INTERFACE VS ABSTRACT CLASS

We will first learn what an abstract class and interface is. We will also do a detailed comparison of interfaces and abstract classes, followed by a listing of points in favor of each of them. We will also discuss on the work around for turning cons to pros in few cases.

An **abstract class** is a class marked as abstract and they cannot be instantiated.

```
class abstract MyAbstractClass{

  abstract void myAbstractMethod();

  void myConcreteMethod(){

    //method body

  }

}
```

Important properties of abstract classes are:

- An abstract method is a method marked abstract keyword with only signature and no body, whereas a concrete method is a method with body.
- Abstract classes have all the properties of a class including constructors.
- Constructor of an abstract class is invoked by a subclass from its constructor using super keyword (e.g. super()). Abstract classes can have state and can be used to provide a skeletal implementation.
- Abstract classes can have abstract and/or concrete methods in any combination.
- Even if an abstract class contains only concrete methods and no abstract methods, it cannot be instantiated.
- A concrete class should extend an abstract class (using keyword extends) and implement all abstract methods.
- Abstract classes can use this pointer.
- Abstract classes can have private or any access level methods.
- Abstract classes can also have synchronized methods.

An **interface** is a contract for a class and contain only abstract methods.

```
interface MyInterface{
```

```
abstract void myExplicitAbstractMethod();

void myImplicitAbstractMethod();

}
```

Important properties of an interface are:

- The abstract keyword is implicit for interface methods, but can also be declared explicitly.
- Interfaces don't have state, behavior or constructors. They are just contracts. But they can have static constant variables (public, static and final).
- Variables of an interface are implicitly public, static and final; though they can be explicitly defined with these modifiers.
- A concrete class should implement an interface (using keyword implements) and implement all abstract methods.
- An interface can extend another interface using extends keyword.

Comparison of basic features of interfaces and abstract classes

1. **Both abstract class and interfaces** are abstractions in Java that **cannot be instantiated**. **The main difference is that an abstract class is a class and an interface is a contract** or specification without any implementation. Also, **an abstract class can be invoked if it has a main method** and can be used for testing its static methods.
2. **Except that an abstract class cannot be instantiated, all other functionality of a regular class still exists**, and its fields, methods, and constructors are all accessed in the same way. There is a common misconception among some that abstract classes can't have constructors because they cannot be instantiated.**Abstract classes have constructors** and they are invoked by the subclasses using super from the subclass constructors.
3. **An abstract classes can have state** just like any other class. However **an interface can't have behavior or state.** "No behavior" is enforced by not allowing methods to have implementation, whereas abstract classes can have concrete methods and "no state" is enforced by making the fields public, static and final by default, whereas abstract classes can have non-final variables and we instantiate variables in an abstract class through constructors. **All methods of an interface must be abstract, whereas abstract classes can have abstract methods and/or concrete methods.** An abstract class without any abstract methods is still a valid abstract class.
4. **You implement an interface** using the implements keyword whereas you **extend a class** using the extends keyword.

Points that favor interfaces

Points that favor interfaces are also the points that are against abstract or concrete classes.

1. **Java permits multiple-inheritance only through interfaces**, but not through classes. A class can implement multiple interfaces, but can extend only one class. An exception to this rule is local inner classes and anonymous inner class, which can both implement or extend maximum one interface or class and not both together.
2. **Existing classes can be easily modified to implement a new interface** irrespective of whether it already implements another interface. But you can extend a new class only if it is not already extending another class.
3. Sometimes a class might belong to more types that does not belong to a rigid type hierarchy. **Interfaces allow the construction of** these **non hierarchical type frameworks.** We can implement all those types or create a type which is the combination of all those interfaces. **Without interfaces, we would have to use a**separate class for every combination known as a **bloated class hierarchy.**
4. **The limitation of an interface that it cannot provide any skeletal implementations can be overcome by providing implementation assistance through an abstract skeletal implementation class**corresponding to an interface. **Examples** for this practice **in Java are AbstractCollection, AbstractList, AbstractSet, AbstractMap etc.** Here even if we have interfaces and abstract class, **while defining a type, we should use an interface** (like myMethod(Map m)) and we can either pass in an existing implementation or one we created by extending an abstract implementation. By using an interface as the type, **you can change the actual implementation later without any change to the code** (here myMethod()).

Points in favor of abstract classes

Points in favor of abstract (or concrete classes) are also the points that are against interfaces.

1. **It is practically not possible to add methods to an interface after it is released without changing existing classes;** you have to make all its implementations implement that new method, or existing implementation classes will fail. **However you can add a concrete method to an abstract class without breaking the existing classes.** This is one case where defining a type in terms of abstract class has advantage.
2. **It is generally a good practice to minimize the accessibility of classes and members** and is an important consideration during information hiding or encapsulation. However **if a class implements an interface, all of the class's methods from interface must be declared public** as all members of an interface are implicitly public. During inheritance, subclass method cannot have a lower level access than superclass method making sure that an instance of a subclass is usable anywhere an instance of super class is usable.
3. **Abstract classes can be used to provide skeletal implementation, whereas interfaces cannot. A class that implements an interface must provide an implementation of all the methods of that interface.** Abstract classes can be inherited without implementing the abstract methods if the derived

class itself is an abstract. However this can be overcome by providing implementation assistance through an abstract skeletal implementation class.

4. **Interfaces are considered to be a bit slower than abstract classes** because it requires an extra indirection during virtual table lookup. A simple virtual table could be used for directly inherited methods, but finding the code that implements an interface method would involve following a pointer to a table of interfaces implemented by a class, then searching through the table to find a virtual table for the implementation of that interface. However modern JVMs are a lot smarter, and most of the lookup is done during dynamic compilation and hence the runtime differences are small or nonexistent. You can refer to the wiki linkhttp://en.wikipedia.org/wiki/Virtual_method_table for details on virtual table.

5. An **interface can extend only interfaces**; an abstract class can extend another class and implement interfaces.

Source : http://javajee.com/interface-vs-abstract-class