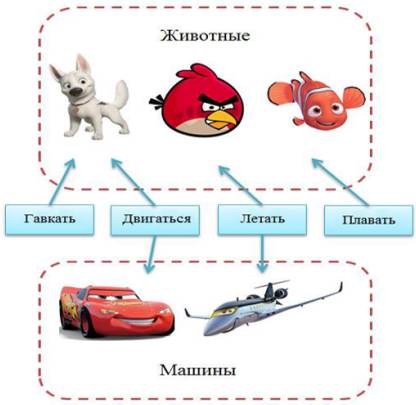
Интерфейсы — это больше чем интерфейсы — это поведение

Java Core (/quests/QUEST_JAVA_CORE)
2 уровень (/quests/lectures/?quest=QUEST_JAVA_CORE&level=2), 8 лекция (/quests/lectures/questcore.level02.lecture08)

ОТКРЫТА

— Привет, Амиго! А вот и снова я. Хочу рассказать тебе еще об одном взгляде на интерфейсы. Понимаешь, класс – это, чаще всего модель какого-то конкретного объекта. Интерфейс же больше соответствует не объектам, а их способностям или ролям.



Например, такие вещи, как машина, велосипед, мотоцикл и колесо лучше всего представить в виде классов и объектов. А такие их способности как «могу ездить», «могу перевозить людей», «могу стоять» — лучше представить в виде интерфейсов. Смотри пример:

Код на Java

Описание

```
1  interface Moveable
2  {
3  void move(String newAddress);
4  }
```

— соответствует способности передвигаться.

```
1  interface Driveable
2  {
3  void drive(Driver driver);
4  }
```

— соответствует способности управляться водителем.

```
1  interface Transport
2  {
3  void addStaff(Object staff);
4  Object removeStaff();
5  }
```

— соответствует способности перевозить грузы.

```
1  class Wheel implements Moveable
2  {
3  ...
4  }
```

— класс «колесо». Обладает способностью передвигаться.

```
1  class Car implements Moveable, Drivable, Transport
2  {
3  ...
4  }
```

— класс «машина». Обладает способностью передвигаться, управляться человеком и перевозить грузы.

```
1  class Skateboard implements Moveable, Driveable
2  {
3  ...
4  }
```

— класс «скейтборд». Обладает способностью передвигаться и управляться человеком.

Интерфейсы сильно упрощают жизнь программиста. Очень часто в программе тысячи объектов, сотни классов и всего пара десятков интерфейсов – ролей. Ролей мало, а их комбинаций – классов – очень много.

Весь смысл в том, что тебе не нужно писать код для взаимодействия со всеми классами. Тебе достаточно взаимодействовать с их ролями (интерфейсами).

Представь, что ты – робот-строитель и у тебя в подчинении есть десятки роботов, каждый из которых может иметь несколько профессий. Тебе нужно срочно достроить стену. Ты просто берешь всех роботов, у которых есть способность «строитель» и говоришь им строить стену. Тебе все равно, что это за роботы. Хоть робот-поливалка. Если он умеет строить – пусть идет строить.

Вот как это выглядело бы в коде:

Код на Java

Описание

```
1  static interface WallBuilder
2  {
3  void buildWall();
4  }
```

— способность «строитель стен». Понимает команду «(по)строить стену» — имеет соответствующий метод.

```
1  static class РабочийРобот implements WallBuilder
2  {
3  void buildWall()
4   {
5  …
6   }
7  }
8  static class РоботСторож implements WallBuilder
9  {
10 void buildWall()
11  {
12 …
13  }
14 }
15 static class Поливалка
16 {
17 …
18 }
```

— роботы у которых есть эта профессия/особенность.

— для удобства я сделал классам имена на русском. Такое допускается в java, но крайне нежелательно.

— поливалка не обладает способностью строить стены (не реализует интерфейс WallBuilder).

```
1  public static void main(String[] args)
2  {
3   //добавляем всех роботов в список
4   ArrayList robots = new ArrayList();
5   robots.add(new РабочийРобот());
6   robots.add(new РоботСторож());
7   robots.add(new Поливалка());
8
9   //строить стену, если есть такая способность
10  for (Object robot: robots)
11  {
12   if (robot instanceof WallBuilder)
13   {
14    WallBuilder builder = (WallBuilder) robot;
15    builder.buildWall();
16    }
17   }
18  }
19 }
```

— как дать им команду – построить стену?

— Чертовски интересно. Даже и не думал, что интерфейсы – такая интересная тема.

— А то! В совокупности с полиморфизмом – это вообще бомба.

‹ (/quests/lectures/questcore.level02.lecture07)

×12 › (/quests/lectures/questcore.level02.lecture09)

Комментарии (37) популярные новые старые

Никита

Введите текст комментария

P0huber 14 уровень 20 ноября 2017, 00:06

Кто в теме? в последнем примере почему указано именно объект робот **for (Object robot: robots)** ? Или его можно заменить на что-то другое? Если использование класса **Object** тут можно можно оправдать тем, что он является суперклассом по отношению к **ArrayList robots**, то откуда взялась ссылка **robot** класса Object?

Ответить 0

BlackHoleSun 17 уровень 21 ноября 2017, 20:18

Видимо потому что у нас ArrayList <Object>, то есть, мы перебираем объекты.
В скобках robot - это автор так переменную назвал, вы можете ее обозвать как угодно, хоть i.

Ответить +4

Роман 18 уровень, Москва 15 ноября 2017, 16:12

Надеюсь дальше будет понятнее суть интерфейсов)
пока не ясен их смысл, есть всё равно все методы надо переопределять.

Ответить +1

Роман 18 уровень, Москва 15 ноября 2017, 16:34

Чуть дальше понял, IDEA переопределяет их за нас :D

Ответить 0

Роман 16 уровень, Харьков 16 ноября 2017, 02:19

Фишка в том, что можно поместить ссылку на объект, который делает то, что заявлено в интерфейсе в переменную типа интерфейса и применять к нему только те методы, которые интерфейс описывает, а объект реализует и не иметь доступ к другим методам объекта.

Если работник умеет грузить (наследуя/реализуя интерфейс) и программировать (внутренний метод работника), то наняв его на работу как грузчика (через переменную интерфейса, в которую помещена ссылка на работника) заставить работника программировать уже не получится :)

```
1  public class Solution {
2      public static void main(String[] args) {
3          iLoadable iLoadWorker = new UniversalWorker();
4          iLoadWorker.toLoad();       // грузит
5          iLoadWorker.toProgram();  // нет доступа к методу
6
7          UniversalWorker uWorker = new UniversalWorker();
8          uWorker.toLoad();       // грузит
9          uWorker.toProgram(); // пЕдалит :)
10     }
11
12     interface iLoadable {
13         void toLoad();
14     }
15
16     // ctrl+i - hotKey для @Override
17     static class UniversalWorker implements iLoadable {
18        public void toProgram() {
19             // Code for programming
20         }
21         @Override
22         public void toLoad() {
23            // Code for loading
24         }
25     }
26 }
```

Ответить +13

Джонни 22 уровень 24 октября 2017, 22:13

Тут есть одна строка:

```
1   WallBuilder builder = (WallBuilder) robot;
```

Про приведение типов все понятно. Но что означает:
"WallBuilder builder"?
Разве можно объявить переменную типа интерфейса?
Не лучше было бы написать:

```
1   Object builder = (WallBuilder) robot;
```

?

Ответить +1

Джонни 22 уровень 24 октября 2017, 22:20

Кажется я понял, если бы мы так написали, то у наших объектов не было бы метода buildWall(). А по-скольку методы объектов переопределены, то у каждого класса вызовется есть реализованный метод buildWall(). По сути, интерфейс тоже класс, следовательно можно объявить переменную типа интерфейса.

Ответить 0

panvasyan 27 уровень, Мариуполь 25 октября 2017, 14:07

Вообще говоря, можно было бы обойтись без временной переменной builder, сразу вызывая метод buildWall() у приведенной к типу (WallBuilder) переменной robot. Дело в удобстве использования. Временную переменную имеет смысл заводить, если далее в коде идут сложные множественные манипуляции с методами "стеностроителя".

Ответить +2

Hexogon 20 уровень 30 сентября 2017, 23:45

А зачем интерфейс указан как статик? Разве и так не подразумевается что интерфейс статичен и не может иметь экземпляры?

Ответить +2

Артем 21 уровень 12 сентября 2017, 02:36

Объясните пжл, что означает это строка: -

WallBuilder builder = (WallBuilder) robot;

Ответить 0

Michael Ginzburg 21 уровень 12 сентября 2017, 20:37

Приведение типов. Т.е. тип Object приводится к типу интерфейса, т.е. тому типу, который умеет строить стену.

Ответить 0

Олег 31 уровень 3 сентября 2017, 16:55

Пока так и не понял чем интерфейс отличается от абстрактного класса :(

Ответить +1

Дмитрий Макаровский 36 уровень 4 сентября 2017, 22:02

в интерфейсах ты можешь писать только названия методов без их реализации.
в абстрактных классах можно писать не только абстрактные методы

Ответить +2

Иван Ермоленко 18 уровень 7 сентября 2017, 19:41

Любой класс может быть унаследован только от одного др. класса, в том числе и абстрактного, а вот реализовать интерфейсов можно множество.

Ответить +2

Михаил Левин 17 уровень, Санкт-Петербург 10 сентября 2017, 15:22

Можно реализовывать сколько угодно интерфейсов, в то время как наследоваться можно только от одного класса.

Ответить +1

Deleted 17 уровень, Днепр 4 октября 2017, 15:30

Интерфейс - полностью абстрактный класс.

Ответить +1

Wh0 17 уровень 3 ноября 2017, 15:10

Главное отличие и польза интерфейста от абстрактного класса в том, что ты не можешь унаследовать объект от двух классов, тем самым приобретя их поведение, но зато , это можно реализовать при помощи интерфейсов. Т.е. можно получить класс унаследованный от супер класса + добавить поведение при помощи интерфейсов, Грубо говоря, - это брендовые костыли)))

Ответить 0

Vitaly 21 уровень, Москва — 17 июля 2017, 10:09

Робот-поливалка нраицца!))*
*к сожалению, данный комментарий не имеет никакой смысловой нагрузки и может являться верным признаком сильной усталости, вызванной попытками систематизировать только что полученные знания.

Ответить +15

Alex 29 уровень — 30 декабря 2017, 08:18

Очень умная мысль
Солидарен.

Ответить 0

Роман Холуев 13 уровень — 6 июля 2017, 11:52

Static interface Я тащусь.

Ответить +4

Красный медведь 14 уровень — 31 мая 2017, 10:29

Почему в лекции не сказано что классы должны быть абстрактными?

Ответить +1

Akira Rokudo 20 уровень, Москва — 25 июня 2017, 08:33

Если бы они были абстрактными(речь не о классе предке роботе) то их нельзя было бы создать.

Ответить 0

Dmitry Mersiyanov 20 уровень, Москва — 25 мая 2017, 06:22

а почему нельзя этот цикл for (Object robot: robots) сделать как for (Robot robot: robots) ? Тогда приведение не пришлось бы делать...

Ответить 0

Max 19 уровень — 4 июля 2017, 12:49

потому что у нас не факт есть производный класс Robot, наследниками которого являются РабочийРобот, поливалка и т д. А класс Object 100% является для них родительским.

Ответить +1

Загрузить еще