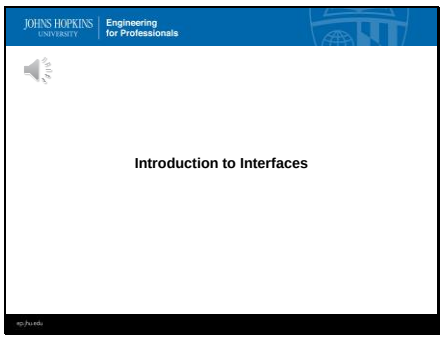
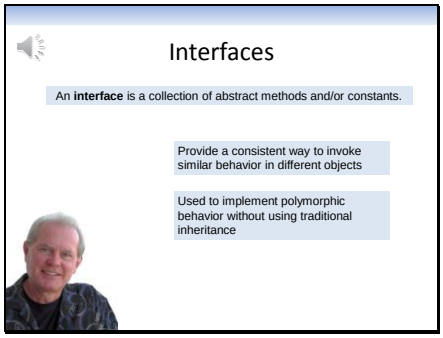
1



In this lecture you will learn about interfaces.

2



The kind of interfaces we will discuss in this unit have nothing to do with user interfaces like menus, windows, and buttons…but are very important and used extensively in Java programming.

In Java, an interface is a collection of abstract methods or a collection of constants.

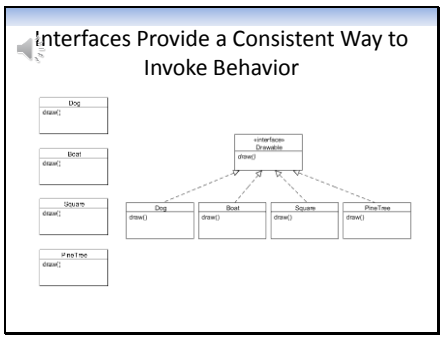
In Java, interfaces are used for two primary reasons.

To provide a consistent way to invoke behavior in different types of objects

And, to get polymorphic behavior without using inheritance.

Let’s look at some examples.

3



Suppose we had a bunch of classes like Dog, Boat, Square, and maybe PineTree.

And, suppose we wanted to be able to draw what objects from these classes looked like…on a video display.

It would really be helpful to design these classes so that the same method name is used to draw objects on the screen…so that programmers wouldn't have to look up or try to remember a bunch of different method names…like drawDog, drawASquare…and so forth.

It would be much easier, and more consistent, if the same method name, say draw(), was used in all these classes.

Now, we couldn't really use inheritance to do this…because our IS-A test would fail…and these classes aren't related in any natural way.

But, we could use an interface to accomplish this. This is one of the purposes of an interface…to provide a consistent way to call methods across different classes.

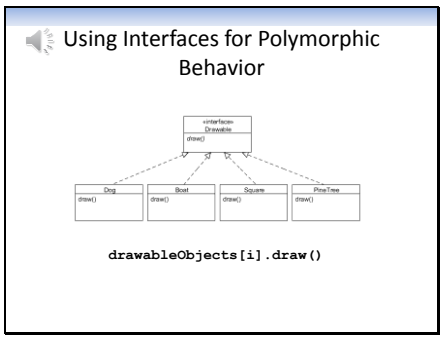
All we have to do is to define an interface, let's call it Drawable, that contains a draw() method, and then have these classes use that interface.

You'll learn how to write Java code to do this a little bit later in this unit. For now, we'll just illustrate how classes can use an interface on a diagram…like this .

This diagram indicates that Drawable is an interface, and that Dog, Boat, Square, and PineTree use this interface.

Draw() is an abstract method, so each class must provide a code definition for it.

4



Another use for interfaces is to get polymorphic behavior across objects that are otherwise unrelated…meaning they're not part of the same inheritance structure.

That's the case with the Dog, Boat, Square, and PineTree classes…they're obviously not part of a common inheritance structure.

However, we can use the fact that they all implement the same interface to get polymorphic behavior…because interfaces work a little bit like classes do…if a bunch of classes implement an interface, then their behavior can be invoked polymorphically.

In this example, if we had an array of Drawable objects called, say, drawableObjects, we could invoke the draw() method polymorphically like this .