

OO Language Features

- Encapsulation
 - Information hiding
- Inheritance
 - Defining a class in terms of another class
- Polymorphism
 - Different definitions for the same method name

Encapsulation

- Classes *encapsulate* a concept
- Class members (variables and functions) have controlled access
- In C++, use access specifiers:
 - public: outside access possible using ‘.’
 - private: for use inside the class only
 - protected: access possible in derived classes

Inheritance

- Objects in the real world have a natural hierarchy (“is-a” relationships)
 - A student is a person
 - A checking account is a bank account
- In C++, :
 - class X: public Y { ... }
 - public and protected members in Y are accessible in X
 - X is called a subclass or a derived class

Polymorphism

- “Many forms”
- Real world analogy:
 - Different types of objects move in different ways
 - Each object type has its own “move()” method
- Polymorphism implies that the language can distinguish which definition to use

Bank Account Application Example

- Check course website for source code
- Demonstrates encapsulation, inheritance, and polymorphism in C++
 - Observe use of access specifiers (e.g., private)
 - checkingaccount is a subclass of bankaccount
 - withdraw() defined in both classes

Separate Compilation in C++

CS 1044
Summer I 2004

What you have been used to so far in this course

- All code is in one source file
- Encapsulation feature not truly exploited
- Is your code reusable?
 - Yes, but you have to cut and paste
 - Better if code to be used elsewhere (or in multiple contexts) is isolated in a separate file
- Some source files using separate compilation are available in the website

Placing Code in Separate Files

- Place prototypes and definitions in a **module.h** file
- Place the actual code in **module.cpp**
- **#include “module.h”** in the program (separate file) that uses the module
- Make all files part of a project
- Include the **module** files in all other projects that use it

Separate Compilation Example 1

- factorial.h:
 - Contains the line **int factorial(int num);**
- factorial.cpp:
 - Contains the function definition:
int factorial(int num) { ... }
- someapplication.cpp:
 - Contains the line **#include “factorial.h”**
 - Contains the **main()** function that invokes the factorial function

Separate Compilation Example 2

- When defining classes, it is more typical to have a class header that does not define member functions
 - Only prototypes are included in class definition
 - Actual code for the function are defined separately (there are special syntax details)
- Convention:
 - .h file contains class header
 - .cpp file contains member function definitions

Example 2 Continued

- Bank account files
 - bankaccount.h, bankaccount.cpp
- Checking account files
 - checkingaccount.h, checkingaccount.cpp
 - Includes bankaccount.h
- Application file
 - application.cpp
 - Includes bankaccount.h and checkingaccount.h

Try the examples out

- Download the files
- Per example, create a .NET project that contains the different files (use “Add Existing Item”)
- See what happens if you remove the **#include** lines
- Try creating new projects that use the existing modules
