0:15 As you continue to explore the world of object oriented programming, you must examine the major design principles of object oriented programs. These principles help to define what exactly object-oriented programs are and how you can create them. When you design object-oriented programs, you create models of how objects are represented in your system. These models cannot be designed without forethought. In order for a system to be object oriented, it should adhere to certain design principles. One of the design principles in object oriented modeling I'm going to talk about is abstraction. Abstraction is one of the main ways that humans deal with complexity. Abstraction is the idea of simplifying a concept in the problem domain to its essentials within some context. Abstraction allows you to better understand a concept by breaking it down into a simplified description that ignores unimportant details. For example, we might want to create an abstraction for a food. In a health context, its nutritional value and not its cost would be part of a simplified description of a food.

1:13 Good abstraction emphasizes the essentials needed for the concept and removes details that are not essential. Also an abstraction for a concept should make sense for the concept's purpose. This idea applies the Rule of Least Astonishment. That is, the abstraction captures the essential attributes and behavior for a concept with no surprises and no definitions that fall beyond its scope. You don't want to surprise anyone trying to understand your abstraction with irrelevant characteristics.

1:42 In object oriented modeling, abstraction pertains most directly to the notion of a class. When you use abstraction to decide the essential characteristics for some concept, it makes the most sense to define all of those details in a class named after the concept. A class is like a template for instances of a concept. An object instantiated from a class then has the essential details to represent an instance of some concept. Later on, we will go into more detail of how to form your classes using abstraction.

2:10 Let's take the concept of a person. What are the essential characteristics of a person that we care about? Well, it's hard to say because person is so vague and we haven't said what the purpose of our person is. The abstractions you create are relative to some context, and there can be different abstractions for one concept. For example, if you are creating a driving app, you would care about a person in the context of a driver. In another example, if you were creating a restaurant app, then you would care about a person in the context of a patron. It is up to you to choose the abstraction that is most appropriate for your purpose. Before we start creating an abstraction, we need a context for our person. Context or specific perspective is critical when forming an abstraction. Let's look at an example where our context is an academic setting, and we want to create an abstraction for a student. What are some of the essential characteristics of a student? We'll include the courses they're currently taking, their grades in each course and their student ID number. These are basic attributes for a student. The attributes do not disappear over time although their values may change since they are essential characteristics of a student. For a course, the student's grade value may change but they always have a great attribute. This means the actual values of these attributes may change, but the attributes themselves do not.

3:24 See if you can identify relevant attributes for a concept.

3:28 Give examples of attributes for a house cat from the perspective of a cat owner. A house cat will have basic attributes like a name, color, favorite nap location, and microchip number. Certain values of these attributes could change. For example, over the course of the day the cat's favorite nap location could change from the living room to a bedroom.

3:48 In addition to attributes, an abstraction should describe a concept's basic behaviors. For a student, those behaviors would be studying, doing assignments, and attending lectures. These are the responsibilities that the student abstraction does for its purpose. See if you can identify relevant behaviors for a concept.

4:07 Give examples of behaviors for a house cat from the perspective of a cat owner.

4:12 A house cat has a pretty low activity lifestyle and not much purpose other than to nap. Perhaps you said, having naps, grooming, catching mice in the house, eating, and using the litter box. Within the context of an abstraction, anything other than a concept's essential attributes and behaviors is irrelevant. When considering our student in the context of an academic setting, we don't care whether the student has a pet or how they clean their kitchen or what their favorite video game is. Those are all irrelevant details to the abstraction in this context. Whenever we make abstractions, we need to remember our context. If the context changes, the right abstraction can as well. Say our context changes and we need to model a

student from a social perspective. How would our definition change? Perhaps the relevant attributes would be the student groups they belong to, their hobbies for study breaks, and the sports teams their in. Now it's your turn, let's take a moment to do another example abstraction.

5:08 What is an abstraction for a dog from the perspective of a dog owner? Well your abstraction may be slightly different. You probably defined attributes like the dog's breed, its size, whether it has long or short hair, pointy or floppy ears and its color. Behaviors of a dog include sleeping, eating and doing tricks. Overall, abstraction is an important principle you use when solving problems and designing your systems. Some of its benefits are simplifying your class design so they are more focused, succinct and understandable to someone else viewing them. As you have learned though, abstractions are formed within a specific context for perspective and you have to carefully decide what is relevant. If the purpose of your system or the problem changes, don't be afraid to update your abstractions accordingly. Abstractions are not a fixed creation, but are a direct result of the problem for which you created them.