Лабораторная работа №1
ОСНОВЫ ПРОГРАММИРОВАНИЯ НА JAVA. СОЗДАНИЕ ПРИЛОЖЕНИЙ В Eclipse

Java — объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине вне зависимости от компьютерной архитектуры. Разработка Java началась в 1990 году, первая официальная версия — Java 1.0, — была выпущена только 26 августа 1996 года. К 1998 году была разработана обновлённая спецификация JDK 1.2, вышедшая под наименованием Java 2.
Выпуск версии Java 8 состоялся 19 марта 2014 года.

При появлении Java двумя основными формами Java-программ являлись **приложение** и **апплет**.

Java-приложения выполняются под управлением специального интерпретатора (java.exe). Приложения похожи на программы, созданные, например, с использованием языка C/C++, хотя для своей работы они требуют присутствия Java виртуальной машины (JVM). Это полноправные приложения, которые существуют и выполняются в локальных компьютерных системах пользователей.

Java-апплеты разработаны для функционирования в сети и выполняются как часть Web-станиц. Апплеты требуют наличия соответствующего Java-браузера, так как они должны загружаться по сети с Web-сервера в обеспечивающую их работоспособность среду исполнения Java на локальном компьютере.

Инструментальная среда разработки программ на Java

Для создания программ на Java возможно использование нескольких сред разработки. Это может быть ***Microsoft Visual J++, JBuilder,IntelliJ Idea, Eclipse или NetBeans IDE***.

Установка и запуск Eclipse IDE

1. Переходим по ссылке http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/keplerr и скачиваем Eclipse, соответствующую операционной системе.
   Далее будет рассмотрена установка для Windows.
2. Распаковываем скаченный .zip архив в директорию **C:\Program Files**
3. **На этом установка завершена!**
4. Чтобы запустить Eclipse IDE, нужно открыть файл **eclipse.exe**, находящийся в папке **C:\Program Files\eclipse\.**
5. При запуске откроется окно, предлагающее выбрать рабочую область (Workspace), где будут храниться программные файлы проекта. Указываем удобную для нас директорию (рис. 3.1.) и нажимаем **OK**.

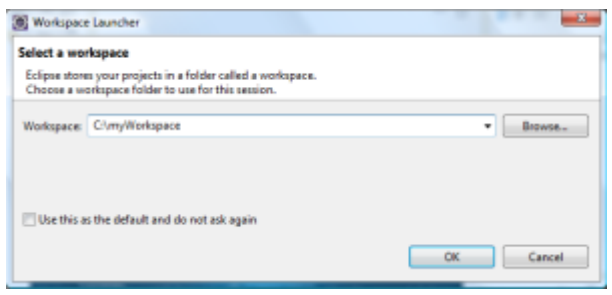

Рис. 3.1.

6. Закрываем приветственное сообщение (рис. 3.2.), тем самым перейдя в рабочую среду.

Рис. 3.2

Начало работы с Eclipse IDE

Создадим новый проект. Для этого выберем меню **File->New->Project.**

В открывшемся окне выберем **Java Project** (рис. 3.3.) и нажмем **Next**.

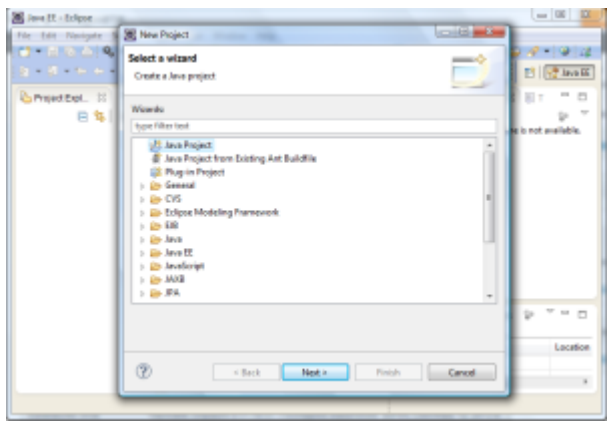

Рис.3.3.

В следующем окне введем имя нашего проекта (рис. 3.4.) и нажмем **Finish**.

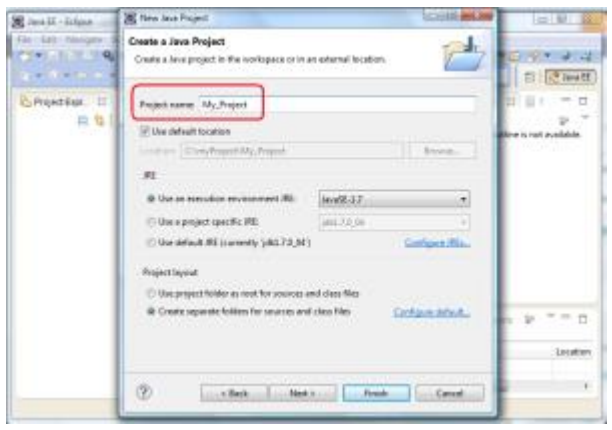

Рис. 3.4

Проект отобразится в левой части экрана и должен в себе содержать элемент **JRE System Library** (рис. 3.5.)

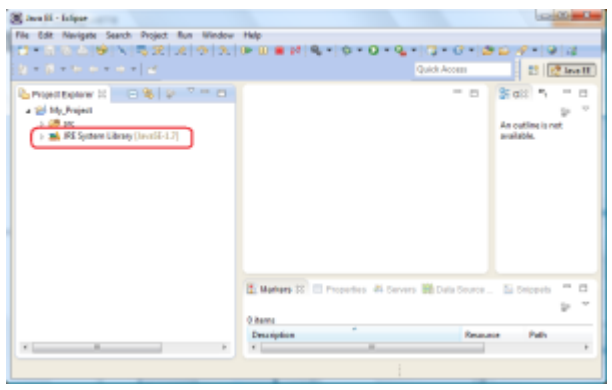

Рис. 3.5.

Если этого элемента нет, то его необходимо добавить вручную! Для этого выберем **Windows -> Preferences**, в открывшемся окне **Preferences** слева выберем **Java -> Installed JREs**, нажмем кнопку **Add…** справа (рис 3.6.). В открывшемся окне выберем **Standard VM** и нажмем кнопку **Next.**

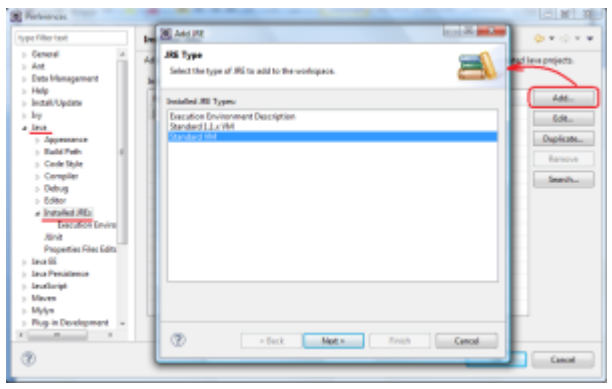

Рис 3.6.

В открывшемся окне **Add JRE**, укажем директорию, в которой установлена Java (рис 3.7.) и нажмем **Finish.**

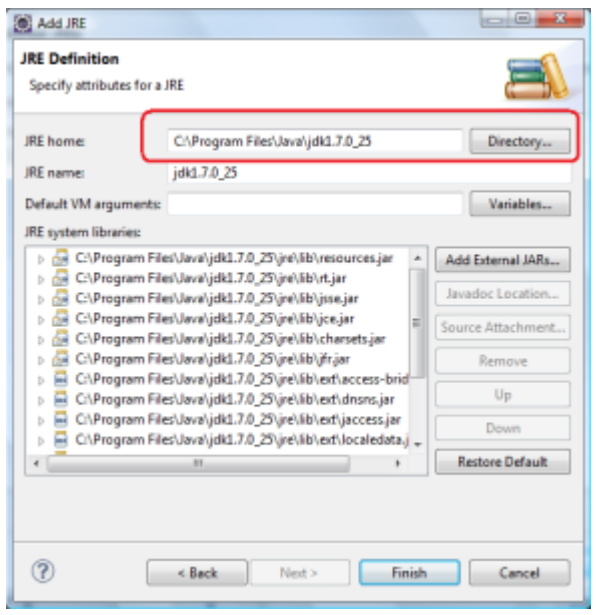

Рис. 3.7.

Далее рассмотрим создание программы **Hello World** в Eclipse.

Первым делом необходимо создать класс. Нажмем правой кнопкой на папке с проектом и выберем из контекстного меню **New -> Class** (рис 2.7.).

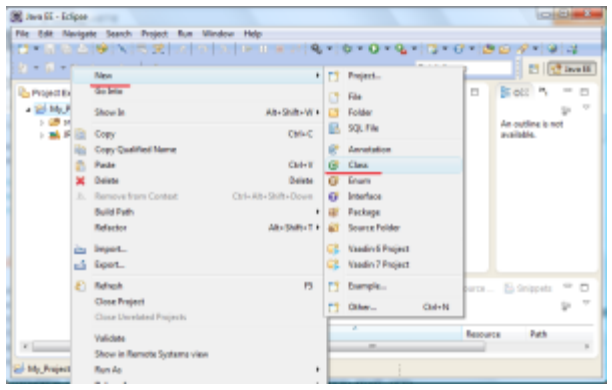

Рис 2.7.

В открывшемся окне **New Java Class** введем имя класса проекта **HelloWorld** и установим флажок для метода **public static void main(String[] args)** (рис 2.8.). **Нажмем Finish**.

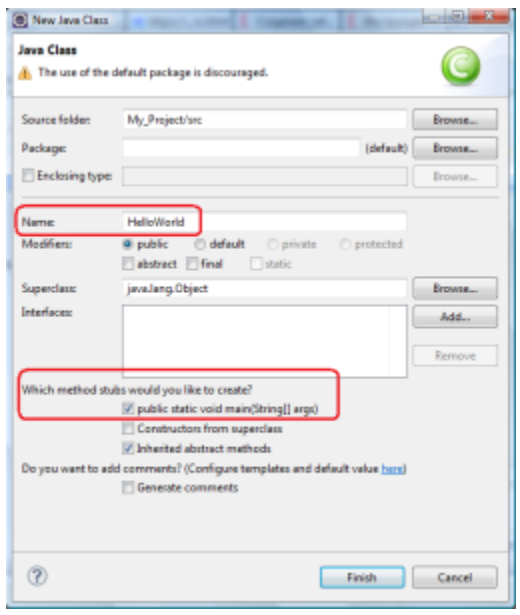

Рис. 2.8.

В итоге, Eclipse создаст новый класс **Hello World**

Откроем созданный класс и завершим нашу программу. Добавим в метод main следующий код (рис 2.9.).

```
System.out.println("Hello World");
```

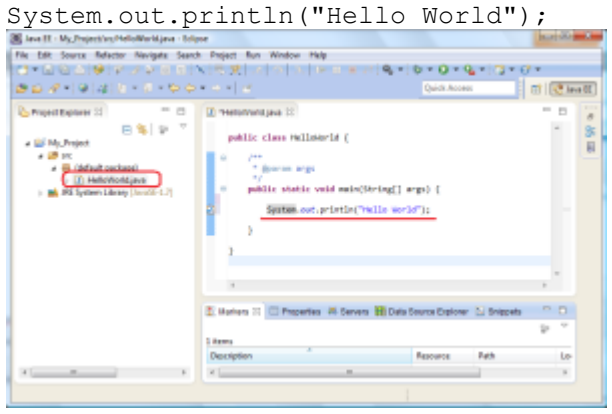

Рис. 3.9.

Сохраним изменения с при помощи клавиш Ctrl+S или специального значка вверху на панели инструментов. Готово!

Далее запустим наш проект, для этого в меню выберем **Run -> Run.**

В результате, в консоле будут напечатаны слова **Hello World** (рис. 3.10).

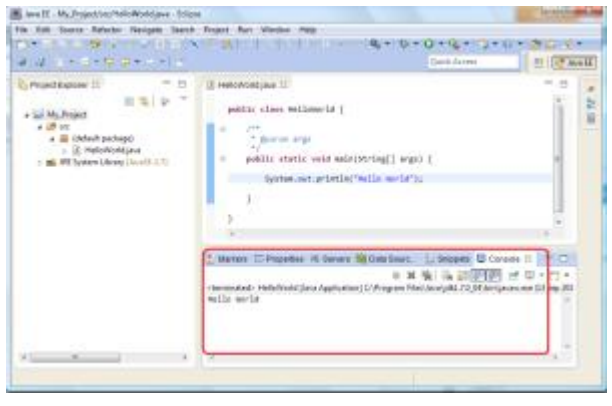

Рис. 3.10

Для запуска программы в дальнейшем, достаточно нажимать специальный значок на панели инструментов, выбрав **Hello World** (рис. 3.11.).

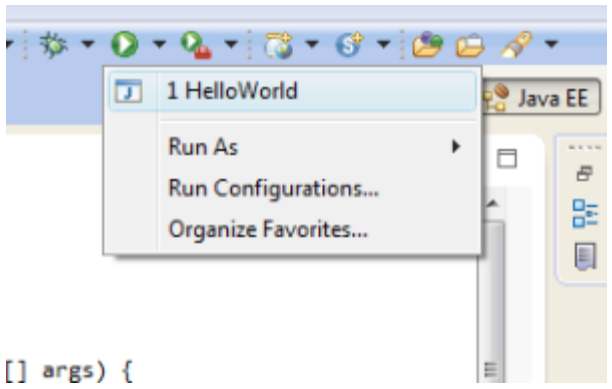

Рис. 3.11.

Для получения исполняемого JAR файла можно использовать команду меню File/Export/Java/Runnable JAR file, кнопка Next, Finish.Выполнить полученный JAR файл можно из командной строки с помощью команды ***java -jar one.jar.***

Структура Java-программы

Программа, написанная на языке Java, состоит из одного или нескольких файлов исходного кода (soursecode), которые представляют собой обычные текстовые файлы с расширением .java.

Соответственно для создания и редактирования файлов исходного кода можно использовать любой текстовый редактор.

Программы, написанные на языке Java, компилируются, но полученный код затем интерпретируется. Соответственно, язык Java является языком как компилируемого так и интерпретируемого типа: Java-программы компилируются компилятором в так называемый байтовый код bytecode (при этом получается промежуточный код, называемый еще “кодом виртуальной машины”, который не является особым для каждого типа процессора, но довольно близок к машинному языку), полученный байтовый код выполняется в режиме интерпретации с помощью интерпретатора на конкретной машине, преобразующего байтовый код уже в двоичный код процессора конкретной машины. Это позволяет обеспечить платформо-независимость программ, написанных на Java.

Все Java-программы содержат в себе классы (classes). Классы представляют собой основу объектно-ориентированных свойств языка. Классы содержат *переменные и методы*. Методы есть не что иное как функции или подпрограммы. В переменных хранятся данные.

Все классы являются производными (или подклассами) от существующих классов. В случае - если не определен **суперкласс,** то по умолчанию предполагается, что таким суперклассом является класс **Object.**

Пакеты содержат в себе классы и помогают компилятору найти те классы, которые нужны ему для компиляции пользовательской программы. Java-программа может содержать в себе любое количество классов, но один из них всегда имеет особый статус и непосредственно взаимодействует с оболочкой времени выполнения (**первичный класс**). Для приложений первичный класс должен обязательно содержать метод **main().**

Рассмотрим простейший пример программы на Java, которая позволяет вывести на экран приветствие “Hello world!”. Для того чтобы запустить эту программу вам придется ввести ее текст с помощью текстового редактора:

```
public class Hello
{
public static void main (String args[]){
System.out.println(“Hello world!”);
}
}
/*
Комментарий Java,
занимающий четыре строки
*/
//Комментарий вмещающийся в одну строку
```

Главным элементом в этой программе является конструкция, обозначенная ключевым словом class. Класс – это совокупность данных и методов, обрабатывающих данные.

В программе на Java должен присутствовать хотя бы один класс (в нашем случае это класс Hello). В Java просто не может быть программного кода вне класса. Общая форма задания класса Hello имеет следующий вид:

```
class Hello //ClassName
{
...
}
```

Ниже приведена общая форма исходного файла Java:

*одиночный оператор **package** (необязателен)*
*любое количество операторов **import** (необязательны)*
одиночное объявление открытого (public) класса
любое количество закрытых (private) классов пакета (необязательны)

Пакет (**package**) — это некий контейнер, который используется для того, чтобы изолировать имена классов. Например, вы можете создать класс List, заключить его в пакет и не думать после этого о возможных конфликтах, которые могли бы возникнуть если бы кто-нибудь еще создал класс с именем List. Если не использовать пакеты, то все идентификаторы, которые используются в программе, располагаются в одном и том же пространстве имен (namespace). Это означает, что нам во избежание конфликтных ситуаций нужно заботиться о том, чтобы у каждого класса было свое уникальное имя. Пакеты являются хорошим механизмом для отделения классов друг от друга, поэтому все встроенные в Java классы хранятся в пакетах. Пакеты — это механизм, который служит как для работы с пространством имен, так и для ограничения видимости.

Оператор import

После оператора package, но до любого определения классов в исходном Java-файле, может присутствовать список операторов import. Общая форма оператора import такова:

import пакет1 [.пакет2].(имя класса|*);

Здесь пакет1 — имя пакета верхнего уровня, пакет2 — это необязательное имя пакета, вложенного в первый пакет и отделенное точкой. И, наконец, после указания пути в иерархии пакетов, указывается либо имя класса, либо метасимвол звездочка. Звездочка означает, что, если Java-транслятору потребуется какой-либо класс, для которого пакет не указан явно, он должен просмотреть все содержимое пакета со звездочкой вместо имени класса. В приведенном ниже фрагменте кода показаны обе формы использования оператора import:

import java.util.Date
import java.io.*;

ЗАМЕЧАНИЕ
Но использовать без нужды форму записи оператора import с использованием звездочки не рекомендуется, т.к. это может значительно увеличить время трансляции кода (на скорость работы и размер программы это не влияет).

Все встроенные в Java-классы, которые входят в комплект поставки, хранятся в пакете с именем java. Базовые функции языка хранятся во вложенном пакете java.lang. Весь этот пакет автоматически импортируется транслятором во все программы. Это эквивалентно размещению в начале каждой программы оператора

import java.lang.*;

Если в двух пакетах, подключаемых с помощью формы оператора import со звездочкой, есть классы с одинаковыми именами, однако вы их не используете, транслятор не отреагирует. А вот при попытке использовать такой класс, вы сразу получите сообщение об ошибке, и вам придется переписать операторы import, чтобы явно указать, класс какого пакета вы имеете ввиду.

class MyDate extends Java.util.Date { }

В Java учитывается регистр символов, т.е., например, идентификаторы hello и Hello будут различаться. В современной практике программирования названия (идентификаторы) классов принято начинать с прописной (большой) буквы.

Так как класс Hello объявлен с модификатором public, то имя файла с его исходным кодом должно совпадать с именем класса. Для классов с модификатором по умолчанию имена файлов могут быть любыми (расширение обязательно .java).

Метод main объявлен в классе Hello в следующем виде:

```
public static void main(String args[]){
System.out.println(“Hello world!”);
}
```

Единственный оператор в этом методе представляет собой вызов метода println(). Метод println() выводит строку "Hello world!".
Символ ‘;’ в Java является разделителем операторов.
Выполнение любого приложения, написанного на Java, начинается с вызова метода main().
Приложения, написанные на языке Java, не являющиеся апплетами, должны иметь один и только один метод main().

Для ввода символов с клавиатуры используется, как правило, класс Scanner, который, в свою очередь использует System.in.

Описание языка JAVA

В языке программирования Java есть специальный термин, обозначающий его базовые элементы. Они называются *лексемами* (tokens).
Существует несколько типов лексем: *литералы, идентификаторы, ключевые слова, символы операций, разделители и комментарии.*

Идентификаторы(identifiers) предназначены для присвоения имен переменным, функциям и классам. При этом в идентификаторе могут использоваться любые комбинации букв, цифр, символов подчеркивания и символов доллара. За исключением следующих ограничений:

- первым символом идентификатора не может быть цифра (причем символы подчеркивания и доллара в начале идентификаторов используются компоновщиком java поэтому их лучше не ставить в начале идентификатора, это ограничение поможет вам при отладке)
- в качестве идентификаторов не могут быть использованы ключевые слова Java.

Большинство программистов при работе с Java используют следующее неформальное правило: имена классов записывать с прописной буквы, а имена переменных и функций со строчной буквы. Прописные буквы или символы подчеркивания используют для разделения слов в многословных именах, например, `secretPassword` или `some_thing`.
Ключевые слова (keywords) *– являются зарезервированными.* Это означает, что их нельзя использовать никаким другим способом, помимо того, для которого они предназначены спецификацией Java (поскольку эти слова зарезервированы их нельзя употреблять в качестве идентификаторов). Например, long, static, class, case, do и др.

Литералы(literal) – это число, символ, строка или специальная константа, например `true` или `false`, которые вы набираете прямо в тексте программы. Чаще всего литералы используются для инициализации объектов нужными значениями. Например, присвоить значение 10 целой переменной можно следующим образом:

```
int i=10; //Присвоение литерала 10 переменной i
```

Комментарии. В языке Java используются три вида комментариев.
Комментарии первого типа начинаются символами /* и заканчиваются символами */. Весь текст между данными последовательностями символов игнорируется компилятором. Это так называемый многострочный комментарий.
Комментарии второго типа располагаются до конца строки и отделяются от основного текста программы символами //. Это так называемый однострочный комментарий.
Кроме того, можно использовать документирующие комментарии: текст комментария находится между последовательностями символов /** и */ и непосредственно перед блоком кода, который необходимо задокументировать .

Переменные(variable) представляют собой имена области памяти, в которых находятся данные определенных типов, в зависимости от вида данных для них могут применяться переменные различных типов.

Типы данных переменных определяют внутреннее представление данных в вычислительных машинах; диапазон или множество значений, которые может принимать переменная, а также операции, в которых могут использоваться переменные определенного типа.
В Java существует два вида типов данных переменных:

- **примитивные типы** (primitive types). К ним относятся стандартные, встроенные в язык типы для представления численных значений, одиночных символов и булевских (логических) значений. Все примитивные типы имеют предопределенный размер занимаемой ими памяти.
- **ссылочные типы** (reference type) - относятся типы, определенные пользователем (классы, интерфейсы) и типы массивов. Все ссылочные типы являются динамическими типами, для них выделяется память во время работы программы.

Переменные примитивных типов передаются в методы по значению, тогда как ссылочные переменные всегда передаются по ссылке.

Примитивные типы. Всего в Java определено восемь примитивных типов: int (4b), short (2b), byte (1b), long (8b), float (4b), double (8b), boolean (true, false), char (2b).

Первые шесть типов предназначены для хранения численных значений, с ними можно производить арифметические операции. Тип char предназначен для хранения символов в стандарте Unicode (2 байта). Булевские (логические) переменные могут иметь одно из двух допустимых значений: true или false.

Ссылочные типы. Переменной ссылочного типа выделяется память при помощи оператора new. Таким образом, каждая переменная ссылочного типа является объектом или экземпляром соответствующего типа.

Примитивные типы данных

Ключевое слово	Тип	Размер	Значения, которые может хранить	Значение по умолчанию
byte	байт	8 бит	значения в диапазоне от -2^7 до 2^7-1	0
short	короткое целое	16 бит	значения в диапазоне от -2^{16} до $2^{16}-1$	0
int	целое	32 бит	значения в диапазоне от -2^{32} до $2^{32}-1$	0
long	длинное целое	64 бит	значения в диапазоне от -2^{64} до $2^{64}-1$	0
float	плавающая точка	32 бит	значения в диапазоне от$1.7*10^{-38}$ до $1.7*10^{38}$	0.0f
double	удвоенная точность	64 бит	значения в диапазоне от *-1.40239846E-45 до 3.40282347E+38*	0.0d
char	Символьный	16 бит	буквы, цифры, символы, основанные на 16-битном наборе символов Unicode от /u0000 до /uffff	'0x0'
boolean	логический (булевой)	8 бит	true либо false	false

В Java существует четыре ***целочисленных типа*** данных: `byte, short, int` и `long.` Как показано в вышеприведенной таблице, каждый из них позволяет работать с различным диапазоном чисел.
Вычисляя результат операции с целочисленными значениями Java подвергает их некоторым интересным преобразованиям. Например, пусть мы складываем переменную `SmallNumber` типа `byte` и переменную `LargeNumber` типа `int`. Обе переменные автоматически приобретают тип `int`, т.е. осуществляется преобразование типа меньшего размера к типу большего размера.
Типы данных с плавающей точкой
Тип `float` указывает, что переменная представляет собой число с плавающей точкой единичной точности длиной 32 бита. Тип `double` – это число с плавающей точкой двойной точноси длиной 64 бита.

Пример объявления переменной с плавающей точкой:

```
float SquareFootage;
double GrossNationalProduct;
```

Символьный тип данных
Тип `char` в действительности является 16-разрядным беззнаковым целым, которое представляет символ в кодировке unicode. Благодаря тому, что Java хранит все символы в формате Unicode, его можно использовать практически с любым существующим в мире письменным языком.
Логический тип данных
Переменные типа `boolean` могут иметь одно из двух значений – `true`(истина) или `false`(ложь).Переменные типа `boolean` можно использовать вместо выражений типа `boolean`, которые используются в операторах цикла и ветвлений. Обратите внимание на то, что слова `false` и `true` являются зарезервированными словами в Java и пишутся без кавычек).

Если программист явно не указал начальное значение переменной, Java инициализирует все примитивные типы данных значениями по умолчанию. Целочисленные переменные и переменные с плавающей точкой инициализируются значением 0. Тип char получает значение `null`, а тип `boolean` – значение `false`.

В приведенном ниже примере создаются переменные каждого из простых типов и выводятся значения этих переменных.

```
class SimpleTypes {
public static void main(String args []) {
byte b = 0x55;
short s = 0x55ff;
int i = 1000000;
long l = 0xffffffffL;
char c = ' a' ;
float f = .25f;
double d = .00001234;
boolean bool = true;
System.out.println("byte b = " + b);
System.out.println("short s = " +s);
System.out.println("int i = " + i);
System.out.println("long l = " + l);
System.out.println("char c = " + c);
System.out.println("float f = " + f);
System.out.println("double d = " + d);
System.out.println("boolean bool = " + bool);
} }
```

Запустив эту программу, вы должны получить результат, показанный ниже:

```
byte b = 85
short s = 22015
int i = 1000000
long l = 4294967295
char c = a
float f = 0.25
double d = 1.234e-005
boolean bool = true
```

Обратите внимание на то, что целые числа печатаются в десятичном представлении, хотя мы задавали значения некоторых из них в шестнадцатиричном формате.

Приведение типов

Иногда возникают ситуации, когда у вас есть величина какого-то определенного типа, а вам нужно ее присвоить переменной другого типа. Для некоторых типов это можно проделать и без приведения типа, в таких случаях говорят об автоматическом преобразовании типов. В Java автоматическое преобразование возможно только в том случае, когда точности представления чисел переменной-приемника достаточно для хранения исходного значения. Такое преобразование происходит, например, при занесении литеральной константы или значения переменной типа byte или short в переменную типа int. Это называется *расширением (widening)* или *повышением (promotion),* поскольку тип меньшей разрядности расширяется (повышается) до большего совместимого типа. Размера типа int всегда достаточно для хранения чисел из диапазона, допустимого для типа byte, поэтому в подобных ситуациях оператора явного приведения типа не требуется. Обратное в большинстве случаев неверно, поэтому для занесения значения типа int в переменную типа byte необходимо использовать оператор приведения типа. Эту процедуру иногда называют *сужением (narrowing),* поскольку вы явно сообщаете транслятору, что величину необходимо преобразовать, чтобы она уместилась в переменную нужного вам типа. Для приведения величины к определенному типу перед ней нужно указать этот тип, заключенный в круглые скобки. В приведенном ниже фрагменте кода демонстрируется приведение типа источника (переменной типа int) к типу приемника (переменной типа byte). Если бы при такой операции целое значение выходило за границы

допустимого для типа byte диапазона, оно было бы уменьшено путем деления по модулю на допустимый для byte диапазон (результат деления по модулю на число — это остаток от деления на это число).

```
int a = 100;
byte b = (byte) a;
```

Автоматическое преобразование типов в выражениях

Когда вы вычисляете значение выражения, точность, требуемая для хранения промежуточных результатов, зачастую должна быть выше, чем требуется для представления окончательного результата.

```
byte a = 40;
byte b = 50;
byte c = 100;
int d = a* b / c;
```

Результат промежуточного выражения (a* b) вполне может выйти за диапазон допустимых для типа byte значений. Именно поэтому Java автоматически повышает тип каждой части выражения до типа int, так что для промежуточного результата (a* b) хватает места.

Автоматическое преобразование типа иногда может оказаться причиной неожиданных сообщений транслятора об ошибках. Например, показанный ниже код, хотя и выглядит вполне корректным, приводит к сообщению об ошибке на фазе трансляции. В нем мы пытаемся записать значение 50* 2, которое должно прекрасно уместиться в тип byte, в байтовую переменную. Но из-за автоматического преобразования типа результата в int мы получаем сообщение об ошибке от транслятора — ведь при занесении int в byte может произойти потеря точности.

```
byte b = 50;
b = b* 2;
```

Исправленный текст :

```
byte b = 50;
b = (byte) (b* 2);
```

что приводит к занесению в b правильного значения 100.

Если в выражении используются переменные типов byte, short и int, то во избежание переполнения тип всего выражения автоматически повышается до int. Если же в выражении тип хотя бы одной переменной — long, то и тип всего выражения тоже повышается до long. Не забывайте, что все целые литералы, в конце которых не стоит символ L (или 1), имеют тип int.

Если выражение содержит операнды типа float, то и тип всего выражения автоматически повышается до float. Если же хотя бы один из операндов имеет тип double, то тип всего выражения повышается до double. По умолчанию Java рассматривает все литералы с плавающей точкой, как имеющие тип double.

Основные операторы

Символы операций (operators) – это символы, которые применяются для обозначения арифметических и логических операций. Арифметические символы определяют операции применяемые к числам (например, 2+3). Символы операций, за исключением знака плюс (+), используются только для арифметических вычислений. Операция + может также использоваться для сцепления строк.

Основные арифметические операторы

Оператор	Краткое описание	Пример
+	Сложение	3+2 или “pre”+”fix”
-	Вычитание	12-3
*	Умножение	length*width
/	Деление	miles/gallons
%	Оператор взятия модуля, определяет остаток от деления первого целочисленного операнда на второй	10%4

Булевы операции

Символ	Действие	Пример
==	равно	if (a==14)
!=	не равно	if (a!=14)
<	меньше чем	if (a<14)
>	больше чем	if (a>14)
<=	меньше или равно	if (a<=1)
>=	больше или равно	if (a>=4)
&	побитовое и	if ((a>14)&(a<17))
&&	и	if((a>14)&&(b>17))
|	побитовое или	if((a==16)|(b==19))
||	или	if ((a==25)||(a==8))
!	нет	if (!((a==16)||(a==3)))
^	исключающее или	if ((a==16)^(a==19))

Оператор присваивания

Оператор	Эквивалентный вид оператора
a+=b	a=a+b
a-=b	a=a-b
a*=b	a=a*b
a/=b	a=a/b
a%=b	a=a%b
a&=b	a=a&b
a|=b	a=a|b
a^=b	a=a^b
a<<=b	a=a<<b
a>>=b	a=a>>b

Приоритет и ассоциативность операций

Большинство операций имеет два операнда и возвращает одно значение (бинарные операторы). Обычно в записи математических формул допустимо использовать несколько операций в одной строке. Порядок вычислений в таком случае определяется группировкой операций в круглые скобки и приоритетом операций. Так, умножение более приоритетно, чем сложение, следовательно, выражение a+b*c эквивалентно a+(b*c). С арифметическими действиями приоритет очевиден, но бывает достаточно сложно вспомнить, что должно выполняться раньше, побитовый сдвиг или “логическое НЕ”. Поэтому в сомнительных случаях лучше всегда использовать круглые скобки. Отметим, что операции с одинаковым приоритетом выполняются в порядке их записи, слева направо.

Когда мы говорим о выполнении большинства операций слева направо, мы имеем в виду ассоциативность операций. Типичный пример операции, которая выполняется справа налево, это операции присваивания (=).

Утилиты

Библиотека классов языка включает в себя набор вспомогательных классов, широко используемых в других встроенных пакетах Java. Эти классы расположены в пакетах java.lang и java.util. Они используются для работы с наборов объектов, взаимодействия с системными функциями низкого уровня, для работы с математическими функциями, генерации случайных чисел и манипуляций с датами и временем.

Класс Math

Класс Math содержит функции с плавающей точкой, которые используются в геометрии и тригонометрии. Кроме того, в нем есть две константы, используемые в такого рода вычислениях: — E (приблизительно 2.72) и PI (приблизительно 3.14159).

Класс Math принадлежит пакету **`java.lang.`**

`Все` методы данного класса являются статическими, и создавать его подклассы и экземпляры нельзя. К статическим классам можно и нужно обращаться без указания имени объекта.

Тригонометрические функции

Приведенные ниже три функции имеют один параметр типа double, представляющий собой угол в радианах, и возвращают значение соответствующей тригонометрической функции.

- sin(double a) возвращает синус угла a, заданного в радианах.
- cos(double a) возвращает косинус угла a, заданного в радианах.
- tan(double a) возвращает тангенс угла a, заданного в радианах.

Следующие четыре функции возвращают угол в радианах, соответствующий значению, переданному им в качестве параметра.

- asin(double r) возвращает угол, синус которого равен r.
- acos(double r) возвращает угол, косинус которого равен r.
- atan(double r) возвращает угол, тангенс которого равен r.
- atan2(double a, double b) возвращает угол, тангенс которого равен отношению a/b.

Степенные, показательные и логарифмические функции

- pow(double y, double x) возвращает y, возведенное в степень x. Так, например, pow(2.0, 3.0) равно 8.0.
- exp(double x) возвращает e в степени x.
- log(double x) возвращает натуральный логарифм x.
- sqrt(double x) возвращает квадратный корень x.

Округление

- ceil(double a) возвращает наименьшее целое число, значение которого больше или равно a.
- floor(double a) возвращает наибольшее целое число, значение которого меньше или равно a.
- rint(double a) возвращает в типе double значение a с отброшенной дробной частью.
- round(float a) возвращает округленное до ближайшего целого значение a.
- round(double a) возвращает округленное до ближайшего длинного целого значение a.

Кроме того, в классе Math имеются методы для получения модуля, нахождения минимального и максимального значений, работающие с числами типов int, long, float и double:

- abs(a) возвращает модуль (абсолютное значение) a.
- max(a, b) возвращает наибольший из своих аргументов.
- min(a, b) возвращает наименьший из своих аргументов.

Пример использования функции ***max:***

```
c = Math.max(a, b);
```

Для выполнения округления используются также методы классов `BigDecimal и RoundingMode.` Ниже показан текст программы с использованием указанных методов:

```
import java.math.BigDecimal;
import java.math.RoundingMode;

public class my_round {

      public static void main(String[] args) {
            //Использование класса BigDecimal
              double templateDouble = 12.1354678578862;
              System.out.println("Template double: " + templateDouble);
              double newDouble = new BigDecimal(templateDouble).setScale(3, RoundingMode.UP).doubleValue();
              System.out.println("New double: " + newDouble);
            //Использование класса RoundingMode
              System.out.println(new BigDecimal(3.675f).setScale(2, RoundingMode.HALF_UP).floatValue());

      }
}
```

Задание 1.

Вычислить значение выражения:

$$t2=\frac{1}{2ab}\ln\frac{\sqrt{c^2-b^2}\,tg\,ax+2}{\sqrt{c^2-b^2}\,tg\,ax-2}$$

Алгоритм решения задачи – линейный и состоит из таких шагов:

- задание в программе типов данных и значений всех переменных:

```
a=12.5; b=1.3; c=14.1; d=2.7; x=3.3; y=1.1;
```

- вычисление значения t2;
- вывод значений t2 на экран.

Ответ: `t2= 0.020985550983192488` округлить до ближайшего целого числа.

Задание 2.

Заданы две булевы переменные *a = true* и *b = false.*

Выполнить над ними следующие операции: логическое И, логическое ИЛИ, логическое исключающее ИЛИ, логическое унарное отрицание. Результаты вывести на экран.

Задание 3.

Создать проект, который содержит два класса: первый содержит метод main, который выводит приветствие Hello world!, а второй - содержит метод **`public static void`** с именем **`ST`**, который выполняет вычисления по формуле, описанной ниже. Нужно вызвать метод **`ST`** из первого класса.

Описание метода ST.

Заданы следующие переменные: короткое целое ***`s = 1024;`*** целое ***`i = 50000;`*** число с плавающей точкой одинарной точности ***`f = 5.6;`*** число с плавающей точкой двойной точности ***`d =.1234, байтовое b = 42 и символьное c = 'a'.`***

Вычислить значение выражения ***`result = (f* b) + (i/ c) - (d* s).`*** Указать тип данных результата и вывести ответ на экран.