**Modifiers**

When a method is described as being abstract, it doesn't have an implementation.  It relies on a subtype to implement the method.  Therefore, certain modifiers, such as private, static, and final cannot be used on an abstract method.  If you think about this logically, it makes sense.  Private, static, and/or final methods cannot be overridden, and all abstract methods must be overridden.  Therefore these modifiers cannot appear in an abstract method declaration.  By the way, that's the trick to remembering what's legal and not legal when defining a concrete implementation of an abstract method: they follow the same rules declared for overriding methods.

**Inheritance**

When we say that a concrete subtype must make sure all abstract methods are overridden, it doesn't necessarily mean that all of the abstract methods are implemented in the subtype.  Basically, the rule is that for a class to be concrete, it must make sure that any method call has some concrete definition that can be invoked.  And that concrete definition could be defined in a supertype (even in an abstract class!)

For example, imagine there are three classes: Person (abstract), Employee (abstract), and Consultant (concrete).  For Consultant to compile, all of the abstract methods in Person and Employee must have a concrete implementation.  As such the following compiles:

public abstract class Person {

protected String firstName;

public abstract String getName();

}

public abstract class Employee extends Person {

protected String title;

public abstract double pay();

public String getName() {

return title + ": " + firstName;

}

}

public class Consultant extends Employee {

public double pay() {

return 80\_000.0;

}

}

Notice that Person defines an abstract method (getName()), and it is not implemented in the Consultant class. This is perfectly OK because Employee defines getName(). When we instantiate a Consultant and call getName(), the concrete implementation defined in Employee is available... and that satisfies the main rule: a class may become concrete (non-abstract), as long as all abstract methods in its hierarchy have a corresponding concrete implementation.