



Интерфейсы

Java Core (/quests/QUEST_JAVA_CORE)
2 уровень (/quests/lectures/?quest=QUEST_JAVA_CORE&level=2), 7 лекция (/quests/lectures/questcore.level02.lecture07)

ОТКРЫТА

— Привет, Амиго! Сегодня у тебя день открытий. Новая и интересная тема – это интерфейсы.

— Ага. День настолько чудесный, что я приду домой и приму ванну полную воды.

— **Интерфейс** – это дитя Абстракции и Полиморфизма. Интерфейс очень напоминает абстрактный класс, у которого все методы абстрактные. Он объявляется так же, как и класс, только используется ключевое слово `interface`. Примеры:

Код

Описание и Факты

```
1 interface Drawable
2 {
3     void draw();
4 }
5 interface HasValue
6 {
7     int getValue();
8 }
```

1) Вместо слова `class` пишем `interface`.

2) Содержит только абстрактные методы (слово `abstract` писать не нужно).

3) На самом деле у интерфейсов все методы — `public`.

```
1 interface Element extends Drawable, HasValue
2 {
3     int getX();
4     int getY();
5 }
```

Интерфейс может наследоваться только от интерфейсов.

Интерфейсов-родителей может быть много.

```
1 class abstract ChessItem implements Drawable, HasValue
2 {
3     private int x, y, value;
4
5     public int getValue()
6     {
7         return value;
8     }
9
10    public int getX()
11    {
12        return x;
13    }
14
15    public int getY()
16    {
17        return y;
18    }
19
20 }
```

Класс может наследоваться от нескольких интерфейсов (и только от одного класса). При этом используется ключевое слово `implements`.

Класс `ChessItem` объявлен абстрактным: он реализовал все унаследованные методы, кроме `draw`.

Т.е. класс `ChessItem` содержит один абстрактный метод: `draw()`.

— Интересно. А зачем нужны интерфейсы? Когда их используют?

— У интерфейсов есть два сильных преимущества по сравнению с классами:

1) Отделение «описания методов» от их реализации.

Раньше я тебе рассказывал, что если ты хочешь разрешить вызывать методы своего класса из других классов, то их нужно пометить ключевым словом `public`. Если же хочешь, чтобы какие-то методы можно было вызывать только из твоего же класса, их нужно пометить ключевым словом `private`. Другими словами мы делим методы класса на две категории: «для всех» и «только для своих».

С помощью интерфейсов, это деление можно усилить еще больше. Мы сделаем специальный «класс для всех», и второй «класс для своих», который унаследует от первого. Вот как это примерно будет:

Было

```
1 class Student
2 {
3     private String name;
4
5     public Student(String name)
6     {
7         this.name = name;
8     }
9
10    public String getName()
11    {
12        return this.name;
13    }
14
15    private void setName(String name)
16    {
17        this.name = name;
18    }
```

Стало

```
1 interface Student
2 {
3     public String getName();
4 }
5
6 class StudentImpl implements Student
7 {
8     private String name;
9     public StudentImpl(String name)
10    {
11        this.name = name;
12    }
13    public String getName()
14    {
15        return this.name;
16    }
17    private void setName(String name)
18    {
19        this.name = name;
20    }
21 }
```

Было

```
1 public static void main(String[] args)
2 {
3     Student student =
4         new Student("Alibaba");
5     System.out.println(student.getName());
6 }
```

Стало

```
1 public static void main(String[] args)
2 {
3     Student student =
4         new StudentImpl("Ali");
5     System.out.println(student.getName());
6 }
```

Мы разбили наш класс на два: **интерфейс** и **класс, унаследованный от интерфейса**.

— И в чем тут преимущество?

— Один и тот же интерфейс могут реализовывать (наследовать) различные классы. И у каждого может быть свое собственное поведение. Так же как `ArrayList` и `LinkedList` — это две различные реализации интерфейса `List`.

Таким образом, мы скрываем не только различные реализации, но и даже сам класс, который ее содержит (везде в коде может фигурировать только интерфейс). Это позволяет очень гибко, прямо в процессе исполнения программы, подменять одни объекты на другие, меняя поведение объекта скрытно от всех классов, которые его используют.

Это очень мощная технология в сочетании с полиморфизмом. Сейчас далеко не очевидно, зачем так нужно делать. Ты должен сначала столкнуться с программами, состоящими из десятков или сотен классов, чтобы понять, что интерфейсы способны существенно упростить тебе жизнь.

2) Множественное наследование.

В Java все классы могут иметь только одного класса-родителя. В других языках программирования, классы часто могут иметь несколько классов-родителей. Это очень удобно, но приносит так же много проблем.

В Java пришли к компромиссу — запретили множественное наследование классов, но разрешили множественное наследование интерфейсов. Интерфейс может иметь несколько интерфейсов-родителей. Класс может иметь несколько интерфейсов-родителей и только один класс-родитель.

Комментарии (39)

популярные

новые

старые

Никита

Введите текст комментария

Vadims Sergejevs 15 уровень

27 марта, 17:13

...

А чем отличие абстрактного класса от интерфейса? Когда следует использовать интерфейс, а когда абстрактный класс?

Ответить

+1

Владислав 16 уровень, Москва

24 февраля, 12:53

...

Друзья, возник один вопрос. Допустим, у нас имеется абстрактный класс, поддерживающий один интерфейс. Если мы создаем класс, наследуем его от абстрактного и переопределяем методы родителя, будет ли наш класс так же поддерживать этот интерфейс автоматически или его тоже нужно будет наследовать от этого интерфейса?

Ответить

0

Logikarma 13 уровень

7 марта, 21:16

...

Будет поддерживать. Нет, не нужно дополнительно прописывать наследование интерфейса для child класса. Если только ты не хочешь убедиться в том факте, что твой child класс будет точно наследоваться от этого интерфейса (тогда ты будешь получать ошибку всегда, когда кто-то удалит интерфейс или нужные тебе методы в нем).
Вот сабж на SOF.
<https://stackoverflow.com/questions/5668429/why-would-both-a-parent-and-child-class-implement-the-same-interface>

Ответить

0

Джонни 22 уровень

24 октября 2017, 19:27

...

Кстати, в 3м примере ошибка.
не

```
1 class abstract ChessItem implements Drawable, HasValue
```

а

```
1 abstract class ChessItem implements Drawable, HasValue
```

Ответить

+5

P0huber 14 уровень

18 ноября 2017, 20:21

...

класс ChessItem содержит один абстрактный метод: draw() исходя из его implements. Будьте внимательны! И поскольку данный класс содержит хотя бы один абстрактный метод(нереализованный от унаследованного), он будет именно АБСТРАКТНЫМ. Там нет ошибки!

Ответить

+2

Джонни 22 уровень

18 ноября 2017, 20:43

...

Не, я про порядок слов вначале.

Ответить

+3

Антон 17 уровень

25 декабря 2017, 14:01

...

порядок слов в данном случае не имеет значения, насколько я успел понять

Ответить

0

Джонни 22 уровень

27 декабря 2017, 21:38

...

Имеет.



Ответить

0

Унан Абраамян 22 уровень

18 января, 17:52

...

Это порядок спецификаторов не имеет значения:

```
1 public static abstract class
```

```
1 public abstract static class
```

```
1 abstract public static class
```

Но Class всегда идёт в конце (перед именем класса).

Ответить

0

Андрей Коробов 13 уровень

19 января, 12:56

Имеет.
И об этом прямо говорит IntelliJ Idea сообщением "Identifier expected" (если поставить "abstract" после "class"). Т.е. Идея планирует увидеть в этом месте (после class) идентификатор класса, а видит служебное слово "abstract".

Ответить

0

Джонни 22 уровень

24 октября 2017, 19:24

Вот теперь-то стало понятно, чем отличаются:

```
1 Map <Integer, String> map = new HashMap <Integer, String>();
```

от

```
1 HashMap <Integer, String> map = new HashMap <Integer, String>();
```

Ответить

+3

P0huber 14 уровень

18 ноября 2017, 20:29

[Interface Map<K,V>](#) - это и есть интерфейс. А [Class HashMap<K,V>](#) - реализация данных Serializable, Cloneable, Map<K,V> интерфейсов, но этот клас также явл-ся родителем для AbstractMap<K,V>. Всю эту инфу вы можете посмотреть в оф гайде в первых обзацах их страниц, просто лишь перейдя по ссылкам :)

Ответить

+2

Artem Ostretsov 14 уровень

17 октября 2017, 08:34

чаще встречаю, что класс __реализует__ интерфейс, а не наследует.

Ответить

0

P0huber 14 уровень

18 ноября 2017, 20:35

Хммм.. А ведь чтобы класс смог реализовать интерфейс он прежде должен быть от него унаследован! Чувствуете соль? Рекомендую вдумчиво прочесть первые главы "[Object-Oriented Analysis and Design with Applications](#)" - толковая книга по ООП, также ООА и ООД - и сразу эта тема еще более просветлится. Это международная читательская соцсеть, если регнитесь - добавляйтесь в друзья. Будем обмениваться очень полезными знаниями по книгам...

Ответить

0

karbofas 18 уровень

21 августа 2017, 20:01

```
class abstract ChessItem implements Drawable, HasValue
используется методы interface Element. Разве не должно быть
class abstract ChessItem implements Element, HasValue?
```

Ответить

0

Vladimir #1171823 16 уровень, Москва

27 августа 2017, 12:11

Нет, не должно. Это просто пример, для сравнения. Если мы "применяем" интерфейс Element, то Drawable и HasValue можно не применять, они уже "включены" в Element.

Ответить

+2

karbofas 18 уровень

27 августа 2017, 20:59

Окей, понял!

Ответить

0

mila 39 уровень, Самара

10 сентября 2017, 18:41

но класс ChessItem реализовал методы int getX() и int getY(), а это методы интерфейса Element. Drawable и HasValue про них не знают

Ответить

+2

Геннадий Шевченко 32 уровень

19 сентября 2017, 07:20

должно!

Ответить

0

Даниленко Виктор 40 уровень, Днепр

20 сентября 2017, 15:50

возникла путаница.. на самом деле в примере всё верно.
-есть 2 интерфейса.
-есть интерфейс extends 2 интерфейса

в нём 2 новых абстрактных метода get
-есть класс использующий 2 интерфейса.
в нём тоже есть 2 метода get,

Вы не забывайте что класс может определять свои методы, и наличие 2х методов get в классе не означает что он обязан implements Element с такими же методами. Это разные сущности, Element со своими get, класс со своими get.

то есть:

1. добавление классу implements, extends - обязывает реализовывать абстрактные методы унаследованные от них.
2. Наличие каких-то методов в классе - не обязывает implements Интерфесы или extends Родителей с такими же названиями методов.

Ответить

+7

P0huber 14 уровень

18 ноября 2017, 20:46

Даниленко Виктор верно заметил, плюсанул.
Кратко говоря, если бы ло бы `class abstract ChessItem implements Element`, то методы `public int getX(){return x;}` и `public int getY(){return y;}` в этом классе были не его собственные, а реализации методов вышеуказанного интерфейса!

Ответить

+2

Клим Бедняков 14 уровень, Snezhinsk

18 августа 2017, 00:16

Я один туплю ? Есть кто половину не понимает нового ?

Ответить

+6

Сергей Черник 33 уровень

3 сентября 2017, 16:58

Говорили, что это одна из самых сложных тем! Наверное потому и задачи простые такие

Ответить

+3

Alex 29 уровень

30 декабря 2017, 07:33

Ну тут надо суть эту уловить ООП, а так вроде ничего сложного)
Можно доп. инфу поискать для пущего понимания
Вот я темы про многопоточность предвкушаю...
Там наверно реально будет весело)

Ответить

0

Сергей Черник 33 уровень

30 декабря 2017, 11:58

Почти прошел многопоточность, подтверждаю, ад и содом. Еще есть время повернуть назад! ;)

Ответить

+1

Alex 29 уровень

30 декабря 2017, 14:31

Слишком поздно, билет в один конец (подписка на год) уже куплен
Только вперед ёпт :)

Ответить

+1

Сергей Черник 33 уровень

30 декабря 2017, 15:40

Ну тогда дайте ему начищенный автомат, вещмешок - и - родина таких не забывает!

Ответить

+1

Греков 14 уровень, Санкт-Петербург

23 июля 2017, 14:36

Является ли фраза о ванне полной воды намеком на суицид путем замыкания?:D

Ответить

+13

Redas Shuliakas 21 уровень

25 июля 2017, 14:36

Я тоже об этом подумал, столько новых сложных тем подкинули)))

Ответить

+3

Orion 22 уровень

7 сентября 2017, 00:10

Замыкания? По-моему это к порезу вен относится :)

Ответить

+2

Ilya Kladov 28 уровень, Харьков

24 сентября 2017, 15:27

главный герой JavaRash - робот

Ответить

+4

Dmitry 16 уровень

10 октября 2017, 23:39

можно гидравлику перезать....

Ответить

+4

Vikentel 23 уровень, Минск

24 июля 2017, 10:26

vikentst 22 уровень, Минск 21 июля 2017, 19:20

Хорошая статейка про реализацию и правила для абстрактных классов и интерфейсов
<http://kostin.ws/java/java-abstract-and-interfaces.html>

Ответить +15

Denis 19 уровень, Москва 21 июля 2017, 22:13

Спасибо, реально хорошо объясняет!

Ответить 0

Roman 31 уровень 17 октября 2017, 10:43

Вот еще интересный момент про создание объекта
<http://code.i-harness.com/ru/q/d09a4f>

Ответить 0

Karahanid 40 уровень, Алматы 15 июля 2017, 11:13

Все же семантически правильнее писать сначала abstract потом class.

Ответить +1

Загрузить еще

javarush.ru/, <https://plus.google.com/114772402300089087607>, https://twitter.com/javarush_ru,



Программистами не рождаются
© 2018