Short answers for the 2nd exercise session

Exercise 1

- *Do you need to manage several Pascal's triangle represented as separate objects ?*
  As said in the statement, the only purpose of this triangle is to compute binomial coefficients which, once we set the n and k parameters, are constants. Hence managing multiple triangles simultaneously would unnecessarily duplicate data. A single triangle is sufficient.

- *Do you need to use instance variables, class variables, or both ? Why (not) ?*
  Using instance variables allows the user to handle multiple instances of a single class, and have different and independent values of these variables **for each** instances. While class variables have their values shared by **all** the instances of the class. We saw in the previous question that we only need to use a single triangle for our computations. Therefore class variables are the most adequate choice here. We will see that we do not even need to instantiate the class (create a new object and work with it), but we can simply work with the class itself.

- *What kind of data structure can you use to manage the coefficients ?*
  Here we actually have many choices, but the abstract structure is half of a matrix. In `Java` you can use a double array, or an instance of the class `Vector`. There are many possibilities.

  Another thing worth highlighting: as the coefficient will grow in size quite quickly with each line of the triangle, the type of a coefficient should be (at least) a `long`, to allow greater numbers than an `int`.

Exercise 2

- *Assume that a user of your Pascal class can request the computation of arbitrary binomial coefficients, possibly for large values of n. How do you compute a binomial coefficient for given values of n and k ? Can you take advantage of previous computations in order to speed up future requests ?*
  The idea is to be able to pick the value of the coefficient directly from the triangle without having to compute it every time. But one cannot just compute the entire triangle beforehand (it is theoretically infinite and technically just too big even with the bound max(`long`)). A solution would be to start with a small triangle (just one or even zero line), then when the user asks for a coefficient, we compute every line up to the one required and return the coefficient. Then, for each subsequent request, if the new n is inferior to the previous one, the coefficient is already in the triangle and we do not need any further computation. If the new n is superior, we just complete the triangle up to this new value.

- *What kind of variable are you going to use to store a coefficient ?*
  As said in the previous exercise, `long` is a good option, but if we require arbitrarily high values of n, we might be forced to design a specific class to represent integers of any size. But we will not consider this problem here.

- *How do you handle computation of a coefficient for invalid values of n and k ?*
  Here again there are different ways to deal with this, some will be studied later in the course (`Errors` and `Exceptions`). For now we can return a specific value that will be interpreted as an error (like -1 for instance as binomial coefficients are always positive). In this case it is important to specify to the user such behavior in the documentation of your code.

- *What is the appropriate visibility for each component of your `Pascal` class ?*
  Although some details may vary depending on the implementations, this class sole purpose is to return a binomial coefficient to a user external of this class. Therefore the only thing the user should be able to access is a (hence `public`) method that computes the coefficient and returns it. All the rest (*i.e.* the triangle, the method to resize the triangle and even possibly the display method used as a debug tool) should be private to forbid access to anything outside of the class. This ensures that there is no way to modify the triangle other than by using a method of the class itself, making it impossible (as long as your resize method is correct) to return a wrong coefficient.

Exercise 3 You can find, together with this correction, a proposition of a solution for the coding part.