Worksheet 4

Memoization

Naive Fibonacci

- As a reminder, the Fibonacci sequence is defined by

$$f_0 = 1$$
$$f_1 = 1$$
$$f_{n+2} = f_{n+1} + f_n$$

  implement a function returning the $\texttt{n}^{\text{th}}$ term of the sequence with the following signature:

```
long fibonacci(int n);
```

- Test your function on "small" inputs (< 10). Now use it to find the value of f_{40} and note the time it takes. Try `45` if you're feeling particularly daring today. What is going on?

Let's add some logging

- Modify the previous function to log its argument each time it is called.
- Retry the previous calls with the small values. What do you notice?

Keeping answers

A simple way to address the issue we have just evidenced is *memoization*: we trade a little space (in memory) against some time (to run the algorithm). If our program could somehow "remember" the answers it gives it wouldn't have to compute the same sums over and over.

- We'll need a place to store terms, so let's create a class `Fibonacci` with a `static` array named `cache` to hold all the terms already found. Regarding its type, as we've seen, terms around f_{40} are already quite large, so we are likely to hit the `long` overflow if we consider terms for indices too high so may want to limit ourselves to a hundred terms, and dimension this cache accordingly. Don't forget to initialize the first two cells of the cache

for f_0 and f_1 (hint: the value for this `static` field could be returned by a `static` function).
- Add your previous function as a `static` method of this new class, renaming it from `fibonacci` to `get` (so we can write `Fibonacci.get` and it kind of makes sense).
- Now modify `get` to check if the result is missing from the cache first, compute the sum and store it only in that case, and finally return the content of the cache.
- Compute values you've computed before to check the results. What do you observe regarding logging messages? In particular, what happens when you compute the same term twice?

Graphical User Interface

Let us now create a graphical widget to expose the model of the Fibonacci sequence we have previously implemented.

The Window

- Starting from the Swing example in the class, create a new file `FiboGUI.java` containing a `FiboGUI` class to hold the be the entry point of your program. Make it create an empty window for now.
- As in the example on the slide, create a second `static` method called `createAndShowGUI` to populate the window.
- Add a `JPanel` to structure the view and use a `GridLayout` for it as in the example in the slides (2×2 should be enough). Remember that adding components should go before calling `.pack()`.
- Then add two components to the `JPanel`:
  - a `JLabel` to print a message describing the expected input
  - a second `JLabel` to print the result

Number input

Now we'll create a `JSpinner` to let the user pick a number. This requires quite some work so instead of creating it directly inline, we will create a separate class named `FiboInput` to handle it. Create a new file holding this class which should inherit `JSpinner`.

- The constructor for the parent class `JSpinner` requires a `SpinnerModel`. We will use the `SpinnerNumberModel` which requires four `Integer` values:
  - an initial value
  - a minimum value
  - a maximum value
  - a step value to say by how much to increase or decrease the current value each time an arrow is pressed

- However, when calling `super` in a constructor, it should be the first line so we'll need to create our `SpinnerModel` in one fell swoop. Add a `static` method called `getRange` to return a `SpinnerModel`. It should create four `Integer` variables as above, initialize them with sensible values, and return a call to the constructor for `SpinnerNumberModel` on the four previous variables. Why can the method return type `SpinnerModel` and not `SpinnerNumberModel` (type returned by the call to the constructor)?
- Implement `FiboInput`'s constructor by simply passing the result of `getRange` to the `super` constructor.
- We can go back to `createAndShowGUI`, add a `FiboInput` variable called `input`. Add it (as in `panel.add(...)`) between both `JLabel`s (intuitively, I expect the input to be next to the label describing it and the label presenting the result to come after, but this is my layout choice, you can design your application differently).

Wiring the logic

- Create a file named `Fibonacci.java` in the same directory as your application and fill it with the previous class for your memoized code (you can remove the debug lines, we don't need them any more).
- We will now need to modify the `FiboInput` to react to input: update the header of the class to make it implement the `ChangeListener` interface. In order to respect this "contract", a class needs to have a `public void stateChanged(ChangeEvent e)` method. Add one to the class and have it log the event `e` to the console. Register it to the Swing engine by calling the `addChangeListener` on `super` in the constructor. What argument should be passed to this method?
- Recompile your application, run it, play with the number input and look at the lines getting logged to the console.
- Now we'd like something more useful to happen when the value is changed: we'll need access to the `JLabel output` from an instance of `FiboInput`. Add a corresponding (`private`) field to the class and an argument to the constructor which you'll use to set it. Update the code of the `stateChanged` handler to write something to `this.output`. Don't forget to update `createAndShowGUI` in `FiboGUI` to call the modified constructor for `FiboInput` properly.
- All that remains is to have the handler use `Fibonacci.get` from the first part. Retrieve the rank from the value of the `FiboInput` (remember: `FiboInput` is a `JSpinner`, we made it inherit the class for this reason! what method could be useful to retrieve its current value? read the documentation carefully: what type does it return? what could we do about it ?). Pass it to `Fibonacci.get`, convert the (`long`) result to `String` and set the output to that.

Recompile everything. We got ourselves a simple but efficient Fibonacci sequence calculator!