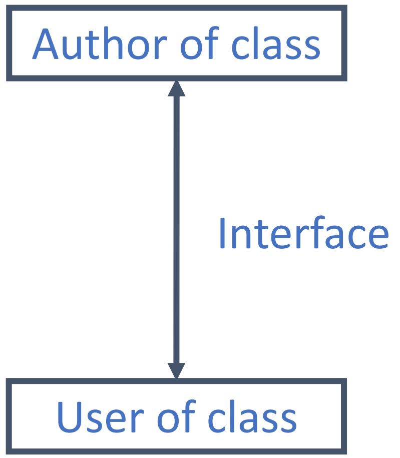
Programming Languages and Tools: Programming with C++ CS:3210:0003

Lecture/Lab #13

Encapsulation via Classes

- Interface: public member functions to interact with objects
  - Must be stable (minimal changes)
- Implementation: member variables + bodies of member functions
  - Hidden, easier to change without breaking code
- Data hiding separates interface and implementation
- Confusingly, encapsulation refers both to:
  1. Data hiding
  2. Bundling of data and functions together



Encapsulation in C++

- Data members are private
- Member functions are public, allowing access to data members
- Benefits:
    1. User of class can avoid implementation details
       Ex: std::string
    2. Allows developer to maintain class invariants
       Ex: in a class modelling a tennis game score, a desirable invariant is that both players can never win the same point or game
    3. Change implementation details without breaking user code

Constructors

- A class type with private member variables
  - Is not an aggregate type
  - Cannot be aggregate initialized
    `ClassName Obj {x, y}; //not allowed`
- Constructor: special member function automatically called after a non-aggregate type object is created
- When object is defined:
  - Compiler looks for a matching constructor
  - If found, memory for object is allocated and constructor called
  - Otherwise, compilation error

Constructors

- Same name as class
- No return type
- Usually public
- Initializes member variables, among other things
  - Using member initializer/initialization list
  - Syntax:
    ```
    constrName(args)
      : privMem1 { initValue1 }, …, privMemn { initValuen }
      { constrBody }
    ```
- Members initialized in the order in which they are defined in the class

Member Initialization

- Priority order for class member initialization:
  1. If a member is listed in the member initializer list, that initialization value is used
  2. Otherwise, if the member has a default initializer, that initialization value is used
  3. Otherwise, the member is default initialized

Function Overloading

- Multiple functions with the same name,
  - as long as they have different parameter types
- Compiler performs overload resolution to match function call to overloaded function
- For successful compilation:
  1. Each overloaded function has to be differentiated from others by either
     - Number of arguments
     - Type of arguments
  2. Each call has to resolve to an overloaded function
- Constructors can also be overloaded

Default Constructor

- Default constructor is either:
  - Constructor with no arguments
  - Constructor with all default arguments
- Class should have only one default constructor
- Implicit default constructor:
  - Generated by compiler
  - Generated when non-aggregate class has no user-declared constructors
  - No args, no member initialization list, empty body
  - Usually used with classes with no data members

Temporary Objects

- Temporary/anonymous/unnamed object: nameless object that exists only for the duration of a single expression
- Syntax:
  1. `className {initializers}`
  2. `{initializers}        //Implicit conversion`

Copy Constructor

- Initialize object with existing object (of same type)
- New object is a copy of object passed as initializer
- Implicit copy constructor initializes each member by copying corresponding member
- Explicit copy constructor:
  - Argument must be reference to object
  - Prefer const reference
  - Should not do anything other than copying object
- Use `= default` to generate default copy constructor
- Prefer implicit copy constructors
- Use `= delete`, if you don't want to allow copy constructors
- Compiler optimization that removes unnecessary copying of objects