Flexible Decorator Pattern?

Asked 16 years, 2 months ago Modified 14 years, 9 months ago

Viewed 2k times

2

I was looking for a pattern to model something I'm thinking of doing in a personal project and I was wondering if a modified version of the decorator patter would work.

Basicly I'm thinking of creating a game where the characters attributes are modified by what items they have equiped. The way that the decorator stacks it's modifications is perfect for this, however I've never seen a decorator that allows you to drop intermediate decorators, which is what would happen when items are unequiped.

Does anyone have experience using the decorator pattern in this way? Or am I barking up the wrong tree?

Clarification

To explain "Intermediate decorators" if for example my base class is coffe which is decorated with milk which is decorated with sugar (using the example in Head first design patterns) milk would be an intermediate decorator as it decorates the base coffee, and is decorated by the sugar.

Yet More Clarification :)

The idea is that items change stats, I'd agree that I am shoehorning the decorator into this. I'll look into the state bag. essentially I want a single point of call for the statistics and for them to go up/down when items are equiped/unequiped.

I could just apply the modifiers to the characters stats on equiping and roll them back when unequiping. Or whenever a stat is asked for iterate through all the items and calculate the stat.

I'm just looking for feedback here, I'm aware that I might be using a chainsaw where scissors would be more appropriate...

design-patterns **decorator**

edited Oct 9, 2008 at 15:23

asked Oct 9, 2008 at 9:19

Omar Kooheji

55.6k ● 71 ● 186 ● 243

what do you mean by intermediate decorators? – cruizer Oct 9, 2008 at 9:28

8 Answers

Sorted by: Highest score (default)

2

To be honest, it sounds like you're really trying to fit a pattern where you don't really need one, just for the sake of using a pattern. Don't be that guy.

Now, if the weapons gave the character some extra strength/stam/hp or whatever, then it might be worth considering. But it doesn't sound like you're going to be modifying (or decorating) the properties of the character at all with these.

Share Improve this answer Follow

edited Oct 9, 2008 at 13:28

answered Oct 9, 2008 at 11:32

Joe Phillips

51k ● 33 ● 108 ● 165

> The Idea was for Items to be able to affect any stats. which is why I went for the decorator. – Omar Kooheji Oct 9, 2008 at 15:18

2

I see what you're trying to do, but one of the things to remember about patterns is that you shouldn't try to shoe-horn your design to fit a pattern. Patterns occur naturally - the behavior you're describing isn't really part of the Decorator Pattern.

With that said, I'd imagine that you're going to want to unequip a weapon via some unique ID, say:

```
Character.unequip(LIGHTSABER);
```

If you'd try to fit this into the Decorator Pattern, you'd have to keep track of the currently equipped items and then, after removing a weapon, you'd have to update the reference of the object decorating the LIGHTSABER to the one LIGHTSABER is decorating. That's a lot of work.

Instead, perhaps it's worth considering @Mitch's idea and let the character's weapons be help in a property bag. Remember that a character HAS-A set of weapons. To me, it seems like composition may be the way to go.

Share Improve this answer

Follow

answered Oct 9, 2008 at 11:39

Tom

15.9k ● 5 ● 50 ● 63

1

There are three answers to implementing propegated stats in a video game:

(1) this will satisfy any hobbist-game you will ever make (and pretty much every professional game as well):

```
character.GetStrength() {
  foreach(item in character.items)
    strFromItems +=
item.GetStrengthBonusForItems();
       foreach(buff in character.buffs)
    strFromBuffs +=
```

```
buff.GetStrengthBonusForBuffs();
  ...

  return character.baseStrength + strFromItems +
...;
}
```

(note the different GetStrength*() functions have nothing to do with each other)

(2) this will satisfy all games that don't have the word 'diablo' in the title:

```
 character.GetStr() { ... // same as above,
strength is rarely queried }
 character.GetMaxHP() {
   if (character._maxHPDirty) RecalcMaxHP();
   return character.cachedMaxHP;
 }
 // repeat for damage, and your probably done, but
profile to figure out
 // exactly which stats are important to your game
```

(3) else

```
 // changes in diablo happen very infrequently
compared to queries,
 // so up propegate to optimize queries.
Moreover, 10 people edit
 // the stat calculation formulas so having the up
propegation match
 // the caculation w/o writing code is pretty
important for robustness.

 character.OnEquip(item) {
     statList.merge(item.statlist);
 }
```

```
 character.GetStrength() {
     statList.getStat(STRENGTH);
 }

 statlist.getStat(id) {
     if (IS_FAST_STAT(id)) return
cachedFastStats[id];
     return cachedStats.lookup(id);
 }

 statlist.merge(statlist) {
      // left for an exercise for the reader
 }
```

And honestly (3) was probably overkill.

Share Improve this answer Follow

edited Oct 11, 2008 at 21:29

answered Oct 11, 2008 at 19:15

Jeff

1,053 ● 1 ● 8 ● 14

1

Hmmm.. I'm thinking that maybe a command pattern would be a good solution to this problem. Here's what I mean:

This is your character class:

```
Public class Character {

 //various character related variables and methods
here...

 Command[] equipCommands;
 Command[] unequipCommands;
```

```
 public Character(Command[] p_equipCommands,
Command[] p_unequipCommands) {

  equipCommands = p_equipCommands;
  unequipCommands = p_unEquipCommands;
 }

 public void itemEquiped(int itemID) {

  equipCommands[itemID].execute(this);
 }

 public void itemUnequiped(int itemID) {

  unequipCommands[itemID].execute(this);
 }
}
```

Here are some examples of commands:

```
public class SwordOfDragonSlayingEquipCommand
implements ItemCommand{

 public void execute(Character p_character) {

  //There's probably a better way of doing this,
but of the top of my head...

p_character.addItemToInventory(Weapons.getIteM(Weapc

  //other methods that raise stats, give bonuses
etc. here...
 }
}

public class SwordOfDragonSlayingUnequipCommand
implements ItemCommand{

 public void execute(Character p_character) {

  //There's probably a better way of doing this,
```

```
but of the top of my head...

p_character.removeItemFromInventory(Weapons.getIteM(

  //other methods that lower stats, remove bonuses
etc. here...
 }
}
```

Of course, this is just a suggestion and definitely open for debate, I'm not saying that this is the best or the only way to do this...

Share Improve this answer

Follow

edited Oct 12, 2008 at 12:50

answered Oct 11, 2008 at 12:33

Sandman

9,682 ● 8 ● 38 ● 41

1

Just keep 2 sets of stats, your base stats and your effective stats. When you equip or unequip an item add or subtract from the effective stats where appropriate. Then you don't have to traverse your equipment list every time you want to know what your stats are.

Share Improve this answer

Follow

answered Oct 12, 2008 at 13:18

Gerald

23.5k ● 10 ● 76 ● 105

1

I know this question is old, but this might help somone else if not the OP. Read this article, to understand how this sort of thing really should be done in games (by one of the devs who worked on the Tony Hawk games):

http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/

Compose your entities/game objects. For constructing entity behaviours in games, never, NEVER rely on inheritance, or on anything that itself relies on inheritance in any way -- this includes the decorator pattern as the OP suggested. You will be tying your own hands. Composition is THE way to go.

Share Improve this answer

Follow

answered Mar 24, 2010 at 23:26

Engineer

8,837 ● 7 ● 67 ● 112

0

Are you looking for the Strategy pattern?

Share Improve this answer

Follow

answered Oct 11, 2008 at 20:52

Thomas Eyde

3,944 ● 3 ● 26 ● 33

Why not code the weapons as follows:

1 = chainsaw 2 = shotgun 4 = rail gun

So now the sum of 6 can only mean that the character possesses the shotgun and the rail gun. This is a fast summary so you do not have to iterate through your list of dictionary of weapons. You still need some structure to contain the weapons, but at least you'll get speed with this approach. This presupposes that you can have only one weapon of each category, but many categories simultaneously.

0

Share Improve this answer

Follow

answered Oct 13, 2008 at 0:45

David Robbins

10k ● 7 ● 54 ● 83