



# 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 работа](#)
- [Пример — очередь объектов](#)
- [Массивы — знакомство](#)
- [Многомерные массивы](#)
- [Абстрактные классы](#)
- [Интерфейсы](#)

## Объектно-ориентированное программирование

Думаю, что количество книг по объектно-ориентированному программированию перевалило за сотни, а может и тысячи. До сих пор остается классическим труд Гради Буча «Объектно — ориентированный анализ и проектирование с примерами приложений на C++». Вы наверняка сможете найти этот труд на просторах интернета.

Также можно посмотреть в Интернете другие ссылки и статьи — их очень много. Например тут — [Объектно-ориентированное программирование](#) или тут — [Введение в программирование на Java](#). Второй ресурс содержит введение в язык Java. Весьма неплохо читается, так что отнеситесь к нему с вниманием.

Я попробую описать идеи ООП со своей колокольни. Сама идея появления очень простая: сложность программ растет. Человек уже давно разработал весьма несложное идеологическое решение этого вопроса — декомпозицию, т.е. разделение задачи на крупные блоки, потом на более мелкие, еще мельче и т.д.

По большому счету из этого понимания и родилось процедурное программирование. Программа представляла собой последовательный вызов процедур, которые принимали какие-то параметры на вход, отдавали какие-то параметры. Когда мы были студентами, то практически все писали подпрограммы числового интегрирования, построения графиков и прочая.

Но программы становились еще сложнее. Причем с катастрофической скоростью. В качестве примера могу привести статистику (правда она была сделана уже после появления ООП) — за 10 лет с 1990 по

- [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](#)

2000 сложность программ выросла в 100 раз.

И идея, которую предложили разработчики языка [Simula 67](#) оказалась великой. Позже на этих идеях были построены самые известные объектно-ориентированные языки, такие как Java, C++, C#, Smalltalk.

## Объекты

В реальном мире нас окружают объекты. Мы живем в квартирах, катаемся на машинах, заказываем товары, оплачиваем счета, делаем заказы. Все это — объекты реального мира. Если вы помните — мы с самого начала ввязались в историю с объектом «робот». Он у нас должен был передвигаться, поворачиваться. А значит писать программы, которые использовали бы именно такой подход будет удобно.

Но что важно отметить — если вы помните, в самом начале я описал две задачи, которые решает программист. ([Введение](#)). Для решения обеих задач подход ООП в огромном количестве случаев прекрасно подходит. Описать задачу на обычном языке удобно в объектах. И перевести задачу, описанную в объектах, на язык программирования, который также поддерживает объекты — опять таки удобно. Наверно это является ключевым, поэтому я повторю свою мысль еще раз — в ООП важно понимать, что во-первых, описание задачи удобно вести в объектах. И переводить описание в язык программирования с объектами — тоже удобно. Давайте порассуждаем немного в отрыве от языков программирования — просто попробуем потеоретизировать.

## Классы

Развивая идею «при программировании задачи удобно использовать объекты» мы переходим к следующему шагу — объекты удобно классифицировать. В вашей программе может быть много тех же роботов — список замечательных роботов, каждый из которых умеет делать одно и то же. Мы можем их распределить по нескольким точкам и начать рисовать сразу десятки квадратов. Еще более может впечатлять идея создать описания объектов типа «воин» и начать сражение сил добра и зла :).

Причем я не просто так привел такой пример — если вы создаете объект, то вы получаете механизмы, чтобы научить этот объект выполнять какие-то операции. Можно научить нашего «воина» оценивать ситуацию вокруг него, принимать решения и выполнять определенные действия. Нападать, защищаться, передвигаться. А теперь представьте, что вы создали описание такого объекта в виде шаблона. И под этот шаблон создали программно штук 500-600 таких «воинов». Каждый «воин» обладает набором параметров — координаты, положение рук и ног, возможно качество брони, набор оружия. Поместив их на поле боя мы теперь можем заставить их сражаться. Причем они будут это делать уже по сути независимо от нас. Не надо писать сложный программный код для управления этими «воинами». Обычный цикл — оценить, принять решение, начать двигаться, рассчитать новые координаты. И все наше поле боя рассчитывается само собой. Остается только нарисовать воинов по уже готовым координатам. Добавление новых воинов не создаст никаких сложностей — лишь мощность процессора надо прибавить. Мы просто каждого просим делать в цикле определенный набор операций до тех пор, пока он жив. Оценили удобство ?

Теперь давайте посмотрим внимательно на наши рассуждения — мы создавали объекты по одному образцу и подобию. Мы говорим о том, что мы создали образец, чертеж, тип или КЛАСС. Именно понятие «класс» принято в ООП. Программист разрабатывает класс (Class) и потом программа создает объекты класса. Можно сказать, что мы, как инженеры, создаем чертеж машины, робота, «воина», а потом создаем объекты по этому чертежу.

Вернее программа их создает.

И тут надо отметить принципиальное отличие класса и объекта. Класс — это описание, объект — это реальное воплощение.

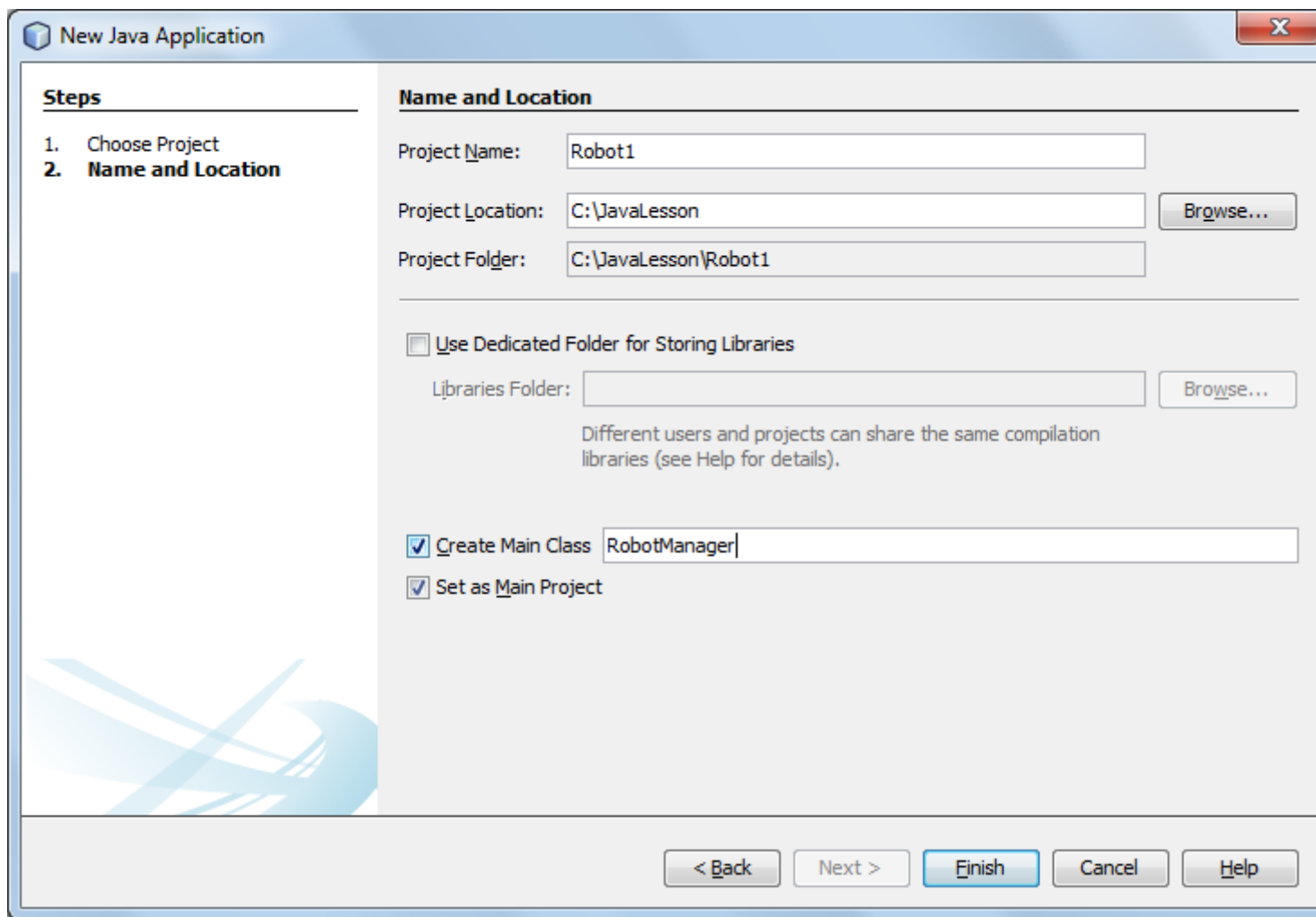
Это различие порождает еще одну важную мысль: например робот как класс (описание, чертеж) вряд ли требует понятия местоположения (координаты робота). Но для реального робота это важно. Теперь давайте подумаем, что наш робот может быть достаточно интеллектуальным — например, он может быть заброшен на Марс. В этом случае алгоритм его работы потребует знания координат. Заметьте — алгоритм требует координат. И эти координаты будут уникальны для каждого робота, находящегося на Марсе (вдруг мы туда десант забросим).

Потихонечку мы пришли к некоторым важным выводам:

1. Объект/класс обладает переменными, которые характеризуют его состояние (например, координаты). Не всегда параметры нужны и классу и объекту.
2. Объект/класс умеет что-то делать (например двигаться, оценивать ситуацию и прочая)

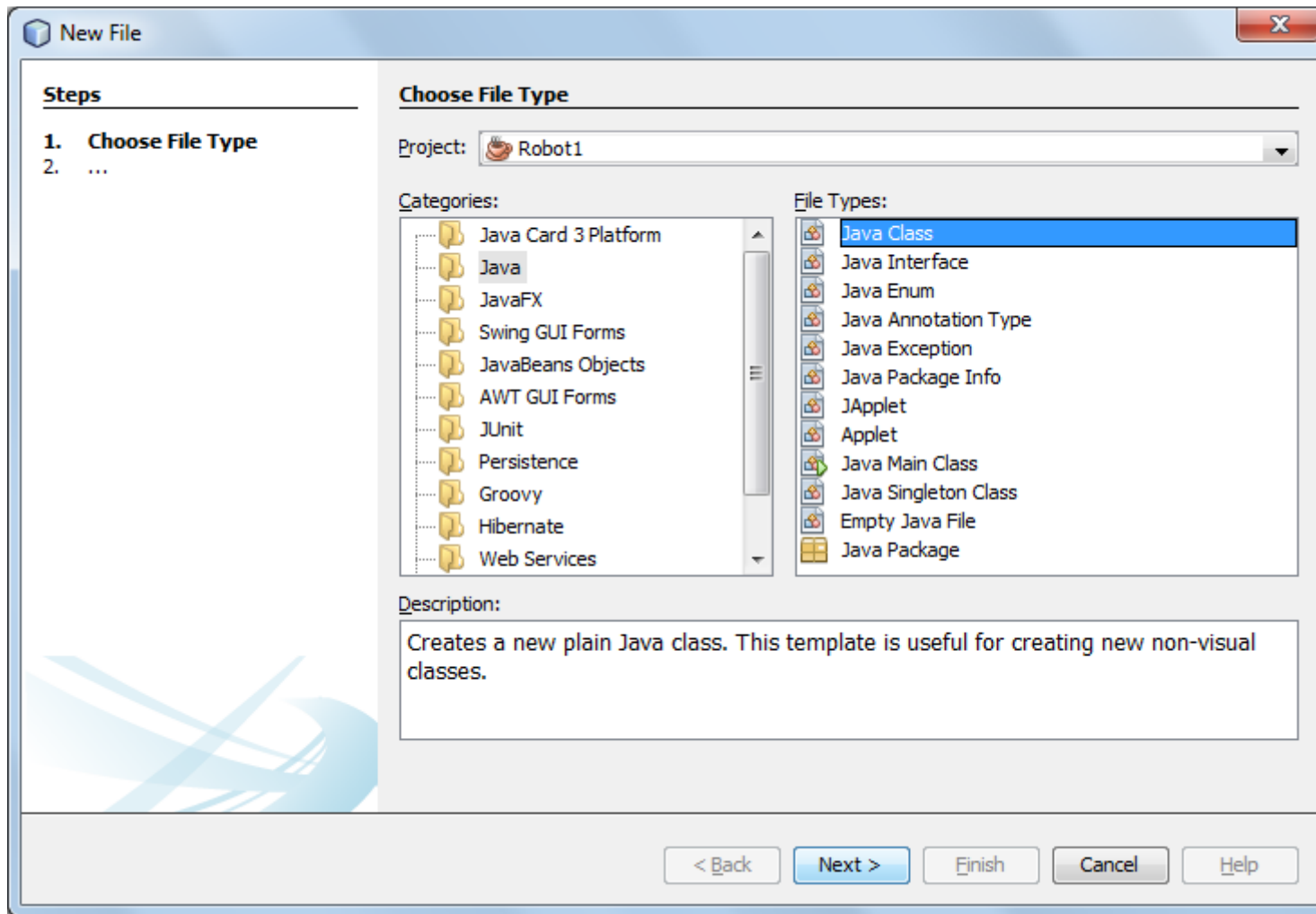
## **Пример класса на Java**

Напишем несложный класс с подробным описанием всего того, что мы в нем увидим. Давайте сразу делать это в рамках проекта на NetBeans. Для этого создаем простой проект. Назовем его Robot1. Запускаем NetBeans и делаем то, что описывалось раньше — выбираем пункт File -> New Project. Далее выбираем простой проект и устанавливаем параметры, которые показаны на рисунке.

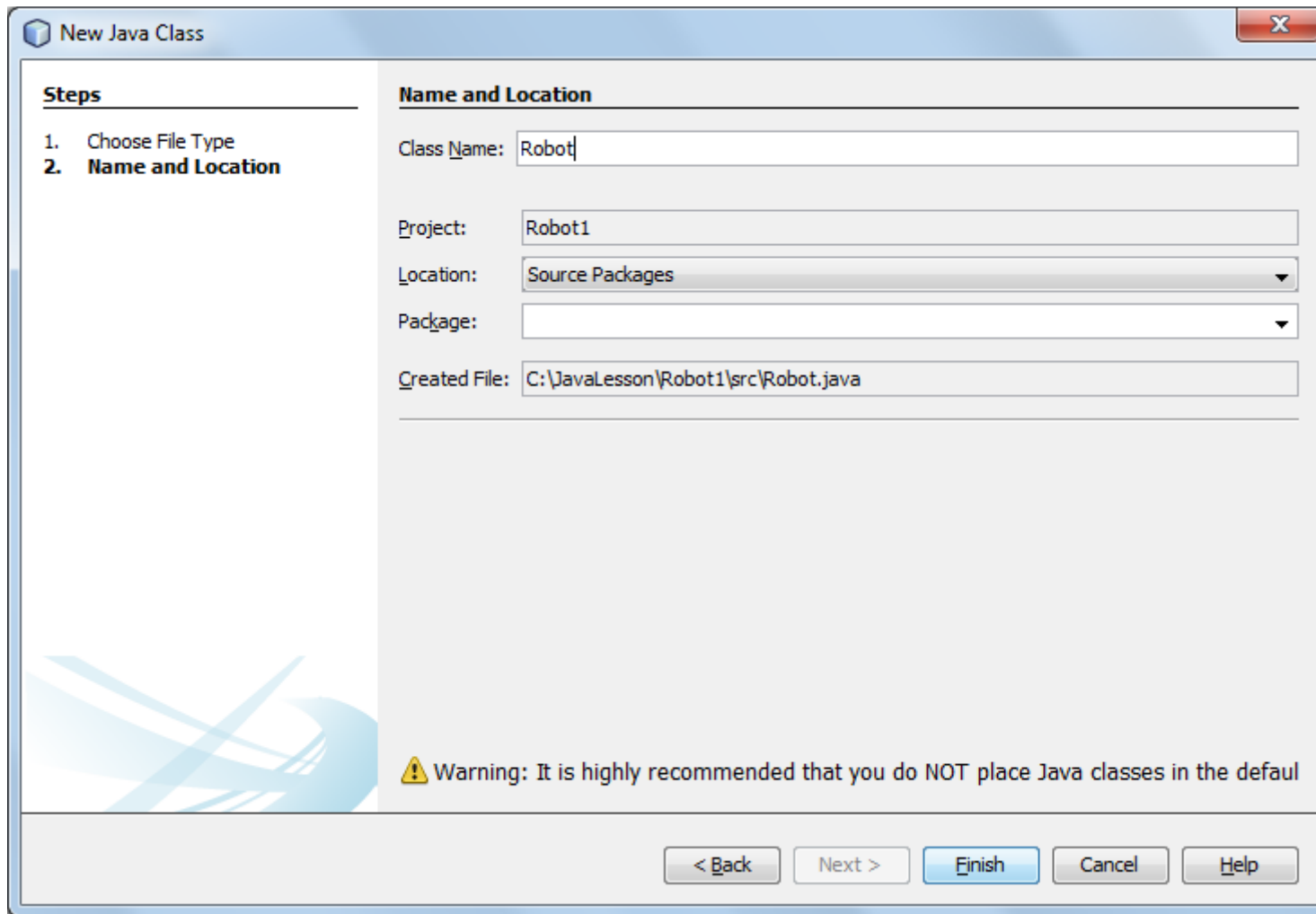


У нас сразу будет создан класс RobotManager, который мы указали в строке «Create Main Class». Пока не будем обращать на него внимания. Он нам потребуется немного позже. Сейчас создадим новый класс Robot.

Для этого выберем пункт меню File -> New File. Вы сможете увидеть, что там есть комбинация Ctrl+N а также видна иконка кнопки на верхней панели. В следующий раз можно использовать их. Перед нами откроется окошко в котором нам надо выбрать Categories: java и File Types: Java Class. Далее нажимаем «Next».



На следующем экране устанавливаем имя класса (Robot). Обратите внимание на сообщение внизу. Оно просит о том, чтобы класс был помещен в пакет. На данный момент будем игнорировать это сообщение, но мы обязательно вернемся к нему, когда будем разбирать пакеты. А пока просто нажимаем «Finish». (Забегая немного вперед — помещать класс в пакет хорошо. А класс без пакета — не хорошо. Но на данный момент можно).



После создания класса мы увидим вот такой код.

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5
6  /**
7   *
8   * @author Антон
```

```
9  */
10 public class Robot
11 {
12
13 }
```

Есть еще один способ создать новый класс. Для этого щелкните правой кнопкой мыши на проекте на панели Projects и в выпадающем меню выберите пункт New -> Java Class.

Итак, мы создали класс Robot. Он пока ничего не умеет делать и не содержит никаких параметров, но тем не менее это полноценный класс. И объекты этого класса можно создавать и что-нибудь с ними делать. (Забегая немного вперед скажу, что даже в таком состоянии класс уже многое умеет, но об этом мы поговорим несколько позже).

То, что это именно класс, говорит слово **class**. Слово **public** мы обсудим позже, но сейчас вы уже можете себе отметить, что внутри файла с именем **Robot.java** у нас есть (должен быть) класс **Robot**. И все начинается со слов **public class Robot**. Т.к. мы пока не обсуждали слово **public**, просто примите к сведению — имя файла ДОЛЖНО совпадать с именем класса, который указывается после именно двух этих слов — **public class**. (Мы еще узнаем, что без слова **public** вы можете указать несколько классов внутри одного файла. Но **public class** должен быть только ОДИН).

Все описание класса (поля, методы) должно находиться внутри фигурных скобок.

А пока давайте расширим возможности нашего класса — сделаем для нашего робота поля, которые будут хранить его координаты и курс — поля *x*, *y* и *course*. Сделать это несложно — мы описываем переменные как показано ниже. (Я убрал лишние комментарии и добавил более полезные).

```
1 public class Robot
2 {
3
4     // Текущая координата X
5     int x = 0;
6     // Текущая координата Y
7     int y = 0;
8     // Текущий курс (в градусах)
9     int course = 0;
10 }
```

Это весьма простой класс. Мало того, он не полный. В нем нет очень важной составляющей — наш класс ничего не умеет делать. Сейчас у него есть просто три параметра: координаты *X* и *Y*, а также курс. Мало того — все параметры изначально равны нулю.

Но здесь мы видим как можно (и нужно) определять параметры класса. А теперь попробуем немного поработать с нашим классом Robot. Откройте в редакторе описание класса RobotManager. Для этого его надо просто найти слева в панели Projects и дважды щелкнуть левой кнопкой мыши.

Обратите внимание на идею — класс RobotManager будет управлять объектами Robot. Название класса не является обязательным — здесь важно «увидеть» взаимодействие объектов. Перед нами класс RobotManager.

```

1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5
6  /**
7   *
8   * @author Антон
9   */
10 public class RobotManager
11 {
12
13     /**
14     * @param args the command line arguments
15     */
16     public static void main(String[] args) {
17         // TODO code application logic here
18     }
19 }

```

Теперь давайте попробуем поработать с нашим классом Robot. Как мы уже говорили при обсуждении элементарных типов мы видели, что переменные любого типа описываются в виде

<тип> имя;

Это же правило распространяется и на классы. Давайте объявим две переменные типа Robot.

```

1  public class RobotManager
2  {
3
4      public static void main(String[] args) {
5          Robot r1;
6          Robot r2;
7      }
8  }

```

И вот тут начинается путь к отличиям переменных типа класс от элементарной переменной. Когда мы с вами объявляли элементарную переменную, мы сразу получали готовый кусочек памяти, куда можно было поместить число, символ или логическое значение. С классами не так. То, что мы с вами объявили `Robot r1;` не создает реальный объект типа `Robot`. В данном случае переменные `r1`, `r2` только ССЫЛКИ (указатели). Прежде чем использовать их, нам надо создать объект и указать на него ссылкой. Выглядит это вот так:



```
Robot r1 = new Robot();
```

Обратите внимание на эту строку — она является очень важной. Мы используем ключевое слово `new` для того, чтобы СОЗДАТЬ ОБЪЕКТ. Т.е. при объявлении переменной типа `Robot` (или другого класса) мы не создаем объект — мы объявляем ссылку на объект. КОТОРЫЙ НАМ НАДО ЕЩЕ СОЗДАТЬ. Если этого не сделать, то ссылка будет указывать в никуда — для этого есть даже специальное слово — `null`.

```
1 public class RobotManager
2 {
3
4     public static void main(String[] args) {
5         // Создаем сразу при объявлении
6         Robot r1 = new Robot();
7         // Создаем отдельным оператором присваивания
8         Robot r2;
9         r2 = new Robot();
10    }
11 }
```

И вот только после создания объектов наши ссылки указывают на них. И только теперь нашими объектами можно пользоваться.

Если немного углубиться в механизм ссылок и объектов, то он достаточно сильно напоминает механизм указателей в C++. В области памяти, которая называется «куча» (heap) выделяется память, в которой находятся данные объекта. А ссылка указывает на эту область памяти. У тех, кто знаком с C++ или Delphi может возникнуть вопрос: «Если память выделена, то ее надо будет очищать?». Так вот в Java существует специальный механизм «Сборщик мусора» (garbage collector), который делает это автоматически. На данный момент мы не будем погружаться в его тонкости — просто имейте в виду, что создавать объекты вам надо самим, а удалять их из памяти вам не требуется. Позже мы подробнее остановимся на этом вопросе.

## Оператор обращения к полю класса

Давайте еще расширим наши знания. Только что мы с вами расширили наш класс переменными, которые содержат значения координат и курса. Нам необходим механизм, который позволит обращаться к этим полям. Выглядит он не просто, а ... очень просто. Вот так: «`.`». Да-да, самая простая точка. Давайте рассмотрим на примере.

```
1 public class RobotManager
2 {
3
4     public static void main(String[] args) {
5         Robot r1 = new Robot();
6         // Установить значение поля X у переменной r1
7         r1.x = 99;
8     }
```

```

9      // Напечатать поле X
10     System.out.println(r1.x);
11 }
12 }

```

Как можно видеть, обращение к полю X объекта, на который указывает ссылка r1, осуществляется через обычную точку. Я специально достаточно долго расписываю «обращение к полю X объекта, на который указывает ссылка r1» — я считаю очень важным моментом понимание механизма ссылок. В принципе он не сложный, но хорошо понимать его очень важно.

А теперь рассмотрим еще один пример:

```

1 public class RobotManager
2 {
3
4     public static void main(String[] args) {
5         // Ссылка r1 указывает на один объект
6         Robot r1 = new Robot();
7         // Ссылка r2 указывает на ДРУГОЙ объект
8         Robot r2 = new Robot();
9         // Установить значение поля X у переменной r1
10        r1.x = 99;
11        r2.x = 123;
12        // Напечатать поле X
13        System.out.println(r1.x);
14        System.out.println(r2.x);
15    }
16 }

```

Чем он важен? Самое главное в нем — увидеть, что мы создали ДВА объекта. На один указывает ссылка r1, на другой — r2. Также не менее важно увидеть, что для КАЖДОГО объекта мы отдельно устанавливаем поле X. И у КАЖДОГО объекта это поле имеет свое значение. Иными словами — каждый объект имеет свой личный набор полей, которые описаны в классе.

Только что мы посмотрели, как объявлять поля в классе. Выполнили эту фразу: «Объект/класс обладает переменными, которые характеризуют его состояние». Теперь обратим наши взоры на вторую часть: «Объект/класс умеет что-то делать».

Для наделения объектов умением что-то делать существует механизм объявления методов класса. Метод можно определить как процедура/функция, которая определена в классе. Например мы можем определить метод, который «передвигает» нашего робота на какое-то количество метров вперед в соответствии с его курсом. В упрощенном варианте — меняет его координаты в соответствии с заданной дистанцией.

Если вы постараетесь вспомнить школьный курс тригонометрии, то для вас не составит труда понять эти формулы:

```

1 x = x + distance * cos(course);

```

```
2 | y = y + distance * sin(course);
```

Поясним: координата X будет увеличиваться на дистанцию (distance), умноженную на косинус угла курса (course). Координата Y — дистанция на синус угла курса. Надеюсь, что это не вызывает больших вопросов. А теперь давайте сразу возьмем быка за рога — посмотрим код, который передвигает нашего робота на нужную дистанцию.

```
1 | public class Robot
2 | {
3 |
4 |     // Текущая координата X
5 |     double x = 0;
6 |     // Текущая координата Y
7 |     double y = 0;
8 |     // Текущий курс (в градусах)
9 |     double course = 0;
10 |
11 |     // Печать координат робота
12 |     void printCoordinates() {
13 |         System.out.println(x + "," + y);
14 |     }
15 |
16 |     // Передвижение на дистанцию distance
17 |     // учтите, что угол надо передавать в радианах. Для того, что бы работать в градусах
18 |     // мы преобразуем градусы в радианы путем деления на 180 и умножением на PI (Пи - 3,14159)
19 |     void forward(int distance) {
20 |         // Обращение к полю объекта X
21 |         x = x + distance * Math.cos(course / 180 * Math.PI);
22 |         // Обращение к полю объекта Y
23 |         y = y + distance * Math.sin(course / 180 * Math.PI);
24 |     }
25 | }
```

Давайте потихонечку разбираться в нашем коде. Как можно видеть мы изменили описание нашего робота. Теперь у него появились МЕТОДЫ. До этого мы с вами рассматривали только поля.

Метод — это действие, которое может производить объект. Вы можете найти и другие определения, но мне это нравится больше всего. Форма записи метода в очень простом виде выглядит вот так:

```
1 | тип_возвращаемого_значения имя(<аргументы>) {
```

```
2 | операция;  
3 | операция;  
4 | операция;  
5 | ...  
6 | }
```

Наш робот теперь умеет делать следующее:

1. Печатать текущие координаты — `printCoordinates`
2. «Ехать» вперед на некоторое количество метров и рассчитывать новые координаты с учетом курса — `forward(int distance)`

Сначала рассмотрим печать координат — `void printCoordinates()`.

Зарезервированное слово «void» означает, что метод ничего не возвращает. Мы позже посмотрим более подробно этот вопрос. А пока просто примем этот факт — если метод не должен ничего возвращать, то он предваряется словом «void». После типа идет имя метода. Синтаксис имени подобен синтаксису имени переменной. Лучше называть методы так, чтобы было понятно, что он делает.

Теперь обратим внимание на внутреннюю часть метода `printCoordinates`. Внутри метода мы выполняем вывод данных — `System.out.println(x + «,» + y);`. Но на что еще надо обратить внимание — мы обращаемся к переменным `x` и `y`. Причем в самом методе мы эти переменные не объявляли. Думаю, что многие из вас догадались — это поля, которые объявлены внутри класса. Т.е. внутри метода мы можем обратиться к полям. Причем обращаемся мы к полям того объекта, у которого вызываем метод. Сам вызов метода делается так же, как и обращение к полю — через оператор «.».

Теперь рассмотрим метод `forward(int distance)`. В отличие от метода `printCoordinates` в методе `forward` мы передаем данные — параметр `int distance`. В теле метода мы опять же обращаемся к полям объекта и делаем вычисления новых координат. И используем переданный параметр по имени.

Наш робот конечно еще далек от совершенства, но тем не менее его уже можно использовать. Можно создать объект (отметим, что при создании его координаты устанавливаются в 0), попросить его проехать вперед и напечатать его текущие координаты. Пришла пора пообщаться с нашим роботом. Смотрим код:

```
1 | public class RobotManager  
2 | {  
3 |  
4 |     public static void main(String[] args) {  
5 |         // Создаем объекта класса Robot  
6 |         Robot robot = new Robot();  
7 |  
8 |         // Вперед на 20 метров  
9 |         robot.forward(20);  
10 |        // Напечатать координаты  
11 |        robot.printCoordinates();  
12 |    }
```

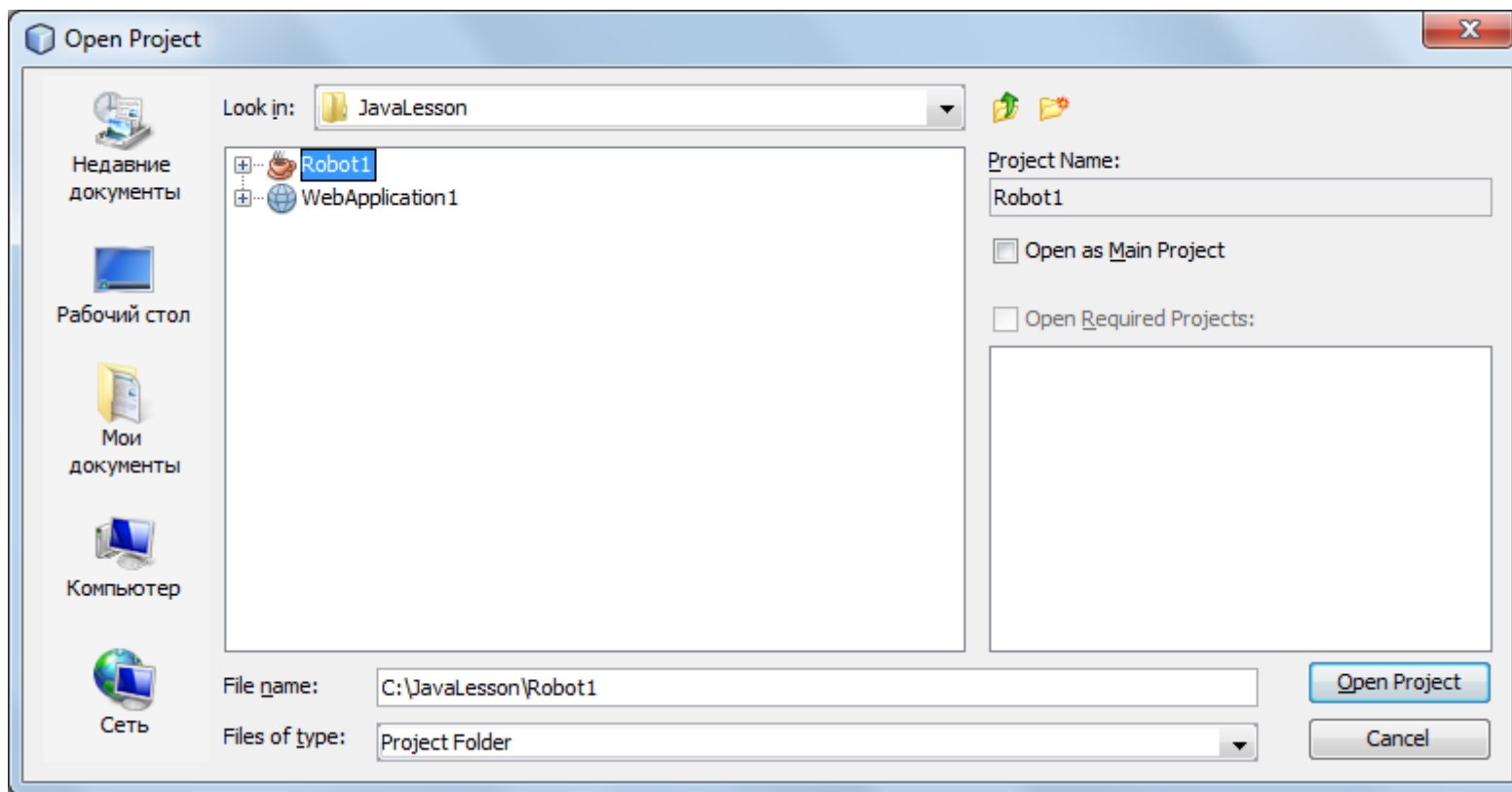
```
13 // Курс 90 градусов - не самый правильный способ
14 // но давайте пока остановимся на нем
15 robot.course = 90;
16 // Вперед на 20 метров
17 robot.forward(20);
18 // Напечатать координаты
19 robot.printCoordinates();
20
21 // Курс 45 градусов
22 robot.course = 45;
23 // Вперед на 20 метров
24 robot.forward(20);
25 // Напечатать координаты
26 robot.printCoordinates();
27 }
28 }
```

Думаю, что вы уже видите, что мы из одного класса — `RobotManager` — управляем созданием и поведением объекта класса `Robot`. Просто внимательно смотрите код и читайте комментарии. Если мы запустим наш проект, то увидим вот такие строки

```
1 20.0,0.0
2 20.0,20.0
3 34.14213562373095,34.14213562373095
```

Если вы внимательно смотрели код, то думаю, что для вас результат не является неожиданностью.

Я выложил архив нашего первого примера здесь — [Robot1](#). Вы можете его скачать и распаковать в директорию `C:\JavaLessons`. Для того, чтобы открыть проект в `NetBeans` вам надо выбрать пункт меню `File -> Open Project`. Посмотрите, что это также можно сделать с помощью комбинации клавиш `Ctrl+Shift+O`. В появившемся окне выберите папку `Robot1` и нажмите «Open Project».



Мы с вами посмотрели как создавать класс, как создавать объект. Заодно освоили (я надеюсь, что вы вместе со мной сделали это) как это делать в NetBeans. Наш робот весьма простой и далек от совершенства — в нем нет еще многих элементов ООП, но тем не менее это уже рабочий код — с чем вас и поздравляю.

## Ссылки на объекты

Возможно, что кому-то сразу стала очевидна природа ссылок на объекты. Но судя по вопросам, которые я регулярно слышал, слышу и скорее всего буду слышать, далеко не всем это сразу становится понятно и очевидно. Для тех, кто не понял и написана эта часть.

**ВАЖНО !!!** Если вы не поняли природу ссылок — это нормально, не думайте, что вы безнадежны. Что гораздо важнее — в этом вопросе необходимо обязательно разобраться.

Итак, мы с вами чуть выше говорили, что описание переменной автоматически НЕ СОЗДАЕТ объект нужного нам типа. Для создания объекта вам надо это сделать явно. Еще раз повторим это утверждение на примере:

---

```
1 | Robot r1;           // Здесь r1 еще никуда не указывает
2 | r1 = new Robot();   // В памяти (в "куче" - heap) выделяется память, в которой
3 |                     // размещается только что созданный объект
```

Здесь важно отметить, что переменная **r1** НЕ ЯВЛЯЕТСЯ объектом. Она является ССЫЛКОЙ на объект, который располагается в “куче” (heap — очень часто говорят — “находится в хипе”).

Проводя бытовую аналогию, переменная это как бумажная форма для записи адреса. Она может быть пустой (незаполненной) — тогда мы говорим, что переменная не инициализирована. Для локальной переменной компилятор будет просто напросто ругаться, если вы ей ничего не присвоите. Вы можете написать так:

```
1 | Robot r1;
2 | System.out.println(r1);
```

И этот код не будет даже компилироваться. Если же вы напишите так, то компилироваться будет.

```
1 | Robot r1 = null;
2 | System.out.println(r1);
```

Если же вы ее заполнили (создали объект и присвоили его переменной), то форма теперь УКАЗЫВАЕТ на адрес, ССЫЛАЕТСЯ на него. Вот почему переменная типа класса является ССЫЛКОЙ. Повторим еще раз пример создания объекта

```
1 | Robot r1 = new Robot();
2 | System.out.println(r1);
```

Вот только теперь ссылка **r1** ссылается на объект типа **Robot**.

Посмотрим еще один пример кода

```
1 | Robot r1 = new Robot();
2 | Robot r2 = r1;
```

А теперь попробуйте ответить на следующий вопрос: “На какой объект указывает переменная `r2` ?”

Если вы предположили, что обе переменные указывают НА ОДИН и ТОТ ЖЕ объект — то вы совершенно правы.

Теперь важно “увидеть” эту ситуацию — у вас ДВЕ ссылки, но только ОДИН объект. По сути у вас две формы, которые указывают на один и тот же адрес.

Фактически мы можем работать с одним и тем же объектом через ДВЕ ссылки — **`r1`** и **`r2`**. Через ОБЕ.

Теперь давайте рассмотрим такой код:

```
1  Robot r1 = new Robot();
2  Robot r2 = r1;
3
4  r2.x = 888;
5
6  System.out.println(r1.x);
```

Посмотрите внимательно на две последние строки. Мы поменяли поле `x` у объекта через переменную-ссылку **`r2`**, а печатаем поле `x` через переменную-ссылку **`r1`**. Вы уже скорее всего догадались, что на экране мы увидим **888**.

Вы точно поняли, почему это происходит ? Если снова не поняли, то проговорите про себя еще раз — ОБЕ ССЫЛКИ УКАЗЫВАЮТ НА ОДИН И ТОТ ЖЕ ОБЪЕКТ.

Чтобы я еще хотел отметить — сам объект через присваивание НЕ КОПИРУЕТСЯ, но ссылки КОПИРУЮТСЯ. Т.е. у нас ДВЕ ссылки. Они являются копиями друг друга. Их даже можно сравнить — и если они указывают на один и тот же объект — они равны.

С одной стороны, понимание ссылок не является каким-то сложным построением. С другой стороны — если вы не понимаете их природу, то вам будет непросто двигаться дальше. Так что постарайтесь четко осознать эту тему.

И вот теперь высказав все необходимые на мой взгляд идеи, которые стоят за ООП, я собираюсь переходить к тому, что называется парадигмами ООП. Не сомневаюсь, что некоторые из вас слышали о них. Но мне бы хотелось донести до вас истоки появления этих парадигм — они появились не на пустом месте — именно этому были посвящены несколько предыдущих абзацев.

Теперь мы можем перейти к следующей части нашего повествования: [Инкапсуляция](#).

## 29 comments to *ООП*



• Апрель 1, 2015 at 22:07  
*javaNoob* says:



Спасибо за доходчивое разъяснение.

Ни для кого не секрет, что не смотря на всю простоту(в силу логичности) ООП парадигма не сразу осваивается.Хоть сто книжек на эту тему прочитай...А тут буквально парой Ваших предложений(своего виденья) картинка стала проясняться :). С удовольствием читаю Ваши статьи. И конечно буду задавать вопросы:)...

[Reply](#)



• Апрель 30, 2015 at 09:14

*Дмитрий* says:

Всем добрый день!

Уважаемые администратор и ученики прошу Вас помочь с моим роботом, точнее с тем, что я с ним сделал. В качестве эксперимента я добавил в метод printCoordinates цикл for вот так:

// Печать координат робота

```
void printCoordinates() {  
    System.out.println(x + «,» + y);  
    for(double i=0; i<=x; i++){  
        for(double k=0; k<=y; k++){  
            System.out.print("*");  
        }  
        System.out.println();  
    }  
}
```

И я никак не могу понять, почему у меня не получается треугольник???

Всем заранее спасибо!

[Reply](#)



o

Июль 7, 2015 at 06:58

*Grif* says:

В указанном примере скорее всего получится прямоугольник со сторонами У\*Х, потому что не соблюден принцип построения треугольников (например — первая строка = 1 символ, каждая последующая строка длинее предыдущей).

Поздний конечно комментарий, но вдруг кому-то пригодится.

У меня всё получается замечательно, правда я использую другую среду разработки — IntelliJ IDEA 14.1.3

[Reply](#)



Ноябрь 1, 2015 at 13:19

[inefoy](#) says:

Он не сложный

[Reply](#)



Ноябрь 3, 2015 at 09:55

*admin* says:

Но он требует понимание ООП, которого к этому этапу пока нет.

[Reply](#)



Апрель 11, 2016 at 13:58

*Владимир* says:

Ребята у меня вот так получается

20.0,0.0

0.0,3.9163938347251766E-14

1.9581969173625883E-14,20.000000000000004

Хотя делал все по уроку. Посоветуйте где не так.

[Reply](#)



Апрель 12, 2016 at 08:32

*Владимир* says:

Разобрался

[Reply](#)



Август 27, 2016 at 21:25

*Alexander* says:

Здравствуйте! Ссылка на «Введение в программирование на языке Java» не работает.

[Reply](#)



o

Август 29, 2016 at 09:47

*admin* says:

Видимо устарела — исправил на другую.

[Reply](#)



Сентябрь 25, 2016 at 16:49

*sl* says:

добрый день!

попытался написать самостоятельно аналогичный код и сразу возник вопрос над которым долго мучался.

в метод `void forward(int distance)` передаются несколько переменных. при этом значения переменных присваиваются разными способами: `robot.forward(20)` и `robot.course = 45`.

правильно ли я понимаю,

1) что выражения со скобками, в частности `robot.forward(20)`, всегда вызывают методы класса???

2) а переменные внутри скобок указываются, если они меняются очень часто при выполнении программы. а те переменные, которые

меняются редко или не достаточно часто, объявляются стандартным способом `robot.course = 90`? (ведь можно было вызвать метод класса `robot.forward(20, значение course)`)?

[Reply.](#)



o

Октябрь 3, 2016 at 18:00

*admin* says:

Вы похоже не совсем правильно поняли.

[Reply.](#)



o

Октябрь 23, 2016 at 04:31

*StickOfFaith* says:

Эти 2 способа работы с полями класса никак не связаны с частотой их использования. Почему автор сделал именно так — не знаю. Почитайте про инкапсуляцию и найдете ответ на первый вопрос. А ответ на второй гораздо проще. Использовать `robot.forward(20, значение course)` можно.

[Reply.](#)



•

Февраль 21, 2017 at 20:57

*Юрий* says:

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

Почему с помощью этих формул не получается вернуть робота обратно в центр?

По идее, он у него должен быть курс  $90^\circ$ , потом  $180^\circ$ , потом  $270^\circ$  и он возвращается в начальные координаты  $x=0$  и  $y=0$ .

Я в каких-то других координатах живу? Помогите разобраться.

[Reply.](#)



o

Февраль 21, 2017 at 21:10

*Юрий* says:

Формулы неверные. Они в принципе не дают возможности получить отрицательное значение. А отрицательные значения должны быть, чтобы координаты уменьшались. Жаль я не математик, не могу правильные формулы вот так навскидку написать)

[Reply.](#)



■

Февраль 22, 2017 at 00:03

*admin* says:

Я не вижу Вашего кода. Формулы верные — математика там еще школьная.

```
double distance = 10.0;
double course = 0.0;
double x = 0.0;
double y = 0.0;
for (int i = 0; i < 4; i++) { x = x + distance * Math.cos(course / 180 * Math.PI); y = y + distance * Math.sin(course / 180 * Math.PI); System.out.println("X:" + x + ", Y=" + y); course += 90; }
```

Результатом выполнения этого кода получается следующее

X:10.0, Y=0.0

X:10.0, Y=10.0

X:0.0, Y=10.0000000000000002

X:-1.8369701987210296E-15, Y=1.7763568394002505E-15

Последняя строка имеет числа в минус 15-й степени - это практически нулевое значение.

[Reply.](#)



■

Февраль 22, 2017 at 06:49

*Юрий* says:

Спасибо за ответ. Я был не прав, все ок с формулами. Математика школьная, в этом вся и проблема, в школе я даже на выпускном выбрал сдавать геометрию, любил ее. Но это было 22 года назад и все это время я ей не пользовался)) надо вспоминать!

Подскажите, как округлить ответ до сотых?

Нашел, что бывает printf, но в данном случае у меня не работает. Ошибку выдает. Я мешаю мух с котлетами?

```
public class Robot {  
  
    double x = 0;  
    double y = 0;  
    double course = 0;  
  
    void printCoordinates() {  
        System.out.printf(«X = » + » %.2f\n»,x + «, » + «Y = » + y);  
    }  
  
    void forward(int distance) {  
  
        x = x + distance * Math.cos(course / 180 * Math.PI);  
  
        y = y + distance * Math.sin(course / 180 * Math.PI);  
  
    }  
  
}
```

[Reply](#)



■ Февраль 22, 2017 at 09:46

*admin* says:

Такой код работает нормально.

```
System.out.printf(«x=%.2f, y=%.2f\n\r», x, y);
```

Общее замечание — когда пишете, что выдает ошибку, лучше указывать какую именно.

А то это равносильно ставить диагноз по фразе «Доктор, у меня голова болит, чем я болен ?».



Март 4, 2017 at 18:23

*Виталий* says:

пол года уже пытаюсь изучить ООП натываясь на подобные статьи (((

[Reply](#)



o

Март 5, 2017 at 18:20

*admin* says:

Ну видимо Вам надо что-то другое — другие слова, другие образы. Книги и статьи все-таки «одновариантный» вариант. Может надо начать не с ООП а просто с умения писать простые вычислительные программы.

[Reply](#)



Сентябрь 22, 2017 at 16:02

*Алексей* says:

Здравствуй,подскажите пожалуйста почему при попытке переписать выражение  
 $x = x + \text{distance} * \text{Math.cos}(\text{course} / 180 * \text{Math.PI});$   
на альтернативное со специальным оператором  
 $x+ = \text{distance} * \text{Math.cos}(\text{course} / 180 * \text{Math.PI});$   
IDE выдает ошибку «illegal start of expression» ?

[Reply](#)



o

Сентябрь 30, 2017 at 10:32

*admin* says:

Нельзя ставить пробелы между + и = — это должно быть вместе — +=

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

[Reply.](#)



Октябрь 1, 2017 at 19:33

*Алексей* says:

Спасибо 😊

[Reply.](#)



Декабрь 24, 2017 at 19:59

*Михаил* says:

А зачем в аргументе синуса и косинуса значение угла делится на  $180 \cdot \pi$ ? Причем, в первоначально приведенных формулах этого нет, что правильно, насколько я помню... И почему когда я оставляю просто угол, программа работает, но выдает другие значения? Причем, при угле в 45 градусов координаты X и Y не равны между собой, как должны быть?

[Reply.](#)



Декабрь 24, 2017 at 21:09

*admin* says:

Угол для расчета тригонометрических функций должен задаваться в радианах, а не в градусах. Для обычного человека это неудобно. Для перевода в радианы надо угол в градусах умножить на  $\pi$  и разделить на 180.

[Reply.](#)





Февраль 10, 2018 at 08:05

*Alexandr* says:

помогите разобраться как это работает

мы в классе RobotManager вписываем значение аргумента для метода robot.forward(20);

В классе Robot получается

```
void forward(int distance) {  
  x = x + distance * Math.cos(course / 180 * Math.PI);  
  y = y + distance * Math.sin(course / 180 * Math.PI);  
}
```

что должно соответствовать

```
void forward(int 20) {  
  x = 0 + 20 * Math.cos(0/ 180 * Math.PI); // = 0  
  y = 0 + 20 * Math.sin(0/ 180 * Math.PI); // = 0  
}
```

как получается что  $x = 20$  а  $y = 0$ ?

[Reply.](#)



o

Февраль 10, 2018 at 10:17

*admin* says:

Это школьный курс тригонометрии — синус и косинус. Когда угол равен 0, то синус угла равен 0, а косинус равен 1.

Соответственно  $\text{Math.sin}(0)$  — это форма записи вызова тригонометрической функции синус от 0. Она дает 0.

$\text{Math.cos}(0)$  дает 1. Дальше простая арифметика — умножение на 0 или 1.

Если интересно разобраться с синусами и косинусами — я тут покопался в интернете и удивился. Почему про синус и косинус рассказывают на примере прямоугольного треугольника (в основном) ? Это по-моему крайне не наглядно. Надо наверно статью написать про это 😊

[Reply.](#)



• Апрель 28, 2018 at 19:31

*Dreamwalker* says:

МОжет кому-то будет интересно проверить свои знания после прочтения статьи) я всегда стараюсь как-то проверить усвоенную инфу вот тут есть тест по оор java <https://proghub.ru/t/oop-java>

[Reply](#)



• Июль 19, 2018 at 22:52

*Galina* says:

Почему в методе forward(int distance) указан тип данных int?  
Причем расчеты с переменными (x, y, course) производятся в формате double.  
Когда допустимо смешивать эти типы данных?

[Reply](#)



○

Июль 21, 2018 at 04:57

*admin* says:

1. Взял целое, чтобы нельзя было передвинуться на вещественное число. Так сложилось. Особых мыслей по этому поводу не имел тогда
2. Числа допустимо всегда, но есть особенности. При операции компилятор приводит все числа к самому «большому» типу — т.е. если сложить целое и вещественное, то результат — вещественное. Если делить целое на целое, получим целое. Если к double прибавлять float — получим double.

[Reply](#)

**Leave a reply**


Comment

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

Имя \*

E-mail \*

Сайт

семь — пять =  

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](#)

