ABSTRACT METHODS AND CLASSES

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.

Syntax: **abstract class class_name { }**

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

Syntax: **abstract return_type function_name ();** **// No definition**

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 non 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.

Note: Methods in an *interface* that are not declared as default or static are *implicitly* abstract, so the abstract modifier is not used with interface methods. (It can be used, but it is unnecessary.)

Example of Abstract class

```
abstract class A{
 abstract void callme();
}
class B extends A{
 void callme(){
  System.out.println("this is callme.");
 }
 public static void main(String[] args){
  B b = new B();
  b.callme();
 }
}
```

Abstract class with concrete (normal) method: Abstract classes can also have normal methods with definitions, along with abstract methods.

```
abstract class A {
 abstract void callme();
 public void normal() {
  System.out.println("this is concrete method");
 }
}

class B extends A{
 void callme() {
  System.out.println("this is callme.");
 }
public void normal() {
  System.out.println("this is concrete method");
 }

 public static void main(String[] args) {
  B b = new B();
  b.callme();
  b.normal();
 }
}
```

Points to Remember

1. Abstract classes are not Interfaces. They are different.
2. An abstract class may or may not have an abstract method. But if any class has even a single abstract method, then it must be declared abstract.
3. Abstract classes can have Constructors, Member variables and Normal methods.
4. Abstract classes are never instantiated.
5. When you extend Abstract class with abstract method, you must define the abstract method in the child class, or make the child class abstract.

When to use Abstract Methods & Abstract Class?

Abstract methods are usually declared where two or more subclasses are expected to do a similar thing in different ways through different implementations. These subclasses extend the same Abstract class and provide different implementations for the abstract methods.

Abstract classes are used to define generic types of behaviors at the top of an object-oriented programming class hierarchy, and use its subclasses to provide implementation details of the abstract class.

Abstract Classes Compared to Interfaces

Abstract classes are similar to interfaces. You cannot instantiate them, and they may contain a mix of methods declared with or without an implementation. However, with abstract classes, you can declare fields that are not static and final, and define public, protected, and private concrete methods. With interfaces, all fields are automatically public, static, and final, and all methods that you declare or define (as default methods) are public. In addition, you can extend only one class, whether or not it is abstract, whereas you can implement any number of interfaces.

Which should you use, abstract classes or interfaces?

- Consider using abstract classes if any of these statements apply to your situation:
    - You want to share code among several closely related classes.
    - You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).
    - You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.
- Consider using interfaces if any of these statements apply to your situation:
    - You expect that unrelated classes would implement your interface. For example, the interfaces Comparable and Cloneable are implemented by many unrelated classes.
    - You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.
    - You want to take advantage of multiple inheritance of type.

Reference: http://docs.oracle.com/javase/tutorial/java/IandI/abstract.html