

Generics

Java Collections
5 уровень, 3 лекция

ОТКРЫТА

- Привет, Амиго!
- Привет, Элли!
- Сегодня мы с Ришей собираемся рассказать тебе все о generic'ах.
- Так я уже вроде бы почти все знаю.
- Почти все, да не все.
- Да? Ладно, я готов слушать.
- Тогда начнем.

Generic'ами в Java, называют классы, которые содержат типы-параметры.

Причины появления generic'ов – см. комментарий в коде:

Пример

```
1  ArrayList stringList = new ArrayList();
2  stringList.add("abc"); //добавляем строку в список
3  stringList.add("abc"); //добавляем строку в список
4  stringList.add( 1 ); //добавляем число в список
5
6  for(Object o: stringList)
7  {
8      String s = (String) o; //тут будет исключение, когда дойдем до элемента-числа
9  }
```

Как решают проблему Generic'и:

Пример

```
1  ArrayList<String> stringList = new ArrayList<String>();
2  stringList.add("abc"); //добавляем строку в список
3  stringList.add("abc"); //добавляем строку в список
4  stringList.add( 1 ); //тут будет ошибка компиляции
5
6  for(Object o: stringList)
7  {
8      String s = (String) o;
9  }
```

Такой код просто не скомпилируется, и ошибка с добавлением данных не того типа будет замечена еще на этапе компиляции.

- Я это уже знаю.

Но разработчики Java немного схаливали при создании Generic и вместо того, чтобы делать полноценные типы с параметрами прикрутили хитрую оптимизацию. На самом деле в типы не добавили никакой информации о их типах-параметрах, а вся магия происходила на этапе компиляции.

Код с generic'ами	Что происходит на самом деле
<pre>1 List<String> strings = new ArrayList<String>() 2 strings.add("abc"); 3 strings.add("abc"); 4 strings.add(1); // тут ошибка компиляции 5 6 for(String s: strings) 7 { 8 System.out.println(s); 9 }</pre>	<pre>1 List strings = new ArrayList(); 2 3 strings.add((String)"abc"); 4 strings.add((String)"abc"); 5 strings.add((String) 1); //ошибка компиляции 6 7 for(String s: strings) 8 { 9 System.out.println(s); 10 }</pre>

— Хитро, да.

— Да, но у этого подхода есть побочный эффект. **Внутри класса-generic'a не хранится никакой информации о его типе-парамetre**. Такой подход позже назвали **стиранием типов**.

Т.е. если у тебя есть свой класс с типами-параметрами, то ты не можешь использовать информацию о них внутри класса.

Код с generic'ами	Что происходит на самом деле
<pre>1 class Zoo<T> 2 { 3 ArrayList<T> pets = new ArrayList<T>(); 4 5 public T createAnimal() 6 { 7 T animal = new T(); 8 pets.add(animal) 9 return animal; 10 } 11 }</pre>	<pre>1 class Zoo 2 { 3 ArrayList pets = new ArrayList(); 4 5 public Object createAnimal() 6 { 7 Object animal = new ???(); 8 pets.add(animal) 9 return animal; 10 } 11 }</pre>

При компиляции все типы параметров заменяются на Object. И информации о типе, который в него передавали, в классе нет.

— Да, согласен, это не очень хорошо.

— Ладно, все не так страшно, я тебе позже расскажу, как эту проблему научились обходить.

Но и это еще не все. Java позволяет задать тип-родитель для типов-параметров. Для этого используется ключевое слово extends. Пример:

Код с generic'ами	Что происходит на самом деле
<pre>1 class Zoo<T extends Cat> 2 { 3 T cat; 4 5 T getCat() 6 {</pre>	<pre>1 class Zoo 2 { 3 Cat cat; 4 5 Cat getCat() 6 {</pre>

НАЧАТЬ ОБУЧЕНИЕ

9		9	
10	void setCat (T cat)	10	void setCat(Cat cat)
11	{	11	{
12	this .cat = cat;	12	this .cat = cat;
13	}	13	}
14		14	
15	String getCatName()	15	String getCatName()
16	{	16	{
17	return this .cat.getName();	17	return this .cat.getName();
18	}	18	}
19	}	19	}

Обрати внимание на два факта.

Во-первых, в качестве типа параметра теперь можно передать не любой тип, а тип `Cat` или один из его наследников.

Во-вторых, в классе `Zoo` у переменных типа `T` теперь можно вызвать методы класса `Cat`. Почему – объяснено в столбце справа (потому что вместо типа `T` везде подставится тип `Cat`)

— Ага. Если мы сказали, что в качестве типа-параметра у нас передают тип `Cat` или его наследников, значит, мы уверены, что методы класса `Cat` у типа `T` обязательно есть.

Что ж. Умно.

< Предыдущая лекция

Следующая лекция >

− +63 +

Комментарии (30)

популярные

новые

старые

JavaCoder

Введите текст комментария

Maks Panteleev Java Developer в Bell Integrator

15 июля 2021, 17:37

Дженерики это как рекурсия и this - ты либо чувствуешь сердцем, либо таращишь глаза и плачешь

Ответить

− +25 +

Hidden #213 Уровень 13 (Forever&Ever)

27 марта, 10:22

😭😭😭 как боженька смолвил

Ответить

− +2 +

Oleg Khilko Уровень 47, Москва, Russian Federation

9 августа, 17:47

Макс, ты шикарен!👍👍

Отлично сказано!

Ответить

− 0 +

Daria Scala Dev

28 января 2021, 20:41

Просто пропиарю здесь огромный [FAQ по дженерикам](#) из рекомендаций Хорстманна, куда можно заглядывать каждый раз, когда забыл, как быстро и с гарантией устроить hear pollution, или задаёшься смутными вопросами "Можно ли заоверрайдить дженерик метод другим дженерик методом?" "Почему нет епит-дженериков?" и "Кто такая эта ваша реификация".)

НАЧАТЬ ОБУЧЕНИЕ

NikeMirum

Уровень 36, Великий Новгород, Россия

20 октября 2020, 10:32

...

Вроде бы простая тема, но у меня почему-то вызвала больше всего недоумения из всех пройденных. Особенно сложно въезжать было в:
"Но разработчики Java немного схаливали при создании Generic и вместо того, чтобы делать полноценные типы с параметрами прикрутили хитрую оптимизацию. На самом деле в типы не добавили никакой информации о их типах-параметрах, а вся магия происходила на этапе компиляции."

Ответить

+1

Максим

Уровень 36, Воронеж, Россия

5 марта 2020, 09:48

...

Очередная лекция на наркоманском языке. И как всегда комментарии рулят)))

Ответить

+20

Vorlock

Уровень 31, Днепр, Украина

23 января 2020, 20:43

...

в сэмпле "Код с generic'ами" и "Что происходит на самом деле" ошибка в 8й строке - пропущен ","
и что это за ересь "Object animal = new ???();"

зы: за такой стайл написания на проектах ногами бьют

Ответить

0

Justinian

Judge в Mega City One

MASTER

2 мая 2020, 03:53

...

злые у вас там на проекте, постарайся больше так не подставляться

Ответить

+9

Даниил

Salesforce Developer в Viseven

MASTER

8 сентября 2019, 14:53

...

Кто не понял, тот поймёт. - Сергей Дружко.
[Обобщения \(Generics\)](#)

Ответить

+28

Самуил Олегович

Уровень 41, Киев, Украина

5 декабря 2019, 19:11

...

Я по ходу тот самый кто должен понять;)

Ответить

+3

Stanislav lushin

Уровень 30, Москва, Россия

6 августа 2019, 15:48

...

А в чем будет различие между Zoo<T extends Cat> и Zoo<Cat> ?

Ответить

+8

Даниил

Salesforce Developer в Viseven

MASTER

7 сентября 2019, 20:28

...

Честно говоря интересно как такое будет вообще работать (я в том смысле что я реально не знаю). Ну то есть вместо объявления Zoo<T extends Cat> написать Zoo<Cat>? Если так можно, то уверен что это не имеет смысла так как проще вообще тогда не указывать параметризированный тип, а просто использовать класс Cat в данном классе без заморочек, так как это будет одно и то же.

Ответить

+1

Даниил

Salesforce Developer в Viseven

MASTER

8 сентября 2019, 14:13

...

В общем как я понял **Zoo<T extends Cat>** может принимать любые типы от Cat до любого его наследника при этом используя все его возможности (отдельные методы), а **Zoo<Cat>** может так же принимать всех наследников Cat, но при этом эти типы буду приведены автоматически к типу Cat и тогда не можно будет использовать все возможности каждого отдельного наследника. По аналогии как сохранить объект в переменную типа Object - нельзя будет пользоваться никаким методами кроме тех что определены в классе Object.

Ответить

+8

Ivan

Уровень 34, Нижний Новгород, Россия

EXPERT

29 ноября 2019, 11:23

...

не, в Zoo<Cat> не будет ничего автоматически приводиться. Cat в данном случае просто случайной название типа-параметра, тоже самое если бы мы написали Zoo<T>, то есть Cat просто слово не привязанное классу, оно просто для нашего понимания работы класса.

Например есть класс Cat с методом says()

1

public class Cat {

2

public void says(){

3

System.out.println("мяяя");

4

}

5

}

И если мы в типа-параметре напишем Zoo2 <Cat>

1

public class Zoo2 <Cat> {

2

Cat cat;

3

НАЧАТЬ ОБУЧЕНИЕ

```
7      }
8
9      void setCat(Cat cat)
10     {
11         this.cat = cat;
12     }
13
14     void sayCatVoice()
15     {
16         ((com.javarush.task.testing.Cat)this.cat).says();
17     }
18 }
```

то мы у переменной cat даже не сможем вызвать says() без явного приведения к классу Cat, потому что тип переменной cat не класс Cat, а тип-параметр Cat.

Ну а при создании объекта Zoo2 мы сможем передавать туда любые объекты, хоть Cat хоть Object. Что плохо, так как при засовывании Objecta вылетит ClassCastException, из за метода says().

А Zoo1 сможет принимать только Cat и наследников, и IDEA не позволит засунуть туда кого то ещё)

Ответить

👍

+15

👍

Ivan

Уровень 34, Нижний Новгород, Россия

EXPERT

29 ноября 2019, 11:39

⋮

кстати если захотеть то можно и Zoo<Cat extends Cat> написать, чтобы в коде была не неизвестная T, а Cat))

Хотя по сути в классе Zoo дженерики вообще не нужны и можно было спокойно оставить правый вариант из лекции.

Ответить

👍

+5

👍

Даниил

Salesforce Developer в Viseven

MASTER

29 ноября 2019, 17:08

⋮

Спасибо за пояснение, реально логично, но не очень очевидно...

Ответить

👍

0

👍

Simon Bashkirov

Уровень 40

23 марта 2020, 17:38

⋮

Zoo<T extends Cat> понадобится, когда мы в клиентском коде захотим использовать зоопарки для конкретных наследников котов.

Например есть у нас class Tiger extends Cat {} и class Leo extends Cat {}

Тогда мы сможем где-то написать Zoo<Tiger> tigerZoo = new Zoo<>();

Смысл это приобретает тогда, когда у нас в Zoo есть методы, возвращающие T (или дженерики от T). Тогда эти методы при вызове на tigerZoo будут возвращаться тигров или (дженерики от тигров), а не просто котов. А в примере в лекции смысла в дженерике нет, можно смело использовать правый вариант, имхо.

Ответить

👍

+2

👍

Andrey Karelin

Уровень 41, Sumy, Украина

1 мая, 22:15

⋮

Еще раз, очевидное, но мало где проговариваемое....

Дженерик тип "T" (указанный в угловых скобках) - это условный тип. Это не тип Cat, String, Void и т.п., а скажем так, любая комбинация букв: T, E, Elknld, fuckThisJob и т.п. Она лишь для того, чтобы четко определить, что если класс например Zoo<T> (типа T), то его метод T getCat(), обязан вернуть точно такой же тип, которым определяли объект Zoo. Если Zoo<String>, то getCat() вернет стринг, если Zoo<Dog>, то getCat() вернет Dog и т.д.

Ответить

👍

0

👍

Макс

Уровень 26, Киев, Украина

EXPERT

4 июня 2019, 09:41

⋮

- 1) что это такое: Object animal = new ???();
- 2) зачем реально нужны эти дженерики?

Ответить

👍

0

👍

Sergey Simonov

Java Developer в МТС Умный дом

19 июня 2019, 13:05

⋮

1. ты не можешь создать экземпляр T. компилятор понимает только конкретные типы.
2. как минимум для того что бы не ловить ClassCastException. не говоря уже о том что ты можешь писать более компактный код

Ответить

👍

+1

👍

Nail

Уровень 23, Кельн

22 августа 2019, 23:49

⋮

как я понял, что бы получать ошибки на этапе написания программы от компилятора, а не от злых клиентов.

Ответить

👍

+1

👍

yury

Уровень 41, Москва, Россия

5 сентября 2019, 23:52

⋮

- 2) Чтобы писать универсальный код, а не создавать классы для всех возможных вариантов. Мы с

1

List list = new ArrayList();

2

list.add(Integer.valueOf(111));

3

list.add("test");

То при извлечении и информация о типах будет убита. Все объекты приведутся к базовому Object.

А если мы напишем конкретно

1

List<String> list = new ArrayList<>();

то компилятор будет знать, что список предназначен строго для String, без вариантов.

Ответить

+5

King

Уровень 41, Санкт-Петербург, Россия

13 февраля 2019, 15:44

Самое смешное, что ни одна опечатка до сих пор не исправлена.

Ответить

0

Павел X.

Team Lead в Иннотех

EXPERT

17 декабря 2018, 18:41

От количества опечаток в данной статье пошла кровь из глаз. Давно добавили бы элементарный функционал типа "выдели текст и нажми комбинацию клавиш, чтобы сообщить об опечатке".

Ответить

+8

Mikhail Bodrov

Уровень 41

7 сентября 2020, 18:38

для этого программистов нужно нанимать, деньги тратить... 🤪

Ответить

+2

Показать еще комментарии

ОБУЧЕНИЕ

- Курсы программирования
- Курс Java
- Помощь по задачам
- Подписки
- Задачи-игры

СООБЩЕСТВО

- Пользователи
- Статьи
- Форум
- Чат
- Истории успеха
- Активности

КОМПАНИЯ

- О нас
- Контакты
- Отзывы
- FAQ
- Поддержка



JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ



НАЧАТЬ ОБУЧЕНИЕ

