



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](#)

Визуализация «похождений» робота

Пришло время посмотреть на нашего робота симпатичным образом. Мы создадим уже более сложное приложение, которое сначала «выгуляет» нашего робота, а потом нарисует его путь на форме. Думаю, что это будет неплохой иллюстрацией отношений между классами и примером использования классов. Для реализации мы создадим следующие классы:

1. Robot — класс робота, который будет передвигаться и поворачиваться. Но что важно отметить — робот будет запоминать весь путь, который он проходил — т.е. будет вести список прямых, которые он проезжал. Для хранения списка мы используем стандартный класс Java ArrayList. Этот класс мы будем рассматривать более подробно позже при изучении коллекций, а сейчас мы просто будем его использовать «вслепую» — просто знайте, что у него есть метод add, который позволяет добавить в список объект. Также список объектов можно «просмотреть» с помощью конструкции for, которая приведена в классе RobotPathComponent.
2. RobotLine — класс для хранения координат одного отрезка пути. Это простой класс, который включает 4 числа — координаты начальной точки (X1, Y1) и координаты конечной точки (X2, Y2).
3. RobotPathComponent — этот класс наследуется от уже знакомого нам класса JComponent. Этому классу передается робот со своим списком отрезков пути. В методе paintComponent путем перебора всех отрезков мы получаем координаты каждого и рисуем линию вызовом метода drawLine класса Graphics.
4. RobotFrame — класс для отображения формы. Мы уже встречались с подобной реализацией.

- [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](#)

5. RobotManager — класс для запуска всего приложения. В его методе main мы сначала создаем робота и указываем ему нарисовать 12-ти угольник. Вы можете задать другое количество, изменив значение переменной COUNT. После «прогулки» мы создаем форму и передаем ей нашего робота для отрисовки его пути. Если задать большое количество сторон, то из-за округлений наш многоугольник не замкнется. Можете попробовать это исправить.

Думаю, что комментариев должно быть достаточно — теперь вы можете посмотреть код наших классов и скачать проект для запуска — [Robot4](#).

Класс Robot

```

1 package edu.javacourse.robot;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Robot
7 {
8     private double x = 0;
9     private double y = 0;
10    protected double course = 0;
11    // Список для хранения линий, по которым перемещался робот
12    // Пока будем использовать его без подробностей
13    private ArrayList<RobotLine> lines = new ArrayList<RobotLine>();
14
15    public Robot(double x, double y) {
16        this.x = x;
17        this.y = y;
18    }
19

```

```

20 public void forward(int distance) {
21     // Запоминаем координаты робота перед перемещением
22     final double xOld = x;
23     final double yOld = y;
24     // Меняем координаты
25     x += distance * Math.cos(course / 180 * Math.PI);
26     y += distance * Math.sin(course / 180 * Math.PI);
27     // Запоминаем координаты пройденного пути в списке
28     // Класс List позволяет добавить объект и хранить его
29     lines.add(new RobotLine(xOld, yOld, x, y));
30 }
31
32 public double getX() {
33     return x;
34 }
35
36 public double getY() {
37     return y;
38 }
39
40 public double getCourse() {
41     return course;
42 }
43
44 public void setCourse(double course) {
45     this.course = course;
46 }
47
48 public ArrayList<RobotLine> getLines() {
49     return lines;
50 }
51 }

```

Класс RobotLine

```

1 package edu.javacourse.robot;
2
3 // Класс для хранения одной линии, которую проехал робот
4 public class RobotLine
5 {
6     private double x1;
7     private double y1;
8     private double x2;
9     private double y2;

```

```

10
11     public RobotLine(double x1, double y1, double x2, double y2) {
12         this.x1 = x1;
13         this.y1 = y1;
14         this.x2 = x2;
15         this.y2 = y2;
16     }
17
18     public double getX1() {
19         return x1;
20     }
21
22     public double getY1() {
23         return y1;
24     }
25
26     public double getX2() {
27         return x2;
28     }
29
30     public double getY2() {
31         return y2;
32     }
33 }

```

Класс RobotPathComponent

```

1  package edu.javacourse.robot.ui;
2
3  import edu.javacourse.robot.Robot;
4  import edu.javacourse.robot.RobotLine;
5  import java.awt.Graphics;
6  import javax.swing.JComponent;
7
8  public class RobotPathComponent extends JComponent
9  {
10     private Robot robot;
11
12     public RobotPathComponent(Robot robot) {
13         this.robot = robot;
14     }
15
16     @Override
17     protected void paintComponent(Graphics g) {

```

```

18     super.paintComponent(g);
19     // Перебираем все линии, которые сохранились у робота
20     // Несколько позже мы разберем эту конструкцию подробно
21     for (RobotLine rl : robot.getLines()) {
22         // Для каждой линии получаем координаты
23         int x1 = (int) Math.round(rl.getX1());
24         int y1 = (int) Math.round(rl.getY1());
25         int x2 = (int) Math.round(rl.getX2());
26         int y2 = (int) Math.round(rl.getY2());
27         // И рисуем линию с координатами
28         g.drawLine(x1, y1, x2, y2);
29     }
30 }
31 }

```

Здесь мы видим конструкцию, которая может вас озадачить — у нее даже комментарии такие Вот эта

for (RobotLine rl : robot.getLines()).

Давайте пока примем ее как есть. Чтобы не было совсем непонятно — эта конструкция перебирает каждый элемент внутри списка и помещает ее в переменную **rl**. Т.е. внутри цикла эта переменная указывает на элемент списка и тип этого элемента **RobotLine**. Можно себе представить как будто мы просматриваем кадры киноплёнки. На каждом шаге мы перемещаем окошко просмотра на следующий кадр. Окошко просмотра — это переменная **rl**. С помощью этой переменной мы можем посмотреть параметры конкретного кадра — с нашем случае линии с координатами начала и конца (X1, Y1, X2, Y2).

Класс RobotFrame

```

1  package edu.javacourse.robot.ui;
2
3  import edu.javacourse.robot.Robot;
4  import javax.swing.JFrame;
5
6  public class RobotFrame extends JFrame
7  {
8      public RobotFrame(Robot robot) {
9          // Устанавливаем заголовок окна
10         setTitle("Robot Frame");
11         // Добавляем компонент для рисования пути робота
12         add(new RobotPathComponent(robot));
13         // Устанавливаем размеры окна
14         setBounds(100, 100, 500, 500);
15     }
16
17 }

```

Класс RobotManager

```
1 package edu.javacourse.robot;
2
3 import edu.javacourse.robot.ui.RobotFrame;
4 import javax.swing.JFrame;
5
6 public class RobotManager
7 {
8     public static void main(String[] args) {
9         // Количество сторон многоугольника
10        final int COUNT = 12;
11        // Длина стороны
12        final int SIDE = 100;
13
14        Robot robot = new Robot(200, 50);
15        // Создаем замкнутую фигуру с количеством углов COUNT
16        for (int i = 0; i < COUNT; i++) {
17            robot.forward(SIDE);
18            robot.setCourse(robot.getCourse() + 360 / COUNT);
19        }
20
21        // Создаем форму для отрисовки пути нашего робота
22        RobotFrame rf = new RobotFrame(robot);
23        rf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24        rf.setVisible(true);
25    }
26 }
```

Можете поиграть с нашим роботом. Учтите, что координаты отсчитываются от верхнего левого угла. Хотелось бы заострить ваше внимание на следующем моменте — по коду уже можно видеть, что мы создавая объекты по сути «играем» с ними, указывая, что им делать и в каком порядке. Большинство программ именно так и создаются — просто классов больше, отношения более сложные. Но идея одна — создать объекты нужных классов и заставить их работать в кооперации. Это требует умения и мастерства, которое надо накапливать создавая программы — другого пути пока не придумали.

Попробуйте в качестве маршрута нарисовать какую-нибудь сложную фигуру — например пятиконечную звезду. Или сделать квадратную спираль от самых краев к центру — каждый квадрат все меньше и меньше.

И теперь нас ждет следующая статья: [Пример — очередь объектов](#).

58 comments to *Визуализация робота*



Апрель 3, 2015 at 10:13

javaNoob says:

Ух!!! Сильная статья! С мощным примером!

Исходя из прочитанного и разобранного кода, я осмелюсь сделать вывод, что полиморфизм является краеугольным камнем ООП. Это свобода реализации различных ситуаций возникающих и могущих возникнуть..

Огромное спасибо!!

[Reply](#)



Апрель 10, 2015 at 22:31

Stitch says:

«Если задать большое количество сторон, то из-за округлений наш многоугольник не замкнется. Можете попробовать это исправить.»

Исправлено. проблема была в том, что при делении `360/COUNT` в делении участвовали два числа типа `int` и результат округлялся, на чем мы и допускали погрешность, которая не позволяла замкнуться фигуре, прокастовав одно из чисел, получили результат без погрешности типа `double`). Повод к статье о вычислениях примитивов, кастовании и т.д. Спасибо Вам за Ваш труд, уроки более чем доступны, прочел с удовольствием, хотя материал мне знаком, но как говорили наши бабушки «повторение — мать учения»)

```
robot.setCourse(robot.getCourse() + 360 / (double)COUNT);
```

[Reply](#)



Сентябрь 2, 2016 at 12:00

Денис says:

Можно немного проще:

```
robot.setCourse(robot.getCourse() + 360. / COUNT);
```

[Reply](#)



• Ноябрь 27, 2015 at 13:31

[Олег](#) says:

Добрый день!

Наткнулся на Ваш сайт и понравился код по визуализации робота но не получается реализовать маршрут робота по квадратной спирали от самых краев к центру – каждый квадрат все меньше и меньше, если нетрудно с чего начать?

[Reply](#)



○

Ноябрь 27, 2015 at 14:51

admin says:

Я бы начал с того, что нарисовал эту фигуру на листочке бумаги. Потом попробовал выяснить закономерность изменения линий и это как раз самая сложная часть. После решения этой задачи все будет проще.

[Reply](#)



○

Сентябрь 21, 2016 at 02:01

Creed says:

Robotmanager с 14 по 20 строки замените на :

```
Robot robot = new Robot(5, 5);
```

```
int side = SIDE;
```

```
int count = 0;
```

```
while(side>0){
```

```
count++;
```

```
side -=5;
```

```
robot.forward(side);
```

```
robot.setCourse(count*90);
```

```
}
```


[Reply](#)



• Ноябрь 27, 2015 at 21:14

[Олег](#) says:

Я так понимаю, надо копать здесь, но как он должен повернуть так как мне надо, не придумал

```
x += distance * Math.cos(course / 180 * Math.PI);  
y += distance * Math.sin(course / 180 * Math.PI);
```

[Reply](#)



◦

Ноябрь 28, 2015 at 20:14

admin says:

Для поворота надо изменить курс — используйте `setCourse`

[Reply](#)



• Декабрь 4, 2015 at 22:15

[Олег](#) says:

если запрограммировать кнопки вперед, назад это понятно, но с помощью `setCourse` не могу разобраться?

[Reply](#)



◦

Декабрь 7, 2015 at 10:09

admin says:

Можно запрограммировать кнопки «Налево» и «Направо» — и тогда курс робота можно изменять.

[Reply](#)



Декабрь 7, 2015 at 11:14

[Олег](#) says:

а как же повернуть не с помощью кнопок?

[Reply](#)



Декабрь 23, 2015 at 14:57

Булат says:

Спиральку вот так сделал:

```
for (int i = 0; i < COUNT; i++) {  
  robot.forward(SIDE+SIDESTEP*i);  
  // robot.setCourse(robot.getCourse() + 360 / COUNT);  
  robot.setCourse(robot.getCourse() + 90);  
}
```

правильно?

[Reply](#)



Декабрь 23, 2015 at 15:06

admin says:

Я не проверял результат, но есть подозрение, что не совсем верно. В спирали больше одной стороны имеют одинаковую длину, а у Вас только одна. Сомневаюсь я в правильности. Вы проверяли работоспособность этого ?

[Reply](#)



Декабрь 24, 2015 at 15:01

Булат says:

Да. это пока единственное что понимаю из курса))))

Вот сделал корректнее:

```
for (int i = 0; i < COUNT; i++) {  
    robot.forward(SIDE*COUNT-SIDESTEP*i);  
    robot.setCourse(robot.getCourse() + 90);
```

```
}
```

[Reply](#)



Декабрь 24, 2015 at 15:37

admin says:

Это уже ближе. В принципе я когда-то написал так:

```
1 package edu.javacourse.robot;  
2  
3 import edu.javacourse.robot.ui.RobotFrame;  
4 import javax.swing.JFrame;  
5  
6 public class RobotManager  
7 {  
8     public static void main(String[] args) {  
9         final int SIDE = 450;  
10        final int STEP = 30;  
11  
12        int current = SIDE;  
13  
14        Robot robot = new Robot(10, 10);  
15        robot.forward(current);  
16        robot.setCourse(robot.getCourse() + 90);  
17        while(current > 0) {
```

```
18         robot.forward(current);
19         robot.setCourse(robot.getCourse() + 90);
20         robot.forward(current);
21         robot.setCourse(robot.getCourse() + 90);
22         current -= STEP;
23     }
24
25     // Создаем форму для отрисовки пути нашего робота
26     RobotFrame rf = new RobotFrame(robot);
27     rf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28     rf.setVisible(true);
29 }
30 }
```

[Reply](#)



• Январь 9, 2016 at 13:28

Александр says:

С какой целью были созданы два пакета? package edu.javacourse.robot.ui и package edu.javacourse.robot
Если все классы «вписать» в один пакет всё будет работать

[Reply](#)



○

Январь 10, 2016 at 01:26

admin says:

Потому что мне показалось будет правильнее — разделить классы, которые отвечают за разные функции по разным пакетам. Это вопрос достаточно субъективный.

[Reply](#)



• Март 3, 2016 at 19:32

Pavel says:

Вариант спирали.

Интересно кто как звезду рисовал.

```
for (int i = 0; i < COUNT; i++) {  
  for (int n = 0; n < 4; n++) {  
    robot.forward(SIDE);  
    robot.setCourse(robot.getCourse() + 90);  
    SIDE *= 0.9;  
  }  
}
```

[Reply.](#)



o

Март 3, 2016 at 19:35

Pavel says:

Вариант звезды:

```
final int COUNT = 5;  
final int SIDE = 100;  
final int alfa = 360 / COUNT / 2;  
Robot robot = new Robot(100, 200);  
for (int i = 0; i < COUNT; i++) {  
  robot.forward(SIDE);  
  robot.setCourse(robot.getCourse() — 2 * alfa);  
  robot.forward(SIDE);  
  robot.setCourse(robot.getCourse() + 360 / (double)COUNT + 2 * alfa);  
}
```

[Reply.](#)



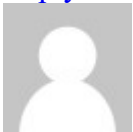
•

Март 17, 2016 at 15:06

Екатерина says:

```
public static void main(String[] args) {  
    final int COUNT = 10;  
    final int SIDE = 120;  
    Robot robot = new Robot(100,200);  
    for (int i = 0; i < COUNT/2; i++) {  
        robot.forward(SIDE);  
        robot.setCourse(robot.getCourse() + 2* 360 / COUNT);  
        robot.forward(SIDE);  
        robot.setCourse(robot.getCourse() + 4*360 / COUNT);  
    }  
}
```

[Reply.](#)



Май 12, 2016 at 13:57

Владимир says:

А есть ли какой то пример, чтобы изображение масштабировалось при изменении окна? Как в случае с овалом.

[Reply.](#)



o

Май 12, 2016 at 14:01

admin says:

В принципе интернет можно спросить о таком примере: <http://stackoverflow.com/questions/15558202/how-to-resize-image-in-java>

[Reply.](#)



o

Май 31, 2016 at 12:33

Nibbler says:

Тоже «залип» с этой идеей... В конце-концов реализовал — правда Мастер, вряд ли, одобрит. Пришлось сделать отдельный класс: DistanceModify.java в котором реализован статический метод setWH, принимающий в качестве параметров ширину, высоту и самого робота. Метод вызывается из тела paintComponent следующим образом: DistanceModify.setWH(getWidth(),getHeight(),robot); Только так можно получить ширину и высоту графического поля. Поскольку после масштабирования окна нужно заново пересчитывать массив линий — алгоритм перемещений тоже ушел в DistanceModify.java. В такой реализации идеально масштабируется прямоугольник:

```
1 package edu.javacourse.robot;
2
3 /**
4  *
5  * @author user
6  */
7 public class DistanceModify {
8     private static final int COUNT = 4;
9     public static void setWH(int width, int height, Robot robot){
10         robot.linesClear();
11         for(int i=0; i<COUNT;i++){
12             if(i%2==0){
13                 robot.forward(width-20);
14             } else {
15                 robot.forward(height-20);
16             }
17             robot.setCourse(robot.getCourse()+90);
18         }
19     }
20 }
```

Если же играть константой COUNT и подставить формулу расчета курса, которую я приводил для звезды ниже — при растягивании окна получаются «метаморфозы», как в калейдоскопе.

Да — массив линий придется очищать перед каждым новым «перестроением»: robot.linesClear(); иначе полная каша получится. Наверное, можно было сделать все проще и красивее. Но пока еще знаю очень мало для этого.

[Reply](#)



Май 27, 2016 at 14:43

Nibbler says:

Вот такой вариант «ленивой» звезды:

```
final int COUNT = 10;  
robot.setCourse(robot.getCourse()+(180-360/COUNT));
```

В детстве, вроде бы, так рисовали 😊

[Reply](#)



o

Февраль 25, 2017 at 20:46

Юрий says:

```
final int COUNT = 5;  
robot.setCourse(robot.getCourse() + (180-180/COUNT));
```

[Reply](#)



•

Июнь 24, 2016 at 16:40

IntelligenceUniverse says:

Звезда

```
public static void main(String[] args) {  
    final int COUNT = 5;  
    final int COUNT2 = 10;  
    final int SIDE = 100;  
  
    Robot robot = new Robot(200, 100);  
    for (int i = 0; i < COUNT; i++) {  
        robot.forward(SIDE);  
        robot.setCourse(robot.getCourse() + 360 / COUNT2);  
        robot.forward(SIDE);  
        robot.setCourse(robot.getCourse() + 180 — 360 / COUNT);  
    }  
}
```

[Reply](#)



Июнь 27, 2016 at 17:23

Firefly says:

По-моему, спираль достаточно просто рисуется (вдруг кому пригодится):

```
// Количество сторон многоугольника
final int COUNT = 4;
// Длина стороны
final int SIDE = 100;

Robot robot = new Robot(150, 100);
// Создаем замкнутую фигуру с количеством сторон 12
for (int i = 0; i < 12; i++) {
    // уменьшаем длину стороны на 20 каждые два шага
    robot.forward(SIDE — 20 * (i/2));
    robot.setCourse(robot.getCourse() + 360 / COUNT);
}
```

[Reply.](#)



Июнь 27, 2016 at 17:25

Firefly says:

А вот у меня вопросик: а можно ли эту визуализацию сделать анимационной?

[Reply.](#)



Июнь 27, 2016 at 18:01

admin says:

Можно. Только для этого надо запускать поток и по таймеру рисовать.

[Reply](#)



Июнь 29, 2016 at 12:51

Firefly says:

Спасибо, значит будет изучать дальше 😊

[Reply](#)



Июль 1, 2016 at 17:58

Vladimir says:

линии с координатами начала и конца (X1, Y2, X2, Y2).

[Reply](#)



Июль 3, 2016 at 18:38

admin says:

Спасибо, исправил.

[Reply](#)



Сентябрь 20, 2016 at 13:31

[Антон](#) says:

```
for (int i=0; i<count; i++){  
robot.forward(side-i);
```

```
robot.setCourse(robot.getCourse()+90);  
}
```

Квадратная спираль

[Reply](#)



o

Ноябрь 1, 2016 at 16:57

v says:

можно модернизировать: `robot.forward(side-i*indent);` //где indent — отступ

[Reply](#)



•

Сентябрь 20, 2016 at 13:41

[Антон](#) says:

Плюс в том, что вне зависимости от количества поворотов всё адекватно рисуется. Если сделать зависимость от размера экрана (самой длинной стороны) и величины `side`, то можно даже потянуть размеры в любые стороны экран — всё будет отображаться правильно. Далее можно сделать `count` зависимой от максимального числа поворотов. Формула будет что-то из разряда `count=(side-1)*4`. Не скажу за достоверность, но это на вскидку.

[Reply](#)



•

Ноябрь 2, 2016 at 07:49

v says:

треугольная спираль

```
final int COUNT = 3;  
final int SIDE = 400;
```

```
int k = 50;
Robot robot = new Robot(50, 50);

for (int i = 0; i < k; i++) {
    robot.forward(SIDE — i*6);
    robot.setCourse(robot.getCourse()+360/COUNT);
}
```

[Reply.](#)



• Ноябрь 2, 2016 at 07:55

v says:

многоугольная спираль, можно достигнуть классической спирали (также прикрутил printCoordinates() для информации из предыдущих уроков)

```
final int COUNT = 10;
final int SIDE = 5;
int k = 50;
int indent = 2;
Robot robot = new Robot(150, 150);
robot.printCoordinates();

for (int i = 0; i < k; i++) {
    robot.forward(SIDE — i*indent);
    robot.setCourse(robot.getCourse()+360/COUNT);
}
```

[Reply.](#)



o Ноябрь 2, 2016 at 07:57

v says:

robot.printCoordinates(); также добавить в цикл надо

[Reply.](#)

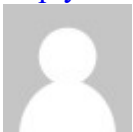


Январь 24, 2017 at 13:24

Паша says:

```
public class RobotManager {  
    public static void main(String[] args){  
  
        final int COUNT = 82;  
        int SIDE = 400;  
  
        Robot robot = new Robot(50,50);  
  
        for (int i = 1; i 4 && i % 2 == 0){  
            SIDE -= 10;  
        }  
  
        robot.forward(SIDE);  
        robot.setCourse(robot.getCourse() + (COUNT * 90) / COUNT);  
    }  
  
    RobotFrame rf = new RobotFrame(robot);  
    rf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    rf.setVisible(true);  
}
```

[Reply](#)



Январь 24, 2017 at 13:26

Паша says:

```
public static void main(String[] args){  
  
    final int COUNT = 82;  
    int SIDE = 400;
```

```

Robot robot = new Robot(50,50);

for (int i = 1; i 4 && i % 2 == 0){
    SIDE -= 10;
}

robot.forward(SIDE);
robot.setCourse(robot.getCourse() + (COUNT * 90) / COUNT);
}

RobotFrame rf = new RobotFrame(robot);
rf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
rf.setVisible(true);
}
}

```

[Reply.](#)



• Февраль 3, 2017 at 22:31

Таня says:

Звезда:

```

public class RobotManager
{
    public static void main(String[] args) {
        final int COUNT = 5;
        final int SIDE = 200;

        Robot robot = new Robot(200, 200);
        for (int i = 0; i < COUNT; i++) {
            robot.forward(SIDE);
            robot.setCourse(robot.getCourse() + 2*360 / COUNT);
        }

        RobotFrame rf = new RobotFrame(robot);
        rf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

```

```
rf.setVisible(true);  
}  
}
```

[Reply](#)



• Февраль 3, 2017 at 23:14

Таня says:

квадратная спираль:

```
public class RobotManager  
{  
    public static void main(String[] args) {  
        final int COUNT = 8;  
        final int SIDE = 200;
```

```
        Robot robot = new Robot(100, 50);  
        for (int i = SIDE; i > 0; i = i — 10) {  
            robot.forward(i);  
            robot.setCourse(robot.getCourse() + 90);  
        }
```

```
        RobotFrame rf = new RobotFrame(robot);  
        rf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        rf.setVisible(true);  
    }  
}
```

[Reply](#)



o

Февраль 3, 2017 at 23:31

Таня says:

сорри, тут переменная COUNT не нужна

[Reply](#)



• Февраль 13, 2017 at 19:14

Владимир says:

Полностью повторил ваш код про рисующего робота. Всё работает.

У меня только один вопрос — при запуске в Netbeans класс RobotPathComponent вызывается 2 или 3 раза. Как такое вообще может быть?!

[Reply](#)



○

Февраль 14, 2017 at 13:12

admin says:

А каким образом Вы это обнаружили ?

[Reply](#)



• Февраль 14, 2017 at 20:57

Владимир says:

Случайно. Решил использовать связанный список для построения маршрута, и вместо метода get() использовал метод pull(). Программа постоянно зависала и выдавала NullPointerException. Поэтому поставил после каждой итерации вывод значения переменных на консоль. И оказалось, что робот проходит маршрут один раз, а вот отрисовывает то два, а то и три раза, т.е. одна и та же программа выдает РАЗНЫЕ результаты. Ума не приложу как это получается.

[Reply](#)



○

Февраль 15, 2017 at 11:20

admin says:

То, что прорисовывается не один раз, а несколько — это как раз можно принять.

В принципе правильно прорисовывать форму надо в отдельном специальном потоке, но в данном случае это уже разговор про потоки исполнения, о которых я рассказываю позже.

[Reply](#)



■ Февраль 15, 2017 at 22:19

Владимир says:

То есть это нормально? До потоков я еще не добрался...

[Reply](#)



■ Февраль 16, 2017 at 09:55

admin says:

Штука в том, что каждый раз при перерисовке формы компонент будет себя перерисовывать.

Как форма решает, когда ей себя перерисовывать — это уже зависит от системы Swing, которая весьма не простая.

Я глубоко не копался внутри Swing, но судя по тому, что я знаю, такой вариант развития событий вполне возможен.

Особенно в нашем не совсем правильном решении.

[Reply](#)



• Май 29, 2017 at 11:52

[E=MC^2](#) says:

Задался вопросом, а как сделать чтобы на RobotFrame отображались действия двух разных роботов одновременно. И еще один вопрос может не по теме: есть ли знак плюс-минус в java? Я захотел сделать чтобы робот рисовал параболу, но получилась только одна ветвь.

[Reply](#)



o

Май 29, 2017 at 13:32

admin says:

1. В той реализации, которая есть — нет возможности. Надо тогда делать список роботов вместо одного робота. Либо делать два компонента RobotPathComponent.
2. Нет такого знака — придумывать надо что-то другое.

[Reply](#)



•

Май 29, 2017 at 12:11

[E=MC^2](#) says:

Вот сделал Архимедову спираль:

```
class Robot
x += distance*Math.cos(distance) ;
y += distance*Math.sin(distance);
class RobotManager
for (double i =0; i<200 ;i++) {

robot.forward((double) 0.1*i);
}
```

[Reply](#)



•

Февраль 18, 2018 at 16:04

Юрий says:

```
Конструктор Робота
public Robot(double x, double y) {
this.x = x;
```

```
this.y = y;  
}
```

После компиляции-декомпиляции превращается в

```
public Robot(final double x, final double y) {  
    this.x = 0.0;  
    this.y = 0.0;  
    this.course = 0.0;  
    this.lines = new ArrayList();  
    this.x = x;  
    this.y = y;  
}
```

какие у Java отношения с оптимизацией ?

[Reply](#)



o

Февраль 19, 2018 at 09:43
admin says:

Не скажу, почему так. Я бы еще задал вопрос по поводу декомпилятора — что он собой представляет. Может это его работа.

[Reply](#)



•

Июль 16, 2018 at 11:55
Сергей says:

Добрый день.

А у меня почему то рисует просто наклонную линию, никакого 12-угольника нет....несколько раз код сравнил, все одинаково....куда «копать» и что делать????

[Reply](#)



o

Июль 17, 2018 at 04:22

admin says:

Код на странице точно работает — я его проверял неоднократно/ Просто копировал все классы и запускал. Так что сравнивайте.

Попробуйте скопировать код со страницы и сравнить в какой-либо программе. Ну или попросите кого-нибудь читать ваш код, а вы будете проверять по моему коду. Или наоборот. Я сам такое видел неоднократно — человек утверждает, что все точно, а потом находится ошибка. Это тоже работа программиста — найти ошибку.

[Reply](#)



•

Июль 17, 2018 at 14:41

Сергей says:

Прям мистика какая-то.....глазами пробежал — код одинаковый, распечатал все классы на бумаге из Вашего и своего проекта — весь код одинаковый, но Ваш проект работает, а мой нет.....потом в своем проекте в классах удалил весь текст, скопировал из Вашего проекта, и о чудо.....все заработало....очень странно :)))

[Reply](#)



•

Июль 17, 2018 at 15:23

Сергей says:

Решил все таки докопаться до ошибки....запустил Debug с точкой в строке, где меняются координаты и обнаружилось, что у координаты у вместо `sin` написан был `cos`, как и у координаты `x`.

и как такая опечатка в глаза при неоднократных проверках не попала...

[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

— два = четыре 

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](#)

