**[ Abstraction ]**

General Rules

A class which contains the **abstract** keyword in its declaration is known as abstract class.

* Abstract classes may or may not contain abstract methods, i.e., non-abstact method ( **public void get() {}** )
* If a class has at least one abstract method, then the class must be declared abstract.
* If a class is declared abstract, it cannot be instantiated.
* To use an abstract class, you have to inherit it from another class, and provide implementations to the abstract methods in it.
* If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

Abstract Classes and methods

An *abstract class* is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

An *abstract method* is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

**abstract void moveTo(double deltaX, double deltaY);**

If a class includes abstract methods, then the class itself *must* be declared abstract, as in:

**public abstract class GraphicObject {**

**// declare fields**

**// declare nonabstract methods**

**// declare abstract methods**

**abstract void draw();**

**}**

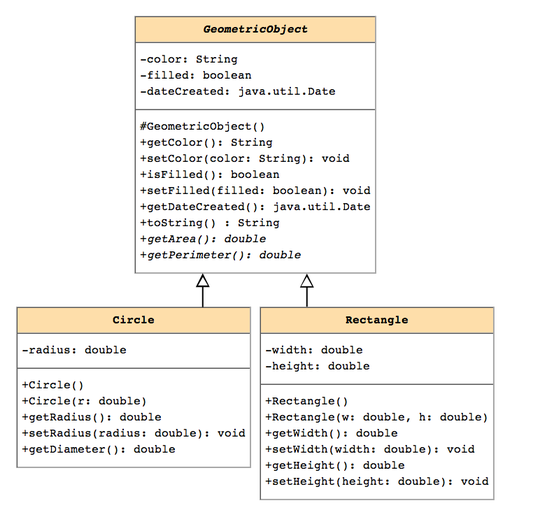
When an abstract class is subclassed, the subclass usually provides implementations for *all* of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

Concrete classes

As mentioned an abstract class is meant to be used as the base class from which other classes are derived. The derived class is expected to provide implementations for the methods that are not implemented in the base class. A derived class that implements all the missing functionality is called a *concrete class.*

UML case follows. Notice italicized class name and methods to emphasize either an abstract class or an abstract method.

UML abstract example



Notice, in the example describing the GeometricObject class, we can't actually implement a getArea() or getPerimeter() method for a GeometricObject in ANY meaningful way, as those methods depend upon what type of shape is involved (circle, rectangle, etc...).

Instead, the Circle and Rectangle subclasses need to override these methods with a meaningful implementation.

So that we don't have to write an implementation for these methods in the GeometricObject class, we declare them as *abstract* methods in this class. Think of abstract methods as "placeholders" for methods that will eventually be defined in some subclass of the current class.

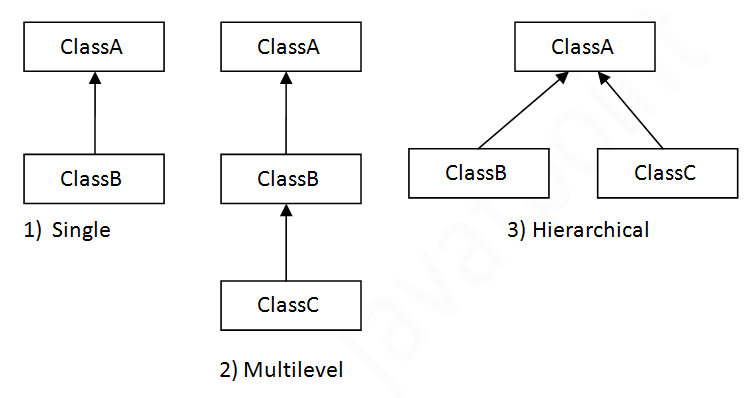
**[ Inheritance ]**

Classes that can be derived from other classes, thereby inheriting fields and methods from those classes. ... The class from which the *subclass* is derived is called a superclass (also a base class or a parent class).

**What methods can or cannot be inherited?**

**Which access modes are useful?**

-Possible hierarchy’s-



**[ Polymorphism ]**

Defined as ‘a state of having *many* shapes’ or ‘the capacity to take on different forms’. When applied to object oriented programming languages like Java, it describes a language’s ability to process objects of various types and classes through a single, uniform interface.

