C++ Inheritance and Composition Summary

Some content adapted from ‘Absolute C++’ by Walter Savitch

Inheritance (Is a)

- Promotes code reuse
- Is elegant
- And delicate (protected exposes innards to derived classes, change base probably break derived)
- Breaks encapsulation (protected again)

Composition (has a)

- Use member variables instead of deriving from base class
- Delegation code needed
- Much better encapsulation (private verses protected)
- Because of this can change much of member variable classes without causing compilation problems or excessive rewrite

General Rules

- Prefer composition over inheritance
  - Don’t be rigid, know when to use which
    - Composition - “Has A”
    - Inheritance - “Is A”
- Objects: Do not return a reference or a pointer to internal data structures from any member function. Make a copy if needed. Avoid getters and setters if possible
- Objects: Design public interface to be complete and minimal.
  - Defensive programming, hide all that you can. All member variables private. Minimal public functions.
  - Makes it easy to change implementation.


A Leaf
Model as an object


A Tree
Model as an object


A Forest
Model as an object

Summary

- Inheritance and Composition
- Hide Data and minimal public interface
- Virtual functions – ensure most derived version of function called
- Virtual functions – allow list of base class pointers that point to variety of derived objects
- Abstract base classes – force implementation of virtual functions
- Some OO design practice, Employee, Forrest, Liquids, CNU