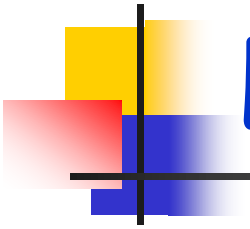
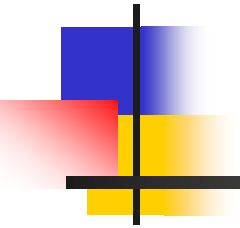
# COMS W3101 Programming Language: C++ (Fall 2016)

Ramana Isukapalli
ramana@cs.columbia.edu

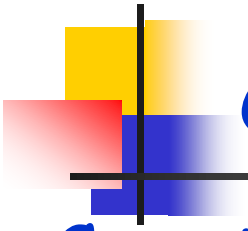


# Lecture-3

- Review
  - Constructor and destructor
  - Data and Member functions
  - Data encapsulation
    - public, private and protected members
- friend functions and friend classes
- Inheritance
- Polymorphism
  - Virtual functions



# Constructor and Destructor



# Constructor and destructor.

## Constructor

- A function with the same name as the class
- Called when an object is created
- A class can have more than one constructor

## Destructor

- Called when an object is cleaned up (goes out of scope)
- One class can have only one destructor

## Examples

```
account x; // constructor code is called
account *y = new account; // constructor code is called
delete (y); // destructor code is called
```

# Constructor and destructor, contd.

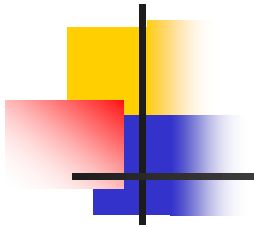
Constructor code

Constructor-1

```
account::account( )
{ user_ssn = -1; accountNumber = -1; }
// OR
account::account( ) : user_ssn (-1),
                      accountNumber(-1) { }
// Constructor-2
account::account (int ssn, int acctNum)
{
      user_ssn = ssn;
      accountNumber = acctNum;
}
```

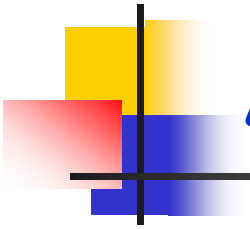
Destructor code

```
account::~account( )
{  // Any memory/resource cleanup, etc. }
```



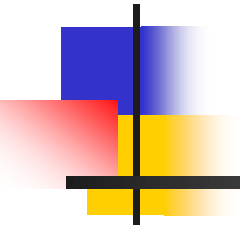
# "Account" example

```
class account
{
  private:
     int user_SSN;              // data
     int accountNumber;         // data
  public:
     account( );                          // Constructor-1
     account(int m, int n);  // Constructor-2
   ~account( );                           // Destructor
     void withdrawMoney (int amount);  // method
     void depositMoney (int amount);     // method:
     void showAccountType( );              // method
};
```



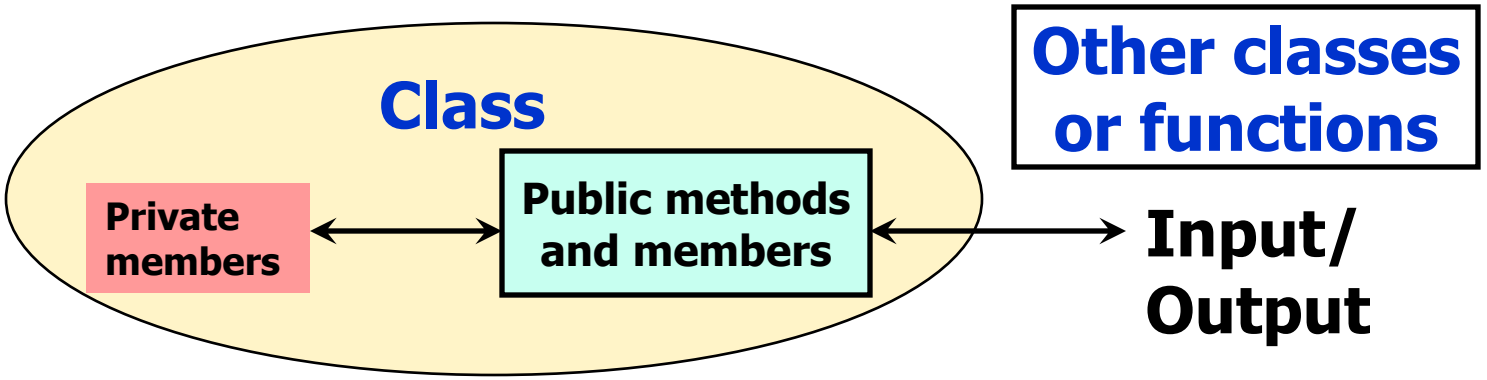
# Account example … contd.

- class has both "data" and "methods".
- Attributes and methods are "members" of a class
- An instance of a class is an object.
- A class should typically correspond to some meaningful entity.
- A class uses methods to interact with other classes/functions.
- private members accessible only to the class (and friends)
- public members are accessible to every class and functions

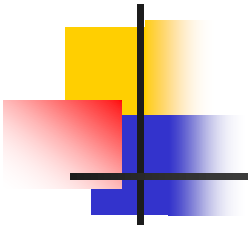


# Data encapsulation

# Data encapsulation … contd.




- Private members are hidden from other classes, fns.
- Public Methods act on data to provide output.
- External classes, functions have access to public methods



# Data encapsulation, contd.

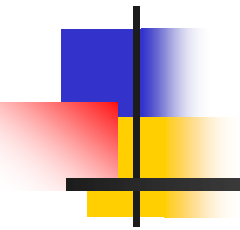
- How can data be hidden?
  - Only class should have access to data
  - Class methods use data
- Define every class member to be one of
  - public – accessible to every the class, other classes, functions and friends
  - private – accessible only to class and friends
  - protected – accessible only to class, friends and children

# Data encapsulation in account example

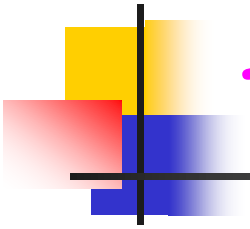
- In an object of account
  - user_ssn and accountNumber are declared private
    - Accessible only to account objects (and friends)
  - Methods are public
    - Anyone can access them.

- Example

```
void function1 ( ) // function, not defined in Account
 {
        account x;
        x.user_ssn = 123; // Will NOT work
        x.showAccountType( ); // Will work
}
```



# friend functions and friend classes



# friend functions

- What if a function genuinely needs to have access to private data?
  - E.g. showAccountInfo (Account acct )
- Need to give access ONLY to that function, not others.
- Use friend function definition
- friend functions of a class have access to private members of the class.

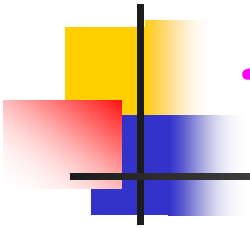
# Example – friend function

```
class account
{
private:
    int user_SSN;
    int accountNumber;
public:
    void deposit (int amount)
    void withdraw (int amount);
friend showAccountInfo
    (class Account);
};
```

```
void showAccountInfo
    (Account acct)
{
    cout << user_SSN << endl;
    cout << accountNumber <<
                endl;
}
```

This is valid.

Friend function can access private members.



# friend class

- Concept of friend can be extended to a class from a function.
- A class gives access to its private members to its friend classes.

```
class account                    class bank
{                                {
  …                                        …
    friend class bank            }
}
```

Members of bank have access to private members of account

# Inheritance – base class & derived class

- **Base class**

```
class account
{
    int user_SSN;
    int accountNumber;
public:
    void deposit (int amount);
    void withdraw (int amount);
    void showAccountType( );
};
```

- **Derived class or child class**

```
class checkingAccount : public account // checkingAccount is
{                                       // derived from account
    int lastCheckCleared;              // not present in account
    void showAllChecksCleared( );      // not present in account
    void showAccountType( );           // defined in both classes
};
```

# Inheritance – base class and derived classes

- Base Class

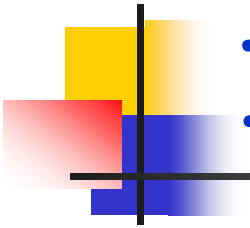
```
class account
{
private:
    int user_SSN;
    int accountNumber;
    int balance;
public:
    account ( ) { }
    account (int ssn, acctNum);
    ~account( ) { }
    void deposit (int amount)
    void withdraw (int amount);
    void showAccountType( );
};
```

- Derived (or child) class-1

```
class checkingAccount : public account
{
public:
    int lastCheckCleared;
    void showChecksCleared ( );
    void showAccountType( )
};
```

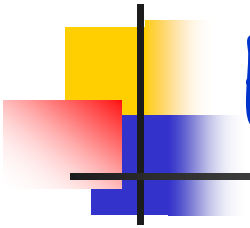
- Derived (or child) class-2

```
class IRA_account : public account
{
public:
    void buyFund (int fund_ID);
    void sellFund (int fund_ID);
    void showAccountType ( );
};
```



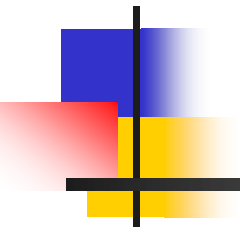
# Inheritance – continued.

- Important points to note:
  - Derived classes have access to members of base classes in this example.
  - Derived classes can have their own members.
    - E.g. lastCheckCleared, showAllChecksCleared( ), buyFund( ), sellFund( ), etc.
    - Members of one derived class are not accessible to another

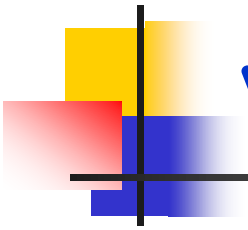


# Examples

- Valid usage in an external function
  - account acct(123456, 5672);
  - checkingAccount ca;
  - acct.deposit (700);
  - acct.withdraw (300);
  - ca.deposit (1000);
  - ca.showAllChecksCleared( )
- Invalid usage in derived class
  - ca.user_SSN = 1234; // Can't access user_SSN
  - ca.accountNumber = 567;



# Polymorphism & virtual functions



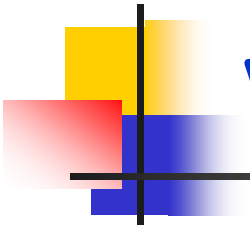
# virtual functions

- Function "void showAccountType( )" is defined in both base and child classes.
  - Supposed to return different values
    - virtual void Account::showAccountType( )
      { cout << "Account" << endl; }
    - void CheckingAccount:: showAccountType( )
      { cout << "Checking Account" << endl; }
    - void IRA_Account:: showAccountType ( )
      { cout << "IRA Account" << endl; }

# virtual functions … contd.

```
main( )
{
   Account *x = new CheckingAccount();
   x→showAccountType( );
   // Will this print "Account" or "Checking
   Account"?
}
```

- This will print
  - Account, if the function is NOT virtual
  - Checking Account, if the function is defined virtual



# Why are virtual functions needed?

- Mainly to enforce class specific functional implementation.
- Should not call base class function from a child object.
- An account object may take different "forms" at different times
  - Checking account, IRA account, etc.
  - showAccountType( ) should compute derived class specific function.

⇒ Polymorphism