5342 users solved this topic. Latest completion was **about 3 hours ago**.

# Theory: Floating-point types

16 minutes 5 / 14 problems solved

Start practicing

A decimal numeral refers generally to the notation of a number in the decimal system, which contains a separator (for example 0.01 or 3.1415). Such numbers represent fractions which are often used in science, statistics, engineering, and many other fields.

Java has two basic types to represent decimal numbers: `float` and `double`. They are called **floating-point types**. In fact, these types cannot represent an arbitrary decimal number, because they support only a limited number of significant decimal digits (6-7 for `float`, and 14-16 for `double`). In addition, `double` can represent a wider range of numbers than `float`.

In practice, programmers mostly use the `double` type and we recommend you to use it for our code challenges. But all the information below is valid for the `float` type as well.

Current topic:
✓ Floating-point types Stage 5
Topic depends on:
✓ Integer types and operations Stage 1
Topic is required for:
✓ Infinity and Not a Number
✓ Numeric literals
Math library Stage 5

## §1. Declaring variables and assigning values

In a program, the `double` literal looks like `5.2` where the dot character separates the integer and fractional parts of a number.

There are several initialized `double` variables below:

```
double zero = 0.0;
double one = 1.0;
double negNumber = -1.75;
double pi = 3.1415;

System.out.println(pi); // 3.1415 or 3,1415 depending on computer settings
```

It is also possible to assign an integer value to a `double` variable:

```
double one = 1;
System.out.println(one); // 1.0 or 1,0
```

If you want to declare and initialize a `float` variable, you should mark the assigned value with the special letter `f` (float literal):

```
float pi = 3.1415f;
float negValue = -0.15f;
System.out.println(pi); // 3.1415 or 3,1415 without f
```

Both types can store only a limited number of significant decimal digits:

```
float f = 0.888888888888888888f; // a value with a lot of decimal digits
System.out.println(f);           // it only prints 0.8888889
```

Floating-point types have a peculiar way to mark values with a mantissa:

```
double eps = 5e-3; // means 5 * 10^(-3) = 0.005
double n = 0.01e2; // means 0.01 * 10^2 = 1.0
```



## §2. Arithmetic operations

You can perform all types of arithmetic operations with floating-point types.

```
double one = 1.0;
double number = one + 1.5; // 2.5

double a = 1.75;
double b = 5.0;
double c = b - a; // 3.25

double pi = 3.1415;
double squaredPi = pi * pi; // 9.86902225
```

For `double` and `float` operands, the operator `/` performs a division, not an integer division.

```
System.out.println(squaredPi / 2); // 4.934511125
```

Pay attention to an important thing that beginners often miss:

```
double d1 = 5 / 4; // 1.0
double d2 = 5.0 / 4; // 1.25
```

In the first case, we perform integer division that produces an integer result and then assign the result to `d1`. In the second case, we perform a real division that produces double value and then assign the value to `d2`.

## §3. Errors during computations

Be careful! Operations with floating-point numbers can produce an inaccurate result:

```
System.out.println(3.3 / 3); // prints 1.0999999999999999
```

Errors can accumulate during computation. In the following example we calculate the sum of ten decimals 0.1:

```
double d = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1;
System.out.println(d); // it prints 0.9999999999999999
```

In the following lessons, we will find out how to deal with this issue. For now, just take it into consideration.

If you want to learn more about floating-point numbers, you may read this article as an addition.

## §4. Reading floating-point numbers

You can use `Scanner` to read the values of both of these types from the standard input.

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

float f = scanner.nextFloat();
double d = scanner.nextDouble();
```

We recommend you use `double` to solve our programming problems.

As an example, consider a program that calculates the `area` of a triangle. To find it, the program reads the `base` and the `height` from the standard input, then multiplies them, and divides by 2. Note that the base and the height are perpendicular to each other.

```java
import java.util.Scanner;

public class AreaOfTriangle {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        double base = scanner.nextDouble();
        double height = scanner.nextDouble();


        double area = (base * height) / 2;



        System.out.println(area);

    }

}
```

Let's calculate the area of a triangle with a base of 3.3 meters and a height of 4.5 meters.

Input 1:

```
3.3 4.5
```

Output 1:

```
7.425
```

As you can see, it's area is 7.425 square meters!

Keep in mind that the output of this program may have a lot of zeros like the output below because an operation with floating-point numbers can produce inaccurate results.

Input 2:

```
2.2 4.01
```

Output 2:

```
4.4110000000000005
```

It is possible to round or format a double result, but we will not do it in this lesson. In the coding problems, output a result as-is.

## §5. The decimal separator

If you solve our coding problems locally or try to repeat our examples, you may encounter a problem with your computer having different locale settings. In this case, the **Scanner** cannot read floating-point numbers with the dot character (**3.1415**). Try to input numbers written with the comma separator (**3,1415**).

If you want to use the dot character without modifying your local settings, try using the following code to create a scanner:

```java
Scanner scanner = new Scanner(System.in).useLocale(Locale.US);
```

You do not need to change anything when submitting your solutions in our forms.

Report a typo

**423** users liked this theory. **10** didn't like it. **What about you?**

Start practicing

Comments (8) Hints (0) Useful links (0) Show discussion