Section 1.4
Class Definitions

1. Class members
2. Access specifiers
3. Member function implementation
4. Code organization
5. Variable scope
6. Namespaces

1.4.1 Class Members

- A *class definition* includes:
  - a class name
  - class members
    - data members
    - member functions
  - access specifiers of all class members
    - public, protected, private
  - default access specifier is *private*

1.4.2 Access Specifiers

- Industrial grade software is typically huge
  - potentially millions of lines of code
  - possibly hundreds of developers over many years
  - you must protect your runtime objects from bad code

- The philosophy:
  - to protect the content of your classes, you must restrict access
    - principle of least privilege
    - ... more on this later...

Access Specifiers (cont.)

- Class definition specifies access level for:
  - every data member
  - every member function

- Access levels:
  - public
  - protected
  - private

Access Specifiers (cont.)

- Public access
  - class member is visible by all objects and global functions
- Protected access
  - class member is visible by objects of sub-class types only
  - this access level only makes sense when using inheritance
- Private access
  - class member is not visible to objects of other class types
  - other objects of the *same class* can access private members

1.4.3 Member Function Implementation

- What is a function implementation?
  - the code for a function
    - the body of the function, the statements inside the braces

- For very small programs
  - function implementations can be inside class definition
  - this is the only way to define classes in Java
  - it gets messy very quickly

- For all other programs
  - function implementations should be in a **separate** file
  - function prototype must be included in class definition

Member Function Implementation (cont.)

- So where does your code go?

- Each class is defined using two files:
  - a header file
    - contains *class definition*
    - class definition contains:
      - data member declarations
      - member function prototypes (**not the code**!)
  - a source file
    - contains *member function implementations* (the actual code)
    - static data member initialization
      - ... more on this later ...

1.4.4 Code Organization

- Basic principles
  - class users
    - the developers who use your class in their program
  - remember
    - very few professional developers code directly for end users
    - you must learn to write code for other developers
  - what your class users need to know
    - class name
    - public members
    - sometimes protected members too

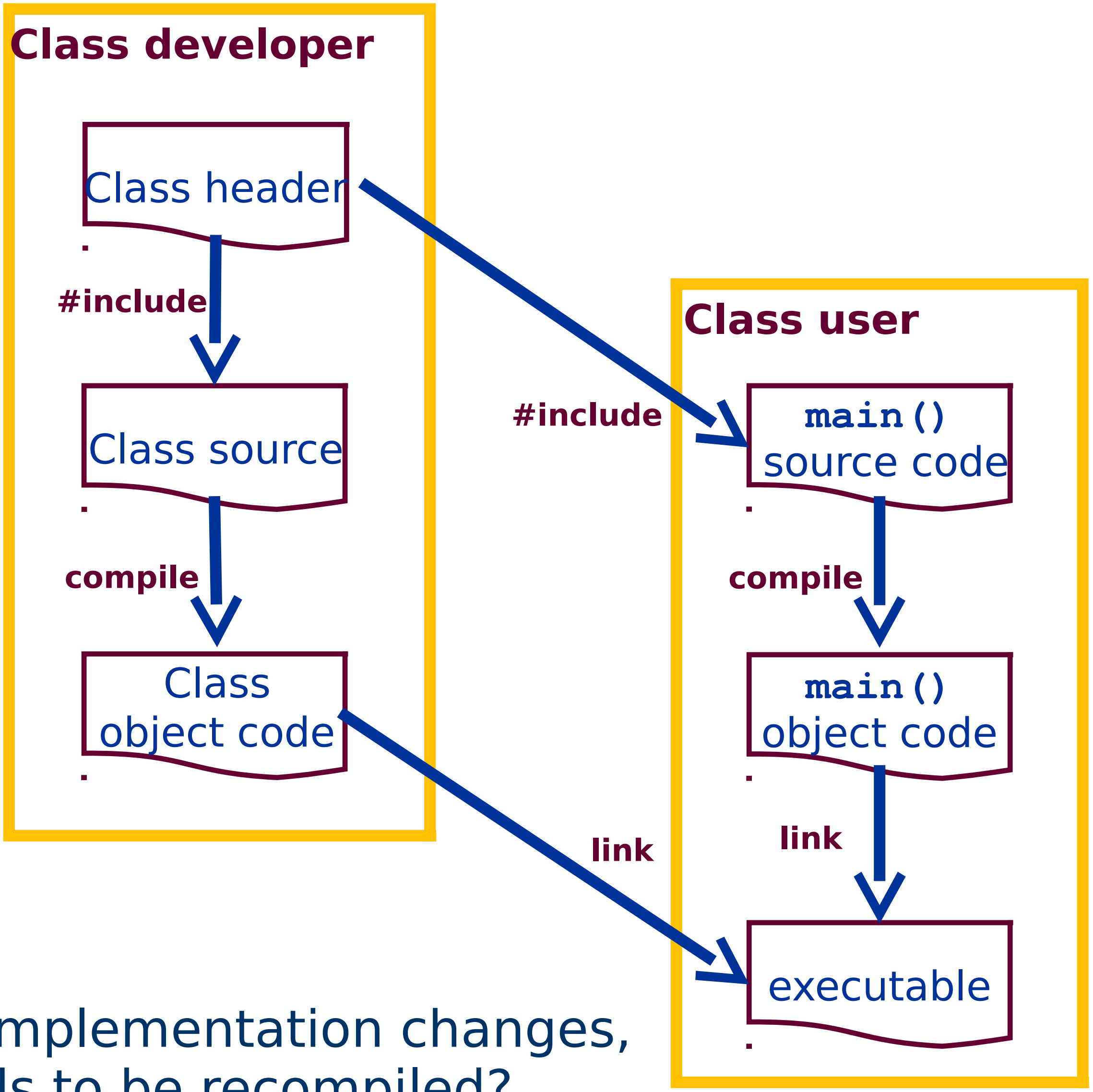
Class Interface

- What is a class interface?
  - not a Java interface!
    - Java uses the word *interface* in a non-standard way
  - in OO design, a *class interface* is what your class users need to know:
    - the class name
    - the class's public members

Class Interface (cont.)

- Your class users need:
  - class definition
    - contained in class header file
    - header file must be included in the class user's code
      - using the C++ **`#include`** preprocessor command
  - class object code
    - contains the class source code, after it is compiled
    - class users need the object code to *link* into their program
- Your class users do **not** need:
  - class source code

Class Interface (cont.)



- Think:
  - if a class implementation changes, what needs to be recompiled?

Class Header File

- Header file contains class definition:
  - data members
  - member function prototypes
- Use include guards!
  - protect against multiple **`include`**s and class re-definitions

Class Header File (cont.)

- **NEVER `#include`** source code !!!
    - you would be forcing re-compilation of all class user code
        - compilation is slow and error-prone
    - understand what belongs in header file vs. source file
    - you must:
        - #1- compile each source file separately into object code
        - #2- link all the object code into one executable
    - class users will have to re-link their code to yours, which is fast

Class Source File

- Source file contains all class-related source code:
  - all member function implementations
    - warning: by default, all functions are **global**
    - use *binary scope resolution operator* to resolve function to its class
  - static data member initializations
    - ... more on this later...

1.4.5 Variable Scope

- What is variable scope?
  - indicates where in the program a variable is visible
- Important types of scope:
  - block
  - file
  - others that are seldom used

Variable Scope (cont.)

- Block scope
  - a block
    - a sequence of statements between a pair of braces
  - a variable declared inside a block has *block scope*
  - it is only visible within that block
  - local variables disappear at the closing brace
  - variables in nested blocks can hide variables in outer block
    - generally, we should avoid reusing variables with the same name
    - use the scope resolution operator to access the global value

Variable Scope (cont.)

- File scope
  - a variable declared outside of any block has *file scope*
  - it is visible everywhere in that file
  - examples:
    - global variables, global functions
  - can be accessed from another file
    - other file must declare it using the **`extern`** keyword

1.4.6 Namespaces

- What is a namespace?
  - it defines a self-contained scope

- Characteristics
  - it groups together a set of:
    - variables
    - functions

- A namespace is **not** a class!
  - it occupies no memory
  - no instances can be created or destroyed

Namespaces (cont.)

- To be used, a namespace must be *scoped in*
  - with **`using`** keyword
  - with binary scope resolution operator

- A namespace may be unnamed
  - it is automatically scoped in