**Операторы перехода break, continue и return**

Java поддерживает три оператора перехода — break, continue и return. Они передают управление в другую часть вашей программы. Рассмотрим каждый из операторов подробно.

**Использование оператора break**

Оператор break в Java используется в трех случаях. Во-первых, как вы уже видели, он заканчивает последовательность операторов в ветвях оператора switch. Во-вторых, его можно использовать для выхода из цикла. В-третьих, он может применяться как "цивилизованная" форма оператора безусловного перехода goto. В этом разделе объясняются два последних случая. Использование **break** для выхода из цикла Используя break, вы можете форсировать немедленное завершение цикла, обходя условное выражение и любой остающийся код в теле цикла. Когда оператор break встречается внутри цикла, второй заканчивается и программное управление передается оператору, следующему за ним. Простой пример:

// Использование break для выхода из цикла.

**public** **class** BreakLoop {

**public** **static** **void** main(String[] args) {

**for** (**int** i = 0; i < 100; i++) {

**if** (i == 4) {

**break**; // завершить цикл, если i = 4

}

System.out.println("i: " + i);

}

System.out.println("Цикл завершен.");

}

}

**Эта программа генерирует следующий вывод:**

i: 0

i: 1

i: 2

i: 3

Цикл завершен.

Хотя цикл for разработан здесь для выполнения своих операторов от 0 до 99 раз, оператор break заставляет его завершиться раньше, когда i равно 4. Оператор break может использоваться с любым из циклов Java, включая преднамеренно бесконечные циклы. Например, ниже показана предыдущая программа, закодированная при помощи цикла while. Вывод этой программы такой же, как у ее предшественницы.

// Использование break для выхода из while-цикла.

**public** **class** BreakLoop2 {

**public** **static** **void** main(String[] args) {

**int** i = 0;

**while** (i < 100) {

**if** (i == 4) {

**break**; // закончить цикл, если i = 4

}

System.out.println("i: " + i);

i++;

}

System.out.println("Цикл завершен.");

}

}

При использовании внутри набора вложенных циклов оператор break будет выходить только из самого внутреннего цикла. Например:

// Использование break во вложенных циклах.

**public** **class** BreakLoop3 {

**public** **static** **void** main(String[] args) {

**for** (**int** i = 0; i < 3; i++) {

System.out.print("Итерация " + i + ": ");

**for** (**int** j = 0; j < 10; j++) {

**if** (j == 4) {

**break**; // закончить цикл, если j = 4

}

System.out.print(j + " ");

}

System.out.println();

}

System.out.println("Цикл завершен.");

}

}

**Эта программа генерирует следующий вывод:**

Итерация 0: 0 1 2 3

Итерация 1: 0 1 2 3

Итерация 2: 0 1 2 3

Цикл завершен.

Как вы видите, оператор break во внутреннем цикле вызывает завершение только этого цикла. Внешний цикл не затрагивается. Сделаем еще два замечания относительно break. Во-первых, в цикле может появиться несколько операторов break. Однако будьте осторожны. Слишком большое их количество создает тенденцию к деструктуризации вашего кода. Во-вторых, break, который завершает оператор switch, воздействует только на данный switch-оператор (но не на включающие его циклы). Замечание: Break не проектировался в качестве нормального средства завершения цикла. Эту цель обслуживает условное выражение заголовка цикла. Оператор break следует использовать для прерывания цикла только тогда, когда возникают некоторые специальные ситуации.

**Использование break как формы goto**

В дополнение к применению в операторах [switch](https://javarush.ru/groups/posts/1832-operator-switch-v-java) и циклах break можно также использовать отдельно, в качестве "цивилизованной" формы оператора goto. Java не содержит оператора goto, потому что он выполняет переход произвольным и неструктурированным способом. Код, интенсивно использующий goto, обычно трудно понять и поддерживать. Он также отменяет некоторые оптимизации компилятора. Существует, однако, несколько мест в программе, где goto — ценная и законная конструкция управления потоком выполнения. Например, goto может быть полезен, когда вы выходите из глубоко вложенного набора циклов. Чтобы обрабатывать такие ситуации, Java определяет расширенную форму оператора break. Используя ее, вы можете выйти из одного или большего количества блоков кода. Этим блокам не нужно быть частью цикла или оператора switch. Это может быть любой блок. Далее, вы можете определить точно, где выполнение будет продолжено, потому что данная форма break работает с меткой и обеспечивает преимущества goto, минуя его проблемы. Оператор break с меткой имеет следующую общую форму: break label; Здесь label — имя метки, которая идентифицирует некоторый блок кода. Когда эта форма break выполняется, управление передается из именованного блока кода (чья метка указана в операторе break) на следующий за этим блоком оператор. Помеченный блок кода обязательно должен включать данный оператор break, но не требуется, чтобы это включение было непосредственным (т. е. break может включаться не прямо в блок со своей меткой, а во вложенный в него блок, возможно, тоже помеченный). Это означает, что вы можете использовать помеченный оператор break, чтобы выйти из набора вложенных блоков. Но вы не имеете возможности использовать break для передачи управления блоку кода, который не включает оператор break. Для именования блока поместите метку в его начале (перед открывающей блок фигурной скобкой). Метка — это любой допустимый идентификатор Java, за которым следует двоеточие. После маркировки блока, его метку можно использовать как аргумент оператора break. Это приведет к тому, что выполнение будет продолжено с конца помеченного блока. Например, следующая программа содержит три вложенных блока, причем каждый имеет свою собственную метку. Оператор break осуществляет переход вперед, за конец блока, маркированного меткой second, пропуская два оператора println().

// Использование break как цивилизованной формы goto.

**public** **class** Break {

**public** **static** **void** main(String[] args) {

**boolean** t = true;

first:

{

second:

{

third:

{

System.out.println("Перед оператором break.");

**if** (t) {

**break** second; // выход из блока second

}

System.out.println("Данный оператор никогда не выполнится");

}

System.out.println("Данный оператор никогда не выполнится ");

}

System.out.println("Данный оператор размещен после блока second.");

}

}

}

**Выполнение этой программы генерирует следующий вывод:**

Перед оператором **break**.

Данный оператор размещен после блока second.

Одним из наиболее частых применений помеченного оператора break является выход из вложенных циклов. Например, в следующей программе внешний цикл выполняется только один раз:

// Использование break для выхода из вложенных циклов.

**public** **class** BreakLoop4 {

**public** **static** **void** main(String[] args) {

outer:

**for** (**int** i = 0; i < 3; i++) {

System.out.print("Итерация " + i + ": ");

**for** (**int** j = 0; j < 100; j++) {

**if** (j == 10) {

**break** outer; // выйти из обоих циклов

}

System.out.print(j + " ");

}

System.out.println("Эта строка никогда не будет выведена");

}

System.out.println("Цикл завершен.");

}

}

**Программа генерирует следующий вывод:**

Итерация 0: 0 1 2 3 4 5 6 7 8 9 Цикл завершен.

Нетрудно заметить, что при прерывании внутреннего цикла до конца внешнего заканчиваются оба цикла. Имейте в виду, что вы не можете сделать break-переход к любой метке, которая не определена для включающего блока. Например, следующая программа недопустима и не будет компилироваться:

// Эта программа содержит ошибку.

**public** **class** BreakErr {

**public** **static** **void** main(String[] args) {

one:

**for** (**int** i = 0; i < 3; i++) {

System.out.print("Итерация " + i + ": ");

}

**for** (**int** j = 0; j < 100; j++) {

**if** (j == 10) {

**break** one; //He верно!

}

System.out.print(j + " ");

}

}

}

Так как цикл, помеченный как one, не включает оператор break, передать управление этому блоку невозможно.

**Использование оператора continue**

Иногда полезно начать очередную итерацию цикла пораньше. То есть нужно продолжить выполнение цикла, но прекратить обработку остатка кода в его теле для данной частной итерации. На самом деле это — goto-переход мимо следующих операций тела в конец блока цикла. Такое действие выполняет оператор continue. В циклах while и do while оператор continue вызывает передачу управления непосредственно условному выражению, которое управляет циклом. В цикле for управление переходит сначала к итерационной части оператора for и затем к условному выражению. Для всех трех циклов любой промежуточный код обходится. Пример программы, которая использует **continue** для печати двух чисел на каждой строке, приведен ниже:

// Демонстрирует continue.

**public** **class** Continue {

**public** **static** **void** main(String[] args) {

**for** (**int** i = 0; i < 10; i++) {

System.out.print(i + " ");

**if** (i % 2 == 0) {

**continue**;

}

System.out.println("");

}

}

}

Этот код использует операцию % (остаток по модулю), чтобы проверять, является ли i четным. Если это так, цикл продолжается без печати символа новой строки (newline). **Вывод программы:**

0 1

2 3

4 5

6 7

8 9

Как и в операторе break, в continue можно определить метку, указывающую, какой включающий цикл следует продолжить. Пример программы, которая использует continue для печати треугольной таблицы умножения от 0 до 9.

// Использование continue с меткой.

**public** **class** ContinueLabel {

**public** **static** **void** main(String[] args) {

outer:

**for** (**int** i = 0; i < 10; i++) {

**for** (**int** j = 0; j < 10; j++) {

**if** (j > i) {

System.out.println();

**continue** outer;

}

System.out.print(" " + (i \* j));

}

}

System.out.println();

}

}

Оператор continue в этом примере завершает цикл, вычисляющий j, и продолжает со следующей итерации цикла, управляемого i. Вывод этой программы:

0

0 1

0 2 4

0 3 6 9

0 4 8 12 16

0 5 10 15 20 25

0 6 12 18 24 30 36

0 7 14 21 28 35 42 49

0 8 16 24 32 40 48 56 64

0 9 18 27 36 45 54 63 72 81

В использовании continue полезен крайне редко. Одна из причин этого заключается в том, что Java обеспечивает богатый набор операторов цикла,которые устраивают большинство приложений. Однако, для тех специальных ситуаций, в которых необходимо досрочное прекращение итерации, оператор continue обеспечивает структурный способ выполнения этой задачи.

**Оператор return**

Последний оператор управления — return. Он используется для явного возврата из метода, т. е. передает программное управление обратно в вызывающую программу. Оператор return относят к категории операторов перехода. Хотя его полное обсуждение должно подождать до обсуждения методов, здесь представлен краткий обзор return. Оператор return можно использовать в любом месте метода для выполнения перехода обратно в вызывающую этот метод программу. Оператор return немедленно заканчивает выполнение метода, в котором он находится. Это иллюстрирует следующий пример:

// Демонстрирует return.

**public** **class** Return {

**public** **static** **void** main(String[] args) {

**boolean** t = true;

System.out.println("Перед оператором return.");

**if** (t) {

**return**; // возврат в вызывающую программу

}

System.out.println("Этот оператор никогда не выполнится.");

}

}

Здесь return выполняет возврат к Java-системе во время выполнения, т. к. именно эта система вызывает метод main(). **Вывод этой программы:**

Перед оператором **return**.

Можно заметить, что заключительный оператор println() не выполняется. В момент выполнения return управление передается обратно в вызывающую программу. И последнее замечание. В предыдущей программе оператор if (t) необходим. Без него Java-компилятор выдал бы ошибку "недостижимый код", потому что он знал бы, что последний оператор println() никогда не будет выполняться. Чтобы предотвратить указанную ошибку, как раз и используется оператор if (t)он обманывает компилятор ради данной демонстрации.