В предыдущих статьях мы познакомились с основными возможностями Java для написания объектно-ориентированных программ. Попробуйте критически переосмыслить все, что было описано, со следующей точки зрения: Вам, как разработчику программ, предоставляется набор инструментов, которые можно использовать для построения своих программ. Какие-то конструкции подходят для одних случаев, другие — для других. Чем больше у Вас появляется опыта в использовании базовых конструкций, тем проще воспринимаются более сложные моменты. Накапливается опыт и Ваш инструментарий расширяется — Вы знакомитесь и уже понимаете как надо использовать новые конструкции и понятия. Увы, но простым чтением для накопления опыта не обойтись — надо писать программы. Только практика может помочь Вам освоиться.
Но прежде чем мы продолжим знакомиться с другими конструкциями Java для работы с классами, я бы хотел предложить познакомиться с весьма заслуженной конструкцией данных — массивами.

Массивы

Если попробовать дать определение массиву, то на мой взгляд оно должно выглядеть так: Массивом называется множество однотипных объектов, объединенных одним именем и доступ к каждому объекту в этом множестве осуществляется по порядковому номеру (индексу).
Важно отметить, что сам по себе массив — это объект, а значит ему присущи те же особенности. что и объекту — его надо создать, у него есть некоторые поля и методы, которые Вы можете использовать для работы. Использование массивов на сегодня не является сильно распространенным явлением, но они занимают достаточно нужную и прочную нишу в современном инструментарии, так что знакомство с

- 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

ними весьма важно. Давайте знакомиться.

В первую очередь нам необходимо узнать как описать массив и как его создать. Как говорила моя бабушка (хотя такое говорили наверно многие бабушки), повторение мать учения. Так что вернемся опять к нашему роботу. Если Вы забыли, то вот его код.

```
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
9      public Robot(double x, double y) {
10         this.x = x;
11         this.y = y;
12     }
13
14     // Передвижение на дистанцию distance
15     public void forward(int distance) {
16         x = x + distance * Math.cos(course / 180 * Math.PI);
17         y = y + distance * Math.sin(course / 180 * Math.PI);
18     }
19
20     // Печать координат робота
21     public void printCoordinates() {
```

```
    public void printCoordinates() {
        System.out.println(x + "," + y);
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public double getCourse() {
        return course;
    }

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

Класс Robot остался таким же каким он был и раньше, а вот класс RobotManager содержит объявление массива роботов и создание массива.

```
package edu.javacourse.robot;

public class RobotManager
{
    public static void main(String[] args) {
        // Объявление массива
        Robot[] rbts;
        // Создание массива для 10 роботов
        rbts = new Robot[10];
    }
}
```

Т.е. для объявления массива мы пишем сначала имя класса, объекты которого будут составлять массив, после них пустые квадратные скобки и имя переменной. Эта переменная — ссылка на массив. Для создания массива мы используем ключевое слово new, снова пишем класс и квадратные скобки, внутри которых указываем размер массива. Как и обычную переменную, мы можем совместить объявление массива и его создание в одну строку. Вот так:

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

```
2
3  public class RobotManager
4  {
5      public static void main(String[] args) {
6          // Объявление массива и создание
7          Robot[] rbts = new Robot[10];
8      }
9  }
```

Теперь в переменной rbts находится ссылка на массив из десяти ссылок на роботов. Я не оговорился и предлагаю Вам еще раз внимательно прочитать предыдущую фразу — массив rbts содержит десять ссылок на объекты типа Robot. Т.к. это десять ссылок, то после создания массива каждая из этих ссылок содержит величину null. попробуем убедиться, что так оно и есть. Для этого обратимся к каждой ссылке из массива rbts по индексу. Для того, чтобы обратиться к конкретному элементу массива опять используются квадратные скобки, внутри которых указывается порядковый номер (индекс). Важно отметить, что номера (индексы) начинаются со значения 0. Т.е. обращение rbts[1] означает обращение ко второму (не к первому) элементу в массиве rbts. При размере массива в 10 элементов, индекс последнего будет равен 9. Единственное требование к индексу — это должно быть целое число. Причем может использоваться переменная, а не только константа. Подкрепим наши новые знания практикой — обратимся к каждому элементу массива rbts

```
1  package edu.javacourse.robot;
2
3  public class RobotManager
4  {
5      public static void main(String[] args) {
6          // Объявление массива и создание
7          Robot[] rbts = new Robot[10];
8
9          // С помощью цикла изменяем переменную i и используем ее
10         // для обращения к элементу массива
11         for(int i=0; i<10; i++) {
12             // Печатаем элемент массива
13             System.out.println(rbts[i]);
14         }
15     }
16 }
```

На самом деле все достаточно просто — в цикле меняется значение переменной i, которая и помогает нам обратиться к нужному элементу массива rbts. Само обращение к элементу массива выглядит вот так:rbts[i]. Это обращение практически не отличается от обращения к обычной переменной. И как я уже упоминал, пока в наших переменных ничего нет, в чем мы и могли убедиться, увидев надписи null в количестве 10 штук. Теперь инициализируем наши переменные, а заодно познакомимся с одним важным свойством массива — как можно узнать его размер.

```java
package edu.javacourse.robot;

public class RobotManager
{
    public static void main(String[] args) {
        // Объявление массива и создание
        Robot[] rbts = new Robot[10];

        // Обратите внимание на запись rbts.length - это свойство
        // (неизменяемое) возвращает размер массива
        for (int i = 0; i < rbts.length; i++) {
            // Создаем объект типа Robot
            rbts[i] = new Robot(i * 10, i * 10);
        }

        // Еще один цикл, который вызывает печать координат у каждого робота
        for (int i = 0; i < rbts.length; i++) {
            rbts[i].printCoordinates();
        }
    }
}
```

У нас теперь два цикла. Первый служит для создания объектов типа Robot, а второй вызывает для каждого метод для печати его координат. Обратите внимание, что теперь мы используем свойство массива length — это позволяет писать более гибкий код. Мы можем создать 15, 43 или 17 объектов и для этого код циклов менять не надо. Это упрощает разработку и избавляет от лишних ошибок. При создании робота мы используем конструктор, который принимает в качестве входных параметров координаты X и Y. Более внимательно посмотрев на код видно, что каждый робот получает координаты, которые пропорциональны его индексу в массиве.

Таким образом работа с массивом — это работа с отдельными его элементами при помощи изменения значения индекса. Именно через индекс Вы можете обратиться к любому объекту внутри массива. Что это нам дает ? Возможность работы с группой объектов с помощью одной переменной, что упрощает создание программ. Представьте, что что у Вас была бы необходимость работать с парой сотен роботов. Будете описывать две сотни переменных ? Или Вы вообще заранее не знаете, сколько роботов надо будет использовать. В случае объявления массива Вам это уже не так важно — 10 или 100. Если Вы имеете алгоритм для управления всеми этими роботами через индекс, то количество Вас уже не смущает.

Начав работу с массивами, мы сразу стали создавать массивы со сложными типами данных — классами. И убедились, что принципы создания объектов остались такими же, как и для обычного объекта — надо обязательно создать объект внутри массива. Само создание массива порождает только набор ссылок (в нашем примере 10 ссылок на объекты типа Robot). И только после создания объектов для КАЖДОЙ ссылки мы получаем полноценный набор объектов внутри массива.
Но мы также знакомились в элементарными типами — byte, int, double. В случае объявления переменной элементарного типа, как Вы возможно помните, создавать ее не надо — она существует сразу при объявлении. Ее надо только инициализировать — Для массивов этот принцип точно такой же — создавая массив целых чисел, вы сразу получаете целые числа, с которыми можно работать. В спецификации языка Java указано, что переменные элементарных типов инициализируются при создании. Для чисел это 0. Рассмотрим простой пример:

```java
package edu.javacourse.robot;

public class ArrayDemo
{
    public static void main(String[] args) {
        int[] demo = new int[10];
        for (int i = 0; i < demo.length; i++) {
            // Переменная доступна и там значение 0
            System.out.println(demo[i]);
            // Присваиваем ей другое значение
            demo[i] = 10 * (i + 1);
        }
        for (int i = 0; i < demo.length; i++) {
            System.out.println(demo[i]);
        }
    }
}
```

По результатам исполнения примера можно видеть. что начальные значения всех элементов массива — число 0. Предлагаю рассмотреть несколько достаточно простых примеров работы с массивами. Примеры не сложные и существуют уже многие годы.
Первый пример — необходимо сосчитать сумму все чисел массива. В этом примере мы увидим любопытную конструкцию инициализации элементов массива. Сам алгоритм достаточно несложный — кроме массива создается переменная, которая будет «накапливать» сумму всех элементов. Т.е. проходя в цикле по всем элементам мы будем прибавлять значение каждого элемента к переменной, начальное значение которой должно быть равно нулю. Смотрим реализацию.

```java
package edu.javacourse.array;

public class SumArray
{
    public static void main(String[] args) {
        // Вы можете использовать инициализацию вот в таком виде
        // Перечисляете элементы массива чееpз запятую
        int[] sample = {12, 56, 7, 34, 89, 43, 23, 9};

        // До расчета суммы переменная для ее хранения содержит 0
        int summa= 0;
        // Выполняем проход по всем элементам и прибавляем каждый к сумме
        for(int i=0; i< sample.length; i++) {
            summa += sample[i];
        }
        System.out.println("TOTAL:" + summa);
    }
```

```
18 }
```

Конструкция инициализации массива записывается просто — внутри фигурных скобок через запятую вводятся нужные числа. Запись достаточно лаконичная и наглядная, но имеет существенный недостаток — набор элементов массива жестко «забит» в коде, что не всегда является удобным. Важно отметить, что внутри скобок можно создавать объекты. Например для создания массива из трех роботов можно написать вот такую конструкцию:

```
1     Robot[] sample = { new Robot(1, 1), new Robot(0, 0), new Robot(12, 5) };
```

Дальше в примере создается переменная summa, которая инициализируется нулевым значением и уже в цикле к ней прибавляется каждый элемент массива. Вот и все.
Следующий пример тоже может быть признан классикой — это сортировка массива методом «пузырька». Далеко не самый эффективный способ сортировки, но в качестве примера очень даже неплохо. Сортировка производится по следующему алгоритму: сравниваются два находящихся рядом элемента массива. Если первый элемент больше второго, то их надо поменять местами. При каждом проходе по всему массиву элементы с меньшим значением потихонечку передвигаются к началу — «всплывают». Отсюда и название метода. Проход по всем элементам повторяется до тех пор, пока хоть одна пара элементов поменялась местами.

```
1  package edu.javacourse.array;
2
3  /*
4   * Пример сортировки массива методом пузырька
5   */
6  public class SortArray
7  {
8      public static void main(String[] args) {
9          int[] sample = {12, 56, 7, 34, 89, 43, 23, 9};
10
11         // выставляем признак "обмена" переменных в true, чтобы начать цикл
12         boolean changed = true;
13
14         // цикл длится до тех пор, пока при проверке массива ни одного обмена не произошло
15         while (changed) {
16             // Надеемся, что обмена данных не будет
17             changed = false;
18             // Проходим по всему массиву
19             for (int i = 0; i < sample.length - 1; i++) {
20                 // Если впереди стоящее число больше, чем следующее - меняем
21                 // их местами и выставляем признак, что был обмен
22                 if (sample[i] > sample[i + 1]) {
```

```
                if (sample[i] > sample[i + 1]) {
                    // Производим обмен с использованием дополнительной переменной
                    int tmp = sample[i];
                    sample[i] = sample[i + 1];
                    sample[i + 1] = tmp;
                    // Выставляем признак обмена в true
                    changed = true;
                }
            }
        }
        // Выводим отсортированный массив
        for (int i = 0; i < sample.length; i++) {
            System.out.println(sample[i]);
        }
    }
}
```

В качестве самостоятельно работы предлагаю решить классическую задачу — нахождение самого большого (маленького) числа в массиве.

Цикл foreach

В этом разделе мы познакомимся с циклом foreach. Он позволяет более просто записать проход по всем элементам массива. Внешне он очень похож на цикл for, но имеет ряд особенностей и ловушек, с которыми мы сейчас познакомимся. Рассмотрим простой пример — инициализация массива и вывод всех его элементов. Обратите внимание на форму записи цикла.

```
package edu.javacourse.array;

public class ForEachExample
{
    public static void main(String[] args) {
        int[] sample = {12, 56, 7, 34, 89, 43, 23, 9};

        // выводим элементы в цикле foreach
        for (int t : sample) {
            System.out.println(t);
        }
    }
}
```

Самой главной в этом примере является строка for (int t : sample) {. Эту запись можно интепретировать следующим образом — цикл проходит по всем элементам массива, каждый раз помещая в переменную t значение следующего элемента массива. Т.е. при каждом проходе цикла в переменной t последовательно будет появляться значение sample[0], sample[1], sample[2] и т.д.

Цикл выглядит более компактно и пользоваться им удобно. Но существует важная особенность, о которой надо обязательно помнить. Я только что ее проговорил, но повторю еще раз: «при каждом проходе цикла в переменной t последовательно будет появляться значение». Иными словами, в переменную t КОПИРУЕТСЯ значение из элемента массива. Таким образом получается, что перменная t и переменная sample[0] — это РАЗНЫЕ переменные. Рассмотрим более сложный пример:

```
1  package edu.javacourse.array;
2
3  public class ForEachExample
4  {
5      public static void main(String[] args) {
6          int[] sample = new int[5];
7
8          System.out.println("До foreach");
9          // выводим элементы в цикле foreach - их значение 0
10         for (int t : sample) {
11             System.out.println(t);
12         }
13
14         // Думаем, что происходит инициализация
15         for (int t : sample) {
16             t = 99;
17         }
18
19         System.out.println("После foreach");
20         // выводим элементы в цикле foreach - снова 0
21         for (int t : sample) {
22             System.out.println(t);
23         }
24     }
25 }
```

При запуске этого примера мы можем увидеть, что элементы массива остались такими, какими и были — равными 0. Для инициализации придется использовать обычную конструкцию.

```
1  package edu.javacourse.array;
2
3  public class ForEachExample
4  {
5      public static void main(String[] args) {
6          int[] sample = new int[5];
7
8          System.out.println("До foreach");
```

```
        System.out.println("до foreach");
        // выводим элементы в цикле foreach - их значение 0
        for (int t : sample) {
            System.out.println(t);
        }

        // Для инициализации элементов foreach не подходит
        for (int i = 0; i < sample.length; i++) {
            sample[i] = 99;
        }

        System.out.println("После foreach");
        // выводим элементы в цикле foreach - теперь 99
        for (int t : sample) {
            System.out.println(t);
        }
    }
}
```

Разница между этими примерами в средней части — посмотрите внимательно.
На этом мы закончим первое знакомство с массивами. В современном мире программирования массивы уже не так популярны, как лет 20 назад. У них есть своя ниша, где они применяются, но на мой взгляд сегодня они уступили свои позиции более современным структурам данных. массивы имеют ряд недостатков, среди которых важным является то. что их размер определяется сразу при их создании и изменить его достаточно сложно. Тем не менее знание массивов должно быть в Вашем арсенале. В следующей части мы продолжим знакомство с массивами и рассмотрим более сложные примеры их использования.

Архивы примеров, арссмотренных в данном разделе можно скачать здесь — RobotArray и ArrayDemo.

И теперь нас ждет следующая статья: Многомерные массивы.

9 comments to *Массивы — знакомство*

- 
  Январь 14, 2016 at 20:43
  Vladimir says:

  Уважаемый Админ.
  Поясните, как отрабатывает код из класса очередь примера предыдущего урока «очередь объектов»:
  // Перемещаем «голову» на следующий элемент

head = head.getNext();
ведь, насколько я понимаю, head.getNext() должен вернуть значение поля next объекта head, но мне не понятно, как когда и чем это поле было заполнено при реализации метода pull.
Еще вопрос: для прохождения курса «Начальный курс Java», необходимо ли пройти материалы выложенных на сайте уроков.

Reply

Январь 15, 2016 at 11:05
admin says:

Метод pull не заполняет поле next — он им только пользуется. А заполнение происходит в методе push. Метод pull «вытаскивает» первый элемент в очереди. Если я вытащил первый элемент, то теперь уже следующий за ним (второй) должен стать первым. И значит head должен указывать на следующий, после вытаскиваемого (первого). Если в очереди только один элемент, то head будет указывать на null — пустая очередь.

Reply

Февраль 11, 2016 at 00:13
Алекс says:

Уважаемый автор, мне также почему-то тяжело въехать в реализацию механизма «//Перемещаем «голову» на следующий элемент» из предыдущего урока.
Насколько я понимаю, в данном участке кода метода pull() два указанных метода выполняются одним и тем же объектом класса ObjectBox (например, объектом ob1).

// Получаем наш объект из вспомогательного класса из «головы»
Object obj = head.getObject();
// Перемещаем «голову» на следующий элемент
head = head.getNext();
// Если это был единственный элемент, то head станет равен null
// и тогда tail (хвост) тоже дожен указать на null.

Но тогда каким образом указатель head перемещается на следующий объект (например, ob2)? Это связано со спецификой хранения и обработки данных в памяти (типа после выполнения метода head.getObject() этот объект стирается?) или может быть next это какой-то оператор?

И могли бы Вы прояснить еще один вопрос: каким образом заполняется поле next для первого объекта класса ObjectBox в методе push()(если объектов больше одного), когда участок кода

```
else {
// Если это не первый элемент, то надо, чтобы последний элемент в очереди
// указывал на вновь прибывший элемент
tail.setNext(ob);
}
```

для первого объекта не выполняется, т.к. правдиво условие if (head == null) ?

Reply

- 

  Февраль 11, 2016 at 13:37
  admin says:

  Честно говоря не понял — это вопрос к какой статье ? В массивах мы об этом не говорили.

  Reply

- Февраль 11, 2016 at 15:43
  Алекс says:

  Вопрос я задавал по уроку «Пример — очередь объектов». Просто там нет возможности добавить комментарий

  Reply

  - 

    Февраль 12, 2016 at 09:31
    admin says:

    Исправил. Перенесите пожалуйста вопрос туда — так всем будет удобнее. Много у кого вопросы возникают

Reply

- 
Ноябрь 7, 2016 at 15:07
v says:

Конструкция

```
int[] sample = {12, 56, 7, 34, 89, 43, 23, 9};
```

действительно жесткая и неудобная для закачки случайных чисел (в качестве примера, например, «пузырька») можно применить

```
for (int i = 0; i < as.demo.length; i++) {
    as.demo[i] = (int) Math.round(Math.random() * 50);
}
```

Reply

- 
Январь 5, 2018 at 12:48
Blindf0ld says:

Добрый день, не совсем понятна строка: «Дальше в примере создается переменная summa, которая инициализируется нулевым значением и уже в цикле к ней прибавляется каждый элемент массива. Вот и все.» где идет пример про создание роботов внутри массива. Здесь мы считаем количество роботов внутри массива или мы должны складывать координаты всех роботов? Перед этим в примере мы искали сумму всех элементов массива. Заранее спасибо.

Reply

  - 
Январь 5, 2018 at 22:29

admin says:

Я не понял, откуда в вопросе взялись роботы 🙂 Пример инициализации — это просто пример инициаизации массива. В том числе и роботов.
Данный пример просто показывает как сложить элементы массива — узнать сумму всех элементов.
Если проводить бытовую аналогию, то узнать вес всех книг можно сложив их в пустую корзину (summa сначала равна 0) и получить общий вес.

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

+ пять = шесть

Add comment