

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационной безопасности

Кафедра инфокоммуникационных технологий

**ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ
ИНФОКОММУНИКАЦИОННЫХ СИСТЕМ
Часть 2**

**Практическая работа 1
Введение в Java. Основы языка**



Минск 2023

Содержание

Практическая работа 1	
Введение в Java. Основы языка.....	3
Что такое JDK? Введение в средства разработки Java	3
Установка и настройка JDK.....	3
Структура Java-программы	4
Простая Java программа.....	4
Команда <code>jar</code>	6
Запуск <code>.jar</code> -файла	6
IntelliJ IDEA и JDK.....	7
Лексические основы языка.....	7
Примитивные типы данных	9
Преобразование типов	10
Арифметические операции	11
Консольный ввод с помощью класса <code>java.util.Scanner</code>	11
Операторы	12
Статический импорт.....	12
Класс <code>Math</code>	13
Псевдослучайные числа	14
Генерация случайных чисел	14
Массивы в Java.....	15

Практическая работа 1

Введение в Java. Основы языка

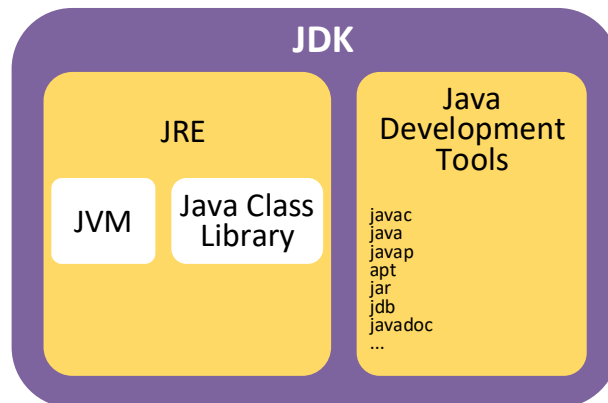
Что такое JDK? Введение в средства разработки Java

Java Development Kit (JDK) является одним из трех основных технологий, используемых в программировании на языке Java. К ним также относятся JVM (Java Virtual Machine) и JRE (Java Runtime Environment). Важно их различать, а также понимать, как они связаны:

- JVM отвечает за исполнение Java-программ;
- JRE создает и запускает JVM;
- JDK позволяет разработчикам создавать программы, которые могут выполняться и запускаться посредством JVM и JRE.

Различие JDK и JRE заключается в том, что JDK представляет собой пакет инструментов для разработки программного обеспечения, тогда как JRE представляет собой пакет инструментов для запуска Java-кода.

JRE может использоваться, как отдельный компонент для простого запуска Java-программ, либо быть частью JDK. JDK требуется JRE, потому что запуск программ является неотъемлемой частью их разработки.



Техническое и обобщенное определение JDK:

- Техническое определение: JDK – это реализация спецификации платформы Java, включающая в себя компилятор и библиотеки классов.
- Обобщенное определение: JDK – это программный пакет, который вы загружаете для создания Java-приложений.

JDK и компилятор Java

В дополнение к JRE, который является средой, используемой для запуска Java-приложений, каждый JDK содержит компилятор Java. Компилятор – это программа, способная принимать исходные файлы с расширением `.java`, которые являются обычным текстом, и превращать их в исполняемые файлы с расширением `.class`.

Установка и настройка JDK

При загрузке JDK необходимо выбрать версию Java. Java поддерживает обратную совместимость, поэтому можно просто загрузить последнюю версию <https://www.oracle.com/java/technologies/downloads/>.

JDK в командной строке

Установка JDK и JRE добавляет команду `java` в командную строку. Проверить это можно выполнив в командной строке команду `java -version`, которая должна вывести в консоль, установленную версию Java.

Команда `javac`

Утилита `javac` находится в директории `/bin`, но автоматически не добавляется в системную переменную `PATH` (представляет собой набор путей до каталогов, в которых расположены исполняемые файлы) во время установки. Можно установить `javac` самостоятельно или установить IDE (интегрированная среда разработки), которая содержит эту команду.

Структура Java-программы

Для большинства языков программирования файл, содержащий исходный код программы, может иметь произвольное имя. Однако для Java действуют другие правила. Работая с Java, необходимо знать, что имя исходного файла очень важно. В примере ниже, файл содержащий исходный код, имеет имя `Intro.java`.

В Java исходный файл называется модулем компиляции. Это текстовый файл, содержащий определения одного или нескольких классов. Компилятор Java требует, чтобы исходный файл имел расширение `.java`. Обратив внимание на код программы, можно увидеть, что класс имеет имя `Intro.java`. Это несовпадение. В Java-программах весь код находится в составе классов. По соглашениям имя класса должно совпадать с именем файла, содержащего текст программы. Нетрудно убедиться, что имя файла соответствует имени класса. В языке Java имена и другие идентификаторы зависят от регистра символов. Соглашение о соответствии имен классов именам файлов может на первый взгляд показаться надуманным, однако оно упрощает организацию и сопровождение программ.

Особенности Java-программы:

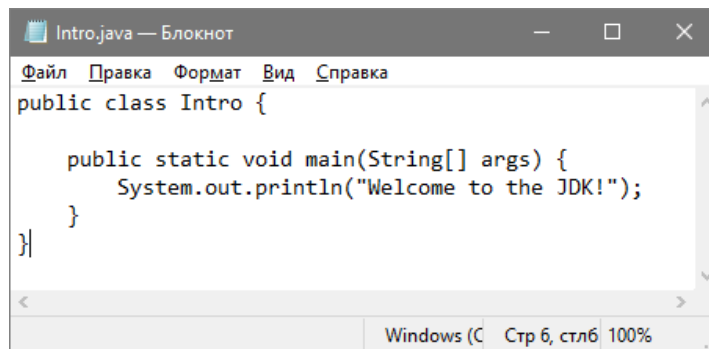
1. Любая программа состоит из одного или нескольких классов.
2. Начало класса отмечается служебным словом `class`, за которым следует имя класса, выбираемое произвольно. Все, что содержится в классе, записывается в фигурных скобках и составляет тело класса.
3. Все действия производятся с помощью методов обработки информации, которые являются аналогами "функций", применяемых в других языках.
4. Методы различаются по именам. Один из методов обязательно должен называться `main()`, с него начинается выполнение программы.
5. Как и положено функции, метод всегда возвращает `return` только одно значение, тип которого обязательно указывается перед именем метода. Метод может и не возвращать никакого значения, играя роль процедуры. Тогда вместо типа возвращаемого значения записывается слово `void`.
6. После имени метода в скобках, через запятую, перечисляются аргументы или параметры метода. Для каждого аргумента указывается его тип и имя.
7. Перед типом значения, возвращаемого методом, могут быть записаны модификаторы. Модификаторы необязательны, но для метода `main()` они необходимы.

Простая Java программа

Шаг 1. Пишем код на Java

Создаем новый текстовый файл под названием `Intro.java` в каком-нибудь редакторе и сохраняем его куда-нибудь на своем компьютере.

Затем добавляем в него следующий код.



```
Intro.java — Блокнот
Файл  Правка  Формат  Вид  Справка
public class Intro {

    public static void main(String[] args) {
        System.out.println("Welcome to the JDK!");
    }
}
```

Рассмотрим первую строку кода `public class Intro {`.

Ключевое слово `public` называется модификатором доступа. Модификатор доступа определяет правила обращения к членам класса из других частей программы. Если член класса предваряется ключевым словом `public`, то к нему может производиться обращение из-за пределов класса.

Ключевое слово `class`, используется для определения нового класса. Как было сказано ранее, класс является основной языковой конструкцией Java, поддерживающей инкапсуляцию. `Intro` – это имя класса. Определение класса начинается открывающей фигурной скобкой `{` и заканчивается закрывающей фигурной скобкой `}`. Элементы, находящиеся между ними, являются членами класса.

Рассмотрим следующую строку кода `public static void main(String args[]) {`.

Строка начинается с метода `main()`. Выполнение любого Java-приложения начинается с вызова метода `main()`. В данном случае метод `main()` должен быть объявлен как `public`, поскольку при выполнении программы он вызывается из-за пределов класса. Ключевое слово `static` допускает вызов метода `main()` до создания экземпляра класса. Указывать его необходимо, поскольку метод `main()` вызывается виртуальной машиной еще до того, как будут созданы какие-либо объекты. Ключевое слово `void` лишь сообщает компилятору о том, что метод `main()` не возвращает значение.

Любая информация, которую необходимо передать данному методу, задается с помощью переменных, указанных в круглых скобках, которые следуют за именем метода. Эти переменные называются параметрами. Если для какого-либо метода параметры не предусмотрены, то после имени метода указывается пара круглых скобок. В данном случае для метода `main()` задан один параметр с именем `args`: `String args[]`. Это массив объектов типа `String` (массив – набор объектов одного типа). Объекты типа `String` хранят последовательности символов. В данном случае с помощью `args` методу передаются параметры, указанные в командной строке при запуске программы. Завершается строка символом `{`. Он означает начало тела метода `main()`. Весь код, содержащийся в составе метода, размещается между открывающей и закрывающей фигурными скобками.

Рассмотрим следующую строку кода `System.out.println("Welcome to the JDK!");`.

Она выводит на экран строку "Welcome to the JDK!", завершая ее символом перевода строки. Вывод информации осуществляет встроенный метод `println()`. В данном случае `println()` отображает переданную ему строку. Строка начинается с `System.out`. `System` – это предопределенный класс, предоставляющий доступ к системным средствам, а `out` – выходной поток, связанный с консолью. Таким образом, `System.out` – это объект, инкапсулирующий вывод на консоль. За выражением `println()` следует точка с запятой. Этим символом заканчиваются все выражения Java. В других строках точка с запятой отсутствует лишь потому, что они не являются выражениями.

Первая закрывающая фигурная скобка завершает метод `main()`, а вторая такая же скобка является признаком окончания определения класса `Intro`.

В языке Java учитывается регистр символов. Например, если вместо `main` ввести `Main` или указать `PrintLn` вместо `println`, код программы будет некорректным. Компилятор Java откомпилирует классы, не содержащие метода `main()`, но запустить их на выполнение не сможет. Таким образом, если допустить ошибку в имени `main`, компилятор скомпилирует программу, но не сообщит об ошибке. В этом случае об ошибке сообщит интерпретатор, когда не сможет найти метод `main()`.

Шаг 2. Компиляция с помощью JDK

Затем используем JDK-компилятор, чтобы превратить текстовый файл в исполняемую программу. Скомпилированный код в Java известен, как байт-код и имеет расширение `.class`.

Для компиляции программ необходимо использовать команду `javac`.

```
C:\Users\Piglet>javac Intro.java
```

Это должно привести к успешной компиляции. Если компиляция прошла удачно, то в консоли не отобразится сообщение об ее успешном завершении. В противном случае любые ошибки будут отображены в консоли.

Шаг 3. Запуск `.class`-файла

После компиляции должен появиться файл `Intro.class` в той же папке, что и `Intro.java`. Для его запуска используем команду `java`. При вводе команды расширение `.class` не указывается. Результат работы команды отобразится в командной строке.

```
C:\Users\Piglet>java Intro
Welcome to the JDK!
```

Команда `jar`

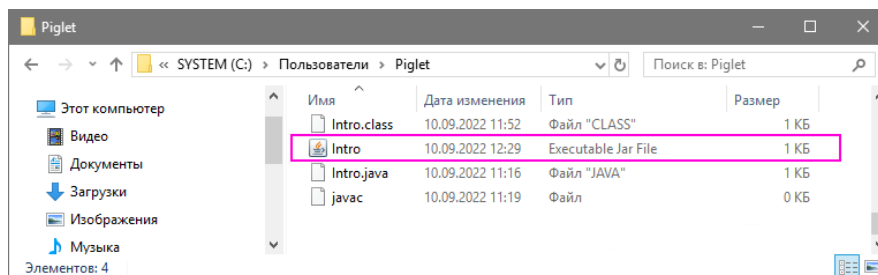
`Javac` является важной частью JDK, но каталог `/bin` содержит и другие инструменты. Наиболее важным после `javac` является инструмент `jar`.

Файл с расширением `.jar` является архивом для Java-классов. После того как компилятор создал `class`-файлы, можно объединить их в `.jar`, который сжимает и структурирует их в необходимом порядке.

Преобразуем `Intro.java` в `jar`-файл, выполним следующую команду в командной строке.

```
C:\Users\Piglet>jar --create --file Intro.jar Intro.class
```

Если все прошло хорошо, то в каталоге появился файл `intro.jar`.



Запуск `.jar`-файла

Запустить `jar`-файл можно используя следующую команду.

```
C:\Users\Piglet>java -cp Intro.jar Intro
Welcome to the JDK!
```

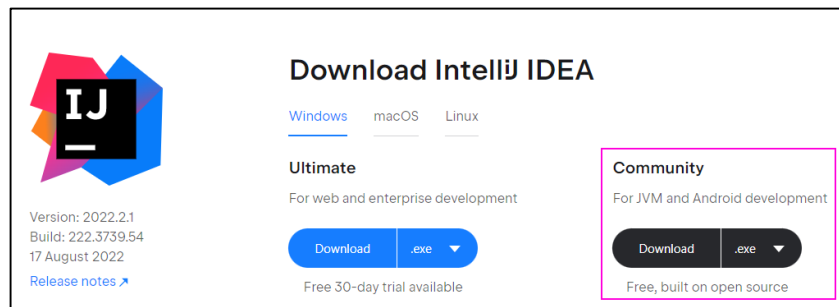
Эта команда говорит Java, что искать метод `main` надо в классе `Intro` по пути `intro.jar`. Для такой маленькой программы не имеет смысла создавать `.jar`-файл. Но такие файлы необходимы, когда программа разрастается, а также использует сторонние библиотеки классов.

Задание

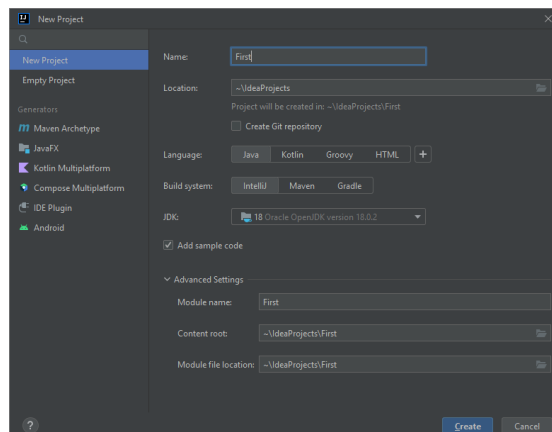
Необходимо установить JDK. Создать текстовый файл под названием `First.java`. В файл поместить код для вывода строки "My first program". Преобразовать текстовый файл в исполняемую программу. Создать `jar`-файл и запустить его.

IntelliJ IDEA и JDK

На [официальном сайте](#) необходимо скачать установщик IntelliJ IDEA для своей операционной системы.



После установки необходимо запустить и настроить проект.



Задание

Необходимо создать и запустить класс `First` в IntelliJ IDEA, который выведет строку "My first program".

Лексические основы языка

Программы на Java – это набор пробелов, комментариев, ключевых слов, идентификаторов, литеральных констант, операторов и разделителей.

Комментарии

Хотя комментарии никак не влияют на исполняемый код программы, при правильном использовании они оказываются весьма существенной частью исходного текста. Существует три разновидности комментариев: комментарии в одной строке, комментарии в нескольких строках и, наконец, комментарии для документирования. Комментарии, занимающие одну строку, начинаются с

символов `//` и заканчиваются в конце строки. Такой стиль комментирования полезен для размещения кратких пояснений к отдельным строкам кода.

Для более подробных пояснений можно воспользоваться комментариями, размещенными на нескольких строках, начав текст комментариев символами `/*` и закончив символами `*/`. При этом весь текст между этими парами символов будет расценен как комментарий и транслятор его проигнорирует.

Третья, особая форма комментариев, предназначена для сервисной программы `javadoc`, которая использует компоненты Java-транслятора для автоматической генерации документации по интерфейсам классов. Соглашение, используемое для комментариев этого вида, таково: для того, чтобы разместить перед объявлением открытого `public` класса, метода или переменной документирующий комментарий, нужно начать его с символов `/**` (косая черта и две звездочки). Заканчивается такой комментарий точно так же, как и обычный комментарий – символами `*/`. Программа `javadoc` умеет различать в документирующих комментариях некоторые специальные переменные, имена которых начинаются с символа `@`.

Зарезервированные ключевые слова

Зарезервированные ключевые слова – это специальные идентификаторы, которые в языке Java используются для того, чтобы идентифицировать встроенные типы, модификаторы и средства управления выполнением программы. На сегодняшний день в языке Java имеется 60 [зарезервированных слов](#). Эти ключевые слова совместно с синтаксисом операторов и разделителей входят в описание языка Java. Они могут применяться только по назначению, их нельзя использовать в качестве идентификаторов для имен переменных, классов или методов.

<code>abstract</code>	<code>continue</code>	<code>for</code>	<code>new</code>	<code>switch</code>
<code>assert***</code>	<code>default</code>	<code>goto*</code>	<code>package</code>	<code>synchronized</code>
<code>boolean</code>	<code>do</code>	<code>if</code>	<code>private</code>	<code>this</code>
<code>break</code>	<code>double</code>	<code>implements</code>	<code>protected</code>	<code>throw</code>
<code>byte</code>	<code>else</code>	<code>import</code>	<code>public</code>	<code>throws</code>
<code>case</code>	<code>enum****</code>	<code>instanceof</code>	<code>return</code>	<code>transient</code>
<code>catch</code>	<code>extends</code>	<code>int</code>	<code>short</code>	<code>try</code>
<code>char</code>	<code>final</code>	<code>interface</code>	<code>static</code>	<code>void</code>
<code>class</code>	<code>finally</code>	<code>long</code>	<code>strictfp**</code>	<code>volatile</code>
<code>const*</code>	<code>float</code>	<code>native</code>	<code>super</code>	<code>while</code>

* не используется

** добавлено в 1.2

*** добавлено в 1.4

**** добавлено в 5.0

Идентификаторы

Идентификаторы используются для именования классов, методов и переменных. В качестве идентификатора может использоваться любая последовательность строчных и прописных букв, цифр и символов `_` (подчеркивание) и `$` (доллар). Идентификаторы не должны начинаться с цифры, чтобы транслятор не перепутал их с числовыми литеральными константами, которые будут описаны ниже. Java – язык, чувствительный к регистру букв. Это означает, что, к примеру, `Value` и `VALUE` – различные идентификаторы.

Переменные

Переменная – это основной элемент хранения информации в Java-программе. Переменная характеризуется комбинацией идентификатора, типа и области действия. В зависимости от того, где объявили переменную, она может

быть локальной, например, для кода внутри цикла `for`, либо это может быть переменная экземпляра класса, доступная всем методам данного класса.

<code>int a, b, c;</code>	Объявляет три целые переменные a, b, c
<code>int d=3, e, f=5;</code>	Объявляет три целые переменные, инициализирует d и f
<code>byte y=22;</code>	Объявляет целую переменную y, инициализирует ее
<code>double pi=3.14;</code>	Объявляет вещественную переменную pi, инициализирует ее
<code>char y="y";</code>	Объявляет символьную переменную y, инициализирует ее

В данном простом примере объявлены две переменные. Первой переменной присваивается значение 2022, второй – 2022/2. Значения обоих переменных выводятся на консоль.

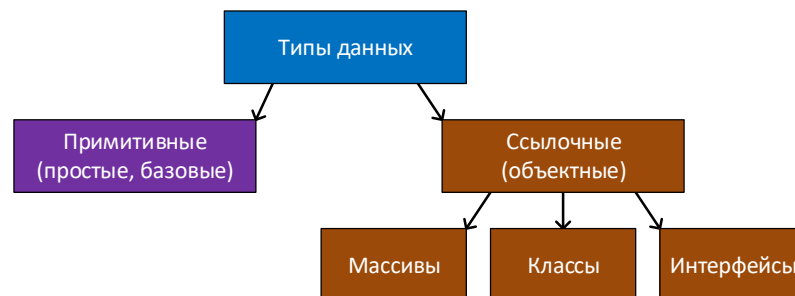
```
public class Main {  
    public static void main(String[] args) {  
        int a;  
        int b;  
        a=2022;  
        System.out.println("a = " + a);  
        b=a/2;  
        System.out.print("b = a/2 = " + b);  
    }  
}
```

Задание

Необходимо написать программу, которая будет вычислять и выводить значение по формуле: $a=4*(b+c-1)/2$; `b` и `c` задается константами в коде самостоятельно.

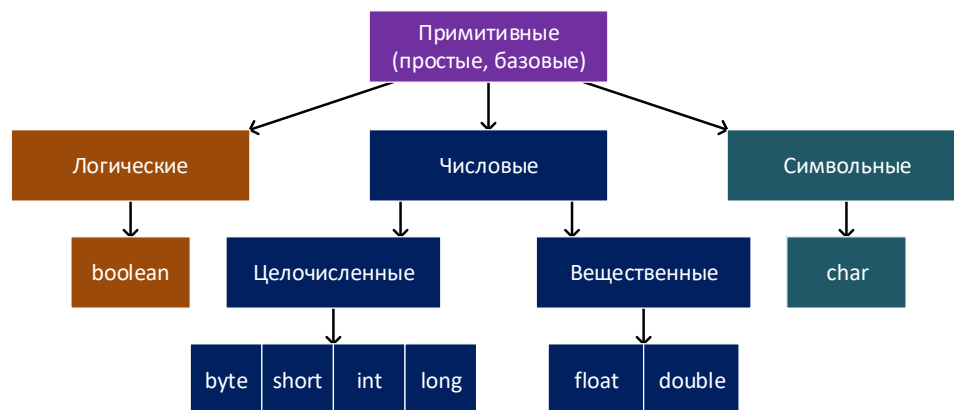
Примитивные типы данных

Типы данных в Java платформенно-независимые, они распределены на две категории: примитивные (простые) и ссылочные (объектные). Ссылочные типы – это массивы, классы и интерфейсы.



Примитивных, или простых, типов всего восемь:

- Целые числа. Эта группа включает в себя типы данных `byte`, `short`, `int` и `long`, представляющие целые числа со знаком.
- Числа с плавающей точкой (вещественные). Эта группа включает в себя типы данных `float` и `double`, представляющие числа с точностью до определенного знака после десятичной точки.
- Символы. Эта группа включает в себя тип данных `char`, представляющий символы, например, буквы и цифры, из определенного набора.
- Логические значения. Эта группа включает в себя тип данных `boolean`, специально предназначенный для представления логических значений.

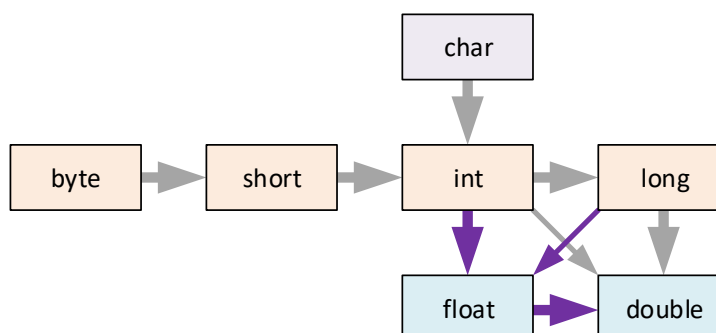


В Java, в отличие от некоторых других языков, отсутствует автоматическое приведение типов. Несовпадение типов приводит не к предупреждению при трансляции, а к сообщению об ошибке. Для каждого типа строго определены наборы допустимых значений и разрешенных операций.

Преобразование типов

Java запрещает смешивать в выражениях величины разных типов, однако при числовых операциях такое часто бывает необходимо. Различают повышающее (разрешенное, неявное) преобразование и понижающее приведение типа.

Повышающее преобразование осуществляется автоматически по следующему правилу (см. рисунок). Фиолетовыми стрелками обозначены преобразования, при которых может произойти потеря точности.



Преобразования между типами, несоединенными стрелками, неявно запрещены. Разработчик, проводя явное преобразование, берет полную ответственность за корректность преобразования на себя.

Явное приведение типа имеет вид.

(целевой тип)значение

```

public class Main {
    public static void main(String[] args) {
        byte b;
        int i=257;
        double d=323.142;
        b=(byte)i;
        i=(int)d;
        b=(byte)d;
    }
}
  
```

Задание

Необходимо модифицировать код программы, представленной выше так, чтобы видеть на консоли результаты приведения типов.

Арифметические операции

Большинство операций в Java аналогичны тем, которые применяются в других подобных языках. Есть унарные операции (выполняются над одним операндом), бинарные – над двумя операндами, а также тернарные – выполняются над тремя операндами. Операндом является переменная или значение (например, число), участвующее в операции. [Арифметические операторы](#) представлены в Java документации.

Задание

1. В переменной `n` хранится вещественное число с ненулевой дробной частью. Необходимо создать программу, округляющую число `n` до ближайшего целого и выводящую результат на экран.

2. В переменных `q` и `w` хранятся два натуральных числа. Необходимо создать программу, выводящую на экран результат деления `q` на `w` с остатком.

Консольный ввод с помощью класса `java.util.Scanner`

Для ввода данных используется класс `Scanner` из библиотеки пакетов `java.util`.

Этот класс надо импортировать в той программе, где он будет использоваться. Это делается до начала открытого класса в коде программы.

В классе есть методы для чтения очередного символа заданного типа со стандартного потока ввода, а также для проверки существования такого символа.

Для работы с потоком ввода необходимо создать объект класса `Scanner`, при создании указав, с каким потоком ввода он будет связан. Стандартный поток ввода (клавиатура) в Java представлен объектом – `System.in`. А стандартный поток вывода (экран) – объектом `System.out`.

```
import java.util.Scanner; // Импортируем класс
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in); // Создаем объект класса Scanner
        int i = 2;
        System.out.print("Введите целое число: ");
        if (sc.hasNextInt()) { // Возвращает истинну если с потока ввода можно считать целое число
            i = sc.nextInt(); // Считывает целое число с потока ввода и сохраняет в переменную
            System.out.println(i * 2);
        } else {
            System.out.println("Вы ввели не целое число");
        }
    }
}
```

Метод `hasNextDouble()`, примененный объекту класса `Scanner`, проверяет, можно ли считать с потока ввода вещественное число типа `double`, а метод `nextDouble()` – считывает его. Если попытаться считать значение без предварительной проверки, то во время исполнения программы можно получить ошибку (отладчик заранее такую ошибку не обнаружит).

```
import java.util.Scanner; // Импортируем класс
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double i = sc.nextDouble(); // Если ввести символ s, то случится ошибка времени исполнения
        System.out.println(i/3);
    }
}
```

Имеется также метод `nextLine()`, позволяющий считывать целую последовательность символов, т.е. строку, а, значит, полученное через этот

метод значение нужно сохранять в объекте класса `String`. В следующем примере создается два таких объекта, потом в них поочередно записывается ввод пользователя, а далее на экран выводится одна строка, полученная объединением введенных последовательностей символов.

```
import java.util.Scanner; // Импортируем класс
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s1, s2;
        s1 = sc.nextLine();
        s2 = sc.nextLine();
        System.out.println(s1 + s2);
    }
}
```

Существует метод `hasNext()`, проверяющий остались ли в потоке ввода какие-то символы.

Операторы

Как известно, любой алгоритм, можно разработать, используя линейные вычисления, разветвления и циклы. Всякий язык программирования должен иметь средства записи алгоритмов. Они называются *операторами* (statements) языка. Все операторы управления языка Java представлены в [документации](#).

Задание

1. С помощью цикла `while` и оператора `if` необходимо определить четность чисел и вывести их (числа от 1 до 10). 1 – нечетное, 2 – четное и т.д.
2. С помощью цикла `do-while` необходимо вывести на экран первые 10 элементов последовательности 2 4 8 16 32 64 128
3. С помощью цикла `for` необходимо подсчитать сумму всех четных чисел в диапазоне от -20 до 20.
4. С помощью оператора `switch` необходимо написать программу, которая по номеру месяца будет определять пору года. Номер месяца вводится с клавиатуры. Нужно предусмотреть проверку на некорректный ввод.

Статический импорт

Аккуратное и правильное использование `import static` улучшает читаемость кода. Для того чтобы получить доступ к статическим членам классов, требуется указать ссылку на класс. К примеру, необходимо указать имя класса `Math`:

```
double r = Math.cos(Math.PI * theta);
```

Конструкция статического импорта позволяет получить прямой доступ к статическим полям и методам класса. Можно импортировать каждый метод отдельно:

```
import static java.lang.Math.PI;
```

или все целиком:

```
import static java.lang.Math.*;
```

Однажды импортированный статический член может быть использован без указания имени класса:

```
double r = cos(PI * theta);
```

Объявление статического импорта аналогично объявлению обычного импорта. При объявлении обычного импорта классы импортируются из пакетов, что позволяет их использовать без указания имени пакета перед именем класса.

При объявлении статического импорта статические члены импортируются из классов, что позволяет им быть использованными без указания имени содержащего их класса.

Использование его оправданно, когда требуется постоянное использование статических членов одного класса из одного или двух других классов.

Чрезмерное использование статического импорта может сделать программу нечитаемой и тяжелой в поддержке ввиду того, что пространство имен увеличится из-за всех статических членов из импорта.

Класс Math

Разработчику на Java доступно множество готовых (или библиотечных) классов и методов, полезных для использования в собственных программах. Наличие библиотечных решений позволяет изящно решать множество типовых задач.

Далее рассмотрим класс `Math`, содержащий различные математически функции:

- `Math.abs(n)` – возвращает модуль числа `n`.
- `Math.round(n)` – возвращает целое число, ближайшее к вещественному числу `n` (округляет `n`).
- `Math.ceil(n)` – возвращает ближайшее к числу `n` справа число с нулевой дробной частью (например, `Math.ceil(3.4)` в результате вернет 4.0).
- `Math.cos(n)`, `Math.sin(n)`, `Math.tan(n)` – тригонометрические функции `sin`, `cos` и `tg` от аргумента `n`, указанного в радианах.
- `Math.acos(n)`, `Math.asin(n)`, `Math.atan(n)` – обратные тригонометрические функции, возвращают угол в радианах.
- `Math.toDegrees(n)` – возвращает градусную меру угла в `n` радианов.
- `Math.toRadians(n)` – возвращает радианную меру угла в `n` градусов.
- `Math.sqrt(n)` – возвращает квадратный корень из `n`.
- `Math.pow(n, b)` – возвращает значение степенной функции `n` в степени `b`, основание и показатель степени могут быть вещественными.
- `Math.log(n)` – возвращает значение натурального логарифма числа `n`.
- `Math.log10(n)` – возвращает значение десятичного логарифма числа `n`.

Все перечисленные методы принимают вещественные аргументы, а тип возвращаемого значения зависит от типа аргумента и от самой функции.

Кроме методов в рассматриваемом классе имеются две часто используемых константы:

- `Math.PI` – число Пи, с точностью в 15 десятичных знаков.
- `Math.E` – число Неппера (основание экспоненциальной функции), с точностью в 15 десятичных знаков.

В коде ниже приведены примеры использования некоторых методов.

```
public class Main {
    public static void main(String[] args) {
        System.out.println(Math.abs(-2.33)); // Выведет 2.33
        System.out.println(Math.round(Math.PI)); // Выведет 3
        System.out.println(Math.round(9.5)); // Выведет 10
        System.out.println(Math.round(9.5 - 0.001)); // Выведет 9
        System.out.println(Math.ceil(9.4)); // Выведет 10.0
        double c = Math.sqrt(3 * 3 + 4 * 4);
        System.out.println(c); // Гипотенуза треугольника с катетами 3 и 4
        double s1 = Math.cos(Math.toRadians(60));
        System.out.println(s1); // Выведет косинус угла в 60 градусов
    }
}
```

Задание

1. В переменных `a` и `b` лежат положительные длины катетов прямоугольного треугольника. Необходимо вычислить и вывести на экран площадь треугольника и его периметр.
2. Натуральное положительное число записано в переменную `n`. Необходимо определить и вывести на экран, сколько цифр в числе `n`.

Псевдослучайные числа

В классе `Math` есть полезная функция без аргументов, которая позволяет генерировать псевдослучайные значения, т.е. при каждом вызове этой функции она будет возвращать новое значение, предсказать которое очень сложно (не вдаваясь в подробности можно сказать, что теоретически это все-таки возможно, именно поэтому генерируемые функцией числа называются не случайными, а псевдослучайными).

Итак, `Math.random()` возвращает псевдослучайное вещественное число из диапазона от 0.0 до 1.0 / $[0;1)$.

Если требуется получить число из другого диапазона, то значение функции можно умножать на что-то, сдвигать и, при необходимости, приводить к целым числам.

В коде ниже приведены примеры создания случайных чисел в разных диапазонах.

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println(Math.random()); // Вещественное из [0;1)  
        System.out.println(Math.random() + 3); // Вещественное из [3;4)  
        System.out.println((int)(Math.random() * 5)); // Целое из [0;4)  
        System.out.println(Math.random() * 5 + 3); // Вещественное из [3;8)  
        System.out.println((int)(Math.random() * 11) - 5); // Целое из [-5;5]  
    }  
}
```

Псевдослучайные числа имеют серьезнейшие практические приложения и используются, например, в криптографии.

Задание

1. Необходимо создать программу, которая будет генерировать и выводить на экран вещественное псевдослучайное число из промежутка $[-3;3)$.
2. В переменные `a` и `b` записаны целые числа, при этом `b` больше `a`. Необходимо создать программу, которая будет генерировать и выводить на экран целое псевдослучайное число из отрезка $[a;b]$.

Генерация случайных чисел

В пакете `java.util` описан класс `Random`, являющийся генератором случайных чисел. На самом деле в силу своей природы ПК не может генерировать истинно случайные числа. Числа генерируются определенным алгоритмом, причем каждое следующее число зависит от предыдущего, а самое первое – от некоторого числа, называемого инициализатором. Две последовательности "случайных" чисел, сгенерированных на основе одного инициализатора, будут одинаковы.

Класс `Random` имеет два конструктора:

- `Random()` – создает генератор случайных чисел, использующий в качестве инициализатора текущую дату (число миллисекунд с 1 января 1970);

- `Random(long seed)` – создает генератор случайных чисел, использующий в качестве инициализатора число `seed`.

Рекомендуется использовать первый конструктор, чтобы генератор выдавал разные случайные числа при каждом новом запуске программы. От генератора можно получать случайные числа нужного типа с помощью методов `nextBoolean()`, `nextInt()`, `nextLong()`, `nextFloat()`, `nextDouble()`.

Вещественные числа генерируются в диапазоне от 0 до 1 (не включая 1), а целые – из всего диапазона возможных значений. Можно сгенерировать целое число в нужном диапазоне (от 0 до `max-1`) методом `nextInt(int max)` или `nextLong(long max)`.

Можно заполнить случайными числами целый массив (предварительно созданный), воспользовавшись методом `nextBytes(byte[] arr)`. Элементы массива `arr` должны иметь тип `byte`.

В коде ниже показан пример использования случайных чисел.

```
import java.util.Random; // Импортируем класс
public class Main {
    public static void main(String[] args) {
        Random r = new Random( seed: 100);
        for (int cnt = 0; cnt < 9; cnt++) {
            System.out.print(r.nextInt() + " ");
        }
        System.out.println();
        r = new Random( seed: 100);
        for (int cnt = 0; cnt < 9; cnt++) {
            System.out.print(r.nextInt() + " ");
        }
        System.out.println();
        byte[] randArray = new byte[8];
        r.nextBytes(randArray);
    }
}
```

Массивы в Java

Объявление и заполнение массива

Массив – это конечная последовательность упорядоченных элементов одного типа, доступ к каждому элементу в которой осуществляется по его индексу.

Размер или длина массива – это общее количество элементов в массиве. Размер массива задаётся при создании массива и не может быть изменен в дальнейшем, т. е. нельзя убрать элементы из массива или добавить их туда, но можно в существующие элементы присвоить новые значения.

Индекс начального элемента – 0, следующего за ним – 1 и т.д. Индекс последнего элемента в массиве – на единицу меньше, чем размер массива.

В Java [массивы](#) являются объектами. Это значит, что имя, которое дается каждому массиву, лишь указывает на адрес какого-то фрагмента данных в памяти. Кроме адреса в этой переменной ничего не хранится. Индекс массива, фактически, указывает на то, насколько надо отступить от начального элемента массива в памяти, чтоб добраться до нужного элемента.

Чтобы создать массив надо объявить для него подходящее имя, а затем с этим именем связать нужный фрагмент памяти, где и будут друг за другом храниться значения элементов массива.

тип[] имя;

где тип – это тип элементов массива, а имя – уникальный (незанятый другими переменными или объектами, в этой части программы) идентификатор, начинающийся с буквы.

Объявить имя для массива и создать сам массив можно в одной строке по следующей схеме.

тип[] имя = new тип[размер];

тип[] имя = {эло, эл1, ..., элn};

В коде ниже приведен пример создания массива целых чисел в диапазоне от 50 до 150. Количество элементов в массиве задает пользователь.

```
import java.util.Scanner; // Импортируем класс
public class Main {
    public static void main(String[] args) {
        System.out.println("Введите количество элементов массива:");
        Scanner sc = new Scanner(System.in);
        int x = 0, mas[];
        if (sc.hasNextInt()) {
            x = sc.nextInt();
        }
        mas = new int[x];
        for (int i = 0; i < mas.length; i++) {
            mas[i] = (int)(Math.random() * 100 + 50);
        }
        for (int i = 0; i < mas.length; i++) {
            System.out.println("mas[" + i + "]=" + mas[i] + ";");
        }
    }
}
```

Задание

1. Необходимо создать массив из всех нечетных чисел от 1 до 99, вывести его на экран в строку, а затем этот же массив вывести на экран тоже в строку, но в обратном порядке (99 97 95 93 ... 7 5 3 1).

2. Необходимо создать массив из 15 случайных целых чисел из отрезка [0;9]. Вывести массив на экран. Подсчитать сколько в массиве четных элементов и вывести это количество на экран в отдельной строке.

Сортировка массива

Сортировкой называется такой процесс перестановки элементов массива, когда все его элементы выстраиваются по возрастанию или по убыванию.

Сортировать можно не только числовые массивы, но и, например, массивы строк (по тому же принципу, как расставляют книги на библиотечных полках). Вообще сортировать можно элементы любого множества, где задано отношение порядка.

Существуют универсальные алгоритмы (сортировка выбором, сортировка пузырьком) которые выполняют сортировку вне зависимости от того, каким было исходное состояние массива. Но кроме них существуют специальные алгоритмы, которые, например, очень быстро могут отсортировать почти упорядоченный массив, но плохо справляются с сильно перемешанным массивом (или вообще не

справляются). Специальные алгоритмы нужны там, где важна скорость и решается конкретная задача.

Многомерные массивы

Массив может состоять не только из элементов какого-то встроенного типа (int, double и пр.), но и, в том числе, из объектов какого-то существующего класса и даже из других массивов.

Массив, который в качестве своих элементов содержит другие массивы называется многомерным массивом.

Чаще всего используются двумерные массивы. Такие массивы можно легко представить в виде матрицы. Каждая строка которой является обычным одномерным массивом, а объединение всех строк – двумерным массивом в каждом элементе которого хранится ссылка на какую-то строку матрицы.

В коде ниже приведен пример создания двумерного массива $m \times n$, заполнение его случайными числами и его вывод.

```
import java.util.Scanner; // Импортируем класс
public class Main {
    public static void main(String[] args) {
        System.out.println("Введите количество элементов массива:");
        Scanner sc = new Scanner(System.in);
        int m = 0, n = 0, a[][];
        if (sc.hasNextInt()) {
            m = sc.nextInt();
            n = sc.nextInt();
        }
        a = new int[m][n];
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                a[i][j] = (int)(Math.random() * 100 + 50);
            }
        }
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                System.out.print(a[i][j] + "\t");
            }
            System.out.println();
        }
    }
}
```

Задание

Необходимо создать двумерный массив из 7 строк по 4 столбца в каждой, заполненными случайными целыми числами из отрезка $[-5; 5]$. Вывести массив на экран. Определить и вывести на экран индекс строки с наибольшим по модулю произведением элементов. Если таких строк несколько, то вывести индекс первой встретившейся из них.

Нерегулярные массивы

Выделяя память под многомерный массив, сначала необходимо указать лишь размерность по одной (а именно по самой левой) координате. Остальные размерности можно указывать по отдельности.

Предположим, например, что необходимо написать программу, в процессе работы которой будет сохраняться число пассажиров, перевезенных микроавтобусом к самолетам. Если микроавтобус делает 10 рейсов в будние дни и по два рейса в субботу и воскресенье, то можно объявить мерности массивов

так, как это показано в приведенном ниже фрагменте кода. Размерность по второй координате для первых пяти значений индекса равна 10, а для остальных элементов – 2.

```
public class Main {
    public static void main(String[] args) {
        int riders[][] = new int[7][];
        // Размерность по второй координате равна 10
        riders[0] = new int[10];
        riders[1] = new int[10];
        riders[2] = new int[10];
        riders[3] = new int[10];
        riders[4] = new int[10];
        // Для остальных двух элементов размерность по второй координате равна 2
        riders[5] = new int[2];
        riders[6] = new int[2];
        int i, j;
        // Формирование произвольных данных
        for (i = 0; i < 5; i++) {
            for (j = 0; j < 10; j++) {
                riders[i][j] = i + j + 10;
            }
        }
        for (i = 5; i < 7; i++) {
            for (j = 0; j < 2; j++) {
                riders[i][j] = i + j + 10;
            }
        }
        System.out.println("Riders per trip during the week:");
        for (i = 0; i < 5; i++) {
            for (j = 0; j < 10; j++) {
                System.out.print(riders[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println();
        System.out.println("Riders per trip on the weekend:");
        for (i = 5; i < 7; i++) {
            for (j = 0; j < 2; j++) {
                System.out.print(riders[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Для большинства приложений использовать нерегулярные массивы не рекомендуется, поскольку это затрудняет восприятие кода другими разработчиками.