**Оглавление**

1.Введение 2

2.Виртуальная машина Java 5

3.Первая программа 6

4.Создание проекта в среде разработки NetBeans 7

5.Точка входа в программу / Главный класс 14

6.Правила именования и комментирование кода 17

7.Ключевые слова 18

8.Переменные 18

9.Примитивные типы данных 19

9.1. boolean 19

9.2.byte 19

9.3.short 19

9.4.int 19

9.5.long 20

9.6.float 20

9.7.double 20

9.8.char 20

10. Массивы 20

10.1.Одномерные массивы 21

10.2.Многомерные массивы 22

11.Арифметические операции 25

11.1.Операция сложения 25

11.2.Операция вычитания 26

11.3.Операция умножения 26

11.4.Операция деления 27

12.Побитовые операции 28

12.1.Побитовый унарный оператор NOT 28

12.2.Побитовое AND 29

12.3.Побитовое OR 29

12.4.Побитовое исключающее OR 29

12.5.Сдвиг вправо 29

12.6.Сдвиг влево 30

12.7.Сдвиг вправо с заполнением нулями 30

13.Логические операции 31

13.1.Логические операции &, | и ^ 32

13.2.Операция инвертирования 32

13.3.Укороченные логические операции 32

13.4.Логические операции с присваиванием 33

14.Операторы сравнения 33

15.Управляющие операторы 34

15.1.Оператор выбора if 34

15.1.1Оператор if ( if-then ) 34

15.1.2Оператор if-else 35

15.1.3Конструкция if-else-if 35

15.1.4Тернарная операция 36

15.2.Оператор выбора switch 37

15.3.Цикл while 40

15.4.Цикл do-while 40

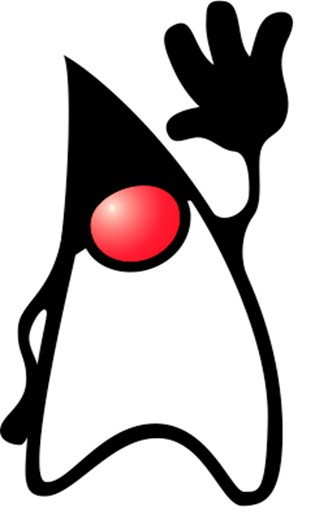
15.5.Цикл for 40

15.6.Оператор break 40

15.7.Оператор continue 40

# Введение

Java, как и любой другой язык следует начать изучать с рассмотрения причин его возникновения. Сегодня Java принадлежит компании Oracle, однако зародился язык в SunMicrosistems (Sun). В 1990 году Sun производил сервера и высокопроизводительные станции, но ничего не мог предложить обычному пользователю. В тот год из компании хотел перейти в компанию NeXT программист Патрик Нотон (Patrick Naughton), объяснение причины было «Они все делают правильно». Прежде чем уволить Нотона, его друг и член совета директоров, президент и исполнительный директор Скотт МакНили (Scott McNealy) попросил Патрика Нотона описать проблему в Sun и ее решение. Просьба была выполнена, в ней было много критики и предложений, но полезно выделить два предложения: привлечь профессиональных дизайнеров для проработки пользовательских интерфейсов, отказаться от использования нескольких оконных технологий, остановить выбор на одной и развивать ее. МакНили разослал все руководящему составу, и вся информация получила поддержку многих, особенно она заинтересовала Билла Джоя (Bill Joy - один из основателей и вице-президент Sun) и Джеймса Гослинга (James Gosling — ведущий разработчик). 5 декабря 1990 года в последний рабочий день Нотон получает предложение от Sun и остается в компании. 1 февраля 1991 года родился проект Green, его задачей было создать что-нибудь необычайное. В проекте приняли участие: Патрик Нотон, Джеймс Гослинг и Майк Шеридан (Mike Sheridan). Обратив внимание на широкое распространение компьютерных чипов и проблему «поиск пульта», которых в домах обычно несколько: от телевизоров, видеомагнитофонов, музыкальных центров, было решено разработать универсальное устройство-пульт, для управления различными бытовыми приборами. Гослинг взялся за программное обеспечение, но быстро пришел к выводу, что C++ недостаточно надежен и приступил к написанию нового языка. Название языка родилось сразу — Oak (дуб), в честь дуба который рос под окном Гослинга. Через несколько месяцев новый язык был совмещен с графической системой разработанной Нотоном. С применением "технологии молотка" было собрано устройство получившее название Star7. В августе 1991 года состоялась первая демонстрация, а 4 сентября 1992 года разработка завершена. Star7 обладал 5” цветным (16 бит) сенсорным экраном и простым интерфейсом. Интерфейс представлял нарисованный дом, по комнатам которого было удобно перемещаться. Для помощи по работе в интерфейсе был создан забавный персонаж Дюк (Duke).



1 ноября 1992 года создается компания FirstPerson, назначение компании — продвижение платформы Oak на рынок. Возглавила FirstPerson Вэйн Роузинг (Wayne Rosing). Стоимость продвигаемого программно-аппаратного решения составляла от 50$, именно это сделало решение невостребованным на рынке, так как производители бытовой техники не сочни нужным увеличивать стоимость своего товара ради такого усовершенствования. FirstPerson делает попытки взаимодействовать с производителями игровых приставок, но опять неудача и в начале 1994 руководство Sun требует разработки нового бизнес плана. Родилась идея создания платформы для кабельных компании, но она не была утверждена руководством и Sun уже готово отказаться Oak, но Билл Джой еще раз оказывает поддержку этому Оак. Наконец обратив внимание на стремительное развитие сети Internet, Джой делает заключение, что WWW очень близок по идеологии приложениям на Oak. На тот момент одним из недостатков WWW и существующих браузеров было то, что вся логика была на сервере, а браузер был способен только отображать текстовый результат. О интерактивности речи не шло, но для увеличения скорости и придачи интерактивности возникает идея перенести часть серверной логики на сторону персонального компьютера пользователя. В связи с тем, что персональные компьютеры использовали разные платформы, такая разработка была дорогостоящей, но не для Oak, который исходно направлен именно на решение именно такой задачи. Для реализации не хватает браузера поддерживающего новую технологию и Нотон его создает. Браузеру дано имя WebRunner и в сентябре 1994 года его демонстрируют руководству Sun. Приложения работающие под управлением WebRunner дано название applet (апплет). На конференции разработчиков Internet-приложений Oak и WebRunner ждет успех. В 1995 году к проекту подключаются маркетологи и переименовывают Oak в Java, а WebRunner в HotJava. В феврале 1995 выходит пресс-релиз о скором выходе альфа-версий. Распространение технологии набирало обороты и 23 марта 1995 года газета Sun Jose Mercury News публикует статью о технологии и объявляет официальный Internet-адрес официального сайта <http://java.sun.com/>. 23 мая 1995 года официально объявлено, что самый популярный браузер Netscape Navigator 2.0 будет поддерживать новую технологию. Вышедший Internet Explorer 2.0 компании Microsoft не заинтересовал конечного пользователя и Microsoft вынужден в марте 1996 года лицензировать Java для выпуска конкурентоспособного Internet Explorer 3.0.

В январе 1996 года в свет выходит финальная версия *JDK* 1.0.

В январе 1997 года выходит версия JDK 1.1.

В декабре 1998 доступной становится следующая версия j2sdk 1.2, благодаря большому количеству изменения ее именуют Java 2 и начиная с нее Java развивается в нескольких направлениях:

Java 2 Platform, Standard Edition (J2SE);

Java 2 Platform, Enterprise Edition (J2EE);

Java 2 Platform, Micro Edition (J2ME).

В мае 2000 года выходим версия j2sdk 1.3

В феврале 2002 года *J2SE* 1.4

Начиная с пятой версии изменен принцип нумерации версии

В 2004 году выходит J2SE 5.0

Пятая версия является последней из группы Java2

В 2006 году выходит Java 6

21 апреля 2009 года компания Oracle покупает SunMicrosistems

В 2011 году после долгого затишья выходит Java 7

В марте 2014года выпущена текущая версия Java 8

# Виртуальная машина Java

Не просто язык, а платформа. Под платформой обычно подразумевается программно-аппаратное решение. Языки программирования обычно вынуждают программистов генерировать исполняемый байт-код отдельно для каждой платформы. Java упростила разработку и

“Write once, run anywhere!”

написанное однажды работает везде.

Единственное условие — программа работает там, где установлена виртуальная машина Java (JVM - Java virtual machine)

Изначально программист пишет код в текстовых файлах с расширением .java, далее компилятор преобразует исходный код в понятный машине байт-код.

Для разработки и запуска приложения требуются:

JRE - Java Runtime Environment - Виртуальная машина, используется для запуска скомпилированных ранее классов

JDK - Java Development Kit - набор инструментов для разработки (компилятор)

JRE входит в состав JDK

После перехода к Oracle, JRE и JDK стали доступны на сайте oracle.ru

Ссылка для загрузки JDK / JRE:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

# Первая программа

Пример приограммы «Привет мир!» есть практически на всех языках программирования, ниже приведен код программы «Привет мир!» на Java

public class FirstClass {

public static void main(String[] args) {

System.out.println("Привет!!!");

}

}

Далее необходимо сохранить код в текстовый файл «FirstClass.java». Программа, хоть и маленькая, написана, но ее надо привести в вид понятный машине, а точнее скомпилировать. В наборе инструментов для разработки (JDK) присутствует компилятор, он расположен в

«папка инсталляции JDK»/bin/java.exe (для Windows)

и

«папка инсталляции JDK»/bin/java (для Unix)

Команда компиляции программы:

javac FirstClass.java

(если ОС не видит исполняемый файл javac, то пропишите полный путь до него, например:

c:\Program Files\Java\jdk1.8.0\_25\bin\javac.exe FirstClass.java

).

После ее исполнения команды, рядом с файлом FirstClass.java, должен создаться еще один – FirstClass.class, именно он хранит в себе понятный виртуальной машине код.

Команда запуска программы:

java FirstClass

(по аналогии с компиляцией, программа может быть запущена следующей командой:

c:\Program Files\Java\jdk1.8.0\_25\bin\java.exe FirstClass

)

Исполняемый файл java.exe присутствует, как в JDK, так и в JRE и, как следствие, конечному пользователю достаточно иметь только JRE.

Обратите внимание на то, что при запуске программы, в отличие от компиляции, не надо указывать расширение файла.

Выше описан пример компиляции программы состоящей из одного класса, однако большинство программ состоят из набора классов. Javac и java позволяют скомпилировать и запустить и их, для этого следует использовать дополнительные параметры этих команд. Так как далее компиляция и запуск программ будет осуществляться с использованием сред разработки, а так же в связи с тем, что при реальной разработке в «ручную» компиляция запускается крайне редко, изучение дополнительных параметров запуска javac и java остается на самостоятельное изучение.

# Создание проекта в среде разработки NetBeans

Integrated Development Environment (IDE) - среда разработки ПО

На сегодняшний день наиболее распространенными являются три среды разработки:

* Eclipse - получила широкое распространение благодаря широким возможностям и бесплатному распространению.

Ссылка для загрузки:

http://eclipse.org

* IntellijIdea - является коммерческим продуктом компании JetBrains, имеет бесплатную версию с ограничением по функциональности (нет поддержки JavaEE). Хорошо спроектированный интерфейс, удобство использования и постоянное совершенствование принесли ей широкое распространение.

Ссылка для загрузки:

http://www.jetbrains.com/

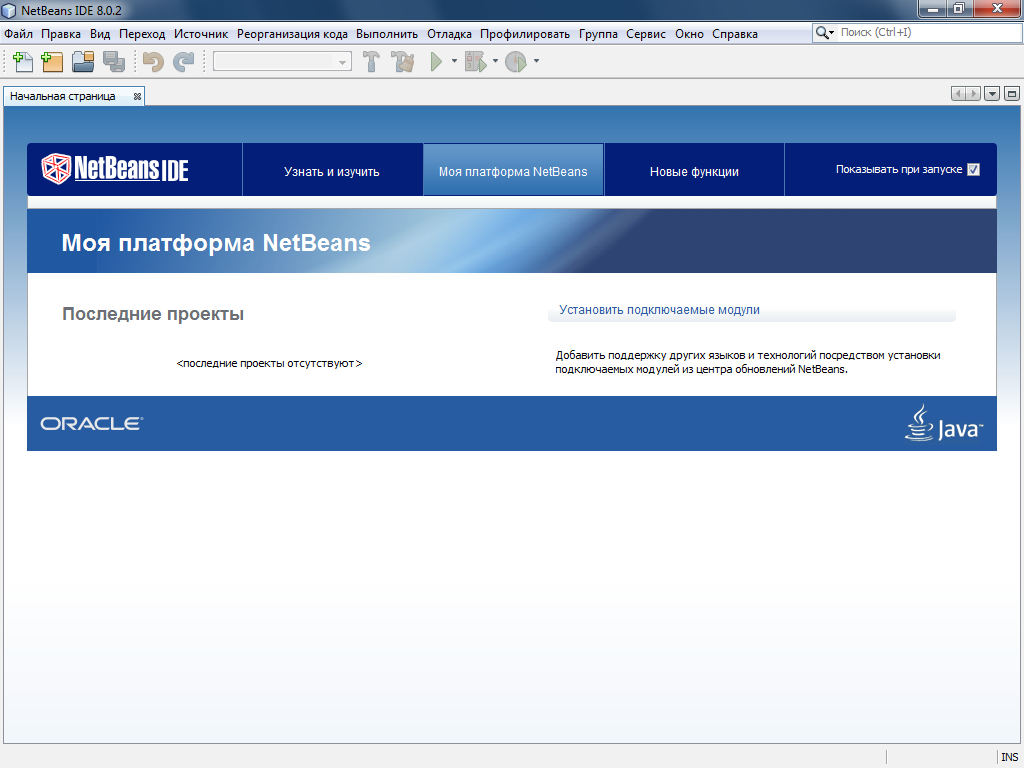
* NetBeans - не уступает по популярности и возможностям двум предыдущим, поддерживается компанией Oracle.

Ссылка для загрузки:

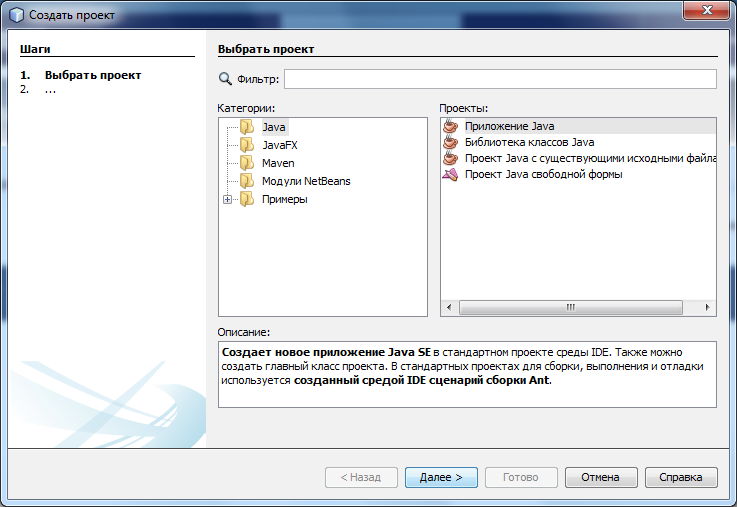
https://netbeans.org/

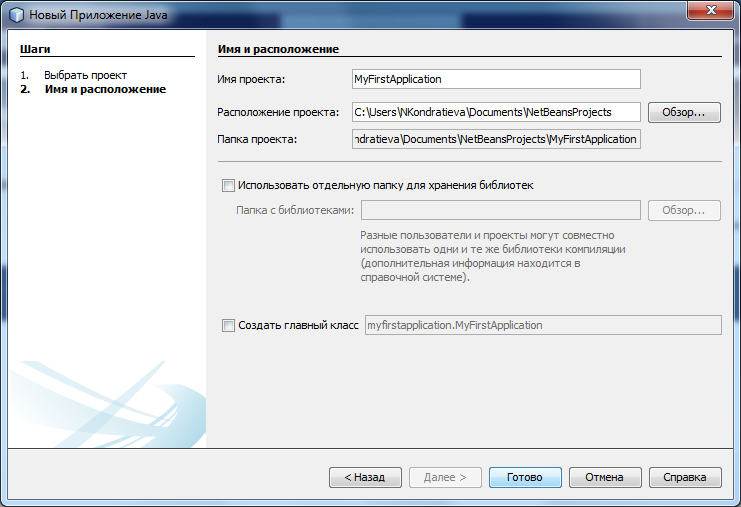
Ниже приведен пример создания и запуска проекта в NetBeans (при самостоятельном создании проекта допустимо использование любой из вышеперечисленных IDE)

При первом запуске IDE открывается следующее окно:

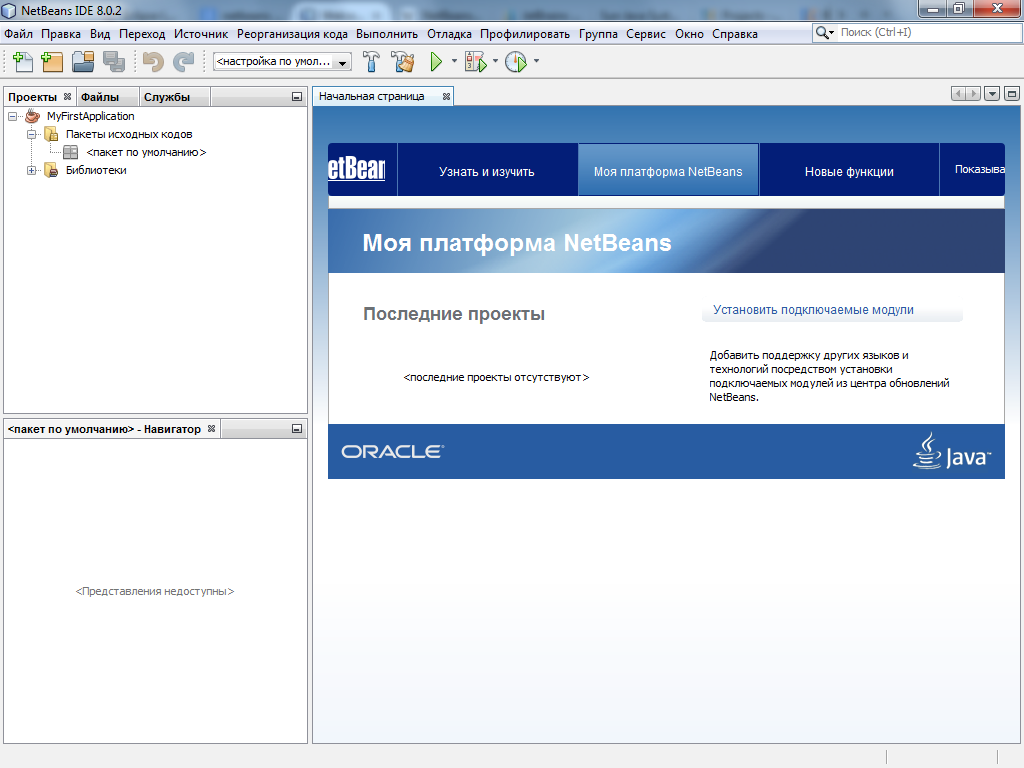
При последующих запусках IDE (при стандартных настройках) автоматически будет восстановлено состояние на момент выхода.

- иконка для создание нового проекта, проект может быть создан через меню: Файл → Создать проект.

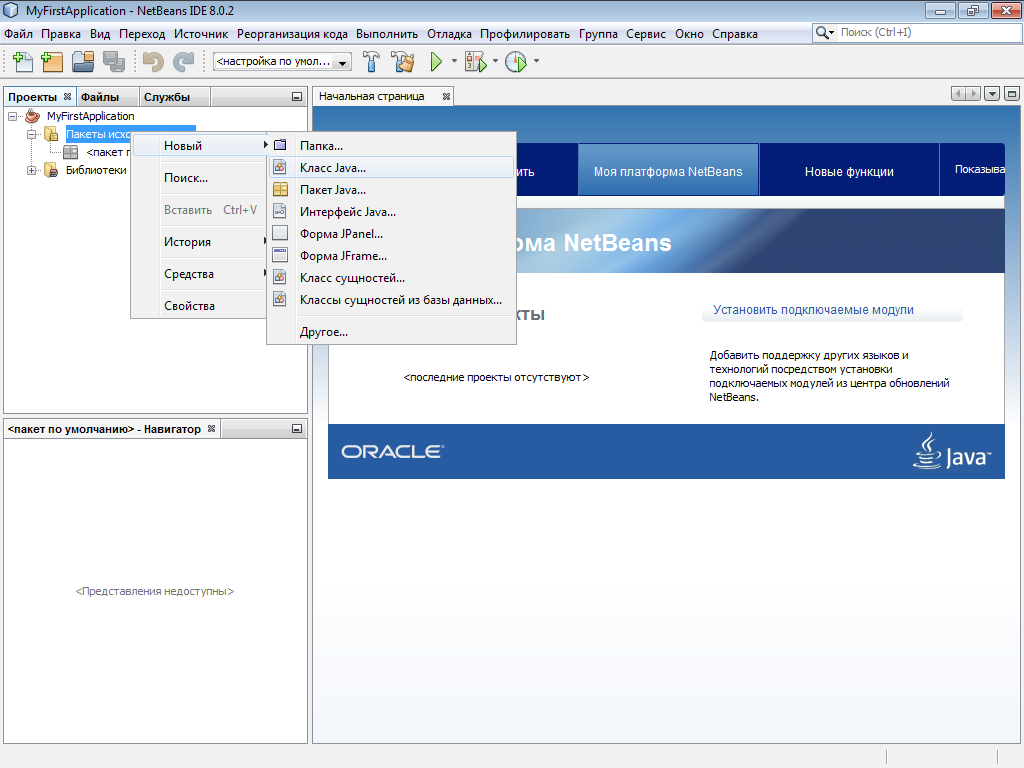
При выборе проекта выбираем категорию «Java и проекты «Приложение Java» 

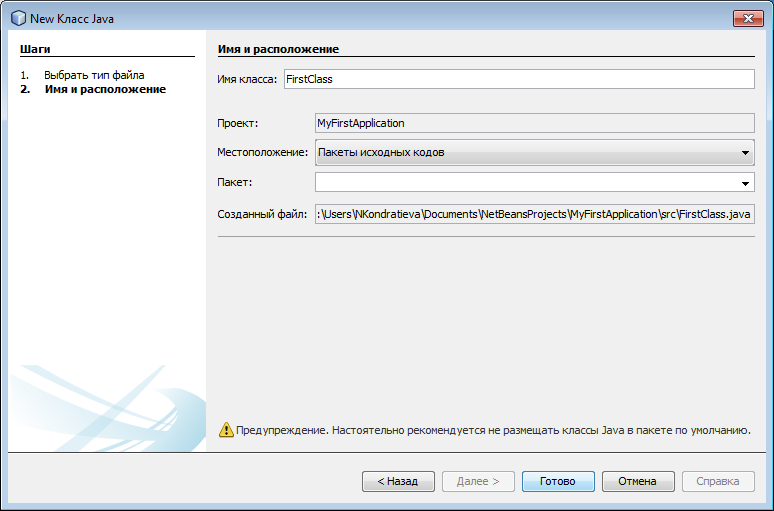
При создании проекта необходимо задать имя и папку, в которой будет храниться проект. 

При создании нового проекта IDE предлагает создать главный класс (точку входа в программу). В текущем примере галка снята и класс будет создаваться вручную.

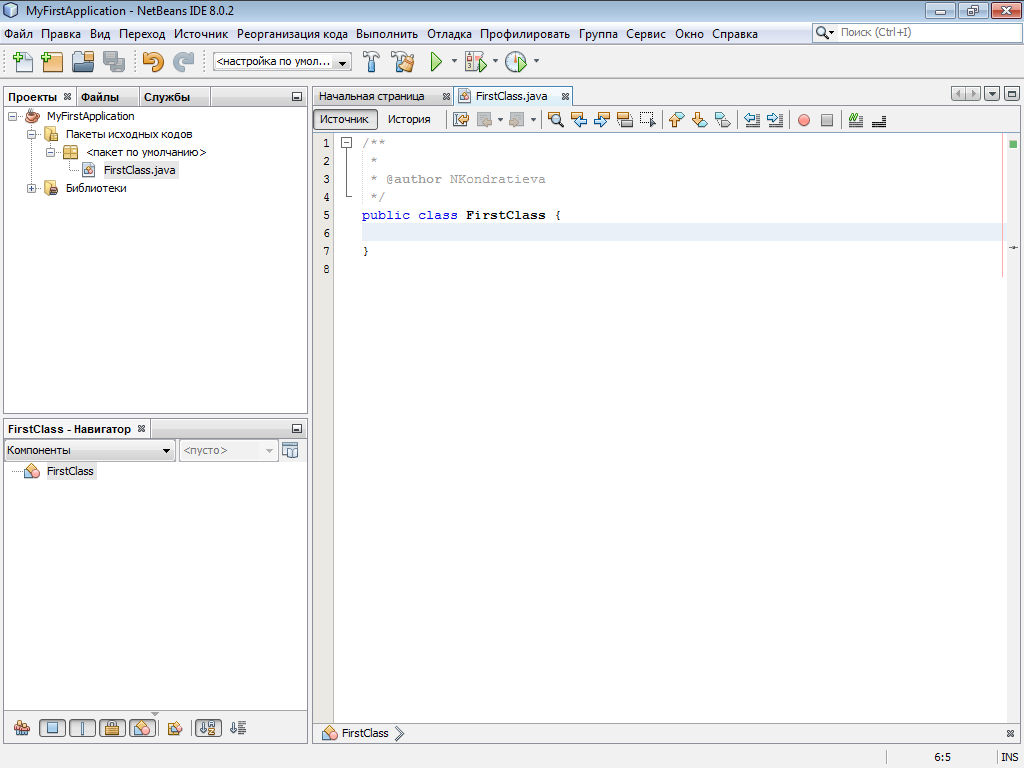
На скриншоте показано то, как выглядит проект в котором нет ни одного класса. 

Ранее, при создании проекта, галка «Создать главный класс» была снята, следовательно создаем класс вручную.

Новые классы должны создаваться в папке «src», NetBeans отображает ее как «Пакеты исходных кодов»



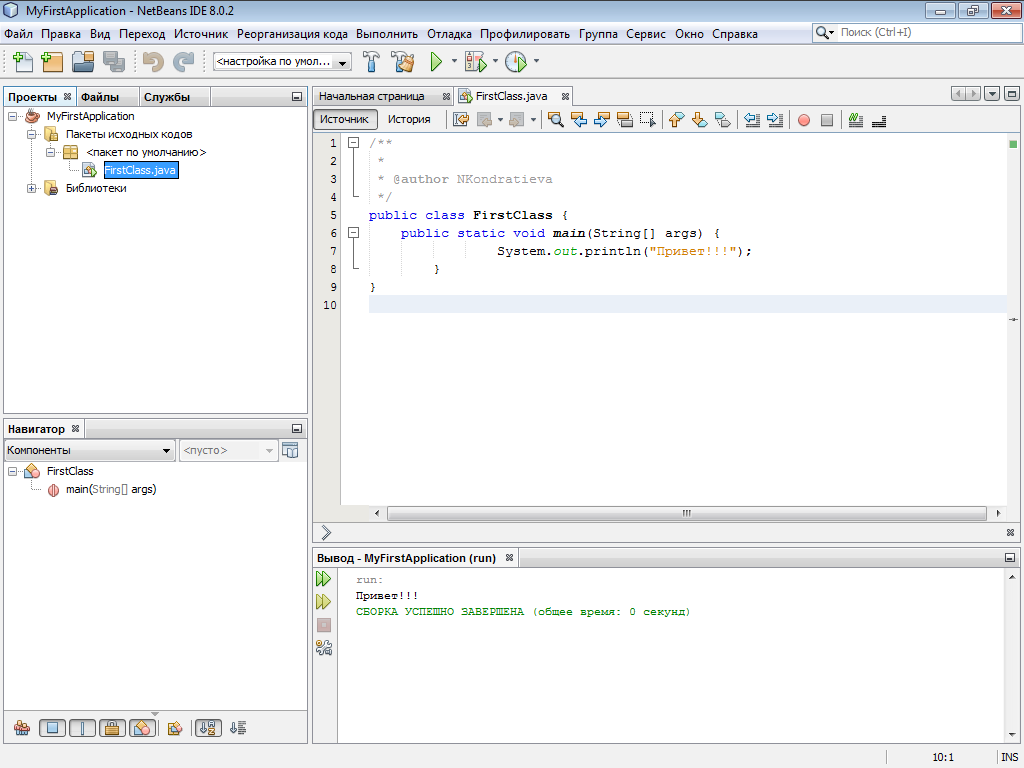
Классу необходимо задать имя, в примере это «FirstClass».

При создании класса, IDE автоматически открывает его для редактирования.

Для написания программы «Привет!!!» воспользуемся кодом из раздела «Первая программа».

Запуск программы может быть осуществлен тремя способами:

* через меню: Выполнить → Запустить проект, при таком первом запуске NetBeans запросит имя главного класса
* используя иконку
* через контекстное меню в структуре проекта «Выполнить файл»

После запуска программы IDE открывает консоль, в NetBeans она открывается во вкладке «Вывод», и в ней отображает ход выполнения программы. 

# Точка входа в программу / Главный класс

Пустой класс:

class FirstClass {

}

Класс в Java является базовым элементом, позволяет описывать структуры объектов.

Из приведенного примера кода видно, что у нас есть класс и называется он FirstClass. Тело класса заключено в фигурные скобки.

В Java существует такое понятие как модификатор доступа. Модификатор позволяет определить область видимости класса, метода, переменной. Для запуска класса, он должен быть публичным (public). Это не единственный модификатор доступа, подробнее о них будет рассказано далее.

После добавления модификатора доступа, класс выглядит следующим образом:

public class FirstClass {

}

Класс виден всем, но в нем нет кода, который следует исполнить. Для описания действий, которые должны происходить в программе используются методы. Все методы класса лежат внутри тела класса.

Класс с пустым методом:

public class FirstClass {

void firstMethod() {

}

}

void - означает, что метод не имеет возвращаемого значения, если в результате выполнения метод должен вернуть данные, то void меняется на тип возвращаемых данных.

firstMethod – имя метода.

После имени всегда следуют круглые скобки, а тело метода заключается в фигурные.

Как и класс, метод должен быть доступен снаружи и для этого его необходимо сделать публичным добавив модификатор доступа public.

public class FirstClass {

public void firstMethod() {

}

}

Методов в классе может быть много, но выполняться программа может начать только с метода имеющего имя «main»

public class FirstClass {

public void main() {

}

}

Программы не всегда запускаются без указания дополнительных параметров, например: команда перемещения в windows

move [источник] [результат]

Источник и результат являются параметрами передаваемыми приложению при запуске.

Для получения параметров метод должен знать о том, что в него будет что-то передано. Так как из консоли можно передавать только строковые переменные и их количество может быть любым, метод main обязан принимать на вход массив строк (подробнее о массивах и строка будет рассказано далее). Для этого в круглых скобках после имени метода необходимо указать тип принимаемого значения и имя переменной, для доступа к этому значению.

public class FirstClass {

public void main(String[] args) {

}

}

Теперь метод main отличается от того, что был в примере первой программы только словом static, для метода являющегося точкой входа в программу оно является обязательным и пока для нас это только маркер того, что мы можем вызвать метод из любой точки программы (подробнее о static будет рассказано далее).

Минимально необходимая конструкция для запуска программы.

public class FirstClass {

public static void main(String[] args) {

}

}

При запуске такой программы она выполнится, но ничего не произойдет, так как тело метода main пустое.

Команда:

System.out.print("Привет!!!");

выводит в консоль все, что указано в круглых скобках. Обратите внимание на то, что текст указывается в двойных кавычках.

Подставив эту команду в тело метода main и запустив приложение результат будет идентичен тому, что был в разделе «Первая программа»

# **Правила именования и комментирование кода**

У каждого программиста может быть свой стиль написания кода, но если каждый будет писать код так как ему будет удобно, то это приведет к снижению скорости чтения вашего кода другими программистами. Для устранения такой проблемы были разработаны правила.

* имена классов всегда пишутся с заглавной буквы, если имя класса должно содержать несколько слов, то каждое слово должно быть написано с заглавной буквы, например: FirstClass
* имена методов пишутся со строчной буквы, если имя метода должно содержать несколько слов, то слова начиная со второго должны быть написаны с заглавной буквы, например: firstMethod
* имена переменных именуются по правилам именования методов
* имена констант (неизменяемых переменных) пишутся заглавными буквами, при этом каждое последующее слово отделяется от предыдущего «\_» (подчеркиванием), например: MAX\_VALUE

Помимо именования, полезными для разработчиков ПО могут быть комментарии в коде. Существуют три способа позволяющие оставить комментарий в коде:

System.out.print("Привет!!!"); // комментарий

текст идущий после «//» до конца строки не учитывается при компиляции и является комментарием.

Если есть необходимость исключить из компиляции толь часть строки или несколько строк, то необходимо исключаемый текст заключить между «/\*» и « \*/» ,например :

System./\*комментарий\*/out.print("Привет!!!");

При необходимости внести комментарий в документацию его заключают между «/\*\*» и «\*/», например:

/\*\*

\*описание класса

\*/

public class FirstClass {…

Ознакомиться с полным набором правил можно в документе Java Code Conventions доступном на сайте http://oracle.com .

# Ключевые слова

Слова, которые легли в основу языка являются зарезервированными и запрещены для использования в качестве имен классов, методов и переменных.

Таблица ключевых слов:

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

К приведенным ключевым словам добавлены еще три зарезервированных: true , false и null.

# Переменные

Объявление переменной состоит из указания типа данных которые сможет хранить переменная и имени переменной. Например:

int width;

где:

* int – тип данных
* width – имя переменной

Синтаксис языка позволяет объявить сразу несколько переменных:

int width, height;

Эта запись является аналогом:

int width;

int height;

Для занесения значения в переменную используется оператор присваивания «=» (равно):

width = 1920;

Присвоить переменной значение можно сразу при объявлении:

int width = 1920, height=1080;

# Примитивные типы данных

Java – строго типизированный язык. Каждая переменная обладает типом и тип строго определен.

Существует 8 элементарных типов (их также называют простыми или примитивными): boolean, byte, short, int, long, float и double.

## boolean

Для хранения логических переменных используется тип данных boolean. Возможных значений только 2: true (истина) и false (лож). Является результатом операторов сравнения и используется в условных операторах.

boolean isActive = true, canBeDrop = false;

## byte

Тип данных byte предназначен для хранения небольших целочисленных значений. byte - может принимать значения от -128 до 127, в оперативной памяти занимает 8 бит;

byte course = 5;

## short

Как и byte предназначен для хранения целочисленных значений, но диапазон принимаемых значений шире: от –32768 до 32767 . Вес переменной в оперативной памяти 16 бит.

short width = 1920;

## int

Является наиболее часто используемым типом данных для хранения целочисленных значений. Может принимать значения от –2147483648 до 2147483647, а вес составляет 32 бита.

int height = 1080;

## long

Для хранения больших целочисленных значений используется тип данных long, он предоставляет возможность хранить значения от -9223372036854775808 до 9223372036854775807 и весит 64 бита.

long time = 1420070400000;

## float

При вычислениях недостаточно обладать переменными только целочисленных типов, для хранения чисел с плавающей точной используется тип данных float. Он способен хранить значения от от 1.4e-45 до 3.4e+38, вес его составляет 32 бита.

float pi = 3.14f;

Обратите внимание на символ f после числа, он говорит о том, что задаваемое значение именно float, а не double, который используется по умолчанию.

## double

Этот тип данных позволяет хранить самые большие значения с плавающей точкой, диапазон принимаемых значений от 4.9e-324 до 1.8e+308. Весит такой тип данных 64 бита.

double max = 1234.56789;

## char

Используется char для хранения символов. Хранятся символы в кодировке Unicode. Значения переменных типа char могут быть от 0 до 65536, а вес переменной этого типа 16 бит.

Занести символ в переменную типа char можно следующим образом:

char symbol = 'X';

В таблице ASCII символу 'X' принадлежит код 88, тип данных char позволяет задавать значения при помощи кодов символов, следующая запись является аналогом предыдущей:

char symbol = 88;

# Массивы

Хранить каждое значение в отдельной переменной не всегда удобно и требуется централизовано хранить набор однотипных данных. Для таких целей существуют массивы.

Массивами называют группы однотипных значений доступ к которым осуществляется по имени одной и той же переменной.

## Одномерные массивы

Объявить переменную хранящую массив можно следующим образом:

int[] values;

где:

int – тип данных, которые должен хранить массивах;

values – имя переменной хранящей массив.

Внимание: запись иногда можно встретить в следующем виде:

int values[];

При создании массива необходимо определить его размер:

new int[7];

где:

new – оператор говорит о том, что будет создан объект не примитивного типа и для его хранения будет выделена область памяти;

int – тип данных, которые будет хранить создаваемый массив;

7 — число определяющее количество значений которое можно занести в создаваемый массив.

Визуально такой массив можно представить себе как объединенный набор ячеек для хранения значений:

| Значение0 | Значение1 | Значение2 | Значение3 | Значение4 | Значение5 | Значение6 |
| --- | --- | --- | --- | --- | --- | --- |

Обратите внимание на то, что нумерация ячеек хранящий значения начинает свою нумерацию с 1.

При таком создании массив автоматически заполняется значениями, при этом массивы числовых типов заполнятся нулями, логического типа во всех ячейках будет хранить значение false. Если массив будет хранить не примитивные типы данных, а ссылочные, то во все ячейки будут заполнены значением null.

Массив предоставляет возможность заполнить себя сразу в момент своего создания:

int[] values = new int[]{ 11 , 22 , 33 , 44 , 55 , 66 , 77 };

Все значения, которые должны быть внесены в массив перечисляются через запятую в фигурных скобках.

Обратите внимание: при использовании такой инициализации массива его размер не указывается, за размер массива отвечает число значений перечисленное в фигурных скобках.

Изменить значение хранимое в ячейке можно следующим образом:

values[3] = 10;

где:

values – имя переменной по которой можно обратиться к массиву

[3] – индекс (или номер) ячейки в которую будет занесено новое значение

10 — заносимое значение.

Получение доступа к значениям хранимым в массиве ( для примера рассмотрим получение значения из ячейки массива и занесение его в переменную):

int singleValue = values[3];

где:

int singleValue – тип и имя переменной, в которую будет заноситься значением хранимое в ячейке массива;

values – имя переменной по которой можно обратиться к массиву

[3] – индекс ячейки, из которой будет взято значение.

## Многомерные массивы

Многомерный массив представляет собой массив в ячейке которого хранится другой массив, иначе говоря многомерный массив — это массив массивов.

Объявление многомерного массива:

int [][] values;

Инициализация многомерного массива:

values = int[3][7];

Представим этот массив в уже заполненном виде:

| [3][7] | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 10 | 20 | 30 | 40 | 50 | 60 |
| 1 | 11 | 22 | 33 | 44 | 55 | 66 | 77 |
| 2 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

При создании массива был указан размер [3][7], как следствие, при визуализации мы получили таблицу с тремя строками и семью колонками.

Заполнение двумерного массива производится схоже с заполнением одномерного:

values[1][2] = 333;

где:

values – имя переменной по которой можно обратиться к массиву

[1][2] – индекс ячейки в которую будет занесено новое значение

333 — заносимое значение.

В результате выполнения этой операции, значение массива лежащее во второй сроке и третьей колонке (до исполнения операции значение равнялось 33) примет значение 333.

Получение доступа к значениям хранимым в двумерном массиве:

int singleValue = values[1][6];

где:

int singleValue – тип и имя переменной, в которую будет заноситься значением хранимое в ячейке массива;

values – имя переменной по которой можно обратиться к массиву

[1][6] – индекс ячейки, из которой будет взято значение.

В приведенном примере используется двумерный массив, так как его удобно представлять графически, однако нет ограничений на n-мерность массивов.

Например, объявление и инициализация 5 мерного массива хранящего логические значения можно записать так:

boolean[][][][][] boleanValues = new boolean[2][8][4][6][10];

Значения многомерного массива, как и одномерного можно, можно определить в момент его инициализации. Пример инициализации двумерного массива:

boolean[][] booleanValues = new boolean{

{ true, false, true, false, true, false, true, false},

{false, false, true, false, false, false, true, false},

{true, true, true, false, true, true, true, false}

};

В результате будет создан массив с размерностью [3][8].

При инициализации многомерного массива обязательным является только определение размера первого массива;

int[][] values = new int[3][];

Определить размерность вложенных массивов можно так:

values[0] = new int[7];

values[1] = new int[7];

values[2] = new int[7];

Эта запись полностью идентична:

values = new int[3][7];

Смысл данная особенность приобретает в тот момент, когда необходимо иметь вложенные массивы разных размеров.

Например, создадим массив со значениями логического типа, размер первого массива будет соответствовать числу месяцев в году, внутри него будут храниться массивы с размером равным числу дней в соответствующем месяце:

boolean[][] days = new boolean[12][];

days[0] = new boolean[31]; // январь

days[1] = new boolean[28]; // февраль

days[2] = new boolean[31]; // март

days[3] = new boolean[30]; // апрель

days[4] = new boolean[31]; // май

days[5] = new boolean[30]; // июнь

days[6] = new boolean[31]; // июль

days[7] = new boolean[31]; // август

days[8] = new boolean[30]; // сентябрь

days[9] = new boolean[31]; // октябрь

days[10] = new boolean[30]; // ноябрь

days[11] = new boolean[31]; // декабрь

Определение длины массива:

int length = days.length;

где:

days – массив, размер которого необходимо определить

length — параметр массива хранящий его размер.

Один из примеров использования возможности определить размер массив: определение длины массива получаемого на вход в метод main, для получения информации о числе параметров переданных приложению снаружи при его запуске.

# Арифметические операции

Имея числовые значения зачастую возникает потребность производить над мини арифметические операции.

## Операция сложения

Операция сложения требует на вход 2 переменных и возвращает результирующее значение.

byte value1 = 5, value2 = 10, value3;

Так как результат может выйти за рамки значений допускаемых byte, операция возвращает значение типа int;

int result = value1 + value2;

Если надо получить результат типа short, то для этого надо явно указать тип (привести тип int к типу short), это действие называют приведение типов.

value3 = (byte) result;

или

value3 = (byte) (value1 + value2);

Обратите внимание на то, что value1+ value2 заключено в скобки, если их убрать, то приведение типов будет относиться только к value1, а не к сумме.

Если потребуется занести результат не в переменную value3, а в value1, которая является одним из слагаемых, это можно сделать следующими способами:

полная форма записи:

value1 = (byte) (value1 + value2);

краткая форма записи:

value1 += value2;

В программировании часто возникает потребность увеличивать значение переменной на единицу, это можно сделать используя как пллную, так и краткую форму записи, также для выполнения такой простой операции существует еще один способ, который называют инкрементацией:

value1++;

## Операция вычитания

Операция вычитания, как и сложения имеет все три записи:

полная форма записи:

value1 = (byte) (value1 — value2);

краткая форма записи:

value1 += value2;

декрементация:

value1--;

## Операция умножения

Как и две предыдущие операции, операция умножения может быть записана с использованием полной и краткой формы:

полная форма записи:

value1 = (byte) (value1 \* value2);

краткая форма записи:

value1 \*= value2;

## Операция деления

Операция деления имеет полную и краткую форму записи.

value1 = (byte) (value1 / value2);

краткая форма записи:

value1 /= value2;

Результат примера будет целочисленный, например:

int value1 = 10, value2 = 5;

int value3 = value1 / value2;

В результате value3 равно 2;

Если необходимо получить в результате выполнения операции не целочисленное значение, а значение с дробной частью, то необходимо:

* сохранять результат в переменную соответствующего типа (если необходимо сохранение)

float value3 = value1 / value2;

В результате value3 равно 2.0

* указать что тип делимого или делителя является типом с плавающей точкой

float value3 = (float) value1 / value2;

или

* float value3 = value1 / (float) value2;

В результате value3 равно 2.5.

Получить остаток от деления можно следующим образом:

int value3 = value1 % value2;

Для данной операции, как и для предыдущих, доступна краткая форма записи:

value1 %= value2;

# Побитовые операции

Значения целочисленных типов данных позволяют производить над содой не только арифметические операции, в Java к ним могут быть применены несколько побитовых операций.

Ниже приведены допустимые побитовые операции:

| Оператор | Описание |
| --- | --- |
| ~ | Побитовый унарный оператор NOT (НЕ) |
| & | Побитовое AND (И) |
| &= | Побитовое AND с присваиванием |
| | | Побитовое OR (ИЛИ) |
| |= | Побитовое OR с присваиванием |
| ^ | Побитовое исключающее OR |
| ^= | Побитовое исключающее OR с присваиванием |
| >> | Сдвиг вправо |
| >>= | Сдвиг вправо с присваиванием |
| << | Сдвиг влево |
| <<= | Сдвиг влево с присваиванием |
| >>> | Сдвиг вправо с заполнением нулями |
| >>>= | Сдвиг вправо с заполнением нулями с присваиванием |

Для примера возьмем два числа:

byte a = 53, b = 11;

Перед применением побитовой операции, для понимания ее работы переведем числа в двоичную систему счисления.

53 запишем как 00110101

11 запишем как 00001011

## Побитовый унарный оператор NOT

В ходе выполнения операции происходит инвертирование всех битов операнда:

если a равно 53 (00110101),

то после выполнения:

a = ~a;

а будет равно -54 (11001010)

## Побитовое AND

При выполнении побитового AND сравниваем по И каждую пару бит.

если a равно 53 (00110101), а b равно 11 (00001011)

то после выполнения:

a = a & b;

а будет равно 1 (00000001)

Для выполнения a = a & b; можно использовать краткую запись — побитовое AND с присваиванием:

a &= b;

## Побитовое OR

При выполнении побитового OR сравниваем по ИЛИ каждую пару бит.

если a равно 53 (00110101), а b равно 11 (00001011)

то после выполнения:

a = a | b;

а будет равно 63 (00111111)

Для выполнения a = a | b; можно использовать краткую запись — побитовое OR с присваиванием:

a |= b;

## Побитовое исключающее OR

При выполнении побитового исключающего OR сравниваем по ИЛИ каждую пару бит и записываем 1 — если биты разные, и 0 — если биты одинаковые.

если a равно 53 (00110101), а b равно 11 (00001011)

то после выполнения:

a = a ^ b;

а будет равно 62 (00111110)

Для выполнения a = a ^b; можно использовать краткую запись — побитовое исключающее OR с присваиванием:

a ^= b;

a |= b;

## Сдвиг вправо

При выполнении операции сдвигаем все биты вправо на указываемое число позиций.

если a равно 53 (00110101)

то после выполнения:

a = a >> 1;

а будет равно 26 (00011010)

если a равно -53 (11001011)

то после выполнения:

a = a >> 1;

а будет равно -27 (11100101)

Если необходимо выполнить сдвиг вправо только на единицу, то можно использовать краткую запись — сдвиг вправо с присваиванием:

a >>= 1;

## Сдвиг влево

При выполнении операции сдвигаем все биты влево на указываемое число позиций.

если a равно 53 (00110101)

то после выполнения:

a = a << 1;

а будет равно 106 (00011010)

если a равно -53 (11001011)

то после выполнения:

a = a << 1;

а будет равно -106 (11100101)

Если необходимо выполнить сдвиг влево только на единицу, то можно использовать краткую запись — сдвиг влево с присваиванием:

a <<= 1;

## Сдвиг вправо с заполнением нулями

При выполнении операции сдвигаем все биты вправо на указываемое число позиций, при этом левый бит принимает значение 0.

если a равно 53 (00110101)

то после выполнения:

a = a >>> 1;

а будет равно 26 (00011010)

если a равно -53 (11001011)

то после выполнения:

a = a >>> 1;

а должно равняться -27 (11100101)

НО!

Операция особенна тем, что перед выполнением принимаемое значение c количеством разрядом меньшем 32 приводится к int и как следствие мы имеем:

-53 это 11111111111111111111111111001011

после a = a >>> 1;

a = 2147483621

(01111111111111111111111111100101)

Если необходимо выполнить сдвиг вправо с заполнением нулями только на единицу, то можно использовать краткую запись — сдвиг вправо с заполнением нулями c присваиванием:

a >>>= 1;

# Логические операции

Логические операции применяются к операндам типа boolean. Результатом исполнения логической операции над двумя операндами типа boolean является значение типа boolean.

Ниже приведены допустимые логические операции:

| Оператор | Описание |
| --- | --- |
| & | Логическая операция И |
| | | Логическая операция ИЛИ |
| ^ | Логическая операция исключающее ИЛИ |
| ! | Логическая унарная операция НЕ |
| && | Укороченная логическая операция И |
| || | Укороченная логическая операция ИЛИ |
| &= | Логическая операция И с присваиванием |
| |= | Логическая операция ИЛИ с присваиванием |
| ^= | Логическая операция исключающее ИЛИ с присваиванием |
| == | Равенство |
| != | Неравенство |

## Логические операции &, | и ^

Работа логических операторов &, | и ^ аналогична работе с отдельными разрядами значений целочисленных типов.

Для примера возьмем две логических переменных и рассмотрим применение к ним логических операторов:

| a | b | a & b | a | b | a ^ b |
| --- | --- | --- | --- | --- |
| true | true | true | true | false |
| true | false | false | true | true |
| false | true | false | true | true |
| false | false | false | false | false |

## Операция инвертирования

Логический оператор ! применяется для инвертирования значения логической переменной.

Если boalean a = true;

То после выполнения:

boolean b = !a;

Результат: b = false

Если boolean a = false;

То после выполнения:

boolean b = !a;

Результат: b = false

## ***Укороченные логические операции***

Укороченные логические операции добавлены в java для оптимизации вычисления результирующего значения.

Для примера работы укороченной логической операции воспользуемся числовой переменной:

byte value = 10;

Рассмотрим ход решения следующего выражения:

boolean result = ( value == 5) && ( value == 10);

Первым действием будет проверено равенство переменной value значению 5. Так как результат отрицательный и его достаточно для определения конечного результата, проверяться равенство переменной valuе значению 10 не будет.

Рассмотрим ход решения следующего выражения:

boolean result = ( value == 10) || ( value == 5);

Первым действием будет проверено равенство переменной value значению 10. Так как результат положительный и его достаточно для определения конечного результата, проверяться равенство переменной valuе значению 5 не будет.

## Логические операции с присваиванием

Логические операции с присваиваниями осуществляются аналогично побитовым операциям.

При необходимости исполнения логических операций:

a = a & b; a = a | b; a = a ^b;

доступно использование логических операции с присваиванием, которые имеет вид:

a &= b; a |= b; a ^= b;

# Операторы сравнения

Операторы сравнения используются для определения отношения одного значения к другому. Результатом работы оператора всегда является логическое значение, например:

даны:

int value1 = 100, value2 = 200;

Оператор — равно:

boolean result = value1 == value2; // если значения равны, то результат result равен true иначе false.

Оператор — больше:

boolean result = value1 > value2; // если первое значение больше второго, то результат result равен true иначе false.

Оператор — меньше:

boolean result = value1 < value2; // если первое значение меньше второго, то результат result равен true иначе false.

Оператор — больше или равно:

boolean result = value1 >= value2; // если первое значение больше или равно второму, то результат result равен true иначе false.

Оператор — меньше:

boolean result = value1 <= value2; // если первое значение меньше или равно второму, то результат result равен true иначе false.

# Управляющие операторы

Линейное исполнение программ является частным случаем. В ходе описания алгоритмов работы возникают потребности:

* выполнения того или иного блока кода исключительно при выполнении определенного условия;
* использование одного блока кода при n-раз подряд;
* прерывания исполнения блока кода

## Оператор выбора if

### Оператор if ( if-then )

Условный оператор if позволяет ограничить исполнение блока кода выполнением определенного условия.

Наиболее простая форма записи выглядит следующим образом:

if (условие) оператор;

где:

условие — это логическое выражение или переменная логического типа;

оператор — это код исполняемый в случае положительного результата

выполнения условия.

Например:

if (value==10) // value – переменная типа int

System.out.println(“Значение равно десять”);

Текст «Значение равно десять» выведется только, если value равно 10.

if (value) // value – переменная типа boolean

System.out.println(“Значение положительно”);

Текст «Значение положительно» выведется только если value равно true

Для исполнения нескольких операторов при выполнении условия, необходимо фигурными скобками объединить операторы в блок кода.

Например:

if (условие){

оператор1;

оператор2;

}

### Оператор if-else

Условный оператор if позволяет не только определить блок кода который исполняется в случае положительного результата при исполнении условия, но и предоставляет возможность определить то, что должно происходить во всех противных случаях.

В общем виде оператор можно записать следующим образом:

if (условие) оператор\_1 else оператор\_2;

Рассмотрим работу оператора на примере:

int a = 10;

if ( a == 10 ){ // условие

System.out.println(“Переменная «а» равна 10.”); //оператор\_1

}else{

System.out.println(“Переменная «а» неравна 10.”); //оператор\_2

}

### Конструкция if-else-if

Блоки кода исполняемый при истинном и ложном результате выполнения условия могут содержать внутри себя вложенные операторы if.

Например:

if (a==1){

System.out.println("Переменная «a» равна 1");

} else {

if (a==2){

System.out.println("Переменная «a» равна 2");

} else {

System.out.println("Переменная «a» неравна 1 и 2");

}

}

При ложном результате первого условного оператора будет исполняться блок содержащий вложенный оператор if. Так как вложенный оператор if является единственным в блоке, заключать его в фигурные скобки нет необходимости. А так как java не принуждает к выставлению переводов строк в коде, запись можно представить в следующем компактном виде if-else-if:

if (a==1){

System.out.println("Переменная «a» равна 1");

} else if (a==2){

System.out.println("Переменная «a» равна 2");

} else {

System.out.println("Переменная «a» неравна 1 и 2");

}

### Тернарная операция

Для записи операция if-then-else можно использовать тернарную операцию, которая обозначается знаком « ? ».

Запись тернарной операции выглядит следующим образом:

условие ? оператор\_1 : оператор\_2

Применяется тернарный оператор для формирования сокращенной записи.

Например:

if (b){

System.out.println(“Да”);

}else {

System.out.println(“Нет”);

}

Можно записать с использованием тернарной операции:

System.out.println( b? “Да” : ”Нет”);

Обратите внимание на то, что тернарная операция используется для совместимых типов данных.

Например:

int a;

if (b){

a = 1;

} else{

a = 0;

}

Можно записать:

int a = b?1:0;

Но при попытке записи:

int a = b?1:””;

компилятор выдаст ошибку, так как строка не может быть записана в числовую переменную «a».

## Оператор выбора switch

Оператор выбора switch часто называют оператором ветвления. Ветвление модно организовать с помощью оператора if-else-if, но в ряде случаев switch оказывается эффективнее.

Рассмотрим вариант вывода дня недели в консоль по его номеру используя оператор if-else-if

int dayNumber = 7;

if (dayNumber == 1) {

System.out.println("Понедельник");

} else if (dayNumber == 2) {

System.out.println("Вторник");

} else if (dayNumber == 3) {

System.out.println("Среда");

} else if (dayNumber == 4) {

System.out.println("Четверг");

} else if (dayNumber == 5) {

System.out.println("Пятница");

} else if (dayNumber == 6) {

System.out.println("Суббота");

} else if (dayNumber == 7) {

System.out.println("Воскресенье");

} else {

System.out.println("Такого дня нет");

}

Для замены if-then-else на switch рассмотрим общий вид записи оператора switch:

switch(параметр){

case: значение\_1

оператор\_1\_1;

оператор\_1\_2;

break;

case: значение\_2

оператор\_2\_1;

оператор\_2\_2;

break;

case: значение\_N

оператор\_N\_1;

оператор\_N\_2;

break;

default:

оператор\_default\_1;

оператор\_default\_2;

}

Параметр определяет то, по какой из ветвей пойдет исполняться программа. В примере с днями недели этим параметром является dayNumber.

Допустимые типы параметров:

* byte
* short
* int
* char
* enum // о нем будет рассказано далее
* String // поддержка типа String добавлена в JDK 7

Значения указанные в case сравниваются с параметром и в случае положительного результата исполняются операторы следующие за case.

Обратите внимание на то, что операторы не объединяются в блок фигурными скобками, связано это с тем, что будут исполнены все операторы расположенные до оператора break или до завершения оператора switch. Эта особенность позволяет исполниться операторам из нескольких ветвей.

Операторы указанные в default исполняются при условии, что ни одно из значений не совпало с параметром.

В отличие от if-else-if switch контролирует набор значений и не позволит в два разных case поставить одно и то же значение или значения отличающиеся по типу от параметра.

Рассмотрим то, как будет выглядеть задача с днями недели с использованием оператора switch:

switch (dayNumber) {

case 1:

System.out.println("Понедельник");

break;

case 2:

System.out.println("Вторник");

break;

case 3:

System.out.println("Среда");

break;

case 4:

System.out.println("Четверг");

break;

case 5:

System.out.println("Пятница");

break;

case 6:

System.out.println("Суббота");

break;

case 7:

System.out.println("Воскресенье");

break;

default:

System.out.println("Такого дня нет");

}

Рассмотрим вариант использования оператора switch в котором при нескольких значениях исполняются одни и те же операторы:

switch (dayNumber) {

case 1:

case 2:

case 3:

case 4:

System.out.println("Рабочий день");

break;

case 5:

System.out.println("Сокращенный рабочий день");

break;

case 6:

case 7:

System.out.println("Выходной");

break;

default:

System.out.println("Такого дня нет");

}

## Цикл while

Циклы позволяют исполнять один и тот же блок кода и предоставляют возможность завершения работы при исполнении условия.

Рассмотрим конструкцию цикла while:

while ( условие ){

операторы;

}

где:

условие — это логическое выражение или переменная логического типа

операторы — тело цикла

Цикл будет повторять исполнение оператора или блока операторов до те пор, пока условие возвращает истину.

Например:

short[] days = new short[]{31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

int mohth = 0;

while ( mohth < 12) {

System.out.println("В " + ( mohth + 1) + "-ом месяце " + days[mohth] + (days[mohth] % 2 == 0 ? " дней" : " день"));

mohth++;

}

Приведенный код решает задачу вывода на экран информации о количестве дней в месяце. Для этого первой строкой создается массив из 12 ячеек (по одной на каждый месяц) и в каждую ячейку занесена информация о числе дней в соответствующем месяце.

## Цикл do-while

## Цикл for

## Оператор break

## Оператор continue