



Java course

- [Начало Java](#)
- [Проект «Отдел кадров»](#)
- [Курсы](#)
- [Статьи](#)
- [Контакты/Вопросы](#)

- [Введение](#)
- [Установка JDK](#)
- [Основные шаги](#)
- [Данные](#)
- [Порядок операций](#)
- [IDE NetBeans](#)
- [ООП](#)
- [Инкапсуляция](#)
- [Наследование](#)
- [Пакеты](#)
- [Переопределение и перегрузка](#)
- [Полиморфизм](#)
- [Статические свойства и методы](#)
- [Отношения между классами](#)
- [Визуализация робота](#)
- [Пример — очередь объектов](#)
- [Массивы — знакомство](#)
- [Многомерные массивы](#)
- [Абстрактные классы](#)
- [Интерфейсы](#)
- [Расширенное описание классов](#)

В этом разделе мы поговорим об еще одном удобном понятии в ООП — абстрактных классах.

Зачем нужны абстрактные классы

Идея абстрактного класса заключается в следующем предположении — для работы иногда вам требуются не полностью готовые классы, а «заготовки» (полуфабрикаты, если хотите). Они уже кое-что умеют, но в «сыром виде» их использовать нельзя. Причем здесь стоит выделить два момента:

1. Создать экземпляр такого класса нельзя
2. Такой класс требует доработки под какие-либо конкретные условия.

Например в Java есть весьма наглядный класс **java.util.Calendar**. В его арсенале достаточно много полезных и нужных функций, но есть одна особенность — он не реализует какой-то конкретный календарь. Думаю, вы знакомы с тем фактом, что на Земле люди живут по разным календарям. Светские власти и католики живут по Григорианскому календарю. Русская православная церковь живет по Юлианскому календарю. А если ваша программа будет работает на марсоходе (что на самом деле так и есть — Java работает на марсоходе Spirit), то ей придется учитывать марсианский календарь. Как вы понимаете календари имеют различия, но вряд ли вы удивитесь, если вам скажут, что возможность прибавить 5 дней к какой-либо дате должна присутствовать во всех вариантах. Т.е. во всех этих календарях есть общий набор функций, который может иметь одинаковую реализацию. Отсюда рождается идея абстрактного класса, который с одной стороны не может создавать объекты, а с другой

- [Исключения](#)
- [Решения на основе классов](#)
- [Список контактов — начало](#)
- [Коллекции — базовые принципы](#)
- [Коллекции — продолжение](#)
- [Список контактов — GUI приложение](#)
- [Что такое JAR-файлы](#)
- [Многопоточность — первые шаги](#)
- [Многопоточность и синхронизация](#)
- [Работаем с XML](#)
- [Reflection — основы](#)
- [Установка СУБД PostgreSQL](#)
- [Базы данных на Java — первые шаги](#)
- [Возможности JDBC — второй этап](#)
- [JDBC — групповые операции](#)
- [Список контактов — работаем с БД](#)
- [Переезжаем на Maven](#)
- [Потоки ввода-вывода](#)
- [Сетевое взаимодействие](#)
- [С чего начинается Web](#)

стороны, может иметь уже готовые функции.

Еще одним примером абстрактного класса может служить уже знакомый нам класс **JComponent**. Этот класс умеет многое, он только не умеет рисовать что-либо. И если его этому научить — создать класс на его основе и переопределить метод **paintComponent**, то мы получим то, что нам надо.

Создание абстрактного класса на самом деле достаточно сложная архитектурная задача. Необходимость использовать именно абстрактный класс проявляется не сразу. Требуется провести анализ задачи и набора классов, который позволит принять решение.

А вот техническая сторона достаточно простая — для объявления абстрактного класса достаточно добавить ключевое слово **abstract** в описании класса.

```
1 abstract public class AbstractModel {
2     ...
3 }
```

Если вы попытаетесь создать объект этого класса, то компилятор выдаст сообщение об ошибке.

Кроме того, что мы можем заставить разработчика НЕ пользоваться нашим классом напрямую, мы можем еще более жестко подойти к наследованию — установить правила, которые заставят класс-наследник реализовать определенные методы.

Для этого необходимо не только класс описать как абстрактный, но и метод, который должен обязательно реализовать наследник. Форма записи достаточно несложная. Здесь только надо отметить, что тело метода отсутствует совсем — сразу за описанием метода ставится точка с запятой.

Например:

```
1 public abstract class AbstractModel
2 {
3     public abstract void processModel();
4 }
```

В нашем примере не только класс объявлен как абстрактный, что запрещает создание экземпляра такого класса, но и требуется переопределить метод **processModel**.

Внимательный читатель возможно отметил факт, что я в одном случае слово **abstract** поставил в самом начале, а в втором примере после слова **public**. Это сделано специально — я хотел продемонстрировать, что можно делать объявления и так и так.

И наконец мы сделаем более сложный пример, который продемонстрирует использование абстрактного класса. В части [Полиморфизм](#) мы создали приложение, которое рисовало на форме три вида фигур: треугольник, прямоугольник, овал. В этом приложении мы использовали абстрактный класс **JComponent**, который не имеет обязательности для переопределения метода для рисования **paintComponent**. Создадим абстрактный класс, который наследуется от **JComponent** и имеет абстрактный метод, который надо переопределить в классах-наследниках. Вот такой класс:

```
1 package edu.javacourse.ui.component;
2
3 import java.awt.Graphics;
4 import javax.swing.JComponent;
5
6 public abstract class AbstractShape extends JComponent
7 {
8     @Override
9     protected void paintComponent(Graphics g) {
10         super.paintComponent(g);
11         paintShape(g);
12     }
13
14     abstract protected void paintShape(Graphics g);
15 }
```

Как видим теперь наш класс не просто абстрактный и значит не может быть создан объект такого класса, но также необходимо переопределить метод **paintShape**. Напишем реализацию наших классов для треугольника, овала и прямоугольника:

Треугольник

```
1 package edu.javacourse.ui.component;
2
3 import java.awt.Graphics;
4
5 // Класс для рисования треугольника
6 public class TriangleComponent extends AbstractShape
7 {
8     @Override
9     protected void paintShape(Graphics g) {
10         g.drawLine(5, getHeight() - 10, getWidth() / 2 - 5, 5);
```

```

10         g.drawLine(0, getHeight() / 2 - 10, getWidth() / 2 + 5, 0);
11         g.drawLine(getWidth() / 2 - 5, 5, getWidth() - 10, getHeight() - 10);
12         g.drawLine(getWidth() - 10, getHeight() - 10, 5, getHeight() - 10);
13     }
14 }

```

Прямоугольник

```

1 package edu.javacourse.ui.component;
2
3 import java.awt.Graphics;
4
5 // Класс для рисования прямоугольника
6 public class RectangleComponent extends AbstractShape
7 {
8     @Override
9     protected void paintShape(Graphics g) {
10         g.drawRect(5, 5, getWidth() - 10, getHeight() - 10);
11     }
12 }

```

Овал

```

1 package edu.javacourse.ui.component;
2
3 import java.awt.Graphics;
4
5 // Класс для рисования овала
6 public class OvalComponent extends AbstractShape
7 {
8     @Override
9     protected void paintShape(Graphics g) {
10         g.drawOval(5, 5, getWidth() - 10, getHeight() - 10);
11     }
12 }

```

Класс для формы

```

1 package edu.javacourse.ui;

```

```

2
3 import edu.javacourse.ui.component.OvalComponent;
4 import edu.javacourse.ui.component.RectangleComponent;
5 import edu.javacourse.ui.component.TriangleComponent;
6 import java.awt.GridLayout;
7 import javax.swing.JFrame;
8
9 public class ShapeFrame extends JFrame
10 {
11     public ShapeFrame() {
12         // Устанавливаем LayoutManager в виде таблицы
13         // размерами 2 строки на 3 столбца
14         setLayout(new GridLayout(2, 3));
15
16         // Создаем и "укладываем" на форму компоненты разных классов
17         add(new OvalComponent());
18         add(new RectangleComponent());
19         add(new TriangleComponent());
20         add(new OvalComponent());
21         add(new RectangleComponent());
22         add(new TriangleComponent());
23
24         // Устанавливаем координаты и размеры окна
25         setBounds(200, 200, 450, 350);
26     }
27 }

```

В конце класс для запуска нашего приложения

```

1 package edu.javacourse.ui;
2
3 import javax.swing.JFrame;
4
5 public class ShapeApplication
6 {
7     public static void main(String[] args) {
8         // Создаем графическое окно
9         ShapeFrame of = new ShapeFrame();
10        // Задаем правило, по которому приложение завершиться при
11        // закрытии этой формы
12        of.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13        // Делаем окно видимым
14        of.setVisible(true);
15    }
16 }

```

Скачать наше приложение можно по ссылке: [ShapeApplication2](#).

Для первого знакомства с абстрактными классами информации будет достаточно. Позволю еще раз отметить: абстрактные классы являются дополнительным удобством при проектировании иерархии классов и позволяют более строго подходить к этому вопросу. Возможно, что на данном этапе изучения вы не полностью осознаете их полезность, но при накоплении опыта вы наверняка поймете и оцените удобство их использования.

И теперь нас ждет следующая статья: [Интерфейсы](#).

33 comments to *Абстрактные классы*



Апрель 3, 2015 at 19:43

javaNoob says:

Лично я, пока узрел в описанном примере, что класс АбстрактнойФигуры стал выполнять роль прослойки-посредника между JComponent и нашими фигурами(навязывая им(фигурам) свою схему прорисовки). Т.е. теперь у фигур появилась некая общая но абстрактная фигура. Архитектура выходит более логичная.

ЗЫ. Пока я это писал, меня озарило — определяя в абстрактном классе какой то функционал мы даём себе гарантию, что любой наследник нашего абстрактного класса будет вынужден следовать его правилам, чего не получилось бы если мы захотели впендюрить вместо абстракта обычный класс. Или не так ? Впрочем я уже сам не понимаю что написал :).

Ещё вопрос — сильно ли усложняется объектно-ориентированное проектирование в Java, если нет чёткого понимания, когда применяются абстрактные классы?

[Reply](#)



Апрель 6, 2015 at 06:18

admin says:

По поводу абстрактного класса с обязательной функциональностью — это совершенно правильная мысль. Если совместить это с объявлением final, то тогда очень красиво получается.

Что касается вопроса — я исхожу из требований задачи. Специально создавать абстрактный класс наверно не есть хорошо. Но если он ложится в логическое описание задачи — почему бы не использовать.

[Reply.](#)



Июль 18, 2015 at 08:50

Serg says:

А скажите, для чего тут нужен `protected void paintComponent(Graphics g)` и почему без него фигуры не рисуются?

[Reply.](#)



Июль 20, 2015 at 06:21

admin says:

Этот метод есть у класса `JComponent` и он вызывается из `paint`. Разработчики Swing решили, что надо сделать именно так — переопределять `paintComponent`. Вот мы так и сделали.

[Reply.](#)



Октябрь 27, 2015 at 08:59

[beginer](#) says:

и все таки я не понял зачем нужны абстрактные классы
и зачем их использовать если есть интерфейсы работающие по той же схеме

[Reply.](#)



Октябрь 27, 2015 at 10:50

admin says:

Интерфейсы — это просто объявление, что класс должен реализовать. Но класс в этом случае должен реализовать ВСЕ методы интерфейса. Абстрактный класс уже предоставляет некоторую функциональность. Например — для табличной модели существует интерфейс `TableModel` — <http://docs.oracle.com/javase/7/docs/api/javax/swing/table/TableModel.html>
Реализовать полностью всю функциональность — это большая задача. Для упрощения заранее сделан класс `AbstractTableModel` — <http://docs.oracle.com/javase/7/docs/api/javax/swing/table/TableModel.html> который позволяет гораздо проще реализовать функциональность — там уже что-то работает.

[Reply.](#)



• Октябрь 27, 2015 at 09:05

[beginer](#) says:

кстати

под созданием экземпляров я понимаю что-то вроде

```
someClass object = new someClass();
```

вначале сказано что экземпляров создавать запрещено

но у меня получилось их создать вышеуказанным способом
какого такого ?!

разве не должны появиться исключения и т.д.?

кстати

недавно начал изучать джаву так что простите за тупые вопросы

[Reply.](#)



○

Октябрь 27, 2015 at 10:54

admin says:

Надо код смотреть — не может компилятор для абстрактного класса `TestClass` разрешить такой оператор

```
TestClass tc = new TestClass();
```

[Reply.](#)



Октябрь 28, 2015 at 08:52

[beginer](#) says:

```
public static void main(String[] args) throws IOException {  
    AbCL object1 = new AbCL() {
```

```
    };  
    object1.a=1;  
    object1.b=1;  
    object1.c=1;  
    object1.s=»a»;  
    object1.d=»b»;  
    object1.f=»c»;  
    }  
}
```

```
abstract class AbCL{  
    int a,b,c;  
    String s,d,f;  
}
```

является ли это созданием экземпляра ??

[Reply](#)



Октябрь 28, 2015 at 11:58

admin says:

Вы здесь схитрили — Вы же создали новый анонимный класс, унаследованный от абстрактного. Так что это уже не абстрактный класс, а его потомок. И он не абстрактный. Так что здесь все честно. Попробуйте сделать в классе AbCL абстрактный метод и тогда в анонимном классе Вам придется его переопределить.

[Reply](#)



Октябрь 28, 2015 at 14:42

[beginer](#) says:

пришлось реализовать абстрактный метод
а в остальном все так же
это считается за экземпляр или нет ?)
я уже капец как запутался

[Reply](#)



Октябрь 28, 2015 at 15:08

admin says:

Вы создали экземпляр класса. Но Вы использовали для создания объекта НЕ класс AbCL — Вы использовали его НАСЛЕДНИКА. То, что класс анонимный, ничего не меняет в плане наследования. Все равно это полноценный НАСЛЕДНИК абстрактного класса AbCL. И этот наследник уже НЕ абстрактный класс. Значит можно создать объект.

[Reply](#)



Октябрь 28, 2015 at 16:12

[beginer](#) says:

что за анонимный класс ?
откуда ?
а что если :

```
package nextstep_01;
```

```
abstract public class NewClass {  
    int a,b,c;  
    void cout(){
```

```
System.out.println(«l9l9l9»);  
}  
}
```

```
package nextstep_01;
```

```
public class NextStep_01 {
```

```
public static void main(String[] args){  
NewClass ob = new NewClass() {};  
ob.a=1;  
ob.b=2;  
ob.c=3;  
ob.cout();  
}  
}
```

[Reply.](#)



■

Октябрь 28, 2015 at 17:36

[beginer](#) says:

кстати

та же самая чертовщина с интерфейсом
что значит анонимные классы ???

[Reply.](#)



■

Октябрь 29, 2015 at 10:20

admin says:

Читайте здесь — http://java-course.ru/begin/class_description_ext/



Октябрь 29, 2015 at 10:18

admin says:

Почитайте тут про анонимные классы — http://java-course.ru/begin/class_description_ext/ У вас такой и получился.

И в этом примере тоже самое — вот строка

```
NewClass ob = new NewClass() {};
```

Вы же не только вызвали конструктор — Вы же еще зачем-то использовали ФИГУРНЫЕ СКОБКИ сразу после конструктора.

Вы создали анонимный класс — посмотрите в директорию, где у вас скомпилированные классы лежат — там их будет больше, чем файлов .java

Было настолько курьезно, что я внес дополнения в статью именно по этому случаю.

[Reply](#)



Октябрь 29, 2015 at 14:30

[beginner](#) says:

хехехе

без обид)

большое спасибо за терпение !

[Reply](#)



Июнь 8, 2016 at 14:16

Nibbler says:

Правильно ли я понимаю, что в абстрактном классе JComponent нет нереализованных методов, которые в обязательном порядке должны быть переопределены в классах-наследниках? Насколько я смог просмотреть исходники, даже метод `paintComponent()`, который мы переопределяем в нашем случае, в родительском классе определен:

```

1      protected void paintComponent(Graphics g) {
2          if (ui != null) {
3              Graphics scratchGraphics = (g == null) ? null : g.create();
4              try {
5                  ui.update(scratchGraphics, this);
6              }
7              finally {
8                  scratchGraphics.dispose();
9              }
10         }
11     }

```

Т.е., в противном случае нам нужно было бы реализовать все-все заявленные но не реализованные методы класса JComponent в нашем классе-наследнике?

[Reply](#)



o

Июнь 8, 2016 at 16:53
admin says:

Именно так — если есть абстрактные методы, то они должны либо быть реализованы в потомке, либо потомок сам должен быть абстрактным. JComponent несколько запутывает — в нем действительно нет абстрактных методов, но сам он абстрактный. Почему так решили сделать — могу только догадываться. Например, чтобы быть уверенным, что работаем именно с его наследником. Или просто, чтобы думали.

[Reply](#)



•

Август 10, 2016 at 17:22
Сергей says:

Как быть с такой ситуацией, если нельзя создать объект абстрактного класса, то почему здесь в примере все прекрасно создается?

Socket socket = new Socket(ipAddressServer, Client.PORT); //создает сокет используя IP-адрес сервера и порт
 //берем входной и выходной потоки сокета, теперь можем получать и отсылать данные клиентом

```
InputStream sin = socket.getInputStream();  
OutputStream sout = socket.getOutputStream();
```

[Reply.](#)



o

Август 10, 2016 at 19:09

admin says:

Где именно создается объект абстрактного класса ?

[Reply.](#)



•

Август 11, 2016 at 01:11

Сергей says:

Конструкция `InputStream sin = socket.getInputStream();`
`InputStream` абстрактный класс. Метод `getInputStream()` класса `Socket` по документации имеет сигнатуру
`public InputStream getInputStream() throws IOException`, то есть, при вызове метод возвращает ссылку на объект типа `InputStream`. И если `InputStream` абстрактный класс, то как может существовать ссылка на его объект, я чего-то не догоняю.

[Reply.](#)



o

Август 11, 2016 at 10:04

admin says:

То, что метод заявлен, как возвращающий `InputStream` совсем не значит, что он действительно возвращает именно этот класс. Он возвращает НАСЛЕДНИКА. Еще раз почитайте о полиморфизме.

[Reply.](#)



Август 11, 2016 at 06:27

Сергей says:

Метод `getInputStream` класса `Socket` возвращает ссылку на объект абстрактного класса `InputStream`:

```
InputStream sin = socket.getInputStream();
```

[Reply](#)



Ноябрь 16, 2016 at 12:42

v says:

как я понял классы `JComponent` и наш `AbstractShape` абстрактного типа, но в нашем классе остались ТОЛЬКО 2 метода, которые предлагает IDE: `paintComponent` (который был) и `paintShape` (который мы создали/переопределили), т.е. мы выкинули ненужные нам методы, но функционал оставшихся методов не изменялся, правильно я понимаю?

[Reply](#)



o

Ноябрь 16, 2016 at 12:52

admin says:

В нашем классе остались ВСЕ методы, которые есть у `JComponent`. Мы ПЕРЕОПРЕДЕЛИЛИ некоторые из них. Но все остальные не исчезли. И мы можем их использовать в нашем классе в том виде, в котором они есть у предка — `JComponent`.

[Reply](#)



Ноябрь 16, 2016 at 16:20

v says:

глупый наверно вопрос: т.е. мы, получается, можем использовать другие методы JComponent в нашем AbstractShape (на текущий момент), которые IDE не предлагает и это будет работать? мое понимание такое в данном примере: мы даем установку (переопределение или разрешение) абстрактному классу AbstractShape использовать метод paintComponent (как есть в полном объеме), который называем paintShape, соответственно, другие методы, если они не переопределены, работать не будут. или IDE обманывает?

[Reply.](#)



o

Ноябрь 16, 2016 at 16:59

admin says:

Можем. Только либо protected (если мы наследники) либо public. По идее там такие есть и IDE должна их показывать.

[Reply.](#)



•

Май 31, 2017 at 11:57

Sid says:

А зачем paintComponent вызывает метод paintShape и почему без него не идет прорисовка объясните пожалуйста

[Reply.](#)



o

Май 31, 2017 at 11:58

Sid says:

Я ведь так понял что paintShape создан именно для того, что бы пошел вызов AbstractShape?

[Reply.](#)



■

Май 31, 2017 at 12:56

admin says:

Вызов paintShape сделан в AbstractSahre для того, чтобы все наследники могли только переопределить paintShape и больше ничего не делать.

[Reply](#)



•

Август 16, 2017 at 15:17

Роман says:

Добрый день!

Уважаемый администратор,

не могу въехать, каким образом вызывается метод paintShape, если мы только создаем объект класса с этим методом(new Component()). То же самое дальше, где происходит вызов метода paintComponent, если мы только наследуем класс который содержит этот метод?

Спасибо!

[Reply](#)



o

Август 17, 2017 at 15:30

admin says:

По порядку — JFrame в момент рисования вызывает у каждого компонента метод paint. Он в свою очередь вызывает paintComponent. Метод paintShape мы определили в классе-предке как абстрактный и (ЧТО ВАЖНО) мы его вызываем в paintComponent. Теперь для каждого класса осталось только определить paintShape. Получаем цепочку:

JFrame вызывает paint — paint вызывает paintComponent — paintComponent вызывает paintShape. У каждого компонента он свой.

[Reply](#)

Leave a reply


Comment

You may use these HTML tags and attributes: <abbr title=""> <acronym title=""> <blockquote cite=""> <cite> <code class="" title="" data-url=""> <del datetime=""> <i> <q cite=""> <s> <strike> <pre class="" title="" data-url="">

Имя *

E-mail *

Сайт

9 + шесть = 

Add comment

Copyright © 2018 [Java Course](#)

Designed by [Blog templates](#), thanks to: [Free WordPress themes for photographers](#), [LizardThemes.com](#) and [Free WordPress real estate themes](#)

