

Лекция 2

**Переменные,
типы данных и операторы,
конвертация типов,
ВВОД-ВЫВОД, класс Scanner,
класс Math**

Вычисление площади круга

```
package ru.vsu.cs.course1;

import java.util.Locale;
import java.util.Scanner;

public class Program {
    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);

        Scanner scanner = new Scanner(System.in);

        System.out.print("Введите радиус круга: R = ");
        double r = scanner.nextDouble();

        double s = Math.PI * r * r;

        System.out.printf(
            "Для круга радиуса R = %1$.3f площадь s = %2$.3f%n",
            r, s
        );
    }
}
```

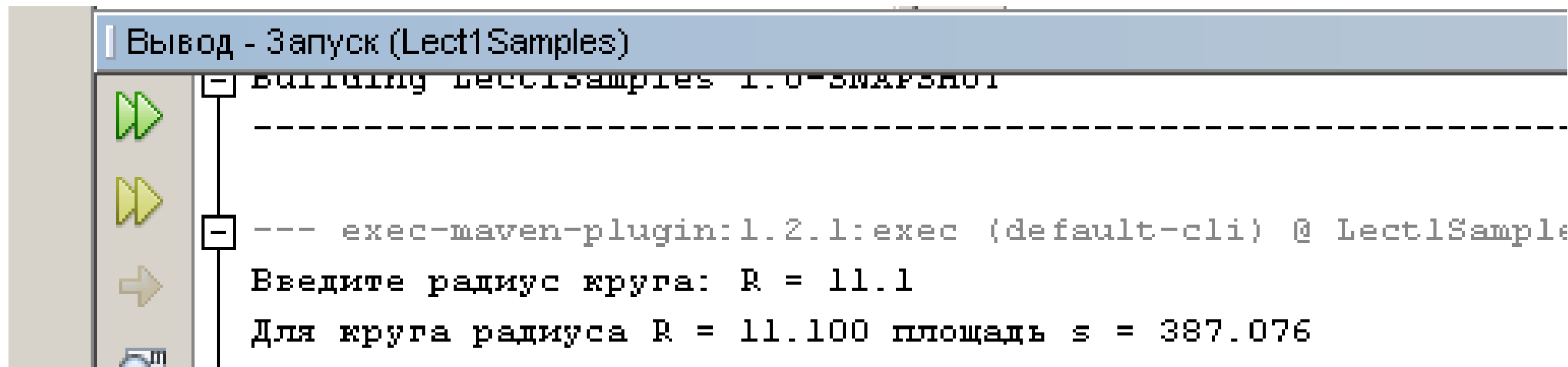
Вычисление площади круга

Программа выполняется по шагам:

```
1) Locale.setDefault(Locale.ROOT);  
2) Scanner scanner = new Scanner(System.in);  
3) System.out.print("Введите радиус круга: R = ");  
4) double r = double.Parse(Console.ReadLine());  
5) double s = Math.PI * r * r;  
6) System.out.printf(  
    "Для круга радиуса R = %1$.3f площадь s = %2$.3f%n",  
    r, s  
);
```

- 1) Первоначальная настройка среды исполнения (не относится к алгоритму)
- 2) Создание сканера (`java.util.Scanner`) для удобства ввода данных
- 3-4) Ввод значение радиуса R (в переменную r)
- 5) Вычисление площади S (в переменную s)
- 6) Вывод результата

Вычисление площади круга (результат выполнения)



```
Вывод - Запуск (Lect1Samples)
Building Lect1Samples-1.0-SNAPSHOT
-----
--- exec-maven-plugin:1.2.1:exec (default-cli) @ Lect1Sample
Введите радиус круга: R = 11.1
Для круга радиуса R = 11.100 площадь s = 387.076
```

Или в консоле:

```
>java -classpath Lect1Samples-1.0-SNAPSHOT.jar ru.vsu.cs.course1.Program
Введите радиус круга: R = 11.1
Для круга радиуса R = 11.100 площадь s = 387.076
```

Переменные и константы

Переменные

- Переменные — именованные ячейки памяти для хранения значений (можно представить квадратик на листе бумаги, рядом с которым написано его имя)

	...
r =	11.1
s =	387.076
	...

- На самом деле любые данные в памяти хранятся в виде нулей и единиц, однако для понимания происходящего такое представление вполне корректно

Просмотр переменных в режиме отладки

The screenshot shows the NetBeans IDE interface with a Java program being debugged. The main editor displays the source code of `Program.java` with several lines highlighted in red and green. The left sidebar shows the project structure and the 'main' method. The bottom panel displays the 'Variables' window, which lists the current state of variables at the execution point.

Annotations:

- указатель на строку, которая будет выполнена следующей**: Points to the green highlighted line 15 in the source code.
- ТОЧКИ ОСТАНОВКИ**: Points to the red highlighted lines 11, 13, and 19 in the source code.
- Выполнить одно действие (один шаг программы) в режиме отладки – F8**: Points to the 'Step Over' button in the top toolbar.
- СПИСОК ВЫЧИСЛЯЕМЫХ ВЫРАЖЕНИЙ**: Points to the list of expressions being evaluated in the 'Variables' window.
- СПИСОК ЛОКАЛЬНЫХ ПЕРЕМЕННЫХ (а также их значениями)**: Points to the list of local variables in the 'Variables' window.

Source Code (Program.java):

```
1 package ru.vsu.cs.course1;
2
3 import java.lang.Math;
4 import java.util.Locale;
5 import java.util.Scanner;
6
7
8 public class Program {
9     public static void main(String[] args) {
10         Locale.setDefault(Locale.ROOT);
11         Scanner scanner = new Scanner(System.in);
12
13         System.out.print("Введите радиус круга: R = ");
14         double r = scanner.nextDouble();
15         double s = Math.PI * r * r;
16
17         System.out.printf(
18             "Для круга радиуса R = %1$.3f площадь s = %2$.3f\n",
19             r, s
20         );
21     }
22 }
23
24
```

Variables Window:

Имя	Тип	Значение
<input checked="" type="checkbox"/> $\text{Math.PI} * r * r$	double	387.07563084879837
<input checked="" type="checkbox"/> $r * 10$	double	111.0
<Введите новый параметр наблюдения>		
<input type="checkbox"/> Статически		
<input type="checkbox"/> args	String[]	#79(length=0)
<input type="checkbox"/> scanner	Scanner	#231
<input type="checkbox"/> r	double	11.1

Отладка (Lect1Samples) 16:35/1:15

Присваивание значений переменной

- Значения переменных могут меняться в ходе выполнения программы (можно представить, что в клетку на листе бумаги вы можете записать новое значение, предварительно стерев старое – будем считать, что значения вы пишете карандашом)
- Например, следующая инструкция
 $r = 0.75;$
запишет в ячейку r значение 0.75
- Правильно читать « r присвоить 0.75», а не « r равно 0.75»

Объявление переменных

- Перед тем, как переменные использовать, их надо объявить.
- Переменные объявляются следующим образом (указывается тип и имя переменной):

```
int a; // a = 0 (значение по умолчанию)
```

- При объявлении переменной можно сразу же ее инициализировать каким-то значением:

```
double b = 1.5;
```

- Можно объявить сразу несколько переменных:

```
int c, d; // c = 0, d = 0
```

```
String
```

```
    s1 = "String 1",
```

```
    s2;
```

Типы данных

- Задают область допустимых значений переменных
- Определяют допустимые операции, которые можно проделать над переменными данного типа, и их семантику
- Описывают правила хранения данных в памяти (в виде нулей и единиц)

Более подробно о типах данных через несколько минут (слайдов)

Константы

- Константы можно понимать, как переменные, значения которых нельзя поменять во время выполнения программы (можно представить, как квадратики на листе бумаги, в которых значения записаны ручкой, т.е. нельзя стереть)
- Значения констант задаются при их объявлении:

```
final int N = 100;
```

```
final String HELLO = "Hello, World!";
```

- Иногда константы называют всеми большими буквами (чтобы по названию сразу было ясно, что это константа)
- Константы, как правило, объявляются в начале программы или функции
- Хорошим стилем считается, если в программе конкретные значения (например, конкретные числа: 10, 7.5 и т. п.) встречаются только в объявлениях констант

Идентификаторы в Java

- Идентификаторы — это названия переменных, функций, классов и т. д.
- Язык Java чувствителен к регистру символов программы, в том числе и к идентификаторам (А и а — разные переменные)
- Названия идентификаторам (переменным и т.д.) надо давать осмысленные, тогда программы будет самодокументированной
 - Не бояться «длинных» идентификаторов (linesCount, isPointInRect и т. п.)
 - Не стоит использовать в идентификаторах русские символы (Java допускает)
 - Самостоятельно прочитать соглашения по именованию идентификаторов языка Java

Что значит осмысленные имена идентификаторы?

- Для переменных (а также полей класса, параметров функций/методов):
 - Название переменной должно говорить о том, какие данные там записаны, примеры:
 - **r** – радиус
 - **index** – индекс (например, символа в строке)
 - **column** – столбец (как правило, номер столбца)
 - **rowCount** – кол-во строк
 - **i, j, k** – переменные цикла (исторически сложилось), часто удобнее использовать **r** и **c** (row и column)
- Для функций/методов:
 - Название должно говорить, что делает данная функция/метод:
 - **execute** – выполнить
 - **drawBorder** – нарисовать границу (например, фигуры, поля)
- Для классов:
 - Название должно говорить, что описывает данный класс:
 - **Point** – точка
 - **Triangle** – треугольник
 - **SimpleTableModel** – простая модель таблицы

Область видимости переменных

- Область видимости переменной — фрагмент программы, в котором данная переменная доступна для использования
- Для простоты пока можно считать, что область видимости переменной задается фигурными скобками (переменная доступна от места своего объявления до соответствующей закрывающейся фигурной скобки, включая вложенные области видимости)
- Нельзя объявлять переменную с тем же именем, которая уже доступна в данной области видимости (за исключением глобальных для функции переменных)
- **Переменные нужно объявлять в максимально узкой области видимости как можно позже (это позволяет предотвратить некоторые глупые ошибки)**
 - Не стоит объявлять все необходимые переменные заранее, как в Pascal'e

```
int a;    // a = 0
...
{
    int a1 = 10;
    ...
    {
        // нельзя:
        // int a1 = 10;
        int a2 = 20;
    }
}
...
{
    // можно:
    int a1 = 20;
    ...
}
```

Типы данных

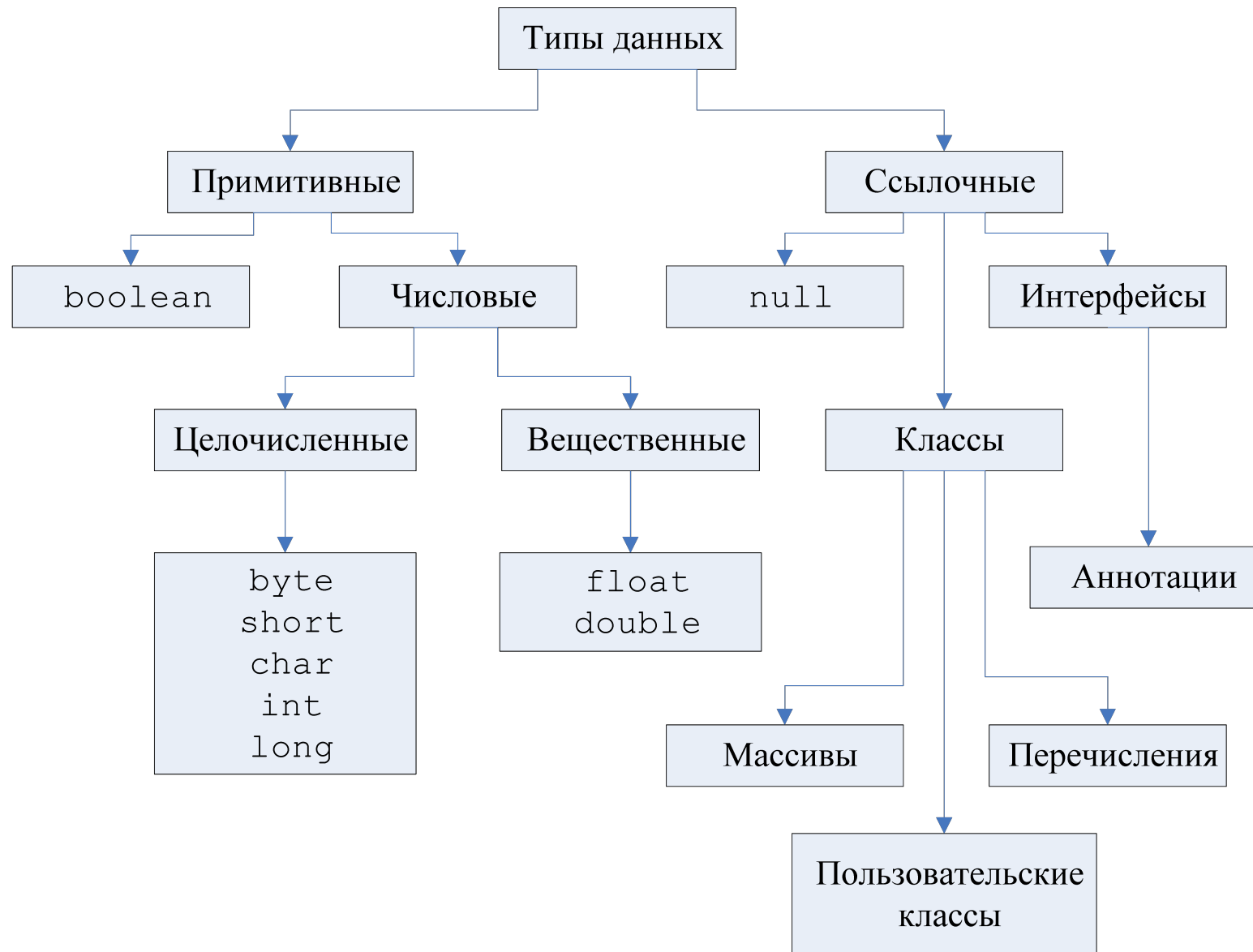
Типы данных

- Задают область допустимых значений переменных
- Определяют допустимые операции, которые можно проделать над переменными данного типа, и их семантику
- Описывают правила хранения данных в памяти (в виде нулей и единиц)

Типы данных Java

- Прimitives (не имеют собственных методов)
 - Числа
 - Целые (**byte**, **short**, **int**, **long**)
 - Вещественные (**float**, **double**)
 - Символы (**char**) – char может рассматривать (и использоваться) и как число от 0 до $2^{16} - 1$
 - Логические значения (**boolean**)
- Ссылочные или сложные – классы, интерфейсы и перечисления (требуют создания объекта перед использованием)
 - Строки (**String**)
 - Классы-обертки над примитивными типами (Byte, Short, Integer, Long, Character, Float, Double, Boolean)
 - Все остальные типы

Типы данных Java (в виде картинки)



Целочисленные типы данных

Тип	Размер (байт)	Диапазон
byte	1	от -128 до 127
short	2	от -32768 до 32767
char	2	беззнаковое целое число, представляющее собой символ UTF-16 (буквы и цифры)
int	4	от -2147483648 до 2147483647
long	8	от -9223372036854775808 до 9223372036854775807

Типы с плавающей точкой (вещественные типы)

Тип	Размер (байт)	Диапазон
float	4	от $-1.4\text{e-}45\text{f}$ до $3.4\text{e+}38\text{f}$
double	8	от $-4.9\text{e-}324$ до $1.7\text{e+}308$

Запомнить и осознать

- Не умничайте и используйте перечисленные ниже 5 типов (про остальные базовые типы забудьте, пока без них не сможете обойтись)
 - `int` – для целых чисел
 - `double` – для вещественных чисел
 - `String` – для строк
 - `char` – для отдельных символов (при необход.)
 - `boolean` – для логических значений

Строгая типизация

- Строгая типизация означает, что
 - Для любого выражения в программе можно формально вывести его тип
 - При компиляции программы весь код (все операции) проверяется на совместимость или возможность преобразования типов, несовместимость считается ошибкой
- Строгая типизация позволяет:
 - Находить некоторые типы ошибок на этапе компиляции (т.е. разработки программы)
 - Генерировать более оптимальный код
- Строгая типизация делает программы более многословными и не допускает применения некоторых приемов, доступных в языках с динамической типизацией

Типы выражений в примере

```
Scanner scanner = new Scanner(System.in);
```

`System.in` → `java.io.InputStream` (чтобы узнать – смотреть документацию)

`new Scanner(System.in)` → `java.util.Scanner`

```
System.out.print("Введите радиус круга: R = ");
```

`"Введите радиус круга: R = "` → `String` (`java.lang.String`)

`System.out.print("Введите радиус круга: R = ")` → `void` («ничто»,
«отсутствие значения»)

```
double r = scanner.nextDouble();
```

`scanner` → `java.util.Scanner` (переменная `scanner` объявлена типа `Scanner`)

`scanner.nextDouble()` → `double`

Арифметические операторы (над числами)

Оператор	Описание	Пример
+	Складывает значения по обе стороны от оператора	A + B даст 30
-	Вычитает правый операнд из левого операнда	A - B даст -10
*	Умножает значения по обе стороны от оператора	A * B даст 200
/	Оператор деления делит левый операнд на правый операнд	B / A даст 2
%	Делит левый операнд на правый операнд и возвращает остаток	B % A даст 0
++	Инкремент - увеличивает значение операнда на 1	B++ даст 21
--	Декремент - уменьшает значение операнда на 1	B-- даст 19

Операторы сравнения (над числами)

Оператор	Описание	Пример
'=='	Проверяет, равны или нет значения двух операндов, если да, то условие становится истинным	(A == B) — не верны
!='	Проверяет, равны или нет значения двух операндов, если значения не равны, то условие становится истинным	(A != B) — значение истинна
>	Проверяет, является ли значение левого операнда больше, чем значение правого операнда, если да, то условие становится истинным	(A > B) — не верны
<	Проверяет, является ли значение левого операнда меньше, чем значение правого операнда, если да, то условие становится истинным	(A < B) — значение истинна
>=	Проверяет, является ли значение левого операнда больше или равно значению правого операнда, если да, то условие становится истинным	(A >= B) — значение не верны
<=	Проверяет, если значение левого операнда меньше или равно значению правого операнда, если да, то условие становится истинным	(A <= B) — значение истинна

Побитовые операторы (над целыми числами)

Оператор	Описание	Пример
& (побитовое и)	Бинарный оператор AND копирует бит в результат, если он существует в обоих операндах.	(A & B) даст 12, который является 0000 1100
 (побитовое или)	Бинарный оператор OR копирует бит, если он существует в любом из операндов.	(A B) даст 61 который равен 0011 1101
^ (побитовое логическое или)	Бинарный оператор XOR копирует бит, если он установлен в одном операнде, но не в обоих.	(A ^ B) даст 49, которая является 0011 0001
~ (побитовое дополнение)	Бинарный оператор дополнения и имеет эффект «отражения» бит.	(~ A) даст -61, которая является формой дополнением 1100 0011 в двоичной записи
<< (сдвиг влево)	Бинарный оператор сдвига влево. Значение левых операндов перемещается влево на количество бит, заданных правым операндом.	A << 2 даст 240, который 1111 0000
>> (сдвиг вправо)	Бинарный оператор сдвига вправо. Значение правых операндов перемещается вправо на количество бит, заданных левых операндом.	A >> 2 даст 15, который является 1111
>>> (нулевой сдвиг вправо)	Нулевой оператор сдвига вправо. Значение левых операндов перемещается вправо на количество бит, заданных правым операндом, а сдвинутые значения заполняются нулями.	A >>> 2 даст 15, который является 0000

Тип результата для бинарных операторов

Оператор	Тип левого операнда A	Тип правого операнда B	Тип выражения: A Оператор B
+ - * / %	int	int	int
+ - * /	int	double	double
+ - * /	double	int	double
+	X	String	String
+	String	X	String
Битовые: << >> &	int	int	int
Логические: &&	boolean	boolean	boolean
Логические: < <= > >= == !=	Int, double, char	int, double, char	boolean

Тип результата для унарных операторов

Оператор	Тип правого операнда A	Тип выражения: Оператор A
Битовые: ~	int	int
Логические !	boolean	boolean

Приоритеты операторов

Высший приоритет

() [] .

++ -- ~ !

* / %

+ -

>> >>> <<

== !=

&

^

|

&&

||

= op=

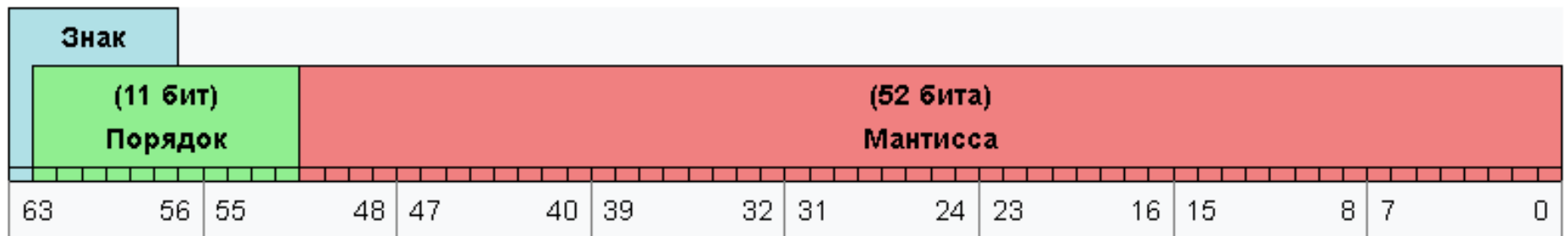
Низший приоритет

Особенности вещественных чисел (чисел с плавающей точкой)

Вещественные типы данных.

Сравнение на равенство.

- Представление в памяти double



```
Console.WriteLine(9876543210.0123456789);           // 9876543210.01235
```

```
Console.WriteLine(0 == 1E-324);                       // true
```

```
Console.WriteLine(105.3256 - 105.32 == 0.0056);      // false
```

- Вычисления накапливают ошибку в дальних знаках после запятой, поэтому корректно сравнивать вещественные числа с заданной точностью, например, так:

```
Math.abs(a - b) < 1e-10
```

Объяснение, почему $105.3256 - 105.32 \neq 0.0056$

```
// Вещественные числа представлены в памяти в двоичном виде:  
// часть битов определяет несколько значащих знаков, часть - порядок,  
// т.е. положение этих знаков относительно запятой.  
// Это означает, что никаких точных десятичных значений в double не бывает  
// - только приближенные
```

```
double a = 105.3256, b = 105.32, c = 0.0056;  
System.out.println(a - b == c);    // напечатает false  
// объяснение  
System.out.printf("%f - %f = %f\n", a, b, a - b);  
System.out.printf("%.20f:%n %s\n", a, doubleToBinaryStr(a));  
System.out.printf("%.20f:%n %s\n", b, doubleToBinaryStr(b));  
System.out.printf("%.20f:%n %s\n", a - b, doubleToBinaryStr(a - b));  
System.out.printf("%.20f:%n %s\n", c, doubleToBinaryStr(c));
```

```
false  
105.325600 - 105.320000 = 0.005600  
105.3256000000000000000000:  
    0100000001011010010101001101011010100001011000011110010011110111  
105.3200000000000000000000:  
    0100000001011010010101000111101011100001010001111010111000010100  
0.0056000000000000116000:  
    0011111101110110111100000000011010001101101110001100000000000000  
0.0056000000000000000000:  
    0011111101110110111100000000011010001101101110001011101011000111
```


Запомнить и осознать

- Вещественные числа в `double` всегда приближительны (в десятичном представлении только 15-16 знаков значащих)
- Вычисления дополнительно накапливают ошибку в младших значащих знаках, поэтому сравнивать `double` числа оператором равенства (`==`) нельзя
- `double` числа, полученные в вычислениях, можно сравнивать только с заданной точностью, например, так:

`Math.abs(a - b) < 1e-10`

double NaN и Infinity

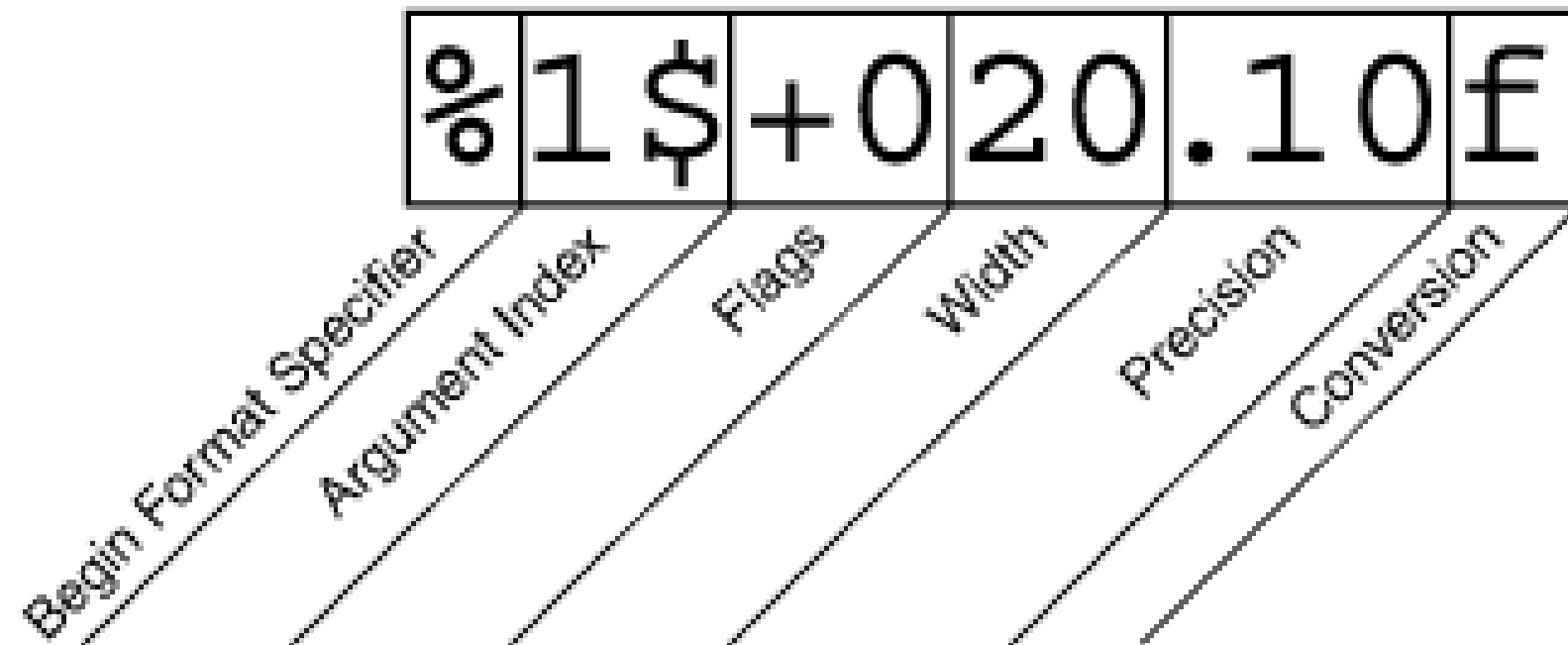
- При некорректных вычислениях в double, например, делении на 0, вопреки распространенному мнению не происходит ошибок, а возникают специальные значения NaN (Not a Number) и Infinity
- Это надо учитывать и дополнительно проверять получившийся результат вычислений в случае возможности некорректных данных

```
double a2 = 100;  
double b2 = 0;  
double b2 = 0; c2 = a2 / b2;  
System.out.println(c2);           // Infinity  
System.out.println(c2 == c2);     // true  
System.out.println(Double.isInfinite(c2)); // true
```

```
double c3 = Math.sqrt(-1);  
System.out.println(c3);           // NaN  
System.out.println(c3 == c3);     // false  
System.out.println(Double.isNaN(c3)); // true
```

Форматированный вывод:
String.format() и printf()

Общий вид форматной подстановки



Совет (настоятельная рекомендация)

- Обязательно смотрите проекты-примеры к лекции (и экспериментируйте с ними):
 - Lect2Samples