Exercises:

1. Revisit the architecture of the *Car*-hierarchy you have created for the exercises of the last lecture.
   a) Add at least one abstract class or make an existing class abstract.
   b) Mind to update the class diagram.
2. Architect the types Shape, Triangle, Circle, Rectangle and Square by the usage of inheritance. Not every type needs to be implemented in its own class, if helpful more types can be introduced. Respect following guidelines:
   a) Stick to the SOLID principle.
   b) All fields needs to be encapsulated.
   c) One of the types needs to be an abstract class.
   d) Following public methods should be implemented reasonably: *getPosition()*/*setPosition()*, *getA()*/*setA()*, *getB()*/*setB()*, *getC()*/*setC()*, *getD()*/*setD()* and *getRadius()*/*setRadius()*.
   e) The operator<< should be overridden to put a representation of the data position, a, b, c, d and radius where reasonable.
   f) Create am UML class diagram.
   g) Prove the functionality of that types with some unit tests.

Remarks:

- Everything that was left unspecified can be solved as you prefer.
- In order to solve the exercises, only use known constructs, esp. the stuff you have learned in the lectures!
- The usage of goto, C++11 extensions, as well as #pragmas is not allowed. The usage of global variables is not allowed.
- **Please obey these rules for the time being:**
  - **The usage of goto, C++11 extensions, as well as #pragmas is not allowed.**
  - **The usage of global variables is not allowed.**
  - **You mustn't use the STL, because we did not yet understood how it works!**
  - **But *std::string*, *std::cout*, *std::cin* and belonging to manipulators can be used.**
- Only use classes for your UDTs. The usage of public fields is not allowed! The definition of inline member functions is only allowed, if mandatory!
- Do not put class definitions and member function definitions into separate files (we have not yet discussed separated compilation of UDTs).
- Your types should apply const-ness as far as possible. They should be const-correct. Minimize the usage of non-const&!
- The results of the programming exercises need to be runnable applications! All programs have to be implemented as console programs.
- The programs need to be robust, i.e. they should cope with erroneous input from the user.
- You should be able to describe your programs after implementation. Comments are mandatory.
- In documentations as well as in comments, strings or user interfaces make correct use of language (spelling and grammar)!
- Don't panic: In programming multiple solutions are possible.
- Don't send binary files (e.g. the contents of debug/release folders) with your solutions! Do only send source and project files.
- If you have problems use the Visual Studio help (F1) or the Xcode help, books and the internet primarily.
- Of course you can also ask colleagues; but it is of course always better, if you find a solution yourself.