C++ Exercises
Set4

Author(s): Tino Alferink, Robert van Ark, Owen Givlin
Previously rated by Andrei
17:00

October 8, 2024

25

This assignment we will describe encapsulation and data hiding using a small example.

Encapsulation involves bundling data and functions operating on that data into a single unit, which is typically a class. This allows better management and interaction with the data.
Data hiding describes that private data inside a class cannot be modified or be seen by sections outside the class. This can be done indirectly using public member functions of the class implemented by the programmer. This way, due to the encapsulation, the class is responsible for its own data integrity.

Listing 1: `bottle.h`

```
#ifndef INCLUDED_BOTTLE_
#define INCLUDED_BOTTLE_

class Bottle
{
    double d_capacity;
    double d_amount;

    public:
        Bottle(double capacity);

        double getCapacity() const;
        double getAmount() const;

                                    // Add amount water to the bottle.
        bool addWater(double amount);
                                    // Remove amount water to the bottle.
        bool removeWater(double amount);
};

#endif
```

This example shows a simple class describing a bottle with a capacity, where water can be added or removed. The functions of Bottle are public, and the data members are private.
The implementation of the class is irrelevant to show encapsulation and data hiding, as the interface provides controlled access and the implementation remains hidden from the user.

26

In this assignment the class Person, as described in C++ Annotations, is implemented and extended with insert and extract members.

Listing 2: `person/person.h`

```
#ifndef INCLUDED_PERSON_
#define INCLUDED_PERSON_

#include <string>
#include <iostream>

class Person
{
    std::string d_name;
    std::string d_address;
    std::string d_phone;
    size_t      d_mass;

    public:
        Person() : d_name("unknown"),
                   d_address("unknown"),
                   d_phone("unknown"),
                   d_mass(0) {};
        Person(std::string const &name,
               std::string const &address,
               std::string const &phone,
               size_t mass);

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

        std::string const &name() const
        {
            return d_name;
        }
        std::string const &address() const
        {
            return d_address;
        }
        std::string const &phone() const
        {
            return d_phone;
        }
        size_t const mass() const
        {
            return d_mass;
        }

                                    // Insert info on this person into os.
        void insert(std::ostream &os) const;
                                    // Set data members using data from is.
        void extract(std::istream &is);
};

#endif
```

Listing 3: person/person.ih

```
#include "person.h"

using namespace std;
```

Listing 4: person/person.cc

```
#include "person.ih"

Person::Person(string const &name, string const &address, string const &phone,
               size_t mass)
:
    d_mass(mass)
{                                   // Logic checks for everything except mass
    setName(name);
    setAddress(address);
    setPhone(phone);
```

```
}
```

Listing 5: person/setName.cc

```
#include "person.ih"

void Person::setName(string const &name)
{
    if(name.empty())
        cout << "A person must have a name.\n";
    else
        d_name = name;
};
```

Listing 6: person/setAddress.cc

```
#include "person.ih"

void Person::setAddress(string const &address)
{
    if(address.empty())
        d_address = "unknown";
    else
        d_address = address;
};
```

Listing 7: person/setPhone.cc

```
#include "person.ih"

void Person::setPhone(string const &phone)
{
    if(phone.empty())
        d_phone = "unknown";
    else if(phone.find_first_not_of("0123456789") == string::npos)
        d_phone = phone;
    else
    {
        cout << "Please provide a phone number only containing digits.\n";
        d_phone = "wrong input";
    }
};
```

Listing 8: person/setMass.cc

```
#include "person.ih"

void Person::setMass(size_t const mass)
{
    if (mass < 0)
        cout << "provide a positive mass, or 0.\n";
    else
        d_mass = mass;
};
```

Listing 9: person/insert.cc

```
#include "person.ih"

void Person::insert(ostream &os) const
{
    os << "Name:    " << d_name << '\n'
       << "Address: " << d_address << '\n'
       << "Phone:   " << d_phone << '\n'
       << "Mass:    " << d_mass << '\n';
}
```

Listing 10: person/extract.cc

```
#include "person.ih"

void Person::extract(istream &is)
{
    string name;
    string address;
    string phone;
    string mass;

    getline(is, name, ',');
    getline(is, address, ',');
    getline(is, phone, ',');
    getline(is, mass);

    setName(name);
    setAddress(address);
    setPhone(phone);
    setMass(stoi(mass));
}
```

27

In this assignment we use our implementation of class Person in a simple program that reads and writes the data of multiple Person objects form and onto cin and cout.

Listing 11: program.h

```
#ifndef PROGRAM_H
#define PROGRAM_H

#include <vector>

#include "../26/person/person.h"

void readPersons(Person *persons, size_t amt);
void writePersons(Person const *persons, size_t amt);

#endif
```

Listing 12: program.ih

```
#include "program.h"

#include <iostream>

using namespace std;

enum
{
    STD_NUM_PEOPLE = 5
};
```

Listing 13: readPersons.cc

```
#include "program.ih"

void readPersons(Person *persons, size_t amt)
{
    for (size_t idx = 0; idx < amt; ++idx)
    {
        cout << "? ";
        Person person = Person();
        person.extract(cin);        // Read data from cin to person.
        persons[idx] = person;
    }
}
```

Listing 14: writePersons.cc

```
#include "program.ih"

void writePersons(const Person *persons, size_t amt)
{
    for (size_t idx = 0; idx < amt; ++idx)
        persons[idx].insert(cout);  // Write person at idx their data to cout.
}
```

Listing 15: main.cc

```
#include "program.ih"

int main(int argc, char *argv[])
{
    size_t numPeople;
    if(argc > 1)                    // Get numPeople from arguments if provided.
        numPeople = stoi(argv[1]);
    else                            // Use predefined numPeople.
        numPeople = STD_NUM_PEOPLE;

    Person people[numPeople];
    readPersons(people, numPeople);
    writePersons(people, numPeople);
}
```

29

Design the class Parser. It's a class using the facilities of the class Line (cf. the previous exercise), and having a Line d_line data member.

Listing 16: line.h

```
#ifndef LINE_INCLUDED_
#define LINE_INCLUDED

#include <string>

class Line
{
    size_t d_pos;
    std::string d_currentLine;

    public:
        bool getLine();
        std::string next();
};

#endif
```

Listing 17: parser/parser.h

```
#ifndef PARSER_INCLUDED_
#define PARSER_INCLUDED_

#include "../line.h"
#include <string>

class Parser
{
    Line d_line;
    bool d_integral;

    public:
        enum Return                 // declared public so it can be used by
        {                           // class calculator
            NO_NUMBER,
            NUMBER,
```

```
            EOLN
        };

        bool reset();
        Return number(double *dest);
        bool isIntegral();
        std::string next();

    private:
        Return convert(double *dest, std::string const &str);
        bool pureDouble(double *dest, std::string const &str);

};

#endif
```

Listing 18: parser/parser.ih

```
#include "parser.h"
#include <string>

using namespace std;
```

Listing 19: parser/convert.cc

```
#include "parser.ih"

Parser::Return Parser::convert(double *dest, string const &str)
{
    try
    {
        return pureDouble(dest, str) ? NUMBER : NO_NUMBER;
    }
   catch (...)                      // conversion failed
    {
        return NO_NUMBER;
    }
}
```

Listing 20: parser/isintegral.cc

```
#include "parser.ih"

bool Parser::isIntegral()
{
    return d_integral;
}
```

Listing 21: parser/next.cc

```
#include "parser.ih"

string Parser::next()
{
    return d_line.next();
}
```

Listing 22: parser/number.cc

```
#include "parser.ih"

Parser::Return Parser::number(double *dest)
{
    string subString = d_line.next();
    if (subString.empty() == true)
        return Parser::EOLN;

    return convert(dest, subString);// Return (NO_)NUMBER based on successful
```

```
                                        // or failing conversion.
}
```

Listing 23: parser/puredouble.cc

```
#include "parser.ih"

bool Parser::pureDouble(double *dest, std::string const &str)
{
    size_t pos = 1;                 // ensure pos is nonzero
    *dest = stod(str, &pos);

    if (pos != str.length())        // no complete conversion
        return false;
    else                            // complete conversion
    {
        d_integral = str.find_first_of(".eE") == string::npos;
        return true;
    }
}
```

Listing 24: parser/reset.cc

```
#include "parser.ih"

bool Parser::reset()
{
    return d_line.getLine();
}
```

30

In this exercise, we create a Calculator class responsible for making simple calculations from the input evaluated from cin.

This class also has no constructor. This is because we just want to initialize a new calculator object, without having to initialize anything integral to the object. This way, the compiler automatically generates a default constructor where the data members are unitialized. This does not matter in our case because run() will overwrite all data members, with run() being the only public member of Calculator.

Listing 25: calculator/calculator.h

```
#ifndef CALCULATOR_INCLUDED_
#define CALCULATOR_INCLUDED_

#include "../../29/parser/parser.h"  // why needed here but not for ex 29??
#include <string>

class Calculator
{
    Parser d_parser;

    double d_firstNr = 0;
    double d_secondNr = 0;
    bool d_firstIsInt;
    bool d_secondIsInt;
    double ZERO_LIMIT = 1e-8;

    enum Operator
    {
        ADDITION,
        DIVISION,
        MODULO,
        MULTIPLICATION,
        SUBTRACTION
    };
```

```
    Operator d_operator;

    public:
        void run();

    private:
        bool expression();
        bool number(double *dest, bool *isInt);
        bool getOperator();
        void evaluate();
        void usageMsg();
        void errorMsg();
        bool containsInvalid();
};

#endif
```

Listing 26: calculator/calculator.ih

```
#include "calculator.h"
#include <string>
#include <iostream>
#include <cstdlib>

using namespace std;
```

Listing 27: calculator/run.cc

```
#include "calculator.ih"

void Calculator::run()
{
    usageMsg();

    while(true)
    {
        cout << "? ";
        if (!d_parser.reset())      // if no new line is filled, break
        {
            break;
        }
        evaluate();
    }
}
```

Listing 28: calculator/expression.cc

```
#include "calculator.ih"

bool Calculator::expression()
{
                        // check if number is provided, while storing in
                        // relevant addresses.
    if (!number(&d_firstNr, &d_firstIsInt))
        return false;

    if (!getOperator()) // get operator andcheck if valid operator is provided
        return false;
                        // check if number is provided, while storing in
                        // relevant addresses.
    if (!number(&d_secondNr, &d_secondIsInt))
        return false;

    if (containsInvalid())
        return false;

    return true;

}
```

Listing 29: calculator/number.cc

```
#include "calculator.ih"

bool Calculator::number(double *dest, bool *isInt)
{
    if (d_parser.number(dest) == Parser::NUMBER)
    {
        if (d_parser.isIntegral() == true)
            *isInt = true;
        else
            *isInt = false;
        return true;                // read substring is number and check int.
    }
    else
        return false;               // if next read substring is not a number.

}
```

Listing 30: calculator/getoperator.cc

```
#include "calculator.ih"

bool Calculator::getOperator()
{
    string operatorStr = d_parser.next();

    if (operatorStr == "+")
    {
        d_operator = Operator::ADDITION;
        return true;
    }
    else if (operatorStr == "/")
    {
        d_operator = Operator::DIVISION;
        return true;
    }
    else if (operatorStr == "%")
    {
        d_operator = Operator::MODULO;
        return true;
    }
    else if (operatorStr == "*")
    {
        d_operator = Operator::MULTIPLICATION;
        return true;
    }
    else if (operatorStr == "-")
    {
        d_operator = Operator::SUBTRACTION;
        return true;
    }
    else
        return false;

}
```

Listing 31: calculator/evaluate.cc

```
#include "calculator.ih"

void Calculator::evaluate()
{
    if(expression())
    {
        switch(d_operator)
        {
            case Operator::ADDITION:
                cout << d_firstNr + d_secondNr << '\n';
                break;
            case Operator::DIVISION:
```

```
                cout << d_firstNr / d_secondNr << '\n';
                break;
            case Operator::MODULO:
                    cout << static_cast<int>(d_firstNr)
                            % static_cast<int>(d_secondNr) << '\n';
                    break;
            case Operator::MULTIPLICATION:
                cout << d_firstNr * d_secondNr << '\n';
                break;
            case Operator::SUBTRACTION:
                cout << d_firstNr - d_secondNr << '\n';
                break;
            default:
                errorMsg();
        }
    }
    else
        errorMsg();
}
```

Listing 32: calculator/usageMsg.cc

```
#include "calculator.ih"

void Calculator::usageMsg()
{
    cout <<
R"(
Usage: On prompt (? ), provide a number (integral or double), followed by a
whitespace (space or tab) followed by the desired operator followed by yet
another whitespace and then ending with the second integral or double number.
Provide an empty line to quit the calculator.
)";
}
```

Listing 33: calculator/errorMsg.cc

```
#include "calculator.ih"

void Calculator::errorMsg()
{
    cout << "Provide a correct statement, see Usage info. \n";
}
```

Listing 34: calculator/containsInvalid.cc

```
#include "calculator.ih"

bool Calculator::containsInvalid()
{
    switch(d_operator)
    {
        case DIVISION:
            if (abs(d_secondNr) < ZERO_LIMIT)
            {
                cout << "error: division by 0\n";
                return true;
            }

        case MODULO:
            if(!d_firstIsInt || !d_secondIsInt ||
                d_firstNr < 0 || d_secondNr < 0)
            {
                cout << "error: modulo with double number or negative number\n";
                return true;
            }
            if(d_secondNr == 0)
            {
                cout << "error: modulo 0\n";
                return true;
```

```
            }


        default:
            return false;
    }
}
```

31

In this exercise, we use the Calculator class created in the previous assignment in a program and use it to perform some calculations. We also provide an example of how the program is used, so calculations and incorrect input handling are shown.

Listing 35: `main.cc`

```
#include "../30/calculator/calculator.h"

int main()
{
    Calculator calculator;
    calculator.run();
}
```

The following is a small illustration of how correct calculations are done and how incorrect input is dealt with:

```
Usage: On prompt (? ), provide a number (integral or double), followed by a
whitespace (space or tab) followed by the desired operator followed by yet
another whitespace and then ending with the second integral or double number.
Provide an empty line to quit the calculator.
? 13 + 15
28
? 15 - .5
14.5
? -4 * 4
-16
? 16.5 / 1.5
11
? 4e3 % 45
40
? 13+15
Provide a correct statement, see Usage info.
? 12 / 0
error: division by 0
Provide a correct statement, see Usage info.
? 12 % -5
error: modulo with double number or negative number.
Provide a correct statement, see Usage info.
? 4 % 0
error: modulo 0
Provide a correct statement, see Usage info.
```