Assignment 1

Handout: **Monday, 13 September 2021**
Due: **23:59:59, Wednesday, 22 September 2021**

Goals:

- To review how floating-point numbers are encoded in binary;
- To understand better the range and precision of different numeric data types;
- To practice the use of control structures;
- To get used to the IntelliJ IDEA IDE;

NOTE

Java SE Development Kit Version 11[1] and IntelliJ IDEA Community Edition Version 2021.2[2] will be used in grading your assignments. Make sure you use the same versions of tools for your development.

[1] https://www.oracle.com/hk/java/technologies/javase-jdk11-downloads.html

[2] https://www.jetbrains.com/idea/download/other.html

1. Floating Point Vs. Integer Numbers (40 points)

Suppose we have a new data type called `miniFloat`, which is like `float` but uses only eight bits. From left to right, the meaning of the bits is as the following:

- 1 bit for the sign: 0 for positive values and 1 for negative values;
- 4 bits in two's complement for the exponent;
- 3 bits for the mantissa;

Consider for example a `miniFloat` value with bit sequence `00100110`. The value is positive (`0`), the exponent is 4_{10} (= 0100_2), and the significand is 1.75_{10} (=1.110_2). Therefore, the whole value is $1.75 \times 2^4 = 28$.

What to do:

[Task 1] Complete method `MiniFloat.miniFloatFromString`;
Method `miniFloatFromString` takes a `String` of eight characters, each being '1' or '0', and returns the value of the corresponding `miniFloat` (as a `float` value);

[Task 2] Complete method `MiniFloat.numIntegralMiniFloats`;
Method `numIntegralMiniFloats` returns the number of all integral `miniFloat` values, i.e., `miniFloat` values that are integers.

Note:

- The absolute value of a `miniFloat` is always calculated using formula $1.\text{mantissa} \times 2^{\text{exponent}}$, with no exceptions.
- Method `getValidMiniFloatBitSequences` returns all the 256 (=2^8) `miniFloat`s in `String`.
- You may add auxiliary methods to class `MiniFloat`, but you may not change the signatures of methods `miniFloatFromString` and `numIntegralMiniFloats`.
- A few methods from class `String` may be used here. A detailed documentation of the class is at https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html.

2. Special Numbers (20 points)

We consider positive numbers that have exactly three different prime factors as being *special*. For example, 30 is special because $60=2^2*3^1*5^1$, while 210 is not special because $210=2^1*3^1*5^1*7^1$.

What to do: In SpecialNumber.java

[Task 3] Complete method isSpecial in class SpecialNumber so that the method returns true if and only if the argument is special.

Note:

- You may add auxiliary methods to class SpecialNumber, but you may not change the signature of method isSpecial, and your code is not allowed to invoke any other methods outside the class.

3. Balanced Brackets (20 points)

A non-empty String containing only six characters, namely '(', ')', '{', '}', '[', and ']', is called *balanced*, if the brackets can be paired into "()"s, "[]"s, and/or "{}"s without changing their positions. For example, "()", "(){}", "({})", and "(()[{}])" are balanced, but "((", "({)}", and "(){" are not.

What to do: In BalancedBrackets.java

[Task 4] Complete method isBalanced in class BalancedBrackets so that the method returns true if and only if the argument String i) is non-empty, ii) contains only the six characters, and iii) is balanced.

Note:

- You may add auxiliary methods to class BalancedBrackets, but you may not change the signature of method isBalanced.

Tests:

You may right click on class MiniFloatTest and select "Run 'MiniFloatTest'" to execute the tests we prepared for class MiniFloat. Similarly, you can also execute the tests we prepared for classes SpecialNumber and BalancedBrackets. If any test fails, your implementation contains bug(s). Note, however, that, since the tests only check a small number of input/output pairs, passing all the tests do *not* mean your implementation is correct.

What to hand in:

The whole **Assignment1** folder with the completed methods in a ZIP file.