Программирование
Тема 3.1 Массивы в Java

Одномерные массивы примитивных типов

- **Объявление ссылки на массив:**

```
базовый_тип[] имя_массива;
```

Например: `int[] arr;`

- **Выделение памяти и инициализация массива:**

```
имя_массива = new базовый_тип [размер];
```

1. `arr = new int[10]; // инициализируется значениями 0`
2. `arr = new int[] {4,7,9,12,3,8,9,2,6,1};`
3. `int N = 10; arr = new int[N];`

```
for(int i=0; i< arr.length; i++) arr[i] = i+1;
```

- **Совмещение объявления, выделения памяти и инициализации:**

```
int[] arr = new int[10];
int[] arr = new int[] {4,7,9,12,3,8,9,2,6,1};
int[] arr = {4,7,9,12,3,8,9,2,6,1}; // new подразумевается
```

Массивы в Java

- Массив — набор элементов одного и того же типа, объединенных общим именем.
- Массивы в Java относятся к ссылочным типам данных. Имя массива является ссылкой на область кучи (динамической памяти), в которой последовательно размещается набор элементов определенного типа. Выделение памяти под элементы массива выполняется с помощью операции new, а за освобождением памяти следит сборщик мусора.
- Рассмотрим следующие типы массивов: одномерные массивы примитивных типов, одномерные массивы объектов и многомерные массивы (двумерные прямоугольные и двумерные ступенчатые).
- Одномерный массив – это фиксированное количество элементов одного и того же типа, объединенных общим именем, где каждый элемент имеет свой номер. Нумерация элементов массива в Java начинается с нуля.
- Одномерный массив в Java реализуется как объект, поэтому его создание состоит из двух этапов. Сначала объявляется ссылочная переменная типа массив, затем выделяется память под требуемое количество элементов базового типа, и ссылочной переменной присваивается адрес нулевого элемента в массиве. Базовый тип определяет тип данных каждого элемента массива. Количество элементов, которые будут храниться в массиве, определяется размером массива. Размер массива может быть задан переменной, но массив не может иметь переменный размер, то есть переменная задающая размер массива должна получить значение до создания массива. Размер массива может быть запрошен через поле length. Поле length. является final, потому выставить через него новый размер массива, увы, не получится.
- В Java элементам массива присваиваются начальные значения по умолчанию в зависимости от базового типа. Для арифметических типов – нули, для ссылочных типов – null, для символов - символ с кодом ноль. Кроме того, можно произвести инициализацию массива с использованием списка инициализации, который размещается в фигурных скобках или поэлементно в цикле.
- При необходимости, этапы объявления ссылочной переменной типа массив, и выделения необходимого объема памяти могут быть объединены в один.
- JVM проверяет выход за границы массива, и в случае необходимости генерирует исключение: ArrayIndexOutOfBoundsException;

Одномерные массивы объектов

- **Пример: Создание массива объектов класса java.util.Date**

```
import java.text.SimpleDateFormat;
import java.util.Date;
class Test {
 public static void main(String args[])
  { Date[] dates = new Date[3];//создание массива ссылок
    for(int i=0;i<dates.length;i++)
         dates[i] = new Date(); //создание объектов
    SimpleDateFormat format1 = new SimpleDateFormat("dd.MM.yyyy");
    for(int i=0;i<dates.length;i++)
       System.out.println(format1.format(dates[i]));
  }
}
```

```
03.03.2015
03.03.2015
03.03.2015
```

Массивы в Java

- Большая часть из того, что сказано о массивах примитивных типов, верна и для массивов объектов. Их размер может быть получен через .length, выход за границы контролируется.
- Создание массива объектов состоит из двух этапов. Сначала с помощью операции new отводится память под массив ссылок на объекты, затем в цикле с помощью операции new выделяется память под объекты.
- В примере на слайде создается массив объектов класса java.util.Date. Если цикл:

```
for(int i=0;i< dates.length;i++)
    dates[i] = new Date();
```

- будет пропущен, то будет ошибка `NullPointerException`, которая означает, что массив ссылок на объекты создан и его элементы инициализированы значением null, однако сами объекты не созданы.

Оператор цикла for в стиле forech

```
for(тип итер_пер : массив) блок_операторов
```

- **Пример:**

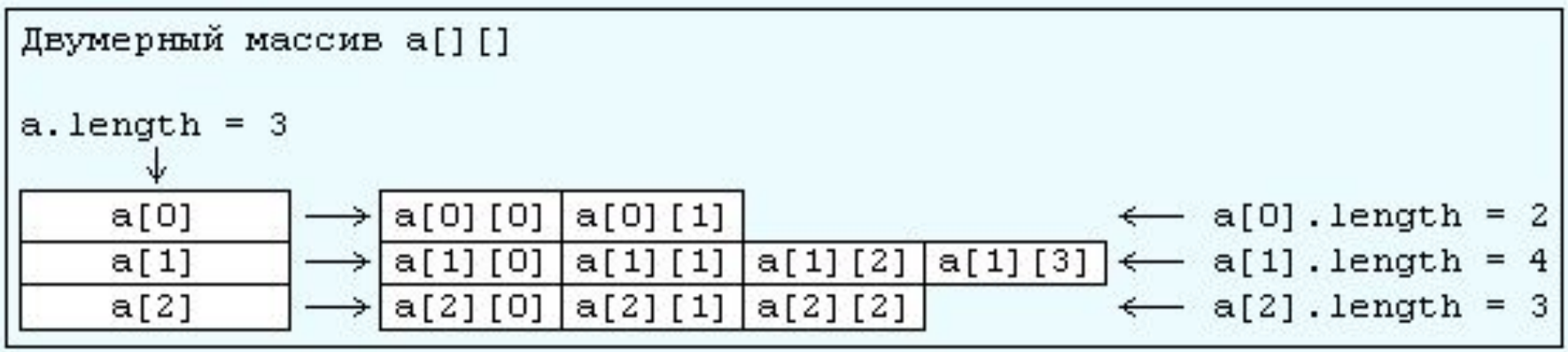
```
import java.util.Random;
public class Test
{ public static void main (String args[])
    { int[] a = new int [5];
      Random random = new Random();
      for (int i = 0; i < a.length; i++)
         a[i] = random.nextInt(50);
      for (int elem : a)
         System.out.print(elem + " ");
      System.out.println();
    }
}
```

```
18 19 26 42 10
```

Массивы в Java

- Во многих языках программирования существует более компактная форма оператора цикла for для перебора элементов массивов и элементов коллекций - foreach.
- В Java решили не добавлять новое ключевое слово, а просто сделали усовершенствованный вид цикла for.
- Оператор for в стиле foreach перебирает элементы массива последовательно, начиная с первого и заканчивая последним.
- В примере на слайде при прохождении цикла итерационной переменной elem автоматически присваивается значение, равное значению следующего элемента массива a.
- В цикле в стиле foreach итерационная переменная доступна только для чтения, т. е. с ее помощью нельзя изменить элемент массива.

Многомерные массивы



- **Пример**: Способы создания двумерного массива

1. Создание прямоугольного двумерного массива

```
int[][] a = new int[10][5];
```

2. Создание ступенчатого двумерного массива

```
int[][] a = new int[3][]; // line 1
a[0] = new int[2]; // line 2
a[1] = new int[4]; // line 3
a[2] = new int[3]; // line 4
```

Массивы в Java

- Для простоты начнем с двумерных массивов, а потом уже перейдем к n-мерным.
- Для двумерного массива не существует такого понятия как его размеры. Можно определить размер массива только по первому индексу. Причина кроется в организации массива. Он представляет собой массив ссылок на массивы, каждый из которых содержит реальные данные. И эти массивы могут иметь разную длину!
- Рассмотрим двумерный массив на слайде. Его длина по первому индексу равна 3. Ее можно получить через a.length. Элементами этого массива являются ссылки на массивы с данными. Длина каждого из этих массивов может быть получена так же через .length. Доступ к каждому из этих массивов осуществляется через его индекс – первый индекс в массиве a.
- Теперь понятно, что можно говорить только о длине двумерного массива по первому индексу. Максимальное значение второго индекса может быть получено только для каждого конкретного значения первого индекса.
- Стандартный способ создания двумерного массива – это использование обычной конструкции:

```
int[][] array1 = new int[10][5];
```

- Эта конструкция создает двумерный массив размером 10x5. О размерах тут можно говорить только потому, что размер по второму индексу явно указан при инициализации. И все строки будут длины 5, массив будет прямоугольным.
- Есть, однако, и другой способ. Для примера создадим массив, приведенный на рисунке слайда. Обратите внимание, в строке 1 при инициализации массива не указана размерность по второму индексу! В строке 1 создается массив по первому индексу, размером 3. Это означает, что выделяется память под три ссылки на массивы-строки. Сами массивы не создаются, выделенная память инициализирована значениями null. В строке 2 создается массив длиной 2. Ссылка на этот массив сохраняется в первом элементе массива по первому индексу. Точно так же в строках 3 и 4 создаются массивы, ссылки на которые сохраняются во втором и третьем элементах массива по первому индексу. В итоге мы имеем массив произвольной формы (ступенчатый массив). Разумеется, мы могли бы создать все три массива-строки одинаковой длины, скажем, 5. Эти действия были бы полностью эквивалентны конструкции `int[][] a = new int[10][5];`

Многомерные массивы

- **Пример: Инициализация двумерных массивов**

```
package primer3;
import java.util.Arrays;
public class Primer3 {
    public static void main(String[] args) {
        int n = 2, m = 3;
        int [][] b = new int [n][m];
        for (int i = 0; i < b.length; i++)
         { for (int j = 0; j < b[i].length; j++)
              System.out.print(b[i][j] + " ");
          System.out.println();
         }
        int [][] c = {{1, 2, 3},{4, 5, 6}};
        int [][] d = new int [][]  {{1, 2, 3},{4, 5, 6}};
        for (int i = 0; i < d.length; i++)
          System.out.println(Arrays.toString(d[i]));
        System.out.println();
         int [][] e = new int [][]  {{1, 2, 3},{4, 5, 6, 7}};
         for (int i = 0; i < e.length; i++)
          System.out.println(Arrays.toString(e[i]));
    }
}
```

```
run:
0 0 0
0 0 0
[1, 2, 3]
[4, 5, 6]

[1, 2, 3]
[4, 5, 6, 7]
```

- **Некоторые методы класса java.util Arrays**
  - **Методы сортировки:**

```
static void sort(type[] a)   // type может быть byte, short, int, long,
                             // char, float, double или тип Object
static void sort(type[] a, int from, int to)
static void sort(Object[] a, Comparator c)
static void sort(Object[] a, int from, int to, Comparator c)
```

  - **Методы бинарного поиска:**

```
static int binarySearch(type[] a, type element)
static int binarySearch(Object[] a, Object element,
Comparator c).
```

  - **Методы заполнения массива:**

```
static void fill(type[], type value)
static void fill(type[], int from, int to, type value)
```

  - **Методы сравнения массивов:**

```
static boolean equals(type[] al, type[] a2)
```

Массивы в Java

- В классе Arrays из пакета java.util собрано множество методов для работы с массивами. Их можно разделить на четыре группы.
- Восемнадцать статических методов сортируют массивы с разными типами числовых элементов в порядке возрастания чисел или просто объекты в их естественном порядке.
- Восемь из них имеют простой вид static void sort(type[] a), где type может быть один из семи примитивных типов byte, short, int, long, char, float, double или тип Object .
- Восемь методов с теми же типами сортируют часть массива от индекса from включительно до индекса to исключительно: static void sort(type[] a, int from, int to).
- Оставшиеся два метода сортировки упорядочивают массив или его часть с элементами типа Object по правилу, заданному объектом с, реализующим интерфейс Comparator.
- После сортировки можно организовать бинарный поиск в массиве одним из девяти статических методов поиска. Восемь методов имеют вид static int binarySearch(type[] a, type element), где type — один из тех же восьми типов. Девятый метод поиска имеет вид static int binarySearch(Object[] a, Object element, Comparator c). Он отыскивает элемент element в массиве, отсортированном в порядке, заданном объектом с.
- Методы поиска возвращают индекс найденного элемента массива. Если элемент не найден, то возвращается отрицательное число
- Восемнадцать статических методов заполняют массив или часть массива указанным значением value.
- Наконец, девять статических логических методов сравнивают массивы.
- Массивы считаются равными, и возвращается true , если они имеют одинаковую длину и равны элементы массивов с одинаковыми индексами.

Массивы в Java

- **Пример.**

```
run:
34 -45 0 0 0 0
34 -45 12 67 -24 45
массивы не равны
```

```
package arrtest;
import java.util.Arrays;
class ArrTest
{ public static void main(String[] args)
  { int[] a1 = new int [] {34, -45, 12, 67, -24, 45};
    int[] a2 = new int [6];
    System.arraycopy(a1, 0, a2, 0, a1.length);
    int k = Arrays.binarySearch(a1, 12);
    Arrays.fill(a1, k, a1.length, 0);
    for (int i = 0; i < a1.length; i++)
        System.out.print(a1[i] + " ");
     System.out.println();
    for(int elem : a2)
       System.out.print(elem + " ");
    System.out.println();
    boolean fl = Arrays.equals(a1, a2);
     if (fl == true) System.out.println("массивы равны");
        else System.out.println("массивы не равны");
  }
}
```

Массивы в Java

Пример.

```
package arrtest;
import java.util.Arrays;
import java.util.Collections;
class ArrTest
{ public static void main(String[] args)
  { int[] a1 = new int [] {34, -45, 12, 67, -24, 45};
   System.out.println(Arrays.toString(a1));
   Arrays.sort(a1);
   System.out.println(Arrays.toString(a1));
   for (int i = 0, j = a1.length - 1; i < j; i++, j--)
   {  int t = a1[i];
      a1[i] = a1[j];
      a1[j] = t;
   }
   System.out.println(Arrays.toString(a1));
    Integer[] a = new Integer[] { 1, 2, 3 };
    System.out.println(Arrays.toString(a));
    Arrays.sort(a, Collections.reverseOrder());
    System.out.println(Arrays.toString(a));
    String[] names = {"Zoe", "Alison", "David"};
    Arrays.sort(names);
    System.out.println(Arrays.toString(names));
   }
}
```

```
run:
[34, -45, 12, 67, -24, 45]
[-45, -24, 12, 34, 45, 67]
[67, 45, 34, 12, -24, -45]
[1, 2, 3]
[3, 2, 1]
[Alison, David, Zoe]
```

Контрольные вопросы

1. Понятие массива. Одномерные массивы в Java: синтаксис описания и примеры использования.
2. Двумерные массивы в Java: синтаксис описания и примеры использования.
3. Класс Arrays в Java: назначение, примеры использования основных методов