Interfaces

An interface is a class-like generic construct that contains only constants and abstract methods. In many ways, it is similar to an abstract class -- but it instead aims to specify the behavior of a class that implements it. Of course, a class can have many different behaviors associated with it, so a class might implement several interfaces. For example, we might want to ensure that objects of a particular class are comparable, edible, cloneable, etc... and could do this by having the class implement interfaces Comparable, Edible, Cloneable, etc...

For a class to implement an interface it must implement all of the abstract methods specified in that interface.

Like an abstract class, you cannot create an instance from an interface using the new operator, however you can:

* create an instance from a class that implements an interface
* use an interface as a data type for a variable, as the result of casting, and so on

Interface example:

//define interface

public interface Client {

void print();

static void print() {

System.out.println("From Static ulility");

}

default void log(String str){

print();

System.out.println("Client logging::"+str);

}

}

public class Chicago **implements** Client{

@Override

public void print() {

// TODO Auto-generated method stub

System.out.println("high from print");

}

public static void main(String[] args) {

Chicago l = new Chicago();

Client c = new Chicago();

l.log("Chicago Data");

c.print();

}

}

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.
  + 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.