**New Interface concept:**

Interfaces in java8 can define methods. Suppose your project was built using java7 and thousands of your classes have used some predefined Interfaces eg. List inteface . If java8 has added some more methods in the interface then all the call implementing the interface must also implement those methods. Hence the concept of defining the methods inside the interface comes into the picture.

Java8 has added forEach() method, which is a default method in Iterable interaface. As List extends Collection and Collection extends Iterable, so list can have access to this method.

**#NOTE: default methods as well as static methods are allowed in java8.**

**public class** Sample **implements** A {  
  
  
 @Override  
 **public void** add() {  
 System.***out***.println(**"Add method"**);  
 }  
  
  
  
 **public static void** main(String[] args) {  
 Sample s = **new** Sample();  
 s.show();  
 s.add();  
 }  
}  
  
**interface** A  
{  
 **void** add();  
 **default void** show()  
 {  
 System.***out***.println(**"This is defined method of interface"**);  
 }  
  
}

**output:**

This is defined method of interface

Add method

**Ques. What if class implements two interfaces having same defined methods? Will this result to diamond problem because it will be like multiple inheritance which java already does not allow in classes having already defined method?**

**Ans.** If class implements two or more interfaces having same defined method then it will have to override the method and have to provide its own implementation. See the below eg.

**public class** Sample **implements** A,B {  
  
  
 @Override  
 **public void** add() {  
 System.***out***.println(**"Add method"**);  
 }  
  
 @Override  
 **public void** show() {  
 System.***out***.println(**"As there is same method in A, B so i have to override"**);  
 }  
  
  
 **public static void** main(String[] args) {  
 Sample s = **new** Sample();  
 s.show();  
 s.add();  
 }  
}  
  
**interface** A  
{  
 **void** add();  
 **default void** show()  
 {  
 System.***out***.println(**"This is defined method of interface"**);  
 }  
  
}  
  
  
**interface** B  
{  
 **void** add();  
 **default void** show()  
 {  
 System.***out***.println(**"This is defined method of interface"**);  
 }  
  
}

**output:**

As there is same method in A, B so i have to override

Add method

**Ques: What if a class extends another class and implementing an interface having same methods.?**

**Ans:** Then class method will run as it will hide the interface method as it has more priority over interface.

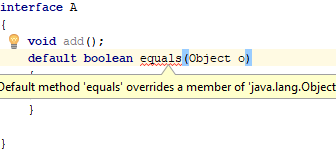
**public class** Sample **extends** C **implements** A {  
  
  
 @Override  
 **public void** add() {  
 System.***out***.println(**"Add method"**);  
 }  
  
 **public static void** main(String[] args) {  
 Sample s = **new** Sample();  
 s.show();  
 s.add();  
 }  
}  
  
**interface** A  
{  
 **void** add();  
 **default void** show()  
 {  
 System.***out***.println(**"This is defined method of interface"**);  
 }  
  
}  
  
**class** C  
{  
 **public void** show()  
 {  
 System.***out***.println(**"show method of class c"**);  
 }  
  
}

**output:**

show method of class c

Add method

**#NOTE : if you have any method that overrides object’s method then it will thriw an error. So an interface cannot override an object’s method.**

****

**Static method:**

**public class** Sample **implements** A {  
 **public static void** main(String[] args) {  
 A.*staticMethod*();  
 }  
}  
  
**interface** A {  
 **static void** staticMethod() {  
 System.***out***.println(**"This is a static method of interface"**);  
 }  
}

**foreach()**

**#NOTE => forEach() method is default method**

**public class** Sample {  
 **public static void** main(String[] args) {  
 List<Integer> list = Arrays.*asList*(1, 2, 3, 4, 5);  
  
 list.forEach(value -> System.***out***.println(value));  
 list.forEach(System.***out***::println);  
  
 }  
}

**lambda:**

Normally we do like below

**interface** A  
{  
 **void** show();  
}  
**public class** Sample {  
 **public static void** main(String[] args) {  
  
 A a = **new** A() {  
 @Override  
 **public void** show() {  
 System.***out***.println(**"Show method"**);  
 }  
 };  
  
 }  
}

Now, in the above we know that A in an interface having only one method which takes no parameter. So why we have to write this boiler plate code?

We can simply write as

A **a** = () -> System.***out***.println(**"Show method"**);

Hence that 5,6 lines code is converted into one line code. So java people enhanced the concept of interfaces that have only one method, instead of writing all the inner classes or different class to provide the implementation we can write simple codes.

A a = **new** A() {  
 @Override  
 **public void** show(**int** i) {  
 System.***out***.println(**"value of i "**+i);   
 }  
};

a.show(5);

**or**

A **a** = (**int** i) -> System.***out***.println((**"value of i "**+i);

In interface A it is mentioned that the data type of i is mentioned so why to mention again in above?

A **a** = i -> System.***out***.println((**"value of i "**+i);

a.show(5);

**Consumer<T>**

**class** ConsumerImpl **implements** Consumer<Integer> {  
 @Override  
 **public void** accept(Integer integer) {  
 System.***out***.println(integer);  
 }  
}  
  
**public class** Sample {  
 **public static void** main(String[] args) {  
  
  
 List<Integer> list = Arrays.*asList*(1, 2, 3, 4, 5, 6);  
  
 Consumer<Integer> consumer = **new** ConsumerImpl();  
 list.forEach(consumer);  
 }}

**public class** Sample {  
 **public static void** main(String[] args) {  
  
  
 List<Integer> list = Arrays.*asList*(1,2,3,4,5,6);  
  
 Consumer<Integer> consumer = **new** Consumer<Integer>() {  
 @Override  
 **public void** accept(Integer integer) {  
 System.***out***.println(integer);  
 }  
 };  
 list.forEach(consumer);  
 }}

So, in the above forEach will iterate the value of list one by one and pass it to the accept method of consumer and print it

**OR**

**public class** Sample {  
 **public static void** main(String[] args) {  
  
  
 List<Integer> list = Arrays.*asList*(1,2,3,4,5);  
  
 Consumer<Integer> consumer = i -> System.***out***.println(i);  
 list.forEach(consumer);  
 }}

**OR**

**public class** Sample {  
 **public static void** main(String[] args) {  
  
  
 List<Integer> list = Arrays.*asList*(1,2,3,4,5);  
  
   
 list.forEach(i -> System.***out***.println(i));  
 }  
**}**