**Interface vs. Abstract Class**

public interface LoginAuth{

public String encryptPassword(String pass);

public void checkDBforUser();

}

Now suppose you have 3 databases in your application. Then each and every implementation for that database needs to define the above 2 methods:

public class DBMySQL implements LoginAuth{

// Needs to implement both methods

}

public class DBHadoop implements LoginAuth{

// Needs to implement both methods

}

public class DBMongo implements LoginAuth{

// Needs to implement both methods

}

But what if encryptPassword() is not database dependent, and it's the same for each class? Then the above would not be a good approach.

Instead, consider this approach:

public abstract class LoginAuth{

public String encryptPassword(String pass){

// Implement the same default behavior here

// that is shared by all subclasses.

}

// Each subclass needs to provide their own implementation of this only:

public abstract void checkDBforUser();

}

Now in each child class, we only need to implement one method - the method that is database dependent.

**More precise**

Methods of a Java interface are implicitly abstract and cannot have implementations. A Java abstract class can have instance methods that implements a default behavior.

Variables declared in a Java interface are by default final. An abstract class may contain non-final variables.

A Java interface should be implemented using keyword “implements”; A Java abstract class should be extended using keyword “extends”.

An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces.

A Java class can implement multiple interfaces but it can extend only one abstract class.