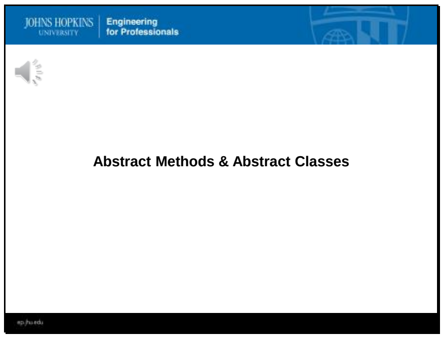
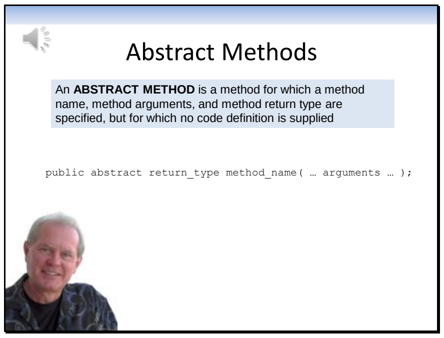
1 
JOHNS HOPKINS UNIVERSITY | Engineering for Professionals

Abstract Methods & Abstract Classes

ep.jhu.edu

In this lecture you will learn how to use abstract methods and abstract classes.

2 
Abstract Methods

An **ABSTRACT METHOD** is a method for which a method name, method arguments, and method return type are specified, but for which no code definition is supplied

```
public abstract return_type method_name( … arguments … );
```

Abstract methods are used extensively in Java.

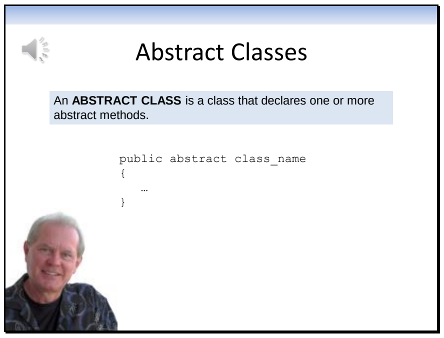
An abstract method is a method that is declared with a method name, method arguments (if any), and method return type…but for which no code definition is supplied.

Abstract methods are used in Java to provide a consistent interface to using one or more classes. Abstract methods are typically declared in a base class and make subclasses responsible for providing the method definition.

Abstract methods are declared by using the ‘abstract’ keyword, as indicated here.

It is important to note that abstract methods cannot have private visibility, because subclasses would then not be able to provide method definitions.

3



Abstract Classes

An **ABSTRACT CLASS** is a class that declares one or more abstract methods.

```
public abstract class_name
{
   …
}
```

An abstract class is defined as a class that contains one or more abstract methods.

Abstract classes are very commonly used in Java, and they serve as building blocks for creating inheritance hierarchies.

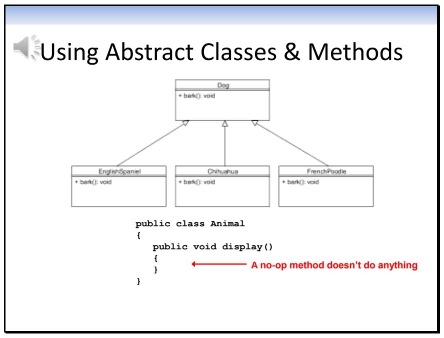
Abstract classes are defined by preceding the class name with the Java keyword ‘abstract’.

It’s very important to note that abstract classes can’t be instantiated…if you try to instantiate an abstract class the Java compiler will issue an error.

While you can’t instantiate an abstract class…you can declare a variable to be associated with an abstract class type.

Let’s take a look at using abstract classes and methods.

4



Using Abstract Classes & Methods

```
public class Animal
{
   public void display()
   {
   }
}
```

A no-op method doesn’t do anything

To illustrate how abstract classes and methods are used, let’s take an application in which we’ll deal with dogs.

Let’s start by re-visiting an example that was used in an earlier lecture, and see how we might do things differently now that we know about abstract classes and methods.

We had a Dog base class and several subclasses. The base class defined a bark() method that was used to implement polymorphic behavior.

The bark() method was overridden by the subclasses, because each type of dog made a different sounding bark.

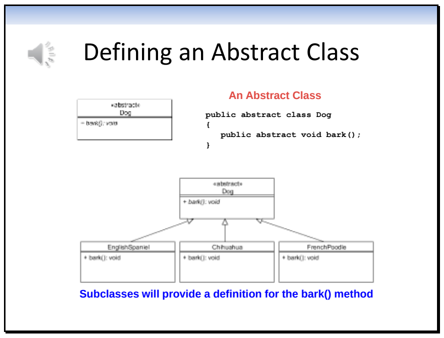
When we wrote the code for this example, we implemented the base class bark() method using something called a no-op method. Recall that a no-op method is a method that doesn’t do anything.

We used a no-op method for two reasons: First, the Dog class was too general to provide a specific barking behavior, and second, because we didn’t yet know about abstract methods.

In practice, this is exactly a situation that calls for using an abstract method.

Let’s see what that will look like.

5



Defining an Abstract Class

An Abstract Class

```
public abstract class Dog
{
   public abstract void bark();
}
```

Subclasses will provide a definition for the bark() method

We’re going to specify bark() as an abstract method. Because it is an abstract method, the Dog base class is an abstract class.

Using design notation, we can represent the Dog base class like this. The ‘abstract’ keyword is added to let us know that Dog is an abstract class, and the bark method is written in italics to distinguish it from other possibly non-abstract methods.

Of course, in this example, there are no other methods illustrated, but this is the notation that will be used to distinguish abstract methods from non-abstract methods.

By the way, non-abstract methods are commonly referred to as concrete methods by object-oriented programmers.

Here’s the class definition for our Dog class .

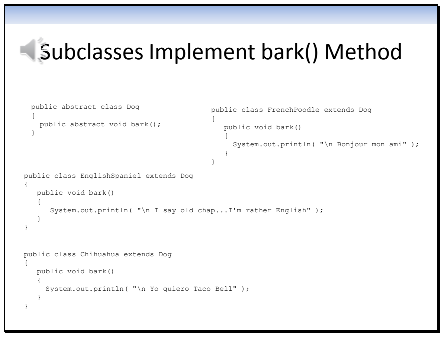
Notice that the abstract keyword is used twice…once with the method name and once with the class name.

Now, our design looks like this .

This design tells us that the Dog subclasses will provide a definition of the base class bark() method. In earlier lectures we would interpret this diagram as meaning the Dog subclasses will override the inherited bark()

method. But, since the bark() method is an abstract method in this case, it tells us that the subclasses will provide a definition of this method.

6



Subclasses Implement bark() Method

```
public abstract class Dog
{
  public abstract void bark();
}
```

```
public class FrenchPoodle extends Dog
{
   public void bark()
   {
     System.out.println( "\n Bonjour mon ami" );
   }
}
```

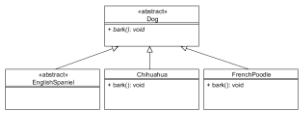
```
public class EnglishSpaniel extends Dog
{
   public void bark()
   {
      System.out.println( "\n I say old chap...I'm rather English" );
   }
}
```

```
public class Chihuahua extends Dog
{
   public void bark()
   {
     System.out.println( "\n Yo quiero Taco Bell" );
   }
}
```

Here's the code for our example. With the exception of the abstract class definition it is exactly the same as in an earlier version.

7

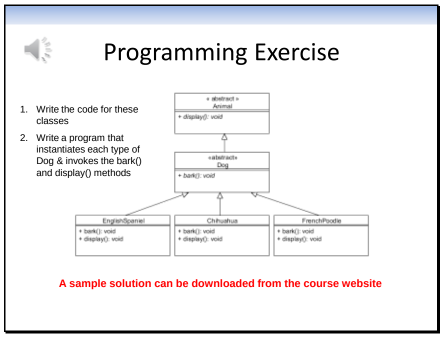
Subclasses Must Provide Abstract Method Implementation



Something that's very important to remember is that if a subclass inherits an abstract method and does not provide a definition for that method…then that subclass is also an abstract class.

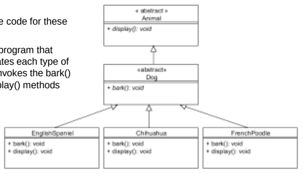
This diagram tells us that the EnglishSpaniel subclass does not provide a definition of the inherited bark() method…so it is an abstract class…and…it can't be instantiated.

8



Programming Exercise

1. Write the code for these classes
2. Write a program that instantiates each type of Dog & invokes the bark() and display() methods



A sample solution can be downloaded from the course website

Now, I'd like you to do a programming exercise.

I'd like you to implement the code for this inheritance structure. Then, I'd like you to write a program called AbstractClassExercise that instantiates each type of Dog and invokes its methods.

The display() methods should indicate which type of dog is being displayed…for example…"displaying Chihuahua"…and the bark() methods should indicate which type of dog is barking. You may use the bark() methods that were illustrated in this lecture or provide your own.

When you have your program working, please download the sample solution from the course downloads folder and compare it to your solution.