Полиморфизм и переопределение

Java Core (/quests/QUEST_JAVA_CORE)
2 уровень (/quests/lectures/?quest=QUEST_JAVA_CORE&level=2), 1 лекция (/quests/lectures/questcore.level02.lecture01)

ОТКРЫТА

— Амиго, ты любишь китов?

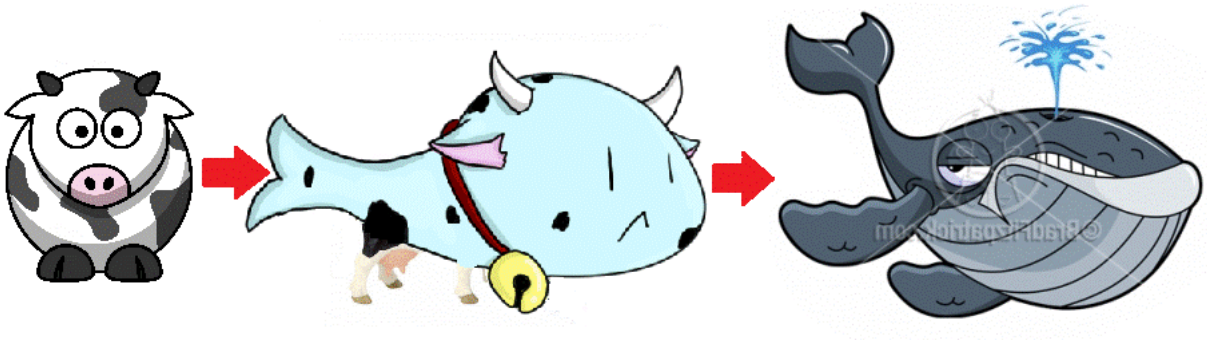
— Китов? Не, не слышал.

— Этот как корова, только больше и плавает. Кстати, киты произошли от коров. Ну, или имели общего с ними предка. Не столь важно.

— Так вот. Хочу рассказать тебе об еще одном очень мощном инструменте ООП – это полиморфизм. У него есть четыре особенности.

1) Переопределение метода.

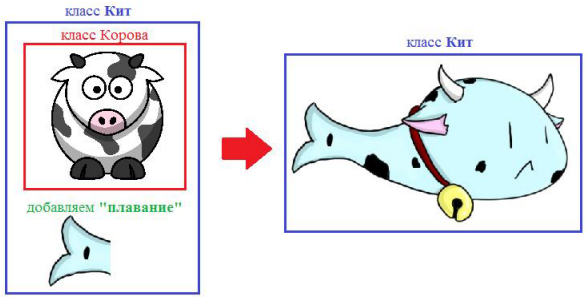
Представь, что ты для игры написал класс «Корова». В нем есть много полей и методов. Объекты этого класса могут делать разные вещи: идти, есть, спать. Еще коровы звонят в колокольчик, когда ходят. Допустим, ты реализовал в классе все до мелочей.



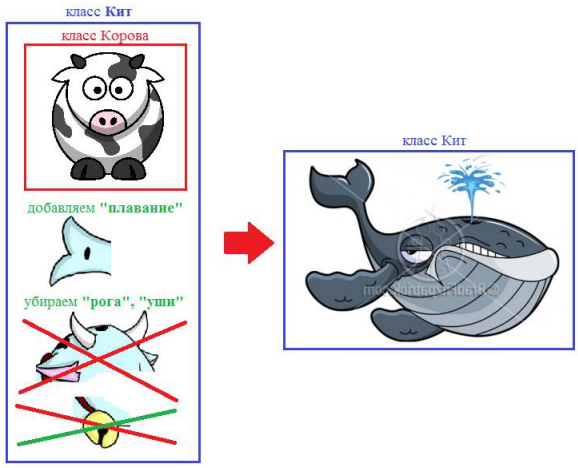
А тут приходит заказчик проекта и говорит, что хочет выпустить новый уровень игры, где все действия происходят в море, а главным героем будет кит.

Ты начал проектировать класс «Кит» и понял, что он лишь немного отличается от класса «Корова». Логика работы обоих классов очень похожа, и ты решил использовать наследование.

Класс «Корова» идеально подходит на роль класса-родителя, там есть все необходимые переменные и методы. Достаточно только добавить киту возможность плавать. Но есть проблема: у твоего кита есть ноги, рога и колокольчик. Ведь эта функциональность реализована внутри класса «Корова». Что тут можно сделать?



К нам на помощь приходит переопределение (замена) методов. Если мы унаследовали метод, который делает не совсем то, что нужно нам в нашем новом классе, мы можем заменить этот метод на другой.



Как же это делается? В нашем классе-потомке мы объявляем такой же метод, как и метод класса родителя, который хотим изменить. Пишем в нем новый код. И все – как будто старого метода в классе-родителе и не было.

Вот как это работает:

Код

Описание

```
1  class Cow
2  {
3  public void printColor()
4  {
5  System.out.println("Я - белая");
6  }
7  public void printName()
8  {
9  System.out.println("Я - корова");
10 }
11 }class Whale extends Cow
12 {
13 public void printName()
14 {
15 System.out.println("Я - кит");
16 }
17 }
```

Тут определены два класса `Cow` и `Whale`. `Whale` унаследован от `Cow`.

В классе `Whale` переопределен метод `printName();`

```
1  public static void main(String[] args)
2  {
3  Cow cow = new Cow();
4  cow.printName();
5  }
```

Данный код выведет на экран надпись «Я – корова»

```
1  public static void main(String[] args)
2  {
3  Whale whale = new Whale();
4  whale.printName();
5  }
```

Данный код выведет на экран «Я – кит»

После наследования класса `Cow` и переопределения метода `printName` , класс `Whale` фактически содержит такие данные и методы:

Код

Описание

```
1  class Whale
2  {
3  public void printColor()
4  {
5  System.out.println("Я - белая");
6  }
7  public void printName()
8  {
9  System.out.println("Я - кит");
10 }
11 }
```

Ни о каком старом методе мы и не знаем.

— Честно говоря, ожидаемо.

2) Но это еще не все.

— Предположим в классе `Cow` есть метод `printAll` , который вызывает два других метода, тогда код будет работать так:

На экран будет выведена надпись Я – белая Я – кит

Код

Описание

```
1  class Cow
2  {
3  public void printAll()
4  {
5  printColor();
6  printName();
7  }
8  public void printColor()
9  {
10 System.out.println("Я - белая");
11 }
12 public void printName()
13 {
14 System.out.println("Я - корова");
15 }
16 }
17
18 class Whale extends Cow
19 {
20 public void printName()
21 {
22 System.out.println("Я - кит");
23 }
24 }
```

```
1  public static void main(String[] args)
2  {
3  Whale whale = new Whale();
4  whale.printAll();
5  }
```

На экран будет выведена надпись
Я – белая
Я – кит

Обрати внимание, когда вызываются метод printAll() написанный в классе Cow, у объекта типа Whale, то будет использован метод printName класса Whale, а не Cow.

Главное, не в каком классе написан метод, а какой тип (класс) объекта, у которого этот метод вызван.

— Ясно.

— Наследовать и переопределять можно только нестатические методы. Статические методы не наследуются и, следовательно, не переопределяются.

Вот как выглядит класс Whale после применения наследования и переопределения методов:

Код

Описание

```
1  class Whale
2  {
3  public void printAll()
4  {
5  printColor();
6  printName();
7  }
8  public void printColor()
9  {
10 System.out.println("Я - белая");
11 }
12 public void printName()
13 {
14 System.out.println("Я - кит");
15 }
16 }
```

Вот как выглядит класс Whale, после применения наследования и переопределения метода. Ни о каком старом методе `printName` мы и не знаем.

3) Приведение типов.

Тут есть еще более интересный момент. Т.к. класс при наследовании получает все методы и данные класса родителя, то объект этого класса разрешается сохранять (присваивать) в переменные класса родителя (и родителя родителя, и т.д., вплоть до Object). Пример:

Код

Описание

```
1  public static void main(String[] args)
2  {
3  Whale whale = new Whale();
4  whale.printColor();
5  }
```

На экран будет выведена надпись
Я – белый.

```
1  public static void main(String[] args)
2  {
3  Cow cow = new Whale();
4  cow.printColor();
5  }
```

На экран будет выведена надпись
Я – белый.

```
1  public static void main(String[] args)
2  {
3  Object o = new Whale();
4  System.out.println(o.toString());
5  }
```

На экран будет выведена надпись
Whale@da435a.
Метод toString() унаследован от класса Object.

— Очень интересно. А зачем это может понадобиться?

— Это ценное свойство. Позже ты поймешь, что очень, очень ценное.

4) Вызов метода объекта (динамическая диспетчеризация методов).

Вот как это выглядит:

Код

Описание

```
1  public static void main(String[] args)
2  {
3  Whale whale = new Whale();
4  whale.printName();
5  }
```

На экран будет выведена надпись
Я – кит.

```
1  public static void main(String[] args)
2  {
3  Cow cow = new Whale();
4  cow.printName();
5  }
```

На экран будет выведена надпись
Я – кит.

Обрати внимание, что на то, какой именно метод printName вызовется, от класса Cow или Whale, влияет не тип переменной, а тип – объекта, на который она ссылается.

В переменной типа Cow сохранена ссылка на объект типа Whale, и будет вызван метод printName, описанный в классе Whale.

— Это не просто для понимания.

— Да, это не очень очевидно. Запомни главное правило:

Набор методов, которые можно вызвать у переменной, определяется типом переменной. А какой именно метод/какая реализация вызовется, определяется типом/классом объекта, ссылку на который хранит переменная.

— Попробую.

— Ты будешь постоянно сталкиваться с этим, так что скоро поймешь и больше никогда не забудешь.

5) Расширение и сужение типов.

Для ссылочных типов, т.е. классов, приведение типов работает не так, как для примитивных типов. Хотя у ссылочных типов тоже есть расширение и сужение типа. Пример:

Расширение типа

Описание

```
1  Cow cow = new Whale();
```

Классическое расширение типа. Теперь кита обобщили (расширили) до коровы, но у объекта типа Whale можно вызывать только методы, описанные в классе Cow.

Компилятор разрешит вызвать у переменной cow только те методы, которые есть у ее типа — класса Cow.

Сужение типа

Описание

```
1  Cow cow = new Whale();
2  if (cow instanceof Whale)
3  {
4  Whale whale = (Whale) cow;
5  }
```

Классическое сужение типа с проверкой. Переменная cow типа Cow, хранит ссылку на объект класса Whale.
Мы проверяем, что это так и есть, и затем выполняем операцию преобразования (сужения) типа. Или как ее еще называют – downcast.

```
1  Cow cow = new Cow();
2  Whale whale = (Whale) cow; //exception
```

Ссылочное сужение типа можно провести и без проверки типа объекта.
При этом, если в переменной cow хранился объект не класса Whale, будет сгенерировано исключение – InvalidClassCastException.

6) А теперь еще на закуску. Вызов оригинального метода

Иногда тебе хочется не заменить унаследованный метод на свой при переопределении метода, а лишь немного дополнить его.

В этом случае очень хочется исполнить в новом методе свой код и вызвать этот же метод, но базового класса. И такая возможность в Java есть. Делается это так: `super.method()`.

Примеры:

Код

Описание

```
class Cow
{
public void printAll()
{
printColor();
printName();
}
public void printColor()
{
System.out.println("Я – белый");
}
public void printName()
{
System.out.println("Я – корова");
}
}

class Whale extends Cow
{
public void printName()
{
System.out.print("Это неправда: ");
super.printName();

System.out.println("Я – кит");
}
}
```

```
public static void main(String[] args)
{
Whale whale = new Whale();
whale.printAll();
}
```

На экран будет выведена надпись
Я – белый
Это неправда: Я – корова
Я – кит

— Гм. Ничего себе лекция. Мои робо-уши чуть не расплавились.

— Да, это не простой материал, он один из самых сложных. Профессор обещал подкинуть ссылок на материалы других авторов, чтобы ты, если все-таки что-то не поймешь, мог устранить этот пробел.

‹ (/quests/lectures/questcore.level02.lecture00)

×12 › (/quests/lectures/questcore.level02.lecture02)

Комментарии (53) популярные новые старые

Никита

Введите текст комментария

Assanali 19 уровень 14 марта, 20:13

Вернулся сюда из Java Core - Level 4 - Lecture3.
в комментарии была ссылка на эту лекцию.
всё стало Намного понятнее.

Ответить +2

Vra 40 уровень 28 февраля, 16:13

Вернулся сюда ибо сообщение написали в комментах.. а раз уж вернулся, то подниму свой коммент.

В лекции Ошибка, расширение перепутано с сужением, да и описание феерический бред.. сам в свое время запутался сильно.. читайте оф. докуументацию, там ошибок нет

вот пруфы (оф. документация Oracle):

5.1.5. Widening Reference Conversion, 5.1.6. Narrowing Reference Conversion

Ответить +7

mark 16 уровень 21 февраля, 10:09

Про сужение и расширение
Ссылка на объект - это что-то вроде пульта от ТВ (т.е. от объекта). Без пульта объект не имеет значения и удаляется. Если у нас иерархия Animal --> Pet --> Dog и мы создаем ссылку (пульт) типа Dog на объект Dog, то на этом пульте будут все кнопки (методы) этого класса и родительских классов: Pet и Animal. Т.е. у пульта типа Dog для объекта Dog - максимальное число кнопок, потому что они наследуются от всех классов-родителей, и объект Dog может реализовывать себя на все 100%.
Если мы сделаем пульт (ссылку) для управления Dog из класса-родителя, к примеру Pet, то объект Dog не сможет раскрыть весь свой потенциал, потому что не все кнопки (методы) будут на пульте (в ссылке).
Pet pet = new Dog();
Это сужение (мы убавили кнопок)

Если у нас пульт типа Pet от объекта Dog, и мы хотим присвоить значение этотого пульта к пульту типа Dog, то говорим компилятору чтоб он добавил кнопок, потому что мы хотим управлять объектом по полной, и в скобках указываем тип пульта который нам нужен :
Pet pet = new Dog();
Dog dog = (Dog) pet;
Это расширение (мы добавили кнопок)

Ответить +2

maxmarra 16 уровень 2 февраля, 14:36

Отличная лекция, спасибо!

Ответить 0

theBaldSoprano 16 уровень, Санкт-Петербург 29 января, 06:57

Судя по джавадокам здесь перепутаны сужение и раширение
см ссылку

Ответить +8

Сергей Марченко 13 уровень 4 февраля, 06:42

Так же показалось, раз пять перечитал противоречивый пример из лекции, таки намудрили.

Ответить +1

vinsler 27 уровень, Санкт-Петербург 16 декабря 2017, 14:57

########################## ВОТ ОНО ГЛАВНОЕ ВСЕЙ ЛЕКЦИИ !

Набор методов,
которые можно вызвать у переменной,
определяется типом переменной. // Cow - тип переменной? cow - переменная ?

// Cow cow = new Whale(); если брать это определение.

А какой именно метод/какая реализация вызовется,
определяется типом/классом объекта, // = new Whale ?

ссылку на который хранит переменная.

iow описание и возможные методы прописываются слева от знака равно,
а реализация справа?

##

Что будет если я вызову метод из типа переменной,
реализации которого нет в типе объекта?

Еще, super.printName(); это просто вызов метода из унаследованного класса, и дополнение его.
Но что делать, если мне нужно в этом методе вырезать 1% кода и 1% кода добавить?)
тогда только переопределять?

Ответить +1

Глеб Заславский 19 уровень 22 декабря 2017, 02:56

Cow cow = new Whale();

Если в классе Whale было что-то, чего нет в Cow, то вызвать это что-то из cow не получится, не смотря на то, что cow хранит в себе экземпляр объекта Whale. Оно и понятно - переменная cow может хранить в себе только то, что есть в классе Cow(), потому что почему? Потому что у этой переменной тип - Cow, и уникальные методы из Whale в эту коробочку (переменную) не влезут.

Если в классе Whale было переопределено что-то, что есть в классе Cow, можно вызвать это что-то, но оно будет вести себя так как написано в Whale.

По второму вопросу.
Да, метод super дёргает исполнение метода из родителя в потомке, ДАЖЕ ЕСЛИ в потомке этот метод переопределен.
Кусок метода переписать нельзя, ты либо переопределяешь весь метод, либо используешь метод родителя. Потому то и надо бить логику в классе на методы, и потому то методы желательно делать максимально простыми.

Ответить +7

vinsler 27 уровень, Санкт-Петербург 22 декабря 2017, 08:06

Грубо говоря, обьем памяти выделяется под Cow, поэтому Whale туда не запихать целиком, поэтому методы сравниваются и пихаются только если == в обоих классах. И метод можно или взять целиком или переписать.

Ответить 0

Вера Сургучёва 14 уровень 21 октября 2017, 16:01

какой раз замечаю, когда Амиго всё понятно, мне не очень, когда для него, что-то трудно для понимания, для меня всё ясно)

Ответить +30

vinsler 27 уровень, Санкт-Петербург 15 декабря 2017, 21:36

на то он и Амиго.)))

Ответить 0

Alex 29 уровень 22 февраля, 08:19

Амиго просто прикалывается

Ответить 0

Степан Карсаков 21 уровень, Минск 17 октября 2017, 18:42

Cow cow = new Whale(); //Классическое сужение типа.

/*Расширяющими преобразованиями являются преобразования от класса A к классу B, если A наследуется от B (важным частным случаем является преобразование от любого ссылочного типа к Object); */
/*Расширение означает переход от более конкретного типа к менее конкретному, т.е. переход от детей к родителям. */
Какое же тут сужение, если наш кит получил тип родителя?

Integer integer = new Integer(10);
Object obj = integer; // <-- расширяющее преобразование.
Integer in = (Integer)obj;// <-- сужающее преобразование.

http://javapapers.com/java/java-cast-and-conversions/

Снизу Витя это уже отметил, насколько я видел 16-го июня.

Ответить +7

Alex.Z 21 уровень, Москва 19 октября 2017, 16:16

Да, перепутали.

Ответить +1

Александр Востриков 16 уровень, Москва 8 января, 23:12

А разве у родителя ***Cow*** есть метод плавать?
Или примеру у класса потомка ***яблока*** есть все параметры и методы класса родителя **фрукты**, но у класса ***фрукты*** нет всех полей и методов класса ***яблока***, соответственно класс ***фрукты*** будет урезан по сравнению с классом ***яблоко***. Если не прав поправте.

Ответить 0

Степан Карсаков 21 уровень, Минск 9 января, 19:30

Урезан, это субъективно всё-таки. Урезан по количеству методов? Да, в теории их может быть меньше.
Вот тут хорошо человек объяснил. Из Javarushevcev)

Ответить +2

Fonzy 32 уровень, Москва 3 октября 2017, 11:56

Да уж, очень "сложные" задачи.

Ответить 0

Эльдар Ахметов 20 уровень, Уфа 22 сентября 2017, 08:44

Подробнее о полиморфизме http://java-course.ru/begin/polymorphism/

Ответить +3

Степан Карсаков 21 уровень, Минск 17 октября 2017, 17:43

Интересный сайт. Особенно понравился раздел "Отдел кадров", есть что почитать/сделать. Спасибо.

Ответить +1

Загрузить еще