C++ Classes and Objects

Spring 2018

Object Oriented Programming (OOP)

- Separation of code and data through encapsulation and data hiding
- Encapsulation: combining code and data into a single object
- Data hiding refers to the ability to hide data from code that is outside of the object

```
class SavingAccount
{
   private:
     double    m_balance ;
   public:
     SavingAccount()   {    }
     SavingAccount(double initBalance) {    }
     void printBalance() {  }
}
```

C++ Class and Object

- Class can contain
  - Data members
  - Or member functions (also called as “methods”)
- Class is a “blueprint” where the objects are created from. It describes the object.
- Class is a “user-defined” type

- An object is an instantiation of a class. You can create as many objects of the class as needed

- Class is the data type. Object is the variable.

```
class SavingAccount
{
  private:
    double   m_balance;
 public:
    SavingAccount()   {    }
    SavingAccount(double initBalance) {    }
}

SavingAccount  myAcc ;
```

Data Members

- A data member is declared in a class, but not in the body of any of the class's member functions
- Every object of a class has its own data members
- Data members are accessible to all member functions of the class

```
class StudentAccount
{
   public:
    long   m_id ;          // data member
    double    m_gpa ;       // data member
    void printGPA() {  }
}
```

Member Methods

- Is a member function in a class
- It consists of:
  - Return type
  - Function/method name
  - Function/method parameters

```
class StudentAccount
{
  public:
    long   m_id ;
    double    m_gpa ;
    void printGPA() {  }    // method member
    void  setGPA(double newGPAValue) ;    // method member
}
```

Access Specifier

- **public**
  - Accessible from anywhere
- **private**
  - Accessible only from within other member functions of the same class (or “friends”).
  - This is the default access
- **protected**
  - Accessible from members of the same class (or “friends”) and also from members of the derived classes

```
class StudentAccount
{
    private:
       long   m_id ;
       double    m_gpa ;
   public:
       void printGPA() {  }    // any body can call this method
   protected:
       void  setGPA(double newGPAValue) ;
}
```

Constructor

- Used to initialize an object of the class when the object is created
- Special member function that must be defined with the same name as the class
- Cannot specify a return type
- Default constructor: constructor with no parameter

```
class SavingAccount
{
    public:
      SavingAccount()  {   }
      SavingAccount(double initBalance) {   }
}
```

Constructor Rule

- Implicitly define a default constructor for every class that does not define any constructor

- If you define any constructor:

  – Cannot specify a return type

```
class SavingAccount
{
    SavingAccount()  {   } // default constructor
    SavingAccount(double initBalance) {   }
}
```

Naming convention

- Class name starts with a capital letter

- Method starts with a lower letter. It is usually named as an action verb to describe the task it performs.

"**const**" keyword

- When "**const**" appears at the end of the method's declaration, it means that the function/method does not modify any data stored in the calling object
- "**const**" appears in both the header declaration and the method definition

```
void printAMessage() const ;
…
void printAMessage()  const
{
}
```

Dot operator (.)

- Supports the method call or accessing to the data member (if its access specifier allows it)

```
BankAccount  acct ;
acct.deposit(100);
acct.withdraw(20);
acct.print() ;
```

“string” class

- Empty string
  - String that contains no character
- “**string**” methods:
  - length(); append(string s);

```
#include <string>

string  s = “Hello, World” ;
cout << s.length() << endl;
cout << s.at(0) << endl;
cout << s.find(“ll”) ;
```

“set” and “get” method

- “set” and “get” methods are used to assign value to or obtain the value from the private data members
- Sometimes are referred as “**mutators**” and “**accessors**”
- Indirectly provide access to the private data members

```
class SavingAccount
{
  double getBalance() ;
  void setBalance(double newBalance) ;
}
```

Member-initializer list

- Initialize the data members with the value of the constructor's parameter name
- Appear between a constructor's parameter list and the left brace that begins the constructor's body
- Separated from the parameter list with a colon (:)

```
class SavingAccount
{
     double  m_balance ;
     SavingAccount();
     //SavingAccount(double initBalance) : m_balance(initialBalance)
    SavingAccount(double initBalance)
   {m_balance = initBalance;
    }
}
```

Information Hiding/Data Abstraction

- Hiding the class's implementation details makes it easier to change the class's implementation, and hopefully eliminating, changes to the client code

- The class is reusable

- The clients do not know how the class's member functions are implemented

```
class SavingAccount
{
    private:
        double   m_balance ;
    public:
        bool    save(double amount) ;
}
```

Important notes

- The "**const**" keyword must appear in both the function prototypes and the function definitions

- Each member function name can be preceded by the class name and the scope resolution operator (::)

- Without scope operator (::), methods are simply recognized as "functions"

```
void     BankAccount::print() const
{
    cout << "Balance: " << m_balance << endl;
}
```

Exercise 9-1:

1. Enhance BankAccount constructor to take another parameter “name” to keep track of the owner name. The print() method must also print the owner’s name together with the balance.
2. Enhance the withdraw() method to print an error message if the amount is > balance and return ‘false’.
3. Enhance the deposit() method to check the amount. If amount <= 0, print an error message and return ‘false’.

Exercise 9-1:

4. Write a main program MyAccount.cpp. The main program creates at least 3 accounts and 3 transactions per account. Then, it calls print() method to print each account.

5. Separate the solution into 3 different files:

    a. Account.h : contains class declarations

    b. Account.cpp: contains the class implementation

    c. MyAccount.cpp: the main program

Exercise 9-2:

1. Define a subclass of BankAccount called SavingAccount with additional data members: m_year_open and m_interest_rate. The constructor should take these additional parameters to initialize these data members
2. Add a method called calculateInterest() which returns a simple interest calculation:

   interest = number_of_years * rate * principal
3. Enhance the print() method to print both the principal and interest
4. Enhance the main program in Exercise 9-1 to create SavingAccount call these new methods.