Professor Hans Noodles

41 уровень

27.05.2019 37239 32

Что такое дженерики в Java

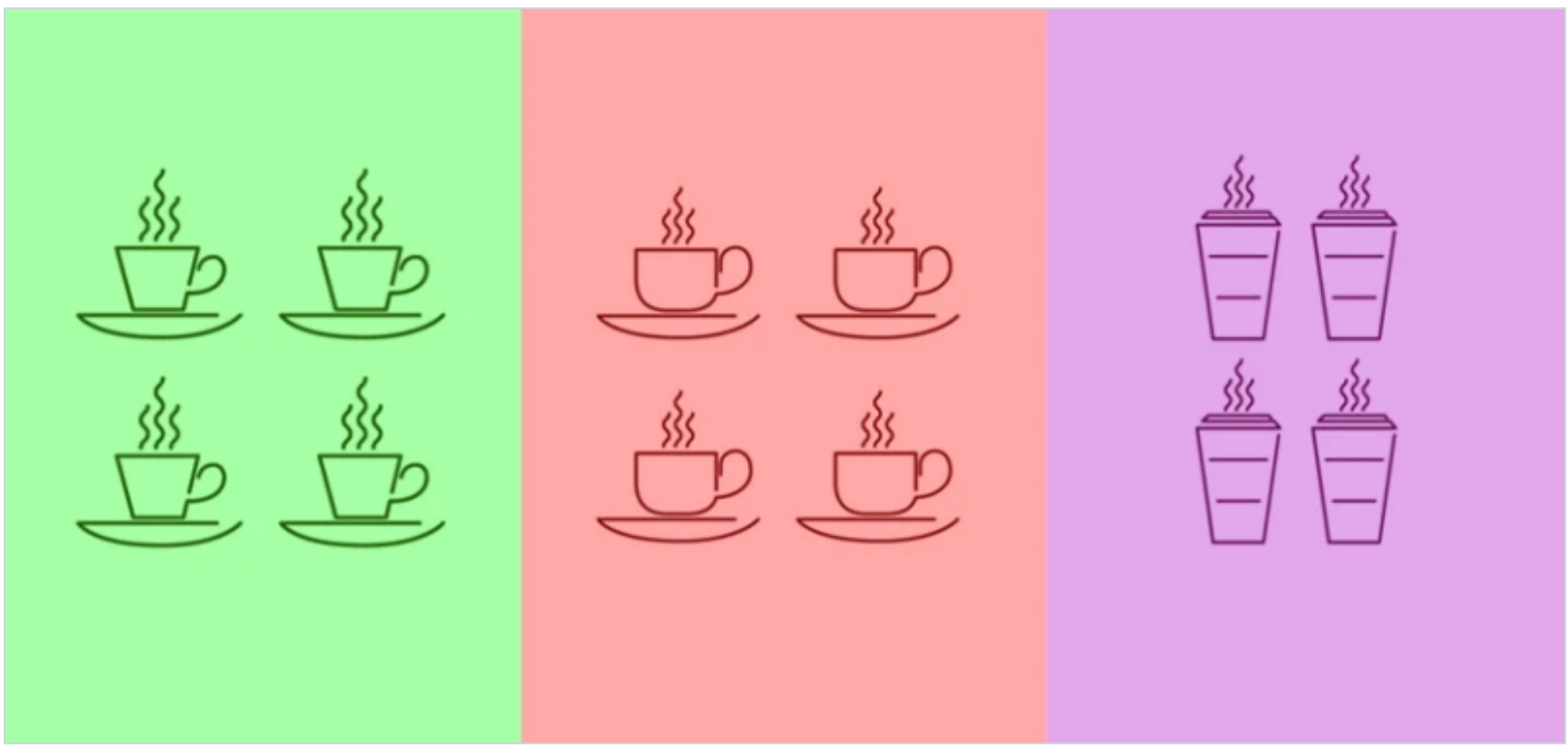
Статья из группы Java Developer

43181 участник

Вы в группе

Привет! Сегодня мы поговорим о дженериках.

Надо сказать, что ты выучишь много нового! Дженерикам будет посвящена не только эта, но еще и несколько следующих лекций.



Поэтому, если эта тема тебе интересна — тебе повезло: сегодня ты узнаешь многое об особенностях дженериков. Ну а если нет — смирись и расслабься! :) Это очень важная тема, и знать ее нужно.

Давай начнем с простого: «что» и «зачем».

Что такое дженерики?

Дженерики — это типы с параметром.

При создании дженерика ты указываешь не только его тип, но и тип данных, с которыми он должен работать.

Думаю, самый очевидный пример уже пришел тебе в голову — это ArrayList! Вот как мы обычно создаем его в программе:

```
3
4  public class Main {
5
6     public static void main(String[] args) {
7
8         List<String> myList1 = new ArrayList<>();
9         myList1.add("Test String 1");
10        myList1.add("Test String 2");
11    }
12 }
```

Как нетрудно догадаться, особенность списка заключается в том, что в него нельзя будет «запихивать» все подряд: он работает исключительно с объектами `String`.

Теперь давай сделаем небольшой экскурс в историю Java и попробуем ответить на вопрос: «зачем?». Для этого мы сами напишем упрощенную версию класса ArrayList.

Наш список умеет только добавлять данные во внутренний массив и получать эти данные:

```
1  public class MyListClass {
2
3     private Object[] data;
4     private int count;
5
6     public MyListClass() {
7         this.data = new Object[10];
8         this.count = 0;
9     }
10
11    public void add(Object o) {
12        this.data[count] = o;
13        count++;
14    }
15
16    public Object[] getData() {
17        return data;
18    }
19 }
```

Допустим, мы хотим, чтобы наш список хранил только числа `Integer`. Дженериков у нас нет.

Мы не можем явно указать проверку o instance of `Integer` в методе `add()`. Тогда весь наш класс будет пригоден только для `Integer`, и нам придется писать такой же класс для всех существующих в мире типов данных!

Мы решаем положиться на наших программистов, и просто оставим в коде комментарий, чтобы они не добавляли туда ничего лишнего:

```
1  //use it ONLY with Integer data type
2  public void add(Object o) {
3     this.data[count] = o;
4     count++;
5  }
```

со строками, а потом посчитать их сумму:

```
1  public class Main {
2
3     public static void main(String[] args) {
4
5         MyListClass list = new MyListClass();
6         list.add(100);
7         list.add(200);
8         list.add("Lolkek");
9         list.add("Shalala");
10
11        Integer sum1 = (Integer) list.getData()[0] + (Integer) list.getData()[1];
12        System.out.println(sum1);
13
14        Integer sum2 = (Integer) list.getData()[2] + (Integer) list.getData()[3];
15        System.out.println(sum2);
16    }
17 }
```

Вывод в консоль:

300
Exception in thread "main" java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Integer
at Main.main(Main.java:14)

Что худшее в этой ситуации?

Далеко не невнимательность программиста. Худшее то, что **неправильный код попал в важное место нашей программы и успешно скомпилировался**.

Теперь мы увидим ошибку не на этапе написания кода, а только на этапе тестирования (и это в лучшем случае!).

Исправление ошибок на более поздних этапах разработки стоит намного больше — и денег, и времени.

Именно в этом заключается преимущество дженериков: класс-дженерик позволит незадачливому программисту обнаружить ошибку сразу же. Код просто не скомпилируется!

```
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class Main {
5
6     public static void main(String[] args) {
7
8         List<Integer> myList1 = new ArrayList<>();
9
10        myList1.add(100);
11        myList1.add(100);
12        myList1.add("Lolkek");//ошибка!
13        myList1.add("Shalala");//ошибка!
14    }
15 }
```

Программист сразу «очухается» и моментально исправится.

Кстати, нам не обязательно было создавать свой собственный класс- `List`, чтобы увидеть ошибку такого рода.

Достаточно просто убрать скобки с указанием типа (`<Integer>`) из обычного ArrayList!

```
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class Main {
5
6     public static void main(String[] args) {
7
8        List list = new ArrayList();
9
10       list.add(100);
11       list.add(200);
12       list.add("Lolkek");
13       list.add("Shalala");
14
15        System.out.println((Integer) list.get(0) + (Integer) list.get(1));
16        System.out.println((Integer) list.get(2) + (Integer) list.get(3));
17    }
18 }
```

Вывод в консоль:

300
Exception in thread "main" java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Integer
at Main.main(Main.java:16)

То есть даже используя «родные» средства Java, можно допустить такую ошибку и создать небезопасную коллекцию.

Однако, если вставить этот код в IDEa, мы увидим предупреждение: “*Unchecked call to add(E) as a member of raw type of java.util.List*”

Нам подсказывают, что при добавлении элемента в коллекцию без дженериков что-то может пойти не так.

Но что значит фраза «raw type»?

Дословный перевод будет вполне точным — «**сырой тип**» или «**грязный тип**».

`Raw type` — это класс-дженерик, из которого удалили его тип.

Иными словами, `List myList1` — это `Raw type`. Противоположностью `raw type` является `generic type` — класс-дженерик (также известный как `parameterized type`), созданный правильно, с указанием типа. Например, `List<String> myList1`.

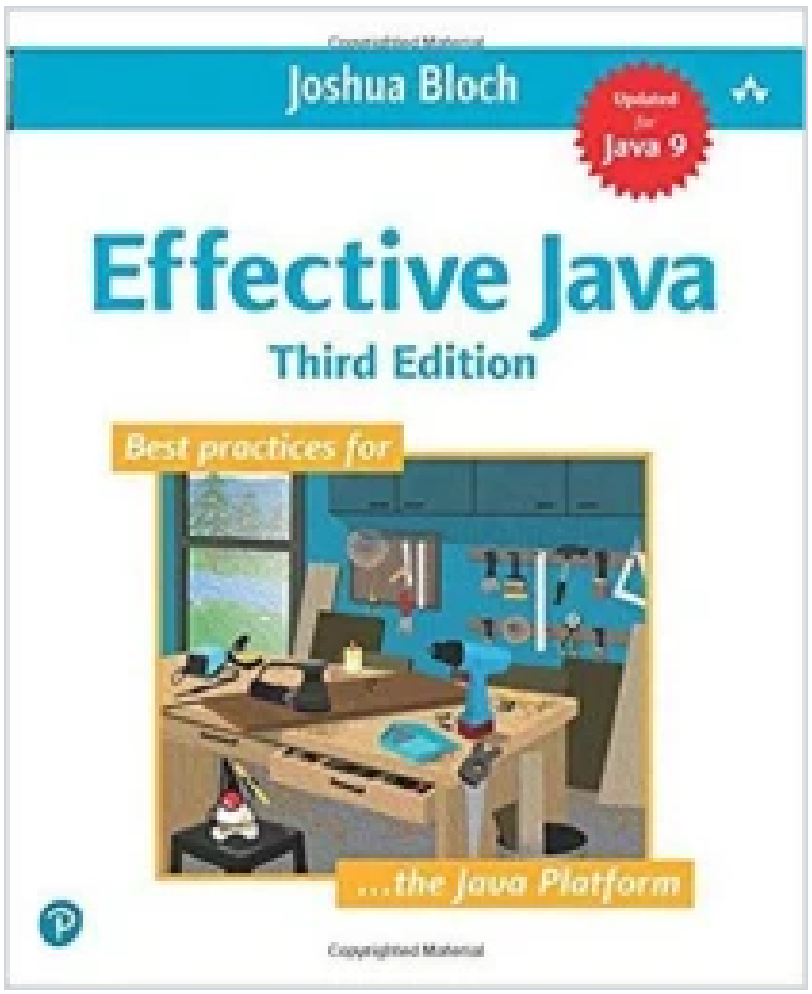
У тебя мог возникнуть вопрос: а почему в языке вообще позволено использовать `raw types`?

Причина проста. Создатели Java оставили в языке поддержку `raw types` чтобы не создавать проблем с совместимостью. К моменту выхода Java 5.0 (в этой версии впервые появились дженерики) было написано уже очень много кода с

Поэтому такая возможность сохраняется и сейчас.

Мы уже не раз упоминали классическую книгу Джошуа Блоха «Effective Java» в лекциях.

Как один из создателей языка, он не обошел в книге и тему использования `raw types` и `generic types`.



Глава 23 этой книги носит весьма красноречивое название: **«Не используйте raw types в новом коде»**

Это то, что нужно запомнить. При использовании классов-дженериков ни в коем случае не превращай `generic type` в `raw type`.

Типизированные методы

Java позволяет тебе типизировать отдельные методы, создавая так называемые generic methods.

Чем такие методы удобны? Прежде всего тем, что позволяют работать с разными типами параметров. Если к разным типам можно безопасно применять одну и ту же логику, дженерик-метод будет отличным решением. Рассмотрим пример.

Допустим, у нас есть какой-то список `myList1`. Мы хотим удалить из него все значения, и заполнить все освободившиеся места новым значением.

Вот так будет выглядеть наш класс с дженерик-методом:

```
1  public class TestClass {
2
3     public static <T> void fill(List<T> list, T val) {
4         for (int i = 0; i < list.size(); i++)
5             list.set(i, val);
6     }
7
8     public static void main(String[] args) {
9
10        List<String> strings = new ArrayList<>();
11        strings.add("Старая строка 1");
12        strings.add("Старая строка 2");
13        strings.add("Старая строка 3");
```

```
16
17          System.out.println(strings);
18
19          List<Integer> numbers = new ArrayList<>();
20          numbers.add(1);
21          numbers.add(2);
22          numbers.add(3);
23
24          fill(numbers, 888);
25          System.out.println(numbers);
26      }
27  }
```

Обрати внимание на синтаксис, он выглядит немного необычно:

```
1   public static <T> void fill(List<T> list, T val)
```

Перед типом возвращаемого значения написано <T>, что указывает на дженерик метод. В данном случае метод принимает на вход 2 параметра: список объектов T и еще один отдельный объект T.

За счет использования <T> и достигается типизация метода: мы не можем передать туда список строк и число. Список строк и строку, список чисел и число, список наших объектов `Cat` и еще один объект `Cat` — только так.

В методе `main()` наглядно демонстрируется, что метод `fill()` легко работает с разными типами данных.

Сначала он принимает на вход список строк и строку, а потом — список чисел и число.

Вывод в консоль:

[Новая строка, Новая строка, Новая строка]
[888, 888, 888]

Представь, если бы логика метода `fill()` нужна была бы нам для 30 разных классов, и у нас не было бы дженерик-методов.

Мы вынуждены были бы писать один и тот же метод 30 раз, просто для разных типов данных! Но благодаря generic-методам мы можем использовать наш код повторно! :)

Типизированные классы

Ты можешь не только пользоваться представленными в Java дженерик-классами, но и создавать собственные!

Вот простой пример:

```
1   public class Box<T> {
2
3       private T t;
4
5       public void set(T t) {
6           this.t = t;
7       }
8
9       public T get() {
10          return t;
```

```
13      public static void main(String[] args) {
14
15          Box<String> stringBox = new Box<>();
16
17          stringBox.set("Старая строка");
18          System.out.println(stringBox.get());
19          stringBox.set("Новая строка");
20
21          System.out.println(stringBox.get());
22
23          stringBox.set(12345);//ошибка компиляции!
24      }
25  }
```

Наш класс `Box<T>` («коробка») является типизированным. Назначив для него при создании тип данных (`<T>`), мы уже не сможем помещать в него объекты других типов.

Это видно в примере. При создании мы указали, что наш объект будет работать со строками:

```
1  Box<String> stringBox = new Box<>();
```

И когда в последней строке кода мы пытаемся положить внутрь коробки число 12345, получаем ошибку компиляции!

Вот так просто мы создали свой собственный дженерик-класс! :)

На этом наша сегодняшняя лекция подходит к концу. Но мы не прощаемся с дженериками! В следующий лекциях поговорим о более продвинутых возможностях, поэтому не прощаемся!)

Успехов в обучении! :)

Andrey Karelin Уровень 41, Sumy, Украина — 30 апреля, 18:03

Вот сколько читаю про дженерики, нигде не проговаривается очевидная вещь, которая помогает понять их суть, а именно то,
что использование типа T, K, V и т.п. обозначает лишь условный тип класса. И главное тут не название, а то какие типы методов/переменных объекта будут возвращаться/работать, по отношению к определенному изначально типу.

Например, указанный в примере метод
public static <T> void fill(List<T> list, T val)
говорит нам о том, что в метод мы можем передать list и val только одинакового между собой типа, и такого же типа, которым при создании был определен класс.

Ответить +11

Глеб Уровень 26, Полоцк, Беларусь — 21 апреля, 19:04

Полезная статья

Ответить 0

Q1R27 Уровень 17, Ukraine — 28 марта, 21:31

Алишев дженерики

Ответить +2

Lex Bekker Уровень 12, Новосибирск, Russian Federation — 8 марта, 20:12

Продолжение
https://javarush.ru/groups/posts/2315-stiranie-tipov
https://javarush.ru/groups/posts/2313-ispoljhzovanie-varargs-pri-rabote-s-dzhenerikami
https://javarush.ru/groups/posts/2324-wildcards-v-generics
возможно не по порядку

Ответить +7

Жора Нет Уровень 39, енакиево, Украина — 17 апреля, 19:34

Ссылки не работают

Ответить 0

Anonymous #2721543 Уровень 20 — 10 мая, 19:18

Работают, если в адресной строке .ru заменить на .com.ua

Ответить 0

LuneFox инженер по сопровождению в **BIFIT** EXPERT — 26 января, 13:45

Добавлю ещё, кстати, что писать именно T не обязательно. Как я понимаю, это конвенция от слова Type. Допускается написание любой лабуды в качестве маркера для дженерика, например:

```
1  public static <KEK> KEK fill(List<KEK> list, KEK value) {
2      for (int i = 0; i < list.size(); i++) {
3          list.set(i, value);
4      }
5      return list.get(0);
6  }
```

При этом повторный код с другим названием этого типа ловится IDE-шкой.

Именно поэтому перед объявлением возвращаемого значения не написать <T>, то программа будет думать, что это не дженерик, а настоящий класс с названием T, и будет ругаться, что не знает такого класса или попросит, чтобы ты создал class T{ }.

Конечно, то, что я написал -- наверняка очевидные вещи, но иногда полезно пощупать код руками и попытаться "сломать систему", чтобы в голове отложилась чёткая картинка, как делать можно, а как нельзя, и почему.

Ответить +9

hamster ClipMaker в **TikTok** — 12 ноября 2021, 10:53

Отличная статья! Спасибо)

Ответить 0

MICRO_MVP_10011 Уровень 37, Москва, Russian Federation — 5 ноября 2021, 17:38

Игорь Full Stack Developer в **IgorApplications** 27 июля 2021, 22:27

Дженирики существуют только на этапе компиляции, в байт коде они уничтожаются (стирание). К примеру нельзя вызвать метод у дженирика, только через приведение типов, но это тот же Object получается, нельзя унаследоваться.

Ответить +1

LuneFox инженер по сопровождению в **BIFIT** EXPERT 26 января, 13:29

Логично, вряд ли джава-машина «прозевает» свой собственный комментарий и положит в список то, что не нужно. Все ошибки будут уже отброшены на этапе компиляции. Дженерики, как я понял, нужны чисто для удобства и защиты от ошибок программистов.

Ответить 0

Игорь Full Stack Developer в **IgorApplications** 5 февраля, 11:59

Да, они нужны для этого, кроме List'a я в своей практике использовал Class<? extends Figure> и тд. Но в других языках это более мощные вещи, к примеру в c++ от него можно даже унаследовать.

Ответить 0

Igor Java/Kotlin Developer 12 июня 2021, 11:12

Очень доступная в плане понимания лекция, но рассказано в ней о дженериках около 10й части.

Ответить 0

мистер т Уровень 35, Москва 11 апреля 2021, 00:56

Продолжение
https://javarush.ru/groups/posts/2315-stiranie-tipov
https://javarush.ru/groups/posts/2313-ispoljhzovanie-varargs-pri-rabote-s-dzhenerikami
https://javarush.ru/groups/posts/2324-wildcards-v-generics
возможно не по порядку

Ответить +18

LuneFox инженер по сопровождению в **BIFIT** EXPERT 26 января, 14:08

Оформи кликабельными ссылками, чтобы народ не мучался с телефонов, пытаясь выделить твой текст).

А порядок, я думаю, по номерам статей - {2313, 2315, 2324} (по крайней мере, я выбрал именно этот порядок), потому что статья[1] ссылается на статью[0], а статья[2] ссылается на статью[0] и статью[1].

Ответить 0

Показать еще комментарии

ЯЗЫК ИНТЕРФЕЙСА

Русский

"Программистами не рождаются" © 2022 JavaRush