We give a simple explanation of abstract classes and abstract methods.

**Making a class abstract**

Consider class Shape and one of its subclasses, Circle, outlined to the right. There would be other subclasses, like Rectangle, and Triangle.

Class Shape is there to hold information that is common to all subclasses, like the position of a shape in a GUI window. We don’t want users to create instances of class Shape because an instance really isn’t a shape; all it holds is the position of shapes.

In order to prevent users from creating instances of class Shape, make Shape abstract by putting keyword `abstract` between `public` and `class`:

```java
public abstract class Shape { … }
```

Doing that makes the expression `new Shape(…)` illegal; if you use it, your program won’t compile.

You can still have Shape variables. Example: you can do this:

```java
Shape sp = new Circle(5, 10, 2.5);
```

But you can’t do this:

```java
Shape sp = new Shape(5, 10);
```

**Reason for making a class abstract:** So you cannot create instances of it; it cannot be “newed”.

**Making a method abstract**

You know the rule in Java that for a variable sp with a Shape perspective, meaning it was declared as a Shape variable, a method call like `sp.area()` is legal only if it is declared in class Shape or one of its superclasses. Your program won’t compile if it has a call

```
sp.area(5, 6, 2.5)
```

because method `area()` is not defined in Shape or in Object. Java has this rule because it wants to be sure that the method exists at runtime. It wouldn’t exist at runtime if some subclass of Shape didn’t declare `area()`, and there is no way to guarantee that.

So that we don’t have to cast down to a subclass to call method `area()`, we put the method in class Shape. This method should not be called, since there is no known area in Shape. So we have it throw an exception.

But we still have a problem. Some subclass may not implement method `area()`. To force all subclasses to implement the method, we make it the method abstract, by placing keyword `abstract` after `public` and replacing the method body by a semicolon “;”.

**Reason for making a method in an abstract class abstract:** So subclasses must implement it.

Note: Some subclass C of Shape could also be abstract. If C is abstract, it doesn’t have to implement method `area()` — but subclasses of C would have to implement it.