# C++ Exercises
# Set 4

Author(s): Huub Exel, Jessay Beukema, Sofia van der Wal, Ioannis Angelo Tassioulas

–

12:31

October 4, 2023

## 25

What is the meaning of 'encapsulation' and 'data hiding',
why are they important when desiging classes
and why is the implementation irrelevant?

Encapsulation:
A programming concept where related sections of code (i.e. related functions
and data) are bound together and are used only in relation to one
another. Encapsulating code ensures that the code remains modular and the
integrity of the data remains. The class encapsulates all actions performed
on its data members. The class object may assume the
responsibility for its own data-integrity.

Data hiding:
When data is hidden, it cannot be touched directly by anything outside of
that class or classes that extend from this class (or as the book calls it,
the data cannot directly by touched by the outer world).
Things can access it, but only by using specific methods that the
programmer made to either get or manipulate the data (called accessors
and manipulators). This means that the data is much safer when compared to
keeping all the data public. With data hiding you enforce the data-integrity.
This is why encapsulation and data hiding go perfectly together.

Consider the example of the Wizard class interface underneath:
(only the interface is required for the understanding of code encapsulation,
because encapsulation is all about how the user approaches the data, the user
only ever has to see the interface of a class and not the implementation.)

```
class Wizard
{
    int d_health;
    int d_magic;

    public:
        Wizard();
        fireball();

};
```

In the example above of the Wizard class, the 2 ints d_health and d_magic,
the constructor Wizard() and the member function fireball() are all
encapsulated together. If the program wants to work with these pieces of
data specifically, it must use the Wizard class member functions.
This also allows the programmer to change the member functions as long as the
input and output stays the same and the declaration stays the same.
The developer using these functions does not need to worry
about the definition of the function, this makes the code more modular and
easy to work with.

```
There are 2 ints defined in the private part of the class. All data in
classes are private unless specified to be public – this is an important
aspect of data hiding that classes implement. Private data is not allowed to
be touched by anything other than the fellow members of the class.
The information inside of d_health and d_magic stay separated
from the publicly available data / local data, and that data remains
"hidden" from code outside of the class (or the classes that inherit
this code, when we will be using 'protected' later on).
```

## 26

```
Implement the class "Person" as discussed in the C++ Annotations with additional members
   insert, and extract.
```

Listing 1: public_header/person.hh

```
#ifndef INCLUDED_PERSON_
#define INCLUDED_PERSON_

#include <string>

class Person
{
    std::string d_name;          // Name of person
    std::string d_address;       // Address field
    std::string d_phone;         // Telephone number
    size_t      d_mass;          // The mass in kg

    public:
        // constructors
        Person()                    // Delegating constructor that sets
        :                           // everything to to an empty string or 0
            Person("", "", "", 0)
        {}
        Person(
            std::string const &name,
            std::string const &address,
            std::string const &phone,
            size_t mass);

        // accessors
        std::string const &name()    const;
        std::string const &address() const;
        std::string const &phone()   const;
        size_t mass()                const;

        // manipulators
        void setName(std::string const &name);
        void setAddress(std::string const &address);
        void setPhone(std::string const &phone);
        void setMass(size_t mass);

        void setDataMember(std::string &&data, size_t counter);

        // static
        static bool const hasOnly(std::string const &&charsAllowed,
                                  std::string const &stringToCheck);

        // other
        void insert(std::ostream &out);
        void extract(std::istream &in);
};

#endif
```

Listing 2: internal_header/person.ih

```
#include "../public_header/person.hh"

#include <iostream>
```

```
using namespace std;
```

Listing 3: constructor/person_all_data_members.cc

```
#include "../internal_header/person.ih"

Person::Person(string const &name, string const &address,
               string const &phone, size_t mass)
:                            // Set all the data members of this person
    d_name(name),
    d_address(address),
    d_phone(phone),
    d_mass(mass)
{}
```

Listing 4: name/name.cc

```
#include "../internal_header/person.ih"

string const &Person::name() const
{                            // Return this persons' name
    return d_name;
}
```

Listing 5: address/address.cc

```
#include "../internal_header/person.ih"

string const &Person::address() const
{                            // Return this persons' address
    return d_address;
}
```

Listing 6: phone/phone.cc

```
#include "../internal_header/person.ih"

string const &Person::phone() const
{                            // Return this persons' phonenumber
    return d_phone;
}
```

Listing 7: mass/mass.cc

```
#include "../internal_header/person.ih"

size_t Person::mass() const
{                            // Return this persons' mass
    return d_mass;
}
```

Listing 8: setname/setname.cc

```
#include "../internal_header/person.ih"

void Person::setName(string const &name)
{                            // Set this persons' name
    d_name = name;
}
```

Listing 9: setaddress/setaddress.cc

```
#include "../internal_header/person.ih"

void Person::setAddress(string const &address)
```

```
{                               // Set this persons' address
    d_address = address;
}
```

Listing 10: setphone/setphone.cc

```
#include "../internal_header/person.ih"

void Person::setPhone(string const &phone)
{                               // Set this persons' phonenumber
    if (phone.empty())          // If phone is empty, set not available
        d_phone = " - not available -";
    else if (hasOnly("0123456789", phone))
        d_phone = phone;        // If phone has something that isn't a number,
    else                        // print a message
        cout << "A phone number may only contain digits\n";
}
```

Listing 11: setmass/setmass.cc

```
#include "../internal_header/person.ih"

void Person::setMass(size_t mass)
{                               // Set this persons' mass
    d_mass = mass;
}
```

Listing 12: setdatamember/setdatamember.cc

```
#include "../internal_header/person.ih"

void Person::setDataMember(string &&data, size_t counter)
{                                      // According to the counter, set a
    switch (counter)                   // datamember everytime this function is
    {                                  // called
        case 0:
            setName(data);
        break;

        case 1:
            setAddress(data);
        break;

        case 2:
            setPhone(data);
        break;

        case 3:                        // For case 3 (mass) stoull has to be
            setMass(stoull(data));     // used to be able to convert the string
        break;                         // to a size_t
    }
}
```

Listing 13: hasonly/hasonly.cc

```
#include "../internal_header/person.ih"

bool const Person::hasOnly(string const &&charsAllowed,
                           string const &stringToCheck)
{
    for (char charToCheck: stringToCheck) // For every char in stringToCheck,
    {                                     // check if it is in charsAllowed
        if (charsAllowed.find(charToCheck) == string::npos)
            return false;                 // If not, return false
    }
    return true;                          // If all characters are in
}                                         // charsAllowed, return true
```

Listing 14: insert/insert.cc

```
#include "../internal_header/person.ih"

void Person::insert(ostream &out)
{
    out << "name: " << Person::name() << ", address: " << Person::address()
        << ", phone: " << Person::phone() << ", mass: " << Person::mass()
        << '\n';                    // Insert the Persons' information in the
}                                   // out (ostream) given
```

Listing 15: extract/extract.cc

```
#include "../internal_header/person.ih"
                                    // Get data from input and put it in the
void Person::extract(istream &in) // datamembers according to interface
{                                   // order, with helper func setDatamember
    string line;                    // Line variable for data
    while (getline(in, line))       // Get data from input and put it in line
    {
        line += ',';                // Add a , to the line for easy processing
        size_t separatorPos, counter = 0;
        while ((separatorPos = line.find(',')) != string::npos)
        {                           // Send piece of data to setDataMember
            setDataMember(line.substr(0, separatorPos), counter);
            line.erase(0, separatorPos + 1);
            ++counter;              // Erase the piece of data from line,
        }                           // including the , right behind it
    }                               // Add one to the counter, so setDataMember
}                                   // handles the input correctly
```

## 28

```
Design a class Line with two members,
one that returns true if the line read contains non-ws characters,
the other returns the next line consisting of non-ws characters.
```

Listing 16: public_header/line.hh

```
#ifndef INCLUDED_LINE_
#define INCLUDED_LINE_

#include <string>

class Line
{
    std::string d_line;
    size_t d_ws_index;              // The index of a whitespace

    public:
        // accessors
        std::string next();

        // manipulators
        bool getLine();
};

#endif
```

Listing 17: internal_header/line.ih

```
#include "../public_header/line.hh"

#include <iostream>
#include <cctype>

using namespace std;
```

Listing 18: next/next.cc

```
#include "../internal_header/line.ih"

string Line::next()
{                                   // If the index is string::npos, return ""
    if (d_ws_index == string::npos)
        return "";                  // Set the beginposition, if the first word
                                    // is found, add one to omit the whitespace
    size_t beginPosition = (d_ws_index != 0) ? d_ws_index + 1 : d_ws_index;
    d_ws_index = d_line.find_first_of(" \t\n\v\f\r", beginPosition);
                                    // Set the position of the next whitespace
    return d_line.substr(beginPosition, d_ws_index - beginPosition);
}                                   // Return the substring
```

Listing 19: getline/getline.cc

```
#include "../internal_header/line.ih"

bool Line::getLine()
{
    getline(cin, d_line);           // Get line from stdin
    for (char character: d_line)    // For every character in that line
    {
        if (!isspace(character))    // Check if the character is a non
            return true;            // whitespace character, if there is at
    }                               // least one non-ws characters, return true

    return false;                   // If not, return false
}
```