



## BufferedInputStream

Java Core (/quests/QUEST\_JAVA\_CORE)  
8 уровень (/quests/lectures/?quest=QUEST\_JAVA\_CORE&level=8), 6 лекция (/quests/lectures/questcore.level08.lecture06)

ОТКРЫТА

— Привет, Амиго! Сегодня я расскажу тебе немного интересных вещей про класс `BufferedInputStream`, но начнем мы с «обертки» и «мешка сахара».

— Это что еще за «обертка» и «мешок сахара»?

— Это метафоры. Слушай. Итак...

Паттерн проектирования «Обёртка» (Wrapper или Decorator) – это довольно простой и удобный механизм расширения функциональности объектов без использования наследования.

Обертка закрывает доступ к оригинальному объекту, "оборачивая" его.



Также обертка может "прозрачно" добавит новую функциональность.

Пусть у нас есть класс `Cat` с двумя методами `getName` и `setName`:

### Код на Java

### Описание

```
1 class Cat
2 {
3     private String name;
4     public Cat(String name)
5     {
6         this.name = name;
7     }
8     public String getName()
9     {
10        return this.name;
11    }
12    public void setName(String name)
13    {
14        this.name = name;
15    }
16 }
```

Класс `Кот(Cat)` имеет два метода: `getName` & `setName`

```
1 public static void main(String[] args)
2 {
3     Cat cat = new Cat("Васька");
4
5     printName(cat);
6 }
7
8 public static printName(Cat cat)
9 {
10    System.out.println(cat.getName());
11 }
```

Пример использования.

В консоль будет выведена строка «Васька».

Допустим нам нужно перехватить вызов методов у объекта `cat` и, возможно, внести туда небольшие изменения. Для этого нам понадобится *обернуть* его в свой класс-обертку.

Если мы хотим «обернуть» вызовы методов какого-то объекта своим кодом, то нам нужно:

- 1) Создать свой класс-обертку и унаследоваться от того же класса/интерфейса что и оборачиваемый объект.
- 2) Передать оборачиваемый объект в конструктор нашего класса.
- 3) Переопределить все методы в нашем новом классе, и вызвать в них методы оборачиваемого объекта.
- 4) Внести свои изменения «по вкусу»: менять результаты вызовов, параметры или делать что-то еще.

В примере ниже мы перехватываем вызов метода `getName` у объекта `cat` и немного меняем его результат.

#### Код на Java

#### Описание

```
1 class Cat
2 {
3     private String name;
4     public Cat(String name)
5     {
6         this.name = name;
7     }
8     public String getName()
9     {
10        return this.name;
11    }
12    public void setName(String name)
13    {
14        this.name = name;
15    }
16 }
```

Класс Кот(Cat) содержит два метода – получить имя и установить имя.

```
1 class CatWrapper extends Cat
2 {
3     private Cat original;
4     public CatWrapper (Cat cat)
5     {
6         this.original = cat;
7     }
8
9     public String getName()
10    {
11        return "Кот по имени " + original.getName();
12    }
13
14    public void setName(String name)
15    {
16        original.setName(name);
17    }
18 }
```

Класс-обертка. Класс не хранит никаких данных, кроме ссылки на оригинальный объект.

Класс в состоянии «пробрасывать» вызовы оригинальному объекту (setName), переданному ему в конструкторе. А также «перехватывать» эти вызовы и модифицировать их параметры и результаты.

```
1 public static void main(String[] args)
2 {
3     Cat cat = new Cat("Васька ");
4     Cat catWrap = new CatWrapper (cat);
5     printName(catWrap);
6 }
7
8 public static printName(Cat named)
9 {
10    System.out.println(named.getName());
11 }
```

Пример использования.

В консоль будет выведена строка  
«Кот по имени Васька».

Т.е. мы тихонечко подменяем каждый оригинальный объект на объект-обертку, в который уже передаем ссылку на оригинальный объект. Все вызовы методов у обертки идут к оригинальному объекту, и все работает как часы.

— Мне понравилось. Решение несложное и функциональное.

— Еще я расскажу тебе про «мешок сахара», но это не паттерн, а метафора. Метафора к слову буфер и буферизация. Что же такое буферизация и зачем она нужна?



Допустим, сегодня очередь Риши готовить, а ты ему помогаешь. Риши еще нет, а я хочу выпить чай и прошу тебя принести мне ложечку сахара. Ты пошел в подвал, там стоит мешок с сахаром. Ты можешь принести мне целый мешок, но мешок мне не нужен. Мне нужна только одна ложка. Тогда ты, как хороший робот, набрал одну ложку и принес мне. Я добавила ее в чай, но все равно не очень сладко. И я попросила у тебя еще одну. Ты опять сходил в подвал и принес еще ложку. Потом пришла Элли, и я попросила тебя принести сахара для нее... Это все слишком долго и неэффективно.

Пришел Риша, посмотрел на все это и попросил тебя принести ему полную сахарницу сахара. Потом я и Элли стали просить сахар у Риши. Он просто давал его нам из сахарницы, и все.

То, что произошло после появления Риши называется буферизацией, а сахарница – это буфер. Благодаря буферизации «клиенты» могут читать данные из буфера **маленькими порциями**, а буфер, чтобы сэкономить время и силы, читает их из источника **большими порциями**.

— Классный пример, Ким. Я все понял. Просьба ложки сахара – это аналог чтения из потока одного байта.

— Да. Класс `BufferedInputStream` – классический представитель обертки-буфера. Он – класс-обертка над `InputStream`. При чтении данных из него, он читает их из оригинального `InputStream`'а большими порциями в буфер, а потом отдает из буфера потихоньку.

— Отлично. Все понятно. А буферы для записи бывают?

— Да, конечно.

— А можно пример?

— Представь себе мусорное ведро. Вместо того, чтобы каждый раз ходить выбрасывать мусор на улице в дезинтегратор, ты просто выкидываешь его в мусорное ведро. А Скарафи раз в две недели выносит его на улицу. Классический буфер.



— Как интересно. И гораздо понятнее, кстати, чем с мешком сахара.

— А метод `flush()` – это вынести мусор немедленно. Можно использовать перед приходом гостей.

## Комментарии (35)

популярные

новые

старые

Никита

Natalie Silina 18 уровень

суббота, 17:54

...

Вот по этой ссылке можно посмотреть объяснение паттерна Decorator с диаграммой и примером  
<https://www.youtube.com/watch?v=4BeEPtpJdHg&index=1&t=525s&list=PLNGfn2X7X7y7hEUyssCt1L7Xaw2G8daVi>  
Смотреть с 1:00:20  
Лектор объясняет очень доступно

Ответить

0

Maxim Sivov 20 уровень, Taraz

4 апреля, 13:31

...

Здесь плохой пример привели. Ключевое понятие написали:  
1) Создать свой класс-обертку и унаследоваться от того же класса/интерфейса что и оборачиваемый объект.  
А на деле class - CatWrapper extends Cat  
Простое наследование, где достаточно переопределить getName()

Ответить

+1

Redas Shuliakas 21 уровень

9 февраля, 11:25

...

Ответить

+2

Victor 29 уровень

8 января, 14:27

...

Ответить

+5

Джонни 23 уровень

8 января, 11:41

...

Опять пример из лекции не компилируется в идее.  
Требуется вызвать оригинальный конструктор с именем кота. Как это сделать? Не пойму.



Ответить

0

Дмитрий Денисов 21 уровень, Санкт-Петербург

18 января, 21:48

...

Ответить

+1

Джонни 23 уровень

19 января, 13:50

...

Не совсем, у кота уже есть имя, поэтому его нужно получить.

Ответить

+1

Fs Jt 20 уровень, Киев

25 февраля, 12:54

...

так идея же и подсказывает, что можно делать.  
я создал конструктор по умолчанию для кота, например.

Ответить

0

Сергей 40 уровень

2 января, 14:02

...

У "CatWraper" вот здесь не понятно :  
"Класс-обертка. Класс не хранит никаких данных, кроме ссылки на оригинальный объект."  
CatWraper наследуется от Cat, значит содержит все поля класса Cat, а значит кроме ссылки есть еще  
"private String name;"  
Или про него надо просто забыть?

Ответить

0

Дмитрий Дедков 26 уровень

6 января, 15:36

...

ссылка на объект, который хранит имя в своём поле. В понятие объект входит его содержимое.

Ответить

0

**Дмитрий Денисов** 21 уровень, Санкт-Петербург

18 января, 21:42

а зачем wrapper экстендит кота?  
и что будет, если во wrapper не реализовать, например, setName. Что будет вызвано catWrapper.setName("бла")? setName суперкласса? Но, что там будет сидеть, если конструктор суперкласса не вызывался?  
Без super(cat.name); в конструкторе wrapper, код вообще не компилируется.

Какой-то гибрид наследования и композиции

Ответить

+1

**Дмитрий Дедков** 26 уровень

19 января, 07:54

чтобы переопределить методы и сделать свою реализацию, если нужно (вроде так). Тут лучше, как обычно, задачи порешать)  
Конструктор суперкласса действительно надо вызывать...

Ответить

0

**Илья** 28 уровень

20 декабря 2017, 06:07

<https://www.youtube.com/watch?v=X7-3wQEIWd4> - добавлю еще это видео

Ответить

+3

**armstel** 22 уровень

3 ноября 2017, 13:27

<https://www.youtube.com/watch?v=5CfXk62siuE> доходчиво объяснил декоратор

Ответить

+7

**NastyaGermanovich** 28 уровень, Санкт-Петербург

16 октября 2017, 19:49

Паттерн проектирования «Обёртка» (Wrapper или Decorator) – это довольно простой и удобный механизм расширения функциональности объектов без использования наследования.(с)  
Получается, что мы все равно наследуем класс-обертку от класса оборачиваемого объекта, в строке выше ошибка или я что-то не понимаю? Подскажите, пожалуйста.

Ответить

0

**Лайт** 21 уровень, Санкт-Петербург

17 октября 2017, 19:28

Если я правильно понял, то мы наследуем класс-обертку от того же класса, от которого унаследован класс объекта, который оборачиваем (и реализуем все интерфейсы, которые он реализует). В противном случае у нашей обертки не будет методов, которые класс объекта наследует от родителя и мы не сможем реализовать весь его потенциал.

Ответить

+1

**NastyaGermanovich** 28 уровень, Санкт-Петербург

18 октября 2017, 19:01

Спасибо за ответ! ;)

Ответить

0

**Sergio** 19 уровень

23 ноября 2017, 10:49

Не согласен. Мы в любом случае вынуждены переопределять методы оборачиваемого класса. Т.е. просто переписываем заново все его методы. Для этого наследование не требуется. Особенно с учетом того, что экземпляр оборачиваемого класса мы получаем в параметрах. При необходимости использовать в обертке нативные методы оборачиваемого класса мы можем просто эти методы транслировать, как и возврат из них.

Ответить

+1

**Джонни** 23 уровень

8 января, 08:25

Как наследование не требуется? Поясни, пожалуйста.Класс-обёртка создаётся же наследованием.

Ответить

0

**Gleed** 25 уровень

5 октября 2017, 22:25

Все понятно, но кто такой Скрафи?

Ответить

+16

**Anton Stezhkin** 19 уровень

14 декабря 2017, 17:30

уборщик из Футурамы.



Ответить

+3

 Загрузить еще

[javarush.ru/](http://javarush.ru/), [G+ \(https://plus.google.com/114772402300089087607\)](https://plus.google.com/114772402300089087607), [Twitter \(https://twitter.com/javarush\\_ru\)](https://twitter.com/javarush_ru), [LinkedIn](#)



Программистами не рождаются  
© 2018

