Specializing Classes

- Often, classes have similar operations or data
- Maintaining such classes separately is wasteful
- Can put common functionality in a base class, and put class-specific features in derived classes that inherit from the base class
- Then, further extensions in new directions can easily be made
Derived Classes

• The declaration
  
  ```
  class derived-class : access-spec
  base-class {
    access-specifier_1 : decl-list_1
    access-specifier_2 : decl-list_2
    ...
  };
  ```
  
  allows derived-class to inherit from base-class, according to access-spec

• Do not re-declare inherited members unless you intend to override them

Inheritance

• Normally, inheritance is public
• public members of the base class are available in the derived class
• private members of the base class are not available to the derived class
• A public or protected member variable cannot be declared in both
• A public/protected member function can be, but don't do it unless base function is virtual (coming up)
Protected members

• protected members are available only to derived classes, and are considered protected in the derived class as well
• Using the protected specifier is the way to allow inheritance for members that would normally be private if declared in the derived class
• Use private for members that are used only in their own class

Constructors

• Constructors for derived classes can be implemented in the same way as for base classes
• They first call the default constructor for the base class (that has no arguments)
• Can select a different base constructor
• Form: deriv: : deriv (deriv-args) : base (base-args) compound-stmt
  where deriv is the derived class and base is the base class
Virtual Functions

• To have member function in derived class override that of base class, declare base class function **virtual**: 
  
  ```cpp
  virtual type name(args);
  ```

• *Always* make base destructor **virtual**, or derived destructor may not be called!

• Add `= 0;` to end of `virtual` declaration to make function **pure virtual**; this makes base class an **abstract class** (can't create objects)

Next Time

All about templates
• Template functions
• Template classes
• Specializing Templates