Difference between decorator design pattern and visitor design pattern

Ask Question

I believe to understand the intent of Decorator and Visitor design pattern.

Though i can list following differences

1. Decorator works on an object, Visitor works on composite structure,
2. Decorator is Structural design pattern, visitor is Behavioral design pattern.

When i think deep down, i cannot convince myself what is the real difference between the two.

design-patterns decorator visitor

asked Feb 20 '12 at 15:25

Tilak

20.8k 10 58 112

8 Answers

Well, they are actually as different as they can be!

You use *Decorator* when you want to enhance existing object with some new, more-or-less transparent functionality like validation or caching. See example here: Should I extend ArrayList to add attributes that isn't null?

Visitor on the other hand is used when you have a hierarchy of classes and want to run different method based on concrete type but avoiding `instanceof` or `typeof` operators. See real-life example: Is This Use of the "instanceof" Operator Considered Bad Design?

> Decorator works on an object, Visitor works on composite structure,

Visitor works on an inheritance hierarchy, *Composite* is a different GoF design pattern.

> Decorator is Structural design pattern, visitor is Behavioral design pattern.

True, but it doesn't really help in understanding how they work?

See also

- Examples of GoF Design Patterns in Java's core libraries
- What's a suitable return type for this Java method?

edited May 23 '17 at 12:09

Community ♦
1 1

answered Feb 20 '12 at 15:32

Tomasz Nurkiewicz
255k 45 576 593

Design patterns are not meant to be categorized by the implementation differences but by when you should use one or the other one.

They serve for absolutely different purposes:

- you will use decorator when you want to dynamically enrich the functionality of objects by providing single elements that decorate other objects so that they do actually add some behavior to them (in fact it is a ***structural*** pattern in the sense that it alters the structure of the objects you are woking with)
- you will use visitor when you want to separate an algorithm from the objects it is used with. What you do is that you have this visitor which is passed to a multitude of different objects, usually a hierarchy, (they are said to *accept* the visitor), this visitor does specific operations according to the type of objects it is visiting in the specific moment. In this way you can have your visitor do whatever it wants with the specific objects without the need to specify these operations in the objects itself (that's why it is ***behavioral***). It's a sort of having abstract methods that are not defined in the object itself.

answered Feb 20 '12 at 15:35
Jack

105k 25 175 283

I like to think that decorator allows to avoid inheriting and then extending the class as is the general principle of OOP to prefer aggregation over inheritance, although you do inherit in a way. Here is a overly simplistic example

```
abstract class Chef{
  public abstract void Prepare();
}

class CookieMaker:Chef{         //Concrete class
   public override void Prepare()
    {
        //Bake in Oven
     }
 }

  // Decorator class
 // This chef adds chocolate topping to everything
 class ChocoChef:Chef{

     public ChocoChef(Chef mychef)
     {
        this.chef = mychef;
     }

     public override void Prepare()
     {
           // Add chocolate topping first
           chef.Prepare()
     }
 }
```

I'v cut short some details for the sake of space. For example, you could abstract out a chef which adds any kind of topping and ChocoChef then becomes its concrete class. Now ChocoChef always adds chocolate toppings no matter what you're preparing. So now you can have either chocolate cookies or chocolate cake by passing the correponding Chef to its constructor. The visitor on the other hand acts on objects and decides to do something based on the object that it is visiting.

```
class Student{
     // Different visitors visit each student object using this method
     // like prize distributor or uniform inspector
     public Accept(IVisitor v)
```

```
    {
        v.Visit(this)
    }
}

 // Visitor visits all student OBJECTS
class PrizeDistributor:IVisitor{
    public override void Visit(Student s)
    {
          //  if(s has scored 100)
          // Award prize to s
    }
}
```

answered Feb 20 '12 at 17:49

Swapnil
172 1 7

The way I interpret it, visitors represent actions that we may want to take on or with an object, but that are not necessarily inherent to an object, and are rather horizontal in relationship. For example, I could "make a marketing pitch" for a car, but I wouldn't program a car object to have a "createMarketingPitch" function because that would be a slippery slope towards creating many many functions on my car object.

On the other hand, a decorator is a pattern that layers functionality on top of an existing object, a vertical relationship that modifies how the object behaves when its normal functions are called. Additionally, whereas a visitor is coded to work with a class of objects, decorators can be assigned to specific instances of the object so that different instances of the same type behave differently than each other.

edited Feb 21 '12 at 23:14

answered Feb 21 '12 at 7:24

arieljake
348 4 10

+1 For describing the visitor pattern for "horizontal relationship", that helped me to visualize when and why you would use it. – ryanp102694 Apr 26 '17 at 9:26

They both "add functionality" to an existing object, without modifying the original class. The difference is:

With decorator You add functionality that wraps basic functionality that this object has (e.g. in addition to perform some basic action also write it to log, in addition to write a file to disk also encrypt it). This also allows us to create different combinations of decorators without subclassing each possible scenario.

With visitor You add a completely new behavior that you don't want to define as part of the basic component class itself (not even as a wrapper to basic functionality), for example because of single responsibility principle, open close principle etc. It's especially useful when that behavior will be different between different subclasses of the same type (if there isn't any complex structure of subclasses but just one class you could just create a new class and include the original class via composition and still achieve the goal of not affecting or modifying the original class). That way you can avoid code like `if (a is ConcreteClass1) {...} else if (a is ConcreterClass2) {...}` without writing virtual methods.

As a result of this difference, with decorator the client code calls the same method which is defined on the interface of the basic component class, it's just "decorated" with extra functionality now, while with visitor the client calls some general "accept" method and sends a visitor to it.

edited Sep 18 '16 at 19:11

answered Sep 16 '16 at 11:59

BornToCode

4,550 5 38 52

The way i think of decorator pattern is when i want to add functionality to an object at runtime. I can add behavior to an object while the program is running by wrapping the object in a decorator class which can extend its methods.

For the visitor pattern i like it when i have to do an operation on "a set" of objects of the same type and collect the information. Lets say i have 10 concrete classes of vegetables and i want to know the total price for all 10. i can use the visitor pattern to "visit" each vegetable object and at the end of the iteration i'd have the total price. Of course you can also use the pattern as a way to decouple some operation from the object(s).

answered Sep 21 '16 at 19:44

j2emanue

18.1k 15 115 214

Yes, they both add some functionality to the existing system at run-time and try to be less reactive (in a good meaning) to dynamic changes, but with some differences.

Visitor is primarily to respect OCP (and sometimes to SRP), to make the system more flexible. You can add whatever Visitor you can as your program evolves without changing the existing system. However, you need to design the system in such a way **beforehand**. You can't add a new Visitor class (or pattern) to already running system and expect it to work without re-compiling, re-testing or whatsoever.

On the other hand, you can use Decorator to enrich **the existing system** functionality by wrapping the abstract base class (that you already have) in a Decorator and provide the enriched feature of yours as a separate object so you can you can create as you need. Moreover, semantically, Decorator rather refers to appearance of sth.

Which one to prefer? IMO, answering this might be more helpful. For me, I don't like the Decorator's way of use the base class. It both uses inheritance and aggregation. If you need to change this (wrapee) class, you end up with re-compiling entire hierarchy/module. But it's handy since and you can change the behavior after **design time**. On the other hand, in Visitor pattern, I don't like the idea of knowing each concrete types in the Visitor implementation. When you add a new base class type, you also need to go and change the Visitor class to add it. But it's useful when you need to **inject** a code to the existing system without altering the structure or you need to separate concerns in a class (Single-User Resp).

Finally, what makes Visitor better than regular inheritance? Depends. Using inheritance you will be more dependent to interface signature. Using Visitor makes your Visitor class dependent to concrete classes. Not to mention you add more behaviours using visitor without changing existing module signature instead of implementing new interfaces in the existing class.

edited Jan 24 '17 at 8:05

answered Nov 1 '16 at 7:53

zgulser

733 1 9 19

Decorator

The **Decorator** pattern can be used to extend (decorate) the functionality of a certain object statically, or in some cases at run-time, independently of other instances of the same class, provided some groundwork is done at design time

When to Use Decorator pattern?

1. Object responsibilities and behaviours should be dynamically added/removed
2. Concrete implementations should be decoupled from responsibilities and behaviours

3. When sub - classing is too costly to dynamically add/remove responsibilities

Related post:

When to Use the Decorator Pattern?

Visitor:

Visitor design pattern is a way of separating an algorithm from an object structure on which it operates. A practical result of this separation is the ability to add new operations to existing object structures without modifying those structures. It is one way to follow the open/closed principle.

When to use Visitor pattern?

1. Similar operations have to be performed on objects of different types grouped in a structure
2. You need to execute many distinct and unrelated operations. It separates Operation from objects Structure
3. New operations have to be added without change in object structure
4. Gather related operations into a single class rather than force you to change or derive classes
5. Add functions to class libraries for which you either do not have the source or cannot change the source

Related post:

When should I use the Visitor Design Pattern?

Useful links:

sourcemaking decorator article

oodesign visitor article

sourcemaking visitor article

edited Sep 24 '17 at 20:23

answered Sep 23 '16 at 6:14

Ravindra babu

26k 5 134 127