Advanced Programming

Zeinab Zali

References: (1) "C++ How to program" Deitel&Deitel, (2) "A Tour of C++" Bjarne Stroustrup, (3) Other useful learning pages such as geeksforgeeks and tutorialpoints

ECE Department, Isfahan University of Technology

Class Members

We want to define a new type (class), for managing a bank-account

- Attributes (data members):
  - name
  - balance
- Methods or behaviors (member functions)
  - getBalance: querying the balance
  - deposit: making a deposit that increases the balance
  - withdraw: making a withdrawal that decreases the balance

Class Definition

```
1   #ifndef ACCOUNT_H
2   #define ACCOUNT_H
3   #include <string>
4   using namespace std;
5   class Account{
6       public:
7           void setBalance(long);
8           long getBalance();
9           void diposit();
10          void withdraw();
11          string name;
12          long balance;
13  };
14  #endif
```

Class Implementation

```
1   #include "Account.h"
2
3   Account::Account(){
4
5   }
6
7   long Account::getBalance(){
8       return balance;
9   }
10
11  void Account::setBalance(long b){
12      balance = b;
13  }
```

The new defined class in action

Classes cannot execute by themselves.

- We must create objects from a class

```
1   #include "Account.h"
2   #include <iostream>
3   #include <string>
4   int main(){
5       Account acc1;
6       //a pointer to a new allocated Account object:
7       Account *acc2 = new Account();
8       //an array of 100 new allocated Account objects:
9       Account acc_arr1[100];
10      Account *acc_arr2 = new Account[100];
11      cout << "Initial acc1 balance: "<< acc1.balance;
12      acc1.setBalance(100);
13      cout << "New balance of acc1: "<< acc1.balance;
14      acc2->setBalance(200);
15      cout << "New balance of acc2: "<< acc2->balance;
16      acc_arr1[0].setBalance(300);
17      acc_arr2[0].setBalance(300);
18  }
```

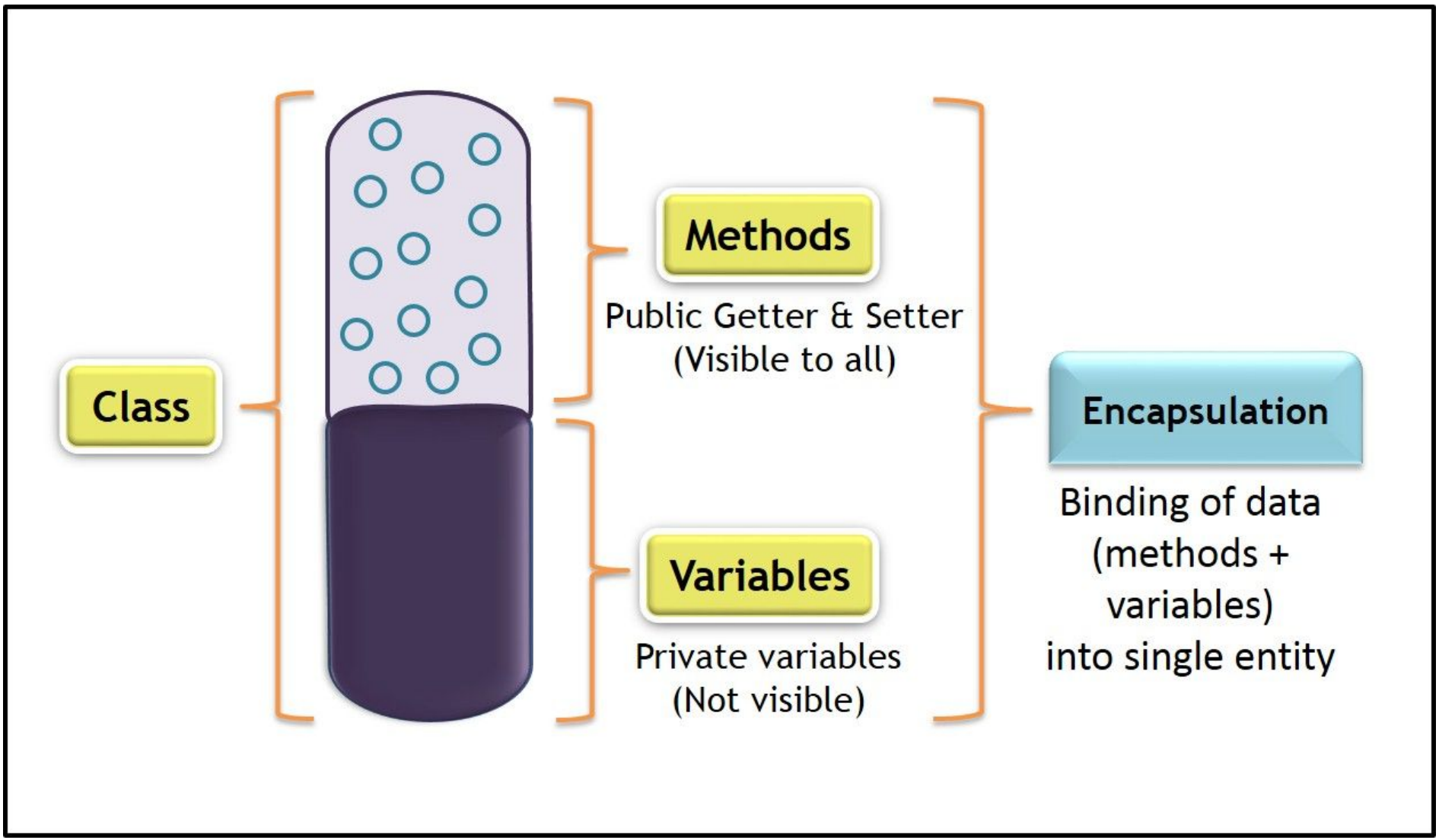
Access Speifier private/public

Try below code for the class definition

- In This way, name and balance are not accessible from the created object.
- Infact all the data and function members of a class are private, unless we declare them as public

```
1   #include <string>
2   using namespace std;
3   class Account{
4           string name;
5           long balance;
6       public:
7           void setBalance(long);
8           long getBalance();
9           void diposit();
10          void withdraw();
11
12  };
```

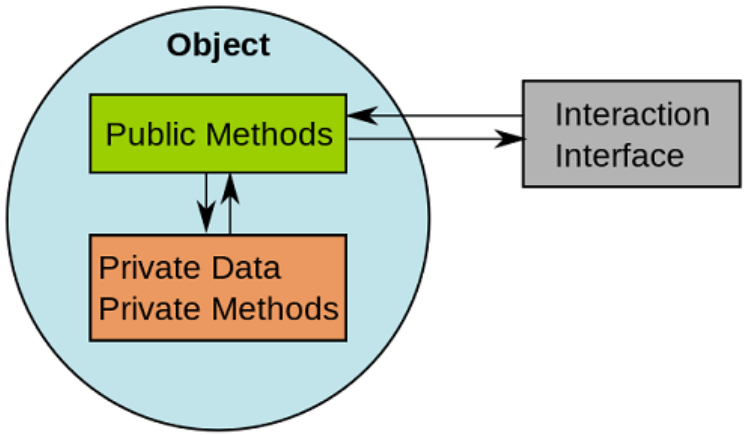
Encapsulation, Why?



Encapsulation, How?

Access specifiers are always followed by a colon (:).

- Declaration of data members or member functions appear after access specifier private: to indicate that they are accessible only to class's member functions.
  - This is known as **hiding** or **encapsulation**
- Data members or member functions listed after access specifier public are "available to the public."
  - They can be used by other functions in the program (such as main), and by member functions of other classes (if there are any).



Constructor, What?

Each class can define a **constructor** that specifies custom initialization for objects of that class

- A constructor is a special member function that must have the same name as the class.
- C++ requires a constructor call when each object is created, so this is the ideal point to initialize an object's data members
- A constructor can have parameters(the corresponding argument values help initialize the object's data members)
- A constructor does not return anything.

Constructor, How?

Implementing constructors

```
1    Account::Account(string str, long val){
2
3        name = str;
4        balance = val;
5    }
6    Account::Account(long val){
7        balance = val;
8    }
```

Using Constructors to initialize objects

```
1        Account acc1("Arman",100);
2        Account *acc2 = new Account(200);
3        Account acc3{"Ali",200}; //creating an account that
     is constructed by Account(string, long), it is same as
     Account acc3("Ali",200)
4        int i{100}; //declaring an integer with value of 3
```

Default Constructor

In any class that does not explicitly define a constructor, the compiler provides a **default constructor** with no parameters

- If you define a custom constructor for a class, the compiler will not create a default constructor for that class

```
1         //below line causes error: no default constructor
    exists for class "Account"C/C++
2       Account acc;
```

Member-Initializer List

A constructor uses a member-initializer list to initialize its data members with the values of the corresponding parameters.

- Member initializers appear between a constructor's parameter list and the left brace that begins the constructor's body

```
1    Account::Account(string str, long val):name(str),
     balance(val){
2
3    }
```

- The member initializer list executes before the constructor's body executes.
- Sometimes this way is the only way of initializing some data members (we see them in future)

Destructor, What?

A **destructor** is another type of special member function.

- The name of the destructor for a class is the tilde character ($\sim$) followed by the class name
- A destructor may not specify parameters or a return type.

```
1    Account::~Account(){
2        cout << "destructor is called\n";
3    }
```

Destructor, How?

- A class's destructor is called implicitly when an object is destroyed.
  - This occurs, for example, as an object is destroyed when program execution leaves the scope in which that object was instantiated.
- Every class has exactly one destructor.
- If you do not explicitly define a destructor, the compiler defines an "empty" destructor.
- We'll build destructors appropriate for classes whose objects contain dynamically allocated memory

this Pointer

Every object has access to its own address through a pointer called **this** (a C++ keyword)

- A common explicit use of the this pointer is to avoid naming conflicts between a class's data members and member function parameters (or other local variables) with the same name.

```
1    void Account::setBalance(long balance){
2        this->balance = balance;
3    }
```

Cascaded Function Calls

Cascaded member function calls is invoking multiple functions sequentially in the same statement.

- Another use of the this pointer is to enable cascaded member-function calls

```
1   Account& Account::withdraw(long val){
2       if(val>=0){
3           balance -= val;
4           transactions[transIndx] = "withdraw " + std
    ::to_string(val);
5           withdrawTrans[wtransIndx] = std::to_string(
    val);
6           if (transIndx==9)
7               transIndx=0;
8           else
9               transIndx ++;
10          if (wtransIndx==9)
11              wtransIndx=0;
12          else
13              wtransIndx ++;
14      }
15      return *this;
```

Cascaded Function Calls

- Cascaded function call:

```
1     cout << "withdraw 100 and 10 units from acc6\n";
2     cout << acc6.withdraw(100).withdraw(10).getBalance()
       <<endl;
3     acc6.getTransaction()[1] = "intrusion"; //dangerous
     ! we can change private transactions array if we don
     't declare return pointer value of getTransaction as
      const
```

- One such usuall example is using multiple « operators with cout to output multiple values in a single statement

static Data Member

In certain cases, only one copy of a variable should be shared by all objects of a class. A **static data member** is used for these and other reasons

- Each object of a class has its own copy of all the data members of the class, except static ones
- A class's static members exist even when no objects of that class exist.
- To access a public static class member when no objects of the class exist, simply prefix the class name and the scope resolution operator (::) to the name of the data member.

static Data Member

- Defining static data member in class definition:

```
1class Account{
2    private:
3        static long count;
4        static long genID;
```

- a static data member must be defined and initialized at global namespace scope, for example in Account.cpp:

```
1    long Account::genID = 0;
2    long Account::count = 0;
```

static Member Function

A **static member function** is a service of the class, not of a specific object of the class.

- To access a private or protected static class member when no objects of the class exist, provide a public static member function
- declaring static member function in calss definiction:

```
1          static long getCount();
2          static void service();
```

static Member Function

- implementing the static member function

```
1    long Account::getCount(){
2        return count;
3    }
4    void Account::service(){
5        //cout << id;    //error, non static data
     member can not be accessed in a static member
     function
6        cout << "Current number of objects from Account
      is " << count << endl; //ok
7    }
```

- call the static function by prefixing its name with the class name and scope resolution operator

```
1    // cout << "Number of created account: " << Account
     ::getCount() <<endl;
2    // Account::service(); // we can also call acc1.
     service();
```

const data members

A class may have const data members, which must be initialized by Member-Initializer List.

- declaring in class definition:

```
1         const long id;
```

- initializing by Member-Initializer List

```
1     Account::Account(long val):id(++genID){
2         if (val>=0)
3             balance = val;
4         else
5             balance = 0;
```

const member functions

C++ disallows member-function calls for const objects unless the member functions themselves are also declared const.

```
1        long getBalance() const;
2        string getName() const;
```

- This is true even for get member functions that do not modify the object.
- This is also a key reason that we've declared as const all member functions that do not modify the objects on which they're called.

```
1    // const Account acc5{"Zahra", 1000};
2    // //error, we can not call a non const member function
      on a const object
3    // //cout << acc5.diposit(20) <<endl;
4    // cout << acc5.getBalance(); //ok, getBalance() is a
     const function
```

Returning Const value

If we return a pointer of private data in a public member function, encapsulation is violated.

- To demonstrate such situation, we added two pointers to array of strings in Account class for storing last ten transactions and last ten withdraw transactions.
- private data members:

```
1        string transactions[10];
2        string *withdrawTrans;
```

- public member functions

```
1        string * getTransaction();
2        string *getWithdrawTrans();
```

Returning Const value

```
1     Account acc6{"Mahsa", 10000};
2     cout<< "withdraw 100 and 10 units from acc6\n";
3     cout<< acc6.withdraw(100).withdraw(10).getBalance() <<
    endl;
4     acc6.getTransaction()[1] = "intrusion"; //dangerous! we
     can change private transactions array if we don't
    declare return pointer value of getTransaction as const
5     cout << acc6.getTransaction()[1] <<endl;
```

Copy Constructor

Copy Constructor is a specialed constructor to create a new copy of an object

- The argument to a copy constructor should be a const reference to allow a const object to be copied.

```
1    Account::Account(const Account& acc):id(acc.id), name(
     acc.name), balance(acc.balance){
2        cout << "copy constructor is called \n";
3        withdrawTrans = new string[10];
4        transIndx = 0;
5        wtransIndx = 0;
6        for(int i=0;i<10;i++){
7            transactions[i] = acc.transactions[i];
8            withdrawTrans[i] = acc.withdrawTrans[i];
```

Copy Constructor

Copy constructors are invoked whenever a copy of an object is needed, such as in

- passing an object by value to a function,
- returning an object by value from a function
- initializing an object with a copy of another object of the same class

Copy Constructor

```
1     //call copy constructor to create acc8
2     //comment the copy constructor, then execute the code
      to see the results by default copy constructor
3     Account acc7{acc6};
4     cout << "acc7 name: " << acc7.getName() << ", id: " <<
      acc7.getId() << ", balance: " << acc7.getBalance()<<
      endl;
5     acc7.withdraw(500);
6     //withdrawTrans is not shared in acc7 and acc6
7     cout << "withdraw transaction[2] of acc6: "<< acc6.
      getWithdrawTrans()[2]<<endl;
8     cout << "withdraw transaction[2] of acc7: "<< acc7.
      getWithdrawTrans()[2]<<endl;
9
10    printInfo(acc7); //copy constructor call in call by
      object
11    Account acc8 = acc6; //copy constructor of acc6 is
      called for initializing acc8
```

- For each class, the compiler provides a default copy constructor that copies each member of the original object into the corresponding member of the new object.
- Dangerous! copy constructors can cause serious problems when used with a class whose data members contain pointers to dynamically allocated memory
  - If we don't provide a copy constructor, the default copy constructor does not allocate new memory for the array in the new copy of the object
  - Our Account class must override the default copy constructor to allocate the memories for the copy instatnce of object.

Pointer Data Members and Copy Constructor

```
1     //call copy constructor to create acc8
2     //comment the copy constructor, then execute the code
    to see the results by default copy constructor
3     Account acc7{acc6};
4     cout << "acc7 name: " << acc7.getName() << ", id: " <<
    acc7.getId() << ", balance: " << acc7.getBalance()<<
    endl;
5     acc7.withdraw(500);
6     //withdrawTrans is not shared in acc7 and acc6
7     cout << "withdraw transaction[2] of acc6: "<< acc6.
    getWithdrawTrans()[2]<<endl;
8     cout << "withdraw transaction[2] of acc7: "<< acc7.
    getWithdrawTrans()[2]<<endl;
9
10    printInfo(acc7); //copy constructor call in call by
    object
11    Account acc8 = acc6; //copy constructor of acc6 is
    called for initializing acc8
```

Composition

A class can have objects of other classes as members. Such a software-reuse capability is called composition (or aggregation)

- Our lovely! Account class can have a Date data member to sotre the expiration date of the account

```
1 class Date{
2     int year;
3     int mon;
4     int day;
5 public:
6     Date();
7     Date(int,int,int);
8     Date(const Date&);
9     int getDay();
10    int getMon();
11    int getYear();
12 };
```

Member object initialization

- If a member object is not initialized through a member initializer, the member object's default constructor will be called implicitly
- The C++ default constructor does not initialize the class's fundamental type data members, but does call the default constructor for each data member that's an object of another class

- expireDate data member initialization:

```
1    Account::Account(long val, Date d):id(++genID),
     expireDate(d){
2        if (val>=0)
3            balance = val;
4        else
5            balance = 0;
```

- Defining an account that has an object of Date class:

```
1    int main(){
2        Date date(1404,12,29);
3        Account acc1;
4        Account acc2("Arman",1000,date);
5    }
```