**[Difference between Abstract class vs Interface in Java](http://javarevisited.blogspot.in/2013/05/difference-between-abstract-class-vs-interface-java-when-prefer-over-design-oops.html" \o "Difference between Abstract class vs Interface in Java)**

When to use interface and abstract class is one of the most popular object oriented design questions and almost always asked in Java, C#, and C++ interviews. In this article, we will mostly talk in the context of Java programming language, but it equally applies to other languages as well. The question usually starts with a difference between abstract class and interface in Java, which is rather easy to answer, especially if you are familiar with the syntax of [Java interface and abstract class](http://javarevisited.blogspot.com/2013/04/10-abstract-class-and-interface-interview-question-java-answers.html). Things start getting difficult when interviewer ask about when to use abstract class and interface in Java, which is mostly based upon a solid understanding of popular OOPS concepts like Polymorphism, Encapsulation, Abstraction, Inheritance, and Composition. Many programmers fumbles here, which is natural because most of them haven't gone through real system design process and haven’t seen the impact of choosing one over other.   
  
The repercussion of design decisions are best known during the maintenance phase, a good design allows seamless evolution while maintaining a fragile design is a nightmare.  
  
 As I have said previously, sometimes [object oriented design interview questions](http://javarevisited.blogspot.com/2012/06/20-design-pattern-and-software-design.html) also helps to understand a topic better, but only if you are willing to do some research and not just mugging the answer. Questions like when to use abstract class and interface fall under the same category.   
  
In order to best understand this topic, you need to work out some scenarios, examples etc. It's best to get this kind of knowledge as part of your work but even if you don't get there, you can supplement them by reading some good books like [Head First design pattern](http://www.amazon.com/dp/0596007124/?tag=javamysqlanta-20) and doing some object-oriented software design exercises.   
  
In this article, we will learn the *difference between abstract class and interface in Java programming* language and based on our understanding of those differences, we will try to find out some tips and guidelines to decide when it's better to use abstract class over the interface or vice-versa.

Difference between abstract class and interface in JavaWhile deciding *when to use interface and abstract class*, it’s important to know difference between abstract class and interface in Java. In my opinion, following two differences between them drives decision about when to use abstract class or interface in Java.

1) Interface in Java can only contains declaration. You can not declare any concrete methods inside interface. On the other hand abstract class may contain both abstract and concrete methods, which makes abstract class an ideal place to provide common or default functionality. I suggest reading my post [10 things to know about interface in Java](http://javarevisited.blogspot.com/2012/04/10-points-on-interface-in-java-with.html) to know more about interfaces, particularly in Java programming language.

2) Java interface can extend multiple interface also Java class can implement multiple interfaces, Which means interface can provide more Polymorphism support than abstract class . By extending abstract class, a class can only participate in one Type hierarchy but by using interface it can be part of multiple type hierarchies. E.g. a class can be Runnable and Displayable at same time. One example I can remember of this is writing GUI application in J2ME, where  class extends Canvas and implements CommandListener to provide both graphic and event-handling functionality..

3) In order to implement interface in Java, until your class is abstract, you need to provide implementation of all methods, which is very painful. On the other hand abstract class may help you in this case by providing default implementation. Because of this reason, I prefer to have minimum methods in interface, starting from just one, I don't like idea of [marker interface](http://javarevisited.blogspot.com/2012/01/what-is-marker-interfaces-in-java-and.html), once annotation is introduced in Java 5. If you look JDK or any framework like Spring, which I does to understand OOPS and design patter better, you will find that most of interface contains only one or two methods e.g. Runnable, Callable,ActionListener etc

I haven't included all syntactical difference between abstract class and interface in Java here, because focus here to learn when to use abstract class and interface and choosing one over other. Nevertheless you can see [difference between interface and abstract class](http://java67.blogspot.com/2012/09/what-is-difference-between-interface-abstract-class-java.html) to find  all those syntactical differences.

**When to use interface and abstract class in Java**

As I said earlier, it's easy to answer questions like difference between abstract class and interface in Java, but difficult to answer follow-ups. Though most of  Java Interview starts with former one, later it goes to see if you have really used abstract class and interface or not. In order to answer this question, you need to have good understanding of OOPS concepts like [Polymorphism](http://javarevisited.blogspot.com/2011/08/what-is-polymorphism-in-java-example.html), [Encapsulation](http://javarevisited.blogspot.com/2012/03/what-is-encapsulation-in-java-and-oops.html), [Abstraction](http://javarevisited.blogspot.com/2010/10/abstraction-in-java.html) and [Inheritance](http://javarevisited.blogspot.com/2012/10/what-is-inheritance-in-java-and-oops-programming.html). Also familiarity with coupling and cohesion is important. You at least should know that effort of designing should lead to reduce coupling and increased cohesion, ease of maintenance etc. In this part, we will see some scenarios, guidelines, rules which can help you to decide *when to use abstract class and interface in Java*.

1) In Java particularly, decision between choosing Abstract class and interface may influence by the fact that multiple inheritance is not supported in Java. One class can only extend another class in Java. If you choose abstract class over interface than you lost your chance to extend another class, while at the same time you can implement multiple interfaces to show that you have multiple capability. One of the common example, in favor of interface over abstract class is [Thread vs Runnable](http://javarevisited.blogspot.com/2012/01/difference-thread-vs-runnable-interface.html) case. If you want to execute a task and need run() method it's better to implement Runnable interface than extending Thread class.

2) Let's see another case where an abstract class suits better than interface. Since abstract class can include concrete methods, it’s great for maintenance point of view, particularly when your base class is evolving and keep changing. If you need a functionality across all your implementation e.g. a common method, than, you need to change every single implementation to include that change if  you have chosen interface to describe your base class. Abstract class comes handy in this case because you can just define new functionality in abstract super class and every sub class will automatically gets it. In short, abstract class are great in terms of evolving functionality. If you are using interface, you need to exercise extra care while defining contracts because its not easy to change them once published.

3) Interface in Java is great for defining Types. Programming for interfaces than implementation is also one of the useful [Object oriented design principle](http://javarevisited.blogspot.de/2012/03/10-object-oriented-design-principles.html) which suggests benefit of using interface as argument to function, return type etc.

4) One more general rule of when to use abstract class and interface is to find out whether a certain class will form a IS-A hierarchy or CAN-DO-THIS hierarchy. If you know that you will be creating classes e.g. Circle, Square than it's better to create an abstract class Shape which can have area() and perimeter() as abstract method, rather than defining Shape as interface in Java. On the other hand if you are going to create classes which can do thinks like, can fly, you can use interface Flyable instead of abstract class.

5) Interface generally define capability e.g. Runnable can run(), Callable can call(), Displayable can display(). So if you need to define capability, consider using interface. Since a class can have multiple capabilities i.e. a class can be Runnable as well as Displayable at same time. As discussed in first point, Since [java does not allow multiple inheritance at class level](http://javarevisited.blogspot.com/2011/07/why-multiple-inheritances-are-not.html), only way to provide multiple capability is via interfaces.

6) Let's see another example of where to use Abstract class and Interface in Java, which is related to earlier point. Suppose you have lot of classes to model which are birds, which can fly, than creating a base abstract class as Bird would be appropriate  but if you have to model other things along with Birds, which can fly e.g. Airplanes, Balloons or Kites than it's better to create interface Flyable to represent flying functionality. In conclusion, if you need to provide a functionality which is used by same type of class than use Abstract class and if functionality can be used by completely unrelated classes than use interface.

7) Another interesting use of Abstract class and interface is defining contract using interface and providing skeletal using abstract class. java.util.List from Java collection framework is a good example of this pattern. List is declared as interface and extends Collection and Iterable interface and AbstractList is an abstract class which implements List. AbstractList provides skeletal implementation of List interface. Benefit of using this approach is that it minimize the effort to implement this interface by concrete class e.g. [ArrayList or LinkedList](http://java67.blogspot.com/2012/12/difference-between-arraylist-vs-LinkedList-java.html). If you don't use skeletal implementation e.g. abstract class and instead decide to implement List interface than not only you need to implement all List methods but also you might be duplicating common code. Abstract class in this case reduce effort to implement interface.

8) Interface also provide more decoupling than abstract class because interface doesn't contain any implementation detail, while abstract class may contain default implementation which may couple them with other class or resource.

9) Using interface also help while implementing [Dependency Injection design pattern](http://javarevisited.blogspot.com/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html) and makes testing easy. Many mock testing framework utilize this behavior.

That's all on **When to use Abstract class and interface in Java**. Though discussion here is centered around Java but given concept of abstract class and interface goes beyond Java and also applicable to other Object oriented language, some of the tips are also applicable to other OOPS languages. Key thing to remember here is there definition of abstract class and interface e.g. in C++ and C# it varies a lot like in C++ and Java. Single most difference is multiple inheritance. We have also discussed some key differences between abstract class and interface in Java, which influence decision of choosing abstract class over interface or vice-versa. Last thing to remember is that interface is extremely difficult to evolve, so put extra care while designing interfaces.