БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ
Кафедра веб-технологий и компьютерного моделирования

А. С. Кравчук, А. И. Кравчук, Е. В. Кремень

ЯЗЫК JAVA

ОСНОВЫ СИНТАКСИСА

Учебные материалы
для студентов специальности 1-31 03 08
«Математика и информационные технологии
(по направлениям)»

МИНСК
2022

УДК 004.432.045:004.738.5Java (075.8)
ББК 32.973.2-018.1я73-1
К78

Утверждено на заседании кафедры
веб-технологий и компьютерного моделирования
8 ноября 2022 г., протокол № 3

Рецензент
кандидат физико-математических наук,
доцент *Г. А. Расолько*

Кравчук, А. С.

К78 Язык Java. Основы синтаксиса : учеб. материалы / А. С. Кравчук, А. И. Кравчук, Е. В. Кремень. – Минск : БГУ, 2022. – 58 с.

Рассматриваются основы снтаксиса языка Java, типы данных, переменные и литералы, в том числе и специализированные типы для работы с данными без потери точности, преобразования базовых типов, простейшие средства ввода-вывода, возможности класса Math для проведения математических вычислений, а также современные приемы работы с датами. Издание ориентировано как на тех, кто не имеет опыта практического программирования на языке Java, так и на тех, кто хотел бы систематизировать и улучшить свои знания. В каждой теме приводится необходимый теоретический материал и код программ, что существенно ускоряет усваивание материала, а также способствует более квалифицированному подходу к программированию.

УДК 004.432.045:004.738.5Java (075.8)
ББК 32.973.2-018.1я73-1

Оглавление

Алфавит языка Java. Идентификаторы. Служебные слова

В алфавит Java входят:

1. все прописные и строчные буквенные символы в кодировке `Unicode`.
2. цифры: `0, 1, 2, … , 9`;
3. специальные знаки: `", {, }, [, ], ', (, ), +, ….`
4. незначащие (неотображаемые) символы: пробел, табуляция, ограничители строк (символ новой строки, возврат каретки, а также их комбинация).

Лексемы

Лексема – единица текста программы, которая не может быть разделена на более мелкие элементы и воспринимается как единое целое.

В языке Java обычно выделяют шесть классов лексем:

- идентификаторы (`identifiers`);
- ключевые слова (`key words`);
- литералы (`literals`);
- разделители (`separators`);
- инструкции (`statements`);
- операторы (`operators`).

Идентификаторы

Идентификатор – последовательность букв, цифр и символов подчеркивания (‘_’), начинающихся с буквы, символа подчеркивания или валютного символа `Unicode` (например, `$, £, ¥`).

Имена (идентификаторы) имеют пакеты, классы, интерфейсы, поля, методы, аргументы и локальные переменные.

Идентификаторы можно записывать символами `Unicode`, то есть, на любом удобном языке.

Длина имени не ограничена.

Пример:

```
i,   engine3, theCurrentTime,  the_current_time,
θ,   переменная,  TIME,    time
```

Прописные и строчные буквы различаются, т.е. два последних идентификатора различны.

Замечание.
Один символ подчеркивания не является идентификатором.

Идентификаторы не могут содержать специальных знаков, за исключением подчеркивания и валютных знаков. Согласно соглашению об именовании, имена переменных не должны начинаться с символа подчеркивания _ или знака валют, несмотря на то, что и то, и другое технически возможно. Обычно знак доллара и другие валютные знаки остаются в резерве для автоматического генерирования кода компилятором или другим препроцессором. Лучше избегать этих символов в создаваемых идентификаторах.

Замечание.
Хотя использование не-`ASCII` *идентификаторов является допустимым в Java, но обычно этим не пользуются и предпочитают символы латинского алфавита.*

В программировании существует несколько общепринятых форматов при написании имен. Согласно им идентификатор может состоять из нескольких слов:

- написанных в нижнем регистре слитно без разделителя (название формата: flat case),
- написанных в нижнем регистре и разделенных символом подчеркивания (пример: group_number, название формата: snake case),
- написанных без разделителя при этом каждое слово должно начинаться с заглавной буквы (пример: GroupNumber, название формата: PascalCase или StudlyCase),
- написанных слитно без пробелов, при этом первое слово начинается с буквы нижнего регистра, а каждое следующее слово внутри фразы пишется с прописной буквы (формат: camelCase).
- Подчеркивания с заглавными буквами, как в UPPER_CASE, обычно используются при объявлении констант. Этот формат написания идентификаторов известен как MACRO_CASE.

Замечание.
Большинство этих форматов в той или иной мере используются в Java.

Требования к написанию идентификаторов:

- имена пакетов всегда пишутся строчными буквами из набора ASCII символов, используется формат flat case;
- имена классов и интерфейсов должны быть существительными в смешанном регистре с заглавной первой буквой каждого внутреннего слова, используется формат PascalCase;
- имена методов должны быть глаголами в смешанном регистре с первой буквой в нижнем регистре, с заглавной первой буквой каждого внутреннего слова (формат camelCase);
- для написания имен экземпляров класса и переменных используется camelCase;
- имена констант должны быть записаны в верхнем регистре со словами, разделенными символом подчеркивания ("_") (формат MACRO_CASE);
- ограничений на длину идентификатора не задокументировано;
- идентификатор не должен совпадать с ключевыми словами, с зарезервированными словами и именами методов библиотеки языка Java;
- следует избегать сокращений имен переменных до одного символа, за исключением временных «одноразовых» переменных. Общие имена для временных переменных: i, j, k, m и n для целых чисел, и c, d, и e для символов;
- по возможности следует использовать стандартные (общеупотребительные) сокращения слов в идентификаторах.
- Идентификатор должен быть не длинным, но нести смысл, поясняющий назначение именованного объекта в программе - любой просматривающий код программист по имени переменной должен догадываться о цели ее использования.
- следует избегать:
  - идентификаторов, которые различаются только одним или двумя символами;
  - идентификаторов, использующих цифры.

Ключевые слова

Ключевые слова – это зарезервированные идентификаторы, которые наделены определенным смыслом. Транслятор языка Java воспринимают только служебные слова, записанные строчными буквами. Далее приведены примеры ключевых слов.

Пример:

```
double, else,   enum,       extern
```

Как и в других языках программирования в качестве идентификаторов нельзя использовать ключевые слова и такие литералы как `true`, `false` и `null`, представляющие собой определенные значения и являющиеся частью самого языка Java.

Замечание.

Нет необходимости перечислять все служебные слова сразу, будем знакомиться с ними по мере освоения языка.

Комментарии

Комментарий – это набор символов, которые игнорируются при создании исполняемого кода. Комментарии нужны исключительно для людей, благодаря им программист может оставить пометки для себя и для других программистов. Что делает код более читабельным и понятным.

Начало *блочного (многострочного) комментария* начинается с символов `/*` и заканчивается символами `*/`. Все, что помещено между ними, игнорируется при создании исполняемого кода.

Блочные комментарии `/* */` не могут быть вложенными.

Хотя символами `/* */` можно оформлять и однострочный комментарий, однако в Java для этого используется специальная пара символов ’/’ (`//`), указывающая начало строки комментария. В этом случае концом комментария считается конец строки, так что нет необходимости отмечать его специальным символом. Этот способ наиболее полезен для коротких комментариев и является наиболее распространенным.

Замечание. Часто именно с помощью однострочных комментариев оформляются многострочные комментарии, для этого в каждой строке многострочного комментария в его начале ставится пара символов ’/’.

Есть еще документирующие комментарии, которые начинаются с `/**` и заканчивается на `*/`. Каждая строчка такого комментария должна начинаться с символа `*`. Такие комментарии при помощи специальной утилиты javadoc могут быть собраны в документацию к проекту.

Согласно «Соглашению о коде для языка программирования Java» все исходные файлы должны начинаться с комментария в стиле C, в котором перечислены имя класса, информация о версии, дата и уведомление об авторских правах:

```
/*
 * Classname
 *
 * Version information
 *
 * Date
 *
 * Copyright notice
 */
```

Типы данных

Известно, что числа могут быть натуральными, целыми, и вещественными. Так как объем памяти ограничен, то при решении реальных задач используются числа, не превышающие какого-то определенного значения.

Во всех языках программирования типы данных не только указывают формат представления чисел, но и устанавливают определенный диапазон возможных значений. Этот диапазон обычно определяется аппаратной организацией компьютера.

Тип – это атрибут, определяющий не только диапазон возможных хранимых значений, но и правила ее обработки.

Базовые (примитивные) типы

В Java имеются следующие *базовые типы*:

- `boolean`: хранит значение `true` или `false`:

```
boolean isActive = false;
boolean isAlive = true;
```

Замечание.

Величины булевского типа могут принимать только значения `false` и `true`, которые являются служебными словами.

- `byte`: хранит целое число от `-128` до `127` и занимает 1 байт:

```
byte a = 3;
byte b = 8;
```

- `short`: хранит целое число от `-32768` до `32767` и занимает 2 байта:

```
short a = 3;
short b = 8;
```

- `int`: хранит целое число от `-2147483648` до `2147483647` и занимает 4 байта:

```
int a = 4;
int b = 9;
```

- `long`: хранит целое число от `−9223372036854775808` до `9223372036854775807` и занимает 8 байт:

```
long a = 5;
long b = 10;
```

- `double`: хранит число с плавающей точкой от `±4.9*10-324` до `±1.8*10308` и занимает 8 байт:

```
double x = 8.5;
double y = 2.7;
```

- `float`: хранит число с плавающей точкой от `-3.4*1038` до `3.4*1038` и занимает 4 байта:

```
float x = 8.5F;
float y = 2.7F;
```

- `char`: хранит одиночный символ в кодировке UTF-16 и занимает 2 байта, поэтому диапазон хранимых значений от `0` до `65535`.

Следует различать разделители, используемые для отделения целой и дробной части в неименованных константах, используемых непосредственно в программном коде и применяемых при интерактивном вводе значения с клавиатуры:

- в качестве разделителя при использовании неименованной константы с плавающей точкой *непосредственно в программе* используется точка (как и в других языках программирования);
- при интерактивном вводе значения с клавиатуры разделители, используемые для выделения целой и дробных частей в числах с плавающей точкой, зависят от используемой локали (`Locale`).

Замечание.

Класс `Locale` *позволяет создать объект, описывающий географический или культурный регион, обеспечивая возможность создания многонациональных программ с учетом региональных настроек дат, времен, чисел, валюты и т. д.*

В учебных целях будем использовать установленную в онлайн-компиляторах «по умолчанию» локаль, а именно английский язык и страна США (“`en`”, “`US`”). При использовании этой локали в качестве разделителя целой и дробной части при вводе чисел с клавиатуры также используется точка.

Строки в Java

Символьные переменные не стоит путать со строковыми, 'a' не идентично "a". Строковые переменные представляют объект класса `String`, который в отличие от `char` не является базовым типом в Java.

Строка — это множество (возможно пустое) из символов, заключенное в двойные кавычки.

Пример.

```
String hello = "Hello...";
```

Замечание.

Для использования строкового типа нет необходимости, что-либо дополнительно подключать.

Переменные

Объявление переменной

Переменная - именованная область памяти, содержание которой меняется во время выполнения программы.

Доступ к значению *переменной* возможен через ее имя. Каждая *переменная* перед использованием в программе должна быть объявлена, т.е. ей должна быть выделена память.

Размер участка памяти, выделяемой для *переменной*, и интерпретация содержимого зависят от типа, указанного в определении переменной. Тип *переменной* изменить нельзя.

Формат объявления *переменных*:

```
тип списокИменПеременных;
```

Пример.

```
int x;
```

Переменные могут быть не только объявлены, но и определены (инициализированы). Формат инициализации:

```
тип имяПерем1 = знач1, …, имяПеремN = значN;
```

Язык программирования Java определяет следующие типы переменных:

- переменные экземпляра (к ним относятся нестатические поля класса);
- переменные класса или статические поля класса;
- локальные переменные или переменные, объявляемые в методах;
- параметры или переменные передаваемые в качестве параметров в метод.

Строгая (сильная) типизация языка Java

Строгая/сильная типизация выделяется тем, что язык не позволяет изменять тип переменной в ходе выполнения программы, а также смешивать в выражениях различные типы. Кроме того, выполнение автоматических неявных преобразований сильно ограничены.

Так, из-за строгой типизации языка Java, обычно, *переменная* базового типа может принимать только те значения, которые соответствуют ее типу. Если переменная представляет целочисленный тип, то она не может хранить числа с плавающей точкой.

Однако переменной *символьного типа* можно присвоить как одиночный символ, заключенный в одинарные кавычки (`char ch = 'e';`) так и целочисленное значение от `0` до `65535`. В этом случае переменная опять же будет хранить символ, а целочисленное значение будет указывать на номер символа в таблице символов `Unicode` (UTF-16).

Пример.

```
char ch=102; // символ 'f'
```

Кроме того ссылка на объект определенного класса может принимать значения ссылок на объекты другого класса, наследующего исходный.

Ключевое слово var

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

Пример.

```
var x = 10;
```

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

Если переменная объявляется с помощью `var`, то обязательно необходимо инициализировать ее, то есть предоставить ей начальное значение, иначе транслятор выдаст ошибку.

Пример.

```
var x;   // Ошибка, переменная не инициализирована
```

Замечание.

В связи со строгой типизацией языка Java при использовании ключевого слова `var` *тип переменной также не может меняться в дальнейшем в ходе выполнения программы*

Общие сведения о константах

Константа – данное, неизменяемое в процессе выполнения программы.

Константы, в отличие от переменных, являются фиксированными значениями, которые можно вводить и использовать в языке Java.

Различают четыре типа *констант*:

- целые;
- с плавающей запятой;
- символьные;
- строковые литералы.

Неименованные константы

Целые *неименованные константы* не имеют дробной части и не содержат десятичной точки. Они представляют целую величину в одной из следующих форм: десятичной, двоичной, восьмеричной, или шестнадцатеричной (Таблица 1).

Таблица 1 - Примеры неименованных целых констант

Десятичная константа	Двоичная константа	Восьмеричная константа	Шестнадцатеричная константа
16	0b10000	020	0x10
127	0b1111111	0177	0x7F
240	0b11110000	0360	0xF0

Десятичная *константа* состоит из одной или нескольких десятичных цифр, причем первая цифра не должна быть нулем (в противном случае число будет воспринято как восьмеричное).

Двоичная *константа* начинается с обязательной последовательности `0b` или `0B` и содержит `0` или `1`.

Восьмеричная *константа* состоит из обязательного нуля и одной или нескольких восьмеричных цифр.

Шестнадцатеричная *константа* начинается с обязательной последовательности `0x` или `0X` и содержит одну или несколько шестнадцатеричных цифр.

Если необходимо задать значение неименованной отрицательной константы в любом из перечисленных кодов, то необходимо явно указать знак «минус» перед соответствующим кодом числа (Таблица 2).

Таблица 2 - Примеры записи отрицательных неименованных целых констант

Десятичная константа	Двоичная константа	Восьмеричная константа	Шестнадцатеричная константа
`-16` `-127`	`-0b10000` `-0b1111111`	`-020` `-0177`	`-0x10` `-0x7F`

Неименованная константа с плавающей точкой – десятичное число, представленное в виде действительной величины с десятичной точкой:

~~знак~~ ~~цифры~~`.`~~цифры~~

или в экспоненциальной форме:

~~знак~~ ~~цифры~~`.`~~цифры~~ `E|e` ~~знак~~ ~~цифры~~

Число с плавающей точкой состоит из целой и дробной части и (или) экспоненты. «По умолчанию» *константы* с плавающей точкой имеют тип `double`.

Пример:

```
115.75, 1.5E-2, -0.025, .075, -0.85E2.
```

Замечание. Если число записывается без точки, то эта запись воспринимается компилятором как неименованная целочисленная константа.

Если программист хочет изменить размеры области памяти для хранения *неименованных констант*, то в Java он может использовать суффиксы:

- `F` (или `f`) – `float` (для *вещественных констант*);
- `L` (или `l`) – `long` (для *целых*).

Пример:

`3.14F` (*константа* типа `float`),
`3L` (*константа* типа `long`).

В связи со строгой типизацией языка Java использовать суффиксы иногда просто необходимо для того, чтобы избежать ошибки компиляции, связанной с невозможным преобразованием типа неименованной константы к типу переменной.

Для улучшения восприятия (для разделения групп цифр) возможно использование символов подчеркивания в числовых литералах. Может быть вставлено любое количество символов подчеркивания между цифрами.

Замечание.
Символы подчеркивания можно вставлять только между цифрами.

Нельзя размещать символы подчеркивания:

- вначале или в конце литерала,
- рядом с десятичной точкой в литерале с плавающей запятой,
- перед суффиксом `F` или `L`,
- в позиции, где ожидается цифра.

Пример:
`1_000_503.05F` (*константа* типа `float`),
`30_678L` (*константа* типа `long`).

Неименованные символьные константы можно разделить на две группы:

- печатные символы;
- непечатные символы.

Неименованная символьная константа в языке Java состоит либо из одного печатного символа, заключенного в апострофы (' ', 'Q'), либо специального управляющего кода (`Esc`-последовательности), заключенного в апострофы и начинающуюся с символа '\' (обратной косой черты, обратный слеш).

Кодируемые с помощью Esc-последовательности управляющие символы:

'\b' – забой (BackSpase);

'\t' – табуляция;

'\n' – конец строки + перевод каретки на новую строку;

'\f' – конец страницы;

'\r' – возврат каретки;

'\"' – двойная кавычка;

'\\' – обратная косая черта

'\uxxxx' – символ Unicode, где xxxx – шестнадцатеричный цифровой код символа Unicode.

'\xxx' – символ кодовой таблицы Latin-1, где xxx - восьмеричный код символа Latin-1.

Замечание.

Отобразить одинарную кавычку в составе строки можно с помощью esc-последовательности \', но одинарный символ «одинарная кавычка» вывести на экран не получается.

Неименованная строковая константа (строковый литерал) отличается от символьного литерала тем, что заключен в двойные кавычки и представляет собой комбинацию буквы, цифры, знаков препинания и прочих специальных символов или esc-последовательностей.

```
String text = "Вот мысль, которой весь я предан,\n"+
              "Итог всего, что ум скопил.\n"+
              "Лишь тот, кем бой за жизнь изведан,\n"+
              "Жизнь и свободу заслужил.";
```

С помощью операции + (конкатенация) можно присоединить к однострочному тексту еще один однострочный текст, но, например, расположенный в другой строке. Для того, чтобы при выводе текста происходил перенос на следующую строку, применяется esc-последовательность '\n'.

Современные версии Java поддерживает тестовые блоки (text blocks) - многострочный текст, облеченный в тройные кавычки. Предыдущий пример можно переписать следующим образом.

```
String text = """
              Вот мысль, которой весь я предан,
              Итог всего, что ум скопил.
              Лишь тот, кем бой за жизнь изведан,
              Жизнь и свободу заслужил.
              """;
```

Весь текстовый блок оборачивается в тройные кавычки, при этом не надо использовать конкатенацию строк и последовательность `'\n'` для их переноса на новую строку.

Таким образом Текстовые блоки позволяют упростить написание многострочного текста.

Именованные константы

Именованные константы – это именованная область памяти, содержание которой не изменяется во время выполнения программы.

Объявления констант примитивного типа

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, однако вначале идет ключевое слово `final`. Формат объявления констант:

```
final тип СПИСОК_ИМЕН_КОНСТАНТ;
```

Пример.

```
final int LIMIT = 5;
```

Константы позволяют задать такие именованные значения, которые не должны больше изменяться. Например, если есть переменная для хранения числа π, то можно объявить ее константой, так как ее значение постоянно.

Перечисления `enum`

Кроме отдельных примитивных типов данных и классов в Java есть такой тип как `enum` или перечисление. Перечисления представляют набор логически связанных констант. Объявление перечисления происходит с помощью ключевого слова `enum`, после которого идет название

перечисления, затем идет список элементов перечисления через запятую. Формат:

```
enum ИдентификаторПеречисления
{
    СПИСОК,
    ПЕРЕЧИСЛИТЕЛЕЙ
}
```

Идентификаторы перечислений начинаются с заглавной буквы, а *имена перечислителей вообще состоят только из заглавных букв*.

Пример:

```
enum Colors
{
    COLOR_YELLOW,
    COLOR_WHITE,
    COLOR_ORANGE
}
```

Замечание.

Перечисление в Java имеют стандартные методы, позволяющие вывести на экран список всех констант в перечислении (метод `values()`*), а также индекс конкретной константы (метод* `ordinal()`*).*

В отличие от C++ в `enum` из Java *отсутствует* автоматическое присваивание целочисленных значений перечислителям в зависимости от их позиции в списке перечисления.

Для того, чтобы присвоить *константные* значения перечислителям необходимо использовать доопределенный самим разработчиком конструктор, а для того, чтобы извлечь значение перечислителей еще и доопределенный пользователем метод (геттер). Все это будет рассмотрено позже.

Простейшая программа на Java

Программа – это последовательность команд (инструкций), которые размещаются в памяти и выполняются процессором в указанном порядке. Программа записывается на языке высокого уровня, наиболее удобном для реализации алгоритма решения определенного класса задач.

Инструкции. Блоки

Основным строительным блоком программы на языке Java являются инструкции (`statement`). Каждая инструкция выполняет некоторое действие, например, вызовы методов, объявление переменных и присвоение им значений. После завершения инструкции в Java ставится точка с запятой (`;`). Данный знак указывает компилятору на конец инструкции.

Пример.

```
char a = 'b';
```

Данная строка является инструкцией и представляет объявление и инициализацию переменной `a`. В данном случае вызов метода является инструкцией и поэтому завершается точкой с запятой.

Запись действий, которые должен выполнить компьютер (программа), состоит из инструкций.

Понятие оператор также используется в Java для определения средств управления ходом вычислительного процесса (например, операторы `if()`, `for()` и др.)

При выполнении программы инструкции выполняются одна за другой, за исключением операторов управления, которые могут изменить последовательное выполнение программы.

Кроме отдельных инструкций распространенной синтаксической конструкцией является блок кода. Блок кода содержит набор инструкций, он заключается в фигурные скобки, а инструкции помещаются между открывающей и закрывающей фигурными скобками. Формат:

```
{
    …
    список;
    инструкций;
    …
}
```

Пример.

```
{
    a = 'b';
    n = 5;
}
```

В этом блоке кода две инструкции, которые присваивают значения переменным.

Пустая инструкция

Простейшей формой инструкции является *пустая инструкция*:

```
;
```

Она ничего не делает. Однако она может быть полезна в тех случаях, когда синтаксис требует наличие инструкции, а она не нужна.

Выполнение программы. Метод `main()`

Java является объектно-ориентированным языком, поэтому всю программу можно представить как набор взаимодействующих между собой классов и объектов.

Формат простейшей программы имеет вид:

```
public class ИмяКлассаПользователя{
    //может отсуствовать
    список;
    свойств;
    //метод main() должен присутствовать в
    //единственном числе во всей программе
    //всегда
    public static void main (String args[]){
          список;
          инструкций и операторов управления программой;
          программы;
    }
}
```

То есть основу программы составляет класс `ИмяКлассаПользователя`. При определении класса вначале идет

спецификатор доступа `public`, который указывает, что данный класс будет доступен всем, то есть можно его запустить из командной строки. Далее идет ключевое слово `class`, а затем имя класса (`ИмяКлассаПользователя`) – произвольно выбираемый идентификатор. После названия класса идет блок кода, в котором расположено содержимое класса.

Замечание.
В ряде online-компиляторов (например, https://www.onlinegdb.com/online_java_compiler) имя класса `ИмяКлассаПользователя` *содержащего программу с именем* `main()` *заранее предопределено и не может быть другим, чем имя* `Main`.

В данном случае содержимое класса состоит только из одного метода `main()`, который имеет:

- спецификатор доступа `public`, указывающий на общедоступность данного метода извне;
- спецификатор `static` указывающий, что метод `main()` – статический, т.е. его при необходимости можно рекурсивно вызвать конструкцией `ИмяКлассаПользователя.main()` (или просто `main()`); пример будет рассмотрен в соответствующем разделе.

Ключевое слово `void` свидетельствует, что метод `main()` не возвращает никакого значения.

Далее в скобках идут параметры метода: `String args[]`. Это массив `args`, который хранит значения типа `String`, то есть строки. При запуске программы с помощью командной строки (консоли) через этот массив можно передать в программу некоторые текстовые значения.

После заголовка метода `main()` идет его блок, который содержит набор выполняемых инструкций.

В рамках языка Java метод `main()` называется *точкой входа* в программу. Этот метод обязательно должен присутствовать в программе в единственном числе. При этом его заголовок может быть только таким:

```
public static void main (String args[])
```

При запуске приложения виртуальная машина Java ищет в главном классе программы метод `main()`, и после его обнаружения запускает его.

Общие сведения по импорту пакетов и классов

Как правило, в Java классы объединяются в пакеты. Если описывать по аналогии, то пакеты в Java совмещают в себе правила работы, принятые в «пространстве имен» и библиотек в C++.

Организация классов в виде пакетов позволяет избежать конфликта имен между классами. Т.к. нередки ситуации, когда разработчики называют свои классы одинаковыми именами. Принадлежность к пакету позволяет гарантировать однозначность понимания значения идентификаторов.

«По умолчанию» Java имеет ряд *встроенных* пакетов, например, `java.lang`, `java.util`, `java.io` и т.д. Это, например, позволяет использовать класс `String`, входящий в пакет `java.lang`, непосредственно в программе без дополнительных действий.

Подключение уже существующего пакета к текущей программе осуществляется с помощью служебного слова `import`. Директива `import` (или если необходимо несколько директив последовательно) указывается в самом начале кода. В конце каждой директивы указывается `ИмяКласса`, который разработчик хотел бы подключить.

Например, если разработчику необходимо подключить класс `ИмяКласса` из пакета с именем `ИмяПакета` стандартной библиотеки языка Java, то ему необходимо воспользоваться инструкцией следующего формата:

```
import java.ИмяПакета.ИмяКласса;
```

Пример.

```
import java.util.Scanner;
```

Если разработчик планирует использовать несколько классов из пакета `ИмяПакета`, то для сокращения количества инструкций с директивами `import` язык Java допускает использование следующей синтаксической конструкции:

```
import java.ИмяПакета.*;
```

Пример.

```
import java.util.*;
```

На самом деле она указывает, что программист может использовать все классы пакета `ИмяПакета`.

Замечание.

Структура пакетов и классов неразрывно связана со структурой папок (каталогов) разрабатываемого проекта в файловой системе. Выражение `import java.ИмяПакета.*;` *означает импорт всех классов из папки с именем* `ИмяПакета`*, но если кроме классов папка* `ИмяПакета` *содержит еще внутренние папки (например, пакет* `ИмяВнутрПакета`*) со своими классами, то для того чтобы их также подключить необходимо написать две инструкции:*

```
import java.ИмяПакета.*;
import java.ИмяПакета.ИмяВнутрПакета.*;
```

Простейшие средства ввода-вывода языка Java

Наиболее простой способ взаимодействия с пользователем представляет консоль: можно выводить на консоль некоторую информацию или, наоборот, считывать с консоли некоторые данные. Для взаимодействия с консолью в Java применяется класс `System` (пакет `java.lang`, встроенный «по умолчанию»), а его функциональность обеспечивает консольный ввод и вывод.

Вывод на консоль

Для использования стандартного потока вывода в классе `System` определен объект `out`. В этом объекте определены методы `print()`, `println()` и `printf()`, которые позволяют вывести на консоль некоторое значение. Различие между ними заключаются в слудующем:

- `print()` - метод выводит строку на консоль и оставляет курсор в той же строке;
- `println()` - метод выводит строку на консоль и переводит курсор на новую строку;
- `printf()` - метод подготавливает форматированную (специально подготовленную) строку и выводит ее на консоль, оставляя курсов в той же строке.

Пример.

```
1 public class MyClass{
2     public static void main (String args[]){
```

```
3           System.out.println("Hello world!");
4           System.out.print("Bye world...");
5       }
6   }
```

В метод `println()` передается любое значение, как правило, строка, которое надо вывести на консоль.

Результат работы программы:

```
Hello world!
Bye world...
```

Рассмотрим пример использовать только метода `System.out.print()`.

Пример.

```
1  public class MyClass{
2      public static void main (String args[]){
3      System.out.print("Hello world! ");
4      System.out.print("Bye world...");
5      }
6  }
```

Результат работы программы:

```
Hello world! Bye world...
```

Однако с помощью `esc`-последовательность '`\n`', включенной в состав выводимой строки также можно осуществить перевод каретки на следующую строку даже с использованием метода `print()`.

Пример.

```
1  public class MyClass{
2      public static void main (String args[]){
3      System.out.print("Hello world \n");
4      System.out.print("Bye world...");
5      }
6  }
```

Нередко необходимо подставлять в строку какие-нибудь данные. Например, у нас есть два числа, и необходимо вывести их значения на экран. В этом случае можно, например, написать так:

Пример.

```
1  public class MyClass{
2      public static void main (String args[]){
3          int n=5;
4          int m=6;
5          System.out.println("n = " + n + "; m = " + m);
6      }
7  }
```

Результат работы программы:

```
n = 5; m = 6
```

Форматированный вывод

Как уже отмечалось метод `printf()` также определен в объекте `out` класса `System`. С его помощью можно переписать предыдущий пример следующим образом (хотя сам результат вывода значений на экран останется таким же как в предыдущем случае).

Пример.

```
1  public class MyClass{
2      public static void main (String args[]){
3          int n = 5;
4          int m = 6;
5          System.out.printf("n = %d; m = %d \n", n, m);
6      }
7  }
```

Результат работы программы совпадает с предыдущим результатом.

Символы `%d` обозначают спецификатор, вместо которого подставляет один из аргументов. Спецификаторов и соответствующих им аргументов может быть несколько. В данном случае у нас только два аргумента, поэтому вместо первого `%d` подставляет значение переменной `n`, а вместо второго - значение переменной `m`. Сама буква `d` означает, что данный спецификатор будет использоваться для вывода *целочисленных* значений.

Кроме спецификатора `%d` можно использовать еще ряд спецификаторов для других типов данных:

- `%x` - для вывода шестнадцатеричных чисел;
- `%f` - для вывода чисел с плавающей точкой;
- `%e` - для вывода чисел в экспоненциальной форме, например, `1.3e+01`;

- `%b` – вывод значения булевского типа;
- `%B` – вывод ЗАГЛАВНЫМИ БУКВАМИ значения булевского типа;
- `%c` - для вывода одиночного символа;
- `%C` - для вывода одиночного символа в ВЕРХНЕМ РЕГИСТРЕ;
- `%s` - для вывода строковых значений;
- `%S` - для вывода строки ЗАГЛАВНЫМИ БУКВАМИ;
- `%h` – вывод шестнадцатеричного кода.

Пример.

```
1  public class MyClass{
2      public static void main (String args[]){
3          String name = "Tom";
4          int age = 30;
5          float height = 1.7f;
6          System.out.printf("Name: %s \tAge: %d \tHeight: %.2f \n",
7                                  name, age, height);
8      }
9  }
```

Результат работы программы:

```
Name: Tom    Age: 30     Height: 1.70
```

При выводе чисел с плавающей точкой можно указать количество знаков после запятой, для этого используем спецификатор на `%.2f`, где запись «`.2`» указывает, что после запятой будет два знака.

Правила форматирования любого из спецификаторов:

`%флаг`**`Длина`**`.Точность`**`спецификаторТипа`**

где **`спецификаторТипа`** – обязательный параметр, любой из буквенных спецификаторов, указанных выше: `d`, `x`, `f`, `b`, `s` и т.д. Параметры: `флаг`, **`Длина`**, `Точность` являются опциональными, т.е. необязательными.

Параметр `флаг` может принимать несколько значений. Самые употребляемые:

- `-` `(минус)` – применяется для всех типов спецификаторов, выравнивание значения по левому краю («по умолчанию» осуществляется выравнивание по правому краю);
- `+` `(плюс)` – применяется для спецификаторов `d` и `f`, выводит на экран знак числового значения переменной (+ или -);

- `,` (запятая) - применяется для спецификаторов `d` и `f`, группирует с помощью запятой по три позиции в числе (для чисел больше 1000);
- пробел - применяется для спецификаторов `d` и `f`, при его использовании знак «минус» будет отображаться перед отрицательными числами и «пробел» будет ставиться вместо знака у положительных чисел.

Параметр **Длина** применяется для всех типов спецификаторов, задает минимальное пространство (количество символов), которое требуется для печати содержимого соответствующего аргумента.

Параметр `Точность` применяется только для спецификатора `f` и указывает количество цифр в дробной части числа, т.е. после точки.

Формирование форматированной строки методом `format()` класса `String`

Метод `format()` класса `String` возвращает отформатированную строку, используя прилагаемый формат и аргументы.

Метод по своей идеологии полностью соответствует уже рассмотренному методу `printf()`, однако в отличие от последнего только осуществляет подготовку строки без вывода ее на экран.

Первым параметром передается строка-шаблон, в которой, на местах, в которые необходимо поставить значения стоят ранее перечисленные спецификаторы (`%s`, `%d` и др.). После строки-шаблона передаются параметры, значения которых и будут подставлены на место указанных спецификаторов (`%s`, `%d` и др.).

Перепишем предыдущий пример с использованием метода `format()`.

Пример.

```
1  public class MyClass{
2      public static void main (String args[]){
3          String name = "Tom";
4          int age = 30;
5          float height = 1.7f;
6          String formatStr =
7                  String.format("Name: %s \tAge: %d \tHeight: %.2f \n",
8                                  name, age, height);
9          System.out.printf(formatStr);
10     }
11 }
```

Результат работы программы совпадает с предыдущим.

Замечание.

Форматированную строку можно вывести на экран любым методом `print()`, `println()` *или как в примере* `printf()`.

Ввод с консоли

Для использования стандартного ввода с консоли в том же классе `System` определен объект `in`. Однако непосредственно через объект `System.in` не очень удобно работать, поэтому, как правило, используют класс `Scanner` (пакет `java.util`). Например, напишем программу, которая осуществляет ввод чисел:

Пример.

```
1  import java.util.Scanner;
2
3  public class MyClass{
4      public static void main(String[] args) {
5          //сообщение
6          System.out.print("Input a number: ");
7          //организация ввода числа
8          Scanner objIn = new Scanner(System.in);
9          int num = objIn.nextInt();
10         //закрываем объект сканера
11         objIn.close();
12         //сообщение
13         System.out.printf("Your number: %d \n", num);
14
15     }
16 }
```

Так как класс `Scanner` находится в пакете `java.util`, то вначале его импортируем с помощью инструкции `import java.util.Scanner`.

В данном случае вначале выводим приглашение к вводу. Для создания самого объекта `objIn` класса `Scanner` используется его конструктор с параметром (объект `System.in`). После этого можно получать вводимые значения, с применяя метода `nextInt()` созданного объекта `objIn`, который возвращает введенное с клавиатуры целочисленное значение.

Замечание.

Необходимо обратить внимание на особенность работы с объектом `objIn` *класса* `Scanner` *– после считывания всех значений его необходимо закрыть вызовом метода* `close()`. *Это автоматически приведет к*

закрытию потока `in`. После этого второй раз открыть сканер, связанный с потоком `System.in`, не выйдет, поскольку `System.in` будет уже закрыт, и передача в `Scanner` `System.in` вызовет ошибку `NoSuchElementException`.

Результат работы программы:

```
Input a number: 5
Your number: 5
```

Принципиально можно обойтись и без использования ключевого слова `import`:

Пример.

```
1  public class MyClass{
2      public static void main(String[] args) {
3          //сообщение
4          System.out.print("Input a number: ");
5          //организация ввода числа
6          java.util.Scanner objIn =
7                      new java.util.Scanner(System.in);
8          int num = objIn.nextInt();
9          //закрываем объект сканера
10         objIn.close();
11         //сообщение
12         System.out.printf("Your number: %d \n", num);
13
14     }
15 }
```

Результат работы программы останется таким же, как и в предыдущем случае. Последний пример демонстрирует аналогию с применением пространства имен из C++.

Поскольку язык Java является строго типизированным, то класс `Scanner` имеет еще ряд методов, которые позволяют получить введенные пользователем значения других базовых типов:

- `next()` - считывает введенную строку до первого пробела;
- `nextLine()` - считывает всю введенную строку;
- `nextDouble()` - считывает введенное число `double`;
- `nextBoolean()` - считывает значение `boolean`;
- `nextByte()` - считывает введенное число `byte`;
- `nextFloat()` - считывает введенное число `float`;

- `nextShort()` - считывает введенное число `short`.

Рассмотрим пример ввода числа с плавающей точкой с помощью метода класса `Scanner`.

Пример.

```
1  import java.util.Scanner;
2
3  public class MyClass{
4      public static void main(String[] args) {
5          System.out.print("Input height: ");
6          Scanner in = new Scanner(System.in);
7          float height = in.nextFloat();
8          in.close();
9          System.out.printf("Height: %.2f \n", height);
10     }
11 }
```

Результат работы программы:

```
Input height: 4
Height: 4.00
```

Операции и выражения с использованием переменных и констант базовых типов

Операндом называется переменная, константа или выражение, участвующее в операции.

Комбинация знаков операций и операндов, результатом которой является определенное значение, называется *выражением*.

Замечание. Каждый операнд в выражении может быть выражением.

Операции бывают:

- унарные (участвует один операнд),
- бинарные (участвуют два операнда)
- условная (тернарная - три операнда).

Основные унарные операции

Унарные операции, в основном, имеют префиксный формат, т.е. знак операции `O` предшествует операнду:

```
O операнд
```

Однако для некоторых унарных операций используется и постфиксный формат, т.е. знак операции `O` следует за операндом:

```
операнд O
```

В таблице ниже (Таблица 3) можно ознакомиться со списком унарных операций.

Таблица 3 – Список унарных операций

Операции	Назначение
+	Унарный плюс, не выполняет никаких действий с арифметическим операндом
-	Унарный минус, меняет знак арифметического операнда
!	Логическое отрицание (НЕ). Меняет значение true или false логической переменной на противоположное (т.е. на false или true).
++	Инкремент (увеличение на единицу): префиксная форма увеличивает операнд до его использования, а постфиксная – после использования в выражении;
--	Декремент (уменьшение на единицу): префиксная форма уменьшает операнд до его использования, а постфиксная – после использования в выражении;

Примеры: `++i;` `j++;` `!n`

Операции инкремента и декремента (`++`, `--`) относятся к унарным арифметическим операциям, которые служат соответственно для увеличения или уменьшения значения, хранимого в переменной.

Операции инкремента и декремента не только изменяют значения переменных, но и возвращают значения. Таким образом, их можно сделать частью более сложного выражения.

Имеется постфиксная и префиксная формы операций инкремента и декремента. В постфиксной форме записи значение переменной, к которой

применена операция, увеличивается (или уменьшается) только после того, как ее значение будет использовано в контексте (Таблица 3).

Пример.

```
1  public class MyClass {
2      public static void main(String args[]) {
3          int i = 1, n, m;
4          n = i++;
5          m = ++i;
6          System.out.printf("n = %d \tm = %d", n, m);
7      }
8  }
```

Результат работы программы:

```
n = 1   m = 3
```

Переменной `i` будет присвоена единица, затем в следующей строке после операции `i++` в переменной `i` будет находится значение `1`, это значение будет присвоено переменной `n` и значение `i` станет равным `2`.

Далее после операции `++i` значение переменной `i` сразу измениться на `1` и станет равным `3`. После этого оно будет присвоено переменной `m`.

Типичной ошибкой является попытка использовать в операции инкремента или декремента операнд, в виде выражения:

```
++(x + 1); //ошибка
```

Замечание.

Операции инкремента и декремента могут применяться и к переменным типа `float`, `double`.

Бинарные операции

Бинарные операции имею формат:

```
операнд1 О операнд2
```

где `О` – знак операции.

Арифметические

В таблице ниже (Таблица 4) можно ознакомиться со списком бинарных арифметических операций.

Таблица 4 – Список бинарных арифметических операций

Операции	Назначение	Группы операций
+	Бинарный плюс. Сложение арифметических операндов	Аддитивные операции
-	Бинарный минус. вычитание арифметических операндов	
*	Умножение двух операндов арифметического типа	Мультипликативные операции
/	Деление операндов арифметического типа (если операнды целочисленные, то дробная часть результата отбрасывается)	
%	Получение остатка от деления арифметических операндов	

Замечание.

В Java операция `%` может использоваться и для вещественных переменных (типа `float` или `double`).

Следует отметить, что числа с плавающей точкой базовых типов не подходят для финансовых и других вычислений, где ошибки при округлении могут быть критичными.

Пример.

```
1 public class MyClass {
2     public static void main(String args[]) {
3       double d = 2.0 - 1.1;
4       System.out.println(d);
5     }
6 }
```

Результат работы программы:

```
0.8999999999999999
```

В данном случае переменная `d` будет равна не `0.9`, как можно было бы изначально предположить, а `0.8999999999999999`. Подобные ошибки точности возникают из-за проблем Java-машины с преобразованием двоичного представления чисел с плавающей точкой в десятичный формат.

Поэтому в таких случаях обычно применяется класс `BigDecimal`, который уже упоминался ранее и будет детально рассмотрен позже. Он позволяет обойти подобные ситуации.

Замечание.

Арифметическая бинарная операция «+» может примениться не только к базовым арифметическим типам, но и к строкам. В случае сложения строк эта операция называется «конкатенация» и обозначает объединение двух строк в одну строку.

Сравнения

В таблице ниже (Таблица 5) можно ознакомиться со списком операций сравнения.

Таблица 5 – Список операций сравнения

Знак	Название
<	Меньше, чем
>	Больше, чем
<=	Меньше или равно
>=	Больше или равно
==	Равно (сравнение)
!=	Не равно

Операции сравнения в результате дают значение типа `boolean`, то есть `true` или `false`.

Пример.

```
1  public class MyClass {
2      public static void main(String args[]) {
3          //переменные целого типа
4          int x1 = 5, x2 = 5, x3 = 3, x4 = 7;
5          //переменные булевского типа
6          boolean isEqual, isNonEqual, isGreater;
7          //возможные выражения
8          isEqual = x1 == x2;      // isEqual = true
9          isNonEqual = x1 != x2;   // isNonEqual = false
10         isGreater = x1 > x3;     // isGreater = true
```

```
11          System.out.println("isEqual = " + isEqual +
12          "\tisNonEqual = " + isNonEqual +
13          "\tisGreater = " + isGreater);
14      }
15  }
```

Результат работы программы:

```
isEqual = true  isNonEqual = false  isGreater = true
```

Логические бинарные операции

Основные логические операции (в программировании и математике) можно применять к логическим аргументам (операндам), а также составлять более сложные выражения, подобно арифметическим действиям над числами.

Пример.

```
//допустимое в Java логическое выражение
(a | b) | (c < 100) & !(true) ^ (q == 5)
```

В таблице ниже (Таблица 6) можно ознакомиться со списком логических бинарных операций.

Таблица 6 – Список логических бинарных операций

Операция	Назначение
`&`	Логическое `И` или конъюнкция. Возвращает `true` если оба операнда равны `true`. В другом случае результат `false`.
`|`	Логическое `ИЛИ` или дизъюнкция. Возвращает `true` если хотя бы один операнд равен `true`. В другом случае результат `false`.
`^`	Логическое исключающее `ИЛИ`. Возвращает `true`, если один и только один из операндов равен `true`. Возвращает `false`, если оба операнда равны `true` или `false`. По сути, возвращает `true`, если операнды — разные.
`&&`	Условное `И` (сокращенное логическое `И`). То же самое, что и `&`, но если операнд, находящийся слева от `&` является `false`, данная операция возвращает `false` без проверки второго операнда.
`||`	Условное `ИЛИ` (сокращенное логическое `ИЛИ`). То же самое, что и `|`, но, если операнд слева является `true`, то операция возвращает `true` без проверки второго операнда.

Побитовые операции

В Java также существуют побитовые операции. Однако следует помнить, что, в частности, логические операции &, | и ^ применяемые к целым числам работают как побитовые для двоичных представлений операндов.

В этом случае (когда операнды являются целыми) эти операции называются поразрядными (или побитовыми) операциями и работают совершенно аналогично соответствующим побитовым операциям в C++.

Кроме того, в Java существуют также операции сдвига. В таблице ниже (Таблица 7) можно ознакомиться со списком бинарных побитовых операций.

Таблица 7 – Список побитовых операций

Операции	Назначение	Группы операций
<<	Сдвиг влево битового представления значения левого целочисленного операнда на количество разрядов, равное значению правого целочисленного операнда	Операции сдвига
>>	Сдвиг вправо битового представления значения левого целочисленного операнда на количество разрядов, равное значению правого целочисленного операнда	
>>>	Беззнаковый сдвиг вправо	
&	Поразрядная конъюнкция (И) битовых представлений значений целочисленных операндов	Поразрядные операции
|	Поразрядная дизъюнкция (ИЛИ) битовых представлений значений целочисленных операндов	
^	Поразрядное исключающее или битовых представлений значений целочисленных операндов	

Замечание.

Операция беззнакового сдвига вправо `a>>>b` *сдвигает отрицательное число* `a` *вправо на* `b` *разрядов вместе с единичкой самом левом бите, т.е. предполагается, что знак это двоичное значение. Например, результат выражения* `-8>>>2` *будет равен* `1073741822`.

Операции присваивания и составного присваивания

В таблице ниже (Таблица 8) можно ознакомиться со списком операций присваивания.

Таблица 8 – Список операций присваивания

Знак	Назначение
=	Присваивание, присвоить значение правого операнда левому
+=	Выполнить соответствующую операцию с левым операндом и присвоить результат левому операнду
-=	
*=	
/=	
%=	
|=	
&=	
^=	
<<=	
>>=	
>>>=	

Операция присваивания (=) рассматривается как выражение, имеющее значение правого операнда после присваивания. Присваивание может включать несколько операций присваивания, изменяя значения нескольких операндов.

Пример:
```
a = b = c; //эквивалентно b = c; a = b;
```

Замечание. Недопустимыми являются: присваивание константе или присваивание выражению.

Пример:
```
2 = a + b; //ошибка
(a + 1) = 2 + b; //ошибка
```

Тернарная (условная) операция

В отличие от унарных и бинарных операций в тернарной условной операции используется три операнда:

```
Выражение1 ? Выражение2 : Выражение3;
```

Первым вычисляется значение `Выражения1`. Если оно истинно, то вычисляется значение `Выражения2`, которое становится результатом. Если при вычислении `Выражения1` получится `false`, то в качестве результата берется значение `Выражения3`.

Приоритет операций для переменных и констант базовых типов

Последовательность вычисления значения выражения зависит от:

- расположения круглых скобок в выражении;
- приоритета выполнения операций (Таблица 9).

Таблица 9 – Приоритет операций

Номер	Операции
1.	Постфиксная форма ++ и --
2.	Унарные операции +, -, ~, !, префиксная форма ++ и --,
3.	*, /, % (мультипликативные)
4.	+, - (аддитивные)
5.	<<, >>, >>> (побитового сдвига)
6.	<, >, <=, >=, instanceof
7.	==, !=
8.	&
9.	^
10.	|
11.	&&
12.	||
13.	? : (тернарная операция)
14.	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>= (операции присваивания)

Замечание!

Операция `instanceof` *позволяет выяснить, является ли переданный в качестве параметра объект объектом определенного класса, экземпляром его подкласса или экземпляром класса, реализующего определенный интерфейс. Будет рассмотрена позже.*

Требования к синтаксису операций и инструкций

- все операнды и знаки операций разделяются пробелами;
- каждая инструкция пишется в отдельной строке;
- объявления переменных одного типа следует писать в разных строках для удобства восприятия и при компиляции определения индукции, содержащей ошибку;
- желательно, чтобы инструкция не превосходила ширины экрана;
- если длинное выражение разбивается на несколько строк, то табуляцией строки выравниваются слева до знака присваивания или открывающей скобки метода.

Преобразование базовых типов данных

Каждый базовый тип данных занимает определенное количество байт памяти. Это накладывает ограничение на операции, в которые вовлечены различные типы данных. Следующий код вызовет ошибку компиляции:

```
int a = 4;
byte b = a; // ! Ошибка
```

Хотя и тип `byte`, и тип `int` представляют целые числа. Более того, значение переменной `a`, которое присваивается переменной типа `byte`, вполне укладывается в диапазон значений для типа `byte` (от -128 до 127). Поскольку в данном случае в рассматриваемом коде осуществляется попытка присвоить некоторые данное, которые занимает четыре байта, переменной, которая занимает всего один байт.

Тем не менее в программе может потребоваться, чтобы подобное преобразование было выполнено. В этом случае необходимо использовать операцию явного преобразования типов (операция `()`):

Явное приведение типов

В любом выражении приведение типов может быть осуществлено явно, в Java для этого достаточно перед выражением поставить в скобках идентификатор соответствующего типа:

```
(тип) выражение
```

Обычно это сужающие преобразования (narrowing) от типа с большей разрядностью к типу с меньшей разрядностью:

Пример

```
long a = 4;
int b = (int) a;
```

Потеря данных при явном преобразовании

Очевидно, что при выполнении явных преобразований можно столкнуться с потерей данных.

Пример.

```
1 public class MyClass {
2     public static void main(String[] args) {
3         int a = 258;
4         byte b = (byte) a;
5         System.out.println(b);
6     }
7 }
```

Результат работы программы:

```
2
```

Объясним почему результатом будет число `2`. В данном случае число `258` вне диапазона для типа `byte` (от `-128` до `127`), поэтому произойдет усечение значения.

Переменная `a`, которая равна `258`, в двоичном системе будет расположено в 4 байтах следующим образом:

`00000000 00000000 00000001 00000010` .

Значения типа `byte` занимают в памяти только `8` бит. Поэтому двоичное представление числа `int` усекается до `8` правых разрядов справа (т.к. нумерация бит в двоичном представлении начинается справа), то есть до значения:

`00000010` ,

что в десятичной системе дает число `2`.

Усечение рациональных чисел до целых

При преобразовании значений с плавающей точкой к целочисленным значениям, происходит усечение дробной части.

Пример.

```
double a = 56.9898;
int b = (int) a;
```

Здесь значение переменной `b` будет равно `56`, несмотря на то, что число `57` было бы ближе к `56.9898`. Чтобы избежать подобных случаев, следует применять функцию округления, которая есть в математической библиотеке Java (будет рассмотрена позже).

Неявное преобразование типов

Некоторые виды преобразований выполняются неявно, т.е. автоматически.

Автоматические преобразования

Стрелками показано (Рисунок 1), какие преобразования типов могут выполняться автоматически.

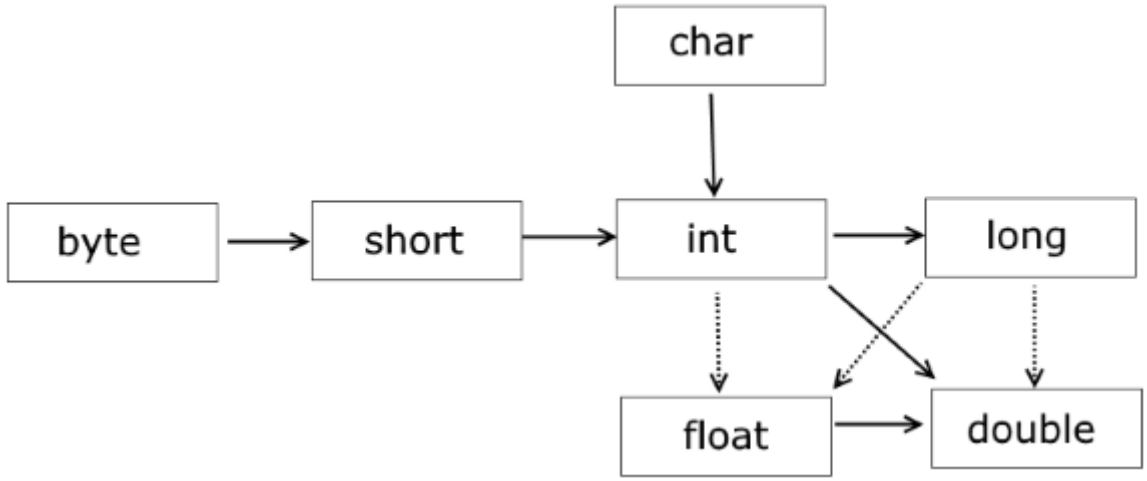

Рисунок 1 – Автоматические преобразования типов в языке Java

Автоматически без каких-либо проблем производятся расширяющие преобразования (`widening`), т.к. они расширяют представление объекта в памяти (увеличивают размер памяти, выделенной под хранение значения, Рисунок 1).

Пример.

```
byte b = 7;
int d = b;  // преобразование от byte к int
```

В данном случае значение типа `byte`, которое занимает в памяти `1` байт, расширяется до типа `int`, которое занимает `4` байта.

Автоматические преобразования с возможной потерей точности

Несмотря на то, что формально следующие автоматические преобразования:

```
int -> float, long -> float и long -> double,
```

также являются расширяющими. Однако в связи с особенностями хранения чисел с плавающей точностью в памяти можно столкнуться с потерей информации.

Пример.

```
1 public class MyClass {
2     public static void main(String[] args) {
3         int a = 2147483647;
4         float b = a;
5         System.out.println(b);
6     }
7 }
```

Результат работы программы:

```
2.14748365E9
```

Преобразование типов при операциях

Преобразование типа при присваивании

Преобразование типов при присвании выглядит давольно понятным. Если слева стоит тип допускающий автоматическое (расширяющее) преобразование типа правого операнда, то тип правого операнда, который присваиваится, приводится к типу операнда стоящего слева (которому присваивается значение).

Если слева стоит тип сужающий размеры хранилища для значений, то для преобразования типов необходимо использовать явное преобразование.

Преобразования при остальных бинарных операциях с базовыми типами данных

Нередки ситуации, когда приходится применять различные операции, например, сложение и произведение, для значений разных типов. Здесь также действуют некоторые правила:

- преобразование типов в любой бинарной операции, отличной от присваивания осуществляется к одному из типов операндов;
- тип, к которому происходит преобразование в бинарной операции, является лидирующим по диапазону хранимых значений.

Большие числа BigInteger и BigDecimal

В Java были созданы два специальных класса — `BigInteger` (для целых чисел) и `BigDecimal` (для чисел с плавающей точкой). Если значения, хранимые в переменных базовых типов (даже таких типов как `long` и `double`), как и во всех языках программирования ограничены размерами памяти, выделяемой под данные переменные, то размер чисел `BigInteger` и `BigDecimal` практически ничем не ограничен.

Они применяются прежде всего, для вычислений с крайне высокими требованиями к точности. Есть, к примеру, программы, в которых от точности вычислений может зависеть человеческая жизнь (ПО для самолетов и ракет или для медицинского оборудования). Поэтому, если даже 150-й разряд после запятой играет важную роль, `BigDecimal` — лучший выбор.

Кроме того, довольно часто эти объекты применяются в мире финансов, где точность вычислений вплоть до самых мелких значений тоже крайне важна.

Для применения в программе классов `BigInteger` и `BigDecimal` следует `import`-ировать два одноименных класса из пакета `java.math`.

Пример.

```
//для подключения класса BigInteger
import java.math.BigInteger;

//для подключения класса BigDecimal
import java.math.BigDecimal;
```

или

```
//для подключения всех классов пакета java.math
import java.math.*
```

Объекты этих классов создаются следующим образом:

```
1  import java.math.*;
2
3  public class MyClass {
4      public static void main(String[] args) {
5
6          BigInteger i = new
7                  BigInteger("11111111111111111111111111111111111111");
8          System.out.println(i);
9
10         BigDecimal x = new
11             BigDecimal("123.44444444444444444444444444444444444444");
12         System.out.println(x);
13     }
14 }
```

Результат работы программы:

```
11111111111111111111111111111111111111
123.44444444444444444444444444444444444444
```

Передача строки в качестве параметра — только один из возможных конструкторов. Оба класса умеют автоматически извлекать из переданных строк числовые значения. В приведенном примере используются строки, потому что числа *могут* превышать максимальные значения `long` и `double`. В этом смысле применение строк – это единственно возможный способ указать компилятору, какое именно огромное число необходимо разработчику.

Замечание.

К сожалению, конструкторов, непосредственно переводящих базовые числовые типы в `BigInteger` или `BigDecimal` не

существует. Однако для этих целей в обоих классах существует статический метод `valueOf()`. *Фактически он осуществляет приведение значений базовых арифметических типов к объектам, рассматриваемых классов.*

Операции с объектами классов

Классы больших чисел не используют в своей работе операции `+`, `-`, `*`, `/`, а предоставляют вместо этого набор методов. Перечислим основные из них в случае класса `BigInteger` для осуществления арифметических операций:

- `BigInteger` **`add`**`(BigInteger other)` - возвращает сумму двух чисел;
- `BigInteger` **`subtract`**`(BigInteger other)` - возвращает разность двух чисел;
- `BigInteger` **`multiply`**`(BigInteger other)` - возвращает произведение двух чисел;
- `BigInteger` **`divide`**`(BigInteger other)` - возвращает частное двух чисел.

Пример.

```
1  import java.math.BigInteger;
2
3  public class MyClass {
4      public static void main(String[] args) {
5          BigInteger i = new
6              BigInteger("11111111111111111111111111111111111111111");
7          System.out.println(i);
8          System.out.println(i.add(BigInteger.valueOf(33333333)));
9          BigInteger n = i.add(BigInteger.valueOf(33));
10         System.out.println(n);
11     }
12 }
```

Результаты работы программы:

```
11111111111111111111111111111111111111111
11111111111111111111111111111111111144444444
11111111111111111111111111111111111111144
```

Основные методы класса `BigDecimal` для выполнения основных арифметических операций по своим названиям полностью совпадают с уже перечисленными методами класса `BigInteger`.

Выбор максимума/минимума из двух объектов

Для сравнения указанных объектов используются методы `min()` и `max()` (определены в обоих классах), позволяющее найти минимальное и максимальное значение из двух переданных больших чисел.

Замечания.

- *методы возвращают не логическое значение `false` или `true`, а именно искомую величину, что упрощает их алгоритмическое применение в следующем программном коде;*
- *методы не являются статическими, а следовательно, к ним обратиться можно только через имя первого операнда участвующего в сравнении.*

Пример.

```
1  import java.math.BigInteger;
2  public class MyClass {
3      public static void main(String[] args) {
4          BigInteger i = new
5              BigInteger("11111111111111111111111111111111111111");
6          BigInteger j = new
7              BigInteger("22222222222222222222222222222222222222");
8          System.out.println(i.max(j));
9      }
10 }
```

Результат работы программы:

```
22222222222222222222222222222222222222
```

Управление округлением BigDecimal

Разработчик можешь установить количество цифр после запятой для объекта класса `BigDecimal` при помощи метода `setScale()`.

Масштабом объекта класса `BigDecimal` называется количество цифр справа от десятичной точки (т.е. количество позиций в дробной части).

Например, необходимо установить для числа 111.5555555555 точность в три цифры после запятой. При этом нельзя передать только одно число 3 в качестве аргумента в метод `setScale()` и таким образом решить поставленную задачу. Кроме числа 3 нам необходимо передать в качестве параметра еще и режим округления (rounding mode).

Всего у `BigDecimal` существует 8 режимов округления:

- `ROUND_CEILING` — округление в большую сторону;
- `ROUND_DOWN` — отбрасывание разряда;
- `ROUND_FLOOR` — округление в меньшую сторону;
- `ROUND_HALF_UP` — округление в большую сторону, если число после запятой >= .5;
- `ROUND_HALF_DOWN` — округление в большую сторону, если число после запятой > .5;
- `ROUND_HALF_EVEN` — округление будет зависеть от цифры слева от запятой. Если цифра слева будет четной, то округление будет произведено вниз, в меньшую сторону. Если цифра слева от запятой нечетная, то округление будет произведено вверх;
- `ROUND_UNNECCESSARY` — используется в тех случаях, когда в какой-то метод нужно передать режим округления, но число в округлении не нуждается. Если попробовать произвести округление числа при выставленном режиме `ROUND_UNNECCESSARY` — выброшено исключение `ArithmeticException`.

Сравнение больших чисел

Для корректного сравнения двух объектов `BigDecimal` следует использовать метод `compareTo()`. Метод сравнивает объект `BigDecimal` с другим объектом `BigDecimal`. Два объекта `BigDecimal`, которые равны по значению, но имеют разный масштаб (например, `2,0` и `2,00`), считаются равными в этом методе. Этот метод предоставляется вместо отдельных методов для каждого из шести логических операций сравнения (<, ==, >, >=, !=, <=).

Результат сравнения двух объектов `x` и `y`: `-1`, `0` или `1`, если `x` численно меньше, равен или больше `y`.

Замечание.
Данный метод также определен и для класса `BigInteger`.

Пример.

```
1  import java.math.BigDecimal;
2
3  public class MyClass {
4      public static void main(String[] args) {
5          BigDecimal x = new BigDecimal("1.5");
6          BigDecimal y = new BigDecimal("2.50");
7          System.out.println(x.compareTo(y));
8      }
9  }
```

Результат работы программы:

```
-1
```

Замечание.

В рассматриваемых классах определены также одноименные методы `equals()`, не рекомендуется применять соответствующий метод в случае сравнения объектов класса `BigDecimal`.

Преобразование объектов классов `BigDecimal` и `BigInteger` к базовым типам

Необходимо отдельно остановиться на возможности преобразовании объектов классов `BigDecimal` и `BigInteger` в базовые типы языка Java:

- `double` **`doubleValue()`** - преобразует объект `BigDecimal` в `double`;
- `float` **`floatValue()`** - преобразует объект `BigDecimal` в `float`;
- `int` **`intValue()`** - конвертирует объект `BigInteger` в объект `int`;
- `long` **`longValue()`** - преобразует объект `BigInteger` в `long`.

К вопросу о возможной потере или искажению информации при преобразованиях к примитивным типам:

- преобразование выполняемое методами `doubleValue()` и `floatValue()` аналогично уменьшающему разрядность примитивному преобразованию из `double` во `float`, если объект `BigDecimal` имеет слишком большую величину для

представления в виде числа с плавающей запятой, оно будет преобразовано в константы `Float.NEGATIVE_INFINITY` или `Float.POSITIVE_INFINITY` по мере необходимости. Даже если возвращаемое значение является конечным, это преобразование может привести к потере информации о точности значения `BigDecimal`.

- преобразование выполняемое методами `intValue()`, `longValue()` аналогично уменьшающему разрядность примитивному преобразованию из `long` в `int`: если это `BigInteger` слишком велико, чтобы поместиться в `int`, возвращаются только младшие 32 бита. Очевидно, что это преобразование может привести к потере информации об общей величине значения `BigInteger`, а также возврату результата с противоположным знаком.

Преобразование объектов классов `BigDecimal` и `BigInteger` друг в друга

Для преобразования объектов `BigDecimal` в `BigInteger` используется методы:

- `toBigInteger()` – преобразует объект `BigDecimal` в объект `BigInteger` с отсечение дробной части;
- `toBigIntegerExact()` - преобразует объект `BigDecimal` в объект BigInteger, проверяя потерянную информацию. Исключение `ArithmeticException` выдается, если объект `BigDecimal` имеет ненулевую дробную часть.

Обратное преобразование следует осуществлять с использованием конструктора класса `BigDecimal`. Он, очевидно допускает использование объекта `BigInteger` в качестве параметра.

Полный список конструкторов и методов классов `BigDecimal` и `BigInteger` читатель сможет найти в интернете (для `BigDecimal` – URL: https://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html, для `BigInteger` – URL: https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html , дата доступа: 07.06.2022).

Математические вычисления и класс Math

Для выполнения различных математических операций в Java в пакете `java.lang` определен класс `Math`.

Методы класса Math

Рассмотрим основные методы:

- `abs(double value)` - возвращает абсолютное значение для аргумента `value`;
- `acos(double value)` - возвращает арккосинус `value`. Параметр `value` должен иметь значение от `-1` до `1`;
- `asin(double value)` - возвращает арксинус `value`. Параметр `value` должен иметь значение от `-1` до `1`;
- `atan(double value)` - возвращает арктангенс `value`;
- `cbrt(double value)` - возвращает кубический корень числа value;
- `ceil(double value)` - возвращает наименьшее целое число с плавающей точкой, которое не меньше `value`;
- `cos(double d)` - возвращает косинус угла `d`;
- `cosh(double d)` - возвращает гиперболический косинус угла `d`;
- `exp(double d)` - возвращает основание натурального логарифма, возведенное в степень `d`;
- `floor(double d)` - возвращает наибольшее целое число, которое не больше `d`;
- `floorDiv(int a, int b)` - возвращает целочисленный результат деления `a` на `b`;
- `log(double a)` - возвращает натуральный логарифм числа `a`;
- `log1p(double d)` - возвращает натуральный логарифм числа `(d + 1)`;
- `log10(double d)` - возвращает десятичный логарифм числа `d`;
- `max(double a, double b)` - возвращает максимальное число из `a` и `b`;
- `min(double a, double b)` - возвращает минимальное число из `a` и `b`;
- `pow(double a, double b)` - возвращает число `a`, возведенное в степень `b`;
- `random()` - возвращает случайное число от `0.0` до `1.0`;

- `rint(double value)` - возвращает число `double`, которое представляет ближайшее к числу `value` целое число;
- `round(double d)` - возвращает число `d`, округленное до ближайшего целого числа;
- `scalb(double value, int factor)` - возвращает произведение числа `value` на `2` в степени `factor`;
- `signum(double value)` - возвращает число `1`, если число `value` положительное, и `-1`, если значение `value` отрицательное. Если `value` равно `0`, то возвращает `0`;
- `sin(double value)` - возвращает синус угла `value`;
- `sinh(double value)` - возвращает гиперболический синус угла `value`;
- `sqrt(double value)` - возвращает квадратный корень числа `value`;
- `tan(double value)` - возвращает тангенс угла `value`;
- `tanh(double value)` - возвращает гиперболический тангенс угла `value`;
- `toDegrees(double value)` - переводит радианы в градусы;
- `toRadians(double value)` - переводит градусы в радианы.

Напомним, что для работы с любым из перечисленных методов нет необходимости подключать подключить класс `Math` с помощью директивы `import`, т.к. пакет `java.lang` подключен автоматически. Поскольку все перечисленные методы являются статическими, то для их вызова не требуется использовать имя какого-либо специально созданного объекта, т.е. формат вызова любого из этих методов будет иметь вид:

```
Math.имяМетода(списокФактическихАргументов)
```

Пример.

```
1  public class Program {
2    public static void main(String[] args) {
3      System.out.println(Math.abs(-21.8));
4
5      System.out.println(Math.sin(Math.toRadians(30)));
6
7      System.out.println(Math.pow(4,2));
8
9      double x = Math.sqrt(16);
10     System.out.println(x);
11
12     System.out.println(Math.cbrt(8));
```

```
13
14      System.out.println(Math.round(1.5));
15      System.out.println(Math.floor(1.6));
16      System.out.println(Math.ceil(1.3));
17    }
18  }
```

Результат выполнения программы:

```
21.8
0.49999999999999994
16.0
4.0
2.0
2
1.0
2.0
```

Генерация случайных чисел

Для генерации случайных чисел класс `Math` предоставляет статический метод `random()`. Данный метод генерирует случайное положительное вещественное (`double`) число в промежутке от `0.0` до `1.0`.

Заголовок метода имеет следующий вид:

```
public static double random()
```

Пример.

```
1  public class Program {
2  public static void main(String[] args) {
3          System.out.println(Math.random());
4
5          double z = Math.random();
6          System.out.println(z);
7      }
8  }
```

Результат выполнения программы:

```
0.5870242686483423
0.17039107081062055
```

С помощью небольших манипуляций, можно использовать метод `random()` класса `Math` для получения целочисленных случайных чисел лежащих в определенном диапазоне, например, в промежутке от `min`

(включительно) до `max` (включительно). Для этого достаточно использовать строку:

```
Math.random() * (max - min) + min;
```

Константы класса Math

Также класс `Math` определяет две константы: `Math.E` и `Math.PI`. Например, вычислим площадь круга:

```
1  public class Program {
2      public static void main(String[] args) {
3          int radius = 10;
4          double area = Math.PI * Math.pow(radius, 2);
5          System.out.printf("Площадь круга с радиусом %d = %f",
6                                  radius, area);
7      }
8  }
```

Результат выполнения программы:

```
Площадь круга с радиусом 10 = 314.159265
```

Работа с датами. Класс `LocalDate`

Класс `LocalDate` из пакета `java.time` предназначен для работы с датами. Функционал этого класса позволяет создавать даты и изменять их, добавляя и отнимая необходимое количество дней/месяцев/лет.

Основные методы `LocalDate` (все методы статические):

- `LocalDate now()` - возвращает объект, который представляет текущую дату;
- `LocalDate of(int year, int month, int day)` - возвращает объект, который представляет дату с определенными годом, месяцем и днем;
- `int getYear()` - возвращает год даты;
- `int getMonthValue()` - возвращает месяц;
- `int getDayOfMonth()` - возвращает день месяца (значение от 1 до 31);

- `int getDayOfYear()` - возвращает номер дня года (значение от 1 до 365);
- `LocalDate plusDays(int n)` - добавляет к дате некоторое количество дней;
- `LocalDate plusWeeks(int n)` - добавляет к дате некоторое количество недель;
- `LocalDate plusMonths(int n)` - добавляет к дате некоторое количество месяцев;
- `LocalDate plusYears(int n)` - добавляет к дате некоторое количество лет;
- `LocalDate minusDays(int n)` - отнимает от даты некоторое количество дней;
- `LocalDate minusMonths(int n)` - отнимает от даты некоторое количество месяцев;
- `LocalDate minusWeeks(int n)` - отнимает от даты некоторое количество недель;
- `LocalDate minusYears(int n)` - отнимает от даты некоторое количество лет.

Пример работы с текущей датой

```
1  import java.time.LocalDate;
2
3  public class Program {
4      public static void main(String[] args) {
5          // получаем текущую дату
6          LocalDate date = LocalDate.now();
7          // отдельно по категориям
8          int year = date.getYear();
9          int month = date.getMonthValue();
10         int dayOfMonth = date.getDayOfMonth();
11         // вывод реузльатато на экран
12         System.out.println(date);
13         System.out.printf("%d.%d.%d",
14                           dayOfMonth, month, year);
15     }
16 }
```

Результат работы программы:

```
2022-06-07
7.6.2022
```

Пример работы с произвольной датой

```
1  import java.time. LocalDate;
2
3  public class Program {
4      public static void main(String[] args) {
5          //создание объекта LocalDate из произв. даты
6          LocalDate date = LocalDate.of(1914, 12, 31);
7          System.out.println(date);
8          //слоложение дат
9          date = date.plusYears(4);
10         date = date.plusMonths(3);
11         date = date.plusDays(14);
12         System.out.println(date);
13         //вычитание дат
14         date = date.minusMonths(10);
15         date = date.minusDays(3);
16         System.out.println(date);
17     }
18 }
```

Результат работы программы:

```
1914-12-31
1919-04-14
1918-06-11
```

Область видимости переменных

Понятие «видимость» переменной (объекта) подразумевает тот факт, что к рассматриваемой переменной (объекту) можно обратиться из какой-либо части программного кода.

Областью действия (видимости) объекта (данного) называется та часть программы, в которой можно пользоваться этим объектом:

- переменная, объявленная в методе, существует/видна с начала объявления до конца метода;
- переменная, объявленная в блоке кода, существует/видна до конца этого блока кода;
- переменные – формальные аргументы метода - существует/видна везде внутри метода;

- свойства объекта существуют все время жизни содержащего их объекта, их видимость дополнительно регулируется специальными модификаторами доступа;
- статические свойства классов существуют все время работы программы, их видимость также определяется модификаторами доступа.

Замечание.

Модификаторы доступа `public`, `private` *будут рассмотрены позже, хотя они имеют такой же смысл, как и в C++.*

Пример.

```
1  public class MyClass{
2     public static void main (String args[]){
3          {
4              int a = 3, b = 0;
5              a = b + 2;
6              b++;
7              System.out.print("В блоке a = " + a +
8                                  ", b =" + b + "\n");
9          }
10         // за пределами блока переменные a и b не видны -
11         // ошибка компиляции
12         System.out.print("За пределами блока a = " + a +
13                             ", b =" + b + "\n");
14     }
15 }
```

Результат выполнения программы:

```
/MyClass.java:12: error: cannot find symbol
        System.out.print("За пределами блока a = " + a +
                                                    ^
  symbol:   variable a
  location: class MyClass
/MyClass.java:13: error: cannot find symbol
                           ", b =" + b + "\n");
                                  ^
  symbol:   variable b
  location: class MyClass
2 errors
```

Литература

1. Блинов, И.Н. Java from EPAM / И.Н. Блинов, В.С. Романчик. - Минск: Четыре четверти, 2020. - 560 с.
2. Руководство по языку программирования Java/ METANIT.COM - [Электронный ресурс], URL: https://metanit.com/java/tutorial/ (дата обращения: 08.06.22)
3. Самоучитель по Java с нуля/ Vertex Academy - [Электронный ресурс], URL: https://vertex-academy.com/tutorials/ru/samouchitel-po-java-s-nulya/ (дата обращения: 08.06.22)
4. Java — Самоучитель / ProgLang - [Электронный ресурс], URL: http://proglang.su/java (дата обращения: 08.06.22)
5. 30 лучших онлайн курсов Java программирования / HashNets - [Электронный ресурс], URL: https://hashnets.com/30-лучших-онлайн-курсов-на-платформе-java/ (дата обращения: 08.06.22)
6. Учебники по Java/ Oracle - [Электронный ресурс], URL: https://docs.oracle.com/javase/tutorial/index.html (дата обращения: 08.06.22)
7. Соглашения о коде для языка программирования Java/ Oracle - [Электронный ресурс], URL: https://www.oracle.com/java/technologies/javase/codeconventions-introduction.html (дата обращения: 08.06.22)

Учебное издание

Кравчук Александр Степанович
Кравчук Анжелика Ивановна
Кремень Елена Васильевна

ЯЗЫК JAVA

ОСНОВЫ СИНТАКСИСА

Учебные материалы
для студентов специальности 1-31 03 08
«Математика и информационные технологии
(по направлениям)»

В авторской редакции

Ответственный за выпуск *Е. В. Кремень*

Подписано в печать 25.11.2022. Формат 60×84/16. Бумага офсетная.
Усл. печ. л. 3,49. Уч.- изд. л. 3,25. Тираж 50 экз. Заказ

Белорусский государственный университет.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/270 от 03.04.2014.
Пр. Независимости 4, 220030, Минск.

Отпечатано с оригинал-макета заказчика
на копировально-множительной технике
механико-математического факультета
Белорусского государственного университета.
Пр. Независимости 4, 220030, Минск.