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

Problems

- Security holes
  - What if I had a method in forest that removes dead trees (those with health=dead). Get an iterator to dead tree and…
    - myTrees.erase(iterator);
  - You just erase the pointer, now have a tree with nothing pointing to it. Memory leak.
  - Good thing all data is private and you returned no pointers or refs to internal data
  - <u>You</u> have control of whether the above happens or not!

Problems

- FORREST How to grow new trees?
  - Currently added by forest
  - Object Oriented (OO) Should really be created by trees themselves
  - Then added to forest
  - Every design has flaws
- TREE.H What about trying to override a method and getting the name slightly wrong?
  - sethealth(season aSeason) instead of setHealth(season aSeason)
  - No warnings from compiler, java uses @override, C++ no such thing.
- Abstract base class will solve this (Tree.h)

Enhancements

- Can form stripped down basis for a forest growth simulator
  - Seasons can be more granular (days for instance)
  - Need a weather object that generates weather based on historical patterns and predicted climate change info
  - Random disaster object as well
  - Location data (struct with x,y, and maybe z location).
  - Trees generate seeds that fall to ground, or are eaten by animals. So need some sort of propagation vector as well.

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