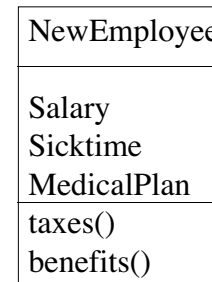


Inheritance Heuristic Revisited

- I was unprepared in last lecture to cover one of the inheritance-related heuristics
- So here is a more complete example, that illustrates the heuristic more clearly

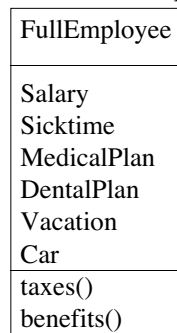
Inheritance Heuristics Revisited

- Consider a start up company...
 - they need a class to store information about employees



Six Months Later

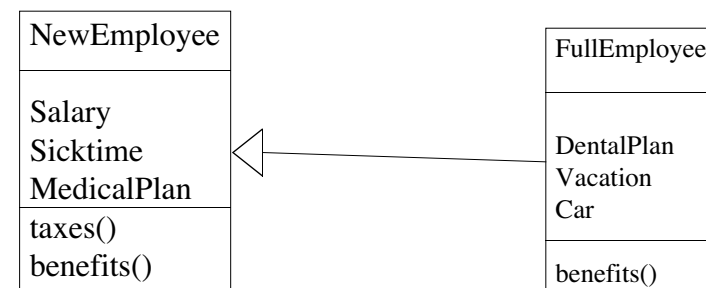
- The company decides to make a distinction between new employees and employees that have been with the company for six months



We notice that the full employee is just a special case of the new employee

so...

Lets use inheritance



Returning to the Heuristic

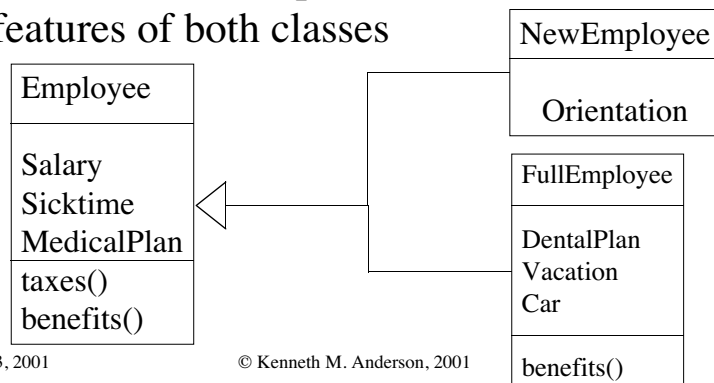
- The heuristic that I had trouble explaining is
 - All base classes should be abstract classes
- This heuristic implies that all the roots of an inheritance tree should be abstract, while only the leaves should be concrete.
 - Why is this a “good thing”?
 - Consider our example...

Adding to NewEmployee

- Assume we decide that all new employees should go to an orientation session
 - we want to add an attribute to track whether an employee has attended the session
 - Can we add this attribute without adding it to the FullEmployee class? (Full Employees either do not need the orientation session or already had it)
 - The answer is no! (because full employee is a subclass of new employee)
 - This is the danger of inheriting from a concrete class
 - (which is the fear that the specialization link between the two classes will not hold up under extension or refinement of the design)

The solution

- Have both classes inherit from an abstract base class, that captures the common features of both classes



Ramifications

- If you violate this heuristic, as we did with this example, you may (probably will) end up in a situation where you need to shift to the abstract base class design
 - Then, you need to introduce a new class, refactor, and change NewEmployee references to Employee references, except when access is needed to the new “orientation” attribute