Exercises:

1. Implement the class hierarchy with *Car*, *Bus*, *VintageCar* etc. as far as needed (as far as you wish :) ). – For following tasks use your freedom and fantasy, there are many ways to implement them and there is basically no wrong solution. If appropriate you can also use methods instead of member functions.
   a) Create a class diagram. Just do it by hand with a pencil, which shows the class hierarchy as far as you wish. Hint: assume, that you have to modify the diagram (erasing and redrawing of architectural parts), to adapt it to the changes you have/want to do.→ This is just the reality using UML!
   b) Additionally add the idea of *Tyre*s in your design. Assume, that a Car can have multiple *Tyre*s.
   c) Implement the method *GetPressure()*, which retrieves the pressure of *Tyre*s.
2. Create a UDT that can not be used as a base type! Write some sentences how it works.
3. What happens, when you access sliced fields? (Tip: reinterpret_cast)

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.