Lambda Expression:

|  |
| --- |
| **What is difference between Lambda Expression and Anonymous class?** |
| |  | | --- | | **Answer:** | | The key difference between Anonymous class and Lambda expression is the usage of 'this' keyword. In the anonymous classes, ‘this’ keyword resolves to anonymous class itself, whereas for lambda expression ‘this’ keyword resolves to enclosing class where lambda expression is written.  Another difference between lambda expression and anonymous class is in the way these two are compiled. Java compiler compiles lambda expressions and convert them into private method of the class. It uses invokedynamic instruction that was added in Java 7 to bind this method dynamically. | |

# Java 8 Interface Changes – default method and static method

Prior to java 8, [interface in java](https://beginnersbook.com/2013/05/java-interface/) can only have abstract methods. All the methods of interfaces are public & abstract by default. Java 8 allows the interfaces to have default and static methods. The reason we have default methods in interfaces is to allow the developers to add new methods to the interfaces without affecting the classes that implements these interfaces.

# Why default method?

For example, if several classes such as A, B, C and D implements an interface XYZInterface then if we add a new method to the XYZInterface, we have to change the code in all the classes(A, B, C and D) that implements this interface. In this example we have only four classes that implements the interface which we want to change but imagine if there are hundreds of classes implementing an interface then it would be almost impossible to change the code in all those classes. This is why in java 8, we have a new concept “default methods”. These methods can be added to any existing interface and we do not need to implement these methods in the implementation classes mandatorily, thus we can add these default methods to existing interfaces without breaking the code.

We can say that concept of default method is introduced in java 8 to add the new methods in the existing interfaces in such a way so that they are backward compatible. Backward compatibility is adding new features without breaking the old code.

**Static methods** in interfaces are similar to the default methods except that we cannot override these methods in the classes that implements these interfaces.

## Java 8 Example: Default method in Interface

The method newMethod() in MyInterface is a default method, which means we need not to implement this method in the implementation class Example. This way we can add the default methods to existing interfaces without bothering about the classes that implements these interfaces.

interface MyInterface{

/\* This is a default method so we need not

\* to implement this method in the implementation

\* classes

\*/

default void newMethod(){

System.out.println("Newly added default method");

}

/\* Already existing public and abstract method

\* We must need to implement this method in

\* implementation classes.

\*/

void existingMethod(String str);

}

public class Example implements MyInterface{

// implementing abstract method

public void existingMethod(String str){

System.out.println("String is: "+str);

}

public static void main(String[] args) {

Example obj = new Example();

//calling the default method of interface

obj.newMethod();

//calling the abstract method of interface

obj.existingMethod("Java 8 is easy to learn");

}

}

Output:

Newly added default method

String is: Java 8 is easy to learn

## Java 8 Example: Static method in Interface

As mentioned above, the static methods in interface are similar to default method so we need not to implement them in the implementation classes. We can safely add them to the existing interfaces without changing the code in the implementation classes. Since these methods are static, we cannot override them in the implementation classes.

interface MyInterface{

/\* This is a default method so we need not to implement this method in the implementation classes \*/

default void newMethod(){

System.out.println("Newly added default method");

}

/\* This is a static method. Static method in interface is similar to default method except that we cannot override

\* them in the implementation classes. Similar to default methods, we need to implement these methods

\* in implementation classes so we can safely add them to the existing interfaces.

\*/

static void anotherNewMethod(){

System.out.println("Newly added static method");

}

/\* Already existing public and abstract method We must need to implement this method in implementation classes. \*/

void existingMethod(String str);

}

public class Example implements MyInterface{

// implementing abstract method

public void existingMethod(String str){

System.out.println("String is: "+str);

}

public static void main(String[] args) {

Example obj = new Example();

//calling the default method of interface

obj.newMethod();

//calling the static method of interface

MyInterface.anotherNewMethod();

//calling the abstract method of interface

obj.existingMethod("Java 8 is easy to learn");

}

}

Output:

Newly added default method

Newly added static method

String is: Java 8 is easy to learn

## Java 8 – Abstract classes vs interfaces

With the introduction of default methods in interfaces, it seems that the [abstract classes](https://beginnersbook.com/2013/05/java-abstract-class-method/) are same as interface in java 8. However this is not entirely true, even though we can now have concrete methods(methods with body) in interfaces just like abstract class, this doesn’t mean that they are same. There are still few differences between them, one of them is that abstract class can have constructor while in interfaces we can’t have constructors.

The purpose of interface is to provide full abstraction, while the purpose of abstract class is to provide partial abstraction. This still holds true. The interface is like a blueprint for your class, with the introduction of default methods you can simply say that we can add additional features in the interfaces without affecting the end user classes.

# Default Method and Multiple Inheritance

The [multiple inheritance](https://beginnersbook.com/2013/05/java-multiple-inheritance/) problem can occur, when we have two interfaces with the default methods of same signature. Lets take an example.

interface MyInterface{

default void newMethod(){

System.out.println("Newly added default method");

}

void existingMethod(String str);

}

interface MyInterface2{

default void newMethod(){

System.out.println("Newly added default method");

}

void disp(String str);

}

public class Example implements MyInterface, MyInterface2{

// implementing abstract methods

public void existingMethod(String str){

System.out.println("String is: "+str);

}

public void disp(String str){

System.out.println("String is: "+str);

}

public static void main(String[] args) {

Example obj = new Example();

//calling the default method of interface

obj.newMethod();

}

}

Output:

Error: Duplicate default methods named newMethod with the parameters () and () are inherited from the types MyInterface2 and MyInterface

This is because we have the same method in both the interface and the compiler is not sure which method to be invoked.

**How to solve this issue?**  
To solve this problem, we can implement this method in the implementation class like this:

interface MyInterface{

default void newMethod(){

System.out.println("Newly added default method");

}

void existingMethod(String str);

}

interface MyInterface2{

default void newMethod(){

System.out.println("Newly added default method");

}

void disp(String str);

}

public class Example implements MyInterface, MyInterface2{

// implementing abstract methods

public void existingMethod(String str){

System.out.println("String is: "+str);

}

public void disp(String str){

System.out.println("String is: "+str);

}

//Implementation of duplicate default method

public void newMethod(){

System.out.println("Implementation of default method");

}

public static void main(String[] args) {

Example obj = new Example();

//calling the default method of interface

obj.newMethod();

}

}

Output:

Implementation of default method