Модификаторы доступа, переопределение методов, реализация абстрактных методов

Java Core (/quests/QUEST_JAVA_CORE)
5 уровень (/quests/lectures/?quest=QUEST_JAVA_CORE&level=5), 1 лекция (/quests/lectures/questcore.level05.lecture01)

ОТКРЫТА

— Я расскажу тебе про «модификаторы доступа». Когда-то я уже рассказывал про них, но повторение – мать учения.

Ты можешь управлять доступом (видимостью) методов и переменных твоего класса из других классов. Модификатор доступа отвечает на вопрос «Кто может обращаться к данному методу/переменной?». Каждому методу или переменной можно указывать только один модификатор.

1) Модификатор «public».

К переменной, методу или классу, помеченному модификатором public, можно обращаться из любого места программы. Это самая высокая степень открытости – никаких ограничений нет.

2) Модификатор «private».

К переменной, методу или классу, помеченному модификатором private, можно обращаться только из того же класса, где он объявлен. Для всех остальных классов помеченный метод или переменная – невидимы. Это самая высокая степень закрытости – только свой класс. Такие методы не наследуются и не переопределяются. Доступ к ним из класса-наследника также невозможен.

3) «Модификатор по умолчанию».

Если переменная или метод не помечены никаким модификатором, то считается, что они помечены «модификатором по умолчанию». Переменные и методы с таким модификатором видны всем классам пакета, в котором они объявлены, и только им. Этот модификатор еще называют «package» или «package private», намекая, что доступ к переменным и методам открыт для всего пакета, в котором находится их класс

4) Модификатор «protected».

Этот уровень доступа чуть шире, чем package. К переменной, методу или классу, помеченному модификатором protected, можно обращаться из его же пакета (как package), но еще из всех классов, унаследованных от текущего.

Таблица с пояснением:

Тип видимости	Ключевое слово	Доступ	Свой класс	Свой пакет	Класс — наследник	Все классы

Закрытый	private	Есть	Нет	Нет	Нет
Пакет	(нет модификатора)	Есть	Есть	Нет	Нет
Защищенный	protected	Есть	Есть	Есть	Нет
Открытый	public	Есть	Есть	Есть	Есть

Есть способ, чтобы легко запомнить эту таблицу. Представь себе, что ты составляешь завещание и делишь все вещи на четыре категории. Кто может пользоваться твоими вещами?

Кто имеет доступ	Модификатор	Пример
Только я сам	private	Личный дневник
Семья	(нет модификатора)	Семейные фотографии
Семья и наследники	protected	Фамильное поместье
Все	public	Мемуары

— Если представить, что классы, лежащие в одном пакете, – это одна семья, то очень даже похоже.

— Хочу также рассказать тебе несколько интересных нюансов насчет переопределения методов.

1) Неявная реализация абстрактного метода.

Допустим, у тебя есть код:

Код

```
1  class Cat
2  {
3   public String getName()
4   {
5    return "Васька";
6   }
7  }
```

И ты решил унаследовать от него класс тигр и добавить новому классу интерфейс

Код

```
1  class Cat
2  {
3   public String getName()
4   {
5     return "Васька";
6   }
7  }
```

```
1  interface HasName
2  {
3   String getName();
4   int getWeight();
5  }
```

```
1  class Tiger extends Cat implements HasName
2  {
3   public int getWeight()
4   {
5    return 115;
6   }
7
8  }
```

Если ты просто реализуешь все недостающие методы, которые тебе подскажет Intellij IDEA, то можешь потом долго искать ошибку.

Оказывается, что в классе Tiger есть унаследованный от Cat метод getName, который и будет считаться реализацией метода getName для интерфейса HasName.

— Не вижу в этом ничего страшного.

— Это не очень плохо, это скорее потенциальное место для ошибок.

Но может быть еще хуже:

Код

```
1  interface HasWeight
2  {
3   int getValue();
4  }
```

```
1  interface HasSize
2  {
3   int getValue();
4  }
```

```
1  class Tiger extends Cat implements HasWeight, HasSize
2  {
3   public int getValue()
4   {
5    return 115;
6   }
7  }
```

Оказывается, ты не всегда можешь унаследоваться от нескольких интерфейсов. Вернее унаследоваться можешь, а вот корректно их реализовать – нет. Посмотри на пример, оба интерфейса требуют, чтобы ты реализовал метод getValue(), и не ясно, что он должен возвращать: вес(weight) или размер(size). Это довольно-таки неприятная вещь, если тебе придется с ней столкнуться.

— Да, согласен. Хочешь реализовать метод, а не можешь. Вдруг ты уже унаследовал метод с таким же именем от базового класса. Обломись.

— Но есть и приятные новости.

2) Расширение видимости. При переопределении типа разрешается расширить видимость метода. Вот как это выглядит:

Код на Java

Описание

```
1  class Cat
2  {
3   protected String getName()
4   {
5    return "Васька";
6   }
7  }
```

```
1  class Tiger extends Cat
2  {
3   public String getName()
4   {
5    return "Василий Тигранович";
6   }
7  }
```

Мы расширили видимость метода с `protected` до `public`.

Использование

Почему это «законно»

```
1  public static void main(String[] args)
2  {
3   Cat cat = new Cat();
4   cat.getName();
5  }
```

Все отлично. Тут мы даже не знаем, что в классе-наследнике видимость метода была расширена.

```
1  public static void main(String[] args)
2  {
3   Tiger tiger = new Tiger();
4   tiger.getName();
5  }
```

Тут вызывается метод, у которого расширили область видимости.

Если бы этого сделать было нельзя, всегда можно было бы объявить метод в Tiger:
public String getPublicName()
{
super.getName(); //вызов protected метода
}

Т.е. ни о каком нарушении безопасности и речи нет.

```
1  public static void main(String[] args)
2  {
3   Cat catTiger = new Tiger();
4   catTiger.getName();
5  }
```

Если все условия подходят для вызова метода базового типа (Cat), то они уж точно подойдут для вызова типа наследника (Tiger) . Т.к. ограничения на вызов метода были ослаблены, а не усилены.

— Не уверен, что понял полностью, но то, что так можно делать, запомню.

3) Расширение типа результата.

В перегруженном методе мы можем поменять тип результата, расширив его.

Код на Java

Описание

```
1  class Cat
2  {
3   public Cat parent;
4   public Cat getMyParent()
5   {
6    return this.parent;
7   }
8   public void setMyParent(Cat cat)
9   {
10   this.parent = cat;
11  }
12 }
```

```
1  class Tiger extends Cat
2  {
3   public Tiger getMyParent()
4   {
5    return (Tiger) this.parent;
6   }
7  }
```

Мы переопределили метод `getMyParent` , теперь он возвращает объект типа `Tiger` .

Использование

Почему это «законно»

```
1  public static void main(String[] args)
2  {
3   Cat parent = new Cat();
4
5   Cat me = new Cat();
6   me.setMyParent(parent);
7   Cat myParent = me.getMyParent();
8  }
```

Все отлично. Тут мы даже не знаем, что в классе наследнике тип результата метода getMyParent был расширен.

«Старый код» как работал так и работает.

```
1  public static void main(String[] args)
2  {
3   Tiger parent = new Tiger();
4
5   Tiger me = new Tiger();
6   me.setMyParent(parent);
7   Tiger myParent = me.getMyParent();
8  }
```

Тут вызывается метод, у которого расширили тип результата.

Если бы этого сделать было нельзя, всегда можно было бы объявить метод в Tiger:
public Tiger getMyTigerParent()
{
return (Tiger) this.parent;
}

Т.е. ни о каком нарушении безопасности и/или контроля приведения типов нет речи.

```
1  public static void main(String[] args)
2  {
3   Tiger parent = new Tiger();
4
5   Cat me = new Tiger();
6   me.setMyParent(parent);
7   Cat myParent = me.getMyParent();
8  }
```

И тут все отлично работает, хотя мы сузили тип переменных до базового класса (Cat).

Благодаря перегрузке вызовется правильный метод setMyParent.

И нет ничего страшного при вызове метода getMyParent, т.к. его результат, хоть и класса Tiger, все равно сможет отлично присвоиться в переменную myParent базового класса (Cat).

Объекты Tiger можно смело хранить как в переменных класса Tiger, так и в переменных класса Cat.

— Ага. Я понял. Надо при переопределении методов беспокоится о том, как все это будет работать, если мы передадим наши объекты в код, который умеет обращаться только с базовым классом, и ничего о нашем классе не знает.

— Именно! Тогда вопрос на засыпку, почему нельзя сузить тип результата при переопределении метода?

— Это же очевидно, тогда перестанет работать код в базовом классе:

Код на Java

Пояснение проблемы

```
1  class Cat
2  {
3   public Cat parent;
4   public Cat getMyParent()
5   {
6    return this.parent;
7   }
8   public void setMyParent(Cat cat)
9   {
10   this.parent = cat;
11  }
12 }
```

```
1  class Tiger extends Cat
2  {
3   public Object getMyParent()
4   {
5    if (this.parent != null)
6     return this.parent;
7    else
8     return "я - сирота";
9   }
10 }
```

Мы переопределили метод getMyParent и сузили тип его результата.

Тут все отлично.

```
1  public static void main(String[] args)
2  {
3   Tiger parent = new Tiger();
4
5   Cat me = new Tiger();
6   Cat myParent = me.getMyParent();
7  }
```

Тогда у нас перестанет работать этот код.

Метод getMyParent может вернуть любой объект типа Object, т.к. на самом деле он вызывается у объекта типа Tiger.

А у нас нет проверки перед присваиванием. Тогда вполне возможно, что переменная myParent типа Cat будет хранить ссылку на строку.

— Отличный пример, Амиго!

В Java перед вызовом метода не проверяется, есть ли такой метод у объекта или нет. Все проверки происходит во время выполнения. И [гипотетический] вызов отсутствующего метода, скорее всего, приведет к тому, что программа начнет выполнять байт-код там, где его нет. Это, в конце концов, приведет к фатальной ошибке, и операционная система принудительно закроет программу.

— Ничего себе. Буду знать.

‹ (/quests/lectures/questcore.level05.lecture00)

×15 › (/quests/lectures/questcore.level05.lecture02)

Комментарии (49) популярные новые старые

Никита

Введите текст комментария

Александр 22 уровень, Москва 6 марта, 09:35

Не совсем понятно про переменные и методы с модификатором по умолчанию, как они себя ведут при наследовании. Верно ли я понял что даже внутри пакета они не наследуются для классов наследников? Или в чем его отличие от package?

Ответить 0

Filakkin 18 уровень, Новосибирск 19 марта, 07:23

Если наследовать класс внутри пакета, то все хорошо.

А вот если ты хочешь наследовать класс из другого пакета, то не получится. Чтобы сделать это нужен модификатор protected.

Ответить 0

Maxim Sivov 19 уровень, Taraz 3 марта, 17:09

Создали экземпляр кота типа Кота, который хранит в себе объект кота класса кота, приведенного к типу кота, ссылающуюся на переменную типа кота...

Ответить +3

Дмитрий 18 уровень, Москва 23 февраля, 20:44

Это уже было в более ранних лекциях и хорошо поясняется в книге "Изучаем Java":

1. То, какие методы мы можем вызвать, зависит от типа переменной.
2. Но то, какую версию метода мы вызовем, зависит от типа объекта.
3. Переменная более широкого (родительского) типа может ссылаться на объект более узкого (наследующего) типа. Это как более объёмный контейнер может вместить более маленький предмет. Но переменная узкого типа (Cat) не может ссылаться на объект более широкого типа (Object).

Ответить +1

Дмитрий Стуков 17 уровень, Санкт-Петербург 6 февраля, 19:22

было сложно, но я понял

Ответить 0

Fs Jt 20 уровень, Киев 25 января, 17:19

2) Расширение видимости.

А какая разница? Ведь даже если

```
1  class Cat
2  {
3   protected String getName()
4   {
5    return "Васька";
6   }
7  }
```

то System.out.println(tiger.getName()); выдаст "Васька". При условии, что этот метод не переопределен в классе Tiger, конечно. А переопределить его можно даже если в классе Cat метод getName public.

Ответить 0

cyprusScorpion 29 уровень 10 января, 14:10

Интересно, а возможен ли вызов метода getMyParent() из класса Object, что-то вроде того, что

Object me = new Tiger();
Object myParent = me.getMyParent();

если в классе Tiger этот метод сужен до Object:

class Tiger extends Cat
{

```
 public Object getMyParent()
 {
  if (this.parent != null)
   return this.parent;
  else
   return "я - сирота";
 }
}
```

Ответить +2

lex_lite 19 уровень 5 февраля, 20:26 •••

Нет, вызов не возможен. В переменной "me" Мы сузили тип объекта Tiger до Object. А у класса Object нет такого метода "getMyParent()".

Но при этом если у переменной "me" мы поменяем тип с Object на Tiger/Cat, то тогда программа скомпилируется. И переменная MyParent будет хранить строку "я сирота".

Ответить +3

Джонни 23 уровень 7 ноября 2017, 02:46 •••

Не понял смысл пункта 2.

Код 1:

```
1  public static void main(String[] args) {
2      Cat cat = new Cat();
3      System.out.println("Cat: " + cat.getName());
4
5      Tiger tiger = new Tiger();
6      System.out.println("Tiger: " + tiger.getName());
7
8      Cat catTiger = new Tiger();
9      System.out.println("catTiger: " + catTiger.getName());
10
11 }
12
13 static class Cat {
14     public String getName()
15     {
16         return "Васька";
17     }
18 }
19
20 static class Tiger extends Cat {
21     public String getName()
22     {
23         return "Тигран Васильевич";
24     }
25     public void mtd(){}
26 }
```

Вывод:
Cat: Васька
Tiger: Тигран Васильевич
catTiger: Тигран Васильевич

Код 2:

```
1  public static void main(String[] args) {
2      Cat cat = new Cat();
3      System.out.println("Cat: " + cat.getName());
4
5      Tiger tiger = new Tiger();
6      System.out.println("Tiger: " + tiger.getName());
7
8      Cat catTiger = new Tiger();
9      System.out.println("catTiger: " + catTiger.getName());
10
11 }
12
13 static class Cat {
14     protected String getName()
15     {
16         return "Васька";
17     }
18 }
19
20 static class Tiger extends Cat {
21     public String getName()
22     {
23         return "Тигран Васильевич";
24     }
25     public void mtd(){}
26 }
```

Вывод:
Cat: Васька
Tiger: Тигран Васильевич
catTiger: Тигран Васильевич

В чём смысл расширения видимости метода класса-наследника и в чём разница?

Ответить 0

Andrew Lan 24 уровень 7 ноября 2017, 20:16

В этом примере разницы действительно не увидишь, даже если разнесёшь классы Cat и Tiger по разным пакетам, всё равно резалт будет один. Здесь надо просто принять на веру факт: "При переопределении типа разрешается расширить видимость метода." (имеется ввиду - в дочернем классе).
В чём смысл расширения видимости метода класса-наследника? Ну, наверное, в том, что при переопределении метода в классе-наследнике (при специфицировании дочернего класса) ты можешь сделать специфицированные методы более открытыми для других классов и объектов программы. Т.е. получается, что базовые классы более закрытые, и это правильно, что бы как можно меньше объектов о них знало. Инкапсуляция, типа.

Ответить +13

Джонни 23 уровень 8 ноября 2017, 20:28

Я тебя понял, спасибо. (=

Ответить 0

Андрей Лисовский 29 уровень, Москва 18 октября 2017, 02:18

Почему изменение типа возвращаемого значения с Cat на Tiger называется расширением, когда обычно это называется сужением? Tiger более специфический класс, чем Cat

Ответить +10

Андрей Лисовский 29 уровень, Москва 14 ноября 2017, 02:16

С этим не поспоришь, но вот с Интуита:
"Расширение означает переход от более конкретного типа к менее конкретному, т.е. переход от детей к родителям."

http://www.intuit.ru/studies/courses/16/16/lecture/27117?page=2

Ответить 0

Андрей Лисовский 29 уровень, Москва 14 ноября 2017, 11:55

Так в статье по вашей ссылке тоже расширение - это преобразование к родителю (неявное), сужение - к потомку (явное).

Курс на интуите - довольно серьезный (уж точно не дилетантский). Но сложноват для новичка и без практики совсем.
В общем, пусть будут восходящие и нисходящие преобразования, как у Эккеля (интересно, как у других авторов)

Ответить 0

Ильяс 23 уровень, Москва 14 ноября 2017, 12:37

И от себя напишу (как я понимаю):
Поскольку ограничение на кол-во символов, то разделю на несколько постов.

```
1  public class SandBox{
2      public static void main(String[] args){
3          Parent p=new Parent();
4          Child c=new Child();
5
6          //Для Child доступны все методы от родителя, плюс свои.
7          //И их можно спокойно вызвать:
8          c.ParentMethod();       //Метод предка
9          c.OnlyChildMethod();    //Свой метод
10         c.method1();    //Сработает не как родительский, а как дочерний (он там перес
11         c.method2();    //Сработает не как родительский, а как дочерний (он там перес
12
13         //Для Parent доступны только свои методы:
14         p.ParentMethod();
15         p.OnlyChildMethod(); //Не правильно. Ошибка компиляции
16         p.method1();    //Сработает родительский вариант
17         p.method2();    //Сработает родительский вариант
```

Ответить +15

Ильяс 23 уровень, Москва 14 ноября 2017, 12:38

```
1  //Теперь немножко поколдуем ниже:
2
3  Parent c_as_p=new Child();  //Для C_as_P доступны только методы Parent (P.S. переопре
4                              //Приведение было неявным. Однако оно сужающее,
5                              //так как наследственные методы недоступны. Ему видны только
6  c_as_p.ParentMethod();     //Вот родительский метод, он спокойно доступен.
```

```
c_as_p.OnlyChildMethod(); //Не правильно. Метод наследника недоступен (не виден), хот
c_as_p.method1();   //Сделает всё как у наследника (этот метод там переопределён)
c_as_p.method2();   //Сделает всё как у наследника (этот метод там переопределён)

Child P_as_C=new Parent();          //Ошибка компиляции
Child P_as_C=(Child)new Parent();       //Ошибки компиляции не будет. Но потом выскоч
Parent p_ac_c=(Child)new Parent();  //Ошибки компиляции не будет. Но потом выскочит E



Parent c1=new Child(); //То же самое, что делали в 3 строке.
c1.OnlyChildMethod();  //Метод всё так же недоступен. Ошибка компиляции.

Child c2=(Child)c1;     //Создаем новую переменную (для того же объекта), где 100% пр
                    //Теперь доступны не только методы родителя, но и наследственные.
c2.OnlyChildMethod();   //А вот теперь метод стал доступным.

Parent c3=c2;           //Сужение обрано (неявное приведение). Аналогично (Parent)p_t
c3.OnlyChildMethod();   //Метод опять стал недоступен. Ошибка компиляции

//Работали с одним и тем же объектом, только хранили в разных переменных, с разным пр
```

Ответить +19

Ильяс 23 уровень, Москва 14 ноября 2017, 12:38

```
}
    static class Parent{
        public void ParentMethod(){
            System.out.println("Parent Method");
        }
        public void method1(){
            System.out.println("Parent method1");
        }
        public void method2(){
            System.out.println("Parent method2");
        }
    }
    static class Child extends Parent{
        public void OnlyChildMethod(){
            System.out.println("Only Child Method");
        }
        public void method1(){
            System.out.println("Child method1");
        }
        public void method2(){
            System.out.println("Child method2");
        }
    }
}
```

Ответить +12

Roxolana Shveda 17 уровень 30 сентября 2017, 19:26

а наследники это не семья? это же дети как бы :D

Ответить 0

Andrew Lan 24 уровень 4 ноября 2017, 18:21

Наверное, это дальние родственники - троюродные внучатые племянники))

Ответить +1

NazFarr 22 уровень 28 сентября 2017, 13:25

вроде бы понятно в теории раз 10 перечитал про эти модификаторы(public privte protected) а вот на практике 80 % моих ошибок именно из за модификаторов доступа. надеюсь после 15 уровня хоть как то улучшаться мои навыки.

Ответить 0

Загрузить еще