DZone

How I Used the Decorator Pattern to Solve my Tech Issue

by Shuhail Kadavath · Nov. 10, 18 · Java Zone · Tutorial

Start coding something amazing with our library of open source Cloud code patterns. Content provided by IBM.

Our code base followed a hybrid monolithic approach where we had a huge codebase with tens of thousands of classes. On a complete build of the code base, we had five major JARs builds with dependencies pulled.

Now, I was working on the module `AccountsManager` . We had one another module, `UpgradeManager` , which allowed us to create a self-upgrade of all the other modules, including our `AccountsManager` , whereby it installs the latest RPM when its given a new RPM path (it does dozens of other things that are out of our scope).

So, we had the class `UpgradeServiceImpl` which can be seen as follows in the `UpgradeManager` :

```
class UpgradeServiceImpl implements UpgradeService {

// So many things here

public UpgradeStatus upgrade(Spec spec){

//some more things here

initiateUpgrade();
}

}
```

In my module, `AccountsManager` rest controller, it was used as below:

```
@RestController("/accounts")
class AccountsManagerUpgrade {

@Autowired UpgradeService upgradeService;

//So many things here


@PostMapping("/upgrade")
```

```
@PostMapping("/upgrade")
public void upgrade(Spec spec) { //spec contains path of new RPM , version etc

//somethings here includin validation etc

return upgradeService.upgrade();

}
}
```

Problem Statement

We had to introduce more logic in the `UpgradeService` of the `UpgradeManager` where more conditions had to be met before saying it's upgraded completely .

We tried this a couple of ways. First, we introduced an `OptionalBean` in `UpgradeService` and only our module creates and loads the bean. This for sure was working, but as you know, it's not always good to touch some one else' code.

So, we came up with a decorated design where we use the original `UpgradeService` without any change, but we decorated it as per our desire. Let's take a closer look.

Step 1

We created an exact replica (only the methods) as that of the `UpgradeService` in our module .

```
class AccountsManagerUpgradeStatusService {

 public UpgradeStatus upgrade(Spec spec) {

 }
 }
```

Step 2

Next, we add the original `UpgradeService` as a dependency in the new class and call the original upgrade method from the `UpgradeService` :

```
class AccountsManagerUpgradeStatusService {

 @Autowired UpgradeService upgradeService;

 public UpgradeStatus upgrade(Spec spec) {

 upgradeService.upgrade(spec);
```

```
}
}
```

Now, we can safely switch the original `UpgradeService` from the `RestController` to the new `AccountsManagerUpgradeStatusService` as shown below:

```
@RestController("/accounts")
class AccountsManagerUpgrade {

//@Autowired UpgradeService upgradeService; - removed this

@Autowired AccountsManagerUpgradeStatusService upgradeService; //added this

//So many things here


@PostMapping("/upgrade")
public void upgrade(Spec spec) { //spec contains path of new RPM , version etc

//somethings here includin validation etc

return upgradeService.upgrade();

}
}
```

Step 3

Now, in the new `AccountsManagerUpgradeStatusService` , we can add it as per our requirement:

```
class AccountsManagerUpgradeStatusService {

 @Autowired UpgradeService upgradeService;

 public UpgradeStatus upgrade(Spec spec) {

 //we can do many things her now
 UpgradeStatus status = upgradeService.upgrade(spec);

 //do some more validation/RPM upgrade check , here and return the status

 return status;

 }
 }
```

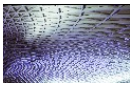
```
15 }
```

As you can see, we can now safely add more validations or checks after we get the status and our upgrade is complete by not touching the original `Upgrade` code.

Happy coding!

Use this tool to look at the contents of GitHub and classify code based on the programming language used. Content provided by IBM Developer.

Like This Article? Read More From DZone



Decorator Design Pattern in Java

Decorator Design Pattern in Java

Toss Out the Inheritance Forest

Free DZone Refcard
Java Containerization

Topics: DECORATOR DESIGN PATTERN , DESIGN PATTERN , SPRING BOOT 2.0 , JAVA , TUTORIAL

Opinions expressed by DZone contributors are their own.

ava Partner Resources

ep insight into your code with IntelliJ IDEA.
tBrains
|
e Build vs. Buy Challenge: Insight Into a Hybrid Approach
elissa
|
ee eBook: SCA Maturity Model - A Framework for Open Source Security and Compliance
exera
|
grating to Microservice Databases
d Hat Developer Program
|