

CSE 143

Classes

[Chapter 3, pp. 125-131]

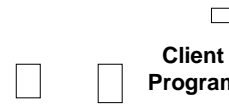
6/26/00 E-1

Bank Accounts (Another ADT Example)

- Data
 - Owner Name
 - Owner SSN
 - Balance
 - Transaction history
 - ...



- Operations
 - Create
 - Deposit
 - Withdraw
 - Balance Inquiry
 - ...



Client Program

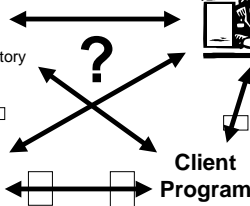
6/26/00 E-2

Bank Accounts (Another ADT Example)

- Data
 - Owner Name
 - Owner SSN
 - Balance
 - Transaction history
 - ...



- Operations
 - Create
 - Deposit
 - Withdraw
 - Balance Inquiry
 - ...



Client Program

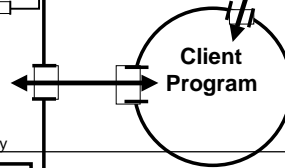
6/26/00 E-3

Bank Accounts (Another ADT Example)

- Data
 - Owner Name
 - Owner SSN
 - Balance
 - Transaction history
 - ...



- Operations
 - Create
 - Deposit
 - Withdraw
 - Balance Inquiry
 - ...



Client Program

Bank Acct ADT

6/26/00 E-4

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**
 - New and major difference between C++ and C
 - Based on C struct.

6/26/00 E-5

Classes vs. Structs

- A lot like a C struct in syntax:

```
class BankAccount {  
    // Class member declarations  
};
```
- Two enhancements support encapsulation
 - Members (= components) can be functions
not just data
 - Can specify *private vs. public* members

6/26/00 E-6

A Bank Account Class (I)

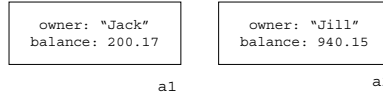
```
// Representation of a bank account
class BankAccount {
public:
    // set account owner to given name
    void init(char name[]);
    // add amount to account balance
    void deposit(double amount);
    // get current account balance
    double amount();
    char owner[30]; //account holder's name
    double balance; //current account balance
};
```

6/26/00 E-7

A Class is a Type

```
BankAccount a1, a2;
```

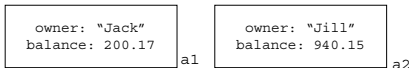
- The code above creates two instances of the BankAccount class.
- Each instance has its own copy of the data members of the class:



6/26/00 E-8

How Do You Access It?

```
BankAccount a1, a2;
```



- Access data members just like a struct

```
if (a1.balance == 200.17) ... // is True
a2 = a1; // allowed
```
- Access member functions that way too:

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

6/26/00 E-9

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];
};
```

6/26/00 E-10

A Bank Account Class (II)

```
// Representation of a bank account
class BankAccount {
public:
    // set account owner to given name
    void init(char name[]);
    // add amount to account balance
    void deposit(double amount);
    // get current account balance
    double amount();
private:
    char owner[30]; // account holder's name
    double balance; // current account balance
};
```

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

6/26/00 E-11

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?

6/26/00 E-12

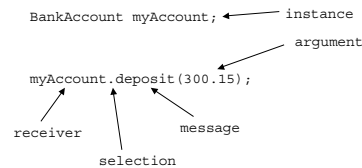
Operations on instances

- 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.

6/26/00 E-13

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)



6/26/00 E-14

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; // ???
```

6/26/00 E-15

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

6/26/00 E-16

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**

6/26/00 E-17

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(char name[]);
    // add amount to account balance
    void deposit(double amount);
    // = current account balance
    double amount();
private:
    char owner[30]; // account holder's name
    double balance; // current account balance
};

#endif
```

BankAccount.h

6/26/00 E-18

Building the Class: Implementation (Code)

```
#include "BankAccount.h"
// set account owner to given name
void BankAccount::init(char name[]) {
    balance = 0.0;
    strcpy(owner, name);
}
// = current account balance
double BankAccount::amount() {
    return balance;
}
// add amount to account balance
void BankAccount::deposit(double amount) {
    balance = balance + amount;
}
```

BankAccount.cpp

6/26/00 E-19

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)

6/26/00 E-20

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

6/26/00 E-21

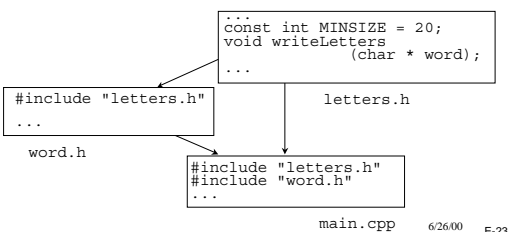
One-Definition Rule (ODR)

- 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.

6/26/00 E-22

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:



Multiple Inclusion Hack

- To avoid this problem, use preprocessor directives:

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

Preprocessor directives

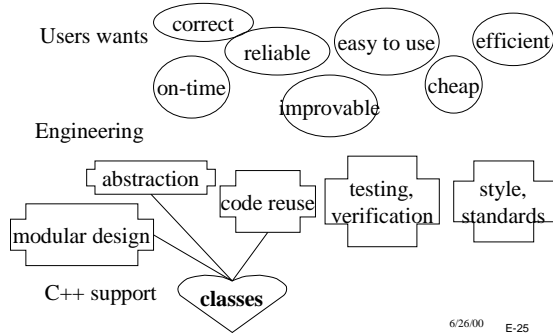
- Read the above as:

"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

6/26/00 E-24

Classes in the Big Picture



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

6/26/00 E-26