Bern University
of Applied Sciences

394661-FS2019-0 - C++ Programming I

EXERCISE-06

TABLE OF CONTENTS

1 Introduction

This exercise of 394661-FS2019-0 will focus on the basic concepts of inheritance. Inheritance is a very powerful tool in object-oriented programming. Understanding the basics of inheritance paths the way to learn the holy grail of object-oriented programming: polymorphism.

You will learn the following topics when completing this exercise:

- Understanding inheritance
- Overriding base class methods
- Order of object construction and destruction for multilevel and multiple inheritance
- Access specifiers: `public`, `protected` and `private`

2 Excercises

Create CMake-Projects with C++ 11 compiler support and Debug/Release build options for the exercise. Add additional files manually to the project to gain full control over the included project files. As an exercise create separate files for each class in exercise 2.1, *i.e.* for all base and derived classes, respectively.

2.1 ATM

In this exercise you will write a base class called `Account` and a derived class called `CheckingAccount` to demonstrate the basic usage of inheritance. It's your job to ensure maximum data encapsulation by choosing the appropriate access level specifiers! Start with the base class providing the following functionality:

- `Account` has a member `balance` and a constructor with the initial balance as argument.
- Add the methods `credit` and `debit` which add money to and subtract money from the account if enough balance is available, respectively. Warn otherwise!
- Write a `printBalance` method to output the current balance.

Write a derived class`CheckingAccount` which inherits from `Account` and extends the base class by:

- A member `fee`
- A constructor with the initial `balance` and `fee` as argument. Use an initialization list to invoke the constructor of the base class
- Override the function `debit` which additionally subtracts the `fee` for every transaction from the account if enough balance and warn otherwise

Test your classes with the following test program (`ex06.cpp`):

```
#include <iostream>
#include "checkingaccount.h"

int main()
{
    CheckingAccount myAccount(1000, 0.5); // initial amount and fee
    myAccount.credit(250);     // ok
    myAccount.printBalance(); // -> 1250
    myAccount.debit(1000);     // ok
    myAccount.printBalance(); // -> 249.5
    myAccount.debit(249.5);    // Waring: Not enough funds
    myAccount.debit(249);      // ok
    myAccount.printBalance(); // -> 0

    return 0;
}
```

2.2 Code Snippets

For each of the following programs, determine what they output, or if they would not compile, indicate why. This exercise is meant to be done by inspection, so do not compile these (otherwise the answers are trivial).

a)

```cpp
#include <iostream>

class Base
{
public:
    Base()
    {
        std::cout << "Base()\n";
    }
    ~Base()
    {
        std::cout << "~Base()\n";
    }
};

class Derived: public Base
{
public:
    Derived()
    {
        std::cout << "Derived()\n";
    }
    ~Derived()
    {
        std::cout << "~Derived()\n";
    }
};

int main()
{
    Derived d;
    Base b;
}
```

b)

```cpp
#include <iostream>

class Base
{
private:
    int m_x;
public:
    Base(int x): m_x(x)
    {
        std::cout << "Base()\n";
    }
    ~Base()
    {
        std::cout << "~Base()\n";
    }

    void print() { std::cout << "Base: " << m_x << '\n';  }
};

class Derived: public Base
{
public:
    Derived(int y):  Base(y)
    {
        std::cout << "Derived()\n";
    }
    ~Derived()
    {
        std::cout << "~Derived()\n";
    }

    void print() { std::cout << "Derived: " << m_x << '\n'; }
};

int main()
{
    Derived d(5);
    d.print();
}
```

2.3 Order of Construction & Destruction

Implement a dummy case of multiple inheritance. Write a `class ABC` (header only!) which derives from `class A`, `class B` and `class C` in the following order:

```
class ABC : public A, public C, public B
```

- What is the order of construction/destruction when creating an instance of `class ABC`?
- Create similar code and output as the code snippet in section 2.2 a), i.e. (header only!), and test some other variants
- What's the general rule for the order of construction and destruction?

2.4 Access Specifiers

Imagine a multilevel inheritance example, where `class C` derives from `class B` and `class B` derives from `class A`.

- Implement the example in a header only file
- `class A` has a member variable `m_A`. How can you make sure that `class B` has access to `m_A`, but not `class C`? Choose access levels accordingly.
- There are two solutions - which one is preferable and why?

3 Submission

Submit your source code (as a zip-file) to Ilias EXERCISE-06 **before the deadline** specified in Ilias.