Worksheet 2

Fun with funs

Collatz returns !

Remember what we did from slide 9 to 14, turning the implementation idea from slide 13 ("Implementing the body") into a proper Java function ?

Look at the code from slide 5 giving a solution for the last exercise of practice 1 printing a Collatz sequence from a given number.

- identify the *interface* of this procedure: what argument(s) does it need ? what are their types ?
- it's a procedure: what will its return type be ?
- can you describe what this function is doing ? Find a good name for it.
- now turn this one-time snippet into a proper function

Now you can easily visualize Collatz sequences from any input number you want. Use it to manually check that the Collatz conjecture holds for all integers between `1` and `10`.

Namespace occupation

The euclidian norm in a 2D-space ("plane") can be defined as a function of the coordinates (x_A, y_A) of a given point A.

$$|A| = \sqrt{x_A^2 + y_A^2}$$

Implement this as a function with signature: `double norm(double x, double y)`. You may have to browse the documentation to find functions to compute the square and square root of a given input number.

Test it on some values where you know the result in advance: the origin `(0, 0)`, the unit on respectively the x and y axis `(1, 0)` and `(0, 1)`, the opposite corner from the origin on the unit square `(1, 1)` (remember that $\sqrt{2} \approx 1.414$).

Now, let's define the same function but for a 3D-space. Can you keep the same name ? What do you suggest ?

Objects

Sharing the namespace

2D-Points

- Declare a simple class `Point2D` to model a geometric point in the plane (2D space). What fields must it contain ? What type will you use for them ?
- Add a `norm` method to compute the same metric as previously. How will the signature differ from the signature of the function defined above ?

3D-Points

- Now create a new class to represent points in a 3D space.
- Add the corresponding `norm` method to it too. Is there any naming conflict ?

Pass by value / pass by reference

- Declare an `short` value `age` and assign it to `4`.
- Create a (`void`) function called `increment` that increases the value of its only (`short`) argument by `1` (modifies it without returning it).
- Apply it on `age`. Did you get any error ? Was `age` modified ?

A native value is nothing more than a (temporary) value in the processor. There's no memory location available. When a function is called in Java, it is passed a copy of its arguments. They are "passed by value". Passing by value is simpler and closer to the mathematical intuition of functions, but it can be costly for huge data structures.

We're now going to define a special wrapper for the `short` type where we have access to the inner value : `RWShort`. This class should have one field of type `short` named `value`.

- Declare a `RWShort` value called `rwAge` holding the same value `4`.
- Modify `increment` so that it now works on `RWShort` and not `short`
- Use it on the `rwAge`. Is it modified now ? Why ?

Java passes arguments by value; but since the value of objects are their reference in memory, this actually amounts to passing by reference ! We get the best of both worlds.

Automatically numbering instances with `static`

- Declare a class called `Serial` having only one `long` field and one constructor taking as argument the number to "wrap" as a `Serial`.
- Now modify it to add a `static` field initialized to `0`.

- Remove the argument to the constructor, and use the value of the `static` field instead, while incrementing it. The `static` field is keeping track of the highest number and "magically" guarantees to generate distinct numbers.
- Complete the class with an `.olderThan` method that takes another `Serial` as argument and is able to return `true` if and only if it was created after the target object (the one on which the method is applied).

This pattern can be very useful to keep track of objects created throughout an application.

Executable code

You will now create your first command-line program.

- Create an empty `Main` class holding the structure for an empty program (draw inspiration from the last section of the slides).
- Now add a method to this class using the code from the function defined in the first exercise ("Collatz returns !"). Should this method be `static` or not ?
- Which `static` method from the `Integer` class lets one convert a `String` to an `int` value ? Read its documentation to understand what happens when its argument doesn't represent a number.
- Modify the `main` function of your class to
  - make sure only one argument is passed to your program
  - convert it to an `int` and don't forget to handle the errors which may arise
  - pass it the collatz method you've imported
- Compile and run your program from a shell: you now have an executable which can display the Collatz sequence from a number passed to it, and it doesn't require its users to type Java or to know anything about the `jshell`.