Exercise Sheet #9

Bulcsú Sándor <sandor@th.physik.uni-frankfurt.de>
Hendrik Wernecke <wernecke@th.physik.uni-frankfurt.de>
Laura Martin <lmartin@th.physik.uni-frankfurt.de>
Christopher Czaban <czaban@th.physik.uni-frankfurt.de>

Problem 1 (*Inheritance and polymorphism*) 14 Pts

(a) Create a class `Square` which has a counter `static int squareCounter` for the number of elements of this class and the members `const double a` for the edge length, `double area` for the area and `double perimeter` for the perimeter. Calling the constructor shall only initialize `a` with a value given as argument to the constructor and increase the counter. Create a member function `void initialize()` to calculate the remaining undefined parameters. Do not call this function in the constructor. Overwirte the destuctor such that it decreases the counter again and prints out the number of remaining class objects. Create a constant member function `printProperties` to print all properties of the square. Do not define any member variable as `public`. (3 Pts)

(b) Convince yourself that you have implemented everything correctly by creating two objects of type `Square` using the following syntax:

```
Square s1(1.);
Square* s2 = new Square(2.);
```

Call `initialize` and `printProperties` for `s1` and `s2`. When are the respective destructors called? For which was memory allocated on the stack and how can one check it? How can one release the memory of the heap again? (3 Pts)

(c) A copy constructor is constructor which creates an object by initializing it with an object of the same class, which has been created previously. Create such a copy constructor for the class `Square` and add a line to the main to show how it is used. (1 Pts)

(d) Create a child class `Cube` that inherits `a` and `area` from `Square` but not `perimeter`. Add two new varibales `double volume` and `surface`. The class `Cube` should have its own counter `static int cubeCounter`. Create the functions `initialize` and `printProperties` for the new class. `Hint:` You have to change/add something in the parent class in order to make it work properly. Some functions must be defined as `virtual`. (3 Pts)

(e) Check if the functions for initializing are called from the respective class when using the following code-snippet in the main:

```
Square* geoms[6];
for(int i=0; i<3; i++){
   geoms[i] = new Square((double)i+1.);
   geoms[i]->initialize();
   geoms[i]->printProperties();
   geoms[i+3] = new Cube((double)i+1);
   geoms[i+3]->initialize();
   geoms[i+3]->printProperties();
}
```

Make sure that the memory allocated on the heap by `new` is released before the end of the main in order to avoid memory leak. (2 Pts)

(f) Overload the `operator /` for both classes. For `Square` it shall return the ratio of the areas and for `Cube` the ratio of the volumes of two objects. Apply overloaded operator to the squares `geoms[0]` and `geoms[1]` and to the cubes `geoms[3]` and `geoms[4]` and print the results.
`Hint:` One can overload operators within the class or outside. If you want to do it outside you have to add constant getter functions for the members `area` and `volume`. (2 Pts)

Problem 2 (*Overloading the stream operators << and >>*) 6 Pts

Following the example of the lecture implement a buffer class with a more extended functionality.

- The basic operations of this class should be the use of the operators << and >> to stream data into and out of a buffer, e.g.

```
int nNumber;
BufferClass mybuffer;
mybuffer << 3;
mybuffer << 4 << 5;
mybuffer >> nNumber;
```

I.e. after streaming into nNumber the integer element 5 should not be in the buffer anymore. (1 Pts)

- Design the class as a template class so that it can be used with different data types such as double or int, e.g.

```
BufferClass<int> int_buffer;
BufferClass<double> double_buffer;
```

(1 Pts)

- The size of the buffer should be specified via the constructor of the class, i.e.

```
BufferClass<int> int_buffer(10);
```

  for a buffer accepting 10 elements of type int. (2 Pts)

- Implement a method to
  - reset the buffer to its initial state,
  - obtain the current load of the buffer,
  - access information about the state of the buffer (empty or full).

  (1 Pts)

- Make proper use of private and public classifications, i.e. if your class has an array to store the data type putting it into the private section forbids misusing it from the outside, e.g.

```
BufferClass<int> int_buffer(10);
double dNumber = int_buffer.array[0]; //
    Must not compile! int_buffer must be
    used with << and >> only!
```

  The same should apply to all other data member elements of your class that should not be exposed directly to the user. For instance

```
class BufferClass
{
    public:
    boolean checkIfEmpty()
    {
        return isEmpty;
    }
    private:
    boolean isEmpty;
};
```

  In this way the user is prevented of having direct access to isEmpty and avoid accidental modification of isEmpty. Instead the user can have access to isEmpty via checkIfEmpty() without being able to modify it. (1 Pts)

- (*optional*) Implement a method to resize the buffer, e.g.

```
BufferClass<int> int_buffer(10);
int_buffer.resize(20);
```

to allow the user to stream more elements into the buffer than previously specified. (0 Pts)