21. Inheritance :: Abstract Classes

One troubling thing remains about the Shape class, but before discussing it, let's go back to the mammal analogy.

Suppose we have pictures of various mammals: a dog, a cat, a pig, a human, and an aardvark. We show these pictures to various people and ask them to tell me what they see. What type of answers do you think we will get? Most likely we will hear:

"That's a dog."
"That's a cat."
" That's a pig."
"That's Homer Simpson."
"That's an aardvark."

Notice that not one person said, "That's a mammal." Why not? What exactly *is* a mammal? What does a mammal look like?

The answer I am trying to lead you to is that there are no mammals in the physical world. Dogs exist. Cats exist. Pigs and aardvark's exist. Homer doesn't really exist, but pretend that he does. **Mammals do not exist**.

The word "mammal" is an **abstraction**. It is a term that is used to categorize dogs, cats, pigs, and so on as **being** mammals. Why? Because those critters all have certain attributes and behaviors in common and that is the definition of what makes a dog a mammal and a cactus not a mammal.

21. Inheritance :: Abstract Classes (continued)

What does this have to do with *Shape*? In essence, *Shape* is an abstraction. Show various pictures of circles, squares, rectangles, ovals, lines, and so on, and ask them what they see. It is not likely that a person is going to say, "I see a shape." They will tell you they see lines, circles, squares, and so on.

How do you draw a shape? Ask a group of people to draw a shape on a piece of paper and you are likely to get papers containing circles, squares, triangles, and the occasional smiley face. It's a bit of a stretch, I know, but shapes don't really exist in the physical world.

Therefore, does it really make sense to instantiate (create) a *Shape* object?

```
Shape someShape = new Shape(10, 20);
```

How would I draw *someShape* on the graphical window? Do I draw a line? A circle? A smiley face?

What I want you to understand is that "shape" is an abstraction and it makes no more sense to instantiate a *Shape* object than it would to instantiate a *Mammal* object.

21. Inheritance :: Abstract Classes (continued)

Consequently, in OO design and programming, there is a way of specifying that a class represents an abstraction and that it make no sense to instantiate objects of that class. Such classes in Java are called **abstract classes** and the defining characteristics are:

1. The class must be declared as an abstract class using the **abstract** reserved word.
2. Abstract classes may not be instantiated (a nonabstract class—one in which objects can be instantiated—is referred to as a **concrete class**).
3. An abstract class is essentially a container for instance variables and instance methods that will be inherited by all subclasses of the abstract class. These instance variables represent data **that all objects** in the class hierarchy (the abstract class and all of its subclasses) have in common.
4. An abstract class may declare instance methods that are not implemented (there is no body). Such methods are known as **abstract methods** and the requirement is that subclasses must implement the abstract methods (if a subclass does not implement all of the inherited abstract methods, that subclass also becomes an abstract class).