

Талипов С.Н.

Визуальное программирование на Java Swing

в NetBeans

Сергей Талипов

**Визуальное программирование  
на Java Swing в NetBeans**

«ЛитРес: Самиздат»

2019

**Талипов С. Н.**

Визуальное программирование на Java Swing в NetBeans /  
С. Н. Талипов — «ЛитРес: Самиздат», 2019

В данном пособии приведен необходимый учебный материал для изучения основ программирования на Java в среде разработки NetBeans. Особенность пособия заключается в его направленности на быструю начальную разработку программ с графическим интерфейсом, что позволяет в дальнейшем без проблем изучать и осваивать более сложный теоретический материал и технологии программирования на Java.

# Содержание

1 Основные сведения	6
1.1 Особенности Java	7
1.2 Среда разработки приложений NetBeans	11
1.3 Простые типы данных	13
1.4. Управляющие конструкции	20
1.5 Одномерные статические массивы	27
1.6 Многомерные статические массивы	28
1.7 Динамические массивы-списки	30
1.8 Работа со строками	32
2 Простейшие программы	38
2.1 Консольные программы	39
2.2 Простейшая программа с графическим интерфейсом в среде NetBeans	45
Конец ознакомительного фрагмента.	56

**Сергей Талипов**  
**Визуальное программирование**  
**на Java Swing в NetBeans**

# **1 Основные сведения**

## 1.1 Особенности Java

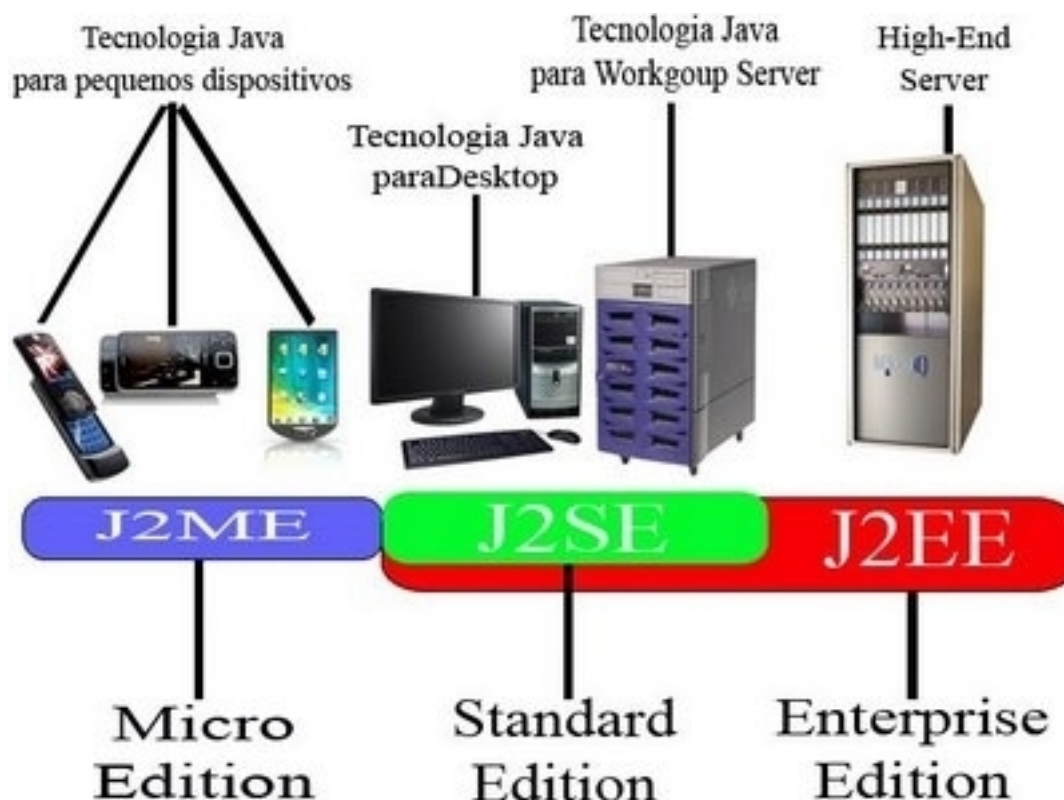
Java (произносится Джава; иногда – Ява) – объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle).

Приложения Java компилируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине (JVM) вне зависимости от компьютерной архитектуры. Дата официального выпуска – 23 мая 1995 года.

Эмблемой Java является чашечка с кофе.



Язык программирования Java произошел от языка «ОАК», что в переводе означает «Дуб». После своего появления язык Java начал развиваться по нескольким направлениям:



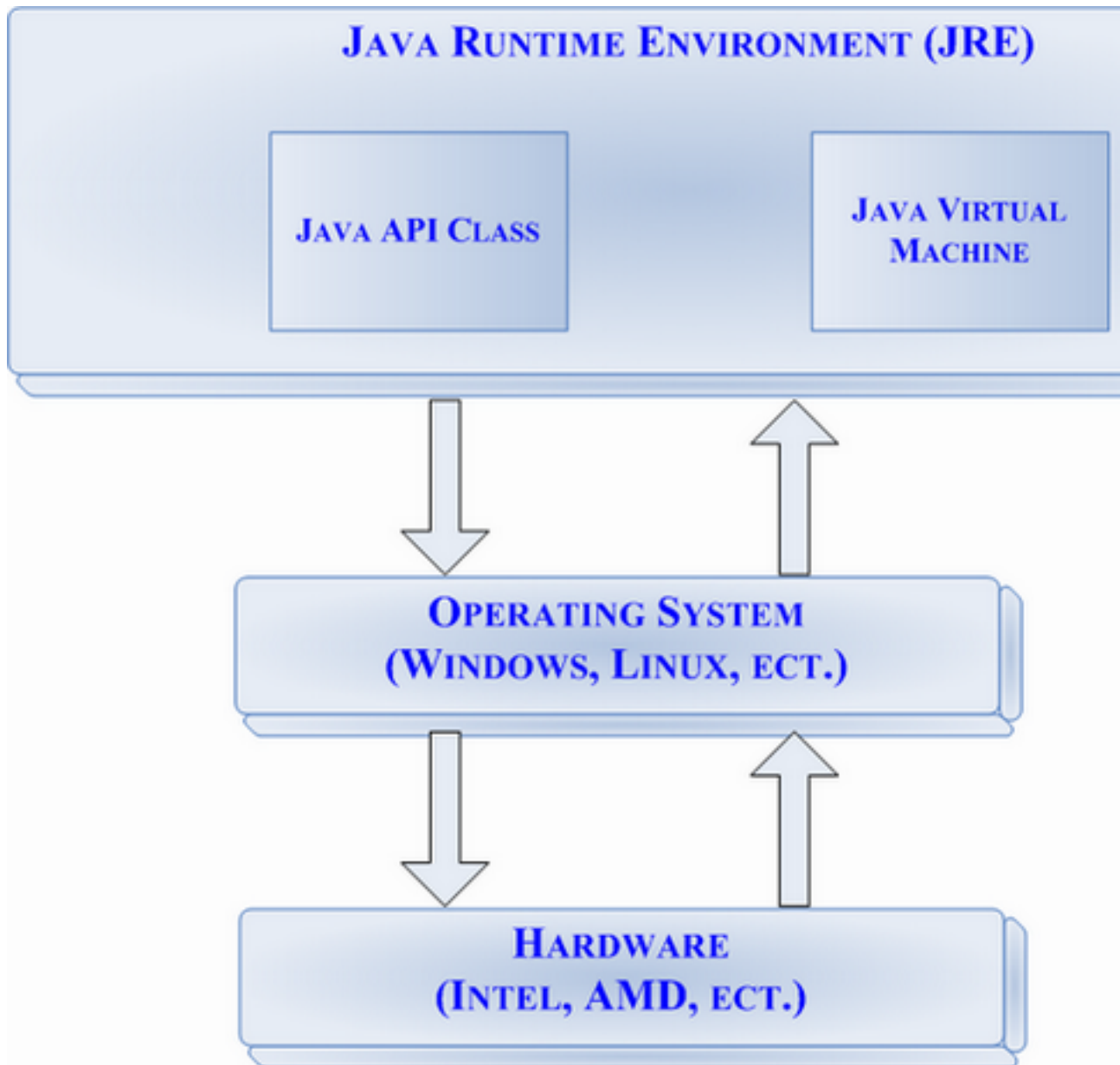
**Java 2 Micro Edition**, сокращенно J2ME – это редакция языка Java для разработки приложений для микрокомпьютеров (мобильных телефонов, Palm и т.д.). Сейчас имеет популярность в связи с развитием мобильных микропроцессорных устройств. В нее входят "облегченные" стандартные классы и классы для написания мидлетов (Midlets). Мидлеты специально разрабатываются для небольших устройств, в них поддерживается графика, звук, реакция на события (нажатие кнопок и т.д.). Java ME наиболее полно соответствует начальному предназначению Java – платформы для написания программ для бытовых устройств.

**Java 2 Standart Edition**, сокращенно J2SE – это стандартная редакция языка Java, используемая для разработки обычных Java приложений. Используя данную редакцию можно создавать консольные приложения и приложения с графическим интерфейсом пользователя. Часто встречается аббревиатура J2SE, которая подразумевает Java 2 Standart Edition.

**Java 2 Enterprise Edition**, сокращенно J2EE – это редакция языка Java для разработки распределенных приложений масштаба предприятия (корпоративных приложений). Данная редакция включает в себя технологию Enterprise Java Beans (EJB), Java Server Pages (JSP) и сервлеты (Servlets). На данный момент J2EE и .Net сейчас два основных соперника на рынке решений для разработки корпоративных приложений.

Механизм исполнения программ на Java включает в себя виртуальную машину Java, операционную систему и аппаратное обеспечение:





**Java Runtime Environment**, сокращенно JRE – это исполнительная среда Java, в которой выполняются программы, написанные на этом языке. Среда состоит из виртуальной машины – Java Virtual Machine (JVM) и библиотеки Java классов. По сути это минимальная реализация виртуальной машины, необходимая для исполнения Java приложений, без компилятора и других средств разработки.

**Java Virtual Machine**, сокращенно JVM – это виртуальная машина Java – основная часть исполняющей среды JRE. Виртуальная машина Java интерпретирует и исполняет байт-код Java. Байт код получают посредством компиляции исходного кода программы с помощью компилятора Java (стандартный – `javac`). В отличие от классических runtime-библиотек, библиотеки Java-классов входят в состав JRE.

**Java Development Kit**, сокращенно JDK – это бесплатно распространяемый корпорацией Oracle комплект разработчика приложений на языке Java, включающий в себя компилятор Java (`javac`), стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему Java (JRE). В состав JDK не входит интегрированная среда разработки на Java (IDE), поэтому разработчик, использующий только JDK, вынужден использовать

внешний текстовый редактор и компилировать свои программы, используя утилиты командной строки.

Несмотря на то, что JRE входит в состав JDK, фирма Oracle распространяет этот набор и отдельным файлом. Это вызвано тем, что установка JRE является необходимым и достаточным условием для выполнения Java-программ. Однако для разработки программ JRE недостаточно, необходимо установить пакет JDK, который может установить и JRE и дополнительные компоненты.



**Современные интегрированные среды разработки**, такие как NetBeans, Oracle JDeveloper, IntelliJ IDEA, Eclipse служат для удобной разработки программного обеспечения на Java. Они опираются на сервисы, предоставляемые JDK, и вызывают для компиляции Java-программ компилятор командной строки из комплекта JDK. Поэтому эти среды разработки либо включают в комплект поставки одну из версий JDK, либо требуют для своей работы предварительной установки JDK на машине разработчика.

Таким образом, для разработки программ на Java достаточно установить JRE+JDK +NetBeans, а только лишь для запуска готовой программы на машине пользователя достаточно установить одну JRE.

Запуск готовых java-программ (с расширением jar) из командной строки производят так:

```
java -jar JavaApplication1.jar
```

В данном примере запускается на выполнение программа JavaApplication1.jar.

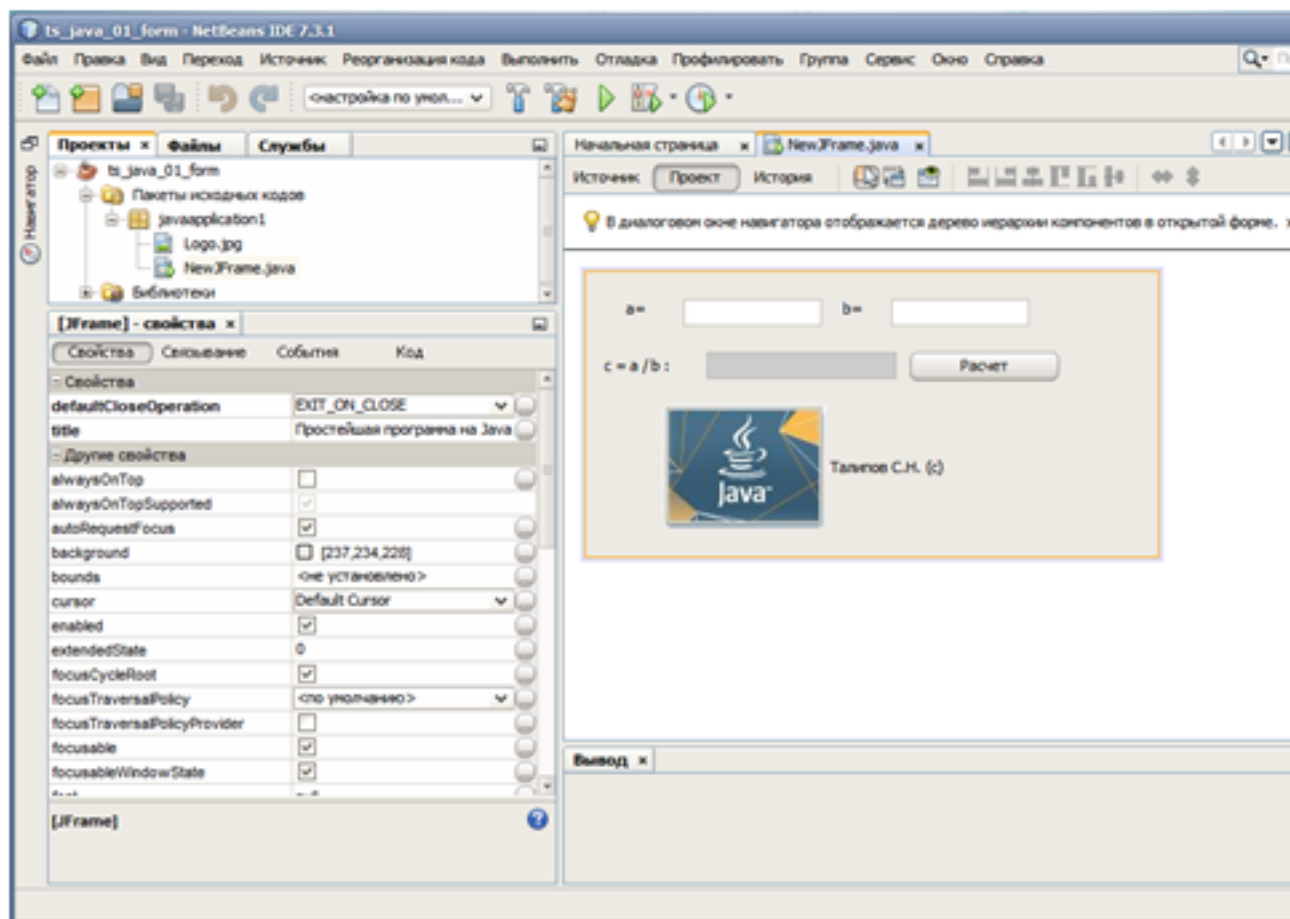
## 1.2 Среда разработки приложений NetBeans

NetBeans IDE – свободная интегрированная среда разработки приложений (IDE) на языках программирования Java, Python, PHP, JavaScript, C, C++, и ряда других.

Проект NetBeans IDE поддерживается и спонсируется компанией Oracle, однако разработка NetBeans ведется независимым сообществом разработчиков-энтузиастов (NetBeans Community) и компанией NetBeans Org.

По качеству и возможностям последние версии NetBeans IDE не уступают лучшим коммерческим (платным) интегрированным средам разработки для языка Java, таким, как IntelliJ IDEA, поддерживая рефакторинг, профилирование, выделение синтаксических конструкций цветом, автодополнение набираемых конструкций на лету, множество predefined шаблонов кода и др.

Для разработки программ в среде NetBeans и для успешной инсталляции и работы самой среды NetBeans должен быть предварительно установлен JDK или J2EE SDK подходящей версии. Среда разработки NetBeans поддерживает разработку для платформ J2SE и J2EE и для мобильных платформ J2ME.



**Рабочая папка проектов по-умолчанию:**

C:\Documents and Settings\<ИМЯ\_ПОЛЬЗОВАТЕЛЯ>\Мои документы\NetBeansProjects.

**Запуск программы на выполнение:** Заходим в меню "Выполнить" – "Запустить проект" (F6).

**Заккрытие проекта:** Заходим в меню "Файл" – "Заккрыть проект" (Для закрытия необходимо предварительно выделить проект мышью в окне «Проекты»).

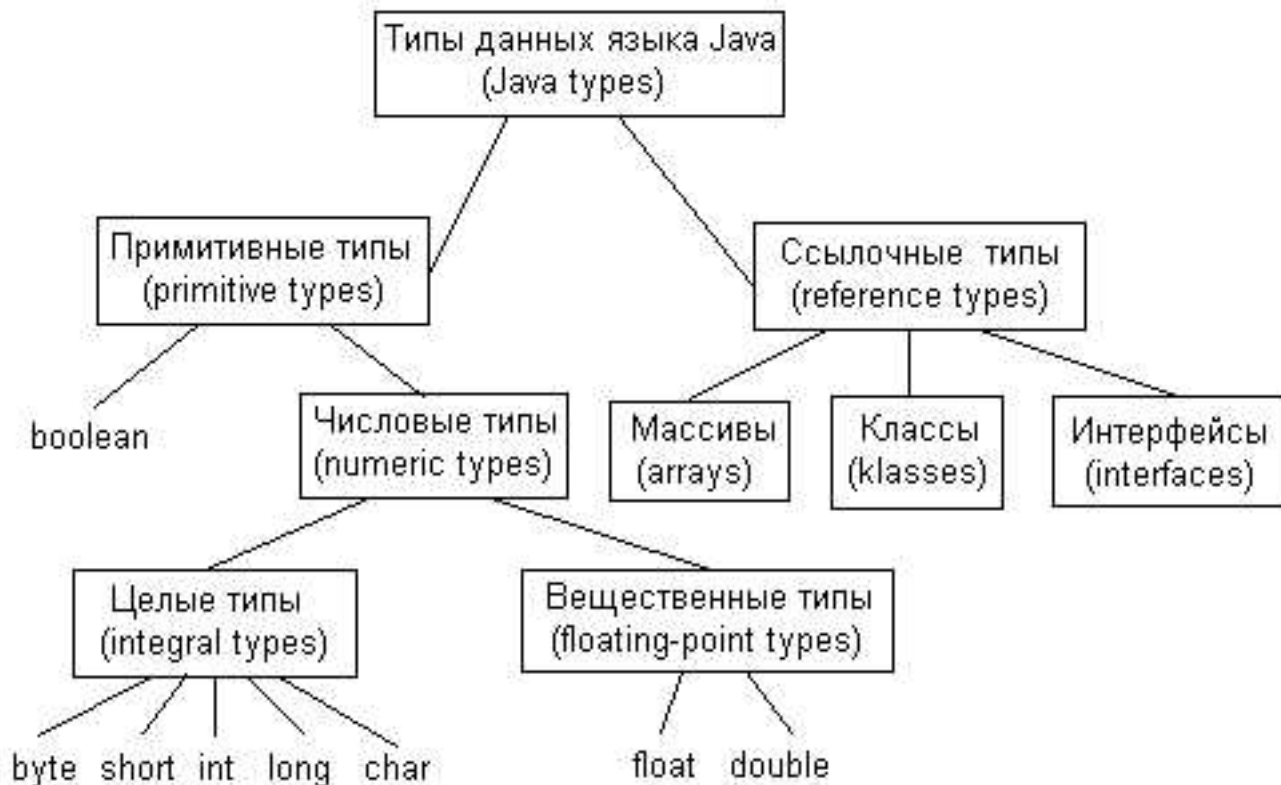
**Сохранение проекта:** Заходим в меню "Файл" – "Сохранить все" (Ctrl+Shift+S).

**Открытие ранее сохраненного проекта:** Заходим в меню "Файл" – "Открыть проект" (Ctrl+Shift+O).

**Настройка конфигурации проекта:** Заходим в меню "Выполнить" – "Установить конфигурацию проекта – Настроить", далее выбираем категорию «Выполнение» – Выбираем главный класс для запуска программы – «ОК».

**Получение готового исполняемого jar-файла:** заходим в меню "Выполнить" – "Очистить и собрать проект" (Shift+F11). Готовый jar-файл находится в папке "dist" проекта.

## 1.3 Простые типы данных



**Целые типы.** Служит для хранения целых чисел.

Тип	Размер, бит	Минимальное значение	Максимальное значение
byte	8	-128	127
short	16	-32768	32767
int	32	-2147483648	2147483647
long	64	-922372036854775808	922372036854775807

```

byte b1 = 50, b2 = -99, b3;
byte a1 = 0xF1, a2 = 0x07;
short det = 0, ind = 1;
int i = -100, j = 100, k = 9999;
long big = 50;
  
```

Оператор	Название	Пример	Примечание
+	Оператор сложения	i+j	В случае, когда операнды i и j имеют разные типы или типы byte, short или char, действуют правила

			автоматического преобразования типов
-	Оператор вычитания	i-j	Результат округляется до целого путем отбрасывания дробной части как для положительных, так и для отрицательных чисел
*	Оператор умножения	i*j	
/	Оператор деления	i/j	
%	Оператор остатка от целочисленного деления	i%j	Возвращается остаток от целочисленного деления
=	Оператор присваивания	v=i	Сначала вычисляется выражение i, после чего полученный результат копируется в ячейку v
++	Оператор инкремента (увеличения на 1)	v++	v++ эквивалентно v=v+1
--	Оператор декремента (уменьшения на 1)	v--	v-- эквивалентно v=v-1
+=		v+=i	v+=i эквивалентно v=v+i
-=		v-=i	v-=i эквивалентно v=v-i
*=		v*=i	v*=i эквивалентно v=v*i
/=		v/=i	v/=i эквивалентно v=v/i
%=		v%=i	v%=i эквивалентно v=v%i

**Символьный тип.** Служит для хранения одного символа.

Тип	Размер, бит	Минимальное значение	Максимальное значение
char	16	0	65536

```
char c1 = 'A', c2 = '?', newLine = '\n';
char s2 = '\u0042';
```

Escape-последовательность	Функция	Значение в Unicode
---------------------------	---------	--------------------

\b	Забой (backspace)	\u0008
\t	Горизонтальная табуляция (horizontaltab)	\u0009
\n	Перевод строки (linefeed)	\u000A
\f	Перевод страницы (form feed)	\u000C
\r	Возврат каретки (carriage return)	\u000D
\"	Двойная кавычка (double quote)	\u0022
\'	Апостроф (single quote)	\u0027
\\	Обратная косая черта (backslash)	\u005C

**Вещественные типы.** Служат для хранения целых и вещественных чисел.

Тип	Разрядность (бит)	Диапазон	Точность
float	32	$3,4\text{e-}38 <  x  < 3,4\text{e}38$	7-8 цифр
double	64	$1,7\text{e-}308 <  x  < 1,7\text{e}308$	17 цифр

```
float x = 0.001, y = -34.789;
double z1 = -16.2305, z2;
float x1 = 3.5f, x2 = 3.7E6f, x3 = -1.8E-7f;
```

Оператор	Название	Пример	Примечание
+	Оператор сложения	$x+y$	В случае, когда операнды $x$ и $y$ имеют разные типы, действуют правила автоматического преобразования типов.
—	Оператор вычитания	$x-y$	
*	Оператор умножения	$x*y$	
/	Оператор деления	$x/y$	

<code>%</code>	Оператор остатка от целочисленного деления	<code>x%y</code>	Возвращается остаток от целочисленного деления <code>x</code> на <code>y</code> . В случае, когда операнды <code>x</code> и <code>y</code> имеют разные типы, действуют правила автоматического преобразования типов.
<code>=</code>	Оператор присваивания	<code>v=x</code>	Сначала вычисляется выражение <code>x</code> , после чего полученный результат копируется в ячейку <code>v</code>
<code>++</code>	Оператор инкремента(увеличения на 1)	<code>v++ ++v</code>	эквивалентно <code>v=v+1</code>
<code>--</code>	Оператор декремента(уменьшения на 1)	<code>v-- --v</code>	эквивалентно <code>v=v-1</code>
<code>+=</code>		<code>v+=x</code>	эквивалентно <code>v=v+x</code>
<code>-=</code>		<code>v-=x</code>	эквивалентно <code>v=v-x</code>
<code>*=</code>		<code>v*=x</code>	эквивалентно <code>v=v*x</code>
<code>/=</code>		<code>v/=x</code>	эквивалентно <code>v=v/x</code>
<code>%=</code>		<code>v%=x</code>	эквивалентно <code>v=v%x</code>

Математические функции, а также константы "пи" (`Math.PI`) и "е" (`Math.E`) заданы в классе `Math`, находящемся в пакете `java.lang`.

Для того чтобы их использовать, надо указывать имя функции или константы, квалифицированное впереди именем класса `Math`.

Оператор класса <code>Math</code>	Примечание
Тригонометрические и обратные тригонометрические функции	
<code>sin(x)</code>	<code>sin(x)</code> – синус
<code>cos(x)</code>	<code>cos(x)</code> – косинус
<code>tan(x)</code>	<code>tg(x)</code> – тангенс
<code>asin(x)</code>	<code>arcsin(x)</code> – арксинус
<code>acos(x)</code>	<code>arccos(x)</code> – арккосинус
<code>atan(x)</code>	<code>arctg(x)</code> – арктангенс
<code>atan2(y, x)</code>	Возвращает угол, соответствующий точке с координатами <code>x,y</code> , лежащий в пределах $(-\pi, \pi]$
<code>toRadians(angdeg)</code>	<code>angdeg / 180.0 * PI</code> ; – перевод углов из градусов в радианы



toDegrees(angrad)	$\text{angrad} * 180.0 / \text{PI}$ ; – перевод углов из радиан в градусы
Степени, экспоненты, логарифмы	
exp(x)	$e^x$ – экспонента
expm1(x)	$e^x - 1$ . При x, близком к 0, дает гораздо более точные значения, чем $\exp(x)-1$
log(x)	$\ln(x)$ – натуральный логарифм
log10(x)	$\log_{10}(x)$ – десятичный логарифм
log1p(x)	$\ln(1 + x)$ . При x, близком к 0, дает гораздо более точные значения, чем $\log(1 + x)$
sqrt(x)	$\sqrt{\phantom{x}}$ – квадратный корень
cbrt(x)	$\sqrt[3]{\phantom{x}}$ – кубический корень
hypot(x,y)	$\sqrt{x^2 + y^2}$ – вычисление длины гипотенузы по двум катетам
pow(x, y)	$X^y$ – возведение x в степень y
sinh(x)	$\text{sh}(x) = \frac{e^x - e^{-x}}{2}$ – гиперболический синус
cosh(x)	$\text{ch}(x) = \frac{e^x + e^{-x}}{2}$ – гиперболический косинус
tanh(x)	$\text{th}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ – гиперболический тангенс
Модуль, знак, минимальное, максимальное число	
abs(m)	Абсолютное значение числа. Аргумент типа int, long, float или double. Результат того же типа, что аргумент
abs(x)	
signum(a)	Знак числа. Аргумент типа float или double. Результат того же типа, что аргумент

signum(x)	
min(m,n)	Минимальное из двух чисел. Аргументы одного типа. Возможны типы: int, long, float, double. Результат того же типа, что аргумент
min(x,y)	
max(m,n)	Максимальное из двух чисел. Аргументы одного типа. Возможны типы: int, long, float, double. Результат того же типа, что аргумент
max(x,y)	
Округления	
ceil(x)	Ближайшее к x целое, большее или равное x
floor(x)	Ближайшее к x целое, меньшее или равное x
round(a)	Ближайшее к x целое. Аргумент типа float или double. Результат типа long, если аргумент double, и типа int – если float. То же, что (int)floor(x + 0.5).
round(x)	
rint(x)	Ближайшее к x целое.
ulp(a)	Расстояние до ближайшего большего чем аргумент значения того же типа ("дискретность" изменения чисел в формате с плавающей точкой вблизи данного значения). Аргумент типа float или double. Результат того же типа, что аргумент
ulp(x)	
Случайное число, остаток	
random()	Псевдослучайное число в диапазоне от 0.0 до 1.0. При этом $0 \leq \text{Math.random()} < 1$
IEEEremainder(x,y)	Остаток от целочисленного деления x/y, то есть $x - y * n$ , где n – результат целочисленного деления

**Булевский (логический) тип.** Служит для хранения логического значения true («Истина») или false («Ложь»).

```
boolean a, b;
a=true; b=a; c=false;
```

Оператор	Название	Пример
&&	логическое "И" ( and )	a&&b
	логическое "ИЛИ" ( or )	a  b

$\wedge$	логическое "исключающее ИЛИ" ( xor )	$a \wedge b$
!	логическое "НЕ" ( not )	$!a$
==	равно	$a == b$
!=	не равно	$a != b$
>	больше	$a > b$
<	Меньше	$a < b$
>=	больше или равно	$a >= b$
<=	меньше или равно	$a <= b$

## 1.4. Управляющие конструкции

Определение управляющих конструкций в Java практически во всём совпадает с C++.

Условные конструкции `if ... else`. Наиболее распространённой формой управляющих структур является конструкция `if ... else`, синтаксис которой выглядит следующим образом:

```
if (БулевскоеВыражение) {  
    Инструкции1;  
} else {  
    Инструкции2;  
}
```

Сначала осуществляется проверка значения булевского выражения. Если результат равен `true`, выполняется блок Инструкции1, в противном случае (и при наличии предложения `else`) – блок Инструкции2. Предложение `else` может быть пропущено, при этом конструкция `if ... else` принимает более краткий вид:

```
if (БулевскоеВыражение) {  
    Инструкции;  
}
```

```
int m = 4;  
if (m == 4) {  
    System.out.println("April");  
}
```

run: April

В этом случае при ложном значении булевского выражения никаких операций не выполняется. Возможна также и вложенность конструкций `if ... else`:

```
if (БулевскоеВыражение1) {  
    Инструкции1  
} else if (БулевскоеВыражение2) {  
    Инструкции2  
} else {  
    Инструкции3  
}
```

```
int month = 4;  
String season;  
if (month == 12 || month == 1 || month == 2) {  
    season = "Winter";  
} else if (month == 3 || month == 4 || month == 5) {  
    season = "Spring";  
} else if (month == 6 || month == 7 || month == 8) {  
    season = "Summer";  
} else if (month == 9 || month == 10 || month == 11) {
```

```
season = "Autumn";  
} else {  
season = "Bogus Month";  
}  
System.out.println("April is in the " + season + ".");
```

run: April is in the Spring.

Некоторым аналогом конструкции `if ... else` является операция «?» со следующим синтаксисом:

БулевскоеВыражение ? Значение1 : Значение2

где Значение1, Значение2 – вычисляемые значения одного типа.

Результатом этой операции будет Значение1, если БулевскоеВыражение истинно, в противном случае –Значение2.

```
int m = 4; String season;  
season = m == 4 ? "April" : "???";  
System.out.println(season);
```

run: April

**Условные конструкции switch – case.** Конструкция `switch` позволяет передавать управление тому или иному блоку кода, обозначенному оператором `case` в зависимости от значения выражения:

```
switch (Выражение) {  
case Значение1:  
Инструкции;  
case Значение2:  
Инструкции;  
...  
default:  
Инструкции;  
}
```

Значение Выражения может иметь один из типов: `byte`, `short`, `int`, `char`. Каждому оператору `case` ставится в соответствие константа-значение. Если значение выражения совпадает со значением оператора `case`, то управление передаётся первой инструкции данного блока `case`.

Для выхода из конструкции после завершения инструкций блока используется команда `break`. Если не прервать исполнение командой `break`, то будет исполняться блок инструкций, соответствующий следующему значению.

Если значение выражения не совпало ни с одним из значений операторов `case`, то выполняется первая инструкция блока `default`. Если же метка `default` отсутствует, выполнение оператора `switch` завершается.

```
int month = 4;
```

```
String season;
switch (month) {
case 12:
case 1:
case 2:
season = "зима"; break;
case 3:
case 4:
case 5:
season = "весна"; break;
case 6:
case 7:
case 8:
season = "лето"; break;
case 9:
case 10:
case 11:
season = "осень"; break;
default:
season = "Нет такого месяца";
}
System.out.println("Апрель – это " + season + ".");
```

run: Апрель – это весна.

**Циклы for.** Выражение for применяется для организации циклического перехода по значениям из заданного диапазона и в общем виде выглядит так:

for (СекцияИнициализации; БулевскоеВыражение; Секция изменения) Инструкция;

Обработка цикла происходит в следующем порядке:

- инициализация;
- проверка условия завершения;
- исполнение тела цикла;
- инкрементация счётчика.

Все секции заголовка цикла for являются необязательными. Если СекцияИнициализации опускается, на её месте остаётся только символ точки с запятой. Если же из заголовка исключается БулевскоеВыражение, в качестве значения логического условия подразумевается литерал true. Исключение всех трёх секций приводит к тому, что цикл становится «бесконечным»:

```
for (;;) {
Инструкции;
}

for (int i = 1; i <= 10; i++) {
System.out.println("i = " + i);
}
```

run:

```
i = 1  
i = 2  
i = 3  
i = 4  
i = 5  
i = 6  
i = 7  
i = 8  
i = 9  
i = 10
```

```
for (int n = 10; n > 0; n--) {  
    System.out.println("n= " + n);  
}
```

```
run:  
n= 10  
n= 9  
n= 8  
n= 7  
n= 6  
n= 5  
n= 4  
n= 3  
n= 2  
n= 1
```

```
int a, b;  
for (a = 1, b = 4; a < b; a++, b--) {  
    System.out.println("a = " + a);  
    System.out.println("b = " + b);  
}
```

```
run:  
a = 1  
b = 4  
a = 2  
b = 3
```

**Циклы while.** Синтаксис циклической конструкции while выглядит так:

while (БулевскоеВыражение)  
Инструкция;

```
int n = 5;  
while (n > 0) {  
    System.out.println("while " + n);  
    n--;  
}
```

```
run:
while 5
while 4
while 3
while 2
while 1
```

Сначала осуществляется проверка булевского выражения. Если результат равен true, выполняется Инструкция (в качестве инструкции может быть использован блок), после чего булевское выражение проверяется вновь, и процесс повторяется до тех пор, пока в результате проверки не будет получено значение false.

**Циклы do ... while.** Если требуется исполнить тело цикла хотя бы 1 раз, используется конструкция do ... while:

```
do
while (БулевскоеВыражение)

int n = 5;
do {
System.out.println("do-while " + n);
} while (-n > 0);

run:
do-while 5
do-while 4
do-while 3
do-while 2
do-while 1
```

В этом случае проверка истинности логического выражения осуществляется после выполнения тела цикла.

В теле циклов можно использовать две особые инструкции:

- break – применяется для завершения выполнения цикла;
- continue – передаёт управление в конец тела цикла (т.е. начинает следующую итерацию). В ситуациях с while и do это приводит к выполнению проверки условия цикла, а при использовании в теле for инструкция continue производит передачу управления секции изменения значения переменных цикла.

**Метки.** В Java отсутствует оператор goto. Но хотя программирование с применением goto и считается плохим тоном, существуют задачи, в которых его использование очень удобно. Основным примером такой ситуации является совершение выхода одновременно из двух вложенных циклов.

Эта проблема решается в Java использованием меток. Любая инструкция в программе может быть снабжена меткой, которая представляет собой содержательное имя, позволяющее сослаться на соответствующую инструкцию.

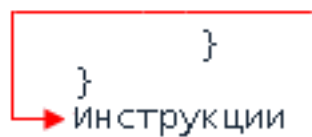
метка: инструкция



Несмотря на то, что метка может быть установлена перед любой инструкцией, на практике имеет смысл применять метки только перед циклическими конструкциями `for`, `while`, `do`, условными конструкциями `if`, `switch` и блоками `{ }`. Чтобы «выбраться» из вложенного цикла или блока, достаточно снабдить меткой соответствующий внешний блок и указать её в команде `break`, которая передаёт управление первой инструкции, следующей за блоком.

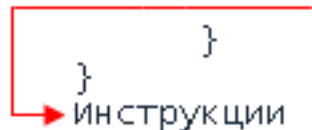
Примеры:

```
labelcycle:
for (...){
    инструкции
    for (...){
        инструкции
        break labelcycle;
    }
}
→ инструкции
```



В данном случае `break` осуществит выход сразу из двух циклов. Ещё один пример выхода из сложной конструкции:

```
lab777:
if (условие) {
    инструкции
    for (...){
        инструкции
        break lab777;
    }
}
→ инструкции
```



```
boolean t = true;
a:
{
b:
{
c:
{
System.out.println("До break");
if (t) {
break b;
}
System.out.println("Не будет выполнено ");
}
System.out.println("Не будет выполнено ");
}
System.out.println("После break");
}
```

run:

До break

После break

**Выход из методов (процедур).** В Java для реализации выхода из методов (процедур) используется метод `return`, который приведет к немедленному завершению работы и передаче управления коду, вызвавшему этот метод. Ниже приведен пример, иллюстрирующий использование оператора `return`:

```
boolean t = true;  
System.out.println("До return");  
if (t) { return; }  
System.out.println("Это не будет выполнено ");
```

run:

До return

Для выхода из программы используется метод `System.exit(КодОшибки)`. Если программа заканчивается нормально, то `КодОшибки` должен быть 0:

```
System.exit(0);
```

## 1.5 Одномерные статические массивы

Одномерные массивы служат для хранения линейного списка с данными. Статические массивы поддерживаются на уровне Java-синтаксиса. Число элементов указывается явно при создании объекта массива или определяется автоматически при перечислении элементов массива. По умолчанию элементы массива объектов устанавливаются в null или в 0 для простых типов. Индексация элементов начинается с 0. Размер массива определяется через функцию length.

Рассмотрим пример работы с одномерными статическими массивами:

```
package tsn01.array;
import java.util.Arrays;
public class TSN01_Array {
    public static void main(String[] args) {
        // Пример работы с одномерными статическими массивами
        int a[], b[]; // Переменные-массивы целых чисел
        a = new int[10]; // Создание массива без инициализации
        b = new int[] { 1, 2, 3, 4 }; // Создание массива с инициализацией
        String s[] = new String[] { "Hello ", "world", "!!!" }; // Описываем и создаем массив строк
        Arrays.fill(a, 0); // Заполнить массив нулями
        a[0] = 20; a[1] = 10; a[2] = 5; a[3] = 33; // Устанавливаем значения 4 элементам
        Arrays.fill(a, 5, 10, -1); // Присвоить с 5 по 9 (10-1) элементам значение "-1"
        Arrays.sort(a); // Сортируем массив
        b[3] = b[1]*0b11+b[2]*0x2; // Рассчитываем значение для 4 элемента
        System.out.println(Arrays.toString(a)); // Вывод на экран массива "a"
        System.out.println(Arrays.toString(b)); // Вывод на экран массива "b"
        System.out.println(Arrays.toString(s)); // Вывод на экран массива "s"
        // Вывод на экран размеров массивов
        System.out.println("Количество элементов в массиве \"a\": " + a.length);
        System.out.println("Количество элементов в массиве \"b\": " + b.length);
        System.out.println("Количество элементов в массиве \"str\": " + s.length);
        System.out.println(s[0] + s[1]); // Доступ к элементам массива
    }
}
```

Результат работы программы:

```
[-1, -1, -1, -1, -1, 0, 5, 10, 20, 33]
[1, 2, 3, 12]
[Hello , world, !!!]
Количество элементов в массиве "a": 10
Количество элементов в массиве "b": 4
Количество элементов в массиве "str": 3
Hello world
```

## 1.6 Многомерные статические массивы

Многомерные статические массивы служат для хранения двумерных таблиц с данными. Рассмотрим пример работы с двумерным статическим массивом:

```
package tsn01.matrix;
public class TSN01_Matrix {
    public static void main(String[] args) {
        // Работа с двумерным статическим массивом чисел
        // Создание исходных данных (элементов массива) и вывод их на экран
        final int r = 4; // Количество строк
        final int c = 5; // Количество столбцов
        int m[][] = new int[r][c]; // Двумерный массив
        int k; System.out.println("Matrix:");
        for (int i = 0; i < r; i++) { // Цикл по строкам
            for (int j = 0; j < c; j++) { // Цикл по колонкам
                k = (int) Math.round(Math.random() * 100); // Получение случайного числа
                m[i][j] = k; // Присвоение элементу массива числа
                System.out.print(String.format("%5d", m[i][j])); // Вывод на экран элемента массива
            } System.out.println("");
        }
        // Поиск минимума и максимума в массиве
        int min = m[0][0], max = m[0][0], maxi = 0, maxj = 0, mini = 0, minj = 0; // Задаем начальные
        значения мин и макс
        for (int i = 0; i < r; i++) { // Цикл по строкам
            for (int j = 0; j < c; j++) { // Цикл по колонкам
                k = m[i][j]; // Получаем элемент массива
                if (k > max) { max = k; maxi = i; maxj = j; } // Поиск максимума
                if (k < min) { min = k; mini = i; minj = j; } // Поиск минимума
            }
        }
        // Меняем максимальный и минимальный элементы в массиве местами
        k = m[maxi][maxj]; m[maxi][maxj] = m[mini][minj]; m[mini][minj] = k;
        // Вывод измененного массива на экран
        System.out.println("New matrix:");
        for (int i = 0; i < r; i++) { // Цикл по строкам
            for (int j = 0; j < c; j++) { // Цикл по колонкам
                System.out.print(String.format("%5d", m[i][j])); // Вывод на экран элемента массива
            } System.out.println("");
        }
    }
}
```

Результат работы программы:

```
Matrix:
42 83 94 96 1
2 64 27 32 10
```

```
20 86 49 14 36
12 35 14 65 97
New matrix:
42 83 94 96 97
2 64 27 32 10
20 86 49 14 36
12 35 14 65 1
```

## 1.7 Динамические массивы-списки

Динамические массивы реализованы на уровне параметризованных классов: Vector и ArrayList. Однако в качестве элементов простые типы выступать не могут, допускаются только объектные типы.

Для управления элементами эти классы используют методы интерфейсов Collection и List:

- add(E o) – добавление элемента в конец;
- add(int index, E element) – вставка элемента в указанную позицию;
- remove(int index) – удаление элемента в указанной позиции;
- remove(Object o) – удаление первого вхождения объекта в списке;
- clear() – удаление всех элементов;
- isEmpty() – определяет, содержит ли список элементы;
- size() – число элементов;
- set(int index, E element) – заменить элемент в указанной позиции новым;
- get(int index) – получить элемент по указанному индексу;
- contains(Object o) – определение, содержится ли указанный объект в списке элементов;
- lastIndexOf(Object o) – поиск последнего вхождения элемента, возвращается индекс элемента или -1;
- indexOf(Object o) – поиск первого вхождения элемента, возвращается индекс элемента или -1;
- toArray() – возвращает копию в виде статического массива;
- toArray(T[] a) – сохраняет элементы в указанный массив.

Пример работы с динамическим массивом целых чисел:

```
package tsn01.arraylist;
import java.util.ArrayList;
public class TSN01_ArrayList {
    public static void main(String[] args) {
        // Работа с динамическим массивом чисел
        ArrayList<Integer> i = new ArrayList<>(); // Создание динамического массива целых
чисел
        i.add(3); // Добавление значения
        i.add(new Integer(3)); // Добавление значения
        if (i.get(0)==i.get(1)) { System.out.println("Эта строка не напечатается..."); }
        if (i.get(0).equals(i.get(1))) { System.out.println("3=3"); }
        i.add(12+5); // Добавление значения
        System.out.println("Размер массива: " + i.size());
        System.out.println("Элементы массива: " + i.get(0).intValue() + ", " + i.get(1)+ ", " + i.get(2));
    }
}
```

Результат работы программы:

```
3=3
Размер массива: 3
Элементы массива: 3, 3, 17
```

Пример работы с динамическим массивом строк:

```
package tsn01.arraylist;
import java.util.ArrayList;
public class TSN01_ArrayList {
    public static void main(String[] args) {
        // Работа с динамическим массивом строк
        ArrayList<String> pozdr = new ArrayList<>(); // Массив пожеланий
        ArrayList<String> fam = new ArrayList<>(); // Массив фамилий
        // Добавление поздравления в массив
        pozdr.add("Удачи"); pozdr.add("Здоровья"); pozdr.add("Денег");
        // добавление фамилии в массив
        fam.add("Петров"); fam.add("Сидоров"); fam.add("Иванов");
        // Проверка количества поздравлений
        if (fam.size() > pozdr.size()) { return; }
        for (int i = 0; i < fam.size(); i++) {
            // Генерируем случайное число в диапазоне от 0 до длины массива поздравлений
            int p = (int) Math.floor(Math.random() * pozdr.size());
            // Генерация поздравления
            System.out.println("Уважаемый " + fam.get(i)
                + "! Поздравляем Вас с этим прекрасным праздником, и желаем Вам "
                + pozdr.get(p).toString().toLowerCase() + "!");
            pozdr.remove(p); // Удаляем элемент с индексом p из массива поздравлений
        }
    }
}
```

## 1.8 Работа со строками

В Java имеется три типа строк: String, StringBuilder и StringBuffer. Статические строки «String» – обычные строки в Java, в которых нельзя изменить символы и их количество после создания строки.

Динамические строки «StringBuilder» – изменяемые строки для использования в однопоточных программах. В однопоточном использовании StringBuilder практически всегда в 1.2-1.5 раза быстрее, чем StringBuffer.

Динамические строки StringBuffer – изменяемые строки для использования в многопоточных программах. Самый медленный тип, но потокобезопасный.

Переменные типа динамических строк могут менять свои значения и длину во время выполнения программы.

Статические строки. Обычные строки в Java описываются классом String и являются статическими, т.е. в существующей строке нельзя изменить символы и их количество.

Кроме стандартного создания оператором new, строки могут быть созданы напрямую из строковой литералы. При этом в целях оптимизации, объекты созданные таким образом дополнительно сохраняются в отдельной области – строковый пул.

```
String s1 = "d" // строка будет сохранена в пуле
// строка не будет сохранена в пуле и будет уничтожена сборщиком мусора
String s2 = new String("a");
```

Один из плюсов разделения строк на статические и динамические – это повышение безопасности там, где строки используются в качестве аргументов (например, открытие баз данных, интернет соединений, механизм загрузки классов).

Операция сцепления. Для строк доступна операция +, позволяющая соединить несколько строк в одну. Если один из операндов не строка, то он автоматически преобразуется в строку. Для объектов в этих целях используется метод toString.

При каждой операции внутренне используется объект динамической строки StringBuilder или StringBuffer. Поэтому для собирания строки из нескольких все равно оптимальней использовать сразу один StringBuilder/StringBuffer.

Выделение подстроки. Есть замечание относительно метода substring – возвращаемая строка использует тот же байтовый массив, что и исходная. Например, вы загрузили строку А из файла в 1мб. Что-то там нашли и выделили в отдельную строку Б длиной в 3 символа. Строка Б в реальности тоже занимает те же 1мб.

```
String s = "very .... long string from file";
String sub1 = s.substring(2,4); // совместно использует ту же память что и s
String sub2 = new String(s.substring(2,4)); // этот объект использует отдельный массив на
4 символа
```

**Основные методы.** Рассмотрим основные методы String:

- equals(Object anObject) – проверяет, идентична ли строка указанному объекту;
- compareTo(String anotherString) – лексиграфическое сравнение строк;
- compareToIgnoreCase(String str) – лексиграфическое сравнение строк без учета регистра символов;



- concat(String str) – возвращает соединение двух строк;
- contains(CharSequence s) – проверяет, входит ли указанная последовательность символов в строку;
- isEmpty() – возвращает true, если длина строки равна 0;
- indexOf(String str) – поиск первого вхождения указанной подстроки;
- replace(CharSequence target, CharSequence replacement) – замена одной подстроки другой;
- substring(int beginIndex, int endIndex) – вернуть подстроку как строку;
- toLowerCase() – преобразовать строку в нижний регистр;
- toUpperCase() – преобразовать строку в верхний регистр;
- trim() – отсечь на концах строки пустые символы;
- length() – определение длины строки;
- valueOf(a) – статические методы преобразования различных типов в строку;
- charAt(int index) – возвращает символ по указанному индексу;
- regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len) – тест на идентичность участков строк, можно указать учет регистра символов;
- regionMatches(int toffset, String other, int ooffset, int len) – тест на идентичность участков строк;
- endsWith(String suffix) – проверяет, завершается ли строка указанным суффиксом;
- startsWith(String prefix) – проверяет, начинается ли строка с указанного префикса;
- startsWith(String prefix, int toffset) – проверяет, начинается ли строка в указанной позиции с указанного префикса;
- getBytes() – возвращает байтовое представление строки;
- getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) – возвращает символьное представление участка строки;
- hashCode() – хеш код строки;
- indexOf(int ch) – поиск первого вхождения символа в строке;
- indexOf(int ch, int fromIndex) – поиск первого вхождения символа в строке с указанной позиции;
- indexOf(String str, int fromIndex) – поиск первого вхождения указанной подстроки с указанной позиции;
- lastIndexOf(int ch) – поиск последнего вхождения символа;
- lastIndexOf(int ch, int fromIndex) – поиск последнего вхождения символа с указанной позиции;
- lastIndexOf(String str) – поиск последнего вхождения строки;
- lastIndexOf(String str, int fromIndex) – поиск последнего вхождения строки с указанной позиции;
- replace(char oldChar, char newChar) – замена в строке одного символа на другой;
- toUpperCase(Locale locale) – преобразовать строку в верхний регистр, используя указанную локализацию;
- toLowerCase(Locale locale) – преобразовать строку в нижний регистр, используя указанную локализацию;

Методы поиска возвращают индекс вхождения или -1, если искоемое не найдено. Методы преобразования (как replace) не изменяют саму строку, а возвращают соответствующий новый объект строки.

**Особенности String.** Неправильное использование типа String приводит к засорению оперативной памяти и как следствие к медленной работе программы. Рассмотрим пример:

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

```
String s = "a";
for (int i = 0; i < 100; i++) {
    s += 'a';
}
System.out.println(s); // Распечатается строка из 100 символов «a»
}
```

Этот код создаст 100 разных строк, которые будут храниться в памяти, пока сборщик мусора Java не удалит их после завершения программы. В результате работы программы переменная «s» будет содержать строку из 100 символов «a», но 99 ранее созданных строк остались «мусором» в памяти без возможности обращения к ним.

Поэтому для изменения строки следует использовать класс обертку `StringBuilder`. Предыдущий пример нужно переписать следующим образом:

```
public static void main(String[] args) {
    StringBuilder s = new StringBuilder("a");
    for (int i = 0; i < 100; i++) {
        s.append('a');
    }
    System.out.println(s); // Распечатается строка из 100 символов «a»
}
```

**Динамические строки.** Динамические строки описываются классами `StringBuilder` и `StringBuffer`. `StringBuffer` более медленный, но потокобезопасный. Переменные типа динамических строк могут менять свои значения и длину во время выполнения программы.

Основные методы динамических строк:

- `append(A)` – преобразовать A в строку и добавить в конец;
- `insert(int offset, A)` – преобразовать A в строку и вставить ее в указанную позицию;
- `delete(int start, int end)` – удалить символы с указанной начальной позиции по указанную конечную позицию;
- `reverse()` – расположить символы в обратном порядке;
- `setCharAt(int index, char ch)` – заменить символ в указанной позиции;
- `setLength(int newLength)` – установить новый размер строки;
- `substring(int start)` – вернуть подстроку с указанной позиции и до конца как строку;
- `substring(int start, int end)` – вернуть подстроку как строку;
- `deleteCharAt(int index)` – удалить символ в указанной позиции;
- `getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)` – сохранить последовательность символов в массив;
- `indexOf(String str)` – поиск первого вхождения подстроки;
- `indexOf(String str, int fromIndex)` – поиск первого вхождения подстроки с указанной позиции;
- `lastIndexOf(String str)` – поиск последнего вхождения подстроки;
- `lastIndexOf(String str, int fromIndex)` – поиск последнего вхождения подстроки с указанной позиции;
- `replace(int start, int end, String str)` – замена участка строки указанной строкой.

**Пример преобразования строк.** В этом примере массив символов и целое число преобразуются в объекты типа `String` с использованием методов этого класса:

```
package ts01.string;
```

```

public class TSN01_String {
    public static void main(String[] args) {
        char s[] = {'J', 'a', 'v', 'a'}; // Массив символов
        String str = new String(s); // str="Java"
        if (!str.isEmpty()) {
            int i = str.length(); // i=4
            str = str.toUpperCase(); // str="JAVA"
            String num = String.valueOf(6); // num="6"
            num = str.concat("-" + num); // num="JAVA-6"
            char ch = str.charAt(2); // ch='V'
            i = str.lastIndexOf('A'); // i=3 (-1 если нет)
            num = num.replace("6", "SE"); // num="JAVA-SE"
            str.substring(0, 4).toLowerCase(); // java
            str = num + "-6"; // str="JAVA-SE-6"
            String[] arr = str.split("-");
            for (String ss : arr) { // В результате будет выведен массив строк (в 3 строчки): JAVA SE 6
                System.out.println(ss);
            }
        } else { System.out.println("String is empty!"); }
    }
}

```

**Пример сравнение строк.** В этом примере рассмотрены особенности хранения и идентификации объектов на примере вызова метода equals(), сравнивающего строку String с указанным объектом и метода hashCode(), который вычисляет хэш-код объекта (hashCode – это цифра, которая формируется для объекта по какому то правилу, например для объекта класса String по такой формуле:  $s[0]*31^{(n-1)} + s[1]*31^{(n-2)} + \dots + s[n-1]$ ):

```

package tsn01.string;
public class TSN01_String {
    public static void main(String[] args) {
        String s1 = "Java";
        String s2 = "Java";
        String s3 = new String("Java");
        System.out.println(s1 + "==" + s2 + " : " + (s1 == s2)); // true
        System.out.println(s1 + "==" + s3 + " : " + (s1 == s3)); // false
        System.out.println(s1 + " equals " + s2 + " : " + s1.equals(s2)); // true
        System.out.println(s1 + " equals " + s3 + " : " + s1.equals(s3)); // true
        System.out.println(s1.hashCode());
        System.out.println(s2.hashCode());
        System.out.println(s3.hashCode());
    }
}

```

В результате на экран будет выведено:

```

Java==Java : true
Java==Java : false
Java equals Java : true

```

```
Java equals Java : true
2301506
2301506
2301506
```

Пример сортировки массива строк методом перебора:

```
package tsn01.string;
public class TSN01_String {
    public static void main(String[] args) {
        String a[] = {" Alena", "Alice ", " alina", " albina", " Anastasya",
            " ALLA ", "AnnA "}; // Массив строк
        for (int j = 0; j < a.length; j++) { // Цикл по массиву строк
            // Удаляем пробелы с концов строк и приводим к верхнему регистру
            a[j] = a[j].trim().toLowerCase();
        }
        // Сортировка строк методом пузырька
        for (int j = 0; j < a.length - 1; j++) { // Цикл по массиву строк
            for (int i = j + 1; i < a.length; i++) { // Цикл по массиву строк
                if (a[i].compareTo(a[j]) < 0) { // Сравнение строк
                    String temp = a[j]; a[j] = a[i]; a[i] = temp; // Обмен значений в массиве строк
                }
            }
        }
        int i = -1;
        while (++i < a.length) { System.out.print(a[i] + " "); } // Вывод массива строк на экран
    }
}
```

В результате на экран будет выведено:

```
albina alena alice alina alla anastasya anna
```

Вызов метода trim() обеспечивает удаление всех начальных и конечных символов пробелов. Метод compareTo() выполняет лексикографическое сравнение строк между собой по правилам Unicode.

**Пример работы с динамическими строками.** Рассмотрим пример преобразования переменной типа «StringBuilder» к «String» через метод toString:

```
package tsn01.string;
public class TSN01_String {
    public static void main(String[] args) {
        StringBuilder s = new StringBuilder("abcd");
        s.append('e');//abcde
        s.delete(1, 2);//acde
        s.insert(1, 'b');//abcde
        s.deleteCharAt(2);//abde
        String ans = s.toString();
        System.out.println(ans); // На экран выведется "abde"
```

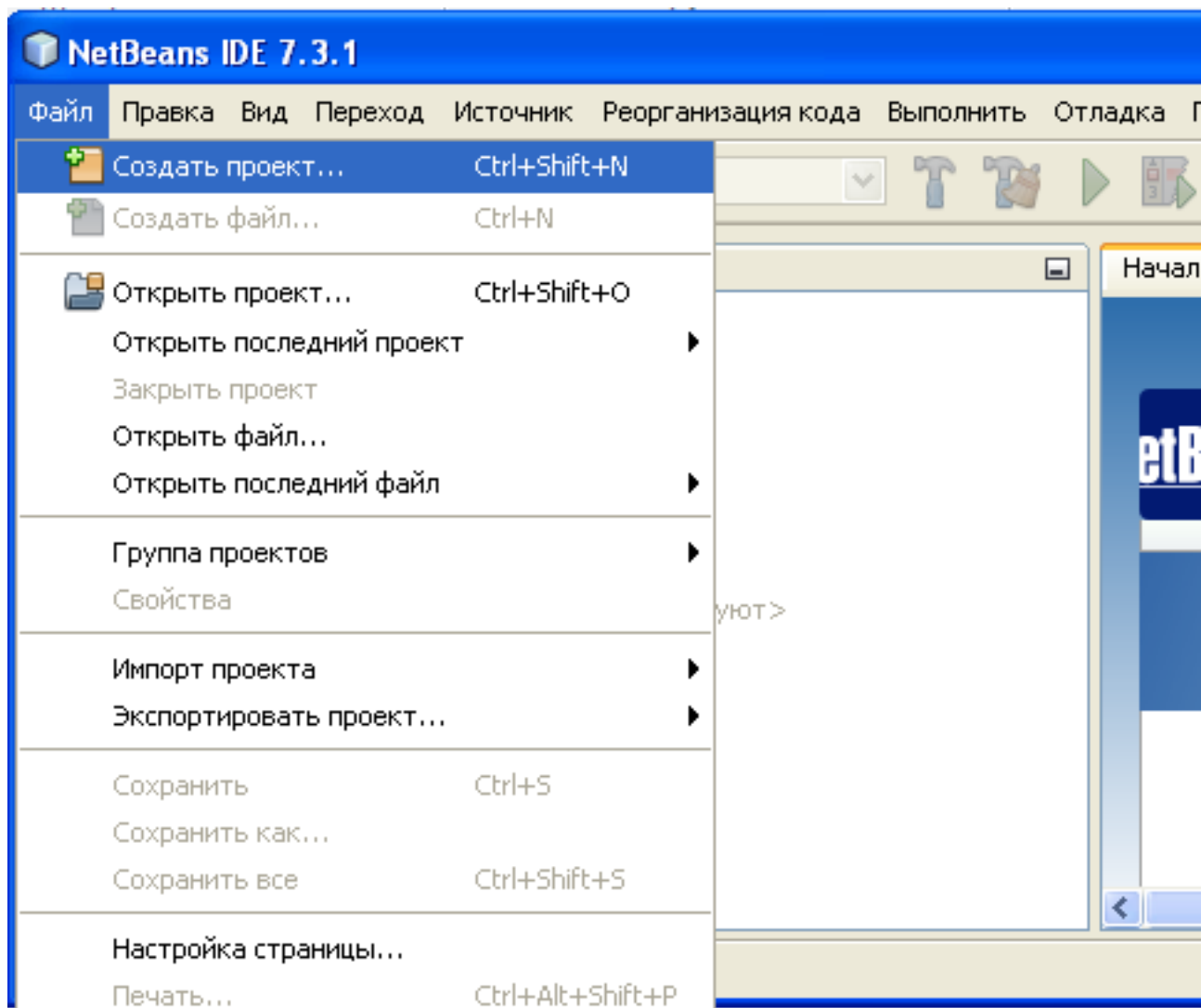
```
}  
}
```

## **2 Простейшие программы**

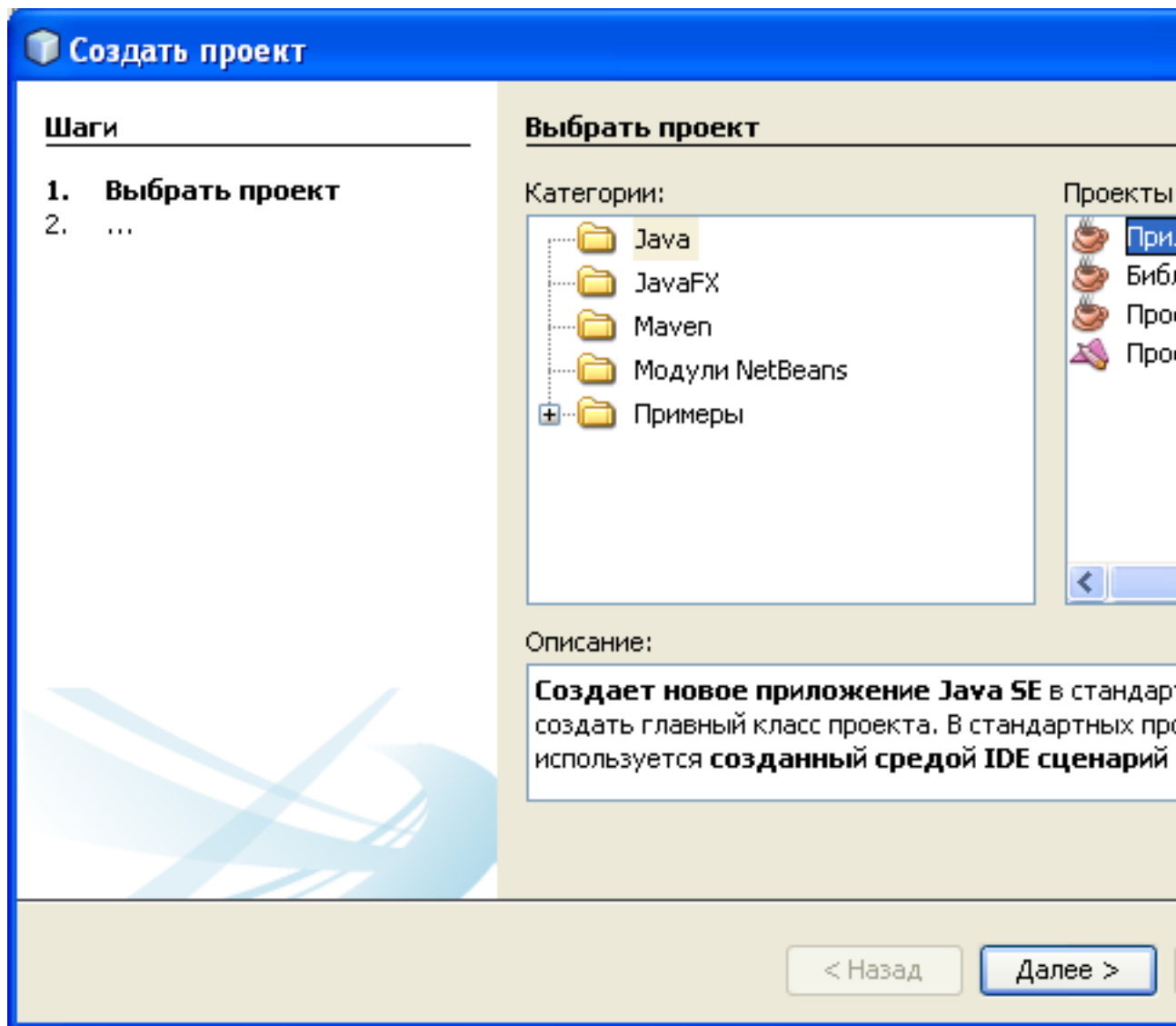
## 2.1 Консольные программы

Консольные программы на Java – это наиболее простой вид программ, не имеющих собственного графического интерфейса, весь ввод и вывод информации происходит в окне консоли. В настоящее время консольные программы не имеют особого практического применения, их заменяют программами с графическим интерфейсом.

Для создания консольной программы необходимо выбрать в меню опцию «Файл» – «Создать проект»:



Выбрать категорию «Java» – «Приложение Java»:



Указать имя проекта и необходимость создать главный класс. В главном классе и будет располагаться консольная программа.

Имя проекта необходимо задать так: «FIO<sub>nn</sub>\_DEMO», где FIO – инициалы автора программы, nn – номер варианта, например, «TSN01\_DEMO».

Название главного класса необходимо задать так: «fionn.demo.App1», где fio – инициалы автора программы, nn – номер варианта, например «tsn01.demo.App1».



**Новый Приложение Java**

**Шаги**

1. Выбрать проект
- 2. Имя и расположение**

**Имя и расположение**

Имя проекта:

Расположение проекта:

Папка проекта:

☐ Использовать отдельную папку для хранения бинарных файлов

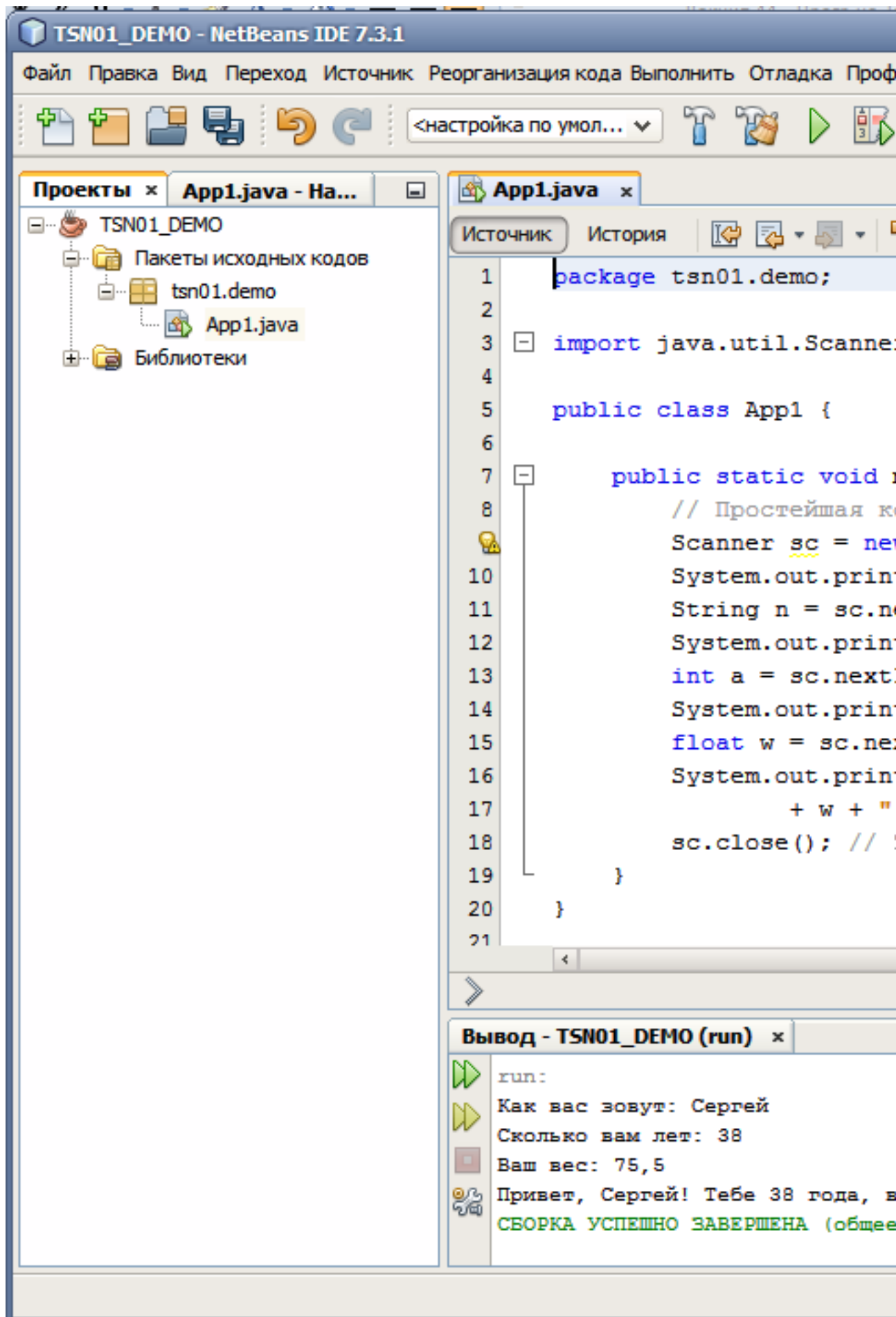
Папка с библиотеками:

Разные пользователи и группы могут использовать разные библиотеки компиляции (например, справочной системе).

☒ Создать главный класс

< Назад

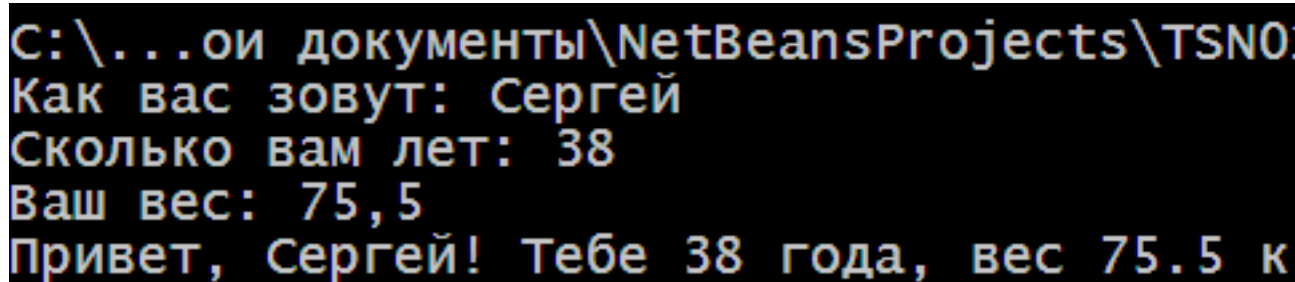
Рассмотрим пример простейшей консольной программы.



```
package tsn01.demo;
import java.util.Scanner;
public class App1 {
    public static void main(String[] args) {
        // Простейшая консольная программа
        Scanner sc = new Scanner(System.in); // Подключение к консоли
        System.out.print("Как вас зовут: "); // Вывод вопроса
        String n = sc.next(); // Ввод с консоли строкового значения
        System.out.print("Сколько вам лет: "); // Вывод вопроса
        int a = sc.nextInt(); // Ввод с консоли целого значения
        System.out.print("Ваш вес: "); // Вывод вопроса
        float w = sc.nextFloat(); // Ввод с консоли вещественного значения
        System.out.println("Привет, " + n + "! Тебе " + a + " года, вес "
            + w + " кг.");
        sc.close(); // Закрытие консоли
    }
}
```

Запуск программы через командную строку Windows с поддержкой русского языка:

```
java -Dfile.encoding=Cp866 -jar TSN01_DEMO.jar
```

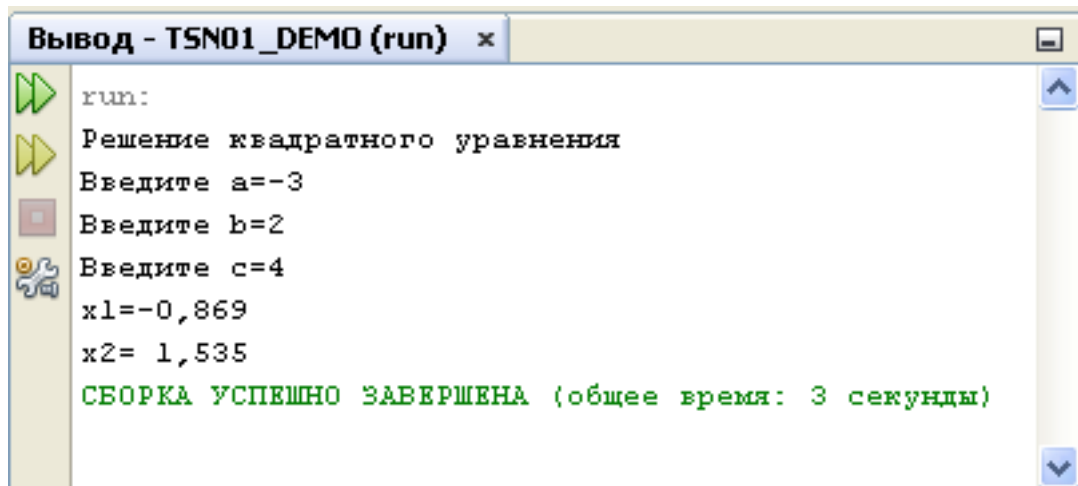


```
C:\...ои документы\NetBeansProjects\TSN01
Как вас зовут: Сергей
Сколько вам лет: 38
Ваш вес: 75,5
Привет, Сергей! Тебе 38 года, вес 75.5 к
```

Рассмотрим пример консольной программы для решения квадратного уравнения.

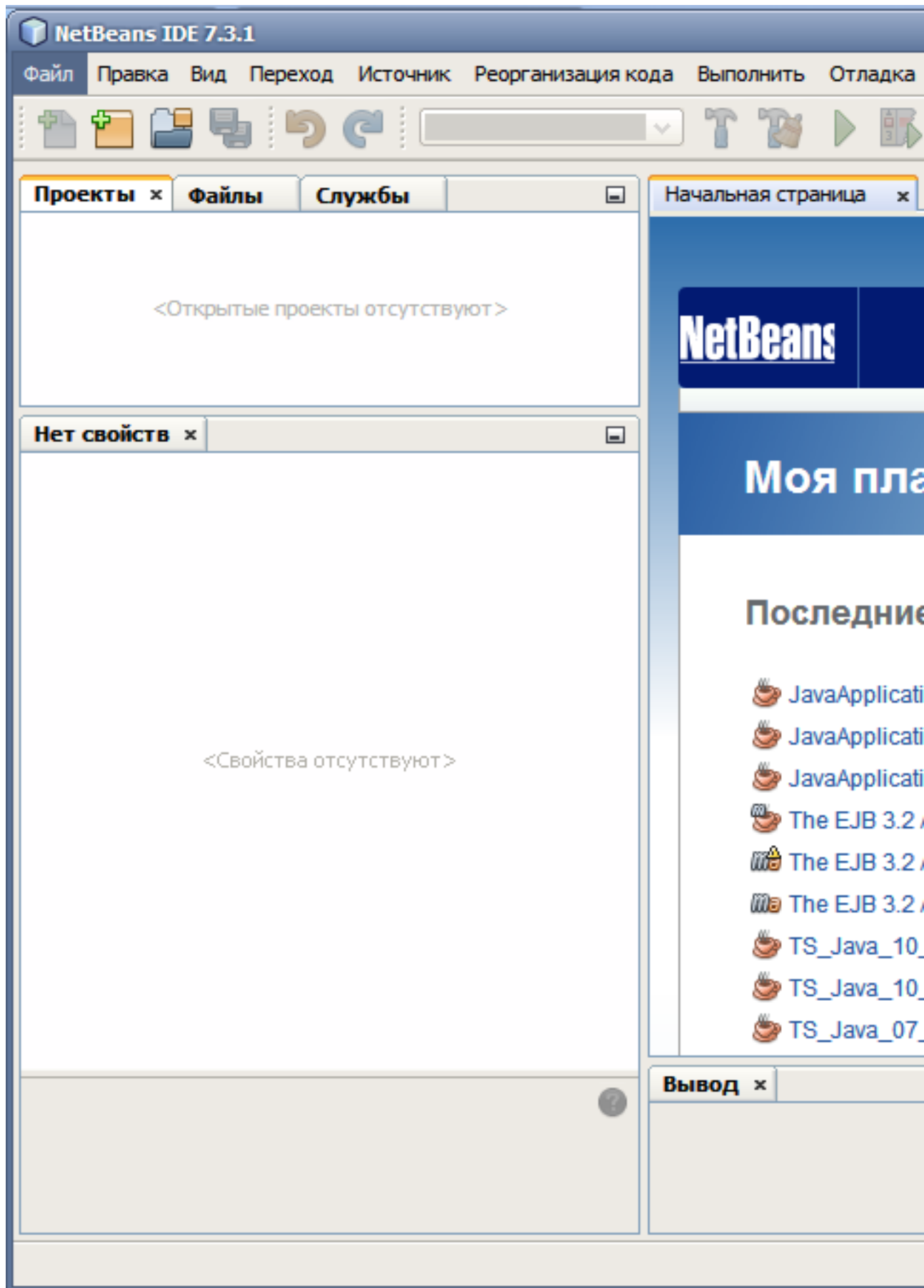
```
package tsn01.demo;
import java.util.Scanner;
public class App1 {
    public static void main(String[] args) {
        // Вычисление квадратного уравнения
        double a, b, c; // Входные переменные
        double x1, x2; // Искомые значения
        double d; // Дискриминант
        try {
            Scanner sc = new Scanner(System.in); //Создаем объект для ввода данных с консоли
            System.out.println("Решение квадратного уравнения");
            System.out.print("Введите a=");
            a = sc.nextDouble(); // Ввод значения "a" с консоли
            System.out.print("Введите b=");
            b = sc.nextDouble(); // Ввод значения "b" с консоли
```

```
System.out.print("Введите c=");  
c = sc.nextDouble(); // Ввод значения "c" с консоли  
d = (b * b) - 4 * a * c; // Расчет дискриминанта  
x1 = (-b + Math.sqrt(d)) / (2 * a); // Расчет "x1"  
x2 = (-b - Math.sqrt(d)) / (2 * a); // Расчет "x2"  
if (!(Double.isNaN(x1)) && (!Double.isInfinite(x1)) // Проверка существования значений  
&& (!(Double.isNaN(x2)) && (!Double.isInfinite(x2)))) {  
System.out.format("x1=%.3f\nx2= %.3f\n", x1, x2); // Вывод ответа  
} else {  
System.out.println("Нет решения!"); // Нет решения  
}  
sc.close(); // Закрываем ввод с консоли  
} catch (Exception e) { // Ввели вместо цифр буквы  
System.out.println("Неверные входные данные!");  
}  
}  
}
```

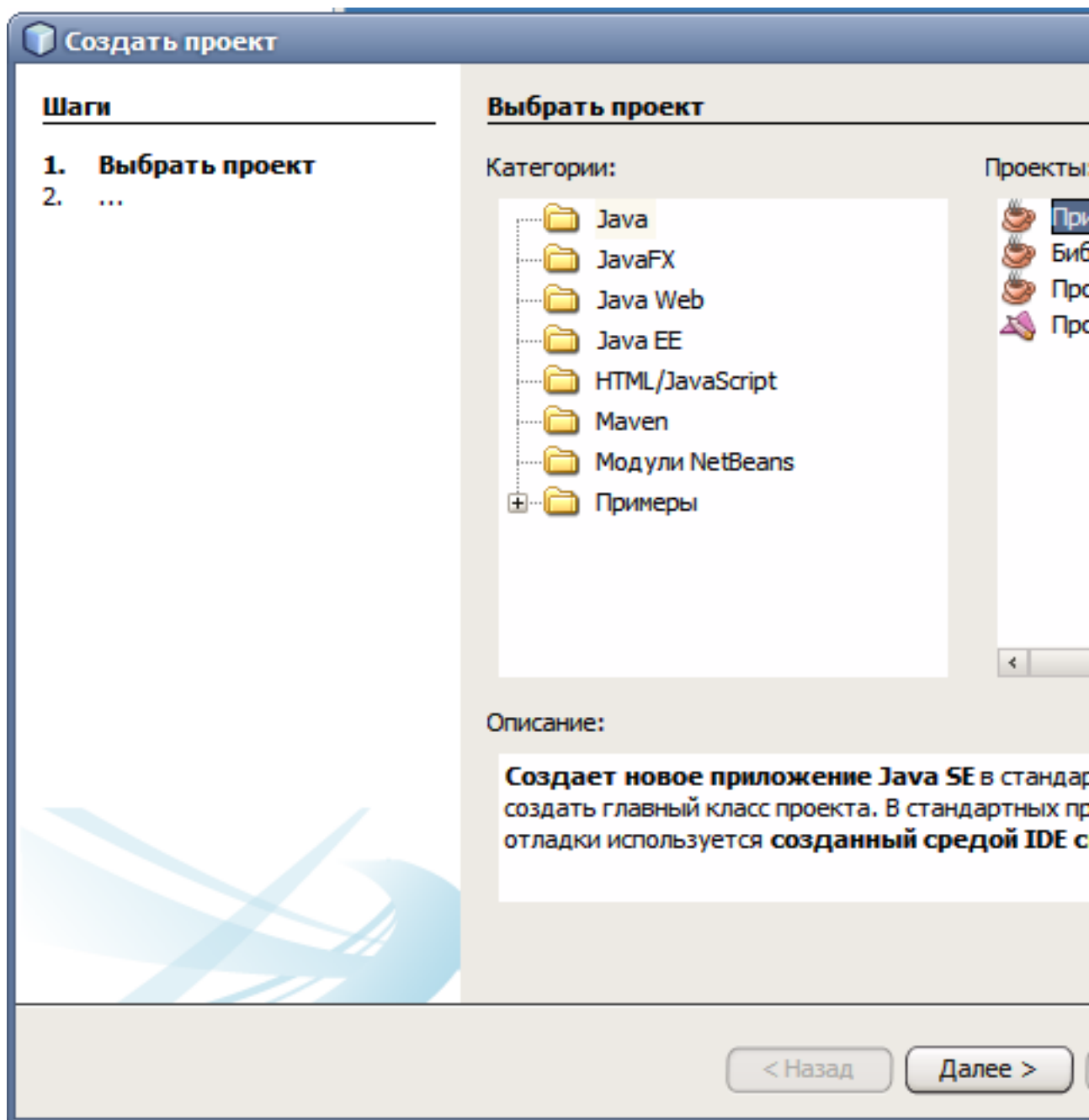


## **2.2 Простейшая программа с графическим интерфейсом в среде NetBeans**

Заходим в меню "Файл" – "Создать проект":

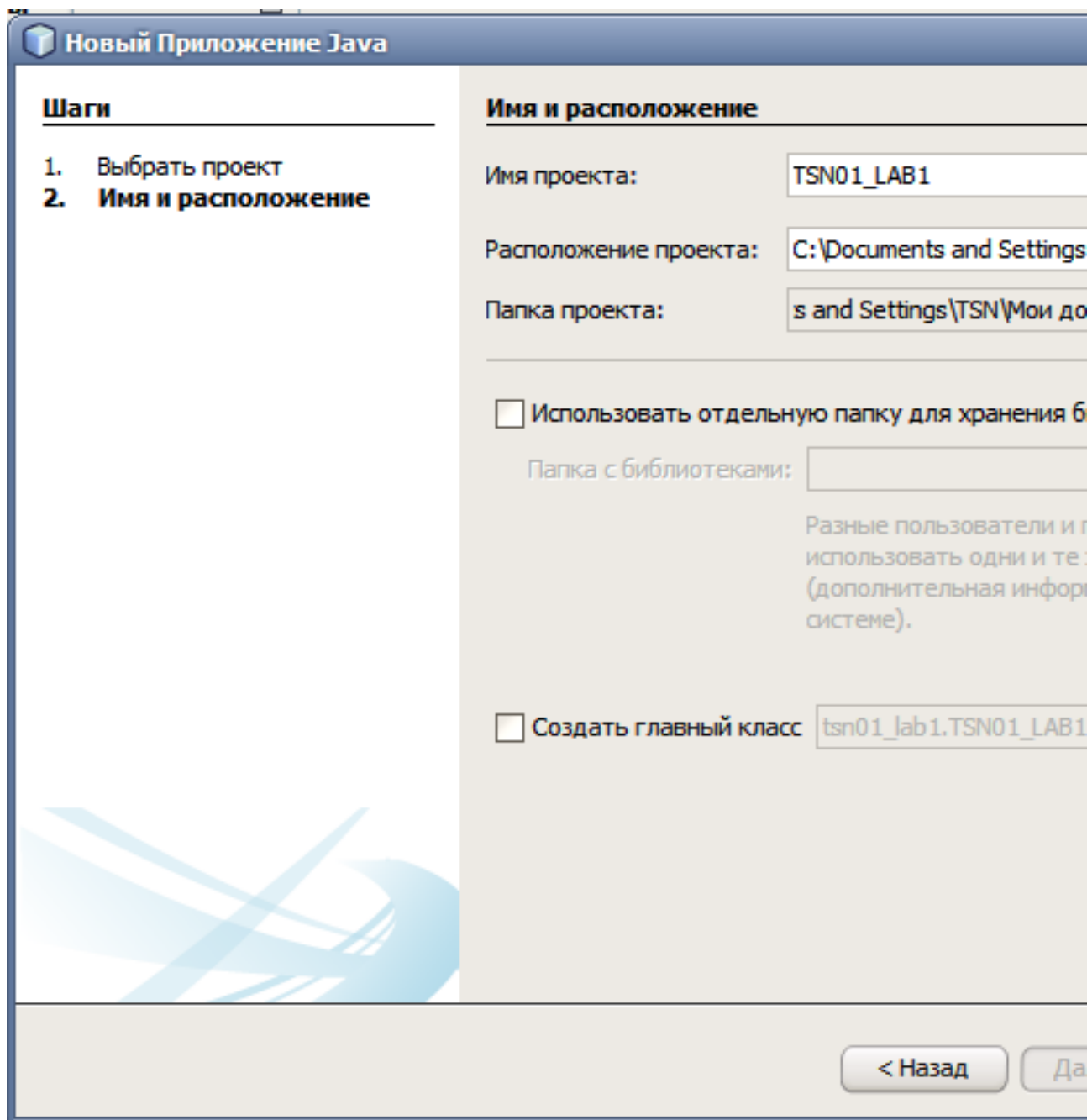


Выбираем категорию "Java", тип проекта "Приложение Java":



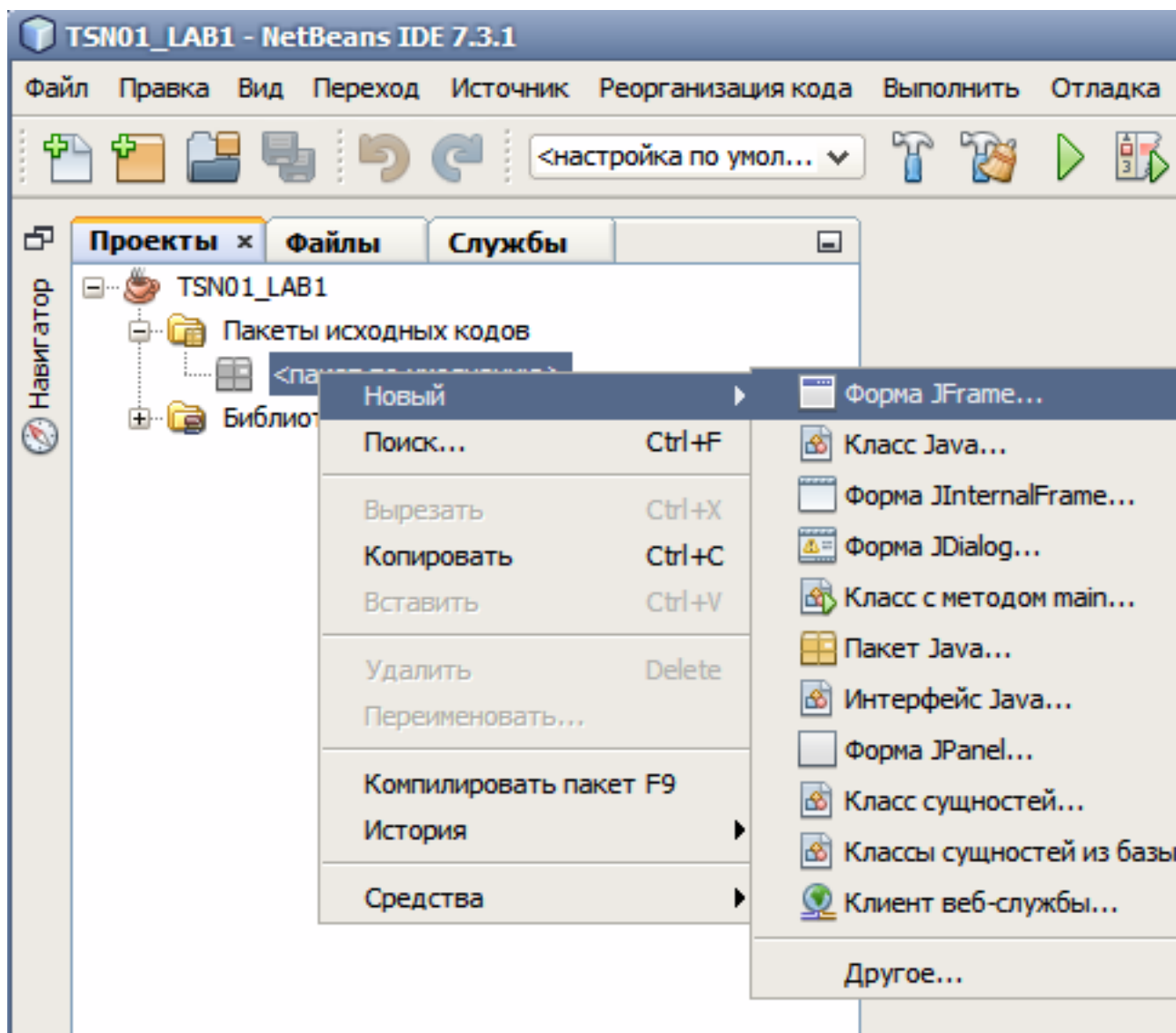
Указываем имя проекта. Имя проекта необходимо задать по шаблону «TSN01\_LAB1», где «TSN» – инициалы автора программы большими буквами, «01» – номер варианта (2 цифры), далее идет знак «\_», слово «LAB» большими буквами и номер лабораторной работы. При необходимости указываем новый каталог для расположения проекта. Нажимаем кнопку «Готово».

При создании приложения необходимо указать, что создавать главный класс не нужно, убрав соответствующую галочку, т.к. главный класс будет располагаться в окне:



Устанавливаем курсор мыши на «пакет по-умолчанию», после этого нажимаем на этом пункте правую кнопку мыши. В появившемся контекстном меню выбираем «Новый» – «Форма JFrame»:

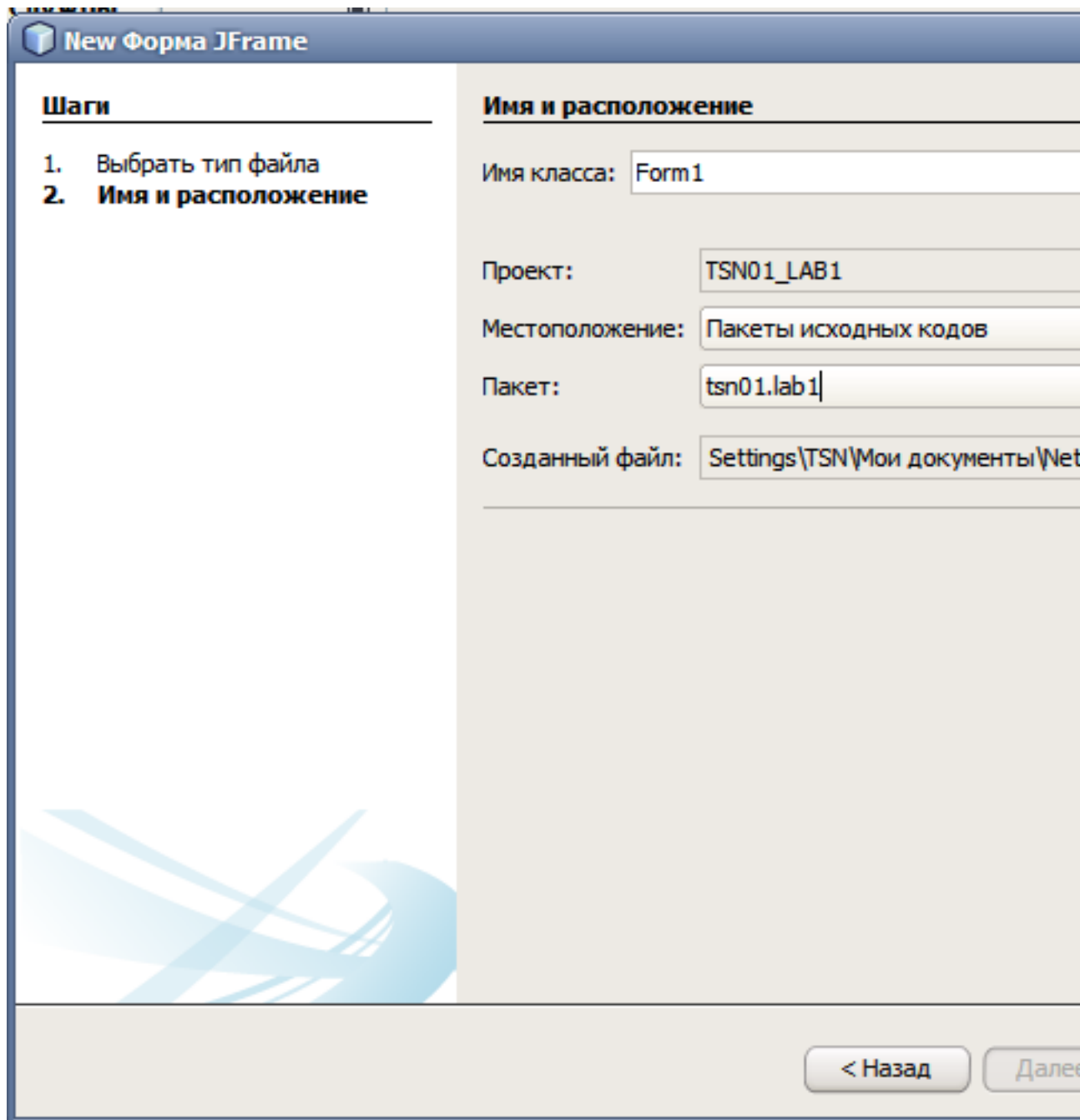




Указываем имя класса (окна). Рекомендуется имя формы задавать по шаблону «Form1», где «Form» – означает, что это форма, первая буква большая, а «1» – номер формы в программе.

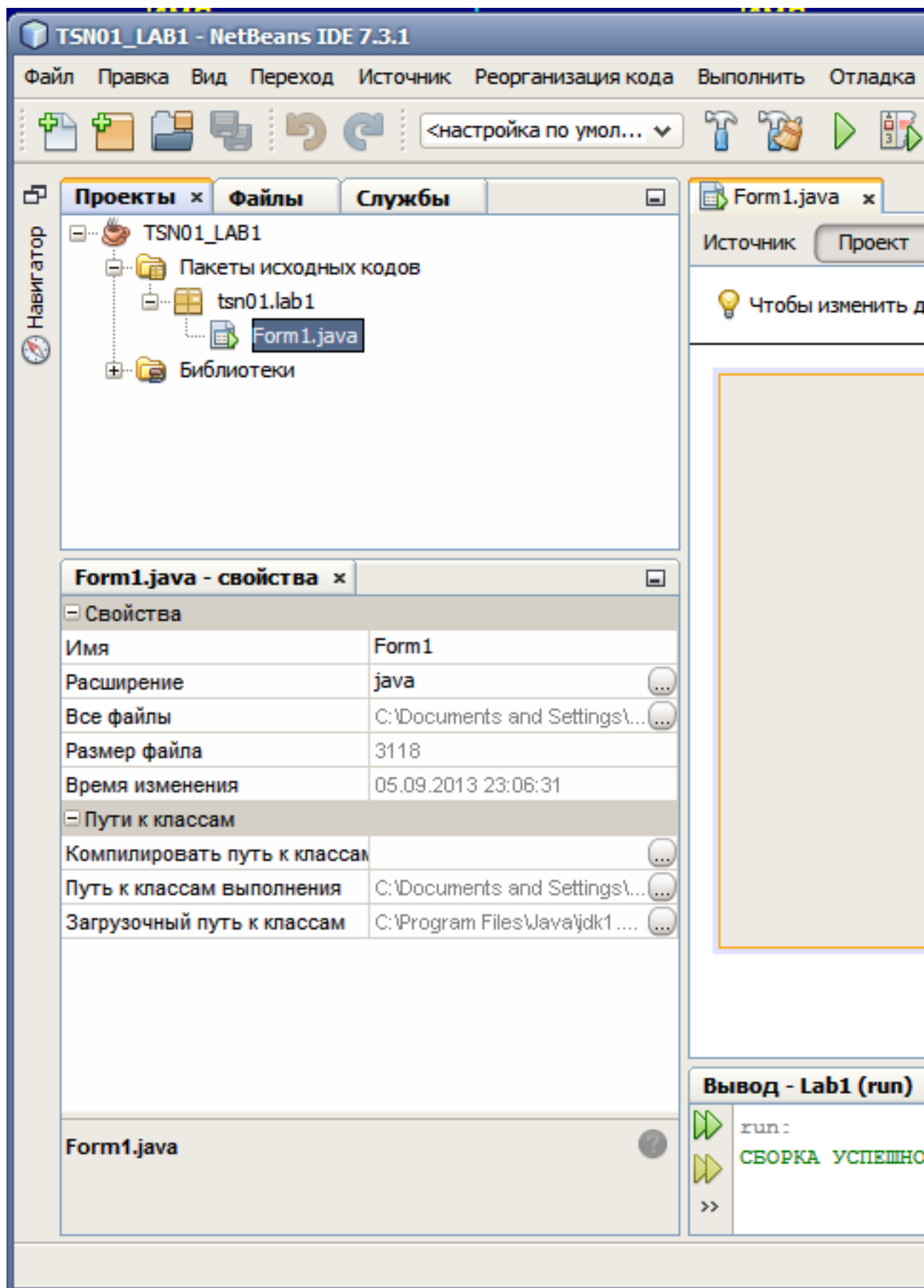
Далее, необходимо указать имя пакета. Имя пакета необходимо задать по шаблону «tsn01.lab1», где «tsn» – инициалы автора программы маленькими буквами, «01» – номер варианта (2 цифры), далее идет знак «.» (точка), слово «lab» маленькими буквами и номер лабораторной работы.

Нажимаем кнопку «Готово».

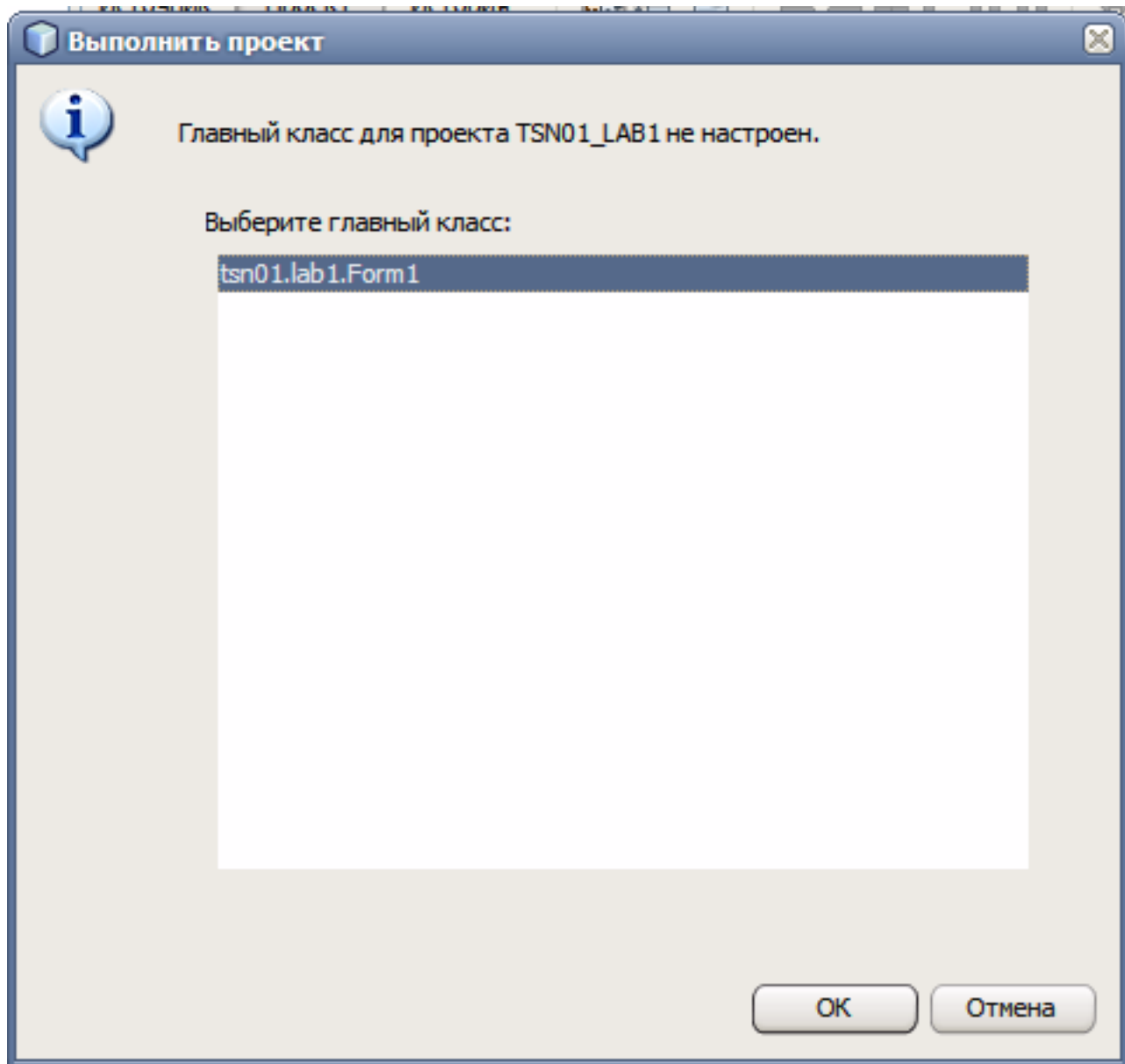


Выделяем в окне «Проекты» полученную форму «Form1». Простейшая программа из пустого окна готова.

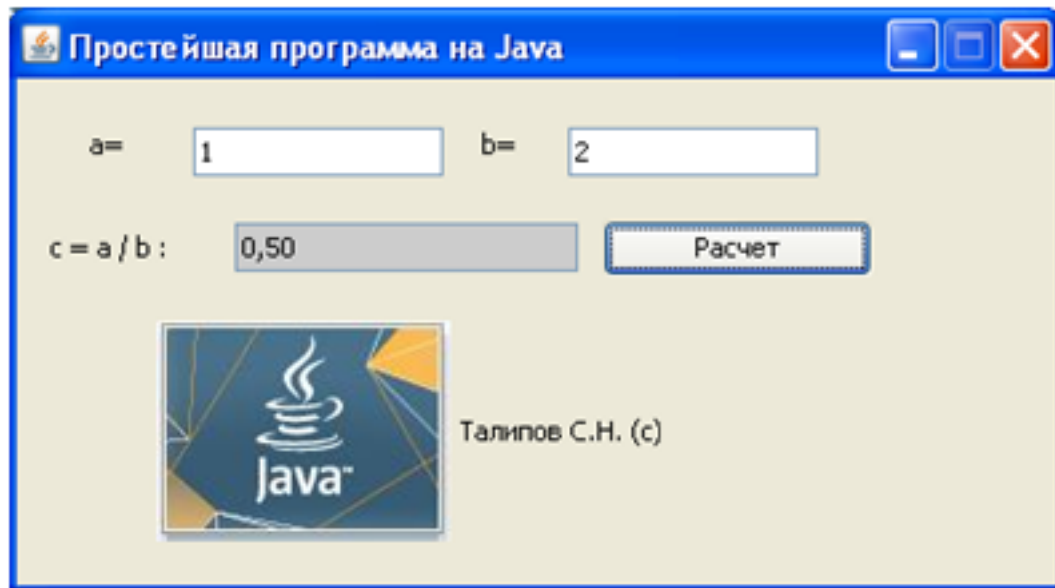
Запускаем полученную программу: «Выполнить» – «Запустить проект» (F6).



Появится окно для указания главного класса в программе. Необходимо выбрать класс и именем формы и нажать «ОК». После этого запустится программа.



Рассмотрим пример кода для кнопки «Расчет» и интерфейс простейшей программы деления двух цифр:



```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
// Вычисление выражения
String a, b, c; // Строковые переменные
double z; // Вещественные переменные
double x, y; // Вещественные переменные
// int x, y; // Целочисленные переменные
a = jTextField1.getText(); // Получение значения из окошка 1
b = jTextField2.getText(); // Получение значения из окошка 2

try { // Начало защищенного блока
x = Double.parseDouble(a); // Преобразование текстового значения в вещественное
y = Double.parseDouble(b); // Преобразование текстового значения в вещественное
// x = Integer.parseInt(a); // Преобразование текстового значения в целочисленное
// y = Integer.parseInt(b); // Преобразование текстового значения в целочисленное

z = x / y; // Вычисление выражения
// z = (double)( x) / (double) (y); // Вычисление выражения

// Проверка на: 0/0, z/0
if ((Double.isNaN(z) == true) || Double.isInfinite(z) == true) {
throw new Exception("error"); // Если нет решение то генерирование ошибки
}

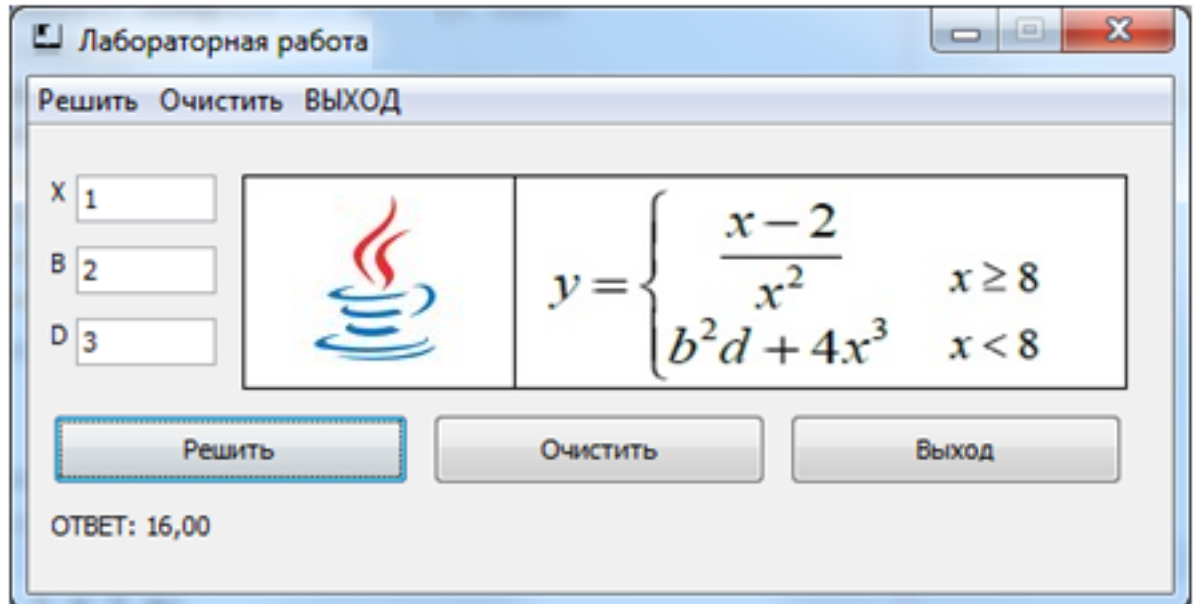
// Описание формата вещественного числа
DecimalFormat df = new DecimalFormat("#0.00");
c = String.valueOf(df.format(z)); // Преобразование числа в строку
jTextField3.setText(c); // Вывод ответа в окошко
// jTextField3.setText(String.valueOf(new DecimalFormat("#0.00").format(z))); // Вывод
ответа в окошко
} catch (Exception ee) { // Обработчики ошибок защищенного блока
Toolkit.getDefaultToolkit ().beep (); // Звуковой сигнал
jTextField3.setText("Неверные данные!"); // Обработка ошибки ввода или вычисления
}
```

```

    } // Конец защищенного блока
}

```

Рассмотрим пример кода кнопки «Решить» и «Выход» для расчета математического значения по условию:



```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
// Решение примера
float x, b, d, y; // Вещественные переменные

try {
x = Float.parseFloat(jTextField1.getText()); // Получение данных
b = Float.parseFloat(jTextField2.getText()); // Получение данных
d = Float.parseFloat(jTextField3.getText()); // Получение данных
} catch (Exception ex) {
Toolkit.getDefaultToolkit().beep(); // Издаем звук
// Выводим окошко с сообщением об ошибке
JOptionPane.showMessageDialog(rootPane, "Ошибка введенных данных!", "Ошибка
ввода", JOptionPane.ERROR_MESSAGE);
jTextField1.requestFocus(); // Устанавливаем фокус на компонент
jLabel5.setText("В введенных значениях допущены ошибки.");
jTextField1.setText(""); // Очистка данных
jTextField2.setText(""); // Очистка данных
jTextField3.setText(""); // Очистка данных
return; // Выход из метода (процедуры)
}

if (x >= 8) { // Вычисление выражения
y = (x - 2) / (x * x);
} else {
y = b * b * d + 4 * x * x * x;
}
}

```

```
        jLabel5.setText("ОТВЕТ: " + String.format("%.2f", y)); // Выдача ответа с двумя знаками  
        после запятой  
    }
```

```
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {  
        // Выход из программы  
        System.exit(0);  
    }
```

```
    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
```

## **Конец ознакомительного фрагмента.**

Текст предоставлен ООО «ЛитРес».

Прочитайте эту книгу целиком, [купив полную легальную версию](#) на ЛитРес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.