Lecture 2
Object Oriented Programming I

A paradigm shift

Lecture Overview

- Overview of programming models:
  - Procedural programming
  - Object Oriented programming

- Object Oriented Features in C++
  - Class
  - Object
  - Methods
  - Attributes

Programming Models

- All programming languages like C, C++, Java etc has an underlying **programming model**
  - Also known as **programming paradigms**
- Programming model tells you:
  - How to organize the information and processes needed for a solution (program)
  - Allows/facilitates a certain way of thinking about the solution
  - Analogy: It is the “world view” of the language
- Popular programming paradigms:
  - **Procedural**: C, Pascal, Fortran, etc
  - **Object Oriented**: Java, C++, C#, etc
  - etc

Bank Account : **A simple illustration**

- Let’s look at C implementation of a simple bank account
- **Basic Information:**
  - ***Account Number*****:** an integer value
  - ***Balance*****:** a double value (should be >= 0)
- **Basic operations:**
  - ***Withdrawal***
    - Attempt to withdraw a certain amount from account
  - ***Deposit***
    - Attempt to deposit a certain amount from account
- Using "**`struct`**" (structure) is the best approach **in C**

Bank Account : **C Implementation**

```
typedef struct {
    int acctNum;
    double balance;
} BankAcct;
```

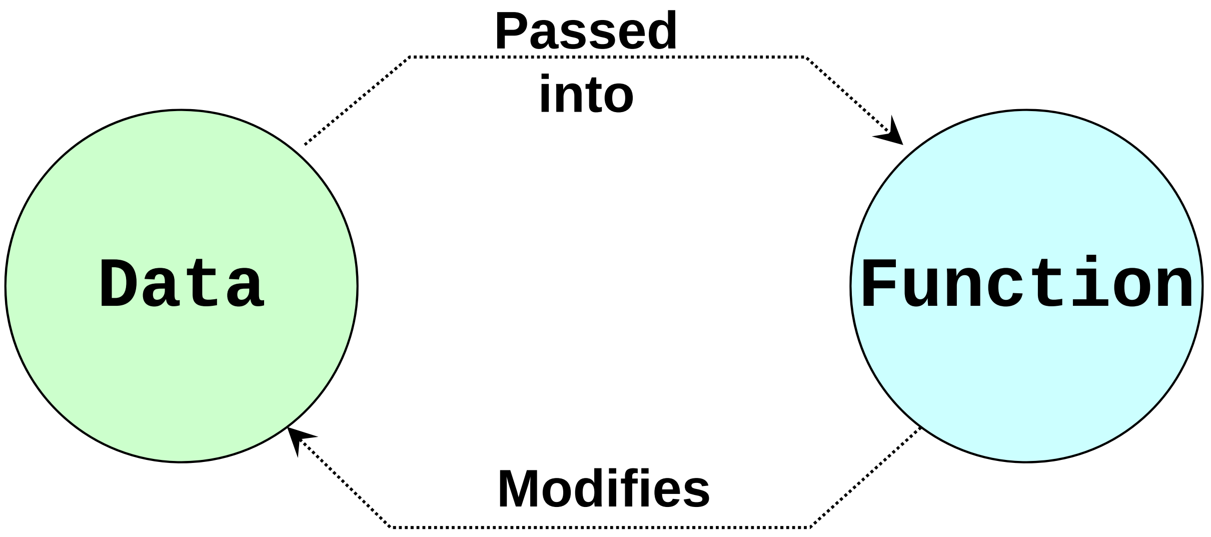
Structure to hold information for bank account

```
void initialize(BankAcct* baPtr, int anum) {
    baPtr->acctNum = anum;
    baPtr->balance = 0;
}
int withdraw(BankAcct* baPtr, double amount) {
    if (baPtr->balance < amount)
        return 0;         // indicate failure
    baPtr->balance -= amount;
    return 1;             // success
}
void deposit(BankAcct* baPtr, double amount) {
    if (amount > 0)
        baPtr->balance += amount;
}
```

Functions to provide basic operations

Bank Account : **C Implementation**

- C treats the data (structure) and process (function) as separate entity:



BankAcct
structure

deposit(...)
withdraw(...)

Bank Account : **Usage Examples**

Correct use of `BankAcct` and its operations

```
BankAcct ba1;

initialize(&ba1, 12345);
deposit(&ba1, 1000.50);
withdraw(&ba1, 500.00);
withdraw(&ba1, 600.00);
deposit(&ba1, -1000.00);
     ...
```

Wrong and malicious exploits of `BankAcct`

```
BankAcct ba2;

deposit(&ba2, 1000.50);

initialize(&ba2, 67890);
ba2.acctNum = 54321;

ba2.balance = 10000000.00;
      ...
```

Forgot to initialize

Account Number should not change!

Balance should be changed by authorized operations only

Procedural language: **Characteristics**

- C is a typical **procedural language**

- Characteristics of procedural languages:
  - View program as a process of transforming data
  - Data and associated functions are separated
    - Require good programming discipline to ensure good organization in a program
  - Data is publicly accessible to everyone

Procedural language: **Summary**

- **Advantages:**
  - Closely resemble the execution model of computer
    - Efficient in execution and allows low level optimization
  - Less overhead during design

- **Disadvantages**:
  - Harder to understand
    - Logical relation between data and functions is not clear
  - Hard to maintain
    - Requires self-imposed good programming discipline
  - Hard to extend / expand
    - e.g. How to introduce a new type of bank account?
      - Without affecting the current implementation
      - Without recoding the common stuff

Object Oriented Languages

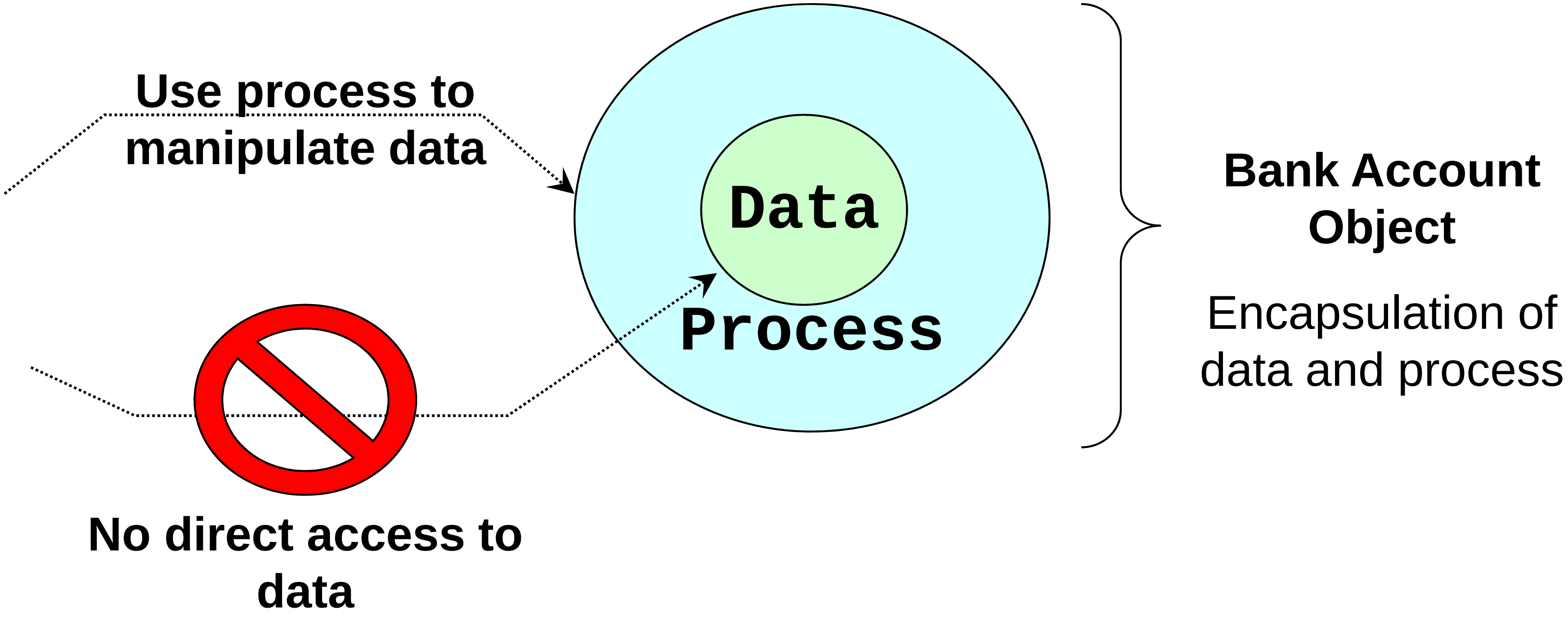
Definition and Motivation

Object Oriented Languages

- Main features:
  - **Encapsulation**
    - Group data and associated functionalities into a single package
    - Hide internal details from outsider
  - **Inheritance**
    - A meaningful way of extending current implementation
    - Introduce logical relationship between packages
  - **Polymorphism**
    - Behavior of the functionality changes according to the actual type of data

Bank Account : OO Implementation

- A conceptual view of equivalent object oriented implementation for the Bank Account


Use process to manipulate data
Data
Process
No direct access to data
Bank Account Object
Encapsulation of data and process

OO language: **Characteristics**

- Characteristics of OO languages:
  - View program as a collection of **objects**
    - Computation is performed through interaction of objects
  - Each object has a set of capabilities (functionalities) and information (data)
    - Capabilities are generally exposed to the public
    - Data are generally kept within the object
- Analogy:
  - Watching a DVD movie in the real world
    - DVD and DVD players are objects with distinct capabilities
    - Interaction between them allows a DVD movie to be played by a DVD player

OO language: **Summary**

- **Advantages:**
  - Easier to design as it closely resembles the real world
  - Easier to maintain:
    - Modularity is enforced
    - Extensible

- **Disadvantages**:
  - Less efficient in execution
    - Further removed from low level execution
  - Program is usually longer with high design overhead

C++ :
Object Oriented Features

What makes C++ Object Oriented

Encapsulation in C++ : Classes

- In C++, a package of **data** + **processes** == **class**
  - A **class** is a user defined **data type**
  - Variables of a class are called **objects**

- Each class contains:
  - **Data**: each object has an independent copy
  - **Functions**: process to manipulate data in an object

- **Terminology:**
  - **Data of a class** :
    - member data (**attributes**)
  - **Functions of a class:**
    - member functions (**methods**)

Accessibility of attributes and methods

- Data and methods in a class can have different level of **accessibilities** (visibilities)
- **`public`**
  - Anyone can access
  - Usually intended for methods only
- **`private`**
  - Only object of the same class can access
  - Recommended for all attributes
- **`protected`**
  - Only object of the same class or its children can access
  - Recommended for attributes/methods that are common in a “family”
  - More on this topic later

Bank Account : C++ Implementation

```
class BankAcct {

private:
  int _acctNum;
  double _balance;

public:
  int withdraw(double amount) {
      if (_balance < amount)
         return 0;
      _balance -= amount;
      return 1;
  }
  void deposit(double amount) {
      if (amount > 0)
          _balance += amount;
  }
};
```

Class name follows normal identifier rule, notice the closing '};' at the bottom

`"private:"` indicates all following definitions have private visibility

We have only private *attributes* in this example

`"public:"` indicates all following definitions have public visibility

Most methods should have public visibility

A method can access *attribute* directly

Bank Account : **Class and Object**

- The class declaration defines a **new data type**
  - No actual variables are allocated!
- To have a *variable* of a class:
  - Create (instantiate) **object**

- The distinction between **class** and **object**
  - Similar to *structure declaration* and *structure variable* in C
  - Analogy: **class** == blue print, **object** == actual house

- To access a **public** attribute or method of an object
  - Use the "**.**" dot operator
  - Similar to structure access in C

Bank Account : Example usage

```
// BankAcct class declaration from previous slide

int main() {
    BankAcct ba1;

    ba1.deposit(1000);
    ba1.withdraw(699.50);

    ba1._acctNum = 1357;
    ba1._balance = 10000000;
}
```

Question: How to initialize?

Interacts with object using **public methods**

Error: Outsider cannot access **private attributes**

Constructors

- The previous implementation for bank account is **incomplete**
  - account number and balance are not initialized
- Each class has one or more specialized methods known as **constructor**
  - Called **automatically** when an object is created
- **Default constructor**
  - Take in no parameter
  - Automatically provided by the compiler if programmer does not define **any constructor method**
- **Non-default constructor**
  - Can take in parameter
  - Can have multiple different constructors

Bank Account : Two Example Constructors

```
class BankAcct {

private:
  //...same...

public:
  BankAcct(int aNum) {
     _acctNum = aNum;
     _balance = 0;
  }

  BankAcct(int aNum, double amt)
  : _acctNum(aNum), _balance(amt) {
  }

  //...other methods are not shown
};
```

Constructor method has the same name as the class with **no return type**

Alternative syntax to initialize object attributes. Known as **initialization list**. Only valid in constructor method.

Bank Account : Example usage 2

```
int main() {
    BankAcct ba1(1234);

    BankAcct ba2(9999, 1001.40);

    BankAcct ba3;
}
```

Make use of 1st constructor

Make use of 2nd constructor

Error: default constructor is no longer valid

- If programmer defines extra constructors:
  - Compiler **no longer provides the default constructor**
  - Programmer have to define default constructor if it is useful

Problem: Print Account Information

- At this point, the `BankAcct` class has some usage problems:
  - Cannot access the account number and balance outside from the class

- Modify the class such that:
  - We can print out the account number and balance as an outsider
  - One possible answer:
    - Implement a simple **`print()`** method for **`BankAcct`** class

What? Where? When? How?

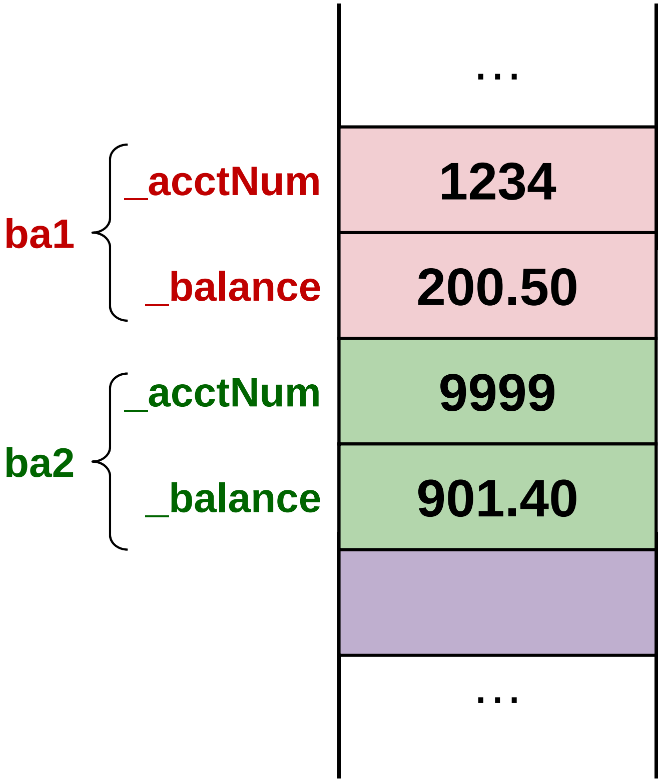
EXAMINING OBJECT

Object : Memory Snapshot

```
class BankAcct {
//... other code not shown ...
int withdraw(double amount) {
    if (_balance < amount)
        return 0;
    _balance -= amount;
    return 1;
}};

int main() {
    BankAcct ba1(1234, 300.50);
    BankAcct ba2(9999, 1001.40);

    ba1.withdraw(100.00);
    ba2.withdraw(100.00);
}
```



		…
ba1	_acctNum	1234
	_balance	200.50
ba2	_acctNum	9999
	_balance	901.40
		…

Object : What is “`this`”

- A common confusion:
  - How does the method “knows” which is the “object” currently executing?

- Whenever a method is called,
  - a **pointer to the calling object** is set automatically
  - Given the name **“this”** in C++, meaning “*this particular object*”
- All attributes/methods are then accessed implicitly through this pointer

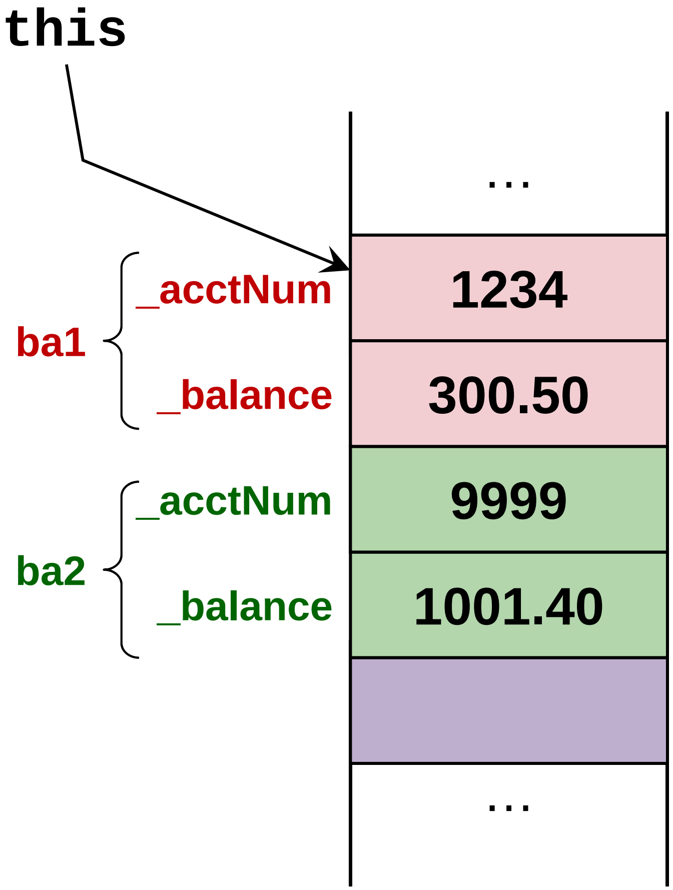
Object : What is "this" (1)

```
class BankAcct {
//... other code not shown ...
int withdraw(double amount) {
    if (_balance < amount)
        return 0;
    _balance -= amount;
    return 1;
}
};

int main() {
    BankAcct ba1(1234, 300.50);
    BankAcct ba2(9999, 1001.40);

    ba1.withdraw(100.00);
    ba2.withdraw(100.00);
}
```

At this point



this

…

ba1

_acctNum 1234

_balance 300.50

ba2

_acctNum 9999

_balance 1001.40

…

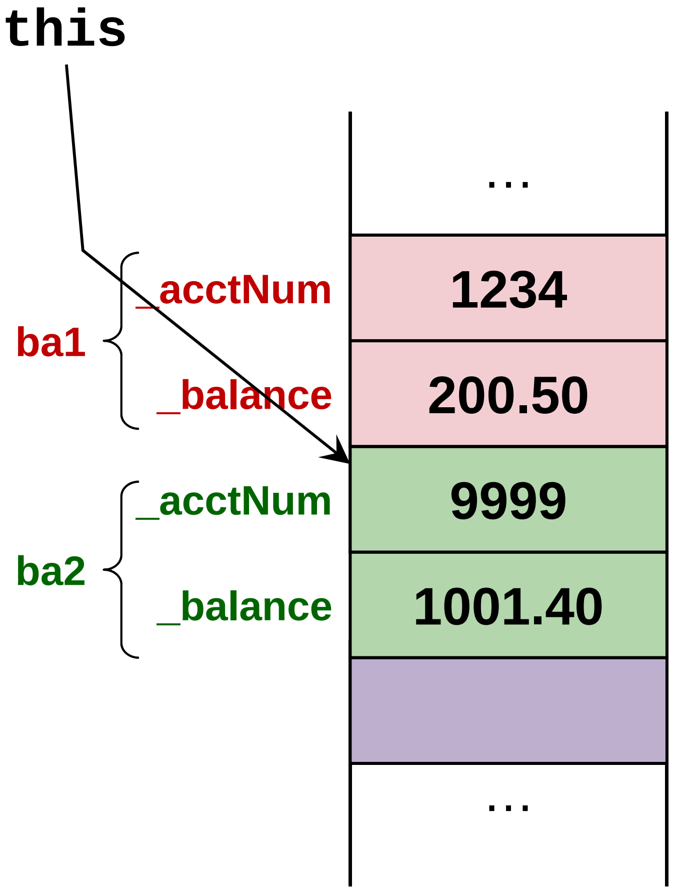
Object : What is "**this**" (2)

```
class BankAcct {
//... other code not shown ...
int withdraw(double amount) {
    if (_balance < amount)
        return 0;
    _balance -= amount;
    return 1;
}
};

int main() {
    BankAcct ba1(1234, 300.50);
    BankAcct ba2(9999, 1001.40);

    ba1.withdraw(100.00);
    ba2.withdraw(100.00);
}
```

At this point



Object : Passed by value

- Objects are **passed by value** (similar to structure in C)

```
// BankAcct class definitions
void transfer(BankAcct& fromAcct,
              BankAcct& toAcct, double amt) {
   fromAcct.withdraw(amt);
   toAcct.deposit(amt);
}

int main() {
    // Simple testing on object passing
    BankAcct ba1(1234, 200.50), ba2(9999, 9001.40);
    transfer(ba2, ba1, 500.00);
}
```

Note that the Bank Accounts are passed by reference (Lecture 1).

Question: What if we remove the “**&**”?

- Additionally, objects tend to contains lots of attributes
  - Recommended to **pass all objects by reference (L1)**
  - Caution: Any function/methods that modifies the object will affect the actual parameter!

Destructor

- **Destructor** is a specialized method of a class
  - Called automatically when
    - Object of the class goes out of **scope**
    - Object of the class get deleted explicitly
- Destructor should be defined for classes that
  - Allocated memory dynamically
  - Requested system resources (e.g. file)
- Syntax for destructor:
  - Method with same name as the class:
    - Prefixed by **~**
    - Empty parameter list and no return type
  - Only one per class
- If destructor is not implemented:
  - A default destructor will be given automatically
    - Suitable for most classes you write in this course

Portion of code delimited by curly braces **{ }**

Destructor : An Example

```
/* class Simple -> */

void f() {
   Simple s(999);
   cout << "End of f()\n";  B
}

int main() {
    Simple s(123), *sptr;

    if (true) {
        Simple s2(456);  A
    }

    f();

C   sptr = new Simple(789);
    delete sptr;

    cout << "End of main\n";
    return 0;
}
```

```
class Simple {
private:
   int _id;
public:
   Simple(int i):_id(i){
       cout << _id << " alive!!\n";
   }
   ~Simple(){
       cout << _id << " died!!\n";
   }
};
```

Output:

```
123 alive!!
456 alive!!     }  A
456 died!!      }
999 alive!!     }
End of f()      }  B
999 died!!      }
789 alive!!     }  C
789 died!!      }
End of Main
123 died!!
```

Life of an Object

- **Allocation ("Birth"):**
  - Happens when:
    - Object declaration or ***new*** keyword is used on object pointer
  - Steps:
    1. The object is allocated in memory
    2. Constructor of the object is called
       - Constructor is chosen base on the parameters provided
- **Alive:**
  - After ***constructor***
  - Object ready to be used
- **Deallocation ("Death"):**
  - Happens when:
    - Object went out of scope or ***delete*** keyword is used on object pointer
  - Steps:
    1. Destructor of the object is called
    2. The memory occupied by the object is returned to the system

OO IN GENERAL

OO Paradigm is not a language!

- Object Oriented Paradigm is:
  - A way to organizing information and process
  - A "worldview" of the programming language

- Even though the examples are in C++, the main ideas can be found in other OO languages:
  - Class, Object
  - Attribute, Methods
  - Visibilities

Other OO Language: Java

```
class BankAcct {

private int _acctNum;
private double _balance;

public BankAcct() {}

public BankAcct(int aNum, double bal) {
    _acctNum = aNum;
    _balance = bal;
}

public boolean withdraw(double amount) {
    if (_balance < amount)
        return false;
    _balance -= amount;
    return true;
}

public void deposit(double amount)
{  ... Code not shown ... }
}
```

Visibility is stated for each attribute

Constructors

Methods

Other OO Language: **Python**

```
class BankAcct:
    _acctNum = 0
    _balance = 0.0

    def __init__(self, aNum, bal):
       _acctNum = aNum
       _balance = bal

    def withdraw(self, amount):
        if _balance < amount:
            return False
        _balance -= amount
        return True

    def deposit(self, amount):
        #code not shown
```

Attribute

Constructor

Methods

Summary

C++ Elements

```
Object Oriented Features:
  - Encapsulation
      class and object
      assessibility
      attribute and method
```

Reference

- **[Carrano]** Chapter 8: Advanced C++Topics

- **[Elliot & Wolfgang]** Chapter P.4, P.5