Solutions to this sheet are due on 28.06.2019 til 14:00. Please hand in a digital version of your answers via e-mail. The e-mail’s subject has to contain cppp19. Do zip-compress your solutions. **Note:** If you copy text elements / code elements from other sources, clearly mark those elements and state the source. Copying solutions from other students is prohibited. All of your files that belong to your solution have to be contained in a single .zip file that is named according to the following naming scheme: `<name>_<surname>_solution_<X>.zip`. Replace `<name>` and `<surname>` with your actual name and replace `<X>` with the number of the exercise sheet. You can look up your results using this link: [https://docs.google.com/spreadsheets/d/1Ve7yE20xTs0XC1Tw08VpvgvLsaCuU60lP2F4oud1s/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1Ve7yE20xTs0XC1Tw08VpvgvLsaCuU60lP2F4oud1s/edit?usp=sharing)

This exercise sheet will help you to familiarize yourself with inheritance and object oriented programming. You can achieve 16 points in total. The code for this exercise sheet can be found on the lecture’s website: [https://www.hni.uni-paderborn.de/fileadmin/Fachgruppen/Softwaretechnik/Lehre/CPP_Programming/SS19/code_08.zip](https://www.hni.uni-paderborn.de/fileadmin/Fachgruppen/Softwaretechnik/Lehre/CPP_Programming/SS19/code_08.zip)

**Exercise 1.**
Consider the following code:

```cpp
#include <iostream>
using namespace std;

class base {
    public:
        virtual `base() = default;
        virtual void iam() { cout << "I am base
"; }
};
```

a) Define two classes `derived_one` and `derived_two` that both inherit from `base`. (1 P.)

b) In each of the derived classes override the `iam()` function to print the name of the derived class. (1 P.)

c) Why is it a good idea to explicitly specify functions that override a virtual function with the keyword `override`? (1 P.)

d) Consider task c) and describe a scenario where one gets into trouble when not having specified overriding functions as `override`. (1 P.)
Exercise 2.
Consider the following two interfaces:

```cpp
#include <iostream>
using namespace std;

struct greetings {
    virtual ~greetings() = default;
    virtual void say_hello() = 0;
    virtual void say_goodbye() = 0;
};

struct politeness {
    virtual ~politeness() = default;
    virtual void say_please() = 0;
    virtual void say_thanks() = 0;
    virtual void say_your_welcome() = 0;
};
```

Define a class `speaker` that implements both of the above interfaces. All interface functions should be implemented by writing an “adequate message” to the command line. Test your class `speaker` by creating an instance and calling all of its member functions.
(2 P.)

Exercise 3.
Consider the following container interface:

```cpp
class container {
    public:
        virtual ~container() = default;
        virtual double& operator[](size_t idx) = 0;
        virtual const double& operator[](size_t idx) const = 0;
        virtual size_t size() const = 0;
};
```

a) Define a class `vec` that implements the container interface. Use a member variable of type `std::vector<double>` to store the elements in your `vec` type. Additionally, provide a constructor `vec(size_t size)` that initializes the member variable in such a way that it is capable of holding `size` elements. (4 P.)

b) Define another class `lst` that implements the container interface, similar to the task above. But this time, use a member variable of type `std::list<double>` to store the elements in your `lst` type. Also provide a constructor `lst(size_t size)` that initializes it such that it is able to store `size` elements. (Hint: When implementing `operator[]` for your list wrapper, the function `std::advance` may be handy to use.) (4 P.)

c) Observe the code shown below. The functions `fill_container()` and `sum_container()` can operate on any type that implements the container interface. Create one variable of your `vec` and one variable of your `lst` type such that they can both store 10 `double` elements. Then call `fill_container()` and `sum_content()` for each of them. You should obtain 55 as a result in both cases. (2 P.)

```cpp
void fill_container(container& c) {
    for (size_t i = 0; i < c.size(); ++i) {
        c[i] = i + 1;
    }
}
```
double sum_container(const container& c) {
    double result = 0;
    for (size_t i = 0; i < c.size(); ++i) {
        result += c[i];
    }
    return result;
}

Exercise 4.
This is an optional exercise: Consider the following code:

```cpp
#include <iostream>
using namespace std;

template <class T>
class base {
    protected:
        T base_value;
    public:
        base(T t) : base_value(t) {}
};

template <class T>
class derived : public base<T> {
    private:
        T derived_value;
    public:
        derived(T t, T u) : base<T>(u), derived_value(t) {}
        void printValues() {
            cout << base_value << 
                << derived_value << 
                << endl;
        }
};

int main() {
    derived<int> d(20, 10);
    d.printValues();
    return 0;
}
```

Try to compile and execute the code. The compilation should result in an error. Can you fix the error? This exercise is intended to google precisely for the right terms in order to find the solution.

(0 P.)

Exercise 5.
Additional material: You may wish to have a look at the talk "Intro to the C++ Object Model", by Richard Powell (CppCon 2015) [https://youtu.be/iLiDezv_Frk](https://youtu.be/iLiDezv_Frk) to deepen and extend your knowledge about C++’s object model.