**Q1) What is an abstract class?**

Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation.

Note:

* If even a single method is abstract, the whole class must be declared abstract.
* Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.
* You can’t mark a class as both abstract and final.

**Q2) Can we instantiate an abstract class?**

An abstract class can **never** be instantiated. Its sole purpose is to be extended (subclassed).

**Q3) What are the differences between Interface and Abstract class?**

|  |  |
| --- | --- |
| **Abstract Class** | **Interfaces** |
| An abstract class can provide complete, default code and/or just the details that have to be overridden. | An interface cannot provide any code at all, just the signature. |
| In case of abstract class, a class may extend only one abstract class. | A Class may implement several interfaces. |
| An abstract class can have non-abstract methods. | All methods of an Interface are abstract. |
| An abstract class can have instance variables. | An Interface cannot have instance variables. |
| An abstract class can have any visibility: public, private, protected. | An Interface visibility must be public (or) none. |
| If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly. | If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method. |
| An abstract class can contain constructors. | An Interface cannot contain constructors. |
| Abstract classes are fast. | Interfaces are slow as it requires extra indirection to find corresponding method in the actual class. |

**Q4) When should I use abstract classes and when should I use interfaces?**

**Use Interfaces when…**

* You see that something in your design will change frequently.
* If various implementations only share method signatures then it is better to use Interfaces.
* you need some classes to use some methods which you don't want to be included in the class, then you go for the interface, which makes it easy to just implement and make use of the methods defined in the interface.

**Use Abstract Class when…**

* If various implementations are of the same kind and use common behavior or status then abstract class is better to use.
* When you want to provide a generalized form of abstraction and leave the implementation task with the inheriting subclass.
* Abstract classes are an excellent way to create planned inheritance hierarchies. They're also a good choice for nonleaf classes in class hierarchies.

**Q5) When you declare a method as abstract, can other non-abstract methods access it?**

Yes, other non-abstract methods can access a method that you declare as abstract.

**Q6) Can there be an abstract class with no abstract methods in it?**

Yes, there can be an abstract class without abstract methods.

**Q7) What is an Interface?**

An interface is a description of a set of methods that conforming implementing classes must have.

Note:

* You can’t mark an interface as final.
* Interface variables must be static.
* An Interface cannot extend anything but another interfaces.

**Q8) Can we instantiate an interface?**

You can’t instantiate an interface directly, but you can instantiate a class that implements an interface.

**Q9) Can we create an object for an interface?**

Yes, it is always necessary to create an object implementation for an interface. Interfaces cannot be instantiated in their own right, so you must write a class that implements the interface and fulfill all the methods defined in it.

**Q10) Do interfaces have member variables?**

Interfaces may have member variables, but these are implicitly public, static, and final- in other words, interfaces can declare only **constants**, not instance variables that are available to all implementations and may be used as key references for method arguments for example.

**Q11) What modifiers are allowed for methods in an Interface?**

Only public and abstract modifiers are allowed for methods in interfaces.

**Q12) What is a marker interface?**

Marker Interface is a special interface in Java without any field and method. Marker interface is used to inform compiler that the class implementing it has some special behavior or meaning. Some example of Marker interface is,

* java.io.serializable
* java.lang.Cloneable
* java.rmi.Remote
* java.util.RandomAccess

All these interfaces do not have any method and field. They only add special behavior to the classes implementing them. However marker interfaces have been deprecated since Java 5, they were replaced by **Annotations**. Annotations are used in place of Marker Interface that play the exact same role as marker interfaces did before.

**Q13) When to use Abstract class and when to use Interfaces?**

**Using Abstract Class**

* 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.

**Using Interfaces**

* You expect that unrelated classes would implement your interface. For example, the interfaces Comparable and Cloneable are implemented by many unrelated classes.
* 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 inheritances.

**1) Abstract class must have only abstract methods. True or false?**

False. Abstract methods can also have concrete methods.

**2) Is it compulsory for a class which is declared as abstract to have at least one abstract method?**

Not necessarily. Abstract class may or may not have abstract methods.

**3) Can we use “abstract” keyword with constructor, Instance Initialization Block and Static Initialization Block?**

No. Constructor, Static Initialization Block, Instance Initialization Block and variables can not be abstract.

**4) Why final and abstract can not be used at a time?**

Because, final and abstract are totally opposite in nature. A final class or method can not be modified further where as abstract class or method must be modified further. “final” keyword is used to denote that a class or method does not need further improvements. “abstract” keyword is used to denote that a class or method needs further improvements.

**5) Can we instantiate a class which does not have even a single abstract methods but declared as abstract?**

No, We can’t instantiate a class once it is declared as abstract even though it does not have abstract methods.

**6) Can we declare abstract methods as private? Justify your answer?**

No. Abstract methods can not be private. If abstract methods are allowed to be private, then they will not be inherited to sub class and will not get enhanced.

**7) We can’t instantiate an abstract class. Then why constructors are allowed in abstract class?**

It is because, we can’t create objects to abstract classes but we can create objects to their sub classes. From sub class constructor, there will be an implicit call to super class constructor. That’s why abstract classes should have constructors. Even if you don’t write constructor for your abstract class, compiler will keep default constructor.

**8) Can we declare abstract methods as static?**

No, abstract methods can not be static.

**9) Can a class contain an abstract class as a member?**

Yes, a class can have abstract class as it’s member.

**10) Abstract classes can be nested. True or false?**

True. Abstract classes can be nested i.e an abstract class can have another abstract class as it’s member.

**11) Can we declare abstract methods as synchronized?**

No, abstract methods can not be declared as synchronized. But methods which override abstract methods can be declared as synchronized.

**12) Can we declare local inner class as abstract?**

Yes. Local inner class can be abstract.

**13) Can abstract method declaration include throws clause?**

Yes. Abstract methods can be declared with throws clause.

**1) Can interfaces have constructors, SIB and IIB?**

No. Interfaces can’t have constructors, SIB and IIB. They show 100% abstractness.

**2) Can we re-assign a value to a field of interfaces?**

No. The fields of interfaces are static and final by default. They are just like constants. You can’t change their value once they got.

**3) Can we declare an Interface with “abstract” keyword?**

Yes, we can declare an interface with “abstract” keyword. But, there is no need to write like that. All interfaces in java are abstract by default.

**4) For every Interface in java, .class file will be generated after compilation. True or false?**

True. .class file will be generated for every interface after compilation.

**5) Can we override an interface method with visibility other than public?**

No. While overriding any interface methods, we should use public only. Because, all interface methods are public by default and you should not reduce the visibility while overriding them.

**6) Can interfaces become local members of the methods?**

No. You can’t define interfaces as local members of methods like local inner classes. They can be a part of top level class or interface.

**7) Can an interface extend a class?**

No, a class can not become super interface to any interface. Super interface must be an interface. That means, interfaces don’t extend classes but can extend other interfaces.

**8) Like classes, does interfaces also extend Object class by default?**

No. Interfaces don’t extend Object class. ( Click [here](https://javaconceptoftheday.com/interface-extend-object-class-java/) for more )

**9) Can interfaces have static methods?**

No. Interfaces can’t have static methods.

**10) Can an interface have a class or another interface as it’s members?**

Yes. Interfaces can have classes or interfaces as their members.

**11) What are marker interfaces? What is the use of marker interfaces?**

**Marker interfaces** in java are interfaces with no members declared in them. They are just an empty interfaces used to mark or identify a special operation. For example, Cloneable interface is used to mark cloning operation and Serializable interface is used to mark serialization and deserialization of an object. Marker interfaces give instructions to JVM that classes implementing them will have special behavior and must be handled with care.

Marker interfaces don’t provide any functionality. In earlier versions of Java (Before Java 5), marker interfaces are used to provide metadata to the readers. With the introduction of **annotations** from Java 5, annotations are used more instead of marker interfaces to provide metadata . But, still many use marker interfaces to mark the special behavior of a class.

## Java’s built-in Marker Interfaces :

These are some built-in marker interfaces in java which are used to mark some special behavior of a class.

**1) java.lang.Cloneable Interface :**

This interface is used to mark the cloning operation. An object of a class which implements Cloneable interface is eligible for field-by-field copying of an object.

**2) java.io.Serializable Interface :**

This interface is used to mark serialization and deserialization of an object. Serialization is a process in which an object state is read from memory and written into a file or a database. Deserialization is a process in which an object state is read from a file or a database and written back into memory. Any class which wants it’s object to be eligible for serialization and deserialization must implement Serializable interface.

**3) java.util.EventListener :**

This is also a marker interface which must be extended by all event listener interfaces.

**4) java.rmi.Remote :**

This is also a marker interface which is used to mark the invocation of a method remotely. Only methods of those classes which implement Remote interface are eligible for invocation by non-local virtual machine.

**5) RandomAccess:**

## User Defined Marker Interfaces :

You can define your own marker interfaces to indicate about any special behavior. Below is such an example. In this example, Cash and Checque are two marker interfaces which are used to indicate whether payment is done by cash or checque.

interface Cash

{

}

interface Checque

{

}

class PaymentOne implements Cash

{

    static void paymentByCash()

    {

        System.out.println("Payment is done by cash");

    }

}

class PaymentTwo implements Checque

{

    static void paymentByChecque()

    {

        System.out.println("Payment is done by checque");

    }

}

public class MainClass

{

    public static void main(String[] args)

    {

        PaymentOne.paymentByCash();

        PaymentTwo.paymentByChecque();

    }

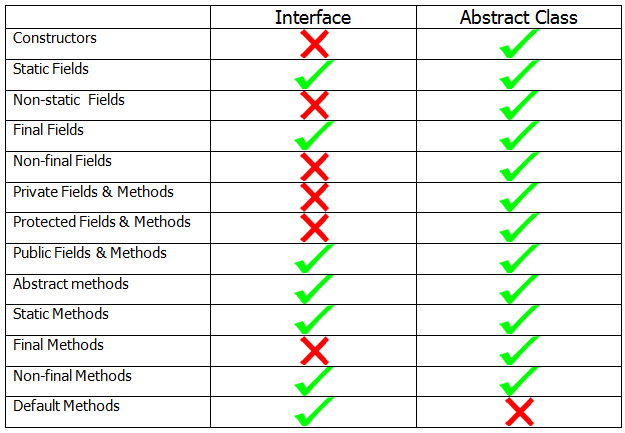
}

## Alternative To Marker Interfaces :

1) **Internal flags** can be used instead of marker interfaces to indicate any special operation.

2) **Annotations** are recommended instead of marker interfaces to indicate any special operation.

**Interface VS Abstract Classes**

[](https://javaconceptoftheday.com/wp-content/uploads/2019/04/InterfaceVsAbstractClassJava8.png)