- Let's discuss the “Shape” exercise. If a Square was defined as a subtype of Rectangle, what is the meaning of the methods GetB()/SetB() for Squares (Do we possibly have just an inappropriate naming here?)? Maybe we could override GetB()/SetB() in the class Square so that they only access/modify the side A. But there is a problem with this approach, if there is a method awaiting a Rectangle and we pass a Square to it (this is possible, because because a Square can substitute a Rectangle). If that method sets the side B to a new value, then also the side A will be modified because the passed argument is of type Square and this is the behavior of Squares. But if the passed argument was a Rectangle this behavior is wrong, because SetB() should not modify A, they are independent data in a Rectangle. -> We have an invariant conflict that can not be solved with the substitution principle (but maybe with explicit type checking...). As possible solution: Square and Rectangle are related in geometry but not as oo types. A Square is no Rectangle in the oo sense, there is no inheritance and no “is-a” relationship. Perhaps a common basetype can be defined, we already have Shape. Technically oo-inheritance is no “is-a” relationship, it just allows to declare variables and methods in a subscope.
- Another motivation for this problem: Complex numbers and real numbers.
  - If we want to apply LSP, we have to define a substitutability between complex number and real number. From a mathematical perspective complex number is also a real number. => I.e. from an oo perspective real number derives from complex number. This means a complex number is more abstract than a real number. But a complex number does also contain two real numbers representing its real part and its imaginary part.
    But this relation implemented in an oo-inheritance hierarchy makes no sense:
    - 1. The more abstract type "complex number" has more data (real/imaginary part) than the more concrete type "real number".
    - 2. It also breaks the DIP, because the more abstract type "complex number" depends on the more concrete type "real number".
    - 3. Another problem is behavior in this case: real numbers define an ordered field, but complex numbers don't, the <,> relations are not defined for complex numbers (broken principle of permanence).