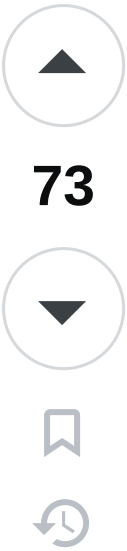
Strategy Pattern V/S Decorator Pattern

Asked 10 years, 2 months ago Modified 6 months ago

Viewed 30k times



I just came across two patterns.

1. Strategy Pattern
2. Decorator

> Strategy Pattern :-
>
> Strategy pattern gives several algorithms that can be used to perform particular operation or task.
>
> Decorator Pattern :-
>
> Decorator pattern adds some functionality to component.

In fact I have found that **Strategy** Pattern and **Decorator** Pattern can also be used interchangeably.

Here is the link :- When and How Strategy pattern can be applied instead of decorator pattern?

What is the difference between Strategy Pattern and Decorator Pattern?

when Strategy Pattern should be used and when Decorator Pattern should be used?

Explain the difference between both with the same example.

design-patterns decorator strategy-pattern

Share

Improve this question

Follow

edited Sep 9, 2017 at 17:07

Ravindra babu

38.9k ● 11 ● 256 ● 219

asked Oct 17, 2014 at 10:19

Nirav Kamani

3,262 ● 7 ● 40 ● 69

8 Answers

Sorted by: Highest score (default)

143

The strategy pattern allows you to *change* the implementation of something used at runtime.

The decorator pattern allows you augment (or add to) existing functionality with additional functionality at run time.

The key difference is in the *change* vs *augment*

In one of the questions you linked to it also points out that with the strategy pattern the consumer is aware that the different options exist, whereas with the decorator pattern the consumer would not be aware of the additional functionality.

As an example, imagine you are writing something to sort a collection of elements. So you write an interface `ISortingStrategy` you can then implement several different sorting strategies `BubbleSortStrategy`, `QuickSortStrategy`, `RadixSortStrategy`, then your application, based on some criteria of the existing list chooses the most appropriate strategy to use to sort the list. So for example if the list has fewer than 10 items we will use `RadixSortStrategy`, if fewer than 10 items have been added to the list since the last sort we will use `BubbleSortStrategy` otherwise we will use `QuickSortStrategy`.

We are changing the type of sort at runtime (to be more efficient based on some extra information.) this is the strategy pattern.

Now imagine someone asks us to provide a log of how often each sorting algorithm is used to do an actual sort and to restrict sorting to admin users. We can add both of these pieces of functionality by creating a decorator which enhances *any* `ISortingStrategy`. We could create a decorator which logs that it was used to sort something and the type of the decorated sorting strategy. And we could add another decorator that checks if the current user is an administrator before calling the decorated sorting strategy.

Here we are adding new functionality to *any sorting strategy* using the decorator, but are not swapping out the core sorting functionality (we used the different strategies to change that)

Here is an example of how the decorators might look:

```
public interface ISortingStrategy
{
    void Sort(IList<int> listToSort);
}

public class LoggingDecorator : ISortingStrategy
{
    private ISortingStrategy decorated;
    public LoggingDecorator(ISortingStrategy
decorated)
    {
         this.decorated=decorated;
    }

    void Sort(IList<int> listToSort)
    {
         Log("sorting using the strategy: " +
decorated.ToString();
         decorated.Sort(listToSort);
```

```
    }
}

public class AuthorisingDecorator :
ISortingStrategy
{
    private ISortingStrategy decorated;
    public AuthorisingDecorator(ISortingStrategy
decorated)
    {
         this.decorated=decorated;
    }

    void Sort(IList<int> listToSort)
    {
         if (CurrentUserIsAdministrator())
         {
             decorated.Sort(listToSort);
         }
```

Share Improve this answer

Follow

edited Aug 6, 2019 at 16:01

Crown

339 ● 2 ● 10

answered Oct 17, 2014 at 11:47

Sam Holder

32.9k ● 14 ● 108 ● 189

2 In order for these two classes to be decorators and for client calls to be transparent (calling against `ISortingStrategy` interface), they should implement `ISortingStrategy` as well as having it as a dependency – dragan.stepanovic Apr 9, 2016 at 11:52

For what it's worth, here is an example of initial confusion about what should be used to solve the problem (colleague suggested decorator/adapter) and the final solution (strategy). It's in C#, but is clearly an example of needing to change functionality at runtime between XML and JSON

serialization. stackoverflow.com/questions/40686736/… – jb62 Nov 21, 2016 at 18:01

14

Strategy_pattern

1. Defines a family of algorithms,
2. Encapsulates each algorithm, and
3. Makes the algorithms interchangeable within that family.

Use Strategy pattern when you have to change algorithm dynamically at run time.

Decorator

Decorator pattern dynamically changes the functionality of an object at runtime without impacting the existing functionality of the objects.

When to use:

1. Add additional functionalities/responsibilities dynamically
2. Remove functionalities/responsibilities dynamically
3. Avoid too much of sub-classing to add additional responsibilities.

Drawbacks:

1. Overuse of Open Closed principle (Open for extension and Closed for modification). Use this feature sparingly where the code is least likely changed.
2. Too many small classes and will add maintenance overhead.

Key difference:

Strategy lets you change the guts of an object. Decorator lets you change the skin.

Few more useful posts:

When to Use the Decorator Pattern?

Real World Example of the Strategy Pattern

strategy by sourcemaking

Share Improve this answer Follow

edited Sep 20, 2017 at 6:17

answered Sep 23, 2016 at 17:27

Ravindra babu

38.9k • 11 • 256 • 219

8

It boils down to this: Use Strategy to choose one piece of core functionality. Use Decorator to add extra functionality. Strategy is the cake we're making and Decorator is all the pretty decorations we're adding to it.

Share Improve this answer

Follow

answered Jul 8, 2015 at 18:26

kirk.burleson

1,251 2 18 30

7

The strategy is pattern is used to "encapsulate what changes". It allows you to define algorithms that can be interchanged at runtime. For example (example taken from Head First Design Patterns):

Say you have a a duck simulator. You want to make your duck objects fly. You could use inheritance for this but it gets messy quickly. Some ducks can't fly (e.g. rubber ducks). The way to do this is to encapsulate what changes i.e. the fly behaviour into its own class that implement IFlybehaviour for example. You can then use composition instead of inheritance and inject the IFlybehaviour into your duck objects. You could also then have a method that sets this fly behaviour so the fly behaviour can be changed at runtime. This is essentially the strategy pattern.

The decorator pattern is used to add functionality to an object at runtime. It allows you to wrap objects within objects. Decorators must have the same supertype as the objects they decorate. This allows you to call a method on the "outmost wrapper" and you can then chain call this same method down through the layers of decorators. It is basically a more flexible approach than subclassing.

As to which ones you choose depends on the problem you want to solve. Do you want to have a family of algorthims that are interchangeable at runtime or do you want to dynamically add more functionality to an object at run time?

The book "Head first design patterns" explains this quite well (probably much better than me) IMO so it is worth a read if you get chance.

Good luck.

Share Improve this answer

Follow

answered Oct 17, 2014 at 10:34

mbx-mbx

1,775 ● 11 ● 23

1 I am reading the same book. I also found the example of StarBuzz Coffee for decorator pattern. But still it is somewhat creating confusion. – Nirav Kamani Oct 17, 2014 at 10:39

can you elaborate the Decorator Pattern more it is the main confusing part? With some programmatic code example? – Nirav Kamani Oct 17, 2014 at 10:42

Does the confusion come from the patterns themselves or when to choose the pattern over the other? – mbx-mbx Oct 17, 2014 at 10:42

5

Strategy is choosing an algorithm.

```
Collections.sort(list, comparator); // comparator
```

```
is the strategy
```

Decorator is wrapping an object while keeping the same type.

```
in = new BufferedInputStream(in);
```

Share Improve this answer

Follow

answered Nov 19, 2018 at 9:18

Mark Jeronimus

9,533 ● 3 ● 40 ● 54

0

Decorator is to add extra functionality to an instance of a class, so it is implemented in runtime dynamically.It is a pattern of behavior extensions. Due to the practical conditions under which Subclassing is back breaking. Strategy is a pattern which different Strategy objects share the same abstract function, the concrete context object can be configured at run time with the concrete strategy object of a family set. As MVC, controller is a case of strategy role, the view restrain the output data structure, while-as the controllers will be different at runtime, which dynamically change the output data of view.

Share Improve this answer

Follow

answered Jan 24, 2018 at 3:58

YoungJeXu

21 ● 4

> Strategy is as a pattern of algorithm encapsulation.
> – YoungJeXu Jan 24, 2018 at 4:00

I would say strategy pattern allows you to choose one of multiple ways to do something, or one of multiple implementations for some operation, while decorator pattern allows you to create an object as it suits your needs by using desired object's (actually its class and/or interface(s)) functionalities and their implementations (whichever you want and how many times you want in order to create needed object).

Both are done dynamically, in other words at runtime.

Strategy pattern example

You are reading content of a folder. It contains plain *.txt*, *.csv* and *JSON* files. Call some method *Read()* from your *IReader* interface, but depending on type of the file, use appropriate implementation of *Read()* method. Use strategy pattern to decide which implementation.

Decorator pattern example

Export the result of *Read()* method to some *XML* file, but depending on the result, some `Header` node in *XML* can contain one, or more nodes and a node can differ in its content. `Header` may, or may not contain `Date` node, which may have one of several date formats; it may, or may not contain `ReadDuration` node, which can be

represented in millisecond, seconds, or minutes (with additional node that says which unit is used); it may, or may not contain a node that represents a quantity of items read, like `NumberOfItems`, or `NumberOfRows`, or something similar.

So some of the *XML* `Header` node examples contain:

- `Date` node where date format is YYYY-MM-DD and `ReadDuration` node which says how many seconds it took to read the file
- `Date` node where date format is DD-MM-YYYY
- `Date` node where format is YYYY-MM-DD, `ReadDuration` node which says how many millisecnods it took to read the file and `NumberOfRows` node
- `ReadDuration` node which says how many millisecnods it took to read the file and `NumberOfRows` node
- `ReadDuration` node which says how many seconds it took to read the file and `NumberOfRows` node
- `Date` node where format is YYYY-MM-DD, `ReadDuration` node which says how many millisecnods it took to read the file, `NumberOfRows` node and `NumberOfItems` node
- and so on

answered Sep 14, 2020 at 23:49


Marko Radivojević
448 ● 2 ● 7 ● 11

0

A simple summary of the other responses:

Decorator lets you change the skin of an object, while Strategy lets you change the guts.

Source: https://refactoring.guru/design-patterns/strategy

Share Improve this answer

Follow

answered Jan 24 at 15:49


krm
2,106 ● 3 ● 18 ● 21