



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](#)

Полиморфизм

Полиморфизм на первый взгляд кажется самой малоинтересной и малоперспективной парадигмой, но на самом деле это совсем не так. Полиморфизм удивительно мощная и востребованная парадигма. Давайте попробуем разобраться, что это такое.

Полиморфизмом называется возможность работать с несколькими типами так, как будто это один и тот же тип и в то же время поведение каждого типа будет уникальным в зависимости от его реализации.

Возможно, что вы ничего не поняли, поэтому попробую описать это иначе и на примере. Давайте так и сделаем.

В предыдущей части мы создавали класс RobotTotal, который наследовался от класса Robot. Если немного подумать, то по парадигме наследования будет интуитивно понятно, что класс RobotTotal является пусть и несколько измененным, но тем не менее классом Robot. Исходя из этого вполне непротиворечивого соображения мы можем написать несколько иную реализацию класса RobotManager

- [Расширенное описание классов](#)
- [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](#)

```
1 package edu.javacourse.robot;
2
3 public class RobotManager
4 {
5     public static void main(String[] args) {
6         // Первое проявление полиморфизма - ссылке на класс-предок
7         // можно присвоить класс-потомок
8         Robot robot = new RobotTotal(0, 0);
9
10        robot.forward(20);
11        robot.setCourse(90);
12        robot.forward(20);
13        robot.setCourse(90);
14        robot.forward(50);
15        // Напечатать координаты
16        robot.printCoordinates();
17        // Напечатать общую дистанцию уже не получится
18        // компилятор выдает ошибку
19        //System.out.println(robot.getTotalDistance());
```

```
20 |     }
21 | }
```

Как видим, создавая объект RobotTotal мы его «сужаем» до объекта класса Robot. С одной стороны это выглядит достаточно логично и непротиворечиво — RobotTotal является роботом. С другой стороны возникает вопрос — а метод forward будет вызываться какой? Класса RobotTotal или Robot? Думаю, что для вас ответ «Как RobotTotal» выглядит предпочтительнее — и это совершенно правильный ответ. Можете в этом убедиться сами, добавив в метод forward какую-либо поясняющую печать. Например так:

```
1 | @Override
2 | public void forward(int distance) {
3 |     super.forward(distance);
4 |     totalDistance += distance;
5 |     System.out.println("RobotTotal");
6 | }
```

В этом можно убедиться еще более удобным и практичным образом — спросить у объекта его класс и у класса спросить его имя. Этот механизм называется Reflection — он позволяет получить информацию об объекте прямо в момент выполнения программы. Мы будем его рассматривать несколько позже. Вот как может выглядеть такой вызов:

```
System.out.println(robot.getClass().getName());
```

Расшифровывается это приблизительно так: сначала у объекта robot получаем его класс методом getClass(). Возвращается объект типа Class (есть такой стандартный класс — Class). И у этого класса есть метод, который возвращает имя — getName(). Вот возможный полный код:

```
1 | package edu.javacourse.robot;
2 |
3 | public class RobotManager
4 | {
5 |     public static void main(String[] args) {
6 |         Robot robot = new RobotTotal(0, 0);
7 |         System.out.println(robot.getClass().getName());
8 |     }
9 | }
```

Как видим мы можем присвоить ссылке на объект класса-предка объект-потомок — и это работает. С одной стороны, мы работаем как-будто с классом Robot, с другой стороны — поведение нашего объекта соответствует классу RobotTotal. **ВАЖНО !!!** А вот в обратную сторону присваивание НЕ работает. На запись вот такого вида

```
RobotTotal robot = new Robot(0, 0);
```

компилятор будет выдавать ошибку.

Думаю, что с технической точки зрения все достаточно понятно. Но возникает логичный вопрос — зачем это вообще надо ? С инкапсуляцией более-менее понятно, с наследованием — в принципе тоже. Но вот этот механизм зачем, какое преимущество мы получим при использовании этой парадигмы ? Для первого приближения рассмотрим наш пример графического приложения, в котором мы создавали свой компонент `OvalComponent`. Мы использовали модифицированный класс `JFrame` (`OvalFrame`) и что весьма важно, мы использовали уже готовый метод `add` для добавления нашего объекта на форму. Я бы хотел заострить ваше внимание на этом весьма тонком моменте — мы использовали УЖЕ существующий метод существующего класса `JFrame`. И этот метод (да и класс тоже) совершенно не осведомлен о нашем новом классе `OvalComponent`. И тем не менее он прекрасно с ним работает — мы сами это видели. Думаю, что вы уже догадались в чем фокус, но я тем не менее проговорю эту мысль — класс `JFrame` умеет работать с классами-потомками от класса `JComponent` и ему не важно, какой точно класс он получил — они для него все являются объектами класса `JComponent`. И это здорово нам помогает. Т.к. наш класс `OvalComponent` рисует себя сам путем вызова метода `paintComponent` где-то у себя, то ситуация еще более восхитительна — вызывается именно НАШ метод `paintComponent`. Значит мы можем написать много разных классов унаследованных от класса `JComponent`, положить их на форму и все они будут рисоваться так, как они сами это умеют. Что еще интересно — для самого себя класс тоже может вести себя полиморфно. Если внимательно посмотреть на код класса `OvalComponent` метод `paintComponent` объявлен как `protected` и вызывается внутри класса `JComponent` и никак иначе. Снаружи другим классам он недоступен. Т.е. все наследники класса `JComponent` предоставляют свои реализации метода `paintComponent` и вызывают его внутри унаследованного метода `paint`, который уже объявлен как `public`. Иными словами — все наследники используют метод `paintComponent` из уже готового метода `paint`. Возможно, вы еще не совсем готовы оценить «красоту игры», но на мой взгляд полиморфизм является просто чудесной штукой. Мы еще вернемся к этой весьма увлекательной парадигме, ну а пока сделаем еще одно графическое приложение, которое позволит нам поместить разные типы компонентов на форму, о которых она не знает, но тем не менее сможет прекрасно ими управлять.

Графическое приложение

Данное приложение несколько сложнее предыдущего — здесь создаются разные компоненты для рисования разных фигур. Приложение содержит 5 классов: 3 класса являются компонентами, которые рисуют внутри себя три разных фигуры (овал, треугольник и прямоугольник), класс для отображения окна и класс для создания и отображения самой формы. Сначала рассмотрим три класса для рисования фигур. Они используют методы класса `Graphics` и вряд ли требуют каких-либо комментариев. Я их поместил в отдельный пакет.

```
1 package edu.javacourse.ui.component;
2
3 import java.awt.Graphics;
4 import javax.swing.JComponent;
5
6 public class OvalComponent extends JComponent
7 {
8     @Override
9     protected void paintComponent(Graphics g) {
10         super.paintComponent(g);
```

```
11     g.drawOval(5, 5, getWidth() - 10, getHeight() - 10);
12 }
13 }
```

```
1 package edu.javacourse.ui.component;
2
3 import java.awt.Graphics;
4 import javax.swing.JComponent;
5
6 public class RectangleComponent extends JComponent
7 {
8     @Override
9     protected void paintComponent(Graphics g) {
10         super.paintComponent(g);
11         g.drawRect(5, 5, getWidth() - 10, getHeight() - 10);
12     }
13 }
```

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

В классе формы мы используем механизм для установки `LayoutManager`. Если в двух словах, то идея заключается в следующем — форме можно указать алгоритм (правила) размещения компонентов. Реализации этого алгоритма выносятся в отдельный класс (который имеет обобщенное название `LayoutManager`) и задается форме (более правильно будет сказать, что задается для контейнера). При рисовании форма использует этот класс для определения, как размещать компоненты. Подробнее можно почитать в статье [Что такое LayoutManager](#).

```

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 }

```

Еще раз обратите внимание, что когда мы вызываем метод **add** мы передаем форме разные объекты. Но что здесь важно — они все наследники **JComponent**. Форме в принципе надо получить еще более «старого» предка — [java.awt.Component](#). Можно конечно углубиться в исходники, но давайте пока немного упростим и поймем главное — форма «думает», что она «работает» в классом JComponent. Именно у этого класса она вызывает метод прорисовки `paintComponent` (повторюсь — это не совсем так, но сейчас мы упрощаем для понимания главной идеи полиморфизма). Мы в какой-то мере «обманываем» форму — подсовываем ей компонент, который за счет наследования имеет модифицированный метод прорисовки и когда форма «полагает», что она вызывает метод `paintComponent` у объекта класса JComponent на самом деле она вызывает метод `paintComponent` у объекта уже нашего нового класса `OvalComponent` или `RectangleComponent`.

Полиморфизм таким образом позволяет вам подменять объекты для тех кто их вызывает и вызывающий даже не знает об этом. Оглушительная возможность.

Заключительный класс для запуска формы мы уже видели. Наверно уже есть смысл отметить, что с точки зрения правильного создания и запуска формы наше приложение не совсем корректно. Но для упрощения мы пока идем на такой шаг — наша форма достаточно простая и может быть создана так, как показано.

```
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 }
```

Больше каких-либо комментариев давать не буду — просто почитайте код, запустите приложение, попробуйте его модифицировать. Можете добавить свои компоненты или поменять порядок отображения компонентов на форме. Полный код приложения вы можете посмотреть [здесь](#) — [ShapeApplication](#). Помните, что именно благодаря полиморфизму мы можем создавать приложения, которые смогут работать с классами, которых возможно еще даже нет.

Мы рассмотрели основные парадигмы ООП и теперь перед нами задача более подробно познакомиться с конструкциями и идеями языка Java. Чем мы с вами и займемся.

И теперь нас ждет следующая статья: [Статические свойства и методы](#)

22 comments to *Полиморфизм*



• Май 13, 2015 at 10:10

newbie says:

подскажите, какой проект и пакет нужно создать в NetBeans IDE, чтобы поэкспериментировать с class GridBagLayoutTest ?

[Reply](#)



o

Май 13, 2015 at 12:50

admin says:

Проект самый простой — Java Application. Посмотреть как поиграть можно здесь — [Что такое LayoutManager](#)

[Reply.](#)



•

Ноябрь 17, 2015 at 02:37

QA says:

> на такой шаг – наца форма достаточно

на такой шаг – наЩа форма достаточно — *щ* заменить на *ш*

[Reply.](#)



o

Ноябрь 18, 2015 at 10:04

admin says:

Спасибо, исправил.

[Reply.](#)



•

Май 24, 2016 at 17:53

[Nibbler](#) says:

В предыдущем примере рисования овала, следуя логике полиморфизма и для собственного понимания, при создании объекта-овала использовал указатель на родительский класс: вместо `OvalComponent oc = new OvalComponent();` написал `JComponent oc = new OvalComponent();` NetBeans добавил в раздел импорта `javax.swing.JFrame` и все заработало! Получается я сослался на метод родительского класса применительно

к объекту дочернего класса, где он был переопределен. Таким образом вызвался переопределенный метод, соответствующий конкретной реализации объекта дочернего класса. Я правильно понял?

[Reply.](#)



o

Май 25, 2016 at 14:10

admin says:

Нет. Вы еще не до конца поняли полиморфизм. Ссылка родительского класса прекрасно ссылается на объект класса-потомка — это как раз особенность полиморфизма.

Но он все равно остается потомком, а не родительских классом. Разберитесь со ссылками — на что они указывают.

[Reply.](#)



•

Сентябрь 19, 2016 at 19:58

Денис says:

Здравствуйте. Подскажите пожалуйста, можно ли рассматривать отношения наследования между суперклассом JComponent и дочерними классами фигур как композицию. Исходя из понятия «время жизни» да и определения композиции тоже, в данном фрагменте примера можно сделать такой вывод. Мои рассуждения верны или я путаю «Божий дар с яйцом». Заранее благодарен.

ЗЫ. поправьте слово в классе ShapeApplication на 10 строке, у вас там опечатка.

[Reply.](#)



o

Октябрь 3, 2016 at 18:13

admin says:

Нет, композиция это скорее форма и все элементы, которые на ней находятся. Это составление нескольких элементов в одно сложное существо 😊

Наследование — это отдельный вариант отношений между классами.

[Reply](#)



Декабрь 9, 2016 at 11:43

Алла says:

Спасибо за курс!

вопрос: хочу нарисовать закрашенный треугольник.
с овалом и квадратом просто — есть fillOval и fillRect
а с треугольником как?

[Reply](#)



Декабрь 9, 2016 at 13:50

admin says:

Посмотрите метод fillPolygon у объекта Graphics

[Reply](#)



Февраль 24, 2017 at 17:08

Юрий says:

я пишу в eclipse и для ShapeFrame у меня получился вот такой код :

```
import java.awt.GridLayout;
import javax.swing.JFrame;
public class ShapeFrame extends JFrame {
    public ShapeFrame() {
        setLayout(new GridLayout(2, 3));
        add(new OvalComponent());
        add(new RectangleComponent());
        add(new TriangleComponent());
    }
}
```

```
add(new OvalComponent());
add(new RectangleComponent());
add(new TriangleComponent());
setBounds(200, 200, 450, 350);
}
}
```

Отличается от вашего тем, что нет вот этих строчек:

```
import edu.javacourse.ui.component.OvalComponent;
import edu.javacourse.ui.component.RectangleComponent;
import edu.javacourse.ui.component.TriangleComponent;
```

все перерыл, так и не понял, как мне импортировать мои собственные классы из этого же пакета в eclipse. В итоге забил, запустил как и есть и все работает и без этих трех импортов. Это какая-то особенность eclipse?

[Reply](#)



Февраль 24, 2017 at 18:20

Юрий says:

Извиняюсь за флуд, допер, как импортировать классы в eclipse.

Но все равно вопрос остался, для чего мы импортируем три наших же класса в класс ShapeFrame, если и без этих трех импортов все работает?

```
package framuga;
import java.awt.GridLayout;
import javax.swing.JFrame;
import framuga.OvalComponent;
import framuga.RectangleComponent;
import framuga.TriangleComponent;
```

```
public class ShapeFrame extends JFrame {
    public ShapeFrame() {
        setLayout(new GridLayout(2, 3));
```

```
// Создаем и «укладываем» на форму компоненты разных классов
add(new OvalComponent());
add(new RectangleComponent());
add(new TriangleComponent());
```

```
add(new OvalComponent());
add(new RectangleComponent());
add(new TriangleComponent());

// Устанавливаем координаты и размеры окна
setBounds(200, 200, 450, 350);

}

}
```

[Reply](#)



o

Февраль 25, 2017 at 13:33
admin says:

Судя по всему у Вас ВСЕ классы находятся в ОДНОМ пакете. У меня в примере не так — посмотрите внимательно.

Основной класс в пакете `edu.javacourse.ui`

Компоненты в пакете `edu.javacourse.ui.component`

Само собой в основной класс их надо импортировать. В Вашем случае конечно же бессмысленно импортировать классы из того же пакета.

[Reply](#)



•

Апрель 11, 2017 at 15:43
Vlad says:

Слишком запутанное пояснение полиморфизма. Полный тупик понимания. Суть полиморфизма заключается в том, что вы можете использовать наследников, как родителей. При этом, если в классе-наследнике был переопределен какой — то метод, то вызовется он. Разве этого не достаточно?

[Reply](#)



o

Апрель 19, 2017 at 10:30

admin says:

Можно и так сказать. Наверно. В том-то и проблема книги/статьи/видео — она рассказывает только одним способом. А разным людям надо по-разному объяснять.

[Reply.](#)



o

Сентябрь 8, 2018 at 17:25

andy says:

«Слишком запутанное пояснение полиморфизма»

+1

[Reply.](#)



■

Сентябрь 9, 2018 at 23:56

admin says:

Полиморфизм вообще самая сложная и самый интересная парадигма. Его очень сложно рассказывать на начальном этапе. Видимо я написал так, что не всем понятно. Такое бывает — разные люди по-разному воспринимают. Это недостаток любой книги — нельзя прочитать иными словами и нельзя задать вопрос.

[Reply.](#)



•

Апрель 11, 2017 at 22:30

Алексей says:

Здравствуйте, решил поиграться с приложением. Добавил класс SquareComponent(хотя конечно название это всего лишь для понимания что там будет происходить), и вместо шестой фигуры которая у Вас треугольник, Я добавил нарисованную фигуру. код в классе в общем такой

```
package edu.javacourse.ui.component;
import java.awt.Graphics;
import javax.swing.JComponent;
import java.util.Scanner;
public class SquareComponent extends JComponent{
    Scanner ent = new Scanner(System.in);
    private int num = 0;
    private int[] h = new int[3];//или тут могло быть/new int[]{2, 3, 5} для проверки
    private int[] l = new int[3];//что массив изменится.
    public void ConstrLine(){
        for(int i = 0; i < 3; i++){
            System.out.print("x["+i+"]= ");
            num = ent.nextInt();
            h[i] = num;
            System.out.print("y["+i+"]= ");
            num = ent.nextInt();
            l[i] = num;
        }
    }
    public void getarr(){
        for(int i = 0; i < 3; i++){
            System.out.println("x="+h[i]+"y="+l[i]);
        }
    }
    @Override
    public void paintComponent(Graphics g){
        super.paintComponent(g);
        g.drawPolygon(h, l, 3);
    }
}
```

конечно же дравПолигон мой не квадрат рисует, а так как три координаты трех линий, то рисует треугольник. и как видно координаты задаю в ручную, но Он рисует по дефолдным координатам(нулевыми), хотя конечно же массив изменился и даже сделал проверку для того что бы убедиться что массив изменен. Естественно что ввод в массив делал до создания объекта:

```
public static void main(String[] args){
    SquareComponent sq = new SquareComponent();
    sq.ConstrLine();
    ShapeFrame of = new ShapeFrame();
}
```

of.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
но все равно рисует массив с дефолдными координатами.

[Reply](#)



Сентябрь 1, 2018 at 17:42

Денис says:

Вроде понятно все, только вот чем отличается

```
Robot robot = new RobotTotal(0, 0);
```

от

```
RobotTotal robot = new RobotTotal(0, 0);
```

[Reply](#)



Сентябрь 1, 2018 at 20:39

admin says:

Еще раз подумайте — есть собака и есть пинчер. Но пинчер тоже СОБАКА. Вы можете смотреть на пинчера как на собаку и как на пинчера со своими особенностями.

Когда вы смотрите на пинчера как на собаку, то вы можете оперировать только теми характеристиками, которые присущи собаке. Во втором случае количество характеристик увеличивается — это уже характеристики не просто собаки, а пинчера.

Причем пинчер сам по себе не меняется — он остается пинчером. Меняется ваш взгляд на него. Подумайте над этим.

[Reply](#)



Сентябрь 1, 2018 at 23:53

Денис says:

Правильно ли я понимаю, в первом случае я не смогу использовать методы/свойства RobotTotal кроме тех, которые либо переопределены либо есть в Robot? А во втором я могу пользоваться всеми свойствами как родителя так и наследника?

[Reply](#)



Сентябрь 2, 2018 at 03:47

admin says:

Да, именно так. Хотя объект один и тот же — RobotTotal. Но ссылки «смотрят» на него по-разному. Это дает возможность создавать потомков и менять их поведение в переопределяемых методах. И когда вы передаете его в кому-то как родителя, принимающий «ничего не подозревает». Таким образом можно создавать новые реализации и подключать их «на лету».

[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

восемь × один = 

Copyright © 2018 [Java Course](#)

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

	94	
	58	
	57	