Конспект по JAVA

Оглавление

[Операции в Java 4](#_Toc187954362)

[Операции сравнения 4](#_Toc187954363)

[Операнды и результат логических операций 4](#_Toc187954364)

[Результаты применения операций &, |, ^, ! 4](#_Toc187954365)

[Приоритеты операций 5](#_Toc187954366)

[Преобразование типов в выражениях 6](#_Toc187954367)

[Ввод и вывод. Вычисления 7](#_Toc187954368)

[Вывод 7](#_Toc187954369)

[Методы вывода на консоль 7](#_Toc187954370)

[Спецификаторы форматирования 7](#_Toc187954371)

[Округление числа до требуемого количества символов после запятой 8](#_Toc187954372)

[Ввод 8](#_Toc187954373)

[Импорт Scanner 8](#_Toc187954374)

[Создание Scanner 8](#_Toc187954375)

[Закрытие Scanner 9](#_Toc187954376)

[Математические функции в Java 9](#_Toc187954377)

[Оператор ветвления 10](#_Toc187954378)

[Оператор ветвления if 10](#_Toc187954379)

[Оператор return 10](#_Toc187954380)

[Оператор switch 11](#_Toc187954381)

[Операторы цикла 13](#_Toc187954382)

[Цикл с предусловием While() 13](#_Toc187954383)

[Оператор цикла ***for***() 13](#_Toc187954384)

[Цикл с постусловием ***do-while***: 14](#_Toc187954385)

[Цикл в стиле for each 15](#_Toc187954386)

[Коллекция List 16](#_Toc187954387)

[Использование 16](#_Toc187954388)

[Синтаксис 16](#_Toc187954389)

[Методы 16](#_Toc187954390)

[Добавление элемента в List 16](#_Toc187954391)

[Получение значения элемента List 16](#_Toc187954392)

[Замена значения элемента List 16](#_Toc187954393)

[Удаление элемента из List 16](#_Toc187954394)

[Размер List 16](#_Toc187954395)

[Очистка List 16](#_Toc187954396)

[Добавление значений другого List 16](#_Toc187954397)

[Удаление значений другого List 16](#_Toc187954398)

[Проверка наличия элемента в list 17](#_Toc187954399)

[Итератор 18](#_Toc187954400)

[Определение 18](#_Toc187954401)

[Вызов итератора 18](#_Toc187954402)

[Некоторые методы итератора 18](#_Toc187954403)

[Коллекция SET 19](#_Toc187954404)

[Определение 19](#_Toc187954405)

[Синтаксис 19](#_Toc187954406)

[Методы 19](#_Toc187954407)

[Коллекция Map 20](#_Toc187954408)

[Определение 20](#_Toc187954409)

[Синтаксис 20](#_Toc187954410)

[Методы 20](#_Toc187954411)

[Добавление и изменение значения в map 20](#_Toc187954412)

[Проверка наличия ключа в map 20](#_Toc187954413)

[Проверка наличия значения ключа в map 20](#_Toc187954414)

[Получение множества всех ключей 20](#_Toc187954415)

[Исключение 21](#_Toc187954416)

[Определение 21](#_Toc187954417)

[Синтаксис 21](#_Toc187954418)

[Try, catch, finally 21](#_Toc187954419)

[Throw 21](#_Toc187954420)

[Классы и объекты 22](#_Toc187954421)

[Определение 22](#_Toc187954422)

[Ключевое слово this 22](#_Toc187954423)

[Модификаторы 22](#_Toc187954424)

[Методы экземпляра класса String 23](#_Toc187954425)

# Операции в Java

## Операции сравнения

Операции равнения дают в результате логическое значение типа boolean:

* == равно
* !=  не равно
* > больше
* < меньше
* >=  больше или равно
* <=  меньше или равно

## Операнды и результат логических операций

Операнды и результат логических операций, имеют тип boolean:

* & (И - результат true, если оба операнда true)
* | (ИЛИ - результат true, если хотя бы один операнд true)
* ^ (Исключающее ИЛИ - результат true, если только один из операндов true)
* && (Укороченное И)
* || (Укороченное ИЛИ)
* ! (НЕ - true меняется на false, и наоборот)

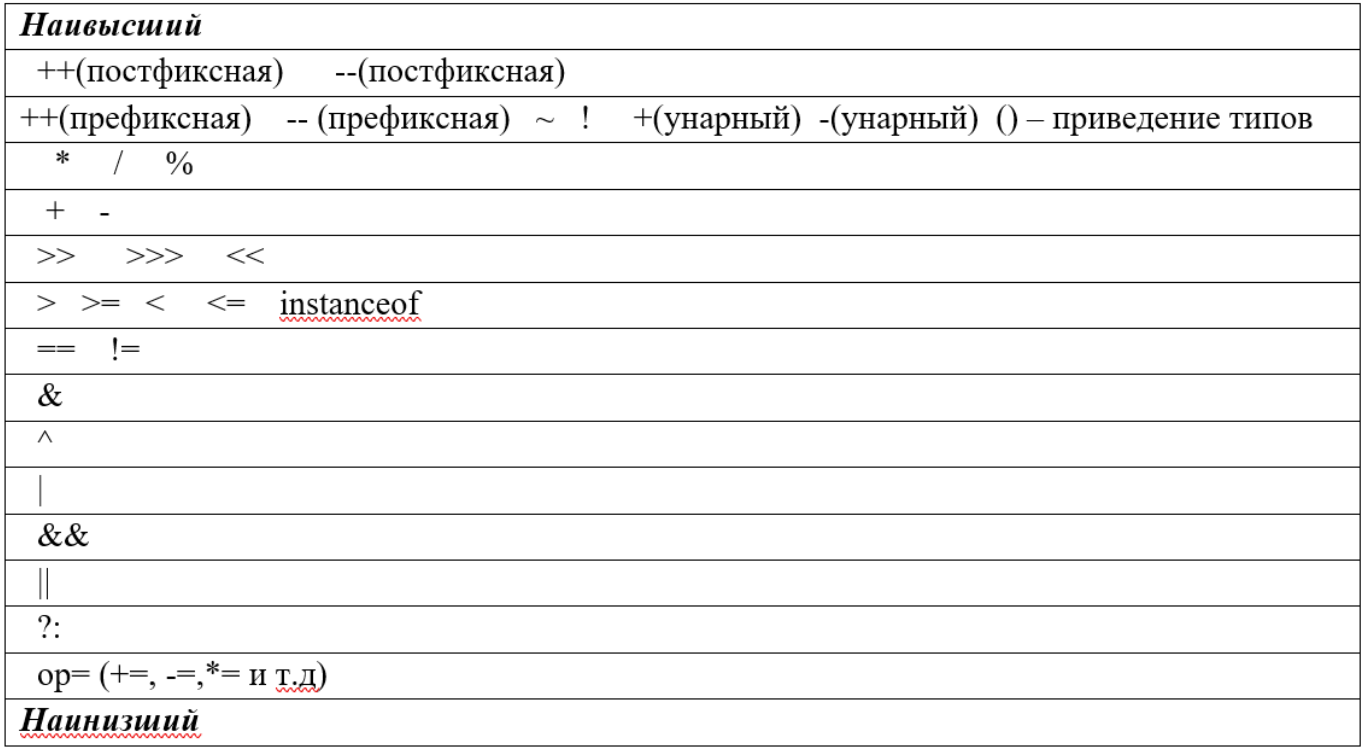
*!!! Укороченные варианты логических операций И и ИЛИ позволяют выполнять быструю оценку значений логических выражений. Если по значению первого операнда очевиден результат, то второй операнд не вычисляется. Это экономит время и повышает эффективность кода!!!*

## Результаты применения операций &, |, ^, !

Результаты применения операций &, |, ^, ! показаны в таблице:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| p | q | p&q | p|q | p^q | !p |
| false | false | false | false | false | true |
| true | false | false | true | true | false |
| false | true | false | true | true | true |
| true | true | true | true | false | false |

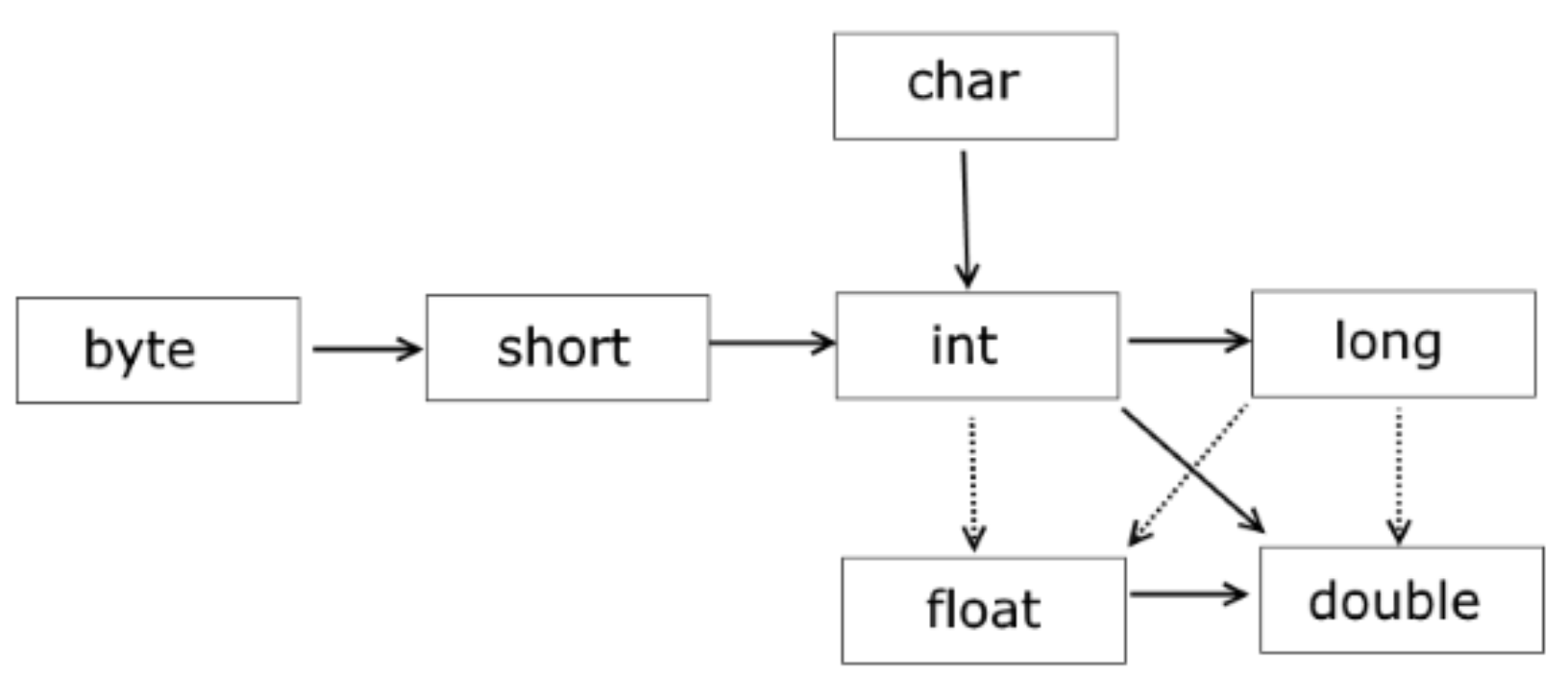
## Приоритеты операций



# Преобразование типов в выражениях

При выполнении операции присваивания  выполняется автоматическое (неявное) преобразование типов при условии, что оба типа (в левой и правой части оператора) являются ***совместимыми***, а целевой тип (в левой части) обладает более широким диапазоном допустимых значений. При этом происходит ***расширение*** типа. Порядок расширения задается цепочкой:

**byte ->** **short->** **int->** **long ->** **float ->** **double**.



# Ввод и вывод. Вычисления

## Вывод

### Методы вывода на консоль

Метод для вывода на консоль:  
System.out.print();

Метод для вывода на консоль и перенос на новую строку:  
System.out.println();

Метод для вывода на консоль и форматироания:  
System.out.printf();

### Спецификаторы форматирования

Спецификаторы форматирования, которые можно использовать в методе printf(), показаны в таблице:

|  |  |
| --- | --- |
| Спецификатор формата | Выполняемое форматирование |
| %s | Строковое представление аргумента |
| %b | Логическое (булево) значение аргумента |
| %c | Символьное представление аргумента |
| %d | Десятичное целое значение аргумента |
| %o | Восьмеричное целое значение аргумента |
| %x | Шестнадцатеричное целое значение аргумента |
| %f | Десятичное вещественное значение с плавающей точкой |
| %e | Экспоненциальное представление вещественного аргумента |
| %g | Выбирает более короткое представление из двух: %е или %f |
| %a | Шестнадцатеричное значение с плавающей точкой |
| %t | Время и дата |
| %n | Вставка символа новой строки |
| %h | Хэш-код аргумента |
| %% | Вставка знака % |

### Округление числа до требуемого количества символов после запятой

double x = 3.123456;   
System.out.printf("x=%8.2f\n", x);

//число выводится в 8 позициях //и округляется до 2 знаков после запятой

## Ввод

### Импорт Scanner

Ввод строки или числа из консоли проще всего выполнить с помощью объекта класса **Scanner.**

Этот класс необходимо импортировать из библиотеки java.util (которая по умолчанию не подключается). Для этого в начало файла добавляется оператор

import java.util.Scanner;

### Создание Scanner

Перед использованием объект класса **Scanner**должен быть создан. Это выполняется в строке кода:

Scanner scan = new Scanner(System.in);

Здесь scan - это имя объекта класса Scanner (можно выбрать любое свое имя). В конструктор объекта класса **Scanner** передается объект System.in, связанный со стандартным потоком от клавиатуры.  Создание объектов классов подробно обсуждается в курсе по ООП.

В классе **Scanner** есть множество методов для ввода данных и одновременного преобразования их к нужному типу. Так, метод next() считывает строку до пробела. Следовательно, имя, состоящее из нескольких слов, им ввести нельзя. А вот метод nextLine() читает всю строку (до нажатия Enter).

Метод nextInt() считывает из входного потока строку до пробела и преобразует в целое типа int (аналогично метод nextDouble() считывает и преобразует в число типа double, и т.д.)

### Закрытие Scanner

Рекомендуется всегда закрывать потоки для того, чтобы освобождать ресурсы компьютера. Т.е. после работы со **Scanner** нужно написать scan.close(). Но нужно учесть, что метод close() закрывает не сам **Scanner**, а поток, с которым он связан. В наших примерах это стандартный поток System.in, связанный с клавиатурой. Если Вы его закроете, то в дальнейшем Вы не сможете создать другой**Scanner,** чтобы продолжать считывать данные с клавиатуры:

## Математические функции в Java

Класс Math входит в пакет java.lang, который подключается по умолчанию. Поэтому его не нужно специально импортировать в начале программы (как, например, класс Scanner).

 Наиболее часто используемые методы класса Math:

* sqrt(x) — извлекает квадратный корень из числа x.
* pow(x, y) —  x возводится в степень y (оба параметра и результат вещественные).
* sin(x), cos(x), tan(x) — тригонометрические функции sin, cos и tg угла x, указанного в радианах.
* asin(x), acos(x), atan(x) — обратные тригонометрические функции, возвращают угол в радианах.
* exp(x) — возвращает значение экспоненты, возведенной в степень x (ex*ex*).
* log(x) — возвращает значение натурального логарифма числа x (lnx*lnx*).
* log10(x) — возвращает значение десятичного логарифма числа x (lgx*lgx*).
* abs(x) — возвращает модуль числа x.
* round(x) — округляет вещественное число до ближайшего целого по правилам математики
* ceil(x) - округляет вещественное число до целого в большую сторону.
* floor(x) - округляет вещественное число до целого в меньшую сторону.
* random() – возвращает вещественное случайное число от 0 до 1

Кроме методов, в классе Math имеется две константы: Math.E (2,718281.. – основание натурального логарифма) и Math.PI (3,141592.. – число Пи).

# Оператор ветвления

## Оператор ветвления if

Условный оператор ***if***позволяет разветвить алгоритм в зависимости от значения некоторого выражения:

if (условие) оператор1;

else оператор2;

Условие – это выражение типа boolean. Если условие истинно, то выполняется оператор1, а если ложно – оператор2.

Ветка else не является обязательной:

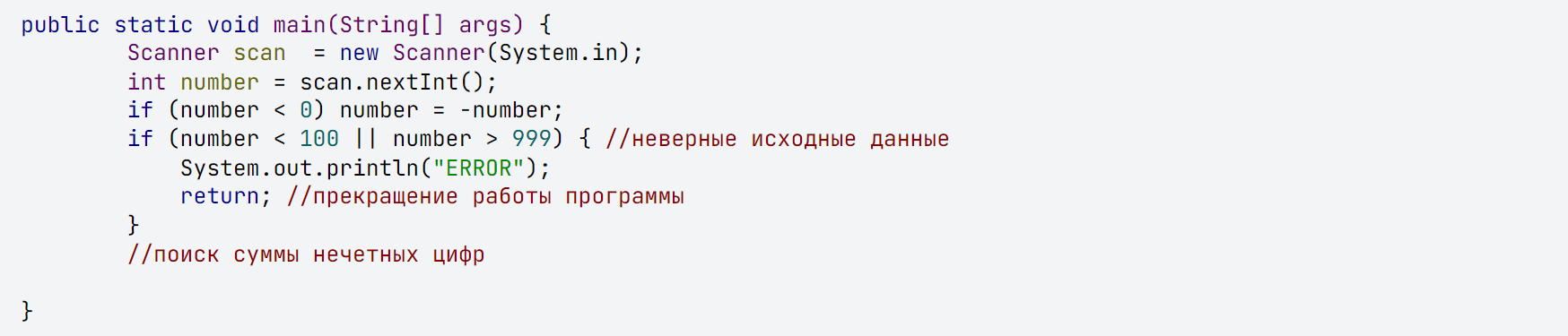
if (условие) оператор;

В этом случае если условие истинно, то выполняется оператор. А если условие ложно, то не выполняется ничего.

## Оператор return

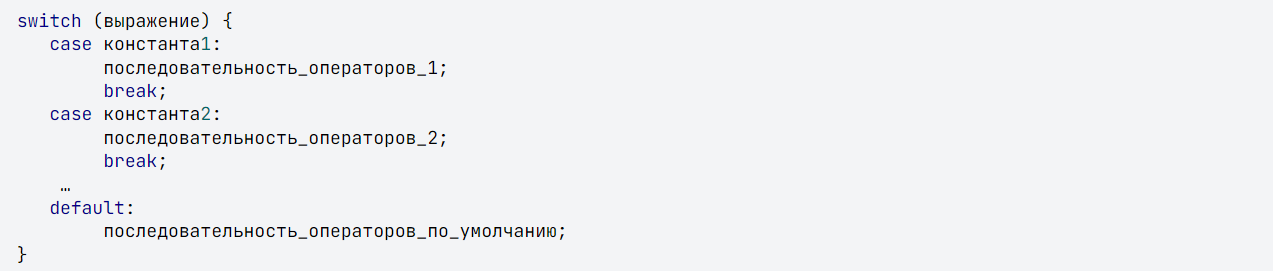
Этот оператор выхода из метода. Он позволяет вернуть управление в тот метод, который ранее его вызвал. Если этот оператор применяется в теле метода main(), то это означает прекращение выполнения программы.

Используя return, можно после ввода исходных данных сразу проверить их корректность и прекратить программу после вывода соответствующего сообщения. Например, в следующей задаче нужно ввести трехзначное число и найти сумму его нечетных цифр. Но если число не трехзначное, то нужно просто вывести "ERROR" и закончить программу. Избежать вложенных операторов if позволит такой код:



# Оператор switch

Оператор выбора ***switch***  обеспечивает многовариантное ветвление программы:



Выражение, управляющее оператором switch, должно быть типа byte, short, int, char или перечислением. Начиная с версии JDK 7 это выражение может иметь тип String .

Если значение выражения в скобках switch совпадает с какой-либо константой, то выполняется соответствующая  последовательность операторов, а при достижении оператора break; происходит выход из конструкции.

Если же ни одна из констант выбора не совпала со значением выражения, то выполняются операторы после слова default. Ветка default может отсутствовать, тогда никаких действий вообще не выполняется.

Оператор break; может отсутствовать (но, как правило, он применяется). При отсутствии break; выполнение программы **продолжается** в следующей ветви case.

Порядок расположения блоков case и default не имеет значения, но принято располагать блоки case по возрастанию значения, а default в самом конце.

Пример:

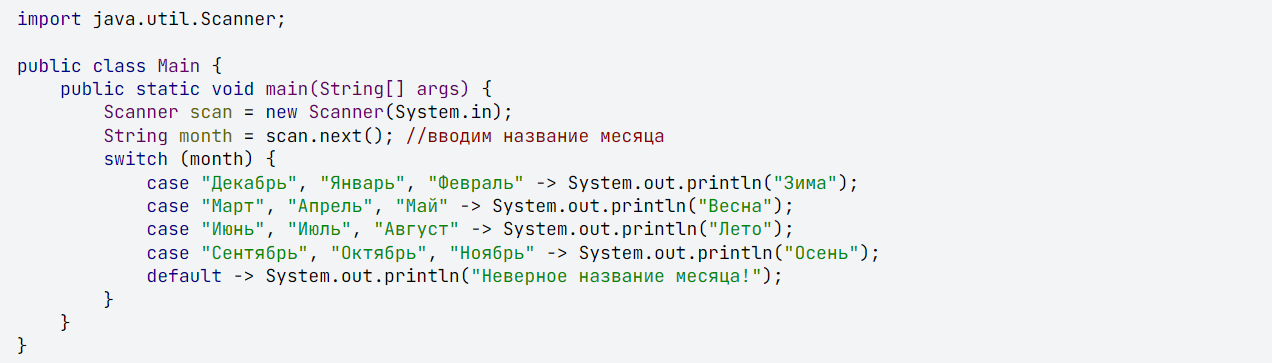


Начиная с Java 12 появился новый синтаксис оператора switch, основанный на использовании стрелки вместо двоеточия (лямбда-синтаксис). Отметим его особенности:

1) Не требуется break; в конце каждой ветки. Код не "проваливается" на другую ветку.

2) Можно использовать несколько меток в одном case, перечисляя их через запятую.

**Пример 1**. Перепишем в новом стиле программу, в которой по месяцу выводится название сезона:



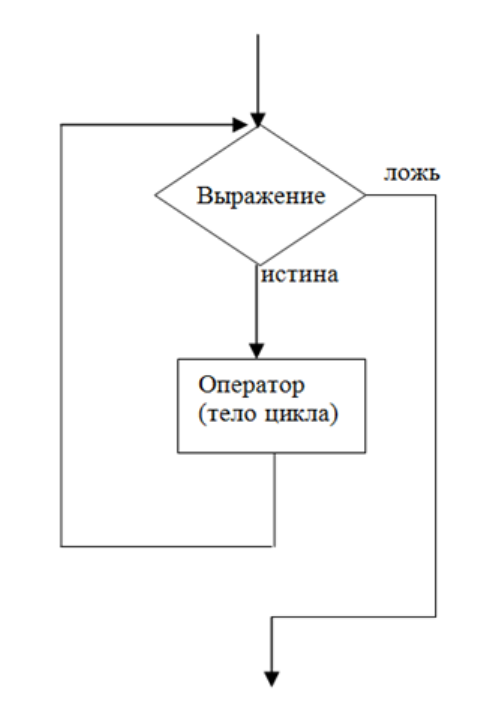
# Операторы цикла

## Цикл с предусловием While()

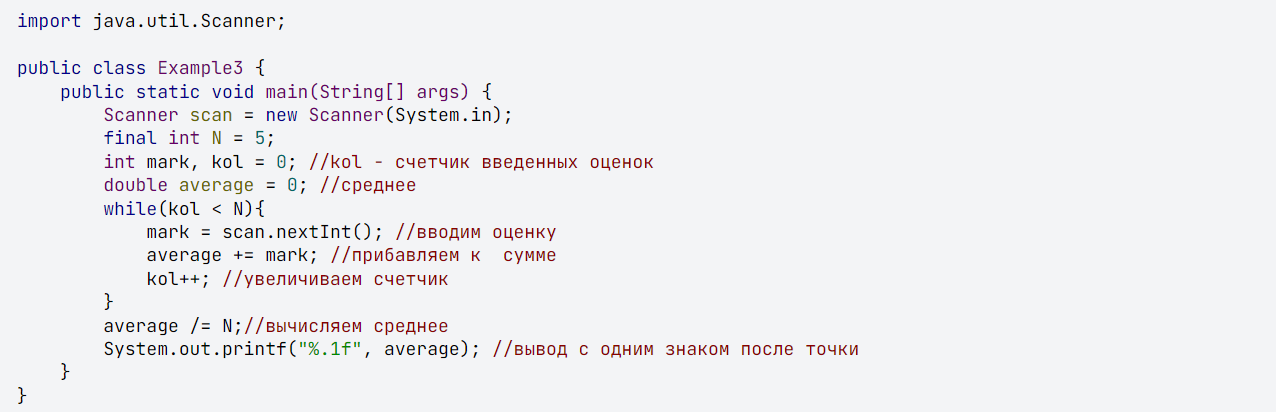
**while(условие) оператор;**

Пока условие истинно, выполняется оператор (тело цикла). Если в теле цикла должно быть несколько операторов, то они объединяются фигурными скобками.

Условие - это  логическое выражение (типа boolean).  Оно**не может**иметь целочисленный тип, как в C++. Проверка условия выполняется до начала выполнения тела цикла. Поэтому  цикл может вообще не выполнится ни разу, если условие изначально было ложным.



Пример:

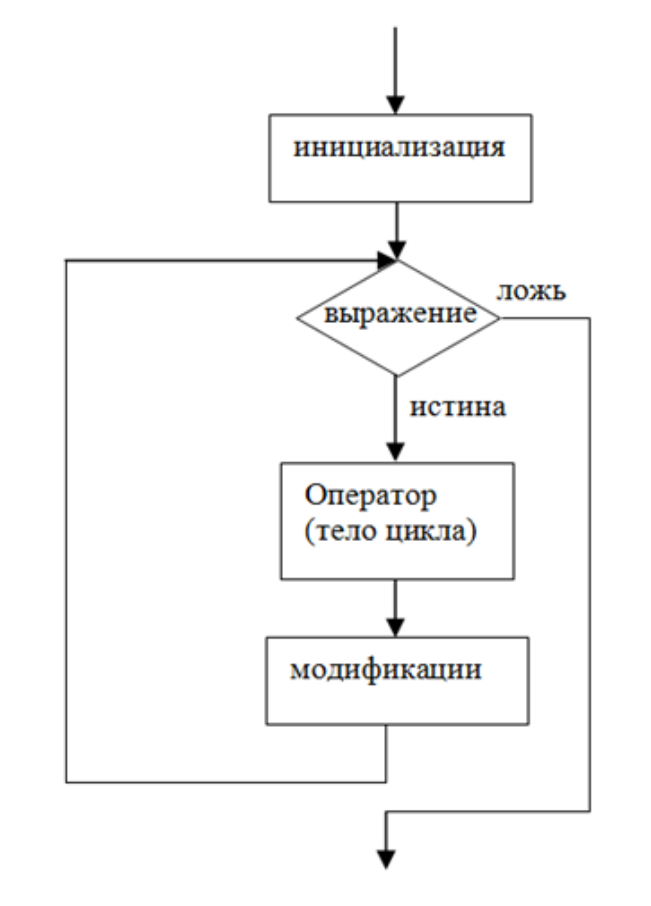


## Оператор цикла ***for***()

Оператор цикла ***for*** в общем случае предназначен для выполнения цикла с известным числом повторений. Традиционная форма этого цикла имеет вид:

**for(инициализация; условие\_повторения; модификация)   оператор;**

Инициализация – это оператор (или несколько операторов через запятую), которые выполняются один раз до начала цикла. Условие\_повторения – это логическое выражение (типа boolean). Цикл  выполняется до тех пор, пока условие повторения не станет ложным. Модификация – это оператор (группа операторов через запятую), которые выполняются на каждом шаге после тела цикла перед переходом к проверке условия повторения.



Код в цикле может вообще не выполниться, если проверяемое условие с самого начала окажется ложным.

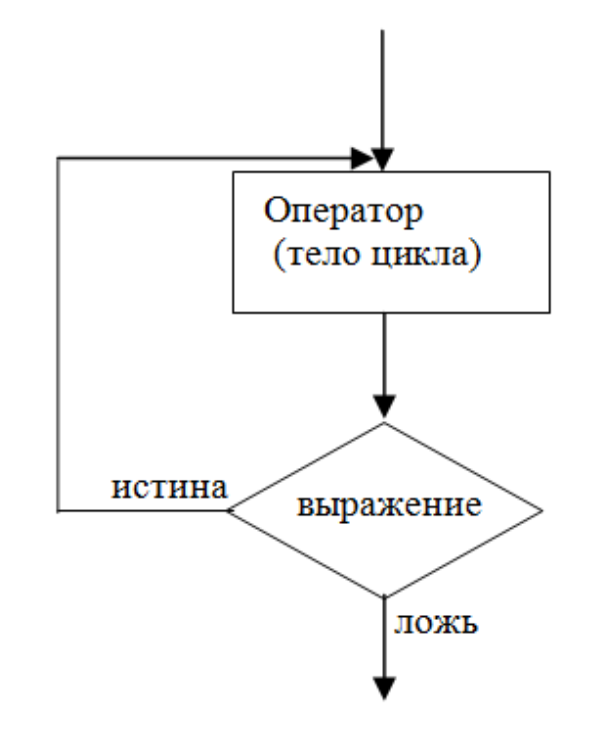
Любая часть цикла for может отсутствовать, но точки с запятыми остаются). По умолчанию пропущенное условие считается истинным.

Если в разделе инициализации объявляется переменная, то область ее действия ограничивается телом цикла.

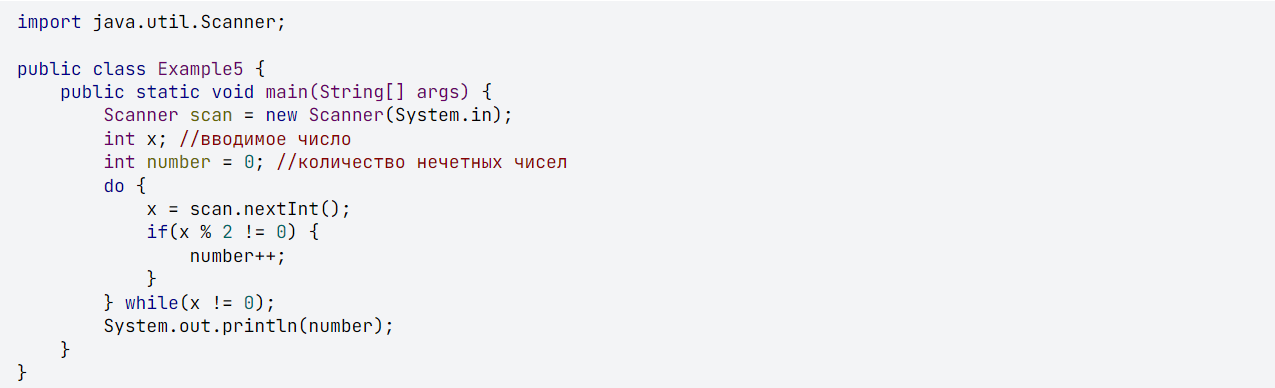
## Цикл с постусловием ***do-while***:

**do  оператор; while (условие);**

Этот цикл всегда выполняется хотя бы один раз. После выполнения тела цикла проверяется условие выполнения. Если оно истинно, то цикл выполняется еще раз. А если ложно – выполнение цикла заканчивается.

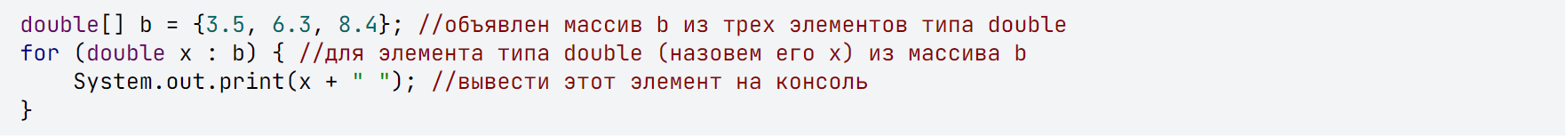


Пример:



## Цикл в стиле for each

Этот вид цикла упрощает  доступ к массивам и коллекциям. Его формат  
**for (тип\_данных имя\_временное : имя\_коллекции)    операторы;**  
Ниже показан пример вывода значений вещественного массива b:



Ограничением этой формы цикла является тот факт, что элементы массива или коллекции перебираются строго последовательно. Например, нельзя пройти только по элементам с нечетными индексами, либо от конца массива к началу.

Кроме того, в цикле for each элементы массива доступны “***только для чтения***”, т.е. их нельзя изменить (поскольку итерационная переменная является копией элемента).

# Коллекция List

## Использование

**List в Java** — это **упорядоченная коллекция элементов, допускающая дублирование**. Это упорядоченный набор объектов, каждый элемент которого занимает определённую позицию в списке.

## Синтаксис

List<Тип> Имя = new ArrayList<>();  
List<Integer> list = new ArrayList<>();

## Методы

### Добавление элемента в List

list.add(значение элемента);

### Получение значения элемента List

list.get(индекс)  
list.get(1)  
a =list.get(1)

### Замена значения элемента List

list.set(индекс, новое значение)  
list.set(2, 100) – элементу с индексом 2 задал новое значение равное 100

### Удаление элемента из List

list.remove(индекс);  
list.remove(2);

Элемент с индексом 2 удален и его место заняло значение элемента с индексом 3

### Размер List

list.size()

### Очистка List

list.clear()

### Добавление значений другого List

list.addAll(название другого листа)  
list.addAll(list2)

### Удаление значений другого List

list.removeAll(название другого листа)  
list.removeAll(list2)

### Проверка наличия элемента в list

list.contains(значение элемента)   
list.contains(2)

# Итератор

## Определение

**Итератор в Java** — это **интерфейс, используемый для последовательного перемещения элементов в коллекции**. Он предоставляет такие методы, как hasNext(), next() и remove(), для перебора коллекций и выполнения манипуляций.

## Вызов итератора

Iterator<тип> имя = имя коллекции.iterator();  
Iterator<Integer> iter = list.iterator();

## Некоторые методы итератора

* **hasNext()**. Возвращает true, если итерация содержит больше элементов.
* **next()**. Возвращает следующий элемент в итерации. Если доступных элементов не осталось, метод выдаёт исключение NoSuchElementException.
* **remove()**. Удаляет следующий элемент в итерации. Этот метод может быть вызван только один раз за вызов next().

**Пример использования интерфейса Iterator**:

List<String> list = new ArrayList<String>();

list.add("Hello");

list.add("World");

Iterator<String> iterator = list.iterator();

while (iterator.hasNext()) {

String element = iterator.next();

System.out.println(element);

}

В этом примере создаётся список строк ArrayList и добавляются в него два элемента. Затем с помощью метода iterator() создаётся объект Iterator, который позволяет перебирать элементы списка.

# Коллекция SET

## Определение

**Set в Java** — это **интерфейс, который расширяет интерфейс Collection**. Он представляет собой **коллекцию уникальных элементов**, которые не могут повторяться. Данные хранятся без определённого порядка, но могут быть упорядочены с использованием компаратора. [1](https://ru.hexlet.io/qna/java/questions/kak-rabotaet-set-java)

**Set имеет несколько реализаций** в стандартной библиотеке Java, включая HashSet, TreeSet и LinkedHashSet:

* **HashSet** — хранит элементы без какого-либо порядка, используя хэш-таблицу для обеспечения быстрого доступа к элементам. [1](https://ru.hexlet.io/qna/java/questions/kak-rabotaet-set-java)
* **TreeSet** — хранит элементы в отсортированном порядке. [1](https://ru.hexlet.io/qna/java/questions/kak-rabotaet-set-java)
* **LinkedHashSet** — хранит элементы в порядке их добавления. [1](https://ru.hexlet.io/qna/java/questions/kak-rabotaet-set-java)

Set обычно используется для проверки наличия элемента в коллекции или для удаления дубликатов из коллекции.

!!!

## Синтаксис

Set<integer> имя = new HashSet<>();  
Set<integer> set = new HashSet<>();

## Методы

**!!! Методы Set почти аналогичны List (set.add(12)), но у Set отсутствует метод get.  
Для вывода значений требуется использовать итератор.  
Так же важно помнить, что у элементов отсутствуют индексы, тоесть удаление происходит не по индексу элемента, а по его значению !!!**

# Коллекция Map

## Определение

**Map в Java** — это **объект, который сопоставляет ключи со значениями**. Он хранит данные парами «ключ — значение», при этом каждый ключ может сопоставляться не более одного значения. [1](https://habr.com/ru/articles/696184/)

Некоторые популярные реализации интерфейса Map в Java:

* **HashMap**. Наиболее распространённая реализация. Использует хеш-таблицу для хранения пар ключ-значение и обеспечивает доступ к элементам за константное время в среднем случае. [4](https://dzen.ru/a/ZJfcK_3jwF1P6BoZ)

## Синтаксис

Map<тип ключа, тип значения ключа> название = new HashMap<>();  
Map<Integer, String> map = new HashMap<>();

## Методы

Некоторые методы соответствуют методам класса List

### Добавление и изменение значения в map

map.put(значение ключа, значение значения ключа);  
map.put(124, "pidor");

### Проверка наличия ключа в map

map.containsKey(значение ключа)  
map.containsKey(124)

### Проверка наличия значения ключа в map

map.containsValue(значение значения ключа)  
map. containsValue (“pidor”)

### Получение множества всех ключей

Set<Integer> set = map.keySet()  
далее прогоняем через итератор:

Iterator<Integer> inter = set.inter();  
while (inter.hasNext()) {  
System.*out*.println(map.get(inter.next()));  
}

# Исключение

## Определение

В java исключением называется любая ошибка, которая возникает в ходе выполнения программы. Это может быть несоответствие типов данных, деление на ноль, обрыв связи с сервером и многое другое. Операции по их поиску и предотвращению называются обработкой исключений.

## Синтаксис

### Try, catch, finally

try {  
 System.*out*.println(4 / 0);  
} catch (ArithmeticException e) {  
 System.*out*.println("Ошибка!");  
} finally {  
 System.*out*.println("Привет");  
}

В блоке try находится код, в котором может быть ошибка  
В catch мы указываем ошибку, которую хотим отловить: (ArithmeticException e) и действие, которое необходимо совершить, если ошибку отловили : System.*out*.println("Ошибка!");  
В блоке finally (**Не обязательный**) содержится код, который выполняется в не зависимости от того поймали ошибку или нет: System.*out*.println("Привет");

### Throw

**Throw в Java** — это **ключевое слово для явного выброса одного исключения** из метода или любого блока кода.  С его помощью можно выбрасывать как проверенные, так и непроверенные исключения.

if (b == 0) {  
 throw ArithmeticException("Значение не равно 0")  
}

# Классы и объекты

## Определение

***Класс*** - это тип данных, описывающий шаблон создания объектов. Объект, созданный на основании этого шаблона, называется ***экземпляром****класса*. Класс описывает *данные*, задающие свойства объектов класса, и *функции*, определяющие их поведение. При этом функции, определенные в классе, получают новое имя: *методы*.

## Ключевое слово this

Иногда требуется, чтобы метод ссылался на вызвавший его объект. Для этой цели в языке Java определено ключевое слово **this**.

Самая распространенная ситуация, когда нужно его использовать:  если локальная переменная (или параметр метода) имеет такое же имя, как и поле объекта. В этом случае локальная переменная "скрывает собой"  поле. А чтобы обратиться именно к полю объекта, нужно использовать конструкцию **this.имя\_поля** .

void changeSurname(String surname){

this.surname = surname;

}

## Модификаторы

***static*** - означает, что это поле одно на весь класс  
***final*** - являются константами и не могут быть изменены после инициализации. Хорошим стилем является имена таких полей писать прописными буквами.  
***private*** — члены класса доступны только внутри данного класса  
***package-private*** — члены класса доступны классам, находящимся в том же пакете  
***protected*** — члены класса доступны классам, находящимся в том же пакете, и подклассам — в других пакетах  
***public*** — члены класса доступны везде

# Методы экземпляра класса String

* **String concat(String s) (аналог «+») — слияние строк;**
* **boolean equals(Object ob) — сравнение строк с учетом регистра;**
* **boolean equalsIgnoreCase(String s) — сравнение строк без учета регистра;**
* **int compareTo(String s) и compareToIgnoreCase(String s) — лексикографическое сравнение строк с учетом и без учета их регистра. Метод осуществляет вычитание кодов первых различных символов вызывающей и передаваемой в метод строки и возвращает целое значение. Метод возвращает значение нуль в случае, когда equals() возвращает значение true;**
* **boolean contentEquals(StringBuffer ob) — сравнение строки и содержимого объекта типа StringBuffer;**
* **String substring(int n, int m) — извлечение из строки подстроки начиная с индекса n и заканчивая индексом m (не включая). Нумерация символов в строке начинается с нуля;**
* **String substring(int n) — извлечение из строки подстроки, начиная с позиции n до конца строки;**
* **int length() — определение длины строки;**
* **int indexOf(char ch)— определение позиции символа  в строке. Возвращает индекс первого появления символа, либо -1, если символ не обнаружен. Аналогично перегруженный метод int indexOf(String str)  поиска индекса начала подстроки str.**
* **int indexOf(char ch, int n), int indexOf(String str, int n) — определение позиции символа  или подстроки, поиск вправо начинается с позиции n. Возвращает -1, если символ/подстрока не обнаружен;**
* **int lastIndexOf(char ch), int lastIndexOf(String str) — определение последней позиции символа или подстроки. Возвращает -1, если символ/подстрока не обнаружен;**
* **int lastIndexOf(char ch, int n), int lastIndexOf(String str, int n) — определение позиции символа  или подстроки, поиск влево начинается с позиции n. Возвращает -1, если символ/подстрока не обнаружен;**
* **String toUpperCase()/toLowerCase() — преобразование всех символов вызывающей строки в верхний/нижний регистр;**
* **String replace(char с1, char с2) — замена в строке всех вхождений первого символа вторым символом. Есть перегрузка этого метода для замены одной подстроки другой:: String replace(String str1, String str2)**
* **String intern() — заносит строку в «пул» литералов и возвращает ее объектную ссылку. Пример использования этого метода показан далее.**
* **String trim() — удаление всех пробелов в начале и конце строки;**
* **char charAt(int position) — возвращение символа из указанной позиции (нумерация с нуля);**
* **boolean isEmpty() — возвращает true, если длина строки равна 0 (пустая строка);**
* **void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) — извлечение символов строки от символа с индексом srcBegin до символа с индексом srcEnd  (не включая) в массив символов dst , размещение в котором начинается с индекса dstBegin;**
* **String[] split(String regex) — поиск вхождения в строку заданного регулярного выражения (разделителя) и деление исходной строки в соответствии с этим на массив строк.**
* **String[] split(String regex, int limit) — деление исходной строки на массив строк, используя разделитель regex (регулярное выражение). Параметр limit означает максимальное количество строк, которое должно быть получено.**
* **Int Integer.parseInt(myString) перевод символа из String в int**

# Создание паузы

Для создание паузы (в примере 5 секунд) требуется следующий код:

Try {

Thread.sleep(5000);

} catch (InterruptedException e) {

e.printStackTrace();

}

# Ожидание перед выполнением следующего кода

try {  
 Thread.*sleep*(10000); //Ожидание в 10 секунд  
} catch (InterruptedException e) {  
 throw new RuntimeException(e);  
}