



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](#)

Отношения между классами (объектами)

По моей практике преподавания вопрос отношений между классами (или объектами) почему-то вызывает проблемы. Из-за чего так происходит, не могу сказать. Когда я изучал ООП, мне это показалось настолько очевидным, что не стоило даже упоминания — вся логика ООП не просто кричит, она вопиет об очевидности таких отношений и правилах их построения. Но тем не менее этот вопрос часто требует объяснений. Чем мы сейчас и займемся, прежде чем продолжим изучение новых конструкций языка Java.

Теория ООП выделяет три основных отношения между классами:

1. Ассоциация
2. Агрегация и композиция
3. Обобщение/Расширение (наследование)

Последний пункт мы с вами уже рассматривали, так что сосредоточимся на первых двух.

Ассоциация

- [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](#)

Ассоциация означает, что объекты двух классов могут ссылаться один на другой, иметь некоторую связь между друг другом. Например Менеджер может выписать Счет. Соответственно возникает ассоциация между Менеджером и Счетом. Еще пример — Преподаватель и Студент — т.е. какой-то Студент учится у какого-то Преподавателя. Ассоциация и есть описание связи между двумя объектами. Студент учится у Преподавателя. Идея достаточно простая — два объекта могут быть связаны между собой и это надо как-то описать.

Агрегация и композиция

Агрегация и композиция на самом деле являются частными случаями ассоциации. Это более конкретизированные отношения между объектами.

Агрегация — отношение когда один объект является частью другого. Например Студент входит в Группу любителей физики.

Композиция — еще более «жесткое отношение, когда объект не только является частью другого объекта, но и вообще не может принадлежат еще кому-то. Например Машина и Двигатель. Хотя двигатель может быть и без машины, но он вряд ли сможет быть в двух или трех машинах одновременно. В отличии от студента, который может входить и в другие группы тоже. Такие описания всегда несколько условны, но тем не менее.

Техническая реализация

На самом деле каких-либо сложных и высокоумных технических решений нет — все достаточно тривиально. В одном классе делается ссылка на другой и наоборот (не всегда). Далее идет развитие данной идеи в зависимости например от количества связей. В машине четыре колеса и она имеет связь со всеми — значит в машине будет ссылка на список колес (или массив). Давайте соединим нашего робота с оператором, который им управляет. Между ними можно установить ассоциацию через ссылки в одном классе на другой класс. Т.е. класс Robot имеет ссылку на класс Operator и наоборот — класс Operator имеет ссылку на класс Robot.

Класс Robot

```

1 package edu.javacourse.robot;
2
3 public class Robot
4 {
5     private double x = 0;
6     private double y = 0;
7     protected double course = 0;
8     // Робот управляется оператором
9     private Operator operator;
10

```

```
11 public Robot(double x, double y) {
12     this.x = x;
13     this.y = y;
14 }
15
16 // Можно узнать какой оператор управляет роботом
17 public Operator getOperator() {
18     return operator;
19 }
20
21 // Можно установить оператора для робота
22 public void setOperator(Operator operator) {
23     this.operator = operator;
24 }
25
26 public void forward(int distance) {
27     x = x + distance * Math.cos(course / 180 * Math.PI);
28     y = y + distance * Math.sin(course / 180 * Math.PI);
29 }
30
31 public double getX() {
32     return x;
33 }
34
35 public double getY() {
36     return y;
37 }
38
39 public double getCourse() {
40     return course;
41 }
42
43 public void setCourse(double course) {
44     this.course = course;
45 }
46
47 public void printCoordinates() {
48     System.out.println(x + "," + y);
49 }
50 }
```

Класс Operator

```
1 package edu.javacourse.robot;
```

```

2
3 public class Operator
4 {
5     private String firstName;
6     private String lastName;
7     // Оператор управляет конкретным роботом
8     private Robot robot;
9
10    public Operator(String firstName, String lastName) {
11        this.firstName = firstName;
12        this.lastName = lastName;
13    }
14
15    public String getFirstName() {
16        return firstName;
17    }
18
19    public String getLastName() {
20        return lastName;
21    }
22
23    // У оператора можно спросить каким роботом он управляет
24    public Robot getRobot() {
25        return robot;
26    }
27
28    // Оператору можно поручить управлять роботом
29    public void setRobot(Robot robot) {
30        this.robot = robot;
31    }
32 }

```

Чего-то другого человечество пока не придумало. Как я уже упоминал, можно несколько разнообразить виды ссылок, но идея останется прежней. Можно делать ссылки только в одном классе — тогда связь будет односторонняя, может быть несколько ссылок или ссылка на список (массив). Но все равно сказать одному классу, что он связан с другим можно только по такому принципу. Кстати, когда мы размещали наши компоненты с овалом, прямоугольником на форме, то тут можно говорить о композиции — форма содержит внутри себя список компонентов класса `JComponent` для хранения всех компонентов, которые ей передают.

Еще один способ взаимодействия объектов

Эта ситуация возникает тогда, когда постоянной связи между объектами не предусматривается вообще, но какие-то данные надо передать от одного другому. Можно воспользоваться вариантом передачи объекта прямо в методе. Например, наш робот может иметь метод, который принимает данные из другого робота — его координаты — для того, чтобы туда переместиться. Значит надо сделать метод, который в качестве параметра будет иметь

объект класса Robot.

Как видите все достаточно банально и просто. Мы очень скоро столкнемся с этими понятиями при рассмотрении примеров.

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

14 comments to *Отношения между классами*



• Февраль 7, 2016 at 13:47

Алекс says:

Здравствуйтесь, спасибо за интересный сайт. Могли бы вы подробнее прокомментировать строку 9 класса Robot

```
private Operator operator;
```

и строки 5-8 класса Oprerator

```
private String firstName;
```

```
private String lastName;
```

```
// Оператор управляет конкретным роботом
```

```
private Robot robot;
```

— т.е. что это за конструкция? Это и не создание нового объекта, и не конструктор. Похоже только на создание ссылки на объект, но могли бы вы объяснить как это работает?

Также похожие конструкции есть в следующей статье

```
private Object object;
```

```
private ObjectBox next;
```

Заранее благодарен.

[Reply](#)



• Февраль 7, 2016 at 14:41

Алекс says:

Извините за поспешный вопрос. Насколько я правильно разобрался, это просто поля класса, ссылки на потенциальные (еще не существующие) объекты. Их следует объявить для последующего прописывания методов данного класса (эти поля используются в качестве атрибутов метода).

[Reply.](#)



Февраль 25, 2017 at 14:52

Юрий says:

Вот эта вот глава совсем непонятная.

```
// Робот управляется оператором
private Operator operator;
// Оператор управляет конкретным роботом
private Robot robot;
```

абсолютно одинаковые команды, почему робот УПРАВЛЯЕТСЯ, а оператор УПРАВЛЯЕТ? Почему не наоборот? Вообще не понял этих примеров.

Вот этот блок что делает?

```
public Operator(String firstName, String lastName) {
this.firstName = firstName;
this.lastName = lastName;
```

```
// У оператора можно спросить каким роботом он управляет
```

```
public Robot getRobot() {
return robot;
```

каким образом мы у него спрашиваем? Запускаем программу ничего не меняется.

Как настроить свой мозг на понимание всего вот этого?

[Reply.](#)



o

Февраль 25, 2017 at 16:18

admin says:

А в жизни как должно быть ? Робот будет управлять оператором или все-таки оператор в ответе за работа и именно он посылает ему команды ?

Но для того, чтобы посылать команды надо иметь СВЯЗЬ с роботом. А как это сделать не имея АССОЦИИ — связи между объектами ? Связь может быть в виде ссылки на нужный объект. А вот какую мысль Вы вкладываете в эту связь — это уже определяется требованиями к задаче.

Ваши вопросы говорят о том, что ООП для Вас пока очень слабо понятная область — дерзайте дальше 😊

По поводу «настроить голову» — читать, пробовать, ошибаться, задавать вопросы и искать ответы. Книги, форумы, статьи, примеры. Если хотите это делать с большей эффективностью — занимайтесь с хорошим тренером/преподавателем.

[Reply](#)



• Апрель 21, 2017 at 03:08

Andrey says:

Не совсем понимаю, в примере с роботом и оператором, это композиция или агрегация? Ведь жизненный цикл поля «private Robot robot» в классе «Operator» ограничен самим классом «Operator» и при разрушении объекта класса «Operator» поле «private Robot robot» тоже разрушается?

[Reply](#)



○ Апрель 21, 2017 at 09:58

admin says:

На мой взгляд это больше агрегация — хотя здесь много субъективного.

Что касается «разрушения» — это же просто ссылка на существующий объект. У робота есть ссылка на оператора. Если надо «отключить» оператора от робота, то надо эту ссылку обнулить.

При учете «сборщика мусора» объект в Java не уничтожается до тех пор, пока на него есть ссылки. Если таких ссылок нет — объект будет утилизирован.

[Reply](#)



Май 20, 2017 at 12:50

Formfaktor says:

Здравствуйте.

Агрегация — это Робот и Оператор.

Композиция — это Робот и, например, Батарея питания.

Т.е. при агрегации передаётся ссылка на уже созданный объект (через метод или конструктор)? А при композиции — ссылка на объект инициализируется при описании класса (в конструкторе или сразу при объявлении переменной экземпляра)?

Я правильно понял?

[Reply](#)



o

Май 22, 2017 at 13:28

admin says:

В общем — правильно. И про конструктор тоже в принципе верное замечание. Хотя такое не всегда делается через конструктор — тут грань достаточно тонкая.

[Reply](#)



■

Май 22, 2017 at 14:10

Formfaktor says:

Спасибо большое. Мня тема отношений между классами тяжеловато даётся. Многие авторы ее по разному описывают ((. И это сбивает с толку.

[Reply](#)



■

Май 22, 2017 at 14:56

admin says:

Здесь важно видеть, что объекты между собой могут иметь связь. Какую — это уже можно пытаться классифицировать, но немало зависит от контекста.

Например, когда автобус едет по шоссе — связь водитель-автобус весьма жесткая. А когда автобус стоит в парке или на ремонте, то водителя там нет и он в общем не нужен. Поэтому каждый раз надо думать о задаче, а связи между объектами — это уже вторично.

[Reply](#)



Октябрь 25, 2017 at 17:43

MyBag says:

Ребята, я около года пытался понять эту логику и наконец понял.

Смотрите, когда автор говорит, что робот управляется оператором или оператор управляет роботом, то он имеет ввиду, что, как бы, в будущем он хочет реализовать такие возможности, а пока что он просто банально передаёт ссылку от одного объекта другому. Пока что эта ссылка почти ничего не значит — просто ссылка, но потом, автор, возможно(если захочет и не передумает), реализует доп. методы, с помощью которых он будет реализовывать какие-то действия, допустим, напишет у робота метод, посылающий сообщение оператору, типа `sendMessageToOperator(String msg, Operator operator)` или просто там типа `sendMsgOperator`. А у него(у объекта робота) уже есть ссылка на конкретный объект конкретного оператора. А оператор в свою очередь, допустим делает команду `stopMove` конкретному роботу.

Но автор сейчас хочет показать какие-то действия на более абстрактном уровне, хочет показать принцип агрегирования и композиции, и, дабы уменьшить количество кода в примере — просто передаёт ссылку и говорит, что робот «как бы» управляется оператором.

[Reply](#)



Февраль 17, 2018 at 17:34

Юрий says:

```
...
this.robot = robot;
...
this.operator = operator;
```

Java оператором «=» делает копию объекта или копию указателя ?

впрочем вопрос риторический, дальше станет понятно.

да и из логики,... зачем бы нам понадобилась копия объекта внутри другого объекта? Хотя почему бы и нет, только это будут не отношения а хранение. Тогда вопрос а как сделать именно копию? Впрочем и до этого дойдёт наверное.

[Reply.](#)



o

Февраль 17, 2018 at 22:13

admin says:

В Java работа идет со ссылками — обе ссылки будут указывать на один и тот же объект. Для создания копии нужно использовать метод clone, который тоже имеет свои особенности — его надо сделать для своего класса. Сейчас я об этом не писал — надо будет делать полную ревизию статей.

[Reply.](#)



•

Июль 25, 2018 at 08:16

v says:

1. Не хватает main метода (некоторым читателям непонятно почему программа не выводит□, набросал, возможно топорно, но так более понятно):

```
1 public class RobotProgram {
2     public static void main(String[] args) {
3         //создаем объект Оператор с последующим созданием объекта Робот
4         Operator operator = new Operator("John", "Smith");
5         operator.setRobotExt(new RobotExt(5, 5, 0));
6         System.out.println(operator.getFirstName() + " " + operator.getLastName());
7         operator.getRobotExt().printCoordinates();
8         operator.getRobotExt().forward(10);
9         operator.getRobotExt().printCoordinates();
10        operator.getRobotExt().setCourse(90);
11        operator.getRobotExt().forward(10);
12        operator.getRobotExt().printCoordinates();
13        //создаем объект Робот с последующим созданием объекта Оператор
14        RobotExt robotExt = new RobotExt();
```

```
15 |         robotExt.setOperator(new Operator("Jim", "Freedom"));
16 |         System.out.println(robotExt.getOperator().getFirstName() + " " + robotExt.getOperator().getLastName());
17 |     }
```

2. Насколько я понимаю отношения между классами бывают «имеет» (композиция, например, машина имеет двигатель) и «является» (наследование, например, треугольник является фигурой). Также композиция может быть реализована делегированием. т.е. методы робота можно по ссылке делегировать в Операторе.

```
1 | private RobotExt robotExt;
2 |
3 | public void forward(int distance) {
4 |     robotExt.forward(distance);
5 | }
```

[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 - = два 

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](#)

