Nuances of working with real numbers

1. Rounding real numbers

As we have already discussed, when a real number is assigned to an `int` variable, it is always rounded down to the nearest smaller integer — the fractional part is simply discarded.

But it's easy to imagine a situation when a fractional number needs to be rounded to the nearest integer in either direction or even rounded up. What do you do in this case?

For this and for many similar situations, Java has the `Math` class, which has the `round()`, `ceil()`, and `floor()` methods.

`Math.round()` method

The `Math.round()` method rounds a number to the nearest integer:

```
long x = Math.round(real_number)
```

But there's another nuance here: this method returns a `long` integer (not an `int`). Because real numbers can be very large, Java's creators decided to use Java's largest available integer type: `long`.

Accordingly, if a programmer wants to assign the result to an `int` variable, then she must explicitly indicate to the compiler that she accepts the possible loss of data (in the event that the resulting number does not fit into an `int` type).

```
int x = (int) Math.round(real_number)
```

Examples:

Statement	Result
`int x = (int) Math.round(4.1);`	4
`int x = (int) Math.round(4.5);`	5
`int x = (int) Math.round(4.9);`	5

`Math.ceil()` method

The `Math.ceil()` method rounds a number **up** to an integer. Here are examples:

Statement	Result
`int x = (int) Math.ceil(4.1);`	5
`int x = (int) Math.ceil(4.5);`	5
`int x = (int) Math.ceil(4.9);`	5

`Math.floor()` method

The `Math.floor()` method rounds a number **down** to an integer. Here are examples:

Statement	Result
`int x = (int) Math.floor(4.1);`	4
`int x = (int) Math.floor(4.5);`	4
`int x = (int) Math.floor(4.9);`	4

Of course, when rounding a number **down** to an integer, it's easier to simply use a type cast operator: `(int)`

Statement	Result
`int x = (int) 4.9`	4

If you find it difficult to remember these names, a short English lesson will help:

- `Math` means mathematics
- `Round` means round
- `Ceiling` means ceiling
- `Floor` means floor

2. How floating-point numbers are structured

The `double` type can store values in the range from `-1.7*10^308` to `+1.7*10^308`. This huge range of values (compared with the `int` type) is explained by the fact that the `double` type (as well as `float`) has a completely different internal structure than integer types. Internally, the `double` type encodes its value as two numbers: the first is called the *mantissa*, and the second is called the *exponent*.

Let's say we have the number `123456789` and store it a `double` variable. When we do, the number is converted to `1.23456789*10^8`, and internally the `double` type stores two numbers — `23456789` and `8`. The significand ("significant part of the number" or mantissa) is highlighted in red, while the exponent is highlighted in blue.

This approach makes it possible to store both very large numbers and very small ones. But because the number's representation is limited to 8 bytes (64 bits) and some of the bits are used to store the *exponent* (as well as the sign of the mantissa and the sign of the exponent), the maximum digits available to represent the *mantissa* is **15**.

This is a very **simplified** description of how real numbers are structured.

---

3. Loss of precision when working with real numbers

When working with real numbers, always keep in mind that *real numbers* **are not exact**. There may always be *rounding errors* and *conversion errors* when converting from decimal to binary. Additionally, the most common source of error is *loss of precision* when adding/subtracting numbers on radically different scales.

This last fact is a little mind-blowing for novice programmers.

If we subtract `1/10^9` from `10^9`, we get `10^9`.

Subtracting numbers on radically different scales	Explanation
1000000000.000000000; - 0.000000001; 1000000000.000000000;	The second number is **extremely small**, which will cause its significand (highlighted in gray) is be ignored. The 15 significant digits are highlighted in orange.

What can we say, programming isn't the same as mathematics.

---

4. Pitfall when comparing real numbers

Another danger lies in wait for programmers when they compare real numbers. It arises when working with real numbers, because round-off errors can accumulate. The result is that there are situations when real numbers are expected to be equal, but they are not. Or vice versa: the numbers are expected to be different, but they are equal.

Example:

Statement	Explanation

`double a = 1000000000.0;`
`double b = 0.000000001;`
`double c = a - b;`
	The value of the variable `a` will be `1000000000.0`
	The value of the variable `c` will be `1000000000.0`
	(the number in the `b` variable is excessively small)

In the above example, `a` and `c` should not be equal, but they are.

Or let's take another example:

Statement	Explanation
`double a = 1.00000000000000001;` `double b = 1.00000000000000002;`	The value of the variable `a` will be `1.0` The value of the variable `b` will be `1.0`

5. An interesting fact about `strictfp`

Java has a special `strictfp` keyword (***strict floating point***), which is not found in other programming languages. And do you know why you need it? It worsens the accuracy of operations with floating-point numbers. Here's the story of how it came to be:

Java's creators:

We really want Java to be super popular and to run Java programs on as many devices as possible. So we made sure that the specification for the Java machine says that all programs must run **the same way** on all types of devices!

Makers of Intel processors:

Hey, everyone! We have improved our processors, and now all real numbers are represented using 10-bytes instead of 8-bytes inside our processors. More bytes means more significant digits. What does that mean? That's right! Now your scientific calculations will be even more accurate!

Scientists and everyone involved in ultra-precise calculations:

Cool! Well done. Excellent news!

Java's creators:

No-no-no, you guys! We already told you that all Java programs must run **the same on all devices**. We're going to forcibly disable the ability to use 10-byte real numbers inside Intel processors.

Now everything is fine again! Don't thank us.

Scientists and everyone involved in ultra-precise calculations:

Have you gone entirely insane? Quickly get everything back as it was!

Java's creators:

Guys, this is for your own good! Just imagine: all Java programs run **the same way on all devices**. That's so cool!

Scientists and everyone involved in ultra-precise calculations:

No. It's not cool at all. Quickly put everything back how it was! Or do know where we'll put your Java?

Java's creators:

Hmm. Why didn't you say so right away? Of course, we'll put it back.

We restored your ability to use all the features of the latest processors.

By the way... We also specially added the `strictfp` keyword to the language. If you write it before the name of a function, then all the operations involving real numbers inside that function will be equally bad on all devices!