Review

Nested objects:
  Initializing: member initializer list
  Construction: inside-out
  Destruction: outside-in

Example:
class Circle {
  private:
    Point center;
    int radius;
    string label;

  public:
    Circle() : center(0, 0), radius(1), label("NONE") {
      return;
    }

    Circle(Point _center, int _radius, string _label) :
      center(_center), radius(_radius), label(_label) {
      return;
    }
};

Inheritance
  Two Uses:
    1) Code Reuse
    2) Polymorphism

Code Reuse
  Existing class provides functionality that we need in a new class
  Two techniques for reusing an existing class: 1) Composition 2) Inheritance
  Composition (a.k.a. Delegation)
    SalesTaxCalculator for U.S. zip codes (use in e-commerce web site)
  (Private) Inheritance
    Stack class (or ClassRoll class) could inherit from ArrayList class
    Composition requires code to instantiate delegate objects
    Instead, we could inherit code from super-class without modification
    May want to use private inheritance to hide subclassing relationship
    The existing class may do something similar to what we need, but not exactly.
    In this case we can:
      Override super-class methods in the subclass and make them behave differently
      Add processing before/after calling super-class method
      Totally replace super-class method in subclass (i.e., don’t call super-class
      method at all)
      Add new functionality in the subclass (new methods and/or variables)

Existing class does something similar to what you need, but not exactly
  Need a SalesTaxCalculator that handles U.S. and Canada
You would like to modify the existing class to do what you need, but you might not have
  the source code, or you don’t want to risk of breaking existing clients of the class
You could write a new SuperSalesTaxCalculator class that composes SalesTaxCalculator
Or, you could extend the super-class by creating a subclass, overriding methods, adding new variables/methods to extend the super-class

Polymorphism
Super-class defines a concept with corresponding method interface
Subclasses represent specializations of the super-class
  Printer (HP, Lexmark, Xerox, …)
  Shape (Rectangle, Ellipse, Polygon, Curve, …)

SUBTYPING
Subclasses override super-class methods to implement subclass-specific behavior
  Printer::DrawText, Printer::DrawLine, Printer::DrawImage, …
Subclass objects can be substituted for super-class objects without breaking the program (Liskov Substitution Principle)
Subclass methods should be called when invoked through super-class pointer

Code Reuse Example
BoundedStringStack example
Just like a class with nested objects has multiple parts, classes that inherit from other classes also consist of multiple parts (A <- B <- C). C instances consist of three parts (A part, B part, C part)
Private vs. Public inheritance
Construction order: top-down
Destruction order: bottom-up
While this example demonstrates reuse, it is not a good example of polymorphism (BoundedStringStack violates the Stack contract)
Exam question example (what would this program print out?)