**1) What is interface in java?**

**Java interface is a blueprint of a class and it is used to achieve fully abstraction and it is a collection of abstract methods.**

**(2) Can we achieve multiple inheritance by using interface?**

**Yes, we can achieve multiple inheritance through the interface in java but it is not possible through class because java doesn't support multiple inheritance through class.**

**(3) How to declare an interface, write a syntax?**

**The syntax of declaring the interface in java by using interface keyword :**

**interface FirstInterface**

**{**

**fields;**

**methods;**

**}**

**(4) Can we create an object of an interface?**

**No, we cannot create an object of interface.**

**(5) Can we declare the interface as final?**

**No, we can't declare the interface as final because the implementation of the interface is provided by another class. If we make the interface as final, it will throw a compile-time error.**

**(6) Which keyword java compiler add before interface fields and methods?**

**In an interface, Java compiler adds public, static and final keywords before fields or data members and add public abstract keywords before methods. In other words, all the fields are public, static and final and all the methods are public and abstract by default in an interface.**

**(7) Does interface extend Object class by default?**

**No, Interface does not extend Object class in java by default but all the classes extend Object class by default.**

**(8) Can an interface extend another interface?**

**Yes, an interface can extend another interface.**

**(9) Can an interface extend a class?**

**No, A class can implement an interface but interface cannot extend a class.**

**(10) Can we put a static method in interfaces?**

**No, we cannot put static methods in interfaces because all the methods are by default public and abstract in the interface and we cannot use abstract and static keywords together.**

**(11) Can we declare an interface with the abstract keyword?**

**Yes, we can declare abstract keyword with interfaces but there is no need to write abstract keyword with interfaces because all the interfaces are abstract by default.**

**(12) What is default keyword in an interface?**

**By the help of default keyword, we can keep non-abstract method in java interface i.e with method body{}. This is the new feature of JAVA 8.**  
 **Syntax of default keyword :**  
 **interface Test**  
**{**  
**default void show()**  
**{**  
**-----**  
**-----**  
**}**  
**}**

**(13) What is marker or tagged interface?**

**Marker interface is an interface that has no data member and method like Serializable, Cloneable, etc.**

**(14) Can we declare a constructor in the interface?**

**No.**

**(15) After compilation of interface program, .class file will be generated for every interface in java... true or false.?**

**This is true.**

**(16) Can we change the value of a field in interface after initialization?**

**No, Because all the fields of the interface are by default final.**

**(17) Difference between abstract class and interface?**

**There are many**[**differences between abstract class and interface**](https://javatutorial95.blogspot.com/2017/07/difference-between-abstract-class-and-interface-in-java.html)**.**

***abstract class***

* **Through abstract class, you cannot achieve multiple inheritance.**
* **You can keep non-abstract method(with method body) in the abstract class.**
* **In the abstract class, fields are not public, static, final and methods are not public abstract by default.**

***interface***

* **Through the interface, you can achieve multiple inheritance.**
* **In an interface, you can't keep non-abstract method but since java 8 it is possible.**
* **In an interface, fields are public, static, and final and methods are public and abstract by default.**

How To Describe Interfaces In Interviews?

* An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods.
* An interface cannot be  instantiate.
* An interface does not contain any constructors.
* An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
* An interface is not extended by a class; it is implemented by a class.
* An interface can extend multiple interfaces.

Question: What is an Interface in Java?

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

Question: What will happen if we define a concrete method in an interface in Java?

By default interface methods are abstract.

if we declare any concrete method in an interface compile time error will come.

Error:Abstract methods do not specify a body.

Question: Can we create non static variables in an interface?

No.We can not create non static variables in an interface. If we try to create non static variables compile time error comes.

By default members will be treated as public static final variables so it expects some value to be initialized.

package codespaghetti.com;

interface JavaInterface{

int x, y; // compile time error

}

Question: What will happen if we do not initialize variables in Java interface.

Compile time error will come because by default members will be treated as public static final variables so it expects some value to be initialized.

package codespaghetti.com;

interface JavaInterface{

int x, y; // compile time error: The blank final field y may not have been initialized

}

Question: Can we declare interface members as private or protected?

No.

package codespaghetti.com;

interface JavaInterface{

private int x; // compile time error: Illegal modifier for the interface field Sample.x; only

public, static & final are permitted

protected int a; // compile time error: Illegal modifier for the interface field Sample.a; only

public, static & final are permitted

}

Question: When we need to use extends and implements?

A class will implements an interface. A class will extends another class. An interface extends another interface.

Question: Can we create object for an interface in Java?

NO. We can not create object for interface. We can create a variable for an interface

package codespaghetti.com;

interface JavaInterface{

void show();

}

package com.instanceofjava;

interface A implements JavaInterface {

void show(){

// code

}

public static void main(String args[]){

JavaInterface obj= new JavaInterface(); // Error: Cannot instantiate the type JavaInterface

}

}

Question: Can we declare interface as final?

No. Compile time error will come. Error: Illegal modifier for the interface Sample; only public & abstract are permitted

Question:   Can we declare constructor inside an interface?

No. Interfaces does not allow constructors.The variables inside interfaces are static final variables means constants and we can not create object for interface.

So there is no need of constructor in interface that is the reason interface doesn't allow us to create constructor.

Question: Question : What will happen if we are not implementing all the methods of an interface in class which implements an interface?

A class which implements an interface should implement all the methods (abstract) otherwise compiler will throw an error. The type Example must implement the inherited abstract method JavaInterface.show() If we declare class as abstract no need to implement methods. No need of overriding default and static methods.

package codespaghetti.com;

interface JavaInterface{

void show();

}

package com.instanceofjava;

interface A implements JavaInterface { // The type Example must implement the inherited

abstract method JavaInterface.show()

public static void main(String args[]){

}

}

Question: How can we access same variables defined in two interfaces implemented by a class?

Yes, By Using corresponding interface.variable\_name we can access variables of corresponding interfaces.

Question: If Same method is defined in two interfaces can we override this method in class implementing these interfaces.

Yes, implementing the method once is enough in class. A class cannot implement two interfaces that have methods with same name but different return type.

Question: 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.

Question: 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.

Question: 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.

Question: 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.

Question: 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.

Question: 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.

Question: Like classes, does interfaces also extend Object class by default?

No. Interfaces don’t extend Object class.

Question: Can interfaces have static methods?

No. Interfaces can’t have static methods. Interfaces can have static methods since Java 1.8.

Question: Advantage and disadvantages of Interfaces

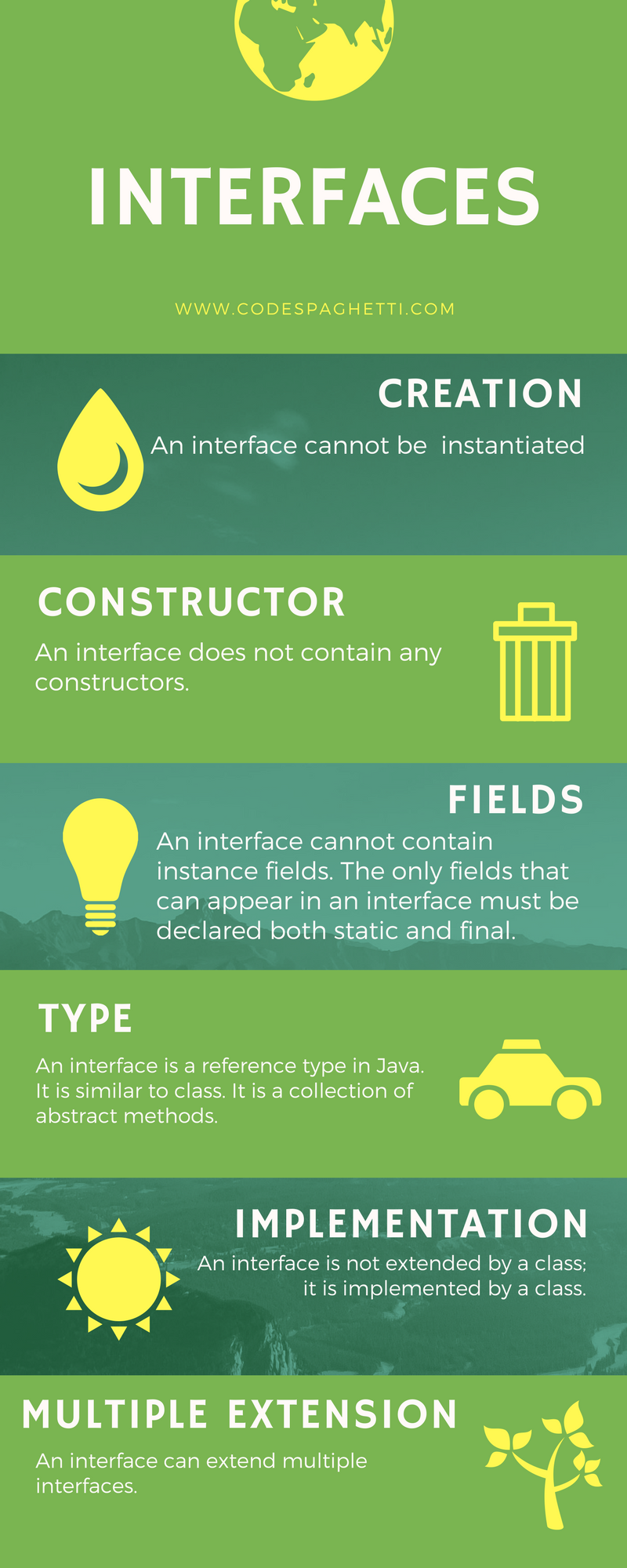
**Advantages**

* Interfaces are mainly used to provide polymorphic behavior.
* Interfaces function to break up the complex designs and clear the dependencies between objects.

**Disadvantages**

* Java interfaces are slower and more limited than other ones.
* Interface should be used multiple number of times else there is hardly any use of having them.

Summary



**1. What is an interface in Java?**

Ans: An interface in Java is a mechanism that is used to achieve complete abstraction. It is basically a kind of class that contains only constants and abstract methods.

**2. Can we define private and protected modifiers for data members (fields) in interfaces?**

Ans: No, we cannot define private and protected modifiers for variables in interface because the fields (data members) declared in an interface are by default public, static, and final.

**3. Which modifiers are allowed for methods in an Interface?**

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

**4. Suppose A is an interface. Can we create an object using new A()?**

Ans: No, we cannot create an object of interface using new operator. But we can create a reference of interface type and interface reference refers to objects of its implementation classes.

**5. Can we define an interface with a static modifier?**

Ans: Yes, from Java 8 onwards, we can define static and default methods in an interface. Prior to Java 8, it was not allowed.

**6. Suppose A is an interface. Can we declare a reference variable a with type A like this: A a;**

Ans: Yes.

**7. Can an interface extends another interface in Java?**

Ans: Yes, an interface can extend another interface.

**8. Can an interface implement another interface?**

Ans: No, an interface cannot implement another interface.

**9. Is it possible to define a class inside an interface?**

Ans: Yes, we can define a class inside an interface.

**10. Which of the following is a correct representation of interface?**



a) public abstract interface A {

   abstract void m1() {};

  }

b) public abstract interface A {

   void m1();

  }

c) abstract interface A extends B, C {

    void m1();

  }

d) abstract interface A extends B, C {

   void m1(){};

  }

e) abstract interface A {

   m1();

  }

f) interface A {

   void m1();

  }

g) interface A {

   int m1();

  }

h) public interface A {

   void m1();

  }

  public class B implements A {

  }

i) public interface A {

    void m1();

  }

  Public interface B {

    void m1();

  }

  public interface C extends A, B {

   void m1();

  }

​

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

**11. Identify the error in the following code.**

a)



1

interface A {

2

void m1();

3

}

4

public class B implements A {

5

void m1(){

6

 System.out.println("One");

7

}

8

}

9

​

Ans: We cannot reduce the visibility of inherited method from interface A.

b)



1

interface A {

2

A() { }

3

void m1();

4

}

5

public abstract class B implements A {

6

public void m1(){

7

 System.out.println("One");

8

}

9

}

10

​

Ans: An interface cannot have a constructor.

c)



1

interface A {

2

int x;

3

void m1();

4

}

5

public class B implements A {

6

int x = 20;

7

public void m1(){

8

 System.out.println("One");

9

}

10

}

11

​

Ans: A variable in an interface must be initialized at the time of declaration.

**12. What will be the output of the following program?**



1

interface A {

2

int x = 10;

3

void m1();

4

}

5

public class B implements A {

6

int x = 20;

7

public void m1(){

8

 System.out.println("One");

9

}

10

}

11

public class Test {

12

public static void main(String[] args){

13

A a = new B();

14

 a.m1();

15

 System.out.println(a.x);

16

}

17

}

18

​

Ans: Output: One, 10.

**13. Can an interface extend multiple interfaces?**

Ans: Yes, an interface can extend multiple interfaces.

**14. Can an interface has instance and static blocks?**

Ans: No.

**15. What will be the output of the following program?**



1

interface A {

2

int x = 10;

3

}

4

interface B {

5

int x = 20;

6

}

7

interface C extends A, B{

8

int x = 30;

9

public static void main(String[] args){

10

int a = A.x;

11

int b = B.x;

12

int c = C.x;

13

14

System.out.println(a);

15

System.out.println(b);

16

System.out.println(c);

17

}

18

}

19

​

Ans: Output: 10, 20, 30

**16. What happens if a class has implemented an interface but has not provided implementation for that method defined in Interface?**

Ans: The class has to be declared with an abstract modifier. This will be enforced by the Java compiler.

**17. Why an Interface method cannot be declared as final in Java?**  
Or, Can a method within an interface be marked as final?

Ans: Not possible. Doing so will result the compilation error problem. This is because a final method cannot be overridden in java. But an interface method should be implemented by another class.

So, the interface method cannot be declared as final. The modifiers such as public and abstract are only applicable for method declaration in an interface.

**18. Can an interface be final?**

Ans: No. Doing so will result compilation error problem.

**19. Why an interface cannot have a constructor?**

Ans: Inside an interface, a constructor cannot be called using super keyword with hierarchy.

**20. Why an Interface can extend more than one Interface but a Class can’t extend more than one Class?**

Ans: We know that Java doesn’t allow multiple inheritance because a class extends only one class. But an Interface is a pure abstraction model. It does not have inheritance hierarchy like classes.

Therefore, an interface allows to extend more than one Interface.

**21. What is the use of interface in Java?**  
Or, why do we use an interface in Java?

Ans: There are many reasons to use interface in java. They are as follows:

a. An interface is used to achieve fully abstraction.  
b. Using interfaces is the best way to expose our project’s API to some other project.  
c. Programmers use interface to customize features of software differently for different objects.  
d. By using interface, we can achieve the functionality of multiple inheritance.

**22. Is it necessary to implement all abstract methods of an interface?**

Ans: Yes, all the abstract methods defined in interface must be implemented.

**23. Can we define a variable in an interface? What type it should be?**

Ans: Yes, we can define variable in an interface that must be implicitly static and final.

**24. Can we re-assign a value to a variable of interface?**

Ans: No, variables defined inside the interface are static and final by default. They are just like constants. We can’t change their value once they got.

**25. What is the difference between abstract class and interface in Java?**

Ans: Refer to this tutorial: [12 Difference between Abstract class and Interface in Java](https://www.scientecheasy.com/2020/07/difference-between-abstract-class-interface.html/)

**26. What is the difference between class and interface in Java?**

Ans: Refer to this tutorial: [Difference between Class and Interface in Java](https://www.scientecheasy.com/2020/07/difference-between-class-interface.html/)

**27. What is a Marker Interface in Java?**

Ans: An Interface that doesn’t have any data members or methods is called marker interface in java. For example, Serializable, Cloneable, Remote, etc.

**28. What is a Nested interface?**

Ans: An interface declared inside another interface is called nested interface. By default, it is static in nature. It is also known as static interface.

**29. Can we reduce the visibility of interface method while overriding?**

Ans: No, while overriding any interface methods, we must use public only. This is because all interface methods are public by default. We cannot reduce the visibility while overriding them.

**30. Can we define an interface inside a method as local member?**

Ans: No, we can’t define an interface as local member of a method like local inner class.

**31. Will the code compiled successfully? If yes, what will be the output?**



1

interface A {

2

void m1(int x, double y);

3

void m2(String z);

4

}

5

class B implements A {

6

public void m1(int x, double y){

7

 System.out.println("Hello");

8

}

9

public void m2(String z){

10

 System.out.println("Java");

11

}

12

void m3(){

13

 System.out.println("Hello Java!");

14

}

15

}

16

public class Test {

17

public static void main(String[] args){

18

B b = new B();

19

b.m1(20, 10.0);

20

b.m2(null);

21

b.m3();

22

}

23

}

24

​

Ans: Yes, the code will be compiled successfully. The output is Hello, Java, Hello Java!.

**32. What will be the output of the following programs?**

a)



1

interface A {

2

void m1(int x, double y);

3

}

4

abstract class B implements A {

5

public void m1(double x, int y){

6

 System.out.println("One");

7

}

8

}

9

public class Test extends B {

10

public void m1(double x, int y){

11

System.out.println("Two");

12

super.m1(20.5, 20);

13

}

14

public void m1(int x, double y){

15

System.out.println("Three");

16

}

17

public static void main(String[] args){

18

Test t = new Test();

19

t.m1(20.5, 10);

20

t.m1(10, 5.5);

21

}

22

}

23

​

Ans: Output: Two, One, Three

b)



1

interface A {

2

void m1(A a);

3

}

4

abstract class B implements A {

5

 void m1(B b){

6

 System.out.println("One");

7

}

8

}

9

public class Test extends B {

10

public void m1(A a){

11

System.out.println("Two");

12

}

13

public void m1(B b){

14

System.out.println("Three");

15

}

16

public static void main(String[] args){

17

A a = new Test();

18

a.m1(new Test());

19

}

20

}

21

​

Ans: Output: Two

c)



1

interface A {

2

void m1(A a);

3

}

4

abstract class B implements A {

5

void m1(B b){

6

 System.out.println("One");

7

}

8

}

9

public class Test extends B {

10

void m1(A a){

11

System.out.println("Two");

12

}

13

void m1(B b){

14

System.out.println("Three");

15

}

16

public static void main(String[] args){

17

A a = new Test();

18

a.m1(null);

19

}

20

}

21

​

Ans: No, *IllegalAcceessError* exception because we cannot reduce the visibility of inherited method from interface A.

d)



1

interface A {

2

void m1();

3

}

4

abstract class B implements A {

5

public void m1(){

6

 System.out.println("One");

7

}

8

}

9

public class Test extends B {

10

public void m1(){

11

System.out.println("Two");

12

super.m1();

13

}

14

public static void main(String[] args){

15

A a = new Test();

16

a.m1();

17

B b = new Test();

18

 b.m1();

19

}

20

}

21

​

Ans: Output: Two, One, Two, One

e)



1

interface A {

2

int m1();

3

}

4

class B implements A {

5

public int m1(){

6

 return 20;

7

}

8

}

9

public class C implements A {

10

public int m1(){

11

 return 30;

12

}

13

}

14

public class Test {

15

public static void main(String[] args){

16

A a = new B();

17

int aa =  a.m1();

18

System.out.println(aa);

19

20

C c = new C();

21

int cc = c.m1();

22

System.out.println(cc);

23

}

24

}

25

​

Ans: Output: 20, 30.

f)



1

interface A {

2

A m1();

3

}

4

class B implements A {

5

public A m1(){

6

 System.out.println("Red");

7

 return null;

8

}

9

}

10

public class C implements A {

11

public A m1(){

12

 System.out.println("Orange");

13

 return null;

14

}

15

}

16

public class Test {

17

public static void main(String[] args){

18

A a = new B();

19

 a.m1();

20

C c = new C();

21

 c.m1();

22

}

23

}

24

​

Ans: Output: Red, Orange.

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



1

interface A {

2

A m1();

3

}

4

class B implements A {

5

public B m1(){

6

 System.out.println("Red");

7

 return new B();

8

}

9

}

10

public class C implements A {

11

public C m1(){

12

 System.out.println("Orange");

13

 return new C();

14

}

15

}

16

public class Test {

17

public static void main(String[] args){

18

A a;

19

 a = new B();

20

 a.m1();

21

 a = new C();

22

 a.m1();

23

}

24

}

25

​

Ans: Yes, the code will be successfully compiled. The result is Red, Orange.

**34. What will be the output of the following program?**



1

interface A {

2

 int x = 20;

3

 interface B {

4

   int x = 30;

5

}

6

}

7

class C implements A {

8

int x = 40;

9

}

10

public class D implements A.B {

11

int x = 50;

12

}

13

public class Test {

14

public static void main(String[] args){

15

System.out.println(A.x);

16

System.out.println(A.B.x);

17

C c = new C();

18

System.out.println(c.x);

19

D d = new D();

20

System.out.println(d.x);

21

}

22

}

23

​

Ans: Output: 20, 30, 40, 50.

**35. Identify the error inside the following code and correct it. What will be the output after the correction of error?**



1

interface A {

2

 void m1();

3

 interface B {

4

   void m1();

5

}

6

}

7

class C implements A.B {

8

public void m1(){

9

 System.out.println("Green");

10

}

11

}

12

public class Test {

13

public static void main(String[] args){

14

A a = new C();

15

a.m1();

16

}

17

}

18

​

Ans: This code will generate compilation error because class C is implementing inner interface, not outer interface. But the reference of a is a type of outer interface. The correct code is as follows:

A.B ab = new C();

After correction, the output is Green.

**36. Is there any error in the following code? If yes, correct it.**



1

interface A {

2

 void m1();

3

 interface B {

4

   void m2();

5

}

6

}

7

class C implements A, A.B {

8

public void m1(){

9

 System.out.println("Green");

10

}

11

}

12

​

Ans: Yes, there is an error in the above code because class C must implement inherited abstract method from inner interface A.B.

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



1

interface A {

2

 void m1();

3

 interface B {

4

   void m2();

5

}

6

}

7

class C implements A, A.B {

8

public void m1(){

9

 System.out.println("Green");

10

}

11

public void m2(){

12

 System.out.println("Red");

13

}

14

}

15

public class Test {

16

public static void main(String[] args){

17

C c = new C();

18

c.m1();

19

c.m2();

20

}

21

}

22

​

Ans: Yes, the above code will be compiled successfully. The output is Green, Red.

**38. Identify the error in the below code. If no error, what will be the output of the program?**



1

interface A {

2

 void m1();

3

 interface B {

4

   void m2();

5

}

6

}

7

interface C extends A, A.B {

8

strictfp default void m1(){

9

 System.out.println("Green");

10

}

11

strictfp default void m2(){

12

 System.out.println("Red");

13

}

14

}

15

public class Test implements C{

16

public static void main(String[] args){

17

Test t = new Test();

18

 t.m1();

19

 t.m2();

20

}

21

}

22

​

Ans: No error. The result is Green, Red.

**39. What will be the output of following program if no error?**



1

public class A {

2

void m1(){

3

System.out.println("One");

4

}

5

interface B {

6

   void m2();

7

}

8

}

9

class C extends A {

10

void m1(){

11

 System.out.println("Two");

12

 super.m1();

13

}

14

}

15

public class Test implements A.B{

16

public void m2(){

17

System.out.println("Three");

18

}

19

public static void main(String[] args){

20

Test t = new Test();

21

 t.m2();

22

C c = new C();

23

 c.m1();

24

}

25

}

26

​

Ans: No error. The output is Three, Two, One.

**40. What is a Functional interface in Java 8?**

Ans: An interface that has exactly one abstract method is called functional interface in java. It is also known as single abstract method interface.

Functional interface can have three kinds of methods: abstract, default, and static methods.

Functional interface can have default methods with implementation. A default method cannot be abstract. For example, java.lang.Runnable and java.util.concurrent.Callable are two very popular Functional interfaces in java 8.

**41. How to define a Functional interface in Java 8?**

Ans: To define a Functional interface in Java 8, we can create an interface with exactly one abstract method like this:



1

interface A {

2

 void m1();

3

}

Another way is to define an Interface with annotation @FunctionalInterface. This annotation gives an instruction to the java compiler to verify that the interface has only one abstract method.

**42. Is it mandatory to use @FunctionalInterface annotation with Functional interface in Java 8?**

Ans: No, it is not mandatory to define a Functional interface with @FunctionalInterface annotation. Java does not impose this rule.

But, if we define an interface with @FunctionalInterface annotation, Java Compiler will give an error in case we define more than one abstract method within that interface.

**43. What is default method in Java 8?**

Ans: A method that is defined inside an interface with the default keyword and concrete method body is called default method. It provides a default implementation for classes that implements an interface.

This method can be overridden by a class that implements interface. A default method is public by definition and cannot be declared with private, protected, static, final, or abstract modifiers.

**44. Which of the following are valid or non-valid default methods defined in an interface?**



1

a) interface A {

2

   abstract void m1();

3

   default int m2() { return 10; }

4

}

5

b) interface A {

6

    abstract void m1();

7

    public default void m2(){ System.out.println("Hello Java");}

8

  }

9

c) interface A {

10

    static default m1() { }

11

  }

12

d) interface A {

13

     abstract void m1();

14

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

15

     public default void m3();

16

    }

17

​

Ans: a and b are valid codes. c and d are non-valid codes.

**45. What is the output of the following program?**



1

interface A {

2

abstract void m1();

3

static void m2(){

4

 System.out.println("Static method");

5

}

6

default void m3(){

7

 System.out.println("Default method");

8

}

9

}

10

class B implements A {

11

public void m1(){

12

 System.out.println("Overridden m1 method");

13

}

14

}

15

public class Test{

16

public static void main(String[] args){

17

A a = new B();

18

a.m1();

19

A.m2();

20

a.m3();

21

}

22

}

23

​

Ans: The result is Overridden m1 method, Static method, Default method.

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



1

interface A {

2

abstract void m1();

3

public static void m2(){

4

 System.out.println("One");

5

}

6

default void m3(){

7

 System.out.println("Two");

8

}

9

}

10

class B implements A {

11

public void m1(){

12

 System.out.println("Three");

13

}

14

void m3(){

15

 System.out.println("Four");

16

}

17

}

18

public class Test{

19

public static void main(String[] args){

20

A a = new B();

21

a.m1();

22

A.m2();

23

a.m3();

24

}

25

}

26

​

Ans: No, the code will not be compiled successfully because we cannot reduce the visibility of inherited m3() method from interface A.

**47. Identify the error in the following code and correct it.**



1

interface A {

2

abstract void m1();

3

abstract default void m2(){

4

 System.out.println("Two");

5

}

6

}

7

class B implements A {

8

public void m1(){

9

 System.out.println("Three");

10

}

11

}

12

public class Test{

13

public static void main(String[] args){

14

A a = new B();

15

a.m1();

16

a.m2();

17

}

18

}

19

​

Ans: Default method cannot be abstract. Remove it and then compile.

**48. Why do we need Functional interface in Java 8?**  
Or, what is the use of Functional interface in Java 8?

Ans: A Functional interface is mainly used in Lambda expressions, method reference, and constructor references.

**49. Can we declare private method inside an interface in JDK 9?**

Ans: Before JDK 9, all methods defined in an interface were implicitly public. But JDK 9 allow to declare private method inside an interface.

The private method defined in an interface cannot be called or overridden by implementing classes.

**50. Can we declare a method in an interface with combinations of modifiers like abstract, public, static, and private in JDK 9?**

Ans: No, because they do not make sense.

**Table: Supported Modifiers in Method Declaration in Interface**

|  |  |  |
| --- | --- | --- |
| **Modifiers** | **Supported?** | **Description** |
| public static | Yes | Supported since JDK 8. |
| public abstract | Yes | Supported since JDK 1. |
| public default | Yes | Supported since JDK 8. |
| private static | Yes | Supported since JDK 9. |
| private | Yes | Supported since JDK 9. It is a non-abstract instance method. |
| private abstract | No | This combination does not make sense because a private method is not inherited. So, it cannot be overridden. |
| private default | No | It is also not possible. |

Hope that these 50 Java interface interview questions and answers will help you to understand the level of questions asked by interviewers in different companies. I hope that you will have practiced interface interview programming questions and enjoyed them.

**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?**

( Click [here](https://javaconceptoftheday.com/marker-interface-java/) to see about marker interfaces in java. )

**What is Interface in Java? OR Explain Interface in Java?**

**Till the Java 7 version,**

* An interface in Java is a pure abstract class which means all methods are abstract and variables are constants
* By default, all methods inside an interface are public & abstract and variables are public, static & final
* An interface is a means to achieve full abstraction in Java

**Post-Java 8 release,**

* An interface can contain default and static methods, in addition to abstract methods
* Though it looks too similar to abstract classes they are actually different in many ways
* Read more about Java 8 default and static methods in Interface in detail

**Can we instantiate an interface?**

* No, we cannot instantiate an interface
* Since interface doesn’t have a constructor and contains only abstract methods and constants, therefore, we don’t need to instantiate
* Instead implementing classes provide concrete implementation for these abstract methods and constants can be accessed using <interfaceName>.<variableName>

**Post-Java 8 release,**

* In addition to abstract methods and constants, Java 8 introduced default and static methods
* default methods can be accessed using implementing class’s reference object
* static methods can be accessed using interface name i.e.; <interfaceName>.<staticMethodName>
* Read more about Java 8 default and static methods in Interface in detail

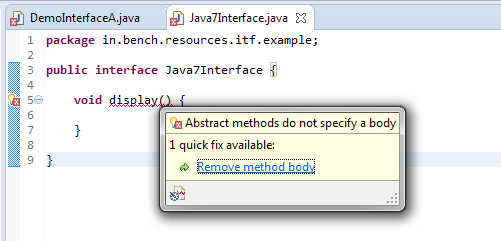
**Can we create an object for an interface?**

* No, we cannot create an object of an interface

**Q) What happens if we define a concrete method inside Interface?**

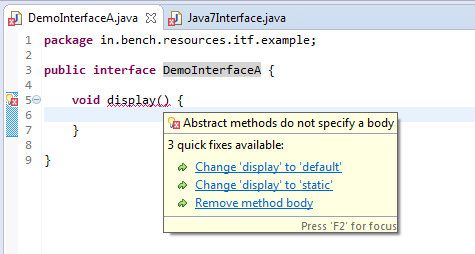
**Till the Java 7 version,**

* A compilation error will be thrown stating below the reason
* **Compile-time error:**Abstract methods do not specify a body



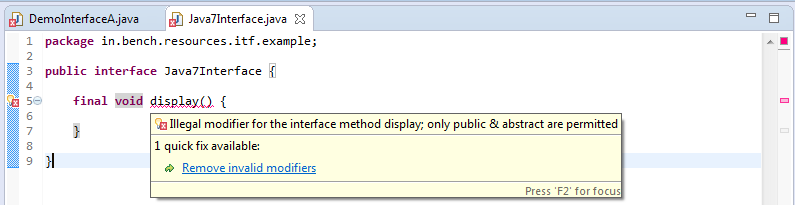
**Post-Java 8 release,**

* Concrete method (method with the body) are allowed with default or static keyword prefixed, as shown below
* Otherwise, even in Java 8 compilation error will be thrown as seen below screen capture



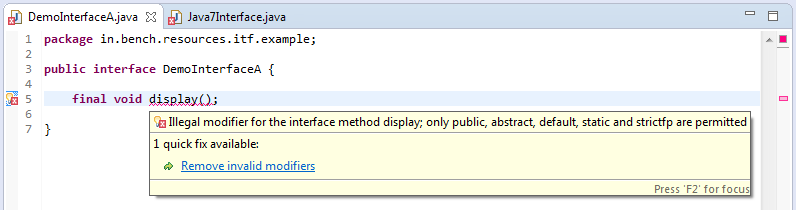
**Can a method inside an interface be declared as final?**

* By default, methods declared inside an interface are public & abstract even if we don’t declare it explicitly compiler adds this modifier during compilation time
* An interface allows only public & abstract modifiers in a method declaration
* If final keyword added in method declaration then the compiler will throw an error as seen in the below screen capture
* Compile-time error: Illegal modifier for the interface method display; only public & abstract are permitted



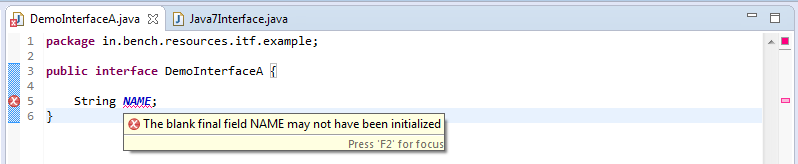
**Post-Java 8 release,**

* Still, the final modifier is not allowed in any of the methods in interface i.e.; abstract, default & static methods
* **Compile-time error:** Illegal modifier for the interface method display; only public, abstract, default, static and strictfp are permitted



**What happens if we don’t initialize variables inside Interface?**

* Compiler throws an error stating final variable needs to be initialized
* As variables defined inside an interface are by default public, static & final. So, the final variable always needs to be initialized where it is declared
* **Compile-time error:** The blank final field <fieldname> may not have been initialized



* No change even post Java 8 release

**Can we declare members as private or protected modifier inside Interface?**

* Variables (members) defined inside an interface are by default public, static & final
* Therefore, no other access-modifier allowed except public
* During the compilation process, compiler inserts/ add a public, static & final keyword for variables
* These are interface variables and are accessed using the interface name
* For example, <interfaceName>.<memberName> from any other class
* No change even post Java 8 release too

**How can we access variables defined inside Interface?**

* Members defined inside the interface can be accessed using interface name from any other class
* For example, <interfaceName>.<memberName> from any other class
* No change even post Java 8 release too

**Can we modify variables defined inside Interface?**

* Since variables defined inside the interface are final therefore we cannot change the value of these variables anywhere (simple OOPS concept)
* If we try to change the value, the compiler throws an error
* **Compile-time error:** The final field <interfaceName>.<fieldName> cannot be assigned
* No change even post Java 8 release too

**Can we re-assign a value to a field of interface?**

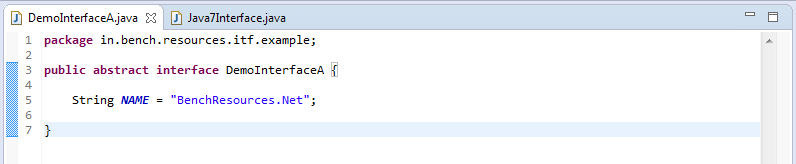
* Re-assigning fields throw a compile-time error as these are final by default
* **Compile-time error:** The final field <interfaceName>.<fieldName> cannot be assigned

**What modifiers are allowed for methods in an interface?**

* **Till the Java 7 version**, only public & abstract are permitted
* **Post-Java 8 release**, only public, abstract, default, static and strictfp are permitted

**Is it ok to add an “abstract” keyword to interface definition?**

* Yes, we can actually add an abstract keyword to interface definition (somewhat similar to abstract classes)



**Whether class compiles successfully if we don’t implement any of the abstract methods from Interface?**

* No, the compilation error will be thrown
* If a class implements an interface then it must provide a definition or concrete implementation for every abstract method

**Post-Java 8 release,**

* Still, implementing class must provide a definition or concrete implementation for every abstract method in an interface
* Exceptional being default and static methods; it is okay if we don’t override the default method
* Read more about Java 8 default and static methods in Interface in detail

**What is the best possible solution if we don’t want to implement a few of the abstract methods from Interface?**

* The best solution is to declare the implementing class as abstract; compilation will succeed
* But next inheriting class (i.e.; extending this class) must provide concrete method implementation or declare again as abstract.

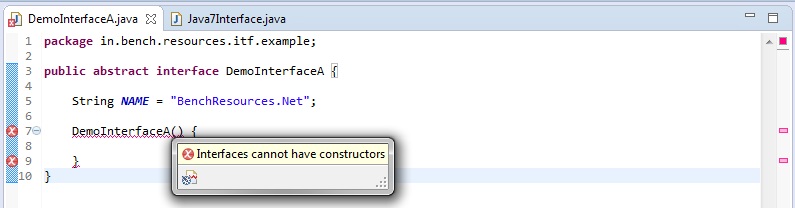
**Release Also: [HashMap vs LinkedHashMap In Java Detail](https://www.softwaretestingo.com/hashmap-vs-linkedhashmap/)**

**Can we reduce the visibility of the methods while implementing an interface?**

* By default abstract methods declared inside an interface are public
* As per overriding rule, access visibility of the methods can be widened further
* So, it is must declare overriding methods as public; as no other access visibility is wider than public
* Read more about Java overriding rules here

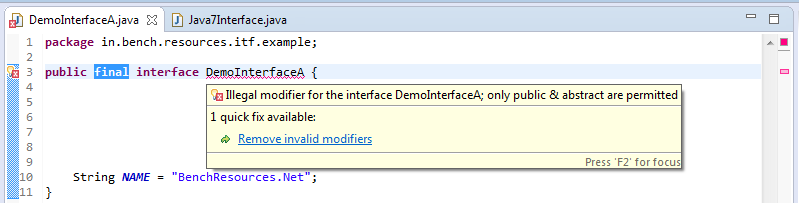
**Can we declare a constructor inside the interface?**

* A compilation error will be thrown stating “Interfaces cannot have constructors”
* **Compile-time error:** Interfaces cannot have constructors



**Can an interface be final?**

* No, an interface cannot be final and compilation error will be thrown
* **Compile-time error:** Illegal modifier for the interface <interfaceName>; only public and abstract are permitted



**Can interface extend any class?**

* No, an interface cannot extend any class
* An interface can only extend one or more other interfaces

**Can an interface implement any other interface?**

* No, an interface cannot implement another interface
* An interface can only extend one or more other interfaces

**Can an Interface extend another interface?**

* Yes, an interface can extend one or more interfaces

**What is a marker interface or tagged Interface?**

* An interface with no fields or methods is known as a marker interface
* A marker interface is used to indicate or provide essential information to [JVM](http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-jvm-419420.html) or compiler
* Alternatively, it is referred to as a tagged interface
* java.io.Serializable or java.lang.Cloneable is an example of a marker or tagged interface
* Marker interface improves readability in comparison with any other alternatives

**Can an interface contain another interface as a member?**

* Yes, an interface can contain another interface
* This is referred to as Nested interface

**What is Nested Interface in Java?**

* An interface declaration contained inside another class or interface is known as Nested interface
* During compilation, compiler inserts/add the static keyword to the nested interface

**DemoInterfaceA.java**

* **public** **interface** DemoInterfaceA **{**
* String NAME = "BenchResources.Net";
* // Nested interface inside another interface
* **interface** NextedItfExample **{**
* **}**

In this article, we will cover some of the interview questions with their justification on [**Java 8**](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html) interface. These are most frequently asked interview question from OOPS concepts

**Java 8 Interface Questions**

**What are the default methods in Java 8?**

* With the release Java 8, the new cool feature is added i.e.; if any new method needs to be added then provide the default implementation for this new method inside interface itself
* This new method will be prefixed with “default” keyword and known as a default method
* In Java 8, default methods are alternatively referred to as Virtual Extension methods or defender methods
* Example of the default method

**InterfaceInJava8.java**

* // only public &amp; abstract are permitted
* **public** **interface** InterfaceInJava8
* **{**
* // old abstract method
* **void** **displayAbstractMethod()**; // by default, this is public and abstract
* // default method with concrete implementation from Java 8
* **default** **void** **displayDefaultMethod()**
* **{**
* System.out.println**(**"InterfaceInJava8 : default method impl inside interface"**)**;
* **}**
* **}**

**Is it ok to define default methods in Java 8 without the “default” keyword inside the interface?**

* No, for defining the default method inside interface “default” keyword is must and it should prefix method declaration
* Without prefixing default keyword results in a compilation error
* **Compile-time error:** Abstract methods do not specify a body
* **Reason:** without default keyword, compiler consider it as an abstract method and as said abstract methods don’t have a body

**Whether multiple inheritances possible i.e.; implementing 2 or more interfaces?**

* Java allows multiple inheritances through interfaces i.e.; a class can implement 2 or more interfaces
* **Post-Java 8 release,**
* With the introduction of default methods, the ambiguity problem might arise when both interfaces have the same method with exactly the same signature

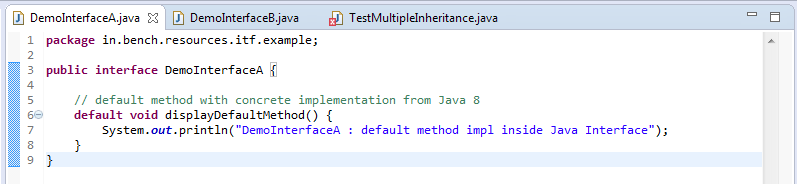
**What happens in multiple inheritances from Java 8?**

* In Java 8, a class can implement 2 or more interfaces and this might arise ambiguity problem with the introduction default method inside interface
* Ambiguity problem arises because; both interfaces can have the same method with exactly the same signature

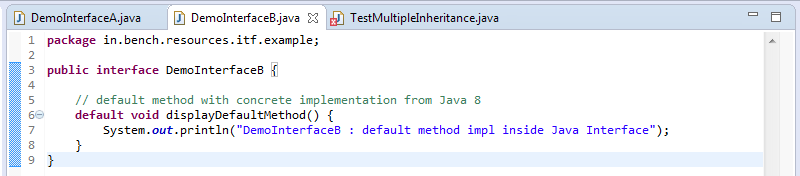
**What happens if a class implements 2 interfaces having exactly the same “*default*” method with the same signature?**

* Results in ambiguity problem with a compiler throwing error
* There are 2 interfaces having the same default methods and a class implements both these interfaces and results in ambiguity problem

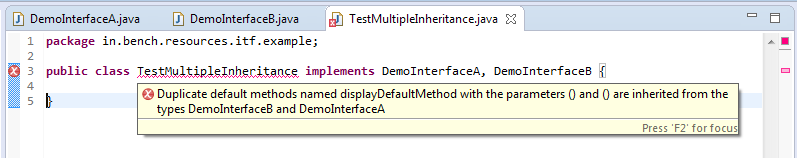
**DemoInterfaceA.java**



**DemoInterfaceB.java**

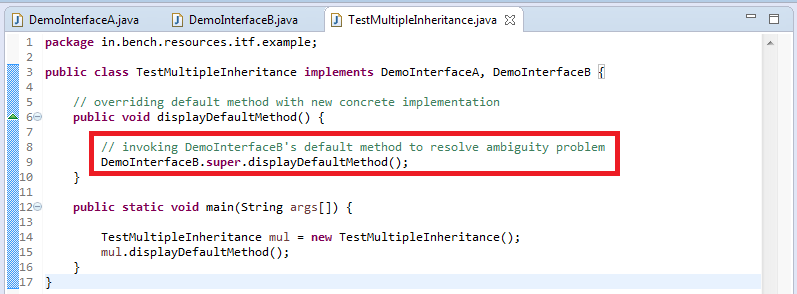


**TestMultipleInheritance.java**



* **Compile-time error:** Duplicate default methods named displayDefaultMethod with the parameters () and () are inherited from the types DemoInterfaceB and DemoInterfaceA

**How can we resolve the ambiguity problem in Java 8 while implementing multiple Interfaces?**

* To resolve the ambiguity problem in Java 8, override the conflicting method
* Now, if we want to invoke the default method from any of the interfaces then call using super keyword
* For example, <interfaceName>.super.<defaultMethodName>******

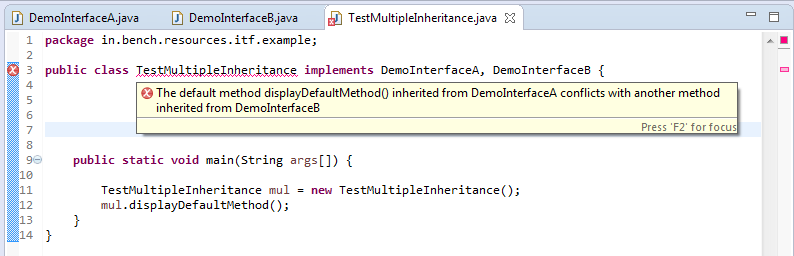
**How to invoke one of the interface’s default methods while implementing two interfaces?**

* Using super keyword
* **Syntax:** <interfaceName>.super.<defaultMethodName>

**How to come overcome, multiple inheritance problems in Java 8?**

* Override the default method in the implementation class
* Altogether provide new implementation or
* Invoke either one of the default method using super keyword
* For example, <interfaceName>.super.<defaultMethodName>

**What happens if a class implements 2 interfaces having the exact same method with the same signature (consider one as default and another abstract)?**

* Compilation fails with an error saying the conflicting method
* **Compile-time error:** The default method displayDefaultMethod() inherited from DemoInterfaceA conflicts with another method inherited from DemoInterfaceB
* To overcome this error, override this conflicting method and provide new implementation or invoke default method’s implementation using super keyword

**Can we declare the static method inside the interface?  
Answer:** Yes, we can declare starting from Java 8

**Is it ok, to define a static method inside the interface?**

**Till the Java 7 version,**

* Defining any concrete method (method with the body) inside interface will throw compilation error, even if it’s a static method

**Post-Java 8 release,**

* Static methods are allowed to define inside interface
* This is a new feature added in Java 8 which will act as a helper method

**How can we access static methods inside Interface?**

* Using interface name
* **Syntax:** <interfaceName>.<variableName>

**What is the interface in Java and what are its uses?**  
**Ans:** The interface in Java is similar to a class, which may contain a method’s signature only but not bodies and it is a formal set of methods and constant declarations that must be defined by the class that implements it. Interfaces are useful for:

* Declaring methods that one or more classes are expected to implement.
* Capturing similarities between unrelated classes without forcing a class relationship.
* Determining an object’s programming interface without revealing the actual body of the class.

Class C implements Interface ‘I’ containing method m1 and m2 declarations. Class C has provided the

**implementation for method m2. Can an object of Class C be created?**  
**Ans:** No. Class C should provide the implementation for all the methods in the interface ‘I’. Since Class C didn’t provide the implementation for the m1 method, it has to be declared as abstract. Abstract classes can’t be instantiated.

**Can a method inside an interface be declared as final?**  
**Ans:** No. Doing so will result in compilation error. “public” and “abstract” are the only applicable modifiers for the method declaration in an interface.

**Can an interface implement another interface?**  
**Ans:** Interfaces don’t provide implementation hence an interface cannot implement another interface.

**Can an Interface extend another interface?**  
**Ans:** Yes. An Interface can inherit another Interface, for that matter, an Interface can extend more than one Interface.

**Can a class extend more than one class?**  
**Ans:** No. A class can extend only one class but can implement any number of interfaces.

**Can an interface be final?**  
**Ans:** No. Doing so will result in a compilation error.

**Can a class be defined inside an interface?**  
**Ans:** Yes. It’s possible.

**Can an interface be defined inside a class?**  
**Ans:** Yes. It’s possible.

**What is a Marker Interface?**  
**Ans:** An interface which doesn’t have any declaration inside, but still enforces a mechanism is known as Marker Interface.

**Can we define private and protected modifiers for variables in interfaces?**  
**Ans:** No. We cannot define private and protected modifiers for variables in interfaces

**What is Externalizable?**  
**Ans:** Externalizable is an interface that extends Serializable Interface and sends data into Streams in compressed format. It has two methods, writeExternal(ObjectOuput out) and readExternal(ObjectInput in).

**What modifiers are allowed for methods in an interface?**  
**Ans:** Only “public” and “abstract” modifiers are allowed for the methods in the interfaces.

**When can an object reference be cast to an interface reference?**  
**Ans:** When the object implements the referenced interface, an object reference is cast to an interface reference.

**What is the difference between an Interface and an Abstract class?**  
**Ans:** An abstract class can have instance methods that implement a default behavior. An Interface can only declare constants and instance methods, but cannot implement default behavior and all methods are implicitly abstract.  
An interface has all public members and no implementation. An abstract class is a class which may have the usual flavors of class members (private, protected, etc.), but has some abstract methods.

**Can an anonymous class be declared as implementing an interface and extending a class?**  
**Ans:** An anonymous class may implement an interface or extend a superclass, but may not be declared to do both.

**What is an abstract class?**  
**Ans:** An abstract class is a class designed with implementation gaps for subclasses to fill in and is deliberately incomplete.

1. **What is an interface?**

An interface is contract which specifies the behavior that a class must implement. The purpose of an interface is to provide multiple implementations to the class. An interface in Java 8 can have **abstract** methods, **static** methods and **default** methods.  
  
**Know more about interfaces**

* + An interface can have abstract, [default](https://www.shristitechlabs.com/java/interviewquestions/top-10-interview-questions-in-interfaces/#default)and [static](https://www.shristitechlabs.com/java/interviewquestions/top-10-interview-questions-in-interfaces/#static) methods.
  + The variables in an interface are **public,static, final**
  + A class can implement multiple interfaces – to get multiple implementations. Refer [question-2](https://www.shristitechlabs.com/java/interviewquestions/top-10-interview-questions-in-interfaces/#single)
  + Multiple classes can implement an interface to be grouped under a single category. Refer [question-3](https://www.shristitechlabs.com/java/interviewquestions/top-10-interview-questions-in-interfaces/#multiple)
  + An interface with one abstract method is called **Functional**Interface. Refer [question-4](https://www.shristitechlabs.com/java/interviewquestions/top-10-interview-questions-in-interfaces/#functional)
  + An interface without methods is called as **Tag/marker**interface. Refer [question-5](https://www.shristitechlabs.com/java/interviewquestions/top-10-interview-questions-in-interfaces/#marker)
  + Interfaces can be extended. Refer [question-7](https://www.shristitechlabs.com/java/interviewquestions/top-10-interview-questions-in-interfaces/#extend)
  + When a class implements an interface, all the methods in the interface should be implemented in the class. If any of the methods are not implemented, make the class abstract, extend it and implement the method in the subclass

**Example**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public interface BonusCalculator {  void calculateBonus(int amount);          void checkDesignation(String desg);          // for default methods use default keyword          default String showDetails(){             return "Calculating Bonus";           }          static void admin(){            System.out.print("Admin Department")          }  } |

Let us see an example for interface and how to call the methods. We will use the above given interface **BonusCalculator**. Now, we will create a class **Manager** that implements the interface. Create an interface reference and point to the Manager object. Then call the methods using the interface reference. You can call the static method only using the interface name.(You cannot call using the interface reference also).  
  
**Example**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | public interface BonusCalculator {  void calculateBonus(int amount);          void checkDesignation(String desg);          default String showDetails(){             return "Calculating Bonus";           }          static void admin(){            System.out.print("Admin Department")          }  }  //class implementing BonusCalculator  public class Manager implements BonusCalculator{    @Override  public void calcBonus(int amount) {  System.out.println("bonus "+amount\*2);  }         @Override  public void checkDesignation(String desg) {  System.out.println("Bonus for "+desg);                 //lines of code  }  }  // Main class  public class InterDemo{       public static void main(String[] args) {    BonusCalculator bs = new Manager();  bs.calcBonus(100);          bs.checkDesignation("Manager");  bs.greetMessage(); // calling default method  BonusCalculator.admin(); // calling the static method  }  } |

1. **What is the use of a class implementing multiple interfaces?**

The reason for a class implementing multiple interfaces is to get multiple functionality.  
**eg.class Employee implements BonusCalculator, IncCalculator{ }**  
Now, Employee class gets the functionality of two interfaces.

1. **Why should multiple classes implement an interface?**

Multiple classes can implement an interface – to be grouped under a category. Lets say we have two classes **Employee** and **Student**. Both the classes implement **Serializable**. It means the object of these classes can be serialized. So, they become Serializable classes(grouping different classes into one category)  
**eg.class Employee implements Serializable{ } class Student implements Serializable{}**

[Become an expert Java Developer](https://academy.shristitechlabs.com/p/java-programming-become-a-java-developer)

1. **What is a Functional Interface?**

A [functional interface](https://www.shristitechlabs.com/java/java8/what-is-a-functional-interface-in-java-8/) is an interface with only one abstract method. But it can have default and static methods. You can also annotate this interface with **@FunctionalInterface**, though not mandatory. This annotation helps to check in the compile time if this is a functional interface.  
Examples: **java.lang.Comparable, java.lang.Runnable,**. Java 8 has introduced a lot of functional interfaces to work with Lambda expressions. Learn about lambda [here](https://www.shristitechlabs.com/java/using-lambda-expressions-with-functional-interface/) **Function, Predicate,Consumer, Supplier, BiPredicate, BiConsumer**are few examples of functional interfaces introduced in Java 8.

**Example**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | @FunctionalInterface  public interface IncCalculator {    void  calcBonus(int x);  default void greetMessage(){  System.out.println("welcome ");  }  } |

1. **What is a marker interface?**

A marker interface is an interface without methods. It is also called as Tag interface. The purpose of this marker interface is to group multiple classes under one category. **java.io.Serializable**is a marker interface. The classes that implement this interface Serializable can serialize their objects. Similarly **java.lang.Cloneable**is a marker interface. The classes that implement Cloneable can clone/duplicate their objects.

1. **What is the use of extending an interface?**

Whenever you want to add new abstract methods in an existing interface without breaking the contract, it is not possible. But you can extend the existing interface and add the new methods. Create a class that implements the sub interface to call these methods.

**Example**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | public interface Calculator {  int add(int a, int b);  int subtract(int a, int b);  int product(int a, int b);  double divide(int a, int b);  }    //adding new methods in sub interface  public interface Scientific extends Calculator{  int square(int a);  int cube(int a);  }    //class implementing the Calculator interface  public class BasicCalculator implements Calculator {    @Override  public int add(int a, int b) {  return (a+b);  }    @Override  public int subtract(int a, int b) {  return (a-b);  }    @Override  public int multiply(int a, int b) {  return (a\*b);  }    @Override  public double divide(int a, int b) {  return (a/b);  }  }    /\*class that implements the sub interface  the other methods are already defined in BasicCalculator  so extend BasicCalculator and define only the methods from sub interface \*/    public class AdvCalculator extends BasicCalculator implements Scientific {    @Override  public int square(int a) {  return a\*a;  }    @Override  public int cube(int a) {  return a\*a\*a;  }    } |

1. **What are default methods and what is the use of them?**

When you want to add new functionality to an existing interface without breaking the contract, then you can use default methods. For creating default methods, use **default** keyword and add definition for the method.  
For example, lets say, we have three classes **A, B, C** implementing an interface **Checker**. Now if we want to add abstract methods to the Checker interface, it is not possible. One way is to extend this interface as shown above or use default methods. Default methods also support backward compatibility.To learn about default methods in detail click [here](https://www.shristitechlabs.com/java/java8/what-is-the-use-of-default-methods-of-an-interface-in-java-8/).  
Let us modify the previous example using default methods.

**Example**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | public interface Calculator {  int add(int a, int b);  int subtract(int a, int b);  int product(int a, int b);  double divide(int a, int b);         //adding new methods as default methods  public default int square(int a) {  return a\*a;  }    public default int cube(int a) {  return a\*a\*a;  }    }    /\* class implementing the Calculator interface. The methods square and cube are     available to this class. So we dont need any sub class or sub interface \*/  public class BasicCalculator implements Calculator {    @Override  public int add(int a, int b) {  return (a+b);  }    @Override  public int subtract(int a, int b) {  return (a-b);  }    @Override  public int multiply(int a, int b) {  return (a\*b);  }    @Override  public double divide(int a, int b) {  return (a/b);  }  } |

1. **Can we override default methods?**

Yes. We can override default methods. To call the default method of the interface from within the overridden method of the implementing class, use **interfaceName.super.methodName()**

**Example**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public interface Checker{  default void greetMessage(){  System.out.println("welcome ");  }  }  class MyClass implements Checker{    public void greetMessage(){  System.out.println("welcome ");                 Checker.super.greetMessage(); // calling default method of Checker  }  } |

One other usecase scenario is, if a class(Employee) implements two interfaces having same default method names, then it will give a compiler error. In this case, you should override the default method in the implementing class  
**Example**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | public interface Checker1{  default void greetMessage(){  System.out.println("In Checker1");  }  }  public interface Checker2{  default void greetMessage(){  System.out.println("In Checker2 ");  }  }  class MyClass implements Checker1, Checker2{      // should override the default method    public void greetMessage(){  System.out.println("In implementing class ");            // to call default methods of the interface          Checker1.super.greetMessage(); // calling default method of Checker1          Checker2.super.greetMessage(); // calling default method of Checker2  }  } |

1. **What is the use of static methods?**

**Static** methods in an interface are similar to default methods. When you want to add new functionality to an existing interface without breaking the contract, then you can use static methods. But it cannot be overridden.

1. **What is the difference between abstract and interface? When to use what?**

**Abstract** classes can have both abstract and concrete methods. A class can extend only one abstract class.  
**Interfaces**in java 8 can have abstract, default and static methods. A class can implement multiple interfaces.  
  
**When to use abstract classes?**  
In a java application having multiple classes, if the class needs its own properties(instance variables), then use abstract class, so that the subclasses can use the properties. For Example, **Vehicle, Car, Bike** has properties as model, price, color. If there is IS-A relationship between classes, then use abstract classes. Car IS-A Vehicle.  
  
**When to use interfaces?**  
In a java application having multiple classes, if the class does not have IS-A relationship on other classes, but wants to use the functionality, then use interfaces.The classes can implement many interfaces to get multiple functionality.For Example, an interface **Insurance** can be used by **Car, Bike**. There is no IS-A relationship. This is complete abstraction.

What is an Interface?

An interface (YouTube video link - <https://www.youtube.com/watch?v=VangB-sVNgg>) defines a contract for responsibilities (methods) of a class.

How do you define an Interface?

An interface is declared by using the keyword interface. Look at the example below: Flyable is an interface.

//public abstract are not necessary

public abstract interface Flyable {

//public abstract are not necessary

public abstract void fly();

}

How do you implement an interface?

We can define a class implementing the interface by using the implements keyword. Let us look at a couple of examples:

Example 1

Class Aeroplane implements Flyable and implements the abstract method fly().

public class Aeroplane implements Flyable{

@Override

public void fly() {

System.out.println("Aeroplane is flying");

}

}

Example 2

public class Bird implements Flyable{

@Override

public void fly() {

System.out.println("Bird is flying");

}

}

Can you tell a little bit more about interfaces?

Variables in an interface are always public, static, final. Variables in an interface cannot be declared private.

interface ExampleInterface1 {

//By default - public static final. No other modifier allowed

//value1,value2,value3,value4 all are - public static final

int value1 = 10;

public int value2 = 15;

public static int value3 = 20;

public static final int value4 = 25;

//private int value5 = 10;//COMPILER ERROR

}

Interface methods are by default public and abstract. A concrete method (fully defined method) cannot be created in an interface. Consider the example below:

interface ExampleInterface1 {

//By default - public abstract. No other modifier allowed

void method1();//method1 is public and abstract

//private void method6();//COMPILER ERROR!

/\*//Interface cannot have body (definition) of a method

//This method, uncommented, gives COMPILER ERROR!

void method5() {

System.out.println("Method5");

}

\*/

}

Can you extend an interface?

An interface can extend another interface. Consider the example below:

interface SubInterface1 extends ExampleInterface1{

void method3();

}

Class implementing SubInterface1 should implement both methods - method3 and method1(from ExampleInterface1)

An interface cannot extend a class.

/\* //COMPILE ERROR IF UnCommented

//Interface cannot extend a Class

interface SubInterface2 extends Integer{

void method3();

}

\*/

Can a class implement multiple interfaces?

A class can implement multiple interfaces. It should implement all the method declared in all Interfaces being implemented.

interface ExampleInterface2 {

void method2();

}

class SampleImpl implements ExampleInterface1,ExampleInterface2{

/\* A class should implement all the methods in an interface.

If either of method1 or method2 is commented, it would

result in compilation error.

\*/

public void method2() {

System.out.println("Sample Implementation for Method2");

}

public void method1() {

System.out.println("Sample Implementation for Method1");

}

}