Professor Hans Noodles

41 уровень

12.07.2019 21115 29

Использование varargs при работе с дженериками

Статья из группы Java Developer

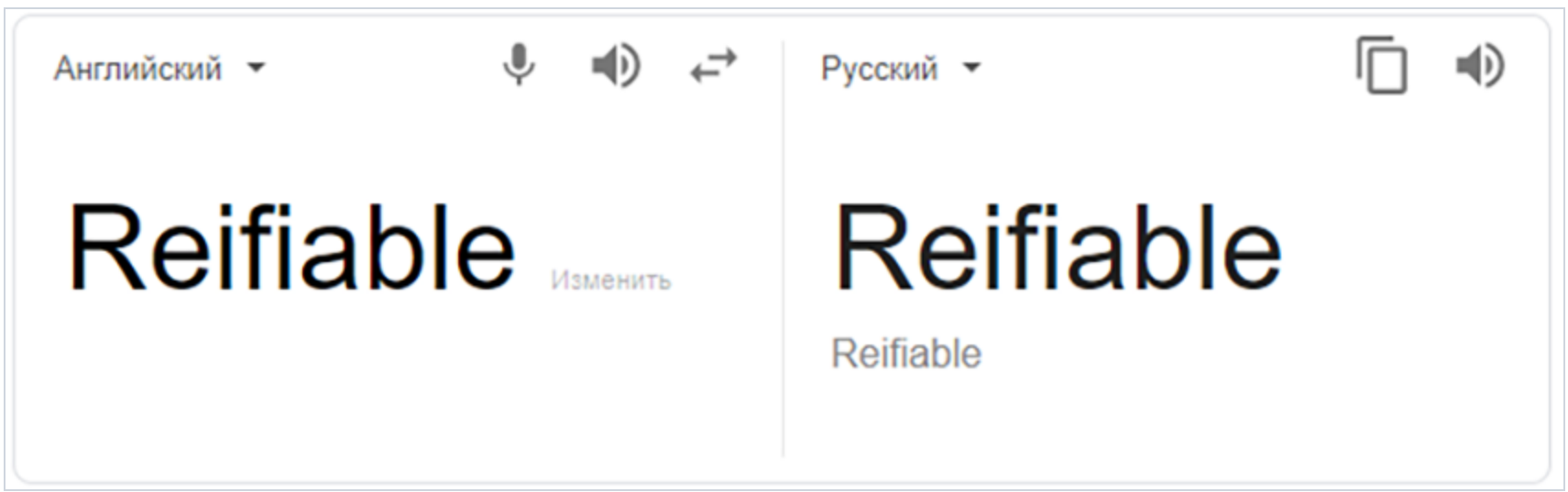
43182 участника

Вы в группе

Привет! На сегодняшнем занятии мы продолжим изучать дженерики. Так уж вышло, что это большая тема, но деваться некуда — это крайне важная часть языка :)

Когда будешь изучать документацию Oracle по дженерикам или читать гайды в интернете, тебе встретятся термины **Non-Reifiable Types** и **Reifiable Types**.

Что это за слово такое — “Reifiable”? Даже если с английским все неплохо, его ты вряд ли встречал. Попробуем перевести!



спасибо, Гугл, ты очень помог -_-

Reifiable-type — это тип, информация о котором полностью доступна во время выполнения.

НАЧАТЬ ОБУЧЕНИЕ

Напротив, **Non-Reifiable Types** — это типы, информация о которых стирается и становится недоступной во время выполнения. Это как раз дженерики — `List<String>`, `List<Integer>` и т.д.

Кстати, ты помнишь, что такое `varargs`?

Если вдруг ты забыл, это аргументы переменной длины.

Они пригодятся в ситуациях, когда мы не знаем, сколько точно аргументов может быть передано в наш метод.

К примеру, если у нас есть класс-калькулятор и в нем есть метод `sum`.

В метод `sum()` можно передать 2 числа, 3, 5 или вообще сколько угодно. Было бы очень странно каждый раз перегружать метод `sum()`, чтобы учесть все возможные варианты.

Вместо этого мы можем сделать так:

```
public class SimpleCalculator {

   public static int sum(int...numbers) {

       int result = 0;

       for(int i : numbers) {

           result += i;
       }

       return result;
   }

   public static void main(String[] args) {

       System.out.println(sum(1,2,3,4,5));
       System.out.println(sum(2,9));
   }
}
```

Вывод в консоль:

```
15
11
```

Так вот, у использования `varargs` в сочетании с дженериками есть некоторые важные особенности.

Давай рассмотрим этот код:

```
import javafx.util.Pair;
import java.util.ArrayList;
import java.util.List;
```

```
6
7       public static <E> void addAll(List<E> list, E... array) {
8
9           for (E element : array) {
10              list.add(element);
11          }
12      }
13
14      public static void main(String[] args) {
15          addAll(new ArrayList<String>(),  //  здесь все нормально
16                  "Leonardo da Vinci",
17                  "Vasco de Gama"
18          );
19
20          // а здесь мы получаем предупреждение
21          addAll(new ArrayList<Pair<String, String>>(),
22                  new Pair<String, String>("Leonardo", "da Vinci"),
23                  new Pair<String, String>("Vasco", "de Gama")
24          );
25      }
26  }
```

Метод `addAll()` принимает на вход список `List<E>` и любое количество объектов `E`, после чего добавляет все эти объекты в список.

В методе `main()` мы дважды вызываем наш метод `addAll()`.

В первый раз мы добавляем в `List` две обычные строки. Здесь все в порядке.

Во второй раз мы добавляем в `List` два объекта `Pair<String, String>`.

И вот здесь мы неожиданно получаем предупреждение:

```
Unchecked generics array creation for varargs parameter
```

Что это значит? Почему мы получаем предупреждение и причем здесь вообще `array`? `Array` — это массив, а в нашем коде нет никаких массивов!

Начнем со второго. В предупреждении упоминается массив, потому что компилятор преобразует аргументы переменной длины (varargs) в массив.

Иными словами, сигнатура нашего метода `addAll()`:

```
1   public static <E> void addAll(List<E> list, E... array)
```

На самом деле выглядит так:

```
1   public static <E> void addAll(List<E> list, E[] array)
```

То есть в методе `main()` компилятор преобразует наш код в это:

```
public static void main(String[] args) {
   addAll(new ArrayList<String>(),
      new String[] {
        "Leonardo da Vinci",
        "Vasco de Gama"
      }
   );
   addAll(new ArrayList<Pair<String,String>>(),
        new Pair<String,String>[] {
            new Pair<String,String>("Leonardo","da Vinci"),
            new Pair<String,String>("Vasco","de Gama")
        }
   );
}
```

С массивом `String` все нормально.

А вот с массивом `Pair<String, String>` — нет.

Дело в том, что `Pair<String, String>` — это Non-Reifiable Type. При компиляции вся информация о типах-параметрах (<String, String>) будет стерта.

Создание массивов из Non-Reifiable Type в Java запрещено.

Ты можешь в этом убедиться, если попробуешь вручную создать массив Pair<String, String>

```
public static void main(String[] args) {

   //  ошибка компиляции! Generic array creation
  Pair<String, String>[] array = new Pair<String, String>[10];
}
```

Причина очевидна — типобезопасность. Как ты помнишь, при создании массива тебе обязательно нужно указать, какие объекты (или примитивы) будет хранить этот массив.

```
int array[] = new int[10];
```

На одном из прошлых занятий мы подробно разобрали механизм стирания типов.

Так вот, в данном случае мы в результате стирания типов потеряли информацию о том, что в наших объектах `Pair` хранились пары `<String, String>`. Создание массива будет небезопасным.

При использовании методов с `varargs` и дженериками обязательно помни о стирании типов и о том, как именно оно работает.

связанные с `varargs` предупреждения при помощи аннотации `@SafeVarargs`

```
@SafeVarargs
public static <E> void addAll(List<E> list, E... array) {

   for (E element : array) {
       list.add(element);
   }
}
```

Если ты добавишь к своему методу эту аннотацию, предупреждение, с которым мы столкнулись ранее, появляться не будет.

Еще одна возможная проблема при совместном использовании `varargs` и дженериков, — загрязнение кучи (heap pollution).

Загрязнение может возникнуть вот в такой ситуации:

```
import java.util.ArrayList;
import java.util.List;

public class Main {

   static List<String> makeHeapPollution() {
       List numbers = new ArrayList<Number>();
       numbers.add(1);
       List<String> strings = numbers;
       strings.add("");
       return strings;
   }

   public static void main(String[] args) {

       List<String> stringsWithHeapPollution = makeHeapPollution();

       System.out.println(stringsWithHeapPollution.get(0));
   }
}
```

Вывод в консоль:

```
Exception in thread "main" java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.Stri
```

Говоря простым языком, загрязнение кучи — это ситуация, при которой в куче должны находиться объекты типа `A`, но в результате там оказываются объекты типа `B` — из-за ошибок, связанных с типобезопасностью.

В нашем примере это и происходит. Сначала мы создали Raw-переменную `numbers`, и присвоили ей дженерик-коллекцию `ArrayList<Number>`. После этого мы добавили туда число `1`.

```
1  List<String> strings = numbers;
```

В этой строке компилятор пытался предупредить нас о вероятных ошибках, выдав предупреждение “*Unchecked assignment...*”, но мы проигнорировали его.

В результате у нас есть дженерик-переменная типа `List<String>`, которая указывает на дженерик-коллекцию типа `ArrayList<Number>`. Эта ситуация явно может привести к неприятностям!

Так и происходит. Используя нашу новую переменную, мы добавляем в коллекцию строку. Произошло загрязнение кучи — мы добавили в типизированную коллекцию сначала число, а потом строку. Компилятор нас предупреждал, но мы его предупреждение проигнорировали, получив результате `ClassCastException` только во время работы программы.

Причем же здесь `varargs`?

Использование `varargs` с дженериками запросто может привести к загрязнению кучи.

Вот простой пример:

```
1  import java.util.Arrays;
2  import java.util.List;
3
4  public class Main {
5
6     static void makeHeapPollution(List<String>... stringsLists) {
7         Object[] array = stringsLists;
8         List<Integer> numbersList = Arrays.asList(66,22,44,12);
9
10        array[0] = numbersList;
11        String str = stringsLists[0].get(0);
12    }
13
14    public static void main(String[] args) {
15
16        List<String> cars1 = Arrays.asList("Ford", "Fiat", "Kia");
17        List<String> cars2 = Arrays.asList("Ferrari", "Bugatti", "Zaporozhets");
18
19        makeHeapPollution(cars1, cars2);
20    }
21 }
```

Что здесь происходит?

НАЧАТЬ ОБУЧЕНИЕ

Из-за стирания типов наши листы-параметры (будем называть их “листами” вместо “списков” для удобства) —

```
List<String>...stringsLists
```

— превратятся в массив листов — `List[]` с неизвестным типом (не забывай, что varargs в результате компиляции превращается в обычный массив).

Из-за этого мы легко можем произвести присвоение в переменную `Object[] array` в первой строке метода — типы-то из наших листов стерлись!

И теперь у нас есть переменная типа `Object[]`, куда можно добавлять вообще что угодно — все объекты в Java наследуются от `Object`!

Сейчас у нас только массив строковых листов. Но благодаря использованию `varargs` и стирания типов мы легко можем добавить к ним лист, состоящий из чисел, что мы и делаем.

В результате мы загрязняем кучу из-за смешивания объектов разных типов. Результатом будет все то же исключение `ClassCastException` при попытке прочитать строку из массива.

Вывод в консоль:

```
Exception in thread "main" java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.Stri
```

Вот к таким неожиданным последствиям может привести использование простого, казалось бы, механизма `varargs` :)

А наша сегодняшняя лекция на этом подходит к концу.

Не забудь решить пару задач, а если останутся время и силы — изучить дополнительную литературу.

“Effective Java” сама себя не прочитает! :)

До встречи!

```java
import java.util.Arrays;
import java.util.List;

public class Main {

   static void makeHeapPollution(List<String>... stringsLists) {
       Object[] array = stringsLists;
       List<Integer> numbersList = Arrays.asList(66,22,44,12);

       array[0] = numbersList;
       String str = stringsLists[0].get(0);
   }

   public static void main(String[] args) {

       List<String> cars1 = Arrays.asList("Ford", "Fiat", "Kia");
       List<String> cars2 = Arrays.asList("Ferrari", "Bugatti", "Zaporozhets");

       makeHeapPollution(cars1, cars2);
   }
}
```

У нас за счет vararg появляется возможность передать в метод сразу два списка, в коде при этом я не могу понять как это работает ибо массиву присваивается список (Object[] array = stringsLists) и никак не описывается ситуация где этих списков несколько. Интересует связь между обоими списками и массивом.

Ответить 0

Anonymous #3068853 Уровень 3 28 мая, 03:40

stringsLists это не список, а массив списков. Её тип List[].
В момент вызова имеет место:
stringsLists[0] == cars1 и stringsLists[1] == cars2.

Ответить +2

Джама Уровень 51, Азербайджан 30 мая, 14:42

Спасибо большое, затупила немного)

Ответить 0

Jet Уровень 24, Санкт-Петербург, США 25 мая, 02:44

24.05.2022 Google:
Reifiable - Повторяемый

Ответить 0

Мирослав Уровень 27, Тбилиси, Грузия 12 мая, 11:26

Конечно не всё так плачебно но все равно как-то так!

Ответить 0

Anonymous #3068853 Уровень 3 28 мая, 03:41

Видео начало проигрываться у меня в голове раньше, чем я перешёл по ссылке)))

Ответить +1

hidden #2595317 Уровень 45 17 марта, 20:32

Ответить 0

Алексей Макаенко PL/SQL Developer в **ООО ИТМ** 7 ноября 2021, 11:59

В этой статье:
"Reifiable-type — это тип, информация о котором полностью доступна во время выполнения.
В языке Java к ним относятся примитивы, RAW-TYPES, а также типы, не являющиеся дженериками."

В следующей статье "Стирание типов":
"Raw Type — это класс-дженерик, из которого удалили его тип."

Я один здесь вижу противоречие? Может быть Raw Type относится все-таки к Non-Reifiable Types?

Ответить 0

Ян Уровень 41, Лида, Беларусь 17 ноября 2021, 00:26

в последнем примере, наверное, имелось ввиду, что возникнет ClassCastException, при попытке

```java
static void makeHeapPollution(List<String>... stringsLists) {
       Object[] array = stringsLists;
       List<Integer> numbersList = Arrays.asList(66,22,44,12);

       array[0] = numbersList;
       String str = array[0].get(0);
   }
```

или я что-то не так понял?

Ответить 0

Roma Уровень 39, Черкассы, Украина 10 января, 15:14

Дело в том, что при присваивании в array объекта stringLists, оба массива ссылаются на один объект. И если мы изменим массив array, то изменится и массив stringLists который ссылается на него. Тогда в stringLists окажется список numbersList.

Ответить +3

Ян Уровень 41, Лида, Беларусь 11 января, 22:43

Спасибо, очень долго с этим разбирался, пытался подогнать под механизм работы, но в конченом счёте так и недоразобрался. После Вашего комментария всё стало понятно(:

Ответить 0

Anonymous #3068853 Уровень 3 28 мая, 03:45

"Raw Type — это класс-дженерик, из которого удалили его тип."
Это надо понимать неформально. Так же как и "круг это правильный многоугольник с бесконечным числом сторон". На самом деле никакой круг не является правильным многоугольником. И никакой сырой тип не является дженериком.

Ответить 0

Edil Kalmamatov Уровень 35 28 сентября 2021, 11:04

Update от Google 28.09.2021:
Reifiable - Реифицируемый

Ответить +1

Сергей Васильев Уровень 37, Санкт-Петербург, Россия 12 марта, 00:03

12.03.2022:
Reifiable - Повторяемый
:)

Ответить 0

Pig Man Главная свинья в **Свинарнике** 11 февраля 2021, 18:19

```java
public static <E> void addAll(List<E> list, E... array) {
    for (E element : array) {
        list.add(element);
    }
}

public static void main(String[] args) {
    ArrayList<Pair<String, String>> pairs = new ArrayList<>();
    addAll(pairs,
            new Pair<String, String>("Leonardo", "da Vinci"),
            new Pair<String, String>("Vasco", "de Gama")
    );

    System.out.println(pairs);
}

// [Leonardo=da Vinci, Vasco=de Gama]
```

Эм.. да нет, это прекрасно работает

Ответить 0

koshi Уровень 16, Уфа, Россия 23 августа 2021, 08:56

И вот здесь мы неожиданно получаем предупреждение:
Unchecked generics array creation for varargs parameter

Ответить 0

Алексей Климов Уровень 4, Москва, Россия 3 декабря 2020, 15:50

*спасибо. Буп, ты очень помог... *

Антропопий Уровень 12 — 2 октября 2020, 12:02

Непонял...

Ответить +2

Alukard Vampire hunter в **The Hellsing** EXPERT — 27 октября 2020, 05:25

Пойми

Ответить +3

LindX Java Developer в **Home** — 11 декабря 2020, 01:49

Кто не понял тот поймет

Ответить 0

LindX Java Developer в **Home** — 11 декабря 2020, 01:49

ежи

Ответить +1

Показать еще комментарии