

Массивы Java

- В этой лекции рассмотрено устройство массивов в Java
- Они представляют собой набор значений одного типа
- Основным свойством массива является длина, которая в Java может равняться нулю
- Массив обладает элементами в количестве, равном длине, к которым можно обратиться, используя индекс, изменяющийся от 0 до величины длины
- Длина задается при создании массива и у созданного массива не может быть изменена.
- Многомерные массивы могут быть не только "прямоугольными", но и любой конфигурации
- Изучается механизм клонирования, позволяющий создавать точные копии объектов

Темы лекции 5. Основные языковые конструкции Java

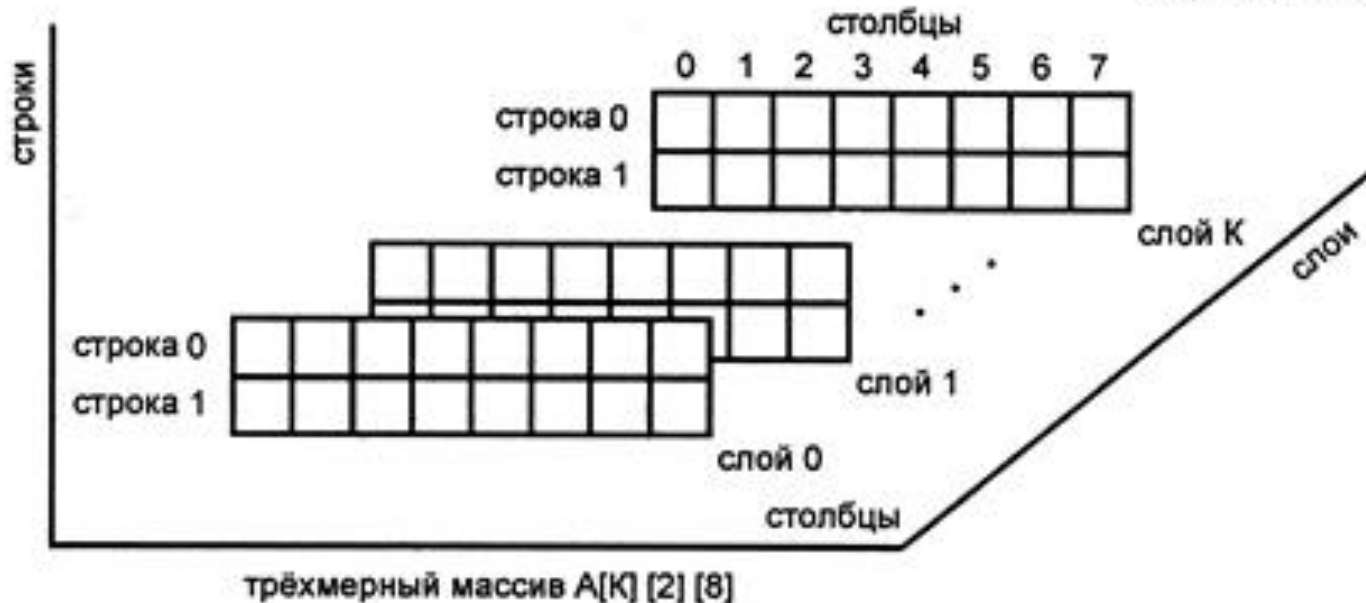
- Массивы как тип данных в Java
- Объявление массивов
- Инициализация массивов
- Многомерные массивы
- Класс массива
- Ошибка `ArrayStoreException`
- Переменные типа массив и их значения
- Клонирование массивов

Массивы как тип данных в Java

- В отличие от обычных переменных, которые хранят только одно значение, массивы (arrays) используются для хранения целого набора значений
- Количество значений в массиве называется его длиной, сами значения – элементами массива
- Значений может не быть вовсе, в этом случае массив считается пустым, а его длина равной нулю
- Массивы в Java являются объектами



Массивы как тип данных в Java



Объявление массивов



Так

```
int A[];
```

или так:

```
int[] A;
```

Количество пар квадратных скобок указывает на размерность массива. Для многомерных массивов допускается смешанная запись:

```
int[] A[];
```

Переменная `A` имеет тип "двумерный массив, основанный на `int`".

Объявление массивов

Чтобы создать экземпляр массива, нужно воспользоваться ключевым словом `new`, после чего указывается тип массива и в квадратных скобках – длина массива.

```
int A[]=new int[5];
```

Переменная инициализируется **ссылкой**, указывающей на только что созданный массив. После его создания можно обращаться к элементам, используя ссылку на массив, далее в квадратных скобках указывается индекс элемента.

Объявление массивов

Индекс меняется от нуля, пробегая всю длину массива, до максимально допустимого значения, на единицу меньшего длины массива.

```
int array[]=new int[5];  
for (int i=0; i<5; i++)  
    array[i]=i*i;  
for (int j=0; j<5; j++)  
    System.out.println(j+"*"+j+"="+array[j]);
```

0*0=0

1*1=1

2*2=4

3*3=9

4*4=16

Объявление массивов

Поскольку для экземпляра массива длина является постоянной характеристикой, для всех массивов существует специальное поле `length`, позволяющее узнать ее значение. Например:

```
Point p[]=new Point[5];  
for (int i=0; i<p.length; i++)  
    p[i]=new Point(i, i);
```

Значение индекса массива **всегда** имеет тип `int`.

Инициализация массивов

Обращение к неинициализированному массиву:

```
Point p[]=new Point[5];  
for (int i=0; i<p.length; i++)  
    System.out.println(p[i]);
```

Результатом будут лишь слова null.

Инициализация элементов массива значениями:

```
Point p[]=new Point[5];  
for (int i=0; i<p.length; i++)  
{  
    p[i].x = i;  
    p[i].y = i*2;  
}
```

Инициализация массивов

Явная инициализация массива записывается следующим образом:

```
int i[]={1, 3, 5};  
int j[]={};    // {} эквивалентно new int[0]  
System.out.println(Arrays.deepToString(i));
```

Длина массива вычисляется автоматически, исходя из количества введенных значений. Далее создается массив такой длины и каждому его элементу присваивается указанное значение.

Инициализация массивов

Можно породить массивы на основе объектных типов, например:

```
Point p=new Point(1, 3);  
Point arr[]={p, new Point(2, 2), null, p};
```

Или

```
String[] catsName = {    "Васька",  
                        "Кузя",  
                        "Барсик",  
                        "Мурзик",  
                        "Леопольд",  
                        "Кот "+"Бегемот",  
                        "Рыжик",  
                        "Матроскин"  
                        };
```

Инициализация массивов

```
int size = 4;  
boolean[] test1 = new boolean[size];  
int[] test2 = new int[size];  
String[] test3 = new String[size];  
Arrays.fill(test1, true); // присваиваем всем true  
Arrays.fill(test2, 9); // присваиваем всем 9  
Arrays.fill(test3, "Мяу!"); // Ну вы поняли
```

[true, true, true, true]

[9, 9, 9, 9]

[Мяу!, Мяу!, Мяу!, Мяу!]

Сравнение массивов

```
// Создаем два массива
int[] a1 = new int[10];
int[] a2 = new int[10];
// заполняем их девятками
Arrays.fill(a1, 9);
Arrays.fill(a2, 9);
System.out.println("Сравним: " + Arrays.equals(a1,
a2));
```

При сравнении мы получим **true**.

//Изменим один элемент у второго массива

```
a2[3] = 11;
System.out.println("Сравним: " + Arrays.equals(a1,
a2));
```

Теперь при сравнении будет выдаваться **false**.

Многомерные массивы

	Столбец 0	Столбец 1	Столбец 2	Столбец 3
Строка 0	<code>a [0] [0]</code>	<code>a [0] [1]</code>	<code>a [0] [2]</code>	<code>a [0] [3]</code>
Строка 1	<code>a [1] [0]</code>	<code>a [1] [1]</code>	<code>a [1] [2]</code>	<code>a [1] [3]</code>
Строка 2	<code>a [2] [0]</code>	<code>a [2] [1]</code>	<code>a [2] [2]</code>	<code>a [2] [3]</code>

Имя массива
Индекс строки
Индекс столбца

```
int a[][]=new int[3][4];
```

Переменная **a** ссылается на двумерный массив, который можно представить себе в виде таблицы 3x5 (3 строки X 4 столбца).

Многомерные массивы

Пример заполнения двумерного массива через цикл:

```
int pithagor_table[][]=new int[5][5];
for (int i=0; i<5; i++)
{
    for (int j=0; j<5; j++)
    {
        pithagor_table[i][j]=i*j;
        System.out.print(pithagor_table[i][j]+" ");
    }
    System.out.println();
}
```

Результатом выполнения программы будет:

0	0	0	0	0
0	1	2	3	4
0	2	4	6	8
0	3	6	9	12
0	4	8	12	16

Многомерные массивы

Используя **x** и **одно число в паре квадратных скобок**, можно обратиться к одномерному массиву, который является элементом двумерного массива. Его можно проинициализировать новым массивом с некоторой другой длиной и таблица перестанет быть прямоугольной – она примет произвольную форму.

```
int x[][]=new int[3][5]; //      прямоугольная
                             таблица

x[0]=new int[7];
x[1]=new int[0];
x[2]=null;
```

Создается один массив массивов (один объект) и три массива чисел, каждый длиной 5 (три объекта). Итого, четыре объекта.

Класс массива

```
[public] class A implements Cloneable,  
java.io.Serializable  
{  
    public final int length;    // инициализируется  
при создании  
    public Object clone()  
    {  
        try  
        { return super.clone(); }  
        catch (CloneNotSupportedException e)  
        { throw new InternalError(e.getMessage()); }  
    }  
}
```

} Таким образом, экземпляр типа массив является полноценным объектом, который, в частности, наследует все методы, определенные в классе Object, например, toString(), hashCode()

Преобразование типов для массивов

- Преобразования между типами массивов, основанных на различных примитивных типах, невозможны ни при каких условиях.

// вызовет ошибку компиляции

```
byte b[]={1, 2, 3};
```

```
int i[]=b;
```

Если требуется копирование элементов, то нужно сначала создать новый массив, а затем воспользоваться стандартной функцией `System.arraycopy()`

Преобразование типов для массивов

- Для ссылочных типов такого строгого правила нет. Например, если создать экземпляр массива, основанного на типе `Child`, то ссылку на него можно привести к типу массива, основанного на типе `Parent`.

```
Child c[] = new Child[3];
```

```
Parent p[] = c;
```

- Универсальное правило: массив, основанный на типе `A`, можно привести к массиву, основанному на типе `B`, если сам тип `A` приводится к типу `B`.

// если допустимо такое приведение:

```
B b = (B) new A();
```

// то допустимо и приведение массивов:

```
B b[]=(B[]) new A[3];
```

Ошибка `ArrayStoreException`

```
Child c[] = new Child[5];
```

```
Parent p[]=c;
```

```
p[0]=new Parent();
```

- С точки зрения компилятора код корректен.
- Преобразование во второй строке допустимо.
- В третьей строке элементу массива типа `Parent` присваивается значение того же типа.
- Виртуальная машина при выполнении программы всегда осуществляет дополнительную проверку перед присвоением значения элементу массива.

Переменные типа массив и их значения

Тип переменной	Допустимые типы ее значения
Массив простых чисел	<ul style="list-style-type: none">• null• в точности совпадающий с типом переменной
Массив ссылочных значений	<ul style="list-style-type: none">• null• совпадающий с типом переменной• массивы ссылочных значений, удовлетворяющих следующему условию: если тип переменной – массив на основе типа А, то значение типа массив на основе типа В допустимо тогда и только тогда, когда В приводимо к А
Object	<ul style="list-style-type: none">• null• любой ссылочный, включая массивы

Клонирование массивов

- Механизм клонирования позволяет породить новые объекты на основе существующего, которые обладали бы точно таким же состоянием, что и исходный.

Истинные выражения:

```
x == x.clone()
```

```
x.clone().getClass() == x.getClass()
```

```
x.equals(x.clone())
```

- Класс `Object` содержит метод `clone()`:

```
protected native Object clone()
```

```
throws CloneNotSupportedException;
```

Клонирование массивов

- При выполнении метода `clone()` сначала проверяется, можно ли клонировать исходный объект.
- Если разработчик хочет сделать объекты своего класса доступными для клонирования через `Object.clone()`, то он должен реализовать в своем классе интерфейс `Cloneable`, который служит признаком для виртуальной машины, что объекты могут быть клонированы.
- Если проверка не выполняется успешно, метод порождает ошибку `CloneNotSupportedException`.

Клонирование массивов

- При выполнении метода `clone()` сначала проверяется, можно ли клонировать исходный объект.
- Если разработчик хочет сделать объекты своего класса доступными для клонирования через `Object.clone()`, то он должен реализовать в своем классе интерфейс `Cloneable`, который служит признаком для виртуальной машины, что объекты могут быть клонированы.
- Если проверка не выполняется успешно, метод порождает ошибку `CloneNotSupportedException`.

Клонирование массивов

```
public class Test implements Cloneable {
    Point p;
    int height;
    public Test(int x, int y, int z)
    { p=new Point(x, y);
      height=z;  }
    public static void main(String s[])
    {      Test t1=new Test(1, 2, 3), t2=null;
          try { t2=(Test) t1.clone(); }
          catch (CloneNotSupportedException e) {}
          t1.p.x=-1;
          t1.height=-1;
          System.out.println("t2.p.x=" + t2.p.x + ",
t2.height=" + t2.height);}}
```

Клонирование массивов

```
public class Test implements Cloneable {
    Point p;
    int height;
    public Test(int x, int y, int z)
    { p=new Point(x, y);
      height=z;  }
    public static void main(String s[])
    {      Test t1=new Test(1, 2, 3), t2=null;
          try { t2=(Test) t1.clone(); }
          catch (CloneNotSupportedException e) {}
          t1.p.x=-1;
          t1.height=-1;
          System.out.println("t2.p.x=" + t2.p.x + ",
t2.height=" + t2.height);}}
t2.p.x=-1, t2.height=3
```

Клонирование массивов

```
int a[]={1, 2, 3};  
int b[]=(int[])a.clone();  
a[0]=0;  
System.out.println(b[0]);
```

1

Клонирование массивов

```
int a[][]={{1, 2}, {3}};  
int b[][]=(int[][] ) a.clone();  
if (...) {  
    // первый вариант:  
    a[0]=new int[]{0};  
    System.out.println(b[0][0]);  
} else {  
    // второй вариант:  
    a[0][0]=0;  
    System.out.println(b[0][0]);}
```

- В первом случае в исходном массиве меняется ссылка, хранящаяся в первом элементе, что не принесет никаких изменений для клонированного объекта. На консоли появится 1.
- Во втором случае модифицируется существующий массив, что скажется на обоих двухмерных массивах. На консоли появится 0.