Приведение типов. Расширение и сужение

Java Core (/quests/QUEST_JAVA_CORE)
4 уровень (/quests/lectures/?quest=QUEST_JAVA_CORE&level=4), 3 лекция (/quests/lectures/questcore.level04.lecture03)

ОТКРЫТА

— Привет, Амиго! Тема сегодняшней лекции – расширение и сужение типов. С расширением и сужением примитивных типов ты познакомился уже давно. На 10 уровне. Сегодня мы расскажем, как это работает для ссылочных типов, т.е. для объектов классов.

Тут все довольно просто на самом деле. Представь себе цепочку наследования класса: класс, его родитель, родитель родителя и т.д. до самого класса Object. Т.к. класс содержит все методы класса, от которого он был унаследован, то объект этого класса можно сохранить в переменную любого из его типов родителей.

Пример:

Код

Описание

```
1  class Pet
2  {
3  public void doPetActions();
4  }class Cat extends Pet
5  {
6  public void doCatActions();
7  }class Tiger extends Cat
8  {
9  public void doTigerActions();
10 }
```

Тут мы видим три объявленных класса: животное, кот и тигр. Кот наследуется от Животного. А Тигр от Кота.

```
1  public static void main(String[] args)
2  {
3  Tiger tiger = new Tiger();
4  Cat cat = new Tiger();
5  Pet pet = new Tiger();
6  Object obj = new Tiger();
7  }
```

Объект класса Tiger всегда можно спокойно присвоить переменной с типом класса-родителя. Для класса Tiger – это Cat, Pet и Object.

Теперь рассмотрим, что же такое расширение и сужение типов.

Если в результате присваивания мы двигаемся по цепочке наследования вверх (к типу Object), то это — расширение типа (оно же — восходящее преобразование или upcasting), а если вниз, к типу объекта, то это — сужение типа (оно же — нисходящее преобразование или downcasting).

Движение вверх по цепочке наследования называется расширением, поскольку оно приводит к более общему типу. Но при этом теряется возможность вызвать методы, которые были добавлены в класс при наследовании.

Код

Описание

```
1 public static void main(String[] args)
2 {
3 Object obj = new Tiger();
4 Pet pet = (Pet) obj;
5 Cat cat = (Cat) obj;
6 Tiger tiger = (Tiger) pet;
7 Tiger tiger2 = (Tiger) cat;
8 }
```

При сужении типа, нужно использовать оператор преобразования типа, то есть мы выполняем явное преобразование.

При этом Java-машина выполняет проверку, а действительно ли данный объект унаследован от Типа, к которому мы хотим его преобразовать.

Такое небольшое нововведение уменьшило количество ошибок в преобразовании типов в разы, и существенно повысило стабильность работы Java-программ.

Код

Описание

```
1 public static void main(String[] args)
2 {
3 Object obj = new Tiger();
4 if (obj instanceof Cat)
5 {
6 Cat cat = (Cat) obj;
7 cat.doCatActions();
8 }}
```

Еще лучше – использовать проверку instanceof

```
public static void main(String[] args)
{
Pet pet = new Tiger();
doAllAction(pet);

Pet pet2 = new Cat();
doAllAction(pet2);

Pet pet3 = new Pet();
doAllAction(pet3);
}

public static void doAllAction(Pet pet)
{
if (pet instanceof Tiger)
{
Tiger tiger = (Tiger) pet;
tiger.doTigerActions();
}

if (pet instanceof Cat)
{
Cat cat = (Cat) pet;
cat.doCatActions();
}

pet.doPetActions();
}
```

И вот почему. Смотрим на пример слева.

Мы (наш код) не всегда знаем, с объектом какого типа мы работаем. Это может быть как объект того же типа, что и переменная (Pet), так и любой тип-наследник (Cat, Tiger).

Рассмотрим метод doAllAction. Он корректно работает в независимости от того, объект какого типа в него передали.

Т.е. он корректно работает для всех трех типов Pet, Cat, Tiger.

```
public static void main(String[] args)
{
Cat cat = new Tiger();
Pet pet = cat;
Object obj = cat;
}
```

Тут мы видим три присваивания. Все они являются примерами расширения типа.

Оператор преобразования типа тут не нужен, так как не нужна проверка. Ссылку на объект всегда можно сохранить в переменную любого его базового типа.

— О, на предпоследнем примере все стало понятно. И для чего нужна проверка, и для чего нужно преобразование типов.

— Надеюсь, что так. Хочу обратить твое внимание на следующую вещь:

С объектом при таком присваивании ничего не происходит! Меняется только количество методов, которое можно вызвать с помощью конкретной переменной-ссылки.

Например, переменная класса Cat позволяет вызывать методы doPetActions & doCatActions, и ничего не знает о методе doTigerActions, даже если ссылается на объект класса Tiger.

— Ну, это-то уже ясно. Оказалось легче, чем я думал.

‹ (/quests/lectures/questcore.level04.lecture02)

×14 › (/quests/lectures/questcore.level04.lecture04)

Комментарии (57) популярные новые старые

Никита

Введите текст комментария

junior 40 уровень, Уфа 5 февраля, 06:16

Тут в лекции напутано или я что-то не так понял?

Разве это расширение?:
Object obj = new Tiger();
Tiger tiger = (Tiger) pet; // <---------- вот тут

Когда мы напишем:
Animal animalCat = new Cat();
Animal animalDog = new YorkshireTerrier();
Это расширяющее приведение (или неявное).

Сужающее приведение(или явное) происходит в обратную сторону:
Animal animalCat = new Cat();
Cat cat =(Cat)animalCat;

Ответить +2

Stanislav 21 уровень, Москва 5 января, 15:38

Только мне не нравится расположение фигурных скобок в лекциях?

Ответить +19

Джахангир Хаджиханов 15 уровень 11 февраля, 04:11

Согласен.
Из-за такого расположения восприятие становится сложным.

Ответить 0

Рустам Сафин 16 уровень, Казань 28 декабря 2017, 22:54

я правильно понял, что оператор преобразования типа нужен только при расширении типа, а при сужении типа он не нужен?

Ответить 0

Alex 30 уровень 3 января, 14:14

Да, сужать (восходящее преобразование вроде) можно спокойно делать.
А чтоб расширять, надо оператор.

Ответить 0

Джонни 22 уровень 1 ноября 2017, 23:12

Хорошо. Допустим, мы объявили следующую переменную:

```
1   Pet pet = new Cat();
```

Применили расширение:

```
1   pet = (Cat) pet;
```

И почему после этого мы НЕ можем таким образом вызвать метод doCatActions()

```
1   pet.doCatActions(); //ошибка компилятора
```

И таким образом тоже не проканывает:

```
1   Cat cat = (Cat) pet;
```

А можем сделать это ТОЛЬКО через еще одно расширение! Минуя вообще те расширения типов.

```
1   ((Cat) pet).doCatActions();
```

Проверил в идее.

Почему так?

Ответить 0

Andrew Lan 24 уровень 2 ноября 2017, 16:46

```
1  Pet pet = new Cat();
2  Cat cat = (Cat) pet;
3  cat.doCatActions(); //так должно работать
4  // ((Cat) pet).doCatActions(); - это равносильно строке выше
```

А вообще в

```
1  pet = (Cat) pet;
```

осуществляется два преобразования - одно явное (именно сужающее, лекция вводит в заблуждение), другое неявное (с расширением типа), т.е. в результате ничего не меняется - ни тип pet не меняется, ни с самим объектом, на который она указывает, ничего не происходит! "Меняется только количество методов, которое можно вызвать с помощью конкретной переменной-ссылки", т.е. с помощью переменной Pet pet нельзя вызвать метод *doCatActions()* дочернего класса Cat, т.к. pet о нём ничего не знает. Как то так

Ответить +1

Джонни 22 уровень 2 ноября 2017, 21:01

Да, при таком объявлении

```
1  Pet pet = new Cat();
```

у переменной pet можно вызвать только те методы, которые имеются в классе Pet, так как у самой переменной тип Pet, несмотря на то, что сам объект является экземпляром класса Cat. Но почему тут

```
1  pet = (Cat) pet;
```

происходит ещё неявное расширение типа? Не могу понять.

Ответить 0

Andrew Lan 24 уровень 3 ноября 2017, 12:54

Если я правильно понимаю:
1. (Cat) pet - первое явное преобразование, сужающее; по ходу создается временная ссылка типа Cat (Cat tmp назовём так), если не вру - то за это отвечает компилятор;
2. pet = tmp - второе неявное преобразование, с расширением tmp до типа Pet; вот если бы на месте pet была другая переменная, типа Cat, тогда другое дело; и опять же если не вру - за это отвечает ява-машина.
Т.е. все остаются "при своих"

Ответить 0

swen922 30 уровень 5 ноября 2017, 01:09

Кажется, не совсем так: прописал в ИДЕЕ каждому из трех классов свой toString, вызвал сразу после создания:

Pet pet = new Cat();
System.out.println(pet.toString());

Так вот отвечает: I'm Cat
То есть текстом, проописанным в классе Cat.

Ответить 0

Джонни 22 уровень 5 ноября 2017, 10:18

Я понял как они проводят приведение типов. Они приводят не ту же самую переменную pet

```
1  pet = (Cat) pet;
```

потому что с как раз ничего не меняется, а создают новую cat

```
1  Pet pet = new Cat();
2  Cat cat = (Cat) pet;
```

и присваивают ей ссылку старой, плюс расширяют функционал путём приведения.
Вот и ответ на мой вопрос.
Спасибо! Разобрались-таки.

Полиморфизм такой полиморфизм.

Ответить +3

Джонни 22 уровень 5 ноября 2017, 10:25

Это из-за переопределения методов.
Тип переменной и тип объекта - две большие разницы.

```
1  Pet pet = new Cat();
```

Тип переменной pet:

```
1  Pet
```

Тип объекта переменной pet:

```
1  ...Cat();
```

Если метод *переопределён* в классе-наследнике, то за его вызов отвечает *тип объекта*.

```
1  ...Cat();
```

https://javarush.ru/quests/lectures/questcore.level02.lecture01

Ответить +5

Andrew Lan 24 уровень 5 ноября 2017, 18:42

Джонни, именно так)

Ответить 0

Andrew Lan 24 уровень 5 ноября 2017, 18:47

swen922, ну конечно же вызывается метод объекта типа Cat! Ссылка именно на этот объект храниться в переменной pet. Только маленькая хитрость - переменная pet имеет тип Pet))

Ответить 0

Владимир 23 уровень, Москва 6 ноября 2017, 11:28

Уф... Спасибо...)))

Ответить +1

swen922 30 уровень 6 ноября 2017, 20:05

Andrew Lan, спасибо, а можно тогда еще уточнить, пожалуйста: какой смысл того, что переменная pet имеет тип Pet, если она все равно ссылается на объект Cat? То есть, как я понимаю, по-сути и сама является объектом Cat (по крайней мере, использует все методы, свойственные Cat). Чем она тогда отличается от "настоящих" объектов Cat по своим свойствам?

Ответить 0

Andrew Lan 24 уровень 6 ноября 2017, 20:58

Смысл вот в чём. Pet - родительский (базовый) класс, или класс-предок, по отношению к Cat, дочернему классу, или классу-потомку. Как правило, классы-потомки пишут для того, чтобы расширить и/или уточнить (специфицировать) функциональность и/или свойства класса-предка, при этом в потомке могут появиться новые методы, а какие-то методы предка переопределяются. Так вот, через ссылку pet мы можем вызвать те методы, которые есть только в предке (в нашем случае - в Pet). Но, если они переопределены в потомке (Cat), то работает всё же переопределённый код.
"...по-сути и сама является объектом Cat (по крайней мере, использует все методы, свойственные Cat)..." - немного не так. Переменная pet НЕ является объектом, она содержит всего лишь ссылку на объект типа Cat, и с помощью этой ссылки, повторюсь, можно вызвать НЕ все методы объекта Cat.
А вообще, по большому счёту, использование ссылок с типом базового класса или реализованного интерфейса нужно для написания более универсального кода, с большей степенью абстракции, т.е. имеет целью упростить и облегчить код и проектирование классов. Как то так, мне кажется.

Ответить +2

Иван Лаврентьев 20 уровень, Москва 23 ноября 2017, 19:26

рискну добавить копейку своего понимания конструкции типа

```
1  Pet pet = new Cat();
```

в этом случае методы, объявленные в Cat становятся недоступными через ссылочную переменную типа pet, то есть имеет место инкапсуляция методов, которые не должны быть доступны других частей программы

Ответить +2

Инга-Виктория Куксова 19 уровень, Санкт-Петербург 9 октября 2017, 18:15

Зачем разводить демагогии, когда можно написать код и попробовать его скомпилировать?

Ответить +1

Артём Харитонов 16 уровень 2 октября 2017, 16:16

Возможно, некоторые моменты здесь упрощенны для наглядности, но хочу понять вот что:
для работы с несколькими типами данных в методе doAllAction можно же создать по методу с различными параметрами (для каждого из типов) и не использовать приведение типов. Или не прав я. Если можно и так, и так, то в чём разница? Спасибо

Ответить 0

Ренат 16 уровень 19 сентября 2017, 05:19

Ответить +4

Vladislav 26 уровень, Днепр 9 сентября 2017, 23:15

короче, UpCast - приведение экземпляра производного класса к базовому типу. (BaseClass up = new Derived Class);

DownCast - приведение экз.базового типа к произ.типу DeridevClass down = (Derived Class) up;

Ответить +4

Miau 21 уровень, Киев 6 сентября 2017, 13:26

Спасибо, все понятно и доходчиво)

Ответить 0

Redas Shuliakas 21 уровень 7 августа 2017, 11:40

Очень непонятно всё, и запутанно, нужно явно гуглить другие источники(((

Ответить 0

сергей климов 27 уровень 1 сентября 2017, 14:29

Первое слово в каждой строке - это тип ссылки.
Именно это слово открывает путь к доступным методам.
Если обьект типа Tiger сохранен в ссылке типа Object, то и иметь он будет только методы Object.
Несмотря на то, что твой обьект Tiger будет иметь кучу методов.

```
Object obj = new Tiger();
Pet pet = (Pet) obj;
Cat cat = (Cat) obj;
Tiger tiger = (Tiger) pet;
Tiger tiger2 = (Tiger) cat;
```

Ответить +10

Загрузить еще