CSE 143

Classes

[Chapter 3, pp. 125-131]

4/4/2001 E-1

ADTs: Great Idea, but...

- How do we actually get modularity, abstraction, ADTs, black boxes, etc. in our programs?
- How do we actually encapsulate?
- Main programming construct: the **class**
  - Based on C struct.
  - C structs contain only data
  - ***C++ classes can also contain operations (functions)***

4/4/2001 E-2

A Bank Account Class (I)

```
// Representation of a bank account
class BankAccount {
public:
 // set account owner to given name
 void init(string name);
 // add amount to account balance
 void deposit(double amount);
 // get current account balance
 double amount();

 string owner;     //account holder's name
 double balance;   //current account balance
};
```

4/4/2001 E-3

A Class is a Type

```
BankAccount a1, a2;
```

- The code above creates two instances of the BankAccount class.
- Each instance has its own copy of the data members of the class:

```
owner: "Jack"          owner: "Jill"
balance: 200.17        balance: 940.15
          a1                     a2
```

4/4/2001 E-4

How Do You Access It?

```
BankAccount a1, a2;
```

```
owner: "Jack"          owner: "Jill"
balance: 200.17 a1     balance: 940.15 a2
```

- Access data members just like a struct

```
if (a1.balance == 200.17) … // is True
a2 = a1;                    // allowed
```

- Access member functions ("methods") that way too:

```
a1.deposit(12.75);          // TA payday!
```

4/4/2001 E-5

How Clients Use a Class

- A class is treated like any programmer-defined type. For example, you can:
  - Declare variables of that type:

    ```
    BankAccount anAccount;
    ```
  - Can have arguments (parameters) of that type:

    ```
    void doSomething (BankAccount anotherAccount);
    ```
  - Use one type to build other types:

    ```
    class Bank {
    public:
       . . .
    private:
       BankAccount accounts[100];
    };
    ```

4/4/2001 E-6

A Bank Account Class (II)

```
// Representation of a bank account
class BankAccount {
public:
   // set account owner to given name
   void init(string name);
   // add amount to account balance
   void deposit(double amount);
   // get current account balance
   double amount();
private:
   string owner;      // account holder’s name
   double balance;    // current account balance
};
```

- Some members are *public,* some are *private*

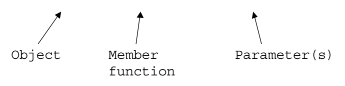
4/4/2001 E-7

Methods

- The class's operations are implemented with functions: "methods"
- To call a method (member function), specify an object (class instance), select the function member with a ‘**.**’, and append a parameter list

```
BankAccount anAccount;

anAccount.init(“Fred Flinstone”);
```



4/4/2001 E-8

Public vs. Private

- **Private** members are **hidden from clients**.
  - The compiler will *not* allow client code to access them.
  - There's a "wall" around them
- **Public** members may be used directly by clients
  - Windows or holes through the wall
- The BankAccount **implementation** can see both
- Trivia: “private” is the default for classes
- For the BankAccount class,
  - How many data members? private? public?
  - How many “methods”?
  - What can the client use directly?

4/4/2001 E-9

Operations on instances

- Most built-in C++ operators DO NOT apply to class instances
- You cannot (for example):
  - use the “+” to add two BankAccount instances
  - use the “==“ to compare to accounts for equality
- To the client, the only valid operations on instances are
  - assignment (“=“)
  - member selection (“.”)
  - plus, can use any operations defined in the public interface of the class.

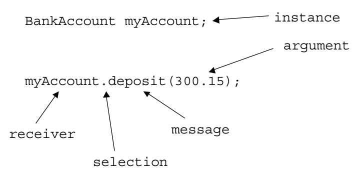
4/4/2001 E-10

Terminology

- Think of a class as a cookie cutter, used to stamp out concrete objects (instances)
- Another view: objects as simple creatures that we communicate with via “messages.” (function calls)

```
BankAccount myAccount;

myAccount.deposit(300.15);
```



4/4/2001 E-11

Information Hiding

- The *private* access modifier supports and enforces information hiding

```
// A client program . . .

BankAccount account;

account.balance = 10000.0;   // NO!  why?
cout << account.balance;     // NO!  why?

account.init(“Jill”);          // ok?
account.deposit(40.0);         // ok?
cout << account.amount();      // ok?
cout << account.amount;        // ????
cout << account;               // ????
```

4/4/2001 E-12

Class Packaging

- C++ allows many legal ways to "package" classes. In CSE143 we generally follow this pattern:
- For each class named X, a pair of files: X.cpp and X.h
  - X.h (specification file)

    ```
    the declaration of only one class X
    maybe some constants
    ```
  - X.cpp (implementation file)

    ```
    #include "X.h"
    contains all the member function definitions and any other
    functions needed to implement them
    ```
- Client programs have #include "X.h"
- Sometimes very closely related classes are packaged together

4/4/2001 E-13

Interface as Contract

The public parts of a class declaration define the ***interface*** *that clients can use.*

Module interface acts as a contract between client and implementer

- Client depends on interface not changing
- Doesn't need to know any details of how module works, just what it does
- Implementer can change anything not in the interface, (e.g. to improve performance)
- Implementation is a "black box" (***encapsulation***), providing ***information hiding***

4/4/2001 E-14

Class Declaration: Interface

```
#ifndef BANKACCOUNT_H     <-- Multiple inclusion hack – more below
#define BANKACCOUNT_H

// Representation of a bank account
class BankAccount {
public:
  // set account owner to given name
   void init(string name);
  // add amount to account balance
   void deposit(double amount);
  // = current account balance
   double amount();
private:
   string owner;       // account holder's name
   double balance;     // current account balance
};

#endif
```

BankAccount.h

4/4/2001 E-15

Building the Class: Implementation (Code)

```
#include "BankAccount.h"
// set account owner to given name
void BankAccount::init(string name) {
   balance = 0.0;
   owner = name;                      scope resolution operator
}                                     need class name here
// = current account balance
double BankAccount::amount() {
   return balance;   <--              but not here
}
// add amount to account balance
void BankAccount::deposit(double amount) {
   balance = balance + amount;
}
```

BankAccount.cpp

4/4/2001 E-16

Implementing Member Functions

- Implementations of member functions use `classname::` prefix
  - indicate which class the member belongs to
  - " `::` " is called the *scope resolution operator*
- Within member function body:
  - Refer to members directly
  - Can access any member, whether public or private!
  - Don't reuse class member names for formal parameters and local variables (bad style)

4/4/2001 E-17

Declaration vs Definition

- In C++ (and C) there is a careful distinction between **declaring** and **defining** an item.
- Declaration: A specification that gives the information needed to **use** an item
  - function prototype
  - class declaration (specification in header file)
- Definition: The C++ construct that actually creates/implements the item.
  - full function w/body

4/4/2001 E-18

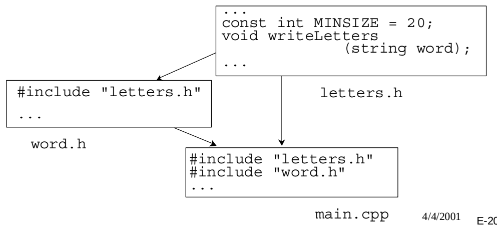
One-Definition Rule (ODR)

- An item (class, function, etc.) may be *declared* as many times as needed in a program (i.e., the same declaration may be #included in many files), but…
- An item must be *defined* (actually created or implemented) ***exactly once*** in a program.

4/4/2001 E-19

Multiple Inclusion

Although an item may be declared in many different compilation units, it is a compile-time error if identifiers (function names, constants, etc.) are declared multiple times in one compilation unit:



```
...
const int MINSIZE = 20;
void writeLetters
                (string word);
...
```

letters.h

```
#include "letters.h"
...
```

word.h

```
#include "letters.h"
#include "word.h"
...
```

main.cpp

4/4/2001 E-20

Multiple Inclusion Hack

- To avoid this problem, use preprocessor directives:

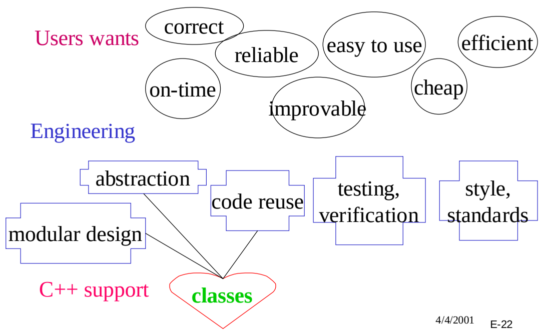
```
// letters.h
#ifndef LETTERS_H
#define LETTERS_H
...
const int MINSIZE = 20;
void writeLetters (string word);
...
#endif
```

Preprocessor directives

- Read the above as:
  "If the symbol `LETTERS_H` has not been defined, compile the code through `#endif` (and define `LETTERS_H`), otherwise skip that code"
- Effect: the header is only processed the first time it encountered (#included) when compiling a particular source file

4/4/2001 E-21

Classes in the Big Picture



4/4/2001 E-22

Summary

- `class` construct for Abstract Data Types
  - Function members (operations)
  - Data members (representation)
- `public` vs. `private` members
- Specification vs Implementation
  - Related concept: Declaration vs Definition
  - Implementation signaled by `classname::`
  - Implementations can access all members, public or private
  - Clients can only access public members
- Clients generally have multiple instances of a few classes

4/4/2001 E-23