

How to use Lambda Expression in Place of Anonymous Class in Java 8

Before Java 8, Anonymous class was the only way you can implement functional idiom in Java. Since prior to Java 8 you cannot pass function to another function, you would have to wrap it into object, [as seen in Strategy Pattern](#).

Those are also known as function objects in Java.

Anonymous class was also handy to create a throw-away implementation of SAM (Single Abstract Methods)[interfaces](#) like Runnable, Callable,

CommandListener or ActionListener. Despite all these goodies and flexibility Anonymous class adds too much boiler plate code, making it [hard to](#) read and understand.

Now with Java 8, you have got the ability to pass function to another function in form of lambda [expression](#), you can easily replace your Anonymous class with lambda expression in Java. Even if you don't want to do it for your older project, you shouldn't use Anonymous class [anymore](#) for the purpose described above, instead you should learn lambda expression and use it in place of Anonymous class. To get the ball rolling, I will show *how you can use lambda expression instead of Anonymous class* to implement `ActionListener` in Java Swing code. Since [lambda expression is of SAM type in Java](#) and `ActionListener` only contains one method `actionPerformed(ActionEvent ae)` you can use lambdas to implement `ActionListener`, result is a much cleaner and concise code. By the way, you cannot always use lambda expression in place of Anonymous class, because of its limitation of being SAM type. If you are using anonymous class to implement an interface with two abstract methods then you cannot replace with lambda of Java 8.

Lambda Expression vs Anonymous Class in Java 8

Lambda expressions are very powerful and can replace many usage of anonymous class but not all. An anonymous class can be used to create a subclass of an abstract class, a concrete class and can be used to implement an interface in Java. It can even add state fields. An instance of anonymous class can also be refereed by using this keyword inside its methods, which means you can call other methods and state can be changed. Since the most common use of [Anonymous class](#) is to provide throwaway, stateless implementation of abstract class and interface with single function, those can be replaced by lambda expressions, but when you have a state field or implementing more than one interface, you cannot use lambdas to replace anonymous class. One more difference between lambda expression and anonymous class is that later introduces a new scope, where names are resolved from Anonymous class's parent class or interface and can shadow names that occur in the lexically enclosing environment, but for lambdas all names are resolved lexically. In short, lambda expression though a great feature has its limitation of being a SAM type. Which means it

can only replace anonymous class when former is used to implement single method, if your anonymous class is implementing more than one interface, Java 8 lambdas cannot help.

How to use Lambda Expression in Java

As I said anonymous class is very powerful but lambda is even better. Let's see a code example of how you can replace Anonymous class with lambda expression in Java 8. This is variation of classical Swing Helloworld program. Here I have a `JFrame` and a button, when you press button `ActionEvent` will be generated, which is handled by the event listener you have attached to button. In this example, we are not doing anything fancy but just printing log message in console to confirm that we have handled the event. What is important in this example is the way we have coded event handling code. First we are using Anonymous class to create implementation of `ActionListener` so that we can override `actionPerformed()` method, which took 5 lines of code, next to it is new Java 8 way of handling event using lambda expression, which just took 1 line of code. Imagine how many lines of code you can reduce in a real world big fat Swing application full of buttons and other GUI components. Lambda is much more readable than anonymous class. You can even use this pattern to implement other SAM interfaces e.g. `Runnable`, `Comparable`, `Comparator` and `Callable` in Java. If you are not sure how to do that, please check these [top 10 Java 8 tutorials](#).

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

/**
 * Java program to demonstrate how you can use lambda expression in place of
 * Anonymous Inner class to implement event handling logic in Swing application.
 * Any listener code is lot concise and more readable using lambda expression.
 */
public class Test extends JFrame {

    private final JButton button = new JButton("Start");

    public Test() {
        super("Java 8 Lambda Example");

        getContentPane().setLayout(new BorderLayout());
```

```

getContentPane().add(button);

button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent evt) {
        System.out.println("Classic way of handling event using Anonymous class");
    }
});

button.addActionListener(e -> System.out.println("Java 8 way"
    + " to handle event using Lambda expression"));

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setSize(400, 200);
}

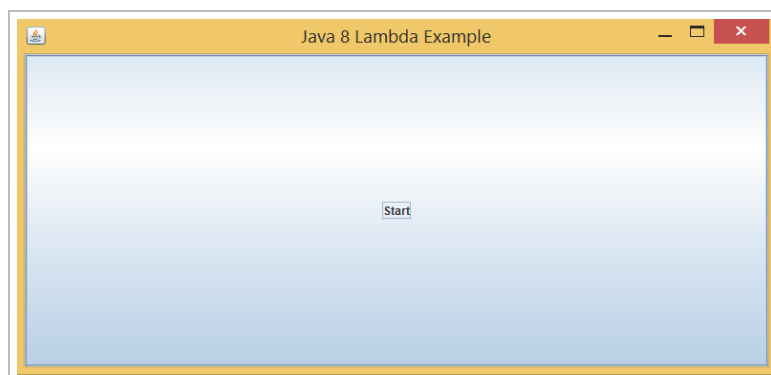
public static void main(String[] args) {
    // Lambda expression to implement Runnable interface
    SwingUtilities.invokeLater(() -> {
        new Test().setVisible(true);
    });
}
}

```

Output:

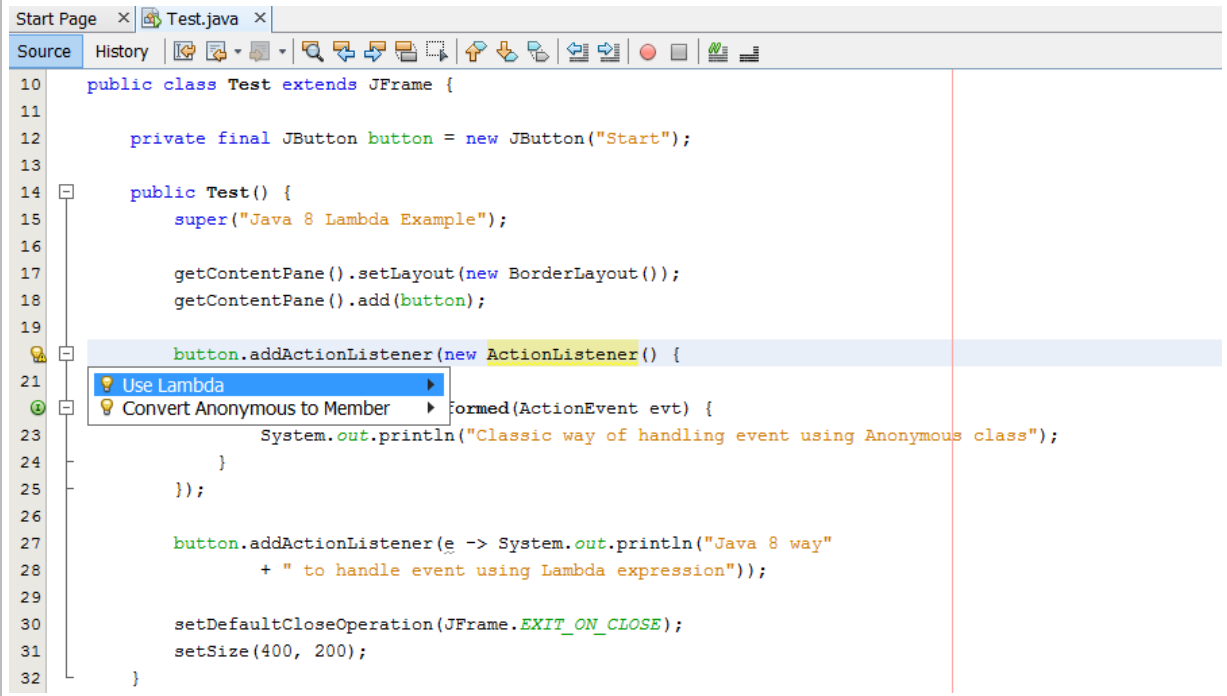
Java 8 way to handle event using Lambda expression
 Classic way of handling event using Anonymous class

When you run this program, you will see following window, created by JFrame. Button is spread all over because of BorderLayout but as soon as you click on button you will see above line printed on console.



Good thing about Java IDEs is that they are now [supporting](#) Java 8 in content assist and providing useful hints to convert anonymous class to lambda expression automatically. Here is the screenshot from Netbeans, where you can clearly see the hint about converting anonymous to lambda in Java

8.



```
10 public class Test extends JFrame {
11
12     private final JButton button = new JButton("Start");
13
14     public Test() {
15         super("Java 8 Lambda Example");
16
17         getContentPane().setLayout(new BorderLayout());
18         getContentPane().add(button);
19
20         button.addActionListener(new ActionListener() {
21             // Use Lambda
22             // Convert Anonymous to Member
23             actionPerformed(ActionEvent evt) {
24                 System.out.println("Classic way of handling event using Anonymous class");
25             }
26         });
27
28         button.addActionListener(e -> System.out.println("Java 8 way"
29             + " to handle event using Lambda expression"));
30
31         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
32         setSize(400, 200);
33     }
34 }
```

That's all on how to use Lambda expression in place of anonymous [class in Java 8](#). If you are coding in Java 8 make sure you use lambda expression instead of anonymous class for implementing Comparable, Comparator, Runnable, [Callable](#), CommandListener, ActionListener and several other interfaces in Java, which got just one single method. You should also remember that because of bing SAM type (Single Abstract Method), you cannot use lambda expression to implement method with more than one abstract method. In short, doesn't matter how powerful Lambda is it cannot replace anonymous class fully, Anonymous class is here to stay.