Students studying for the AP Computer Science exam, starting in the Fall of 2014, should append this text at the end of Chapter Ten, Lesson One.

Object-Oriented Design Approaches

When you are designing software, it is important to have a well-defined *process*, or series of steps, you can follow to ensure success. There are different ways to complete a software design process. Larger projects may have more than one person involved, and each team member may be doing something other than programming to contribute to the project!

Top-Down Design

One common design approach is *top-down design* or *stepwise refinement*. This means you look at the big picture first and start breaking the overall problem down into smaller and smaller sub-systems. You don't need to fill in all the details of every single piece, but you should start with a good idea of the main responsibilities, behavior, and data management requirements for the major sub-systems. This is very similar to functional decomposition.

The top-down design method requires a great deal of planning before any code is ever written. When you do start writing code, the larger pieces are written first to coordinate sub-systems that may not yet exist. In order to get the main pieces to compile, you can write temporary or "stub" implementations of the sub-systems that don't have any real data or behavior, just empty functions.

Once the larger system pieces are completed you can turn your attention to the stubbed-out sub-systems. Each sub-system itself can be subject to top-down design, breaking it into as many pieces as necessary to arrive at a small, well-understood, easily codeable component. Continue in this manner until all of the individual sub-systems are complete.

There are some downfalls to this design approach. One of the biggest pitfalls is that it's hard to test the entire system until all of the code is complete. Any problems that are discovered during test might be difficult to identify and resolve because you have so much new code running at once.

Bottom-Up Design

The opposite of top-down design is *bottom-up design*. In a bottom-up approach, the small individual components are designed or identified first. You can write a complete component with well-defined inputs, outputs, and behavior and test that component without relying on the rest of the system. The individual components are then linked together to form larger systems. This process continues until a complete top-level system is formed.

The obvious benefit of bottom-up design is that you can write and test useful pieces of the system right away, even if some of the higher level components are not yet well defined. However, you might find that without a detailed plan for the whole system, your individual components or sub-systems do not fit together as well as you thought. A programmer might really need to understand the overall system flow in order to write useful sub-systems.

The Combination Approach

What is the best approach to designing your computer programs, top-down or bottom-up design? In reality very few software design processes are purely one or the other. It's important to understand the top-level system architecture and the roles and responsibilities of the major sub-systems. But often small pieces of the system can be identified early and handed off to programmers to begin work in a bottom-up manner. As a result you may have a combination of top-level system components and small "slices" of bottom-level functionality ready for testing early in the process. Once you establish a working system with a few features, you can expand the system by implementing other sub-systems to bring in new features.

