Toward understanding inheritance

- Consider/review the math quiz program we discussed earlier
  - We had three kinds of math questions, easy/medium/hard
  - What do we do to add a new kind, or change definition
    - Edit working program, add code to code that already works
    - Not haphazard, but dangerous: if it ain’t broke, don’t touch

- The “open-closed” principle of program building/design
  - Programs/classes should be open for extension, but closed to modification (Bertrand Meyer and Robert Martin)
  - How do we change code without altering it?
    - This is a goal, in practice we modify some, but minimize

- Inheritance lets us realize the principle (in theory)
What about templates?

- We want to write one sort function that will sort ints, doubles, Dates, anything comparable using operator <
  - Templates let us do this in C++, a templated function is a code factory: generates real code when instantiated
  - Potential code-bloat, but very powerful generic code

- We want a vector of anything, not restricted like StringSet which stores only strings
  - Templated or container classes can hold different kinds of items, again these are code factories
  - tvector is one example of kind of classes in STL, the standard template library (little inheritance)
Use interfaces, not implementations

- See “questface.h” in Tapestry, and classes that implement this interface including MathQuestion and WhatsTheQuestion
  - Quiz can use *any* kind of question! (Really?)
  - What methods must question classes implement?
  - Conceptually, what is a question? Behavior?

- **We have parent, base, or super-class for interface**
  - *Child or Subclasses extend* the parent class (*implement* interface)
  - pointers to parent class, which can point to child object
    - I’m pointing at an animal (could be marsupial or bird)

- Child/sub classes don’t have access to private data/functions, but the data are there (alternative: use protected)
Syntactic details

- **Functions in parent class that are changeable in subclasses** must be declared as `virtual`
  - The “real” function is determined at runtime
  - In parent/base/super class make all functions virtual
  - We must have a virtual destructor that’s implemented

- **For interface only functions, that have no default version**
  - Pure virtual functions, aka abstract functions
  - Syntax requires = 0 in .h file
  - Can’t instantiate an object of an abstract (base) class
    - Doesn’t matter, really, we use pointers to objects