0:14

Welcome back. In the previous module, you were introduced to how design fits into the software development process and how to complete a conceptual design using CRC cards. In this module, we are going to dive deeper into Object-Oriented Modeling. We will start off by talking about modeling problems and how programming languages evolve toward object orientation. Then, we're going to explore four major design principles used in Object-Oriented Modeling. These principles help in problem solving and lead to software that is flexible, reusable and maintainable. These principles are key to having a good design for your software. We'll also show you how to express these design principles using both UML Class Diagrams and Java Code. In this module, you will have your first capstone assignment. The rest of the work for the capstone projects will be completed in the fourth module of each course. However, you will occasionally have capstone assignments in the earlier modules once you've learned the content required to complete that assignment. Ready? Let's get started. If you wanted to make a house, you wouldn't start nailing without a design and just figure out the details later. Similarly, for a complex software problem, you don't dive right into solving it in code. There's a design step in between that iteratively deals with both the problem space and the solution space. You need conceptual design to break down the problem further and further into manageable pieces. You also need technical design to describe and refine the solution, so that it is clear enough for developers to implement as working software. Over the years, people have tried many approaches to make the design activity easier. For example, there are design strategies in programming languages suited for solving certain kinds of problems. If you had a data processing problem, you may have used Top Down Programming. This strategy map the processes in the problem to routines to be called. As you broke down the processing needs top down, you made a tree of routines for the eventual solution. These routines would be implemented in a programming language that supported subroutines. To make design easier, you don't want a big mental jump during design work between a concept in the problem space and how to deal with it in the solution space. If these concepts could be described in a design that made sense to both users and developers, that would be great. This would help ensure the two groups can discuss their understanding and common terms. For many kinds of complex problems, it makes sense to think about the concepts using objects. For example, any noun in a problem description could be an important object. The real world, where problems arise, is just full of objects. This has led to the popularity of Object-Oriented Programming with object-oriented languages. But even here, you still don't go straight from the problem to writing the code. There's a conceptual design involving object-oriented analysis to identify the key objects in the problem. There's also technical design involving object-oriented design to further refine the details of the objects, including their attributes and behaviors. The design activities happen iteratively and continuously. The goal during software design is to construct and refine models of all the objects. These models are useful throughout the design process. Initially, the focus will be on the entity objects from the problem space. As a solution in software arises, you introduced control objects that receive events and coordinate actions. You also introduce boundary objects that connect to services outside your system. The models are often expressed in a visual notation called Unified Modeling Language or UML. In Object-Oriented Modeling, you have different sorts of models or UML diagrams to focus on different software issues, like a structural model, to describe what the objects do and how they relate. It's like having a scale model of a building to understand the spatial relationships. To deal with complexity, you can apply design principles and guidelines to simplify objects. Break them down into smaller parts and look for commonalities that can be handled consistently. There's a continual need to critique and evaluate the models to ensure the design addresses the original problem and satisfies quality goals. Qualities are expected to be reusable, flexible and maintainable. The models also serve as design documentation for your software and can be easily mapped to skeletal source code, particularly for an object-oriented language like Java. That can give a good start for the developers implementing the software.