Abstract Class

What Does Abstract Class Mean?

An abstract class, in the context of Java, is a superclass that cannot be instantiated and is used to state or define general characteristics. An object cannot be formed from a Java abstract class; trying to instantiate an abstract class only produces a compiler error. The abstract class is declared using the keyword abstract.

Subclasses extended from an abstract class have all the abstract class's attributes, in addition to attributes specific to each subclass. The abstract class states the class characteristics and methods for implementation, thus defining a whole interface.

Explanation about Abstract Class

Abstract classes serve as templates for their subclasses. For example, the abstract class Tree and subclass, Banyan_Tree, has all the characteristics of a tree as well as characteristics that are specific to the banyan tree.

Understanding the differences between an abstract class and an interface is essential. An interface only has method declarations or abstract methods and constant data members, while an abstract class may have abstract methods, member variables and concrete methods. Because Java only supports single inheritance, a class can implement several interfaces but can extend only one abstract class.

Java: Abstract classes and abstract methods

Abstract classes:

- abstract classes can’t be instantiated, only subclassed.
- other classes *extend* abstract classes.
- can have both abstract and concrete methods.

- similar to interfaces, but (1) can implement methods, (2) fields can have various access modifiers, and (3) subclasses can only extend one abstract class.

Abstract methods:

- abstract method bodies must be empty (no curly braces)
- subclasses must implement the abstract class’s abstract methods

Eclipse example: abstract_classes_methods

Detailed description

If you add the word abstract before a class, it means that other classes must extend it in order to use it. Here’s an example:

```
public abstract class Animal

{

public void eat(Food food)

{

// do something with food....

}



public void sleep(int hours)

{

try

{

// 1000 milliseconds * 60 seconds * 60 minutes * hours

Thread.sleep ( 1000 * 60 * 60 * hours);
}

catch (InterruptedException ie) { /* ignore */ }

}



public abstract void makeNoise();

}
```

Now if another class wants to use this class, it has to extend it:

```
public Dog extends Animal

{

public void makeNoise() { System.out.println ("Bark! Bark!"); }

}



public Cow extends Animal

{

public void makeNoise() { System.out.println ("Moo! Moo!"); }

}
```

The reason you don't use an interface is because interfaces require you to implement all of the methods. When you use a class to extend another class, you aren't required to implement all of the methods.

Abstract classes

Abstract (which Java supports with abstract keyword) means that the class or method or field or whatever cannot be instantiated (that is, created) where it is defined. Some other object must instantiate the item in question.

If you make a class abstract, you can’t instantiate an object from it. I think you only use abstract classes if you want to provide a utility that people can use to perform some action, but not which they can modify or somehow instantiate and use in a different way.

Sometimes, a Java developer writes a class that should never be instantiated. That is, it’s a class that will never have an object associated with it. Such a class usually serves as the basis for other classes. Our Mammal class is an example of such a class. All the mammals in the real world are specific types of mammals rather than just mammals. Cats, dogs, mice, humans, whales, and so on are all mammals, of course, but they all have more specific names and more specific traits. So, there’s no actual animal that’s a generic mammal. Consequently, we don’t want anyone to create an instance of our Mammal class, because that would be bad modeling.

Java lets developers declare that a class should never have an instance by using the abstract keyword. In Java, abstract means that the class can still be extended by other classes but that it can never be instantiated (turned into an object). Returning to our example, we can have Mammal be abstract (because there’s no such thing as a generic mammal) and still have Cat, Dog, and Mouse extend Mammal (because cats, dogs, and mice are mammals).

The hard part is figuring out when a class should be abstract. Modeling the animal kingdom is a simple example, so it’s not hard to see that Mammal should be an abstract class.

Abstract classes can include abstract methods. Any class that extends a class with an abstract method must implement that method. For example, our Mammal class includes an abstract speak() method. Any class that extends Mammal must implement the speak method, and that implementation must have the same signature. So, in this case, the implementations must return void and accept no arguments.

Abstract classes can also include regular methods that their descendant classes can use without needing to implement them. Listing 6-1 shows both kinds of methods within the Mammal class.

Listing 6-1. Methods within an abstract class

```
package com.apress.java7forabsolutebeginners.examples.animalKingdom;
```

```
abstract class Mammal { // And here’s a method for making the sound. // Each child class must implement it. abstract void speak();
```

```
// All descendant classes can call this and do // not need to implement their own versions of it protected void sayWhatIAm() { System.out.println(“I am a mammal”); } }
```

If the subclass has a method with the same name as the parent’s method that the subclass extends, the subclass’ method overwrites the parent. If you want to use the parent class’s method instead, you use the `super` keyword, like this:

```
super.startRobot();
```

You can’t have an abstract method in a concrete class. If you want an abstract method, the class must also be abstract. Otherwise users could instantiate the class and try to implement its methods.

However, if your class is abstract, it may have some methods that are abstract and others that are concrete.