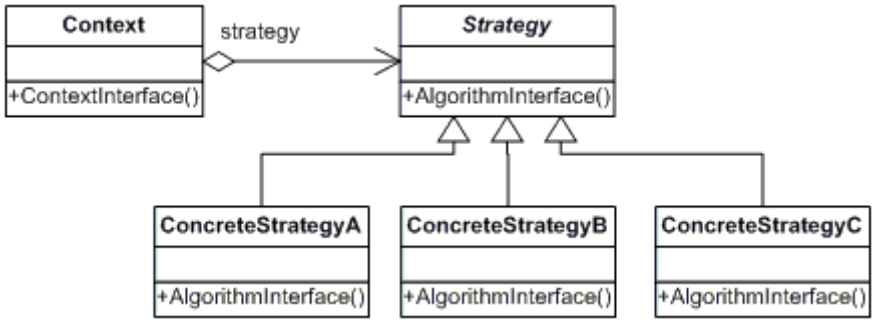
badbadboy
2,315 4 28 42

2 I think this is a great answer – Calanus May 11 '11 at 15:34

22

I think the Class-Diagrams of both pattern are showing the differences.

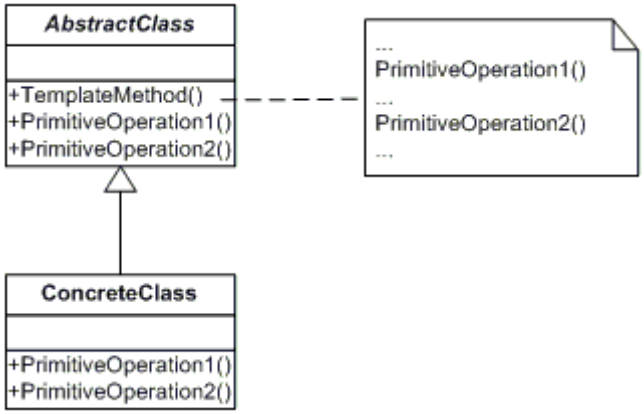
Strategy



Link to image

Template Method

Defer the exact steps of an algorithm to a subclass



Link to Image

edited Mar 8 '14 at 19:39

Mayank Jaiswal

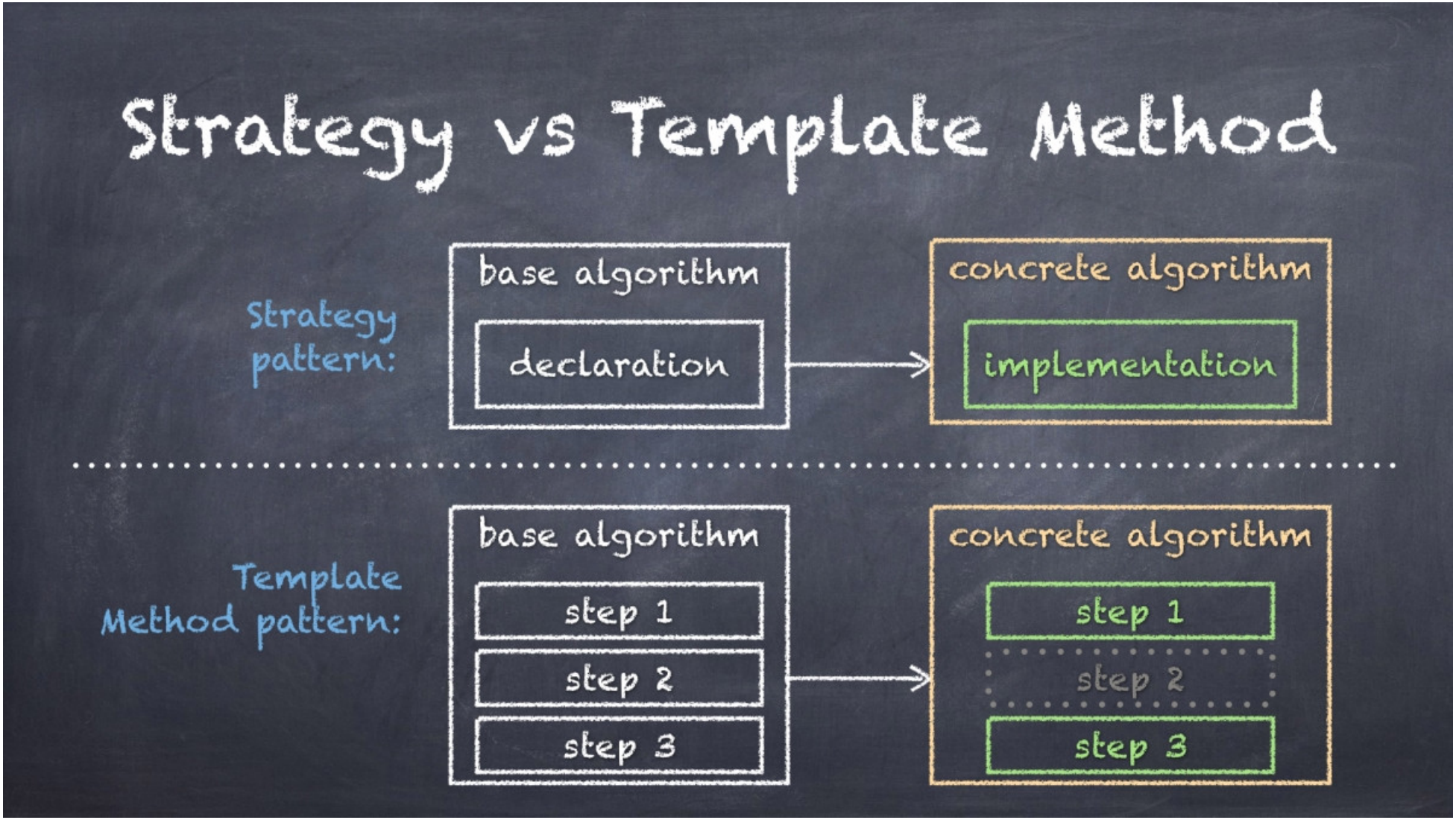
7,099 5 28 38

answered Mar 21 '09 at 15:21

Ludwig Wensauer

1,297 2 26 40

19



Similarities

Strategy and Template method patterns have a lot of similarities between them. Both Strategy and Template method patterns can be used for satisfying the Open-Closed Principle and making the software module easy to extend without changing its code. Both patterns represent

Differences

Here are some of the differences I have observed while studying these two patterns:

1. In Strategy, the coupling between the client and strategy is more loose whereas in Template Method, the two modules are more tightly coupled.
2. In Strategy, mostly an interface is used though abstract class can also be used depending on the situation, and concrete class is not used whereas in Template method mostly abstract class or concrete class is used, interface is not used.
3. In Strategy pattern, generally entire behaviour of the class is represented in terms of an interface, on the other hand, Template method is used for reducing code duplication and the boilerplate code is defined in base framework or abstract class. In Template Method, there can even be a concrete class with default implementation.
4. In simple words, you can change the entire strategy (algorithm) in Strategy pattern, however, in Template method, only some things change (parts of algorithm) and rest of the things remain unchanged. In Template Method, the invariant steps are implemented in an abstract base class, while the variant steps are either given a default implementation, or no implementation at all. In Template method, the component designer mandates the required steps of an algorithm, and the ordering of the steps, but allows the component client to extend or replace some number of these steps.

Image is taken from the bitesized blog.

edited Sep 3 '17 at 14:47

answered Sep 12 '16 at 14:02

Yogesh Umesh Vaity
6,148 4 35 38

16

Inheritance versus aggregation (is-a versus has-a). It's two ways to achieve the same goal.

This question shows some of trade-offs between choices: Inheritance vs. Aggregation

edited May 23 '17 at 12:18

Community ♦
1 1

answered Mar 21 '09 at 13:32

flicken
12.3k 4 23 26

10

Both are very similar, and both are consumed by the client code in similar ways. Unlike what the most popular answer above says, **both allow algorithm selection at run-time**.

The difference between the two is that while the *strategy pattern* allows different implementations to use completely different ways of the achieving the desired outcome, the *template method pattern* specifies an overarching algorithm (the "template" method) which is be used to achieve the result -- the only choice left to the specific implementations (sub-classes) are certain details of the said template method. This is done by having the the template method make call(s) to one or more *abstract* methods which are overridden (i.e. implemented) by the sub-classes, unlike the template method which itself is not abstract and not overridden by the sub-classes.

The client code makes a call to the template method using a reference/pointer of the abstract class type pointing to an instance of one of the concrete sub classes which can be determined at run time just like while using the Strategy Pattern.

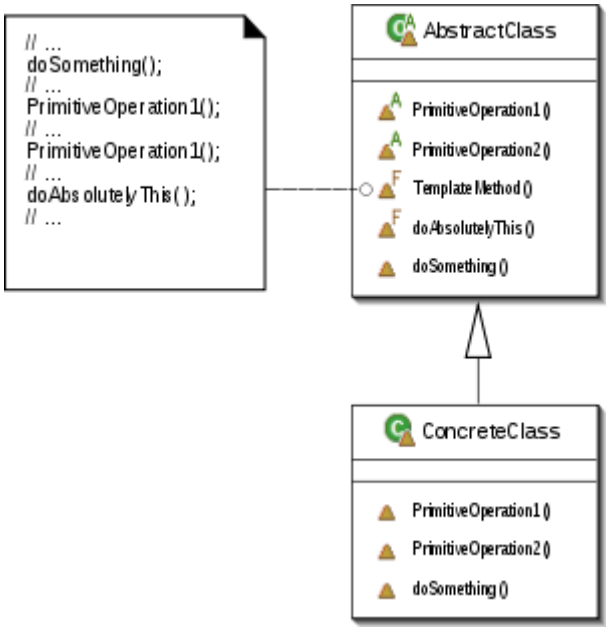
answered Jan 25 '12 at 20:03 community wiki Himanshu P

5

Template Method:

1. It's based on *inheritance*
2. Defines skeleton of algorithm which can't be changed by sub classes. Only certain operations can be overridden in sub classes
3. Parent class completely *controls the algorithm* and differs only certain steps to concrete classes
4. Binding is done at compile time

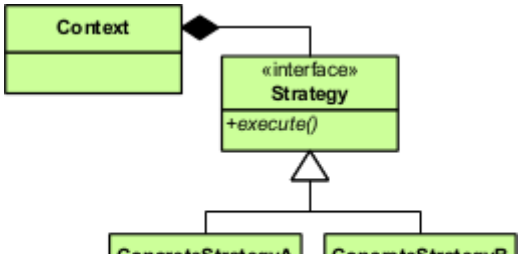
Template_method structure:



Strategy:

1. It's based on *delegation/composition*
2. *It changes guts of the object* by modifying method behaviour
3. It's used to *switch between family of algorithms*
4. It changes the behaviour of the object at run time by completely *replacing one algorithm with other algorithm at run time*
5. Binding is done at run time

Strategy structure:



Have a look at Template method and Strategy articles for better understanding.

Related posts:

Template design pattern in JDK, could not find a method defining set of methods to be executed in order

Real World Example of the Strategy Pattern

edited Sep 24 '17 at 4:00

answered Feb 10 '16 at 6:38

Ravindra babu

31.7k 6 172 144

3

No, they are not necessarily consumed in the same way. The "template method" pattern is a way of providing "guidance" to future implementers. You are telling them, "All Person objects must have a Social Security Number" (that's a trivial example but it gets the idea across correctly).

The strategy pattern allows **multiple** possible implementations to be switched in and out. It is not (usually) implemented through inheritance, but instead by letting the caller pass in the desired implementation. An example might be allowing a ShippingCalculator to be provided with one of several different ways of calculating taxes (a NoSalesTax implementation, and a PercentageBasedSalesTax implementation perhaps).

So, sometimes, the **client** will actually tell the object which strategy to use. As in

```
myShippingCalculator.CalculateTaxes(myCaliforniaSalesTaxImpl);
```

But the client would never do that for an object that was based on Template Method. In fact, the client might not even know an object is based on Template Method. Those abstract methods in the Template Method pattern might even be protected, in which case the client wouldn't even know they exist.

answered Mar 21 '09 at 13:47

Charlie Flowers

11.8k 7 63 82

I would suggest you to read this article. It explains the differences on a real case example.

"*As one can see implementing classes also depend upon the template method class. This dependency causes to change the template method if one wants to change some of the steps of the algorithm. On the other side strategy completely encapsulates the algorithm. it gives the implementing classes to completely define an algorithm. Therefore if any change arrives one does need to change the code for previously written classes. This was the primary reason I choose strategy for designing up the classes.*

One feature of template method is that template method controls the algorithm. Which can be a good thing in other situation but in my problem this was restricting me to design the classes. On the other side strategy does not control the steps of an algorithm which enables me to add completely different conversion methods. Hence in my case strategy helps me for implementation.

One drawback of strategy is that there is too much code redundancy and less code sharing. As it is obvious in the presented example of this article I have to repeat the same code in four classes again and again. Therefore it is hard to maintain because if the implementation of our system such as step 4 which is common to all is changed then I will have to update this in all 5 classes. On the other side, in template method, I can only change the superclass and the changes are reflected into the sub classes. Therefore template method gives a very low amount of redundancy and high amount of code sharing among the classes.

Strategy also allows changing the algorithm at run-time. In template method one will have to re-initialize the object. This feature of strategy provide large amount of flexibility. From design point of view one has to prefer composition over inheritance. Therefore using strategy pattern also became the primary choice for development."

answered Feb 28 '14 at 12:31

Iulian Rosca

625 3 12 24

2

The Template pattern is similar to the Strategy pattern. These two patterns differ in scope and in methodology.

Strategy is used to allow callers to vary an entire algorithm, like how to calculate different types of tax, while Template Method is used to vary steps in an algorithm. Because of this, Strategy is more coarsely grained. The Template allows finer-grained controls in the sequent of operations, and yet allows the implementations of these details to vary.

The other main difference is that Strategy uses delegation while Template Method uses inheritance. In Strategy, the algorithm is delegated to the another xxxStrategy class that the subject will have a reference to, but with Template you subclass the base and override methods to make changes.

from http://cyruscrypt.blogspot.com/2005/07/template-vs-strategy-patterns.html

iRunner
555 4 16 31

2

In strategy pattern subclasses are running the show and they control the algorithm. Here code is duplicated across the subclasses. The knowledge of the algorithm and how to implement it is distributed over many classes.

In template pattern, base class has algorithm. It maximizes the reuse among the subclasses. Since algorithm lies in one place, base class protects it.

answered Sep 5 '11 at 4:46

ruju
21 1

2

> Strategy Design Pattern

- Supports composition.
- Provides you the flexibility to change the behavior of object at runtime.
- Less coupling between the client code and the solution/algorithm code.

> Template Method Design Pattern

- Favours inheritance over composition
- Define algorithm in your base class. Individual pieces of algorithm can be customized in child classes.

answered Nov 10 '16 at 9:04

Mangu Singh Rajpurohit
6,142 2 36 52

1

Template method is about letting subclasses redefine certain steps of the algorithm, without changing the main structure and steps of the algorithm, defined in the base class. Template pattern usually uses inheritance, so a generic implementation of algorithms can be provided in the base class, which the subclass might choose to override if needed.

```
public abstract class RobotTemplate {
    /* This method can be overridden by a subclass if required */
    public void start() {
        System.out.println("Starting....");
    }

    /* This method can be overridden by a subclass if required */
    public void getParts() {
        System.out.println("Getting parts....");
    }

    /* This method can be overridden by a subclass if required */
    public void assemble() {
        System.out.println("Assembling....");
    }

    /* This method can be overridden by a subclass if required */
    public void test() {
        System.out.println("Testing....");
    }

    /* This method can be overridden by a subclass if required */
    public void stop() {
        System.out.println("Stopping....");
    }

    /*
     * Template algorithm method made up of multiple steps, whose structure and
     * order of steps will not be changed by subclasses.
     */
    public final void go() {
        start();
        getParts();
        assemble();
        test();
        stop();
    }
}
```

```
    private String name;

    public CookieRobot(String n) {
        name = n;
    }

    @Override
    public void getParts() {
        System.out.println("Getting a flour and sugar....");
    }

    @Override
    public void assemble() {
        System.out.println("Baking a cookie....");
    }

    @Override
    public void test() {
        System.out.println("Crunching a cookie....");
    }

    public String getName() {
        return name;
    }
}
```

Note in the above code, the go() algorithm steps will always be the same, but the subclasses might define a different recipe for performing a particular step.

Strategy Pattern:

Strategy pattern is about letting client selects concrete algorithms implementation at runtime. All algorithms are isolated and independent, but implement a common interface, and there is no notion of defining particular steps within the algorithm.

```
/**
 * This Strategy interface is implemented by all concrete objects representing an
 * algorithm(strategy), which lets us define a family of algorithms.
 */
public interface Logging {
    void write(String message);
}
```

```
 */
public class ConsoleLogging implements Logging {

    @Override
    public void write(String message) {
        System.out.println(message);
    }

}

/**
 * Concrete strategy class representing a particular algorithm.
 */
public class FileLogging implements Logging {

    private final File toWrite;

    public FileLogging(final File toWrite) {
        this.toWrite = toWrite;
    }

    @Override
    public void write(String message) {
        try {
            final FileWriter fos = new FileWriter(toWrite);
            fos.write(message);
            fos.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }

}
```

For full source code, check out my github repository.

answered Feb 16 '16 at 5:38

Saad
462 4 13

0

And the abstract implementation of the same interface AbstractMessageSource, which has common implementation of resolving messages and exposes resolveCode() abstract method so that sub-classes can implement them in their ways. AbstractMessageSource is an example of template method.

http://docs.spring.io/spring/docs/4.1.7.RELEASE/javadoc-api/org/springframework/context/support/AbstractMessageSource.html

answered Jun 22 '16 at 18:33

amolu

1 1

0

In the template method of this design pattern, one or more algorithm steps can be overridden by subclasses to allow differing behaviors while ensuring that the overarching algorithm is still followed(Wiki).

The pattern name Template method means what it is. Say we have a method CalculateSomething() and we want to template this method. This method will be declared in the base class a non virtual method. Say the method looks like this.

```
CalculateSomething(){
    int i = 0;
    i = Step1(i);
    i++;
    if (i> 10) i = 5;
    i = Step2(i);
    return i;
```

} Step1 and Step2 method implementation can be given by derived classes.

In Strategy Pattern there is no implementation provided by the base (This is the reason why the base is really an interface in the class diagram)

The classic example is sorting. Based on the number of objects needs to be sorted the appropriate algorithm class(merge, bubble, quick etc.) is created and the entire algorithm is encapsulated in each class.

Now can we implement the sorting as a template method? Certainly you can, but you wont find much/any commonality to be abstracted out and placed in the base implementation. So it defeats the purpose of template method pattern.

answered Jul 28 '16 at 13:12

protected by AZ_ Jul 9 '18 at 7:51

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?