Парадигмы ООП

Для начала я их просто перечислю, а потом попробую описать подрообнее. Итак:

1. Инкапсуляция
2. Наследование
3. Полиморфизм

Теперь мы рассмотрим более подробно каждую парадигму.

Инкапсуляция

Не пытайтесь найти перевод этого термина с английского — вы получите encapsulation. Зато если вы попробуйте присмотреться, вы увидите главное — «капсула». Каждый объект не должен выставлять наружу все свои параметры для изменения просто так. Например, если у нашего робота есть координаты X и Y, то не вызывает сомнений факт, что их нельзя менять прямо. Робот должен поменять свои координаты в результате передвижения. Нельзя обратиться к переменной X внутри объекта Robot и сделать самое простое присваивание. Это будет как минимум нелогично. Лучше такого вообще не позволять. Т.е. мы таким образом должны создавать описание класса, чтобы нельзя было просто так

получать доступ к его внутренним переменным. Это будет похоже на то, как если бы мы при производстве телевизора давали людям доступ ко всей схеме и каждый мог переключать проводки внутри него напрямую. Обычный человек таким телевизором вряд ли бы пользовался. Конечно нашлось бы несколько энтузиастов, которые обрадовались такому положению дел. Но это скорее всего экзотика. Более правильно в случае с роботом было сделать так, чтобы можно было получить значения координат. А менять координаты можно было бы только в результате движения робота. Проехал он 10 метров — поменялись его координаты X и Y в соответствии с расстоянием и курсом. Причем автоматически. Т.е. закрытие внутренних переменных — хорошая идея. Для правильного поведения объекта. Нам самим будет проще — при условии, что класс более-менее разумно спроектирован и реализован. Но возникает резонный вопрос — а как тогда обращаться к внутренним переменным. В какой-то момент мы будем вынуждены это сделать. Об этом мы узнаем прямо сейчас. А заодно посмотрим на класс. Перед нами достаточно простое описание робота. Рассмотрим его внимательно.

```
1  public class Robot
2  {
3
4      // Текущая координата X
5      private double x = 0;
6      // Текущая координата Y
7      private double y = 0;
8      // Текущий курс (в градусах)
9      private double course = 0;
10
11     // Передвижение на дистанцию distance
12     public void forward(int distance) {
13         // Обращение к полю объекта X
14         x = x + distance * Math.cos(course / 180 * Math.PI);
15         // Обращение к полю объекта Y
16         y = y + distance * Math.sin(course / 180 * Math.PI);
17     }
18
19     // Печать координат робота
```

```
20      public void printCoordinates() {
21          System.out.println(x + "," + y);
22      }
23
24      public double getX() {
25          return x;
26      }
27
28      public double getY() {
29          return y;
30      }
31
32      public double getCourse() {
33          return course;
34      }
35
36      // Мы ррассмотрим этот метод несколько позже
37      public void setCourse(double course) {
38          this.course = course;
39      }
40  }
```

В этом примере есть несколько моментов, на которые надо обратить внимание.
Первый — возле объявления внутренних переменных появилось ключевое слово private. Это слово определяет видимость переменной (и не только — мы это увидим). Что значит «видимость»? По сути это означает в какой части кода программы можно будет обратиться к этой переменной.
Всего в Java существует три варианта таких слов:

1. private
2. protected
3. public

Слово «private» дает самую слабую видимость — только внутри описания/методов самого класса. Т.е. в методе forward переменные и course видимы. Но если вы попробуете обратиться к этим перменным так, как мы делали в предыдущей части из класса RobotManager, то ваша программа не скомпилируется. Вам будет указано, что у класса Robot переменная x (или y) — не доступна.
Слово «protected» позволяет видеть переменные другим классам, но не всем. Когда мы доберемся до понятие наследования и пакетов, мы рассмотрим «protected» еще раз. Пока дадим просто описание — переменные будут видны в классах-наследниках, и классах, находящихся в том же пакете. Может пока не очень понятно, но наберитесь терпения.
И наконец «public». Это описание позволяет обращаться к переменной/методу откуда угодно. Вы наверняка увидели public в коде класса Robot при объявлении методов.

Если вы никогда до этого не сталкивались с объектами и классами, то у вас может возникнуть вполне резонный вопрос «А зачем это надо — видимость какая-то?». Я попробую на него ответить.
Как мы уже говорили, было бы нелогично позволять «переносить» робота просто присваиванием новых координат — так настоящие роботы не передвигаются. Программа — это попытка отражения реальной задачи и приближение к реальности дает возможность упростить себе жизнь. Если робот «не позволяет» произвольно перемещать его куда угодно — это большое подспорье. Вы не сделаете ошибку в процессе программирования — вам это просто не позволят. Если вы думаете, что все это забавы лобастых дядек — попробуйте представить себе комнату, в которой провода с электричеством не имеют изоляции. Вообще. И теперь попробуйте представить, что вам надо сделать ремонт в этой комнате. Мне когда-то сразу стало понятно, насколько это важно.
Современные программы включают огромное количество строк кода и помнить все особенности, ограничения и прочие моменты реализации — весьма сложно. А учитывая, что большинство программ пишется командой программистов, которая меняется со временем, то «тайные знания» кода передать практически невозможно. Вы конечно можете предложить каждому вновь прибывшему программисту прочитать и понять весь код. Но вряд ли такое возможно.
Теперь если мы попробуем управлять нашим новым роботом программой, которая была написана раньше (а именно класс RobotManager), то мы не сможем ее скомпилировать. Предлагаю посмотреть код, который позволяет управлять нашим обновленным роботом.

```
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         // Это более корректный способ менять курс.
14         // Реализация внутри робота не сильно отличается, но
15         // мы в любой момент сможем сделать более продвинутую версию
16         // Но класс RobotManager об этом даже не узнает
17         robot.setCourse(90);
18         // Вперед на 20 метров
19         robot.forward(20);
20         // Напечатать координаты
21         robot.printCoordinates();
22
23         // Курс 45 градусов
24         robot.setCourse(45);
25         // Вперед на 20 метров
26         robot.forward(20);
27         // Напечатать координаты
28         robot.printCoordinates();
```

```
    }
}
```

Я рекомендую посмотреть код, в нем есть важные комментарии. Но я еще раз заострю ваше внимание на моменте ограничения доступа к переменным класса Robot. Теперь ни один класс не сможет «испортить» координаты робота — мы сможем воздействовать на них только через команды роботу передвинуться или повернуться. И больше никак. (Не буду тут категоричен — на самом деле можно и туда забраться, но это уже будет низкоуровневое програмирование, которое отличается от прикладного по своим идеям). Здесь очень важным моментом является понимание, что теперь мы в большей степени приблизились к более адекватному моделирования задачи — она теперь более «живая», более настоящая. И это служит хорошим подспорьем при создании программы.
В этом приближении к реальности и есть, на мой взгляд, главная цель инкапсуляции. Проектирование программы с помощью ООП сводится таким образом к проектированию и написания классов, которые умеют делать определенную работу. По сути ваша программа превращается в набор готовых инструментов — роботы, самолеты, счета, графические окна, кнопки и т.д. И каждый инструмент «прячет» реализацию своих функций, отображая другим классам только свои «публичные» возможности. Если инструменты уже есть, то остается только написать программу, которая правильно ими управляет. А если инструмент делает не совсем то, что нам надо ? В этом случае его надо переделать. Или создать новый. Для исправления некоторых функций класса можно воспользоваться еще одной парадигмой — наследованием — о которой мы поговорим в следующей части.

Передача параметров и слово this

Прежде чем мы начнем рассмотрение следующей парадигмы ООП — наследование — мне бы очень хотелось остановиться на весьма важном моменте — передача параметров в метод и из метода. Как не странно, многие начинающие программисты, хоть и действуя интуитивно, достаточно правильно пишут программы и общаются с параметрами, в реальности не очень хорошо себе представляют механизм передачи параметров. Тем не менее понимание этого вопроса достаточно нужная вещь.
Итак, шаг первый — формальные параметры и фактические. Формальный параметр — это параметр, который вы описываете внутри метода. А фактически — это то, что вы передаете при вызове метода. Предлагаю рассмотреть все на примере. Рассмотрим упрощенный класс Robot, у которого мы оставим только одно поле course и два метода — для чтения и записи. Эту пару методов часто называют сеттер/геттер (set/get). Их очень часто создают для работы с полями класса и практически все IDE имеют редакторские возможности для создания этой пары методов. Если пойти еще немного дальше, то могу сказать. что именование этих методов используется во многих технологиях Java. Но пока давайте сосредоточимся только на параметрах. Итак у нас есть класс Robot с одним полем.

```
public class Robot
{
    // Текущий курс (в градусах)
    private double course = 0;

    public double getCourse() {
        return course;
    }

```

```
    public void setCourse(double course) {
        this.course = course;
    }
}
```

Самое главное находится в методе setCourse. Только здесь мы объявили параметр при вызове метода. Параметр в скобках double course является формальным параметром. В этой переменной будет находится то число, которое мы будем передавать внутрь этого метода. А фактическим будет именно число. Иными словами — формальный параметр как ящик, в который мы можем положить какую-то величину (в данном случае вещественное число). А вот то, что мы положим в этот ящик и будет фактическим параметром. Думаю, что с этим разобрались. Так что давайте разбираться дальше.
Вы наверняка обратили внимание на выражение внутри нашего метода:

this.course = course

Объяснение это конструкции очень простое. Внутри нашего класса уже есть свойство course (private double course). И мы хотим именно свойству присвоить значение, которое находится в формальном параметре course. Но простое написание

course = course

привело бы нас к бесполезной операции — мы присвоили бы переменной (формальному параметру) course ее же значение. Наше цель — присвоить значение из локальной переменной (фактического параметра) в свойство объекта. И именно для этого используется ключевое слово this. Это просто ссылка на сам объект внутри его метода. В каждом методе класса можно обратиться к объекту, для которого вызывается метод. (Из этого утверждения есть исключение, но мы его рассмотрим позднее). Резонный вопрос: «А если бы формальный параметр имел бы другео имя. Например не course, а localCourse?». Очень просто — словоthisможно было бы не использовать и наш метод выглядел бы вот так

```
    public void setCourse(double localCourse) {
        course = localCourse;
    }
```

Но т.к. большинство IDE умеет генерировать такие методы, то они создаются так, как было показано ранее. Если вам не нравится — создавайте сами и не используйте this. Это дело вкуса. Однозначного мнения нет. Для облегчения жизни разработчикам IDE часто подсвечивают свойства класса и можно сразу увидеть, что переменная является не локальной переменной или формальным параметром, а именно свойством.

И теперь следующий шаг — **параметры в методы передаются путем копирования**. В случае передачи элементарного типа это значит, что создается копия числа и работа внутри метода идет с копией. Т.е. если мы объявили переменную внутри какого-то метода, присвоили ей число 99, а затем передали ее в другой метод, то внутри этого метода будет другая переменная (наш формальный параметр), значение которой станет 99. Но если этой переменной внутри метода присвоить число 11, то мы поменяем значение только этой переменной. Рассмотрите пример

```java
public class TestVariable
{
    public static void main(String[] args) {
        double first = 99;
        // Создаем экземпляр класса
        TestVariable tv = new TestVariable();
        System.out.println("Main method:" + first);      // Здесь мы увидим 99
        tv.testMethod(first);
        System.out.println("Main method:" + first);      // И здесь мы снова увидим 99
    }

    public void testMethod(double first) {
        System.out.println("Test method:" + first);       // Здесь мы увидим 99
        first = 11;
        System.out.println("Test method:" + first);       // Здесь мы увидим 11
    }
}
```

Для запуска нашего класса можно использовать следующее «умение» NetBeans — вы можете запускать определенный класс. Для этого откройте в редакторе нужный класс и нажмите комбинацию Shift+F6 или выберите пункты меню Run->Run File.
Запустите наш пример и вы увидите, что внутри метода testMethod мы поменяли значение переменной first, но это никак не повлияло на значение переменной first в методе main. Как вы уже наверняка поняли переменная first в разных методах — это совершенно разные переменные. Пусть и называются одинаково. Само собой две одинаковых переменных внутри одного метода недопустимы. А в разных — не проблема.

Мы рассмотрели вариант когда в метод передаются данные элементарных типов. Теперь настало время посмотреть на объекты. С ними будет все несколько сложнее.
Давайте вспомним важный момент, который мы рассматривали ранее — при создании переменной сложного типа (класса) мы не создаем объект сразу. Мы создаем ссылку на объект, который потом надо создать с помощью оператора new.
Т.е. когда мы передаем объект в метод, на самом деле мы передаем не объект, а ссылку на него. И ссылка копируется. Т.е. у нас возникает две ссылки, которые указывают на один и тот же объект. И обе эти ссылки позволят нам обращаться к его свойствам, вызывать его методы. Что это означает? Это означает, что если мы поменяем свойство объекта при обращении к нему через одну ссылку, то и вторая ссылка позволит нам вернуть измененное свойство. Давайте рассмотрим простой пример. Используем снова наш класс Robot и класс для его управления RobotManager.

```java
public class Robot
{
    // Текущий курс (в градусах)
    private double course = 0;

    public double getCourse() {
        return course;
    }
```

```
 9
10      public void setCourse(double course) {
11          this.course = course;
12      }
13  }
```

```
 1  public class RobotManager
 2  {
 3
 4      public static void main(String[] args) {
 5          // Создаем объект для управления роботом
 6          RobotManager rm = new RobotManager();
 7
 8          // Создаем объекта класса Robot
 9          Robot robot = new Robot();
10          // Курс 45 градусов
11          robot.setCourse(45);
12          // Напечатать курс
13          System.out.println(robot.getCourse());      // Здесь будет 45
14
15          // вызываем метод и передаем туда робота
16          rm.changeCourse(robot);
17
18          // Напечатать курс
19          System.out.println(robot.getCourse());      // Здесь будет 180
20      }
21
22      private void changeCourse(Robot robot) {
23          robot.setCourse(180);
24      }
25  }
```

Запускаем класс RobotManager и осознаем результат :). Он предсказуем — у нашего робота поменялось значение курса. Итак, наша ссылка на робота скопировалась. Но обе ссылки работали с одним и тем же роботом. Но что произойдет, если внутри метода changeCourse мы сделаем так:

```
1      private void changeCourse(Robot robot) {
2          robot = new Robot();
3          robot.setCourse(180);
4      }
```

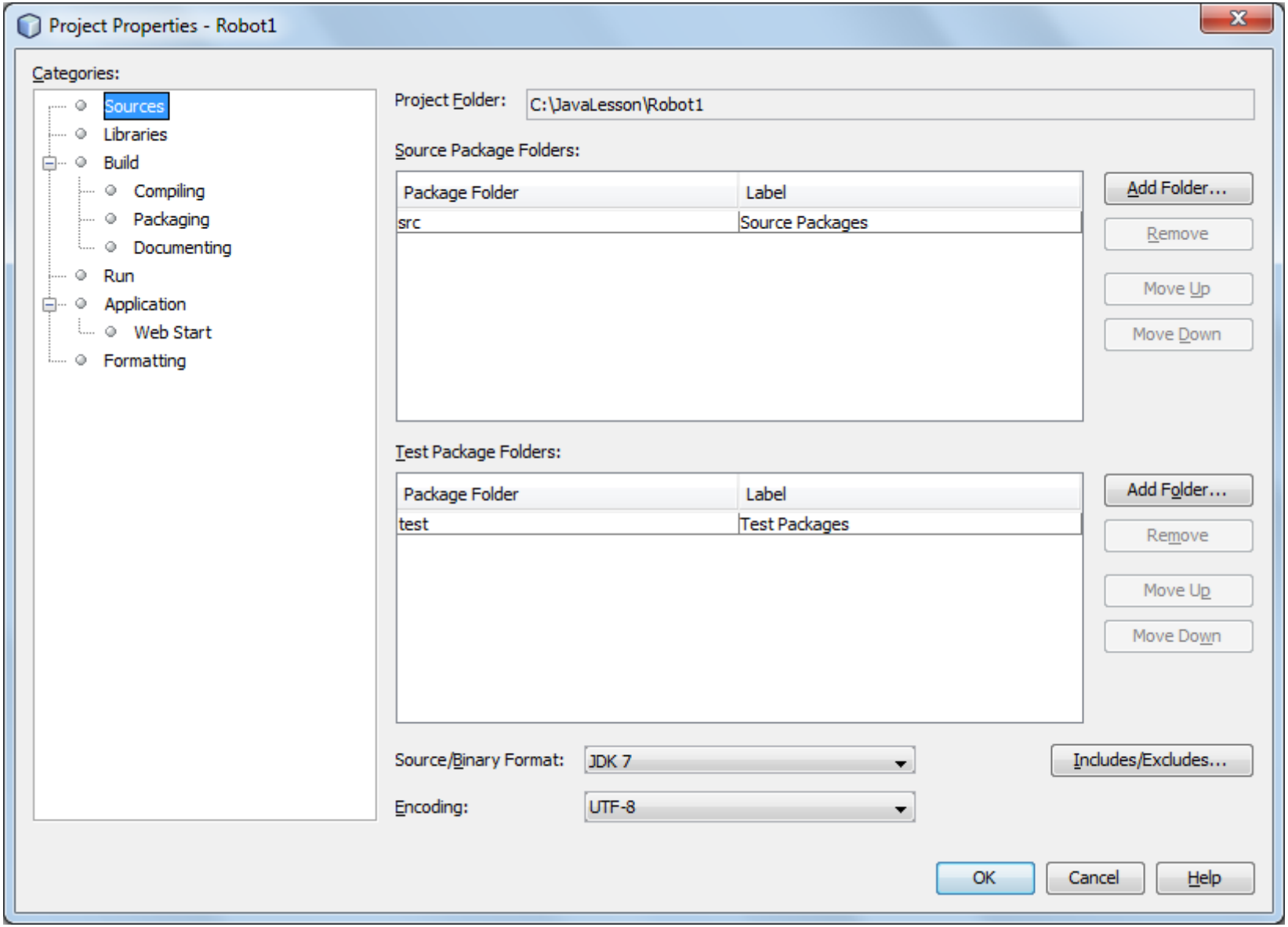
Локальная ссылка теперь будет указывать на новый объект и все действия будут производиться над новым объектом. А старый робот не будет меняться. Надеюсь, что теперь механизм передачи параметров более понятен.

Запуск проекта и метод main

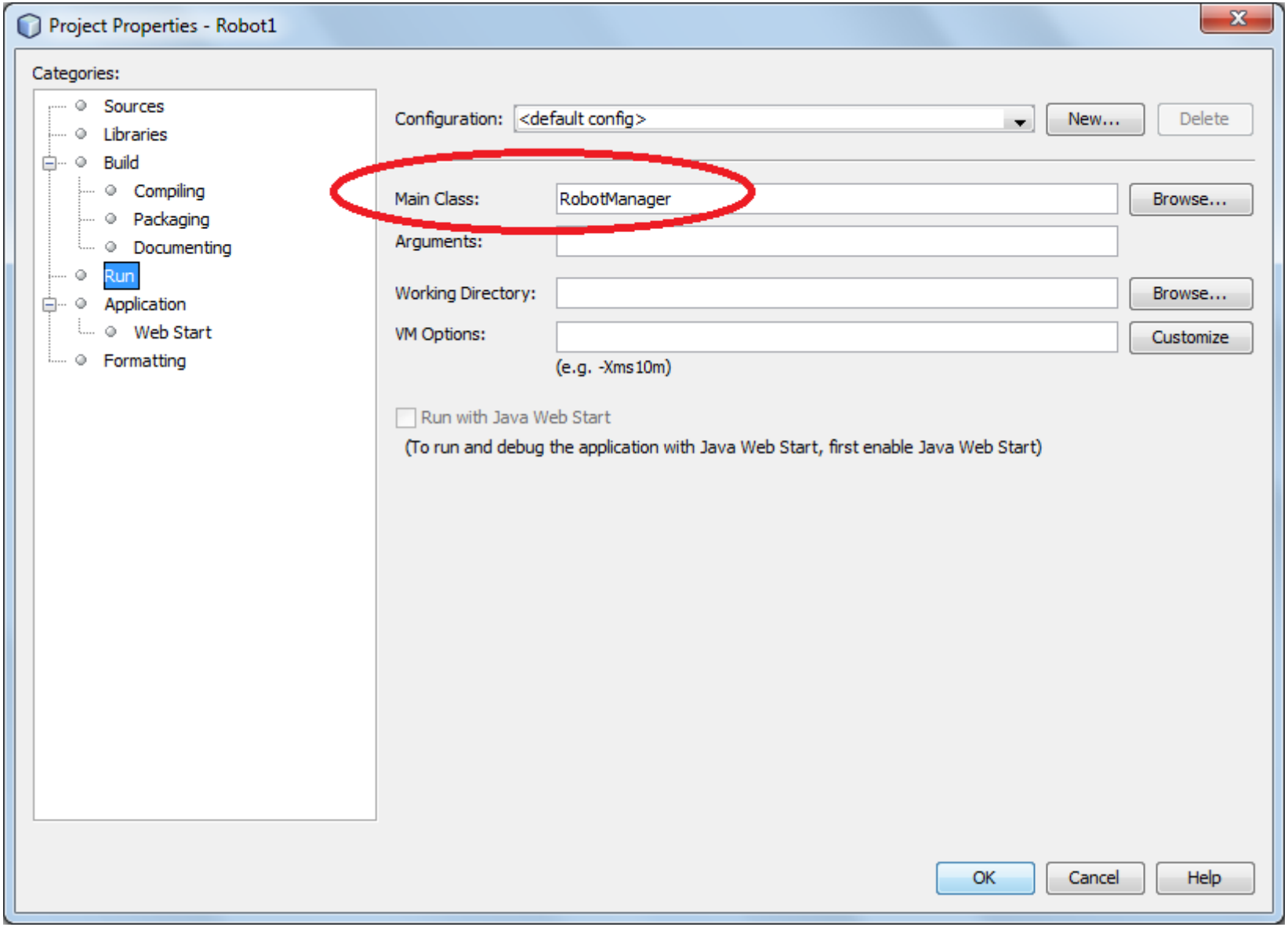
Мы уже сталкивались с этим методом, когда запускали примеры и вы наверняка догадались, что именно этот метод является по сути «точкой входа» в программу. Это действительно так и есть. JVM может запустить только тот класс, который имеет такой метод. Мы пока разобрали не все слова, которые составляют описание этого метода, но не будем спешить. Мы все разберем в процессе изучения. Пока просто примите как необходимость — для запуска программы вам нужно передать JVM класс, в котором именно так и записано

```
1 public static void main(String[] args) {
2     ...
3 }
```

И еще один момент. Только что мы с вами рассмотрели класс TestVariable и даже запускали его с помощью комбинации Shift+F6. Но также мы запускали класс RobotManager простым нажатием F6. В нашем проекте у нас одновременно существует два класса, которые можно запускать, потому что в обоих есть метод main. И здесь нет никакой путаницы. На самом деле загадка раскрывается очень просто — проект на NetBeans содержит внутри себя класс, который надо запускать по умолчанию. Вы можете в этом убедиться следующим образом: Нажмите правой кнопкой мышки на проект (Robot1) и в выпавшем меню выберите самы нижний пункт (Properties). Перед вами появиться окошко со свойствами проекта:



Выберите в левой части пункт «Run» и вы увидите, что наш проект использует по умолчанию класс RobotManager.



Если вы введете в выделенную строку другой класс, то тогда NetBeans по клавише F6 будет запускать его. Вот и весь секрет.

И теперь нас ждет следующая статья: Наследование

36 comments to *Инкапсуляция*

- 

  Апрель 2, 2015 at 12:45
  Татьяна says:

  Добрый день! Я только начинаю изучать джаву поэтому может мой вопрос покажется странным но очень хочется разобраться для себя. Не понимаю принципа применения геттеров и сеттеров вообще. Если мы говорим о безопасности объявляем переменные приват и тут же сеттеры и геттеы связанные с ними объявляем паблик. А код становится больше. Что я не понимаю?

  Reply

  - 

    Апрель 2, 2015 at 21:18
    admin says:

    Но я могу и запретить тот же сеттер. Могу сделать какие-то проверки или дополнительные действия — например, просто присвоить новую ширину в переменную width нельзя, надо еще изменить сам размер. В этом вся фишка — можно управлять доступом.

    Reply

- 

  Июль 7, 2015 at 09:21
  Grif says:

  Большое спасибо за уроки, очень кропотливый и полезный труд!
  Лет 20 назад баловался на бейсике и ассемблере, не серьёзно, так, для себя на платформе ZILOG Z80.
  Уже пытался проникаться ООП … но не очень успешно, тяжело мне дался и Ваш пример, я его модернизтровал и для лучшего понимания получил такой код у менеджера роботов:

```
public class RobotManager
{
public static void main(String[] args) {
// Создаем объект для управления роботом
RobotManager rm = new RobotManager();
```

```
// Создаем объекты класса Robot
Robot robot = new Robot();
Robot robot1 = new Robot();
// Курс 45 градусов
robot.setCourse(45);
robot1.setCourse(75);
// Напечатать курс
System.out.println(«robot = » + robot.getCourse()); // Здесь будет 45
System.out.println(«robot1 = » + robot1.getCourse()); // Здесь будет 75

// вызываем метод и передаем туда робота
rm.changeCourse(robot);

// Напечатать курс
System.out.println(«robot after rm = » + robot.getCourse()); // Здесь будет 120
System.out.println(«robot1 = » + robot1.getCourse()); // Здесь будет 75

// вызываем метод и передаем туда робота
rm.changeCourse(robot1);
// Напечатать курс
System.out.println(«robot1 after rm = » + robot1.getCourse()); // Здесь будет 120
}

private void changeCourse(Robot robot) {
robot.setCourse(120);
robot = new Robot();
robot.setCourse(180);
System.out.println(«robot2 in rm = » + robot.getCourse()); // Здесь будет 180
}
}
```
Ещё раз большое спасибо, мне кажется я начал что-то понимать 🙂

Reply

- 

Июль 24, 2015 at 13:36
Gala says:

Здравствуйте! Большое спасибо автору за труд!

Есть один момент, который я никак не могу понять. Почему в примере TestVariable все System.out.println идут не по порядку? При выводе результата последний идет
Main method:99.0. ?

Заране спасибо за ответ)))

Reply

- 
  Июль 24, 2015 at 13:48
  admin says:

  Должно быть так:
  Main method:99.0
  Test method:99.0
  Test method:11.0
  Main method:99.0

  Встречный вопрос — а в каком порядке, по-вашему, они должны быть ?

  Reply

  - 
    Февраль 23, 2017 at 10:55
    Юрий says:

    Присоединяюсь к вопросу. Тоже не понимаю почему идет в таком порядке. По порядку должно быть :
    Main method:99.0
    Main method:99.0
    Test method:99.0
    Test method:11.0

    а выдается :
    Main method:99.0
    Test method:99.0

Test method:11.0
Main method:99.0

каким образом Test method втыкается в середину?

попытался поменять местами строки, чтобы то, что выдается в консоли вглядело последовательно, но код не работает. Почему?

```
public class TestVariable {
public static void main(String[] args) {
double first = 99;
System.out.println(«Main method:» + first); // Здесь мы увидим 99
System.out.println(«Main method:» + first); // И здесь мы снова увидим 99
}

public void testMethod(double first) {
TestVariable tv = new TestVariable();
tv.testMethod(first);
System.out.println(«Test method:» + first); // Здесь мы ничего не видим
first = 11;
System.out.println(«Test method:» + first); // И здесь мы ничего не видим

}
}
```

Reply

Февраль 23, 2017 at 15:21
admin says:

У Вас ошибка в рассуждениях. Весь вывод идет в соответствии с кодом.
1. Первый шаг печатает 99 из метода main
2. Второй шаг печатает 99 из метода testMethod
3. Третий шаг печатает 11 из метода testMethod
4. Четвертый шин печатает 99 из метода main

Посмотрите код внимательнее — там все честно и логично. Ваш вариант как раз неправильный.

Reply

Февраль 24, 2017 at 08:06
Юрий says:

Разобрался! Огромное вам спасибо за учебник и отдельное спасибо, что отвечаете на вопросы, эта поддержка очень сильно помогает в освоении материала.
Я пытался начать обучение JAVA вот с этого курса https://www.udemy.com/java-tutorial/
Но к середине понял, что погряз в теории и в голове ничего не откладывается.
Случайно наткнулся на ваш учебник, когда искал что такое полиморфизм. Решил пройти ваши уроки с самого начала.
Нравится, что разжевывается каждая мелочь, дотошным людям, вроде меня, это крайне необходимо.

Февраль 25, 2017 at 13:34
admin says:

Спасибо, заходите еще. Удачи.

Сентябрь 30, 2015 at 21:10
avirdoz says:

Доброго времени суток. Подскажите пожалуйста, в первом листинге какие роли в классе Robot выполняют методы:

```
public double getX() {
return x;
}

public double getY() {
return y;
}

public double getCourse() {
return course;
```

}

Reply

Октябрь 1, 2015 at 10:06
admin says:

В общем странный вопрос — если вы внимательно читали идею инкапсуляции, то он не должен был возникнуть. Эти методы позволяют получить данные полей — координаты x и y а также курс — course.

Reply

Октябрь 1, 2015 at 10:58
avirdoz says:

Просто убирая эти методы из программы она компилируется и выполняется. И я не вижу в методе main обращения к этим методам.

Reply

Октябрь 1, 2015 at 14:33
admin says:

В данном примере не используется. Но это не значит, что такие методы не нужны — Вам же когда-нибудь потребуется узнать (не в этой задаче, но в другой), какие координаты у робота. А он их отдать не сможет — значит Вы написали «неполноценный» класс.

Reply

Декабрь 7, 2015 at 09:09

alex says:

Здравствуйте, спасибо большое за интересные уроки. В одном из примеров в классе «RobotManager» объявляется переменная «rm» — объект этого же класса. Не могу понять как это возможно, класс же у нас получается не до конца описан, а его объект уже создан, причем внутри него же. Можете это как то прокомментировать? Может я что то упустил или не так понимаю.

Reply

Декабрь 7, 2015 at 10:11
admin says:

Порядок методов в классе в общем не важен для компилятора. Вы можете объявлять их в любом месте и они в момент компиляции все равно будут найдены.

Reply

Декабрь 23, 2015 at 10:03
Виталий says:

Очень понятно написано, спасибо!
Подскажите, как мне обратиться к экземпляру класса в такой ситуации:
Есть 3 класса. В первом классе создаем экземпляр второго, а уже из третьего класса нужно вызвать заполненные поля второго.
Значит мне нужно создавать экземпляр первого класса и уже из него вызывать нужный мне второй?

Reply

Декабрь 23, 2015 at 11:03
admin says:

Как я понял, есть объект класса A, в нем есть ссылка на объект класса B. Теперь из объекта класса C надо вызвать поля объекта класса B.
Значит в объект класса C надо передать объект класса A и через него добраться до объекта класса B.
Также надо, чтобы у объекта класса A был метод, позволяющий получить доступ к объекту класса B.

Ну или просто в объект класса C передать ссылку на объект класса B — если такое возможно. Так конечно проще. Но будет ли это правильнее — надо смотреть по задаче.

Reply

Декабрь 23, 2015 at 12:19
Виталий says:

```
public class firstClass{
Cat cat = new Cat();
cat.setName(«Васька»)
}

public class Cat{
private String name;
public void setName(String name) {
this.name = name;
}

public String getName() {
return name;
}
}

public class thirdClass{
вот тут то и надо получить переменную (имя), установленную в первом классе
}
```

В дальнейшем планируется, что эта переменная будет использоваться и в других классах. Поэтому и встает вопрос: как все-таки правильно ее использовать?
Спасибо

Reply

Декабрь 23, 2015 at 13:13

admin says:

Если речь идет о получении объекта Cat из объекта firstClass, то в данной постановке это невозможно. Надо объявить Cat cat как поле класса

```
public class firstClass
{
private Cat cat;

public firstClass() {
cat = new Cat();
cat.seName(«Васька»);
}

public Cat getCat() {
return cat;
}
}
```

Reply

- Март 22, 2016 at 11:15
  Alex says:

  Огромная просьба! Начиная с этой статьи и далее, в конце статьи вставьте ссылки на следующую? Часто читаю с мобильного телефона, меню навигации в мобильном браузере (по крайней мере у меня в win phone на IE) не работает. А читать статьи хочется последовательно! Спасибо!

  Reply

  - Март 22, 2016 at 14:28
    admin says:

    Постараюсь учесть.

    Reply

Сентябрь 11, 2016 at 21:57
Вячеслав says:

Добрый день. Большое спасибо за курс!
Подскажите, как можно обратиться ко всем объектам класса одновременно (либо по очереди в цикле)? Например, надо одинаково переместить всех роботов, но мы не хотим перечислять их имена, или даже не помним их,.. и вообще забыли сколько роботов создали.

Reply

Сентябрь 13, 2016 at 02:02
admin says:

Я не знаю как можно такое организовать. Тем более одновременно. Наверно есть способы узнать у JVM список ссылок на объекты одного класса, но это достаточно специфические запросы — я о таких не читал.

Reply

Октябрь 23, 2016 at 04:17
StickOfFaith says:

Можно создать массив типа Robot[] robots, или использовать ArrayList robots. Для первого случая можно впихать так robots[i] = new Robot(); . Можно в цикле пройтись по массиву и вызвать метод robots[i].doSomething(). Не забудьте сначала приниициализировать массив Robot[] robots = new Robot[n]. Для второго можно robots.add(new robot), а вызвать метод можно так robots.get(i).doSomething().

Reply

Май 19, 2017 at 16:35
Formfaktor says:

Можно такое организовать. Нужно использовать в классе статические члены (методы и переменные). НО! поскольку ключевое слово «static» еще не рассматривалось то, как говорят у нас, не буду лезть поперед батька в пекло 🙂
Решил немного освежить знания. Спасибо Вам огромное за Ваш труд.

Reply

- 
Декабрь 4, 2016 at 20:07
ЮРА says:

Здравствуйте.Можете объяснить почему в первом примере используется одновременно и сеттер и геттер курса?

Reply

  - 
Декабрь 5, 2016 at 09:52
admin says:

Чтобы иметь доступ к полю «курс» для чтения и записи. Смысл инкапсуляции — управлять процессом доступа к полям. Т.к. поле «курс» закрыто для внешних классов, но им надо иметь возможность менять или читать курс, то сделаны эти методы.

Reply

- 
Декабрь 11, 2016 at 13:18
Кирилл says:

Совсем ничего не понимаю(в примере с Роботом и его менеджером. Как на робота влияет его начальная координата?В этом случае никак?так и должно быть?мы просто швыряем его куда хотим? Я менял его начальную координату, а результат не изменился…
// Текущий курс (в градусах)
private double course = 5;
и не понимаю как работает строчка rm.changeCourse(robot); в классе менеджера. Откуда она берет на сколько поменять курс? Почему именно 180 становится в итоге?
Третий день пытаюсь понять,чувствую,что не понимаю какую-то мелочь,которую уже не в силах заметить…потому пишу

Reply

Декабрь 12, 2016 at 11:08
admin says:

В этом примере обсуждается понимание ссылки на объект и их копирование при вызове функции/метода. Мы передаем ссылку на робота в метод changeCourse — таким образом у нас будет две ссылки на один и тот же объект. Мы меняем курс робота через одну ссылку (внутри метода changeCourse) и видим это изменение через другую ссылку (уже в методе main). Вот и весь пример.

Reply

Май 3, 2017 at 00:17
Андрей says:

Добрый день!
Вы не могли прокомментировать ваш код из данного урока :
public class TestVariable
{
public static void main(String[] args) {
double first = 99;
// Создаем экземпляр класса
TestVariable tv = new TestVariable();
System.out.println(«Main method:» + first); // Здесь мы увидим 99
tv.testMethod(first);
System.out.println(«Main method:» + first); // И здесь мы снова увидим 99
}
Конкретно вопрос по строке :
TestVariable tv = new TestVariable();
Что такое экземпляр класса?
Объект?
Данная строка запускает конструктор класса TestVariable.
Конструктор создает объект tv класса TestVariable.
Но в классе конструктор не описан.

И как мы из класса создаем объект? Разве так можно делать?
Не совсем понимаю этот пример.

Reply

Май 3, 2017 at 11:20
admin says:

Я под экземпляром класса подразумеваю именно объект.
«Конструктор создает объект» — это неправильное понимание. Объект создается командой new. А конструктор — это по сути обязательный метод, который выполняется сразу после создания объекта.
«Но в классе конструктор не описан» — то, что конструктор не описан, не значит, что его нет. В этом случае конструктор создается автоматически без входных параметров (так называемый конструктор по умолчанию). Если у предка конструктор с параметрами, то придется в наследнике его написать обязательно.

Reply

Май 4, 2017 at 10:28
Андрей says:

Спасибо большое за комментарий!
Остался непонятным следующий момент:
Структура этого кода такая:

Класс А {
создание объекта класса А;
}

Можно ли создавать объект класса из этого же класса?
Почему у меня возник вопрос:
Полное описание класса, а соответственно и объекта этого класса, находится в блоке между фигурными скобками {}.
Если внутри этих скобок создать объект, то он получается «недоношенным»,т. к. полного описания объекта еще не произошло.

Reply

Май 4, 2017 at 12:21
admin says:

Не путайте описание класса и исполнение программы по этому описанию. Описание компилируется в байт-код и загружается в память. Это как напечатанная инструкция.
Дальше исполнение уже загруженного кода идет по шагам, которые описаны в этой инструкции. Кто мешает в инструкции ссылаться на пункты этой же инструкции ?
Никто не мешает внутри класса A использовать объекты этого же класса. Логически это вполне обосновано. Например машина-погрузчик может использовать другую машину-погрузчик для своих целей. Точно так же и в коде.

Reply

Май 6, 2017 at 12:39
Андрей says:

Антон, спасибо за комментарий!
Я не на 100% понял момент с созданием объекта класса из этого же класса, но поразбираюсь еще)

Май 19, 2017 at 17:09
Formfaktor says:

Здравствуйте, Андрей. Попробуйте почитать о ключевом слове «this». Может это поможет Вам разобраться до конца в этом вопросе.
Когда мы пишем конструкцию типа :
class A {

}
То уже только этим описываем класс. Примитивный, как амеба, но класс. И можем смело создавать объекты (экземпляры) этого класса (типа). Как говорил автор, во время компиляции компилятор сам добавит конструктор поумолчанию. Поэтому

class A {
A aObj = new A();
}
вполне логично. Ведь матрица, слепок, чертёж уже готов.

- 
  Декабрь 31, 2017 at 19:45
  Олег says:

  С НОВЫМ 2018 ГОДОМ, ДРУЗЬЯ!!!

  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

9 − 9 =

Add comment