2IP67.5 – Instruction 2

Preliminaries

Exercises, or parts, that are marked with a ***** are extra material for the interested student, probably more difficult and not part of the main stream.

Remember:

- Start every file you write with a line of comment with your name(s) and the date.
- Work in pairs, if possible.
- Keep your code neat from the beginning on, with proper indentation, etc.
- Have each exercise checked by one of your instructors.

2.1 Repetition

1. Consider the following class:

```
1  class Repetition {
2      int k;
3      int sum;
4
5      void computeSum() {
6          k = 0;
7          sum = 0;
8          while (k<=10) {
9              sum = sum + k;
10             k = k+1;
11         }
12         System.out.println("The sum is "+sum);
13     }
14 }
```

2. Use this class as an inspiration to write a program (including a main method etc.) that computes the *factorial* (Dutch: *faculteit*) of 12, that is, $1 * 2 * 3 * \ldots * 12$.

3. Modify this program such that it asks for a number and then computes the factorial of this number.

4. Write a program that reads positive numbers until you enter a 0. Then it should print the product of these numbers (without the last 0). What does your program do with negative numbers? Make a sensible decision.

5. Extend this program such that it also prints the greatest number of the sequence. You may assume that the sequence is not empty.

6. And have it also print the smallest number. Is this harder?

2.2 Number Crunch

2.2.1 Maxing out

The built-in numeric types of Java are limited in the size of the numbers you can store. Try this by adding two large integer numbers, e.g., 1234567 and 1234567.

1. We want to find the largest possible int experimentally. As you have seen, two positive numbers of which the sum is too large for an int, results in a *negative* number. Use this information to write a loop to find the largest possible int.
2. Do the same for the largest (or smallest, if you want) negative number.
3. These values can be obtained directly with the expressions `Integer.MAX_VALUE` and `Integer.MIN_VALUE`.
4. * As you see, these values are not each other's opposite. Try to find a reason for this. In your head or on the web (e.g., the wikipedia article on *two's complement*).

2.2.2 Floating Around

The floating point types in Java (float and double) can store much larger values than int can, but this comes at a price: they have limited *precision*.

1. What happens when you add 1 to a very large `double` number?
2. There are not only gaps between large numbers, but also gaps between smaller numbers. There are a lot of real numbers that cannot be represented with Java floating point numbers. Write a program that calculates $x + 0.2 - x$ for $x = 1$. Try it also for $x = 0$ and $x = 10$ and for other values than 0.2, e.g., 0.5 and 0.1. Can you explain this?
3. Write a program that calculates $x + 0.2 - x$ for $x = 0, 1, 2, \ldots, n$ for $n = 50$.
4. Do you see a pattern? Experiment more with this program. Improve the "visualization" by printing only a "+" sign when the result is greater than .2 and "–" when it is smaller and use larger numbers for n. Print the pluses and minuses after one another, on one line, or print a new line when the sign changes. Print the values for i where the sign changes. Can you explain (a bit of) this?

2.3 The Game of Nim

Nim is a game for two players with simple rules. It is played all over the world with matches, straws, or anything that is at hand and can serve as playing tokens. Many versions exist, though all share the idea that the tokens are divided in one or more piles and that each player in turn takes a number of tokens from a pile. The player who takes the last token wins, or loses, depending on the rules. The name comes from the German word "Nimm", meaning "take".

We adopt the following version. The tokens are divided in at least two piles. A player takes an arbitrary number of tokens (at least one) from one of the piles. Last player wins.

1. Start with a program that plays one move in a game for two human players. Write a class Nim with a method `play` that plays the game. Declare an array `piles` of integers that represents the piles. Start with three piles of seven tokens. Print the numbers in the pile and ask the player from which pile he wants to draw and then how many tokens he wants to take. Execute his move and print the new situation.

   Write a demo class and try it out.

2. Now turn this into a game by putting a loop around the code you wrote in part 1. When should this loop end? Declare a variable that keeps track of (the number of) the player (1 or 2) that is to move. Announce the winner at the end of the game.

3. Have the game start with a random number of tokens per pile between 3 and 10. Use the command `Math.random()`. This command gives you a random double between 0 and 1 (including 0, excluding 1). Note that `(int) r` rounds down the double `r` and turns it into an int. So for instance

   ```
   double r = Math.random();
   int x = (int)(8*r);
   ```

   gives you a random *int* between 0 and 7.

4. * Add some checks on the input of the human player (allowed pile number, allowed number of tokens). How does your program react to incorrect input?

5. * Starting with a random number of piles is more difficult. Explain why.

Kees Huizing – c.huizing@tue.nl – 8 Sep, 2009