Конспект лекций по Java. Занятие 5

В.Фесюнов

1. Массивы в Java

Мы не рассмотрели еще массивы. В Java есть как одномерные, так и многомерные массивы. Но реализация массивов в Java имеет свои особенности. Во-первых массив в Java это объект. Этот объект состоит из размера массива (поле length) и собственно массива.

Рассмотрим сначала простейшие одномерные массивы базовых типов.

int intAry[]; или int[] intAry;

Это описание переменной (или поля) intAry — ссылки на массив. Соответственно, в этом описании размер массива не указывается и сам массив не создается. Как и любой другой объект массив должен быть создан операцией **new** .

intAry = new int[10];

Создает массив из 10 целых и адрес этого массива заносит в intAry. Как обычно в Java допустимо объединять описание и инициализацию.

int[] intAry = new int[10];

Для массивов допустима инициализация списком значений.

int intAry[] = {1, 2, 3, 4};

Здесь описан массив из 4-х элементов и сразу определены их начальные значения.

Элементы массивов в Java нумеруются с 0. При обращении к элементу массивы его индексы задаются в квадратных скобках. Например.

```
x = a[3];  // x присваивается значение 3-го (4-го по порядку) элемента массива a
intAry[i] = 0;  // i — му элементу массива intAry присваивается значение 0
```

Java жестко контролирует выход за пределы массива. При попытке обратиться к несуществующему элементу массива возникает IndexOutOfBoundException.

Для двумерных массивов ставится не одна пара скобок, а две, для трехмерных — три и т.д. Например.

```
s = someAry[i][0];
tAry[i][j][k] = 10;
```

Как уже указывалось, массив является объектом, который, в частности, хранит поле length — размер массива. Это позволяет задавать обработку массивов произвольного размера. Рассмотрим пример метода, вычисляющего сумму элементов массива.

```
public double sum(double ary[]) {
      double s = 0;
       for ( int i = 0; i < ary.length; i++ ) {
              s += ary[i];
       }
       return s;
}
```

1.1. Массивы объектов

Одномерный массив объектов — это массив ссылок на объекты. Соответственно, нужно создать как массив, так и сами объекты. Наиболее частая ошибка у начинающих при работе с массивами классов примерно следующая. Создается сам массив, например,

A[] a1 = new A[10];

а потом сразу идет попытка работы с элементами этого массива. Но здесь построен только массив ссылок, а сами объекты еще не созданы.

Пусть у нас есть некоторый класс SomeClass и нужно построить массив из 4-х

объектов этого класса.

Вариант 1. (явное создание)

```
SomeClass objAry[] = new SomeClass[4];
for (int j = 0; j < 4; j++ )
     objAry[j] = new SomeClass();
```

Вариант 2. (использование списка инициализации)

```
SomeClass objAry[] = new SomeClass[] {
                                      new SomeClass(),
                                      new SomeClass(),
                                      new SomeClass(),
                                      new SomeClass(),
                                      };
```

1.2. Многомерные массивы

Они строятся по принципу "массив массивов". Разберемся с этим. Массив вляется объектом. Двумерный массив — это массив ссылок на объекты-массивы. Трехмерный массив — это массив ссылок на массивы, которые, в свою очередь, являются массивами ссылок на массивы.

Это выглядит несколько сложно. Но, к счастью, есть сокращенные варианты создания массива, которые позволяют сразу разместить все необходимые массивы ссылок. Кроме того, если удается использовать списки инициализации (а это возможно, когда размер массива заранее известен), то все гораздо проще. Пусть, например, требуется создать массив целых 3*3.

Вариант 1. (явное создание)

int ary[][] = new int[3][3];

Вариант 2. (использование списка инициализации)

```
int ary[][] = new int[][] {
```

```
                            {1, 1, 1},
                            {2, 2, 2},
                            {1, 2, 3},
                            };
```

- Внимание: в варианте 1 массив создан, но его элементы имеют неопределенное значение. Если попытаться их использовать, возникнет Exception.

Соответственно, все несколько усложняется при использовании массивов объектов. Создадим аналогичный массив объектов класса SomeClass.

Вариант 1. (явное создание)

```
SomeClass ary[][] = new SomeClass[3][3];
for ( int k = 0; k < 3; k++ )
    for ( int j = 0; j < 3; j++ )
           ary[k][j] = new SomeClass();
```

Вариант 2. (использование списка инициализации)

```
SomeClass ary[][] = new SomeClass[][] {
        { new SomeClass(), new SomeClass(), new SomeClass(), },
        { new SomeClass(), new SomeClass(), new SomeClass(), },
        { new SomeClass(), new SomeClass(), new SomeClass(), },
         };
```

Глядя на примеры со списком инициализации, можно задаться вопросом, что будет, если мы в одних строках зададим одно количество элементов, а в других — другое. Например.

```
int ary[][] = new int[][] {
                            {1, 1, 1, 1},
                            {2, 2, 2},
                            {1, 2, 3, 4, 5},
                            };
```

В Java такое допустимо и именно потому, что многомерный массив является массивом

ссылок на массивы. Т.е. каждый массив следующего уровня является самостоятельным массивом и может иметь свой размер. Причем, создание таких "непрямоугольных" массивов возможно не только с использованием списка инициализации, но и явно.

```
int ary[][] = new int[3][];
ary[0] = new int[5];
ary[1] = new int[2];
ary[2] = new int[6];
```

- Увлекаться такими "непрямоугольными" массивами не стоит. На практике очень редко встречаются задачи, в которых подобные возможности могут потребоваться. Но знать о них нужно. Хотя бы для того, чтобы понять смысл ошибки, возникшей в результате непреднамеренного создания подобного массива.

1.3. Присваивание и копирование

В приведенных выше примерах имя массива — это переменная-ссылка (или поле-ссылка), содержащая адрес объекта массива. Поэтому присваивание таких переменных друг другу — это не копирование массивов, а копирование ссылок на объекты. Например.

```
int ary[][] = new int[][] {
                          {1, 1, 1, 1},
                          {2, 2, 2},
                          {1, 2, 3, 4, 5},
                          };

int copyAry[][] = ary;
```

В данном примере все абсолютно корректно, но copyAry — это не ссылка на копию массива, а еще одна ссылка на тот же массив. Для создания копии придется написать соответствующий метод.

В состав стандартной библиотеки Java входят разнообразные средства работы с массивами. В пакете java.util имеется класс Arrays, который обеспечивает множество полезных операций над массивами (см. документацию).

1.4. Резюмируем основные правила

- **1.** Массивы являются объектами специфической формы. В частности, любой массив имеет поле length, которое определяет его размер.
- **2.** Массивы индексируются от 0.
- **3.** Java жестко контролирует выход за границы массива (прерывание IndexOutOfBoundException).
- **4.** Массив элементарного типа, например int, — это действительно массив значений (т.е. массив целых чисел). Массив объектов — это массив ссылок на объекты. Т.е. недостаточно создать сам массив, нужно еще создать объекты, входящие в него.
- **5.** Существуют два способа создания массива — операцией new и явной инициализацией.
- **6.** Для многомерных массивов существует возможность задания разного размера массивов второго, третьего и т.д. измерений, но это "экзотика".

2. Конструкторы классов

Мы уже познакомились с понятием класса в Java. Мы определили, что класс является как бы шаблоном для создания объектов класса (экземпляров класса) и что класс состоит из полей и методов класса. Но мы еще не рассмотрели очень важную составляющую классов — **конструкторы класса** .

Конструктор класса — это специальный метод класса. Он вызывается при создании объекта класса. Конструкторы (в своем описании) отличаются от других методов класса тем, что их имя совпадает с именем класса. Кроме того, при описании любого метода класса, кроме конструктора, мы обязаны указать тип возвращаемого значения, а если метод не возвращает никакого значение, то мы должны вместо типа явно указать void. При описании конструктора тип возвращаемого значения вообще не указывается.

Разберем более подробно, что происходит при создании объекта класса.

- Ищется класс объекта среди уже используемых в программе классов. Если его нет, то он ищется во всех доступных программе каталогах и библиотеках. После обнаружения класса в каталоге или библиотеке выполняется создание и инициализация статических полей класса. Т.е. для каждого класса статические поля

инициализируются только один раз.
- Выделяется память под объект.
- Выполняется инициализация полей класса.
- Отрабатывает конструктор класса.

Это не полная схема, а упрощенная.

Итак, что такое **конструктор** .

- 1. Это специальный метод класса.
- 2. Его имя совпадает с именем класса.
- 3. Конструктор не возвращает никакого значения.
- 4. Конструктор, как и любой другой метод, может иметь параметры.
- 5. Конструктор без параметров называется конструктором по умолчанию (default constructor).
- 6. В классе может быть несколько конструкторов. В этом случае они должны иметь разные наборы параметров.
- 7. Если в классе нет ни одного конструктора, то генерируется пустой конструктор по умолчанию. Если в классе есть хотя бы один конструктор, то конструктор по умолчанию не генерируется.

До сих пор в примерах мы использовали примерно следующий синтаксис создания объектов.

SomeClass obj = new SomeClass();

В данном случае создается объект и при его создании используется конструктор без параметров (конструктор по умолчанию). Но возможен и другой вариант.

SomeClass obj = new SomeClass(1, 'a');

Здесь при создании объекта вызывается конструктор с двумя параметрами. Т.е. в классе SomeClass должен быть описан конструктор, имеющий один арифметический параметр (например, int) и один символьный параметр.

Для того чтобы обе вышеприведенные строки были корректными, описание класса SomeClass должно выглядеть примерно так.

```
class SomeClass {
```

```
   . . .
    public SomeClass() {     // это конструктор по умолчанию
         . . .
    }

    SomeClass( int a, char c) {   // это конструктор с двумя параметрами
         . . .
    }
   . . .
}
```

2.1. Вызов одного конструктора из другого

В Java есть одна удобная возможность, позволяющая сократить суммарный объем кода конструкторов. Обычно все конструкторы класса имеют общую часть кода, поскольку они должны выполнять одинаковые действия, которые отличаются только некоторыми деталями. Можно вынести эту общую часть кода в отдельный метод и вызывать его из всех конструкторов. Но есть и другая возможность. В Java можно вызвать один конструктор из другого. Для этого используется ключевое слово **this** . Рассмотрим это на примере.

```
public class Point {
     private double x, y;

     public Point(double x, double y) {
           this.x = x;
           this.y = y;
     }

     public Point() {
           this(0, 0);     // вызов конструктора с двумя параметрами
     }
     . . .
}
```

В классе Point (точка) имеется два конструктора. Один, более общий, с двумя параметрами, предназначен для конструирования точки плоскости с заданными

координатами. Другой, без параметров, строит точку с координатами (0, 0). В нем для экономии кода просто вызывается первый из конструкторов с параметрами (0, 0).

Есть одно ограничение на вызов одного конструктора из другого. Такой вызов должен быть первым оператором в вызывающем конструкторе.

3. Работа со строками (класс String)

Первый стандартный класс Java, который мы рассмотрим — это класс String. Этот класс определен в стандартной библиотеке Java. Он используется для работы со строками.

Откроем документацию по классу String. Во-первых, обратим внимание на конструкторы класса. В документации конструкторы класса описываются сразу после описания полей класса. Конструкторы класса String предоставляют широкие возможности конструирования строк.

public String() Создает пустую строку

public String(char[] value) Создает строку из массива символов.

public String(byte[] bytes) Создает строку из массива байт, преобразуя байты в символы в соответствии с кодировкой по умолчанию.

Есть и другие конструкторы класса String.

В силу важности строк в Java для класса String существуют расширенные возможности языка. По общим правилам создания объектов мы должны были бы при построении строки писать так

String str = new String("какая-то строка");

Такая запись допустима, но существует ее упрощенный вариант:

String str = "какая-то строка";

Для строк определена операция сложения, которая означает конкатенацию (сцепление) строк. Определена операция сложения строки с числом. При этом сначала число преобразуется в строку, а потом выполняетс конкатенация полученных строк.

Также определена операция сложения строки с любым объектом. Она выполняется так. Сначала для этого объекта вызывается метод toString(), потом выполняется конкатенация полученных строк. Метод toString() есть у всех объектов Java (рассмотрим подробнее при изучении наследования).

Примеры сложения строк с числами нам уже встречались — в операторах типа

System.out.println("результат=" + x);

Вернемся к документации по классу String. Следует обратить внимание на следующие методы этого класса.

public char charAt(int index) Выбирает из строки символ с индексом index (символы индексируются от нуля).

public int compareTo(String anotherString) Сравнивает строку с другой строкой

public int indexOf(int ch) Ищет символ в строке

public int indexOf(String str) Ищет указанную параметром строку в данной

public int length() Возвращает длину строки

public String substring(int beginIndex, int endIndex) Выделяет подстроку из строки

public String trim() Удаляет из строки начальные и концевые пробелы

Набор методов valueOf(...) позволяет переводить значения различных типов в строки.

4. Практическая работа

Мы изучили достаточно много материала по Java и теперь можем применить это на практике. В частности, теперь можно более конкретно рассмотреть различия между статическими и обычными методами классов.

Вернемся к примеру с точкой плоскости (класс Point). Реализуем в этом классе ряд методов для работы с точками плоскости.

- 1. Расстояние между двумя точками.
- 2. Расстояние от данной точки до другой точки.
- 3. Расстояние от данной точки до начала координат.

- 4. Сдвиг точки по оси X на заданную величину.
- 5. Такой же сдвиг по оси Y.
- 6. Два метода для получения X- и Y-координаты точки.