

# JAVA 2

# Switch

Оператор `switch` позволяет делать выбор между несколькими вариантами дальнейшего выполнения программы. Выражение последовательно сравнивается с списка значений оператора `switch` и при совпадении, выполняется набор операторов, связанных с этим условием. Если совпадений не было, выполняется блок `default` (блок `default` является необязательной частью оператора `switch`).

```
switch (выражение) {  
  case значение 1:  
    набор _ операторов 1;  
    break;  
  case значение 2:  
    набор _ операторов 2;  
    break;  
  ...  
  default:  
    набор _ операторов;  
}
```

## Циклы for

```
for (инициализация; условие;  
итерация) {  
набор _ операторов;  
}
```

# Цикл `foreach`

Ещё одной разновидностью цикла `for` является цикл `foreach`, который в большинстве случаев используется для прохождения по всем элементам массива или коллекции без необходимости знать индекс проверяемого элемента. В приведённом ниже примере мы проходим по элементам массива `sm` типа `String`, и каждому такому элементу присваиваем временное имя `o`, то есть «в единицу времени» `o` указывает на один элемент массива.

# Вложенные циклы

```
for (инициализация; условие; итерация){  
  for (инициализация; условие; итерация){  
    ...  
  }  
}
```

# Циклы while

Цикл `while` работает до тех пор, пока указанное условие истинно. Как только условие становится ложным, управление программой передается строке кода, следующей непосредственно после цикла. Если заранее указано условие, которое не выполняется, программа в тело цикла даже не попадет.

```
while (условие) {  
набор _ операторов;  
}
```

# do-while

Цикл do-while очень похож на ранее рассмотренные циклы, только в отличие от for и while, в которых условие проверялось в самом начале (предусловие), в цикле do-while условие выполнения проверяется в самом конце (постусловие), это означает, что цикл do-while всегда выполняется хотя бы один раз.

```
do {  
набор _ операторов;  
while (условие);
```



# Кодовые блоки

```
public static void main ( String args []) {    // <- начало кодового блока main  
int w = 1 , h = 2 , v = 0 ;  
  
if ( w < h ) {                                // <- Начало кодового блока if  
v = w * h ;  
w = 0 ;  
}  
}
```

# Массивы

Массив представляет собой набор однотипных переменных с общим для обращения к ним именем.

```
тип _ данных[] имя _ массива = new тип _ данных[размер _ массива];
```

При создании массива сначала объявляется переменная, ссылающаяся на массив, затем выделяется память для массива, а ссылка на неё присваивается переменной массива. Следовательно, память для массивов в Java динамически распределяется с помощью оператора `new`. В следующей строке кода создается массив типа `int`, состоящий из 5 элементов, а ссылка на него присваивается переменной `arr`.

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

В переменной `arr` сохраняется ссылка на область памяти, выделяемую для массива оператором `new`. Этой памяти должно быть достаточно для размещения в ней 5 элементов типа `int`. Доступ к отдельным элементам массива осуществляется с помощью индексов. Индекс обозначает положение элемента в массиве, индекс первого элемента равен нулю. Так, если массив `arr` содержит 5 элементов, их индексы находятся в пределах от 0 до 4. Индексирование массива осуществляется по номерам его элементов, заключенным в квадратные скобки. Например, для доступа к первому элементу массива `arr` следует указать `arr[0]`, а для доступа к последнему элементу этого массива — `arr[4]`. В приведенном ниже примере программы в массиве `arr` сохраняются числа от 0 до 4.

## Другой вид инициализации массива

тип \_ данных[] имя \_ массива = { v1 , v2 , v3 , ..., vN } ;

# Homework

1. Задать целочисленный массив, состоящий из элементов 0 и 1. Например: [ 1, 1, 0, 0, 1, 0, 1, 1, 0, 0 ]. С помощью цикла и условия заменить 0 на 1, 1 на 0;
2. Задать пустой целочисленный массив размером 8. С помощью цикла заполнить его значениями 0 3 6 9 12 15 18 21;