**b. abstract class and interface, identify usage of each**

**Interface**

The interface is a blueprint that can be used to implement a class. The interface does not contain any concrete methods (methods that have code). All the methods of an interface are abstract methods.

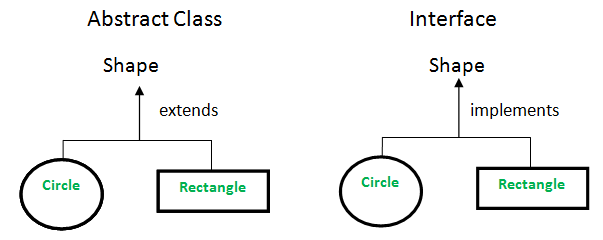
An interface cannot be instantiated. However, classes that implement interfaces can be instantiated. Interfaces never contain instance variables but, they can contain public static final variables.

**Abstract Class**

A class which has the abstract keyword in its declaration is called abstract class. Abstract classes should have at least one abstract method. , i.e., methods without a body. It can have multiple concrete methods.

Abstract classes allow you to create blueprints for concrete classes. But the inheriting class should implement the abstract method.

Abstract classes cannot be instantiated.



**Example of Abstract class**

Abstract class Bike {

Abstract void run();

}

Class Honda4 extends Bike {

Void run() {System.out.println(“running safely”); {

Public static void main(String args[]) {

Bike obj = new Honda4();

Obj.run();

}  
}

**Example of interface**

Interface printable {

Void print();

}

Class A6 implements printable {

Public void print() {System.out.println(“Hello”); }

Public static void main(String args[]) {

A6 obj = new A6();

}

}

**Important Reasons For Using Interfaces**

* Interfaces are used to achieve abstraction.
* Designed to support dynamic method resolution at run time
* It helps you to achieve loose coupling.
* Allows you to separate the definition of a method from the inheritance hierarchy

**Important Reasons For Using Abstract Class**

* Abstract classes offer default functionality for the subclasses.
* Provides a template for future specific classes
* Helps you to define a common interface for its subclasses
* Abstract class allows code reusability.