**1. What is Abstraction in Java?**

Ans: [Abstraction in Java](https://www.scientecheasy.com/2020/05/java-abstraction.html/) is a technique by which we can hide the data that is not required to users. It hides all unwanted data so that users can work only with the required data.

**2. How to achieve or implement Abstraction in Java?**

Ans: There are two ways to implement abstraction in java. They are as follows:

a) Abstract class (0 to 100%)  
b) Interface (100%)

**3. What is Abstract class in Java? How to define it?**

Ans: An abstract class in java is a class that is declared with an abstract keyword.

Example of Abstract class:

abstract class Test {

abstract void show();

}

**4. What is the difference between abstract class and concrete class?**

Ans: There are mainly two differences between an abstract class and concrete class. They are:

a) We cannot create an object of abstract class. Only objects of its non-abstract (or concrete) sub classes can be created.

b) It can have zero or more abstract methods that are not allowed in a non-abstract class (concrete class).

**5. What is Abstract in Java?**

Ans: Abstract is a non-access modifier in java that is applicable for classes, interfaces, methods, and inner classes.

**6. Can abstract modifier applicable for variables?**

Ans: No.

No, in Java, the **abstract** modifier is not applicable to variables. The **abstract** keyword is primarily used with classes and methods to create abstract classes and define abstract methods within them.

Variables in Java cannot be abstract because abstractness doesn’t apply to fields or variables. Variables in classes can be instance variables or class variables, and they are typically concrete in nature—meaning they have a definite value or reference assigned to them.

However, variables can be declared as **final**, making them constants whose value cannot be changed once initialized. For instance:

**7. What is Abstract method in Java?**

Ans: A method which is declared with abstract modifier and has no implementation (means no body) is called abstract method in java.

It does not contain any body. It has simply a signature declaration followed by a semicolon. It has the following general form as given below.

Syntax:

abstract type MethodName(arguments); // No body

For example:

abstract void msg(); // No body.

**8. Can an abstract method be declared as static?**

Ans: No.

An abstract method can't be declared as static because abstract methods require each subclass to provide its unique implementation, while static methods belong to the class itself and cannot be overridden.

Top of Form

**9. Can an abstract method be declared with private modifier?**

Ans: No, it cannot be private because the abstract method must be implemented in the child class. If we declare it as private, we cannot implement it from outside the class.

**10. What is Concrete method in Java?**

Ans: A concrete method in Java is a method which has always the body. It is also called a complete method in java.

**11. When to use Abstract class in Java?**

Ans: An abstract class can be used when we need to share the same method to all non-abstract sub classes with their own specific implementations.

**12. When to use Abstract method in Java?**

Ans: An abstract method can be used

a) When the same method has to perform different tasks depending on the object calling it.  
b) When you need to be overridden in its non-abstract subclasses.

**13. Is abstract class a pure abstraction in Java?**

Ans: No, It provides 0 to 100% abstraction.

**14. Is it possible to create an object of abstract class in Java?**

Ans: No. It is not possible but we can create an object of its subclass.

**15. Is it possible that an abstract class can have without any abstract method?**

Ans: Yes.

**16. Can an abstract class have constructor?**

Ans: Yes.

**17. Is abstract class allow to define private, final, static, and concrete methods?**

Ans: Yes.

**18. Is it possible to achieve multiple inheritance through abstract class?**

Ans: No.

**19. Can we make an abstract class without abstract keyword?**

Ans: No, a class must be declared with abstract keyword to make an abstract class.

**20. Can we define an abstract method inside non-abstract class (concrete class)?**

Ans: No, we cannot define an abstract method in non-abstract class.

For example:

class Test {

abstract void show();

}

The above code will generate a compile-time error.

**21. Which among the following code have errors?**

a) abstract class A {

void m1();

}

b) public class A {

abstract void m1();

}

c) abstract public class A {

abstract void m1();

}

d) abstract public class A {

void m1() { }

}

e) public abstract class A {

abstract void m1();

A(){ }

void m2() { }

}

f) public abstract class A {

abstract int x = 100;

abstract void m1();

abstract void m2();

}

g) public abstract class A {

abstract void m1();

}

public class Test {

public static void main(String[] args) {

A a = new A();

}

}

h) public abstract class A {

abstract void m1();

A(){ }

static void m2() {System.out.println("Hello Java!"); }

}

public class B extends A {

void m1(){

A.m2();

}

}

i) public abstract class A {

abstract void m1();

private A(){ }

}

public class B extends A { }

Ans: a, b, f, g, i.

**22. Will the following code compile successfully? If yes, what will be the output of program?**

public abstract class A {

abstract void m1(A a);

}

public class B extends A {

void m1(A a) {

System.out.println("One");

}

}

public class C extends B {

void m1(B b) {

System.out.println("Two");

super.m1(new B());

}

}

public class Test {

public static void main(String[] args) {

C c = new C();

c.m1(new B());

}

}

Ans: Yes, the above code will be compiled successfully. The output of above program is Two, One.

**23. What will happen if we do not override all abstract methods in subclass?**  
Or, what will happen if we do not provide implementation for all abstract methods in subclass?

Ans: Java compiler will generate compile time error. We will have to override all abstract methods in subclass.

**24. What is the difference between Abstraction and Encapsulation?**

Ans: Abstraction hides the implementation details from users whereas, [encapsulation](https://www.scientecheasy.com/2020/07/encapsulation-in-java.html/) wraps (binds) data and code into a single unit.

**25. Why abstract class has constructor even though you cannot create object?**

Ans: We cannot create an object of abstract class but we can create an object of subclass of abstract class. When we create an object of subclass of an abstract class, it calls the constructor of subclass.

This subclass constructor has a super keyword in the first line that calls constructor of an abstract class. Thus, the constructors of an abstract class are used from constructor of its subclass.

If the abstract class doesn’t have constructor, a class that extends that abstract class will not get compiled.

**26. Why should we create reference to superclass (abstract class reference)?**

Ans: We should create a reference of the superclass to access subclass features because superclass reference allows only to access those features of subclass which have already declared in superclass.

If you create an individual method in subclass, the superclass reference cannot access that method. Thus, any programmer cannot add their own additional features in subclasses other than whatever is given in superclass.

**27. What is the advantage of Abstract class in Java?**

Ans: The main advantages of using abstract class are as follows:

* Abstract class makes programming better and more flexible by giving the scope of implementing abstract methods.
* Programmer can implement abstract method to perform different tasks depending on the need.
* We can easily manage code.

**28. Will the code compile successfully? If yes, what will be the output?**

public abstract class A {

abstract void m1();

}

public class B extends A {

void m1(){

System.out.println("m1 in class B");

}

}

public class Test {

public static void main(String[] args) {

B b = new B();

b.m1();

}

}

Ans: Yes, the code will be compiled successfully. The output is m1 in class B.

**29. Consider the below given code.**

public abstract class A {

abstract void m1();

void m2(){

System.out.println("One”);

}

}

**How to call m2() method in the above code?**

Ans: Make it static and call as A.m2();

**30. Identify the errors in the following code.**

public abstract class A {

abstract void m1();

void m2(){

System.out.println("One");

}

}

public class B extends A {

void m2(){

System.out.println("Two");

}

}

public class Test {

public static void main(String[] args) {

B b = new B();

b.m2();

}

}

Ans: The abstract method m1() has been not implemented (overridden) in subclass B. Therefore, we will get a compile-time error.

**31. Will the given code compile successfully? If yes, how?**

a)

public abstract class A {

abstract void m1();

}

public class B extends A {

private void m1(){

System.out.println("One");

}

}

public class Test {

public static void main(String[] args) {

B b = new B();

b.m1();

}

}

Ans: No, the code will not be compiled successfully because we cannot reduce the visibility of the inherited method from A.

b)

public abstract class A {

abstract void m1();

}

public class B extends A {

protected void m1(){

System.out.println("One");

}

}

public class Test {

public static void main(String[] args) {

B b = new B();

b.m1();

}

}

Ans: Yes, the code will be compiled successfully. The output is One.

c)

public abstract class A {

public abstract void m1();

}

public class B extends A {

protected void m1() {

System.out.println("One");

}

}

public class Test {

public static void main(String[] args) {

B b = new B();

b.m1();

}

}

Ans: No, compile-time error because we cannot reduce the visibility of inherited method m1() from A.

d)

public abstract class A {

private abstract void m1();

}

public class B extends A {

protected void m1(){

System.out.println("One");

}

}

public class Test {

public static void main(String[] args) {

B b = new B();

b.m1();

}

}

Ans: Abstract method m1() in class A cannot be private.

e)

public abstract class A {

protected abstract void m1();

}

public class B extends A {

void m1(){

System.out.println("One");

}

}

public class Test {

public static void main(String[] args) {

B b = new B();

b.m1();

}

}

Ans: No, compile time error.

Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

Example of abstract class and interface in Java

Let's see a simple example where we are using interface and abstract class both.

Backward Skip 10sPlay VideoForward Skip 10s

1. //Creating interface that has 4 methods
2. **interface** A{
3. **void** a();//bydefault, public and abstract
4. **void** b();
5. **void** c();
6. **void** d();
7. }
9. //Creating abstract class that provides the implementation of one method of A interface
10. **abstract** **class** B **implements** A{
11. **public** **void** c(){System.out.println("I am C");}
12. }
14. //Creating subclass of abstract class, now we need to provide the implementation of rest of the methods
15. **class** M **extends** B{
16. **public** **void** a(){System.out.println("I am a");}
17. **public** **void** b(){System.out.println("I am b");}
18. **public** **void** d(){System.out.println("I am d");}
19. }
21. //Creating a test class that calls the methods of A interface
22. **class** Test5{
23. **public** **static** **void** main(String args[]){
24. A a=**new** M();
25. a.a();
26. a.b();
27. a.c();
28. a.d();
29. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=Test5)

Output:

I am a

I am b

I am c

I am d