ЛАБОРАТОРНАЯ РАБОТА 2

Тема: Массивы в программах на Java

Под массивом подразумевают набор однотипных данных (переменных). Переменные, относящиеся к одному массиву, называются элементами этого массива. К элементам обращаются по общему имени и индивидуальному набору целочисленных индексов. Массивы бывают статическими и динамическими. Под статические массивы память выделяется при компиляции программы. Для динамических массивов память выделяется в процессе выполнения программы. В Java все массивы являются динамическими. Количество индексов, необходимых для идентификации элемента массива, называется размерностью массива.

Одномерный массив — это такой массив, в котором идентификация элементов осуществляется с помощью одного индекса. Для объявления одномерного массива необходимо задать тип, к которому относятся элементы массива, название массива, а также количество элементов, входящих в массив. Синтаксис объявления одномерного массива имеет вид:

```
тип[] имя=new тип[размер];
```

Вначале указывается тип элементов массива, после которого ставятся пустые квадратные скобки. Далее следует имя массива, оператор присваивания, инструкция динамического выделения памяти `new` , снова тип элементов массива и в квадратных скобках размер массива (количество элементов в массиве). Приведенный процесс объявления массива можно разбить на две команды:

```
тип[] имя;
имя=new тип[размер];
```

Обращение к элементу одномерного массива осуществляется через имя массива с указанием в квадратных скобках индекса элемента. Индексация элементов массива начинается с нуля. Например, ссылка на первый элемент массива `nums` будет иметь вид `nums[0]`. Если в массиве 100 элементов, то последний элемент массива имеет индекс 99. Длину массива можно узнать с помощью свойства `length` — переменной, которая создается при объявлении массива, и ее значением является количество элементов массива. Например, длину массива `nums`, можно узнать через `nums.length`, обратиться к последнему элементу этого массива можно через запись `nums[nums.length-1]`.

Ниже приведен пример объявления, инициализации и использования массивов:

Пример 1. Объявление и инициализация одномерного массива

```
class MyArray{
      public static void main(String[] args){
      // Индексная переменная и размер массива
            int i,n;
            // Объявление переменной массива
            int[] data;
            // Прямая инициализация элементов массива:
            data=new int[]{3,8,1,7};
            // Длина второго массива
            n=data.length;
            // Объявление второго массива
            int[] nums=new int[n];
```

```
        // Заполнение второго массива в цикле
        for(i=0;i<n;i++){
              nums[i]=2*data[i]-3;
              System.out.println("nums["+i+"]="+nums[i]);}
        }
    }
```

В Java массивы могут быть не только одномерными, но и с большей размерностью. С практической точки массивы размерности выше второй используют редко. Двухмерный массив в Java технически реализуется одномерным массивом, элементами которого являются также одномерные массивы. Это дает определенную гибкость в использовании двухмерных массивов. Для объявления двухмерных массивов используются две пары квадратных скобок и синтаксис объявления может быть следующим:

```
тип[][] имя=new тип[размер_1][размер_2];
```

Как и в случае одномерного массива, данная команда представляет собой объединение двух отдельных команд:

```
тип[][] имя;
имя=new тип[размер_1][размер_2];
```

Например, командой `double[][] data=new double[3][4]` создается массив с именем `data`. Элементами массива являются значения типа `double`. Размер массива по первому индексу равен 3, а по второму — 4.

Для инициализации двухмерного массива используют вложенные инструкции цикла или список значений, заключенный в фигурные скобки. Элементами списка являются заключенные в фигурные скобки списки значений элементов по каждому из индексов. Например, инициализация двухмерного массива с помощью списка значений:

```
double data[][]={{0.1,0.2,0.3},{0.4,0.5,0.6}};
```

Этой командой создается и инициализируется двухмерный массив `data` размерами 2 на 3 (по первому индексу размер массива 2, по второму индексу — 3). Другими словами, массив `data` — это массив из двух элементов, которые, в свою очередь, являются массивами из трех элементов. Так, элемент `data[0][0]` получает значение 0.1 , элемент `data[0][2]` — значение 0.3 , элемент `data[1][0]` — значение 0.4 , а элемент `data[1][2]` — значение 0.6 .

Другой пример позволяет создать двумерный массив с различной размерностью строк:

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

Целочисленный массив состоит из двух элементов-массивов, первый из которых имеет размерность 3, а второй — 2. В Java двухмерные массивы не обязаны быть прямоугольными, то есть размерности по двум индексам могут быть разными. В данном случае элемент `nums[0][0]` имеет значение 1, элемент `nums[0][1]` — значение 2, элемент `nums[0][2]` — значение 3, элемент `nums[1][0]` — значение 4, элемент `nums[1][1]` — значение 5, а элемента `nums[1][2]` не существует.

Ниже приведен пример 2, в котором создается двухмерный массив и инициализируется с помощью вложенных инструкций цикла.

Пример 2. Объявление и инициализация двухмерного массива

```
class MyDArray{
      public static void main(String[] args){
            // Индексные переменные и размерность массива:
            int i,j,n=3;
            // Создание двухмерного массива:
            int[][] nums=new int[n-1][n];
            // Вложенные инструкции цикла:
            for(i=0;i<n-1;i++){
                  for(j=0;j<n;j++){
                  // Заполнение элементов массива:
                        nums[i][j]=10*(i+1)+j+1;
                        // Вывод значений в одну строку:
                        System.out.print(nums[i][j]+" ");}
            // Переход на новую строку
            System.out.println();}
      }
}
```

В примере 3 приведен код программы, в которой создается двухмерный «непрямоугольный» массив.

Пример 3. Создание непрямоугольного массива

```
class ArrayDemo{
      public static void main(String[] args){
            // Индексные переменные и размер массива:
            int i,j,n;
            // Создание массива (второй размер не указан):
            int[][] nums=new int[5][];
            // Определение первого размера массива:
            n=nums.length;
            // Цикл для создания треугольного массива:
            for(i=0;i<n;i++){
                  nums[i]=new int[i+1];}
            // Вложенные циклы для заполнения элементов массива:
            for(i=0;i<n;i++){
                  for(j=0;j<nums[i].length;j++){
                        // Присваивание значения элементу массива:
                        nums[i][j]=10*(i+1)+j+1;
                  // Вывод значения на экран:
                  System.out.print(nums[i][j]+" ");}
            // Переход на новую строку:
      System.out.println();}
      }
}
```

Подобным образом создаются многомерные массивы. В примере 4 приведен пример создания трехмерного массива размером три по каждому из индексов, который определяет тензор Леви–Чевита. Компоненты этого тензора имеют три индекса и отличны от нуля, только если все индексы различны. Элемент с индексами 0, 1 и 2 равен единице. Любой элемент, который получается циклической перестановкой этих индексов, также равен 1. Прочие элементы равны –1. Таким образом, всего три единичных элемента и три элемента со значением –1, остальные равны нулю.

Пример 4. Создание трехмерного массива

```
class MyTArray{
      public static void main(String[] args){
            // Индексные переменные:
            int i,j,k;
            // Объявление трехмерного массива:
            byte[][][] epsilon=new byte[3][3][3];
            // Обнуление элементов массива:
            for(i=0;i<3;i++)
                  for(j=0;j<3;j++)
                        for(k=0;k<3;k++)
                              epsilon[i][j][k]=0;
            // Единичные элементы массива:
            epsilon[0][1][2]=epsilon[1][2][0]=epsilon[2][0][1]=1;
            // Элементы со значением -1:
            epsilon[1][0][2]=epsilon[0][2][1]=epsilon[2][1][0]=-1;
      }
}
```

Кроме числовых массивов в Java широко используются символьные массивы. В примере 5 приведена простая демонстрация использования символьного массива.

Пример 5. Символьный массив

```
class CharArray{
      public static void main(String[] args){
      char[] words=new char[]
{'С','и','м','в','о','л','ь','н','ы','й',' ','м','а','с','с','и','в'};
      System.out.println(words);
      }
}
```

Символьный массив создается стандартным способом: одновременно с объявлением переменной массива `words` списком символов инициализируются элементы массива. В результате выполнения команды `System.out.println(words)` на экран выводится сообщение *Символьный массив* . Для вывода значений элементов символьного массива аргументом метода `println()` указано имя массива - переменная массива `words`. Причина этого связана со способом автоматического преобразования разных объектов (в том числе символьного массива) в текстовый формат.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

1. Используя массивы напишите программу на Java, в которой реализовано скалярное и векторное произведение двух векторов, представленных в трехмерном Декартовом пространстве.

2. Напишите программу, в которой создается одномерный массив заданной длины и инициализируется числами Фибоначчи.

3. Напишите программу транспонирования квадратной матрицы задаваемого размера.

4. Напишите программу произведения квадратных матриц типа double.

5. Одной из часто используемых подзадач с массивами является сортировка элементов по одному из заданных условий: по-возрастанию, по не убыванию, по не возрастанию или по-убыванию. Разнообразие алгоритмов сортировки требует понимания их свойств при применении на практике. Напишите программу (или программы) на Java, в которой для одних и тех же размеров одномерных массивов выполняется сортировка элементов по 4 различным алгоритмам. Результатом работы программы должно быть время выполнения каждого алгоритма сортировки. Для инициализации элементов массивов используйте генератор случайных чисел `Math.random()`, например

```
a[i]=1000.0*Math.random()/(Math.random()+Math.random());
```

где умножение на 1000.0 применяется для расширения диапазона значений и может быть заменено на любое другое число.

Для определения времени выполнения той части программы, которая реализует алгоритм сортировки, можно использовать следующую конструкцию

```
            long t1, t2;

// определяем системное время в миллисекундах до сортировки
            t1=System.currentTimeMillis();

            {код, реализующий алгоритм сортировки}

// определяем системное время в миллисекундах после сортировки
            t2=System.currentTimeMillis();

// выводим время выполнения сортировки в секундах
            System.out.println("time = "+(double)(t2-t1)/1000+" s");
```