Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебный предмет
«Разработка кроссплатформенных приложений»

Инструкция

по выполнению лабораторной работы
«Разработка, отладка и испытание циклических алгоритмов и программ»

Минск 2025 г.

Лабораторная работа № 3

Тема работы: «Разработка, отладка и испытание циклических алгоритмов и программ»

1 Цель работы

Сформировать умения разрабатывать программы с использованием операторов цикла.

2 Задание

Номер варианта соответствует вашему номеру по списку, если номер по списку больше последнего варианта, то необходимо начать подсчет варианта с начала (например, учащийся с номером по списку 22 должен выполнить 2 вариант в первом задании и 3 вариант во втором задании).

Два задания обязательны для выполнения!

Задание 1

Вычислить сумму по варианту

Вариант 1

$$s = 4 * \sum_{i=1}^{40} 1 + 7i$$

Вариант 2

$$Y = 2 \sum_{i=1}^{20} (2i^2 - 5/i)$$

Вариант 3

$$s = 4 + \sum_{i=1}^{50} i^2$$

Вариант 4

$$Y = 3 \sum_{i=1}^{20} e^{-0,25i}$$

Вариант 5

$$s = 8 \sum_{i=1}^{20} (i + 1)^2$$

Вариант 6

$$s = 61 / \sum_{i=1}^{6} i + 5$$

Вариант 7

$$s = 5{,}3\sum_{i=1}^{20} i^3$$

Вариант 8

$$s = 5i + \sum_{i=1}^{10} i$$

Вариант 9

$$s = \sum_{i=1}^{20} (2i + i^2)$$

Вариант 10

$$v = 20 + 12\sum_{i=1}^{50} i$$

Вариант 11

$$s = \sum_{i=1}^{20} (i^2 - 3i + 1)$$

Вариант 12

$$n = \sum_{i=1}^{11} i^2 - 11i$$

Вариант 13

$$s = 9 - \sum_{i=1}^{21} \frac{i}{i+1}$$

Вариант 14

$$n = 7\sum_{i=5}^{12} 2i + 9i$$

Вариант 15

$$s = 4\sum_{i=1}^{8} (i^2 + 5)^{-1}$$

Вариант 16

$$n = \sum_{i=1}^{10} i^2 + 8{,}4$$

Вариант 17

$$Y = 7\sum_{i=1}^{20} \frac{\sin(a_i)}{\cos(a_i) + 2}$$

Вариант 18

$$s = \sum_{k=1}^{10} \frac{2^k}{k} - 14k$$

Вариант 19

$$s = \sum_{n=1}^{50} \frac{\cos(nx)}{4n^2 - 1}$$

Вариант 20

$$s = 7i + \sum_{i=1}^{5} \frac{1}{2}$$

Задание 2

Использовать цикл while и do while. Задачу решить двумя способами, с помощью двух циклов.

1. Составьте программу табулирования функции $Y = \sqrt{1 - 0{,}4x^2} - \cos x$, изменяющейся на интервале $[-1; 1{,}5]$ с шагом 0,25.
2. Найти сумму всех четных чисел, то тех пор, пока четное число не будет меньше 200.
3. Составить таблицу значений функции для переменной $y = e^x + 7$ для переменной *x*, изменяющейся на отрезке [0; 10] с шагом 0,5.
4. Составить программу, через сколько лет г. Махачкала станет городом с миллионным населением, если по статистике средний прирост населения составляет 0,8%. Перепись населения показала, что на 2004 год численность населения составила 600000.
5. Найти сумму сходящегося ряда $S = \sum_{i=1}^{\infty} \frac{i}{i^3+2}$, с заданной точностью ε.
6. Составить программу накопления капитала в банке за 2 года на простых процентах, если стартовый капитал равен 100$.
7. Написать программу подсчета количества воскресений в 2006 году, если первый воскресный день пришелся на первое января.
8. Найти сумму сходящегося ряда $S = \sum_{x=1}^{\infty} \frac{\sin x}{x^2+5x}$, с заданной точностью ε.
9. Составьте программу табулирования функции $y = \ln x + e^x$, изменяющейся на интервале [a;b] с шагом h.
10. Вычислите сумму всех чисел Фибоначчи, которые меньше 1000. {Числа Фибоначчи определяются следующим образом: $a_1 = 1$, $a_2 = 1, a_i = a_{i-1} + a_{i-2}$ для всех $i > 2$}
11. Найдите первое число Фибоначчи, больше заданного n.

12. Проверить численно первый замечательный предел $\lim_{x\to 0}\frac{\sin x}{x}=1$, задавая для *x* значения $1,\frac{1}{2},\frac{1}{4},\frac{1}{8},\ldots$ до тех пор, пока левая часть равенства не будет отличаться от правой менее чем на заданную погрешность ε.
13. Проверить численно второй замечательный предел $\lim_{n\to\infty}(1+\frac{1}{n})^n=e$, задавая для *x* значения 1, 2, 4, 8, … до тех пор, пока левая часть равенства не будет отличаться от правой менее чем на заданную погрешность ε.
14. Найти число слагаемых, необходимых для достижения точности 0,00001, в следующем разложении: $\pi=4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\ldots\right)$.
15. Найти число слагаемых, необходимых для достижения точности 0,00001, в следующем разложении: $\pi=3+4\left(\frac{1}{2\cdot3\cdot4}-\frac{1}{4\cdot5\cdot6}+\frac{1}{6\cdot7\cdot8}+\ldots\right)$.
16. Сколько множителей надо взять в произведении $\prod_{k=1}^{\infty}\left(1+\frac{(-1)^k}{2k+1}\right)=\frac{\sqrt{2}}{2}$, чтобы равенство выполнялось до шестой значащей цифры, то есть с погрешностью не более 0,000001?
17. Для заданных m и n вычислить число сочетаний C_m^n по рекуррентной формуле: $C_m^n=\frac{m-n+1}{n}C_m^{n-1}, C_m^1=m.$
18. Леспромхоз ведет заготовку деловой древесины. Первоначальный объем её на территории леспромхоза составлял *p* кубометров. Ежегодный прирост составляет *k%*. Годовой план заготовки – *t* кубометров. Будет ли лес вырублен полностью? Если да, то через сколько лет? (*p, k, t* вводятся с клавиатуры)
19. У гусей и кроликов вместе 2n лап. Сколько может быть гусей и кроликов? Вывести все возможные сочетания.

3 Оснащение работы

Задание по варианту, ЭВМ, среда разработки IntelliJ IDEA.

4 Основные теоретические сведения

Операторы цикла

Операторами цикла Java являются for, while и do-while. Эти операторы образуют конструкции, обычно называемые циклами. Как наверняка известно читателям, циклы многократно выполняют один и тот же набор инструкций до тех пор, пока не будет удовлетворено условие завершения цикла. Как вы вскоре убедитесь, Java предлагает средства создания циклов, способные удовлетворить любые потребности программирования.

Цикл for

Начиная с версии JDK 5, в Java существуют две формы цикла for. Первая — традиционная форма, используемая начиная с исходной версии Java. Вторая — новая форма "for-each". Мы рассмотрим оба типа цикла for, начиная с традиционной формы.

Общая форма традиционного оператора for выглядит следующим образом:

```
for(инициализация; условие; повторение)
{ // тело
}
```

Если в цикле будет повторяться только один оператор, фигурные скобки можно опустить.

Цикл for действует следующим образом. При первом запуске цикла программа выполняет инициализационную часть цикла. В общем случае это выражение, устанавливающее значение управляющей переменной цикла, которая действует в качестве счетчика, управляющего циклом. Важно понимать, что выражение инициализации выполняется только один раз. Затем программа вычисляет условие, которое должно быть булевским выражением. Как правило, выражение сравнивает значение управляющей переменной с целевым значением. Если это значение истинно, программа выполняет тело цикла. Если оно ложно, выполнение цикла прерывается. Затем программа выполняет часть повторение цикла. Обычно это выражение, которое увеличивает или уменьшает значение управляющей переменной. Затем программа повторяет цикл, при каждом прохождении вначале вычисляя условное выражение, затем выполняя тело цикла и выполняя выражение повторения. Процесс повторяется до тех пор, пока значение выражения повторения не станет ложным.

Ниже приведена версия программы подсчета "тактов", в которой использован цикл for.

```
// Демонстрация использования цикла for.
class ForTick {
public static void main(String args[]) {
int n;
for(n=10; n>0; n—)
System.out.println("такт " + n) ;
}
}
```

Объявление управляющих переменных цикла внутри цикла for

Часто переменная, которая управляет циклом for, требуется только для него и не используется нигде больше. В этом случае переменную можно объявить внутри инициали-зационной части оператора for. Например, предыдущую программу можно переписать, объявляя управляющую переменную л типа int внутри цикла for:

// Объявление управляющей переменной цикла внутри цикла for.

```
class ForTick (public static void main(String args[]) {
//в данном случае переменная n объявляется внутри цикла for
for(int n=10; n>0; n—)
System.out.println("такт " + n) ;
}
}
```

При объявлении переменной внутри цикла for необходимо помнить о следующем важном обстоятельстве: область и время существования этой переменной полностью совпадают с областью и временем существования оператора for. (То есть область существования переменной ограничена циклом for.) Вне цикла for переменная прекратит свое существование. Если управляющую переменную цикла нужно использовать в других частях программы, ее нельзя объявлять внутри цикла for.

В тех случаях, когда управляющая переменная цикла не требуется нигде больше, большинство программистов Java предпочитают объявлять ее внутри оператора for. В качестве примера приведем простую программу, которая проверяет, является ли введенное число простым. Обратите внимание, что управляющая переменная цикла i объявлена внутри цикла for, поскольку она нигде больше не требуется.

Разновидности цикла for

Цикл for поддерживает несколько разновидностей, которые увеличивают его возможности и повышают применимость. Гибкость этого цикла обусловлена тем, что его три части: инициализацию, проверку условий и итерационную не обязательно использовать только по прямому назначению. Фактически каждый из разделов оператора for можно применять в любых целях. Рассмотрим несколько примеров.

Одна из наиболее часто встречающихся вариаций предполагает использование условного выражения. В частности, это выражение не обязательно должно выполнять сравнение управляющей переменной цикла с каким-либо целевым значением. Фактически условием, управляющим циклом for, может быть любое булевское выражение. Например, рассмотрим следующий фрагмент:

```
boolean done = false;
for(int i=1; !done; i++) {
// ...
if(interrupted()) done = true;
}
```

В этом примере выполнение цикла for продолжается до тех пор, пока значение переменной done не будет установлено равным true. В этом цикле проверка значения управляющей переменной цикла i не выполняется.

Приведем еще одну разновидность цикла for. Оставляя все три части оператора пустыми, можно умышленно создать бесконечный цикл (цикл, который никогда не завершается). Например:

```
for( ; ; ) {
```

```
// ...
}
```

Этот цикл может выполняться бесконечно, поскольку условие, по которому он был бы прерван, отсутствует. Хотя некоторые программы, такие как командные процессоры операционной системы, требуют наличия бесконечного цикла, большинство "бесконечных циклов" в действительности представляют собой всего лишь циклы с особыми условиями прерывания. Как вы вскоре убедитесь, существует способ прерывания цикла — даже бесконечного, подобного приведенному примеру — который не требует использования обычного условного выражения цикла.

Версия «for-each» цикла for

Начиная с версии JDK 5 в Java можно использовать вторую форму цикла for, реализующую цикл в стиле "for-each" ("для каждого"). Как вам, возможно, известно, в современной теории языков программирования все большее применение находит концепция циклов "for-each", которые быстро становится стандартными функциональными возможностями во многих языках. Цикл в стиле "for-each" предназначен для строго последовательного выполнения повторяющихся действий по отношению к коллекции объектов, такой как массив. В отличие некоторых языков, подобных C#, в котором для реализации циклов "for-each" используют ключевое слово f oreach, в Java возможность применения цикла "for-each" реализована за счет усовершенствования цикла for. Преимущество этого подхода состоит в том, что для его реализации не требуется дополнительное ключевое слово, и никакой ранее существовавший код не разрушается. Цикл for в стиле "for-each" называют также усовершенствованным циклом for. Общая форма версии "for-each" цикла for имеет следующий вид:

```
for (тип итер-пер : коллекция)
блок-операторов
```

Здесь тип указывает тип, а итер-пер — имя итерационной переменной, которая последовательно будет принимать значения из коллекции, от первого до последнего. Элемент коллекция указывает коллекцию, по которой должен выполняться цикл. С циклом for можно применять различные типы коллекций, но в этой главе мы будем использовать только массивы. (Другие типы коллекций, которые можно применять с циклом for, вроде определенных в каркасе коллекций Collection Framework, рассматриваются в последующих главах книги.) На каждой итерации цикла программа извлекает следующий элемент коллекции и сохраняет его в переменной итер-пер. Цикл выполняется до тех пор, пока не будут получены все элементы коллекции.

Поскольку итерационная переменная получает значения из коллекции, тип должен совпадать (или быть совместимым) с типом элементов, хранящихся в коллекции. Таким образом, при выполнении цикла по массивам тип должен быть совместим с базовым типом массива.

Цикл for в стиле "for-each" позволяет автоматизировать этот процесс. В частности, применение такого цикла позволяет не устанавливать значение

счетчика цикла за счет указания его начального и конечного значений, и исключает необходимость индексации массива вручную. Вместо этого программа автоматически выполняет цикл по всему массиву, последовательно получая значения каждого из его элементов, от первого до последнего. Например, с учетом версии "for-each" цикла for предыдущий фрагмент можно переписать следующим образом:

```
int nums[] = { 1, 2, 3, 4, 5, б, 7, 8, 9, 10 };
int sum = 0;
for(int x: nums) sum += x;
```

При каждом прохождении цикла переменной x автоматически присваивается значение, равное значению следующего элемента массива nums. Таким образом, на первой итерации x содержит 1, на второй — 2 и т.д. При этом не только упрощается синтаксис программы, но и исключается возможность ошибок выхода за пределы массива.

Ниже показан пример полной программы, иллюстрирующей применение описанной версии "for-each" цикла for.

```
// Использование цикла for в стиле for-each.
class ForEach {
public static void main(String args[]) {
int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int sum = 0;
// использование стиля for-each для отображения и суммирования значений
for(int x : nums) {
System.out.println("Значение равно: " + x) ;
sum += x;
}
System.out.println ("Сумма равна: " + sum) ;
}
}
```

Эта программа генерирует следующий вывод:

```
Значение равно: 1
Значение равно: 2
Значение равно: 3
Значение равно: 4
Значение равно: 5
Значение равно: 6
Значение равно: 7
Значение равно: 8
Значение равно: 9
Значение равно: 10
Сумма равна: 55
```

Как видно из этого вывода, оператор for в стиле "for-each" автоматически выполняет цикл по элементам массива, от наименьшего индекса к наибольшему.

Хотя повторение цикла for в стиле "for-each" выполняется до тех пор, пока не будут обработаны все элементы массива, цикл можно прервать и раньше, используя оператор break. Например, следующая программа суммирует значения пяти первых элементов массива nums.

```
// Использование оператора break в цикле for в стиле for-each.
class ForEach2 {
public static void main(String args[]) {
int sum = 0;
int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
// использование цикла for для отображения и суммирования значений
for(int x : nums) {
System.out.println("Значение равно: " + x) ;
sum += x; v if (x == 5) break; // прекращение цикла после получения 5 значений
}
System.out.println("Сумма пяти первых элементов равна: " + sum);
}
}
```

Программа генерирует следующий вывод:

```
Значение равно: 1
Значение равно: 2
Значение равно: 3
Значение равно: 4
Значение равно: 5
Сумма пяти первых элементов равна: 15
```

Как видите, выполнение цикла прекращается после получения значения пятого элемента. Оператор break можно использовать также и с другими циклами Java. Подробнее этот оператор будет рассмотрен в последующих разделах настоящей главы.

При использовании цикла в стиле "for-each" необходимо помнить о следующем важном обстоятельстве. Его итерационная переменная является переменной "только для чтения", поскольку она связана только с исходным массивом. Операция присваивания значения итерационной переменной не оказывает никакого влияния на исходный массив. Иначе говоря, содержимое массива нельзя изменять, присваивая новое значение итерационной переменной. Например, рассмотрим следующую программу:

```
// Переменная цикла for-each доступна только для чтения.
class NoChange {
```

```
public static void main(String args[]) {
int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
for(int x : nums) {
System.out.print (x + " ");
x=x*10; // этот оператор не оказывает никакого влияния на
массив nums
}
System.out.println();
for(int x : nums)
System.out.print (x + " ");
System.out.println ();
}
}
```

Первый цикл for увеличивает значение итерационной переменной на 10. Однако эта операция присваивания не оказывает никакого влияния на исходный массив nums, как видно из результата выполнения второго оператора for. Генерируемый программой вывод подтверждает сказанное:

```
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
```

Цикл while

Оператор while — наиболее часто используемый оператор цикла Java. Он повторяет оператор или блок операторов до тех пор, пока значение его управляющего выражения истинно. Он имеет следующую общую форму:

```
while{условие){ //тело цикла
}
```

Условием может быть любое булевское выражение. Тело цикла будет выполняться до тех пор, пока условное выражение истинно. Когда условие становится ложным, управление передается строке кода, непосредственно следующей за циклом. Фигурные скобки могут быть опущены, только если в цикле повторяется только один оператор.

В качестве примера рассмотрим цикл while, который выполняет обратный отсчет, начиная с 10, вывод ровно 10 строк "тактов":

```
// Демонстрация использования цикла while.
class While {
public static void main(String args[]) {
int n = 10; while(n > 0) {
System.out.println("такт " + n) ;
n—;
}
}
}
```

После запуска эта программа выводит десять "тактов":

```
такт 10
```

```
такт 9
такт 8
такт 7
такт 6
такт 5
такт 4
такт 3
такт 2
такт 1
```

Поскольку цикл while вычисляет свое условное выражение в начале цикла, тело цикла не будет выполнено ни разу, если в самом начале условие оказывается ложным. Например, в следующем фрагменте кода метод println () никогда не будет вызван:

```
int a = 10, b = 20;
while(a > b)
System.out.println("Эта строка отображаться не будет");
```

Тело цикла while (или любого другого цикла Java) может быть пустым. Это обусловлено тем, что синтаксис Java допускает применение нулевого оператора (содержащего только символ точки с запятой). Например, рассмотрим следующую программу:

```
// Целевая часть цикла может быть пустой.
class NoBody {
public static void main(String args[]) {
int i, j;
i = 100;
j = 200;
// вычисление среднего значения i и j
while (++i < --j) ; // в этом цикле тело цикла
отсутствует
System.out.println("Среднее значение равно " + i);
}
}
```

Эта программа вычисляет среднее значение i и j.

Она генерирует следующий вывод:

```
Среднее значение равно 150
```

Этот цикл while работает следующим образом. Значение i увеличивается, а значение j уменьшается на единицу. Затем программа сравнивает эти два значения. Если новое значение i по-прежнему меньше нового значения j, цикл повторяется. Если значение i равно или больше значения j, выполнение цикла прекращается. По выходу из цикла переменная i будет содержать среднее значение исходных значений i и j. (Конечно, эта процедура работает только в том случае, если в самом начале значение i меньше значения j.) Как видите, никакой потребности в наличии тела цикла не существует. Все действия выполняются внутри самого условного выражения. В профессионально

написанном Java-коде короткие циклы часто не содержат тела, если само по себе управляющее выражение может выполнять все необходимые действия.

Цикл do-while

Как вы видели, если в начальный момент условное выражение, управляющее циклом while, ложно, тело цикла вообще не будет выполняться. Однако иногда желательно выполнить тело цикла хотя бы один раз, даже если в начальный момент условное выражение ложно. Иначе говоря, существуют ситуации, когда проверку условия прерывания цикла желательно выполнять в конце цикла, а не в его начале. К счастью, Java поддерживает именно такой цикл: do-while. Этот цикл всегда выполняет тело цикла хотя бы один раз, поскольку его условное выражение проверяется в конце цикла. Общая форма цикла do-while следующая:

```
do { // тело цикла
} while {условие);
```

При каждом повторении цикла do-while программа вначале выполняет тело цикла, а затем вычисляет условное выражение. Если это выражение истинно, цикл повторяется. В противном случае выполнение цикла прерывается. Как и во всех циклах Java, выражение должно быть булевским.

Ниже приведена измененная программа вывода тактов, которая демонстрирует использование цикла do-while. Она генерирует такой же вывод, что и предыдущая версия.

```
// Демонстрация использования цикла do-while.
class DoWhile {
public static void main(String args[]) {
int n = 10;
do {
System.out.println("такт " + n);
n—;
} while(n > 0) ;
}
}
```

Хотя с технической точки зрения в приведенной программе цикл записан правильно, его можно переписать в более эффективном виде:

```
do {
System.out.println("такт " + n) ;
} while(—n > 0);
```

В этом примере операции декремента переменной п и сравнения результирующего значения с нулем объединены в одном выражении (—n>0). Программа работает следующим образом. Вначале она выполняет оператор — n, уменьшая значение n на единицу и возвращая новое значение переменной n. Затем программа^ сравнивает это значение с нулем. Если оно больше нуля, выполнение цикла продолжается. В противном случае цикл прерывается.

Цикл do-while особенно удобен при выборе пункта меню, поскольку обычно в этом случае требуется, чтобы тело цикла меню выполнялось, по меньшей мере, один раз. Рассмотрим следующую программу, которая реализует очень простую систему справки по операторам выбора и цикла Java:

```
// Использование цикла do-while для выбора пункта меню.
class Menu {
public static void main(String args[])
throws java.io.IOException {
char choice;
do {
System.out.println("Справка по:");
System.out.println(" 1. if");
System.out.println (" 2. switch");
System.out.println (" 3. while");
System.out.println(" 4. do-while");
System.out.println (" 5. for\n");
System.out.println("Выберите интересующий пункт:");
choice = (char) System.in.read();
} while ( choice < '1' || choice > '5');
System.out.println("\n");
switch(choice) {
case '1':
System.out.println("if:\n");
System.out.println("if(условие) оператор;");
System.out.println("else оператор;");
break;
case '2':
System.out.println("switch:\n");
System.out.println("switch(выражение) (") ;
System.out.println(" case константа:");
System.out.println(" последовательность операторов");
System.out.println(" break;");
System.out.println(" // ...");
System.out.println("}");
break;
case '3':
System.out.println("while:\n");
System.out.println("while(условие) оператор;");
break;
case '4':
System.out.println("do-while:\n");
System.out.println("do (");
System.out.println(" оператор;");
System.out.println (" } while (условие);
```

```
"); break;
case '5':
System.out.println("for:\n");
System.out.print("for(инициализация;                условие;
повторение)");
System.out.println (" оператор;");
break;
}
}
}
```

Пример вывода выполнения этой программы выглядит следующим образом:

```
Справка по:
1. if
2. switch
3. while
4 . do-while
5. for
Выберите интересующий пункт: 4
do-while: do {
оператор;
} while (условие);
```

В этой программе в цикле do-while осуществляется проверка допустимости введенного пользователем значения. Если это значение недопустимо, программа предлагает пользователю повторить ввод. Поскольку меню должно отобразиться, по меньшей мере, один раз, цикл do-while является прекрасным средством решения этой задачи.

Отметим еще несколько особенностей приведенного примера. Обратите внимание, что считывание символов с клавиатуры выполняется посредством вызова метода System, in. read (). Это — одна из функций консольного ввода Java. Хотя мы и отложим подробное рассмотрение методов консольного ввода-вывода до главы 13, покамест отметим, что в данном случае метод System, in. read () используется для выяснения выбора, осуществленного пользователем. Этот метод считывает символы из стандартного ввода (возвращаемые в виде целочисленных значений — именно потому тип возвращаемого значения был приведен к char). По умолчанию данные из стандартного ввода помещаются в буфер построчно, поэтому, чтобы любые введенные символы были пересланы программе, необходимо нажать клавишу .

Консольный ввод Java может вызывать некоторые затруднения при работе. Более того, большинство реальных Java-программ будут графическими и ориентированным на работу в оконном режиме. Поэтому в данной книге консольному вводу уделяется не очень много внимания. Однако в данном случае он удобен.

Операторы перехода

В Java определены три оператора перехода: break, continue и return. Они передают управление другой части программы. Рассмотрим каждый из них.

Использование оператора break

В Java оператор break находит три применения. Во-первых, как уже было показано, он завершает последовательность операторов в операторе switch. Во-вторых, его можно использовать для выхода из цикла. И, в-третьих, его можно использовать в качестве "цивилизованной" формы оператора безусловного перехода ("goto"). Рассмотрим последние два применения.

Использование оператора break дли выхода из цикла

Используя оператор break, можно вызвать немедленное завершение цикла, пропуская условное выражение и любой остальной код в теле цикла. Когда программа встречает оператор break внутри цикла, она прекращает выполнение цикла, и управление передается оператору, следующему за циклом. Ниже показан простой пример.

```
// Использование оператора break для выхода из цикла.
class BreakLoop {
public static void main(String args[]) {
for(int i=0; i<100; i++) {
if(i == 10) break; // выход из цикла если i равно 10
System.out.println("i : " + i);
}
System.out.println("Цикл завершен.");
}
}
```

Эта программа генерирует следующий вывод:

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

Как видите, хотя цикл for должен был бы выполняться для значений управляющей переменной от 0 до 99, оператор break приводит к более раннему выходу из него, когда значение переменной i становится равным 10.

Оператор break можно использовать в любых циклах Java, в том числе в преднамеренно бесконечных циклах. Например, в предыдущей программе можно было применить цикл while. Эта программа генерирует вывод, совпадающий с предыдущим.

```
// Использование оператора break для выхода из цикла while.
class BreakLoop2 {
public static void main(String args[]) {
int i = 0;
while (i < 100) {
if(i == 10) break; // выход из цикла, если i равно 10
System.out.println ("i: " + i) ;
i++;
}
System.out.println("Цикл завершен.");
}
}
```

В случае его использования внутри набора вложенных циклов оператор break осуществляет выход только из самого внутреннего цикла. Например:

```
// Использование оператора break во вложенных циклах.
class BreakLoop3 {
public static void main(String args[]) {
for (int i=0; К3; 1++) {
System.out.print("Проход " + i + ": ");
for (int j=0; j<100; j++) {
if (j == 10) break; // выход из цикла, если j равно 10
System.out.print (j + " ");
}
System.out.println ();
}
System.out.println("Циклы завершены.");
}
}
```

Эта программа генерирует следующий вывод:

```
Проход 0: 0 1 2 3 4 5 6 7 8 9
Проход 1: 0 1 2 3 4 5 6 7 8 9
Проход 2: 0 1 2 3 4 5 6 7 8 9
Циклы завершены.
```

Как видите, оператор break во внутреннем цикле может приводить к выходу только из этого цикла. На внешний цикл он не оказывает никакого влияния.

При использовании оператора break необходимо помнить следующее. Во-первых, в цикле можно использовать более одного оператора break. Однако при этом следует соблюдать осторожность. Как правило, применение слишком большого числа операторов break приводит к деструктуризации кода. Во-вторых, оператор break, который завершает последовательность операторов в операторе switch, оказывает влияние только на данный оператор switch, а не на какие-либо содержащие его циклы.

Использование оператора continue

Иногда требуется, чтобы повторение цикла осуществлялось с более раннего оператора его тела. То есть на данной конкретной итерации может требоваться продолжить выполнение цикла без обработки остального кода в его теле. По сути, это означает переход в теле цикла к его окончанию. Для выполнения этого действия служит оператор continue. В циклах while и do-while оператор continue вызывает передачу управления непосредственно управляющему условному выражению цикла. В цикли for управление передается вначале итерационной части цикла for, а потом условному выражению. Во всех этих трех циклах любой промежуточный код пропускается.

Ниже приведен пример программы, в которой оператор continue используется для вывода двух чисел в каждой строке.

```
// Демонстрация применения оператора continue.
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("");
}
}
}
```

В этом коде операция % служит для проверки четности значения переменной i. Если оно четное, выполнение цикл продолжается без перехода к новой строке. Программа генерирует следующий вывод:

```
0 1 2 3 4 5 6 7 8 9
```

Как и оператор break, оператор continue может содержать метку содержащего его цикла, который нужно продолжить. Ниже показан пример программы, в которой оператор continue применяется для вывода треугольной таблицы умножения чисел от 0 до 9.

```
// Использование оператора continue с меткой.
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();
```

```
}
```

В этом примере оператор continue прерывает цикл подсчета значений j и продолжает цикл со следующей итерации цикла подсчета i. Вывод этой программы имеет следующий вид:

```
o 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
```

Удачные применения оператора continue встречаются редко. Одна из причин состоит в том, что Java предлагает широкий выбор операторов цикла, удовлетворяющих требованиям большинства приложений. Однако в тех случаях, когда требуется более раннее начало новой итерации, оператор continue предоставляет структурированный метод выполнения этой задачи.

Оператор return

Последний из управляющих операторов — return. Его используют для выполнения явного возврата из метода. То есть он снова передает управление объекту, который вызвал данный метод. Как таковой этот оператор относится к операторам перехода. Хотя полное описание оператора return придется отложить до рассмотрения методов в главе 6, все же кратко ознакомимся с его особенностями.

Оператор return можно использовать в любом месте метода для возврата управления тому объекту, который вызвал данный метод. Таким образом, оператор return немедленно прекращает выполнение метода, в котором он находится. Следующий пример иллюстрирует это. В данном случае оператор return приводит к возврату управления системе времени выполнения Java, поскольку именно она вызывает метод main ().

```
// Демонстрация использования оператора return.
class Return {
public static void main(String args[]) {
boolean t = true;
System.out.println("До выполнения возврата.");
if (t) return; // возврат к вызывающему объекту
System.out.println("Этот оператор выполняться не
будет.");
}
}
```

Вывод этой программы имеет вид:

```
До выполнения возврата.
```

Как видите, заключительный оператор println () не выполняется. Сразу после выполнения оператора return программа возвращает управление вызывающему объекту.

И последний нюанс: в приведенной программе использование оператора if (t) обязательно. Без него компилятор Java сигнализировал бы об ошибке "unreachable code" ("недостижимый код"), поскольку выяснил бы, что последний оператор println () никогда не будет выполняться. Во избежание этой ошибки в демонстрационном примере пришлось ввести компилятор в заблуждение с помощью оператора if.

5 Порядок выполнения работы

1. Выделить ключевые моменты задачи;
2. Построить алгоритм и теоритическую объектную модель решения задачи;
3. Запрограммировать полученный алгоритм и объектную модель;
4. Провести тестирование полученной программы.

6 Форма отчета о работе

Лабораторная работа № ___

Номер учебной группы _____
Фамилия, инициалы учащегося _____
Дата выполнения работы _____
Тема работы: _____
Цель работы: _____
Оснащение работы: _____
Результат выполнения работы: _____
_____
_____

7. Контрольные вопросы и задания

1. Назовите известные вам циклы языка Java.
2. Как на базе цикла **for** реализовать бесконечный цикл?
3. Как на базе цикла **while** реализовать бесконечный цикл?
4. С помощью каких операторов можно досрочно прервать работу цикла?
5. С помощью какого оператора можно досрочно перейти к следующей итерации цикла?

8. Рекомендуемая литература

1. **Урванов, Ф. В.** Java. Состояние языка и его перспективы. / Ф.В. Урванов. - Санкт-Петербург : БХВ-Петербург, 2023. - 368 с.
2. **Копец Дэвид.** Классические задачи Computer Science на языке Java. - Санкт-Петербург : Питер, 2022. - 288 с
3. **Васильев А. Н.** Java. Объектно-ориентированное программирование: Учебное пособие / А.Н. Васильев. - Санкт-Петербург : Питер, 2021. - 400 с.
4. **Эванс Бенджамин**. Java для опытных разработчиков. 2-е изд. — (Серия «Библиотека программиста»). - Санкт-Петербург : Питер, 2024. - 736 с.
5. **Лой Марк.** Программируем на Java. 5-е межд. изд. . - Санкт-Петербург : Питер, 2023. - 544 с.
6. **Гетц Брайан**. Java Concurrency на практике. - Санкт-Петербург : Питер, 2021. - 464 с.