ПЕРЕМЕННЫЕ И МАССИВЫ

Для хранения данных в программе предназначены **переменные**. Переменная представляет именованную область памяти, которая хранит значение определенного типа. Каждая переменная имеет тип, имя и значение. Тип определяет, какую информацию может хранить переменная или диапазон допустимых значений.

Переменные объявляются следующим образом:

```
1    тип_данных имя_переменной;
```

Например, определим переменную, которая будет называться x и будет иметь тип int:

```
int x;
```

В этом выражении мы объявляем переменную x типа int. То есть x будет хранить некоторое число не больше 4 байт.

В качестве имени переменной может выступать любое произвольное название, которое удовлетворяет следующим требованиям:

- имя может содержать любые алфавитно-цифровые символы, а также знак подчеркивания, при этом первый символ в имени не должен быть цифрой
- в имени не должно быть знаков пунктуации и пробелов
- имя не может быть ключевым словом языка Java

Кроме того, при объявлении и последующем использовании надо учитывать, что Java - регистрозависимый язык, поэтому следующие объявления int num; и int NUM; будут представлять две разных переменных.

Объявив переменную, мы можем присвоить ей значение:

```
int x;      // объявление переменной
x = 10;     // присвоение значения
System.out.println(x);  // 10
```

Также можно присвоить значение переменной при ее объявлении. Этот процесс называется **инициализацией**:

```
int x = 10;     // объявление и инициализация переменной
System.out.println(x);  // 10
```

Если мы не присвоим переменной значение до ее использования, то мы можем получить ошибку, например, в следующем случае:

```
int x;
System.out.println(x);
```

Через запятую можно объявить сразу несколько переменных одного типа:

```
int x, y;
x = 10;
y = 25;
System.out.println(x);  // 10
System.out.println(y);  // 25
```

Также можно их сразу инициализировать:

```
int x = 8, y = 15;
System.out.println(x);  // 8
System.out.println(y);  // 15
```

Отличительной особенностью переменных является то, что мы можем в процессе работы программы изменять их значение:

```
int x = 10;
System.out.println(x);  // 10
x = 25;
System.out.println(x);  // 25
```

Ключевое слово var

Начиная с Java 10 в язык было добавлено ключевое слово **var**, которое также позволяет определять переменную:

```
var x = 10;
System.out.println(x);  // 10
```

Слово var ставится вместо типа данных, а сам тип переменной выводится из того значения, которое ей присваивается. Например, переменной x присваивается число 10, значит, переменная будет представлять тип int.

Но если переменная объявляется с помощью var, то мы обязательно должны инициализировать ее, то есть предоставить ей начальное значение, иначе мы получим ошибку, как, например, в следующем случае:

```
var x;      // ! Ошибка, переменная не инициализирована
x = 10;
```

Константы

Кроме переменных, в Java для хранения данных можно использовать **константы**. В отличие от переменных константам можно присвоить значение только один раз. Константа объявляется также, как и переменная, только вначале идет ключевое слово **final**:

```
final int LIMIT = 5;
System.out.println(LIMIT);  // 5
// LIMIT=57; // так мы уже не можем написать, так как LIMIT - константа
```

Как правило, константы имеют имена в верхнем регистре.

Константы позволяют задать такие переменные, которые не должны больше изменяться. Например, если у нас есть переменная для хранения числа pi, то мы можем объявить ее константой, так как ее значение постоянно.

Массив представляет набор однотипных значений. Объявление массива похоже на объявление обычной переменной, которая хранит одиночное значение, причем есть два способа объявления массива:

```
тип_данных название_массива[];
// либо
тип_данных[] название_массива;
```

Например, определим массив чисел:

```
int nums[];
int[] nums2;
```

После объявления массива мы можем инициализовать его:

```
int nums[];
nums = new int[4];  // массив из 4 чисел
```

Создание массива производится с помощью следующей конструкции: new тип_данных[количество_элементов], где new - ключевое слово, выделяющее память для указанного в скобках количества элементов. Например, nums = new int[4]; - в этом выражении создается массив из четырех элементов int, и каждый элемент будет иметь значение по умолчанию - число 0.

Также можно сразу при объявлении массива инициализировать его:

```
int nums[] = new int[4];    // массив из 4 чисел
int[] nums2 = new int[5];   // массив из 5 чисел
```

При подобной инициализации все элементы массива имеют значение по умолчанию. Для числовых типов (в том числе для типа char) это число 0, для типа boolean это значение false, а для остальных объектов это значение **null**. Например, для типа int значением по умолчанию является число 0, поэтому выше определенный массив nums будет состоять из четырех нулей.

Однако также можно задать конкретные значения для элементов массива при его создании:

```
// эти два способа равноценны
int[] nums = new int[] { 1, 2, 3, 5 };

int[] nums2 = { 1, 2, 3, 5 };
```

Стоит отметить, что в этом случае в квадратных скобках не указывается размер массива, так как он вычисляется по количеству элементов в фигурных скобках.

После создания массива мы можем обратиться к любому его элементу по индексу, который передается в квадратных скобках после названия переменной массива:

```
int[] nums = new int[4];
// устанавливаем значения элементов массива
nums[0] = 1;
nums[1] = 2;
nums[2] = 4;
nums[3] = 100;

// получаем значение третьего элемента массива
System.out.println(nums[2]);    // 4
```

Индексация элементов массива начинается с 0, поэтому в данном случае, чтобы обратиться к четвертому элементу в массиве, нам надо использовать выражение nums[3].

И так как у нас массив определен только для 4 элементов, то мы не можем обратиться, например, к шестому элементу: nums[5] = 5;. Если мы так попытаемся сделать, то мы получим ошибку.

Длина массива

Важнейшее свойство, которым обладают массивы, является свойство **length**, возвращающее длину массива, то есть количество его элементов:

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

Нередко бывает неизвестным последний индекс, и чтобы получить последний элемент массива, мы можем использовать это свойство:

```
1    int last = nums[nums.length-1];
```

Многомерные массивы

Ранее мы рассматривали одномерные массивы, которые можно представить как цепочку или строку однотипных значений. Но кроме одномерных массивов также бывают и многомерными. Наиболее известный многомерный массив - таблица, представляющая двухмерный массив:

```
1    int[] nums1 = new int[] { 0, 1, 2, 3, 4, 5 };
2
3    int[][] nums2 = { { 0, 1, 2 }, { 3, 4, 5 } };
```

Визуально оба массива можно представить следующим образом:

Одномерный массив nums1

0	1	2	3	4	5

Двухмерный массив nums2

0	1	2
3	4	5

Поскольку массив nums2 двухмерный, он представляет собой простую таблицу. Его также можно было создать следующим образом: int[][] nums2 = new int[2][3];. Количество квадратных скобок указывает на размерность массива. А числа в скобках - на количество строк и столбцов. И также, используя индексы, мы можем использовать элементы массива в программе:

```
1    // установим элемент первого столбца второй строки
2    nums2[1][0]=44;
3    System.out.println(nums2[1][0]);
```

Объявление трехмерного массива могло бы выглядеть так:

```
1    int[][][] nums3 = new int[2][3][4];
```

Зубчатый массив

Многомерные массивы могут быть также представлены как "зубчатые массивы". В вышеприведенном примере двухмерный массив имел 2 строчки и три столбца, поэтому у нас получалась ровная таблица. Но мы можем каждому элементу в двухмерном массиве присвоить отдельный массив с различным количеством элементов:

```
1    int[][] nums = new int[3][];
2    nums[0] = new int[2];
3    nums[1] = new int[3];
4    nums[2] = new int[5];
```

foreach

Специальная версия цикла for предназначена для перебора элементов в наборах элементов, например, в массивах и коллекциях. Она аналогична действию цикла **foreach** , который имеется в других языках программирования. Формальное ее объявление:

```
1    for (тип_данных название_переменной : контейнер){
2        // действия
3    }
```

Например:

```
1    int[] array = new int[] { 1, 2, 3, 4, 5 };
2    for (int i : array){
3
4        System.out.println(i);
```

```
5    }
```

В качестве контейнера в данном случае выступает массив данных типа int. Затем объявляется переменная с типом int

То же самое можно было бы сделать и с помощью обычной версии for:

```
1    int[] array = new int[] { 1, 2, 3, 4, 5 };
2    for (int i = 0; i < array.length; i++){
3        System.out.println(array[i]);
4    }
```

В то же время эта версия цикла for более гибкая по сравнению for (int i : array). В частности, в этой версии мы можем изменять элементы:

```
1    int[] array = new int[] { 1, 2, 3, 4, 5 };
2    for (int i=0; i<array.length;i++){
3        array[i] = array[i] * 2;
4        System.out.println(array[i]);
5    }
```

Перебор многомерных массивов в цикле

```
1     int[][] nums = new int[][]
2     {
3         {1, 2, 3},
4         {4, 5, 6},
5         {7, 8, 9}
6     };
7     for (int i = 0; i < nums.length; i++){
8         for(int j=0; j < nums[i].length; j++){
9
10            System.out.printf("%d ", nums[i][j]);
11        }
12        System.out.println();
13    }
```

Сначала создается цикл для перебора по строкам, а затем внутри первого цикла создается внутренний цикл для перебора по столбцам конкретной строки. Подобным образом можно перебрать и трехмерные массивы и наборы с большим количеством размерностей.