

Java. Лекция 2

Основы Java

Когда-либо настанет один из тех дней ...



Книги

- Герберт Шилдт. — **Java: методики программирования**
- Хорстманн К.С., Корнелл Г. — **Java (2 тома)**
- Герберт Шилдт. — **Java 8: руководство для начинающих**

Online

- <http://www.javasoft.com>
- Отличный электронный учебник!

Основы Java

- Идентификаторы (Identifiers)
- Комментарии (Comments)
- Ключевые слова (Keywords)
- Типы данных (Data types)
- Выражения и операторы (Expressions and operators)
- Приоритет операторов (Operator precedence)
- Оператор присваивания (The assignment statement)
- Организация кода (Organizing code)
- Операторы выбора (Selection statements)
- Итерационные операторы (Iteration statements)

Структура

- Java «программы» состоят из файлов, содержащих классы.
- Классы состоят из объявления переменных и блоков кода, которые будем называть методами (и конструкторами)
- Если у вас есть опыт программирования на других языках, то изучение Java может показаться довольно простым занятием, **но будьте внимательны!**

Идентификаторы (Identifiers)

- Используются для именованя
 - Переменных (Variables)
 - Методов (Methods)
 - Классов (Classes)
- Чувствительные к регистру символов (CaSe SeNsItIvE)
- Правила
 - Первый символ должен быть символом подчеркивания или латинской буквой
 - Последующий символы могут быть символами подчеркивания, латинской буквой или цифрой
 - Любой длинны!

Идентификаторы (Identifiers)

- **Стиль**
 - Название класса начинается с заглавной буквы
 - Название переменных и методов начинается с малого прописного символа
 - Константы содержат все заглавные символы

- **Примеры**

```
class Widget
```

```
class LinkedList
```

```
Variables: x, n pressure, isEmpty,  
           hasMore, steamTemp
```

```
Constants: FEETPERMILE, HOURS_PER_WEEK
```


Комментарии (Comments)

// до конца строки

/* Это пример многострочного
комментария, который занимает

* столько строчек, сколько

* может уместиться между

* подряд идущими символами

* начала комментария (/ *) и

* конца комментария (* /)

*

// Конец многострочного комментария -->>> */

Пример

```
/*
```

```
Comments about this section  
yada - yada - yada
```

```
*/
```

```
x = 3;    // Here is some  
y = 4;    // info about  
z = 5;    // these lines
```

Пример

```
/*  
    Comments about this section  
    yada - yada - yada  
// (note)                                     */  
x = 3;    // Here is some  
y = 4;    // info about  
z = 5;    // these lines
```

Пример

```
/*
```

```
Comments about this section
```

```
yada - yada - yada
```

```
x = 3;    // Here is some
```

```
y = 4;    // info about
```

```
z = 5;    // these lines
```

Ключевые слова (Keywords)

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

Типы данных (Data types)

- Что такое типы данных?
- Для чего они нужны?
- Чем они хороши?
- Чем они плохи?

- Java считается строго типизированным языком.

Примитивные типы данных Java

- **boolean** – true (истина) / false (ложь)

- byte

- short

- **int**

- long

Целые числа (Whole Numbers)

- Float

- **double**

Дробные числа (Fractional Numbers)

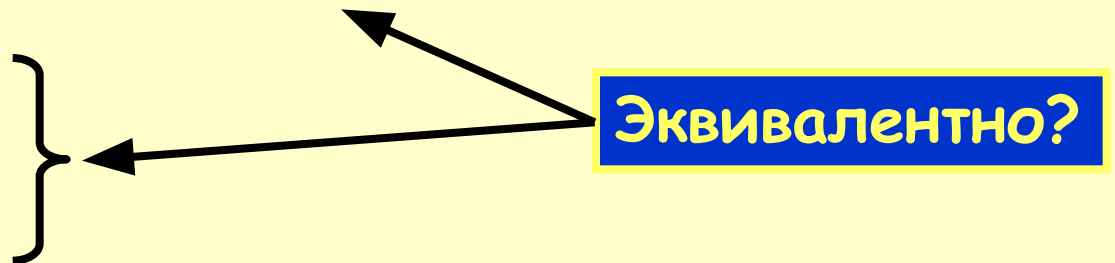
- **char** -- Обычно используется для печатаемых символов

Дианазон значений примитивных типов данных

Type	Size	Min	Max	Default
boolean	1	false	true	false
char	16			'\u0000' (null)
byte	8	-128	127	(byte) 0
short	16	-32,768	32,767	(short) 0
int	32	-2,147,483,648	2,147,483,647	0
long	64	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	0L
float	32	Approx $\pm 3.4E+38$ with 7 significant digits		0.0F
double	64	Approx $\pm 1.7E+308$ with 15 significant digits		0.0D
void				

Объявления (Declarations)

```
int i;  
int a, b, c;  
double d = 42.0;  
float f = 38.0; /* ОШИБКА (ERROR) */  
float f = 38.0F;  
char ch1;  
char ch2 = 's';  
boolean isFull = true;  
boolean isFull;  
isFull = true;
```



Ссылки (References)

- Переменные содержат «ссылку» («указатель») на объект
- Это можно представить, как адрес объекта в памяти
- Манипуляции адресами (наподобие C) отсутствуют
 - Таким образом, отсутствует необходимость в получении адреса и наличии операторов разыменования
- **<Тип (название некоторого класса)> <Идентификатор>;**

String name;

Queue myQueue;

CokeMachine cc;

Можно также осуществлять инициализацию в этой же строке

Значимые и ссылочные типы данных

- Д/З Куча, стек, значимые и ссылочные типы данных. Java compiler

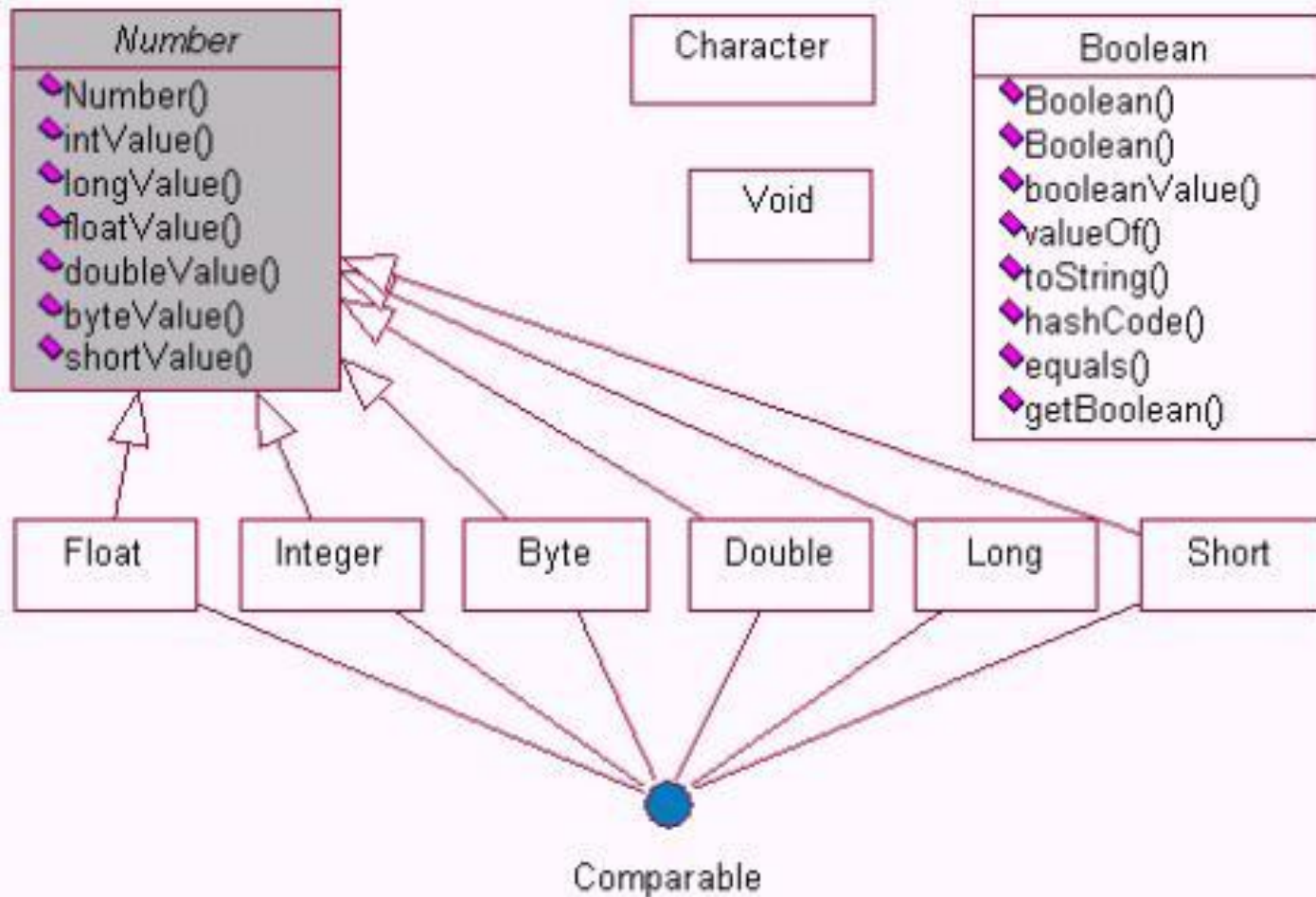
Что такое null?

- Если осуществить декларирование переменных
 - `Person john;`
 - `String name;`
- ... и не присвоить им значение
 - `john = new Person();`
 - `String name = "John Smith";`
- ... то за исключением значимых типов данных, все переменные (ссылочные) будут иметь значение `null`
- `null` является приемлемым значением переменной, однако количество действий, которых можно сделать с ним, ограничено:
 - It's an error to refer to its fields, because it has none
 - It's an error to send a message to it, because it has no methods
 - The error you will see is `NullPointerException`

Классы оболочки

- Кроме базовых типов данных широко используются соответствующие классы (wrapper классы: Boolean, Character, Integer, Byte, Short, Long, Float, Double. Объекты этих классов могут хранить те же значения, что и соответствующие им базовые типы.
- Объекты этих классов представляют ссылки на участки динамической памяти, в которой хранятся их значения и являются классами-оболочками для значений базовых типов. Указанные классы находятся в библиотеке **java/lang**, являются наследниками абстрактного класса **Number** и реализуют интерфейс **Comparable**, представляющий собой интерфейс для работы со всеми скалярными типами.
- Объекты классов-оболочек не могут принимать участия в арифметических операциях и по умолчанию получают значение **null**.

Классы оболочки



Классы оболочки

- Статический метод **valueOf(String s)** класса **Number** переопределен для всех подклассов, соответствующих примитивным типам и выполняет действия по преобразованию значения, заданного в виде строки к значению соответствующего объектного типа данных.
- Методы вида **byteValue()**, **intValue()** и др., определенные во всех объектных типах, преобразуют значение объектного типа данных к базовому типу. Определены для всех объектных типов.
- Статические методы **getNumericValue(char ch)** или **digit(char ch, int radix)** преобразуют символ к соответствующему целочисленному значению в заданной системе счисления.
- Java включает два класса для работы с высокоточной арифметикой: **BigInteger** и **BigDecimal**, которые поддерживают целые числа и числа с фиксированной точкой произвольной точности.
- Строка в Java представляет объект класса **String**. При работе со строками можно использовать перегружаемую операцию "+" объединения строк, а также методы класса **String**. Строковые константы заключаются в двойные кавычки и не заканчиваются символом '\0', это не ASCII-строки, а массивы символов.

Классы оболочки в J2SE 5.0

- Серьезных различий между базовыми типами и классами оболочками стало меньше. Теперь разрешено участие объектов в арифметических операциях, в том числе с участием базовых типов:

```
Integer j = new Integer(1);
```

```
Integer k = ++j;
```

```
int i = 2;
```

```
k = i + j + k;
```

- Однако следующий код генерирует исключительную ситуацию **NullPointerException** при попытке присвоить базовому типу значение **null** объекта класса **Integer**.

```
class NewProperties {  
    static Integer i = null;  
    public static void main(String[] args) {  
        int j = i; //генерация исключения  
    }  
}
```


Классы оболочки в J2SE 5.0

- Несмотря на то, что значения базовых типов могут быть присвоены объектам классов оболочек, сравнение объектов между собой происходит по ссылкам.

```
int i = 7;
```

```
Integer oa = i;
```

```
Integer ob = i;
```

```
System.out.print("oa == i" + (oa == i));//true
```

```
System.out.print("ob == i" + (ob == i));//true
```

```
System.out.print("oa == ob" + (oa == ob));//false
```

```
System.out.print("oa.equals(i)" + oa.equals(i));//true
```

- Значение базового типа может быть передано в метод **equals()**. Однако ссылка на базовый тип не может вызывать методы:

```
boolean b = i.equals(oa);//ошибка компиляции
```

- При инициализации объекта класса оболочки значением базового типа преобразование типов необходимо указывать явно, то есть код

```
Float f = 1;
```

- вызывает ошибку компиляции.

Константы (Constants)

```
public static final int FEETPERMILE = 5280;  
public static final String MY_GT_NUM="gt1234x";  
public static final boolean DEBUG = true;  
public static final int JAN = 1;  
public static final int FEB = 2;  
public static final int MAR = 3;  
и т.д.
```

```
if (DEBUG)  
{  
    /* Do Something */  
}
```

Строки (Strings)

- Строки – это объекты

```
String someStr = "This is a string";  
someStr = "Strings can be " + "concatenated.";  
someStr = "If they won't fit on one line " +  
          "you need to do it like this";  
someStr = "They can be printed";  
System.out.println(someStr);
```

Выражения (Expressions)

- В Java используется инфиксная нотация

`(x * x + y * y < 25) && (x > 0)`

- в противоположность префиксной нотации (Lisp)
(прямая польская нотация)

`(and (< 25 (+ (* x x) (* y y))) (> x 0))`

- и постфиксной нотации (программируемые калькуляторы)
(обратная польская нотация)

`((25 ((x x *) (y y *) +) <) (x 0 >) and)`

Операторы (Operators)

Арифметические:	+ - * / %
Отношения:	> < == <= >= !=
Логические:	&& !
Присвоение	=
	+= -= *= /= &= =
	^= %= <<= >>= >>>=
Условный:	? :
Побитовые:	& ~ ^ << >> >>>
Inc/Dec:	++ --

Операторы (Operators)

Арифметические:	+ - * / %
Отношения:	> < == <= >= !=
Логические:	&& !
Присвоение	=
	+= -= *= /= &= =
	^= %= <<= >>= >>>=
Условный:	? :
Побитовые:	& ~ ^ << >> >>>
Inc/Dec:	++ --

Приоритет операторов

$1 + 2 * 3$

- Когда возникает сомнение по поводу расстановки скобок?

В следующей таблице показан приоритет операторов. Операторы перечислены в порядке повышения их приоритета: чем выше строчка в таблице, тем выше приоритет. Операторы с более высоким приоритетом выполняются раньше операторов с относительно более низким приоритетом. Расположенные на одной строке операторы имеют одинаковый приоритет.

postfix operators	[] . (params) expr++ expr--
unary operators	++expr --expr +expr -expr ~ !
creation or cast	new (type)expr
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	
logical AND	&&
logical OR	
conditional	? :
assignment	= += -= *= /= %= &= ^= = <<= >>= >>>=

Когда операторы с одинаковым приоритетом появляются в одном и том же выражении, правила должны определять, которой из операторов будет исполнен в первую очередь. Все бинарные операторы, за исключением операторов присваивания, выполняются слева направо, а операторы присвоения – справа налево.

Оператор присваивания

$x = 1 + 1;$

- Вычислить выражение справа
- Поместить результат в переменную слева
- Левая сторона должна быть способна хранить значения
- $3 = x; //$ **Неправильно!**
- Любая операция должна заканчиваться;
- Возможно?
 $a = b = c = 0;$
- Это хороший подход?
- $=$ не $==$

Классическая двойственность

- Языки программирования всегда боролись с разницей между операцией присвоения и эквивалентности.

- **Equality (Boolean Result)**

- BASIC $A = B$
- Pascal $A = B$
- FORTRAN $A .EQ. B$
- C $A == B$
- Pseudocode $A = B$
- Scheme $(= A B)$
- Java $A == B$

- **Assignment**

- BASIC $LET A = B$
- Pascal $A := B$
- FORTRAN $A = B$
- C $A = B$
- Pseudocode $A \leftarrow B$
- Scheme $(set! A B)$
- Java $A = B$

Распространенные ошибки

- Неправильное использование ++ или -- (префиксная или постфиксная форма записи)
- Неправильное использование = и ==
- Использование
 & вместо &&
 | вместо ||

Предостережение!

- Просто, потому что язык позволяет сделать что-то подобное, ни сколько не означает, что это хорошая идея

● `_ += _ -- - ++_;`

`/* Ясно? */`

Организация кода

- Блоки (Block statements)

- Блоки кода окружены фигурными скобками

```
{  
    int i;  
    x = 1;  
    y = 2;  
    i = x + y;  
}
```

- Можно использовать в качестве «единого» оператора
- Примечание: переменные, объявленные в блоке, видимы и существуют только в блоке!

Организация кода

- Пустое выражение
 - Просто одна точка с запятой ;
 - Нет необходимости его использовать!

Операторы выбора (Selection statements)

- Используется для управления потоком выполнения программы
- If
- If с Else
- Сложные конструкции if
- Switch

if

```
if (<boolean_value>)
```

```
    <Executed if boolean is true>
```

```
<Executed in any case>
```

```
if (x > 0)
```

```
    System.out.println("X greater than zero!");
```


if

```
if (y > 0) ;  
    System.out.println("Always gets printed!");
```

```
if (z > 0 && isEmpty)  
{  
    x = y;  
    System.out.println("Hello");  
}
```

if/else

```
if(<boolean_value>
    <Executed if boolean is true>
else
    <Executed if boolean is true>
<Executed in all cases>
```

```
if(a == b)
{
    System.out.println("Always use curlies");
}
else
{
    System.out.println("What do you think?");
}
```

if/else – другой стиль

```
if(a == b) {  
    System.out.println("Always use curlies");  
} else {  
    System.out.println("What do you think?");  
}
```

Complex

```
if(a == b) {
    System.out.println("A equals B");
} else {
    if(c == d) {
        System.out.println("C equals D");
    } else {
        if(e == f) {
            System.out.println("E equals F");
        } else {
            System.out.println("None");
        }
    }
}
```

Complex

```
if(a == b)
    System.out.println("A equals B");
else if(c == d)
    System.out.println("C equals D");
else if(e == f)
    System.out.println("E equals F");
else
    System.out.println("None");
```

Complex

```
if(a == b)
{
    System.out.println("A equals B");
}
else if(c == d)
{
    System.out.println("C equals D");
}
else if(e == f)
{
    System.out.println("E equals F");
}
else
{
    System.out.println("None");
}
```

Сложные выражения условий



```
if(a == b)
    System.out.println("A equals B");
else if(c == d)
    System.out.println("C equals D");
else if(e == f)
    System.out.println("E equals F");
else
    System.out.println("None");
```

Сложные выражения условий

```
if(a == b)
    System.out.println("A equals B");
else if(c == d)
    System.out.println("C equals D");
else if(e == f)
    System.out.println("E equals F");
else
    System.out.println("None");
```


Сложные выражения условий

```
if(a == b)
    System.out.println("A equals B");
else if(c == d)
    System.out.println("C equals D");
else if(e == f)
    System.out.println("E equals F");
else
    System.out.println("None");
```



Вложенные условия

```
if(a == b) {  
    if(c == d) {  
        /* do something here */  
    } else {  
        /* do something else */  
    }  
} else {  
    if(x == y) {  
        /* what to do? */  
    } else {  
        /* another option */  
    }  
}
```

Вложенные условия

```
if(a == b) {  
    if(c == d) {  
        /* do something here */  
    } else {  
        /* do something else */  
    }  
} else {  
    if(x == y) {  
        /* what to do? */  
    } else {  
        /* another option */  
    }  
}
```

Switch

- Используется, когда существует единственное значение для тестирования
- в основном `int`
- и множество "cases« (случаев)
- Распространенная ошибка: забывают **break**

Switch

```
switch (month) {  
    case APR:  
    case JUN:  
    case SEP:  
    case NOV:  
        numDays = 30;  
        break;  
    case FEB:  
        if(year % 4 == 0)  
            numdays = 29;  
        else  
            numdays = 28;  
        break;  
    default:  
        numdays = 31;  
}
```

Какие еще возможны ошибки?

Итерационные операторы

- Цикл `while`
 - Проверка вначале
 - Выполняется 0 и более раз
- Цикл `do while`
 - Проверка вконце
 - Выполняется 1 и более раз
- Цикл `for`
 - Эквивалентен циклу `while`
 - Очень популярный цикл!

Цикл While


```
while (<boolean_value>)  
    <statement>
```

Пример:

```
i = 0;  
while (i < 10)  
    i++;
```

Цикл While

```
int i = 1;  
int total = 0;  
while(i <= 10)  
{  
    total += i;  
    i++;  
}
```



Примечание: отсутствует ;

Цикл Do While

```
do  
    <statement>  
while (<boolean>) ;
```

Цикл Do While

```
int choice;  
do  
{  
    choice = menu();  
    switch(choice)  
    {  
        /* details omitted */  
    }  
} while(choice != 0);
```

Примечание



Цикл For

```
for (<init>; <testExpr>; <increment>)  
    <statement>
```

<init> и <increment> может быть несколько,
разделенные запятой

<init>, <testExpr> и <increment> являются
необязательные!

```
for (; ; ) // Правильное выражение!
```

Цикл For

```
for(i=0; i < MAX; i++)  
    System.out.println(i);
```

```
for(i=0; i < MAX; i++)  
{  
    System.out.println(i);  
}  
System.out.println(i);
```

```
for(int i=0; i <= MAX; i++)  
{  
    System.out.println(i);  
}  
System.out.println(i);
```

Цикл For

```
for(i=0, total=0; i < MAX; i++)  
    System.out.println(i);
```

```
for(i=0; i < MAX && something; i++);  
{  
    System.out.println(i);  
}
```

```
for(int i=0, j=10; i <= MAX; i++,j--)  
{  
    System.out.println("Sum = " + i + j);  
}
```

while – эквивалент for

```
total = 0;
i = 0
while(i < MAX)
{
    /* Work */
    total += i;
    i++;
}
```

```
total = 0;
for(i = 0; i < MAX; i++)
{
    /* Work */
    total += i;
}
```

Практические примеры

Ввод строки из потока ввода, связанного с консолью

```
import java.io.*;
public class InputStr {
public static void main(String[] args) {
/* байтовый поток ввода System.in передается конструктору при создании объекта класса
InputStreamReader */
InputStreamReader is = new InputStreamReader (System.in);
/* производится буферизация данных, исключая необходимость обращения к источнику данных при
выполнении операции чтения */
BufferedReader bis = new BufferedReader(is);
try {
System.out.println("Введите Ваше имя и нажмите <Enter>:");
/*чтение строки из буфера; метод readLine() требует обработки возможной ошибки
при вводе с консоли в блоке try */
String name = bis.readLine();
System.out.println("Привет, " + name);
} catch (IOException e) {
System.out.print("ошибка ввода " + e);
}}
```

- **JDeveloper:** Go to project properties, click on Run/Debug settings, select your Run configuration (Default) and Click Edit. Locate "Tool settings" and check "Allow program input" in Additional Runner Options. Save settings and run your program again. Now you will see under the console input field for your inputs.

Задание

- Написать приложение, выводящее на консоль N раз строчку
- «This is my first program written in Java programming language!»
- Значение N задается с консоли. Перед выводом очередной строчки вычисляется случайное значение, лежащее в диапазоне от 0 до 2. Если оно больше 0.7, то программа досрочно завершает свою работу.

Типы данных и операции над ними

```
public class TypeByte {
    private static int j;
    public static void main(String[] args) {
        int i = 3;
        byte b = 1,
        b1 = 1 + 2;
        //b = b1 + 1; //ошибка приведения типов
        b = (byte)(b1 + 1); //0
        show(b);
        //b = -b; //ошибка приведения типов
        b = (byte)-b; //1
        show(b);
        //b = +b1; //ошибка приведения типов
        b = (byte)+b1; //2
        show(b);
        b1 *= 2; //3
        show(b1);
        b1++; //4
        show(b1);
        //b = i; //ошибка приведения типов
        b = (byte)i; //5
        show(b);
        b += i++; //работает!!! //6
        show(b);
        float f = 1.1f;
```

Преобразование типов данных

```
public class Types {  
    public static void main(String[] args) {  
        Float f = new Float(10.01); //double в Float  
        String s1 = Float.toString(0f); //float в String  
        String s2 = String.valueOf(f); //Float в String  
        Byte b = Byte.valueOf("120"); //String в Byte  
        double d = b.doubleValue(); // Byte в double  
        short s = (short) d; // double в short  
        Character ch = new Character('3');  
        /*Character в int */  
        int i = Character.digit(ch.charValue(), 10);  
        System.out.println("s1=" + s1 + ", s2=" + s2);  
        System.out.print("b=" + b + ", s=" + s + ", d=" + d + ", i=" + i);  
    }  
}
```

Выход за цикл, помеченный OUT

- Расширение возможностей получили оператор прерывания цикла **break** и оператор прерывания итерации цикла **continue**, которые можно использовать с меткой, для обеспечения выхода из вложенных циклов, например:

```
public class DemoLabel {
    public static void main(String[] a) {
        int j = -3;
        OUT: while (j < 10) {
            if (j == 0)
                break OUT;
            else {
                j++;
                System.out.println(j);
            }
        }
        System.out.println("end");
    }
}
```