0:00 [MUSIC]

0:15 The idea behind object-oriented modeling and programming is to create computer representations of concepts in the problem space. It lets you model the relative attributes in behaviors so that a computer can simulate them. One design principle called generalization, helps us to reduce the amount of redundancy when solving problems. You probably already have experience with a form of generalization and don't even know it. Many behaviors and systems in the real world operate through repetitious actions. We can model behaviors using methods. It lets us generalize behaviors and it eliminates the need to have identical code written throughout a program. Take this array creation and initialization code for instance. We can generalize repetitious code that we would need to write by making a separate method and calling it. This helps us to reduce the amount of near identical looking code throughout our system. Methods are a way of applying the same behavior to a different set of data. Generalization is frequently used when designing algorithms, which are meant to be used to perform the same action on different sets of data. We can generalize the actions into its own method, and simply pass it through a different set of data through arguments.

1:19 So where else can we apply generalization? Well, if we can reuse code that is inside a method and a method is inside a class, then can we reuse code from a class? Can we generalize classes? Generalization happens to be one of the main design principles of object-oriented modeling and programming. But it's achieved differently than what we've just seen with methods. So how is this done? Generalization can be achieved by classes through inheritance. In generalization we take repeated, common, or shared characteristics between two or more classes and factor them out into another class. Specifically, you can have two classes, a parent class and a child class. When a child class inherits from a parent class, the child class will have the attributes and behaviors of the parent class. You place common attribute and behaviors in your parent class. There can be multiple child classes that inherit from a parent class, and they all will receive these common attributes and behaviors. The child classes can also have additional attributes and behaviors, which allow them to be more specialized in what they can do. In standard terminology, a parent class is known as a superclass and a child class is called the subclass.

2:27 Let's say you want to model an adorable cat named Mittens. Mittens has four legs, a tail, knows how to walk, run and eat, but these attributes and behaviors can also be used to describe Doug the dog. Doug also has four legs, a tail, can walk, run and eat. If you were to design classes, cat and dog, based on these characteristics, you would have a lot of overlapping code. What would happen if you want to model another similar characteristic? You would need to add the same code to both of your classes. This doesn't sound too bad because you only have two classes, but what if you have several classes that share common characteristics? It would be time consuming and error-prone to carefully add code to all of them. That leads to a system that is not flexible, maintainable, or reusable. Let's look at Mittens and Doug to see what commonalities we can find between them. I would say they are both a type of animal with legs, a tail and have a shared set of behaviors like, walking, running and eating. An animal, then, is a general idea. This means that an animal is a broad term used to describe a large grouping of more distinct classes. The term animal can be used to define a set of common characteristics and behaviors that belong to different specific types of animals, like cat and dog. In this example, we can generalize the common attributes and behaviors of the cat and dog class into a superclass that we will call animal. Keep in mind that we can name these classes whatever we want, but since we're creating meaningful obstructions of things in the real world, our classes should be named after the things we are trying to model. This makes our code easier to understand. The subclasses will inherit attributes and behaviors from the superclass. Since cats and dogs are both animals, they inherit from the animal superclass. One of the advantages of doing this is that any changes to the code that is common to both subclasses, can be made in just a superclass. The second benefit is that we can easily add more animals to our system, without having to write out all the common attributes and behaviors for them. Through inheritance, all subclasses of the animal class will be endowed with the animal classes attributes and behaviors.

4:26 Inheritance and methods exemplify the generalization design principle. There are techniques that left us apply a rule called D.R.Y., which stands for Don't Repeat Yourself. We can write programs that are capable of performing the same tasks but with less code. It makes code more reusable because different classes or methods can share the same blocks of code. Systems become easier to maintain because we do not have repetitious code. Generalization will help you build software that is easier to expand, easier to apply changes to and easier to maintain. By learning how to identify commonalities between classes and their behaviors, you can design highly robust software solutions