

Lesson 04 Demo 02

Implementing Anonymous Classes

Objective: To demonstrate the implementation of anonymous classes

Tools required: Eclipse IDE

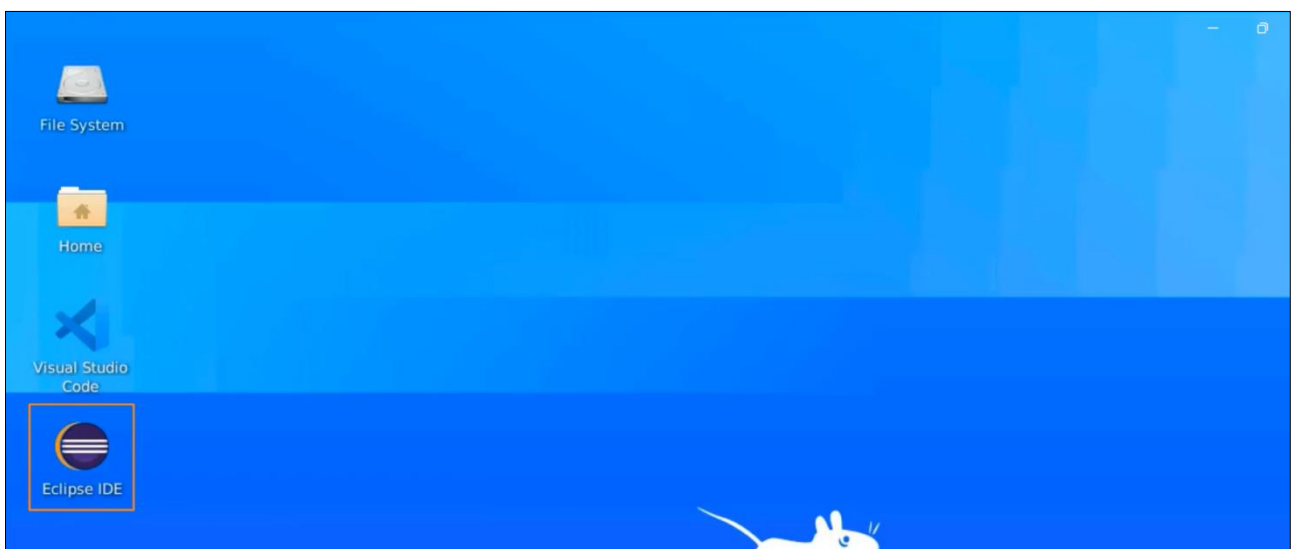
Prerequisites: None

Steps to be followed:

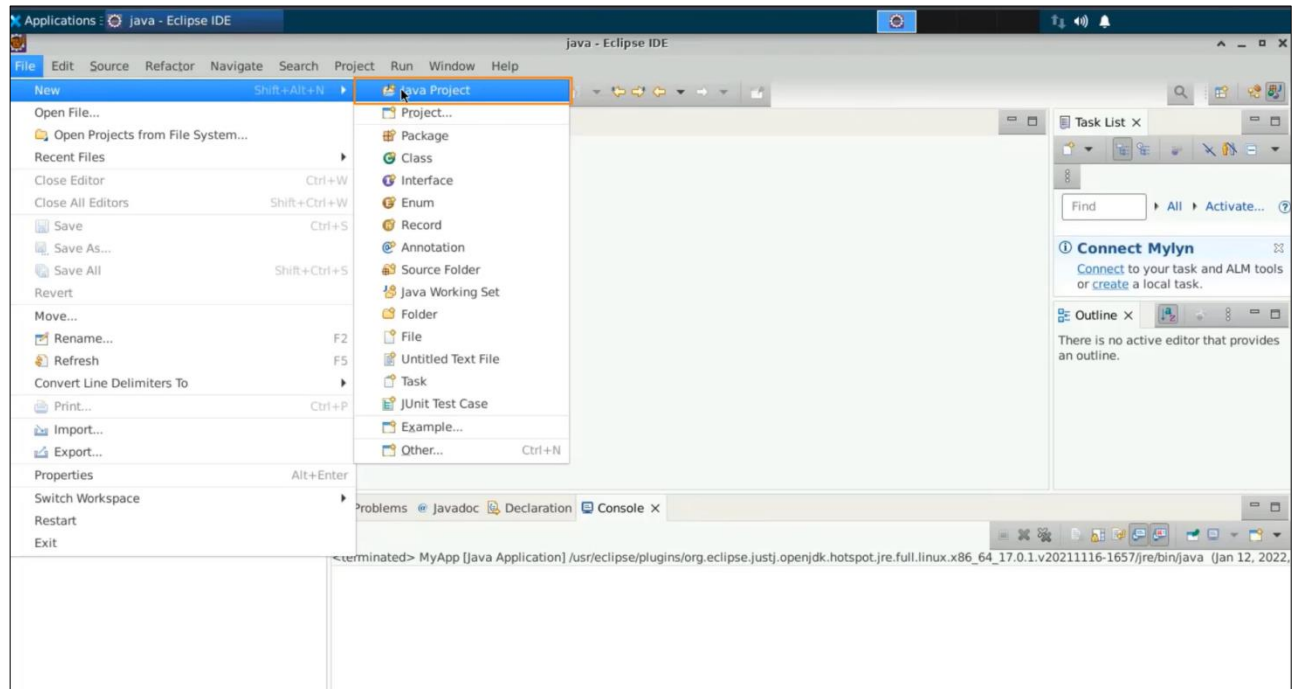
1. Open the IDE and create a new project
2. Write an interface and execute the code with example data
3. Execute the code and print the message accordingly
4. Override the methods called on success and failure
5. Use anonymous classes

Step 1: Open IDE and create a new project

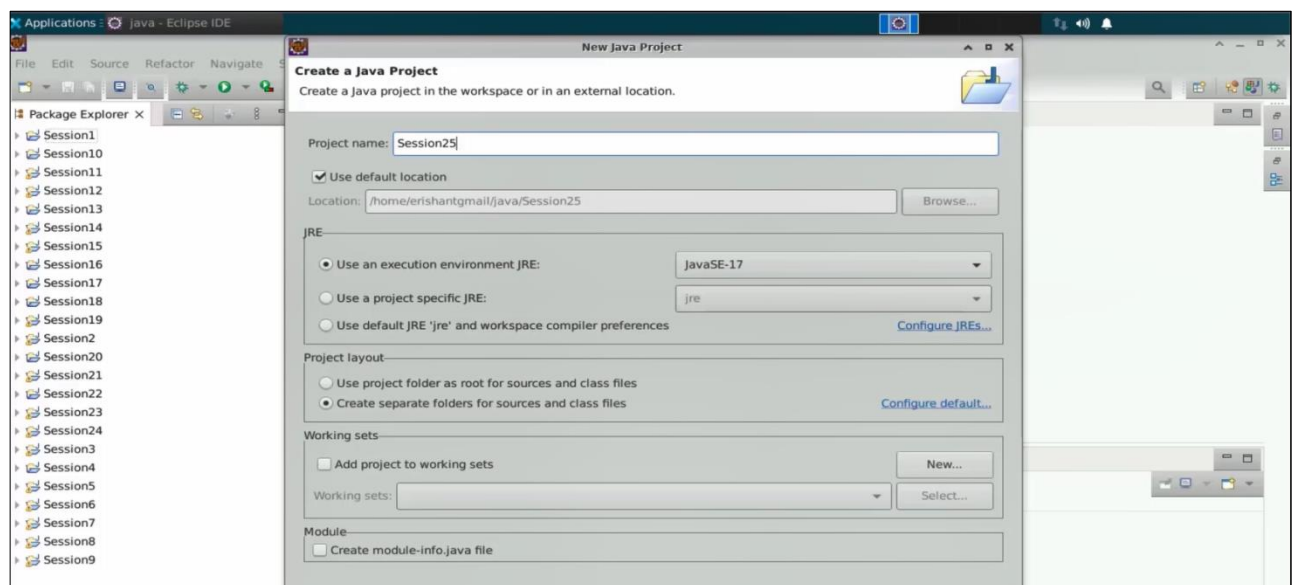
1.1 Open the Eclipse IDE



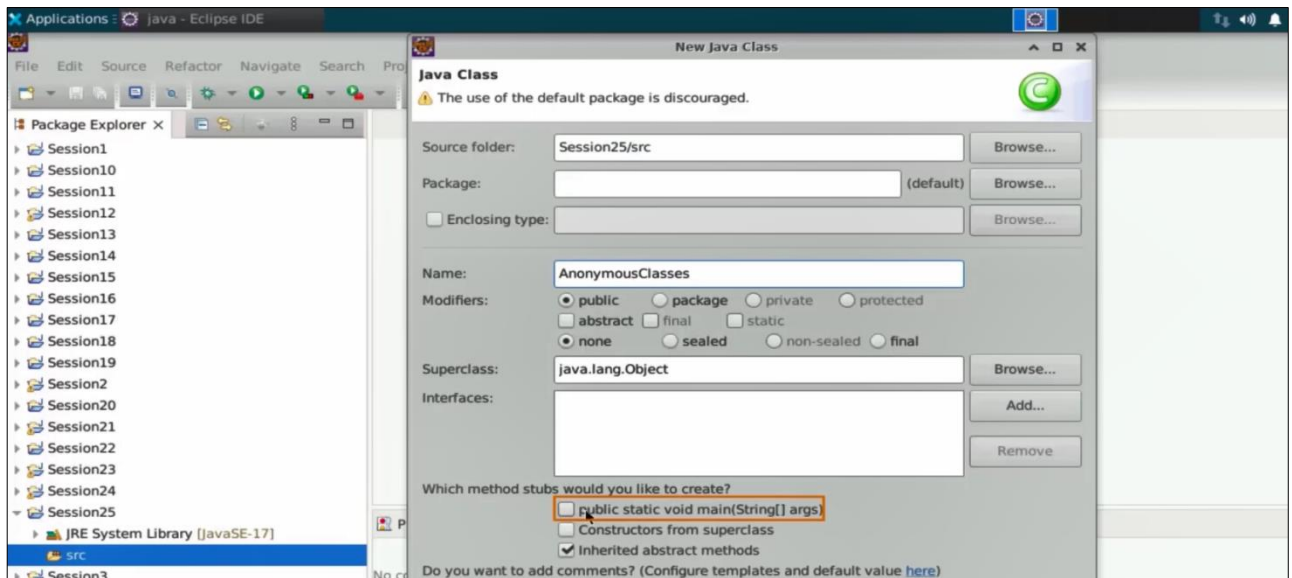
1.2 Select **File**, then **New**, and then **Java project**



1.3 Name the project "Session25", uncheck "Create a module-info.java file", and press Finish.

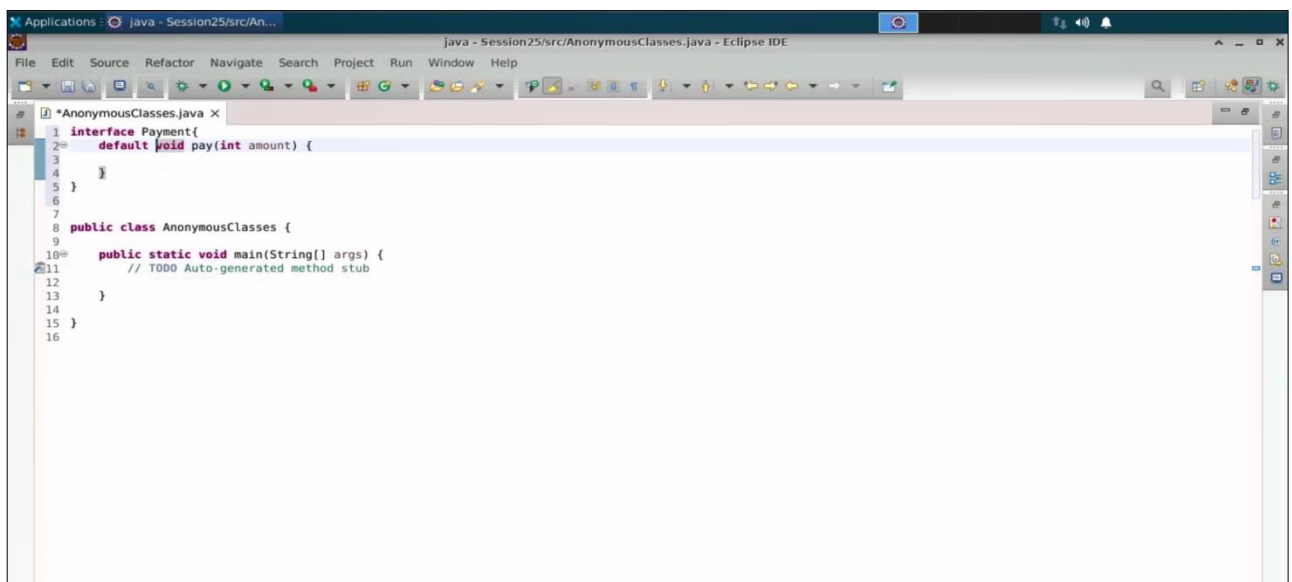


1.4 With a **Session25** on the src, do a right-click and create a **new class**. Name this class as an **AnonymousClasses**, then select the **main method**, and then select **finish**.

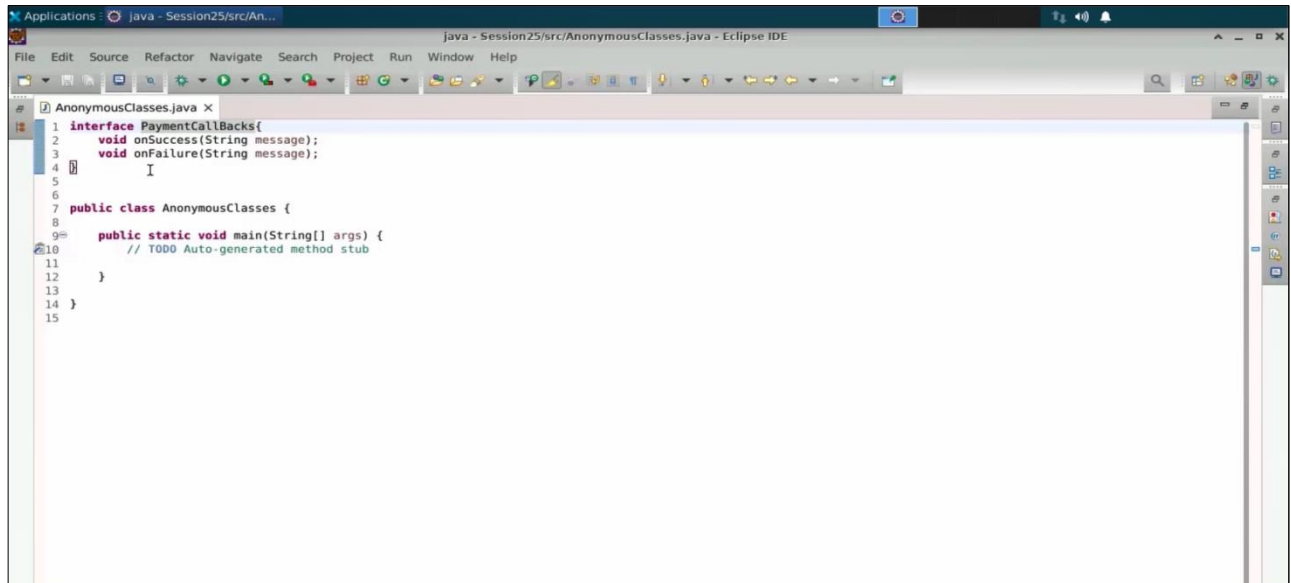


Step 2: Write an interface and execute the code with example data

2.1 Let us write one of the interfaces, which goes interface as the payment. Then define a method called pay an amount. And mark this method as default, since you want the definition.



- 2.2 You can make it simpler, with the method called void on success. And a method called void on failure. Hence, there are two methods for the payment interface. Let us select the input as the message for both and these are like payment callbacks.

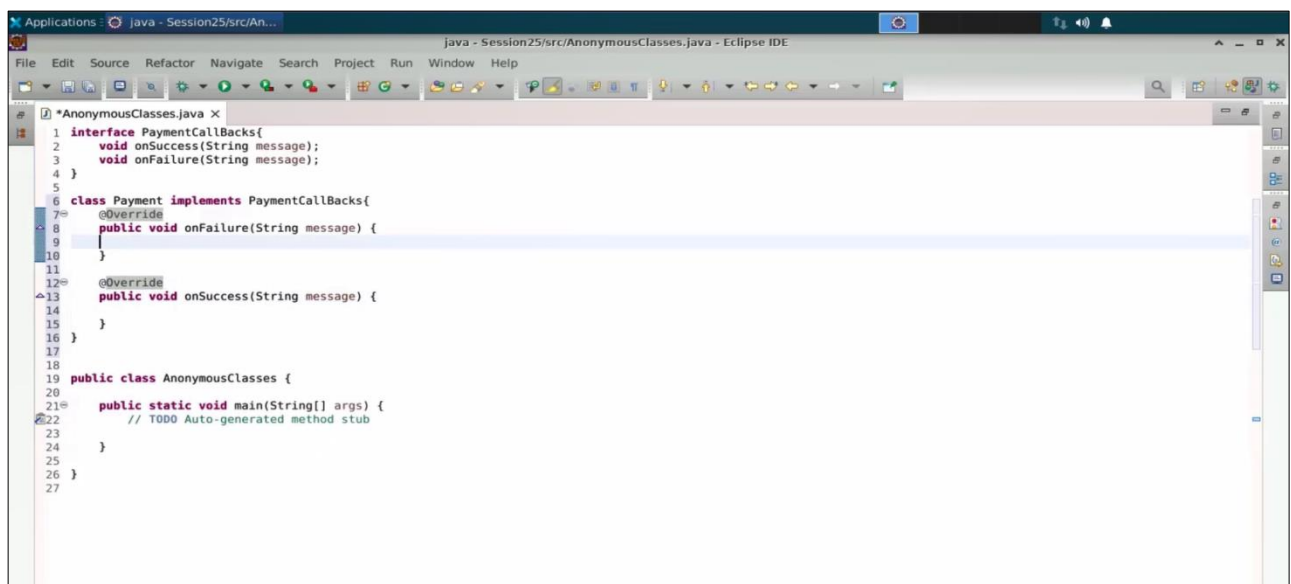


```

1 interface PaymentCallbacks{
2     void onSuccess(String message);
3     void onFailure(String message);
4 }
5
6
7 public class AnonymousClasses {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11     }
12 }
13
14
15

```

- 2.3 Now, for the class called `Payment`, you can implement the payment callbacks. When you implement the callbacks, you need to override the `onFailure` and `onSuccess` methods.



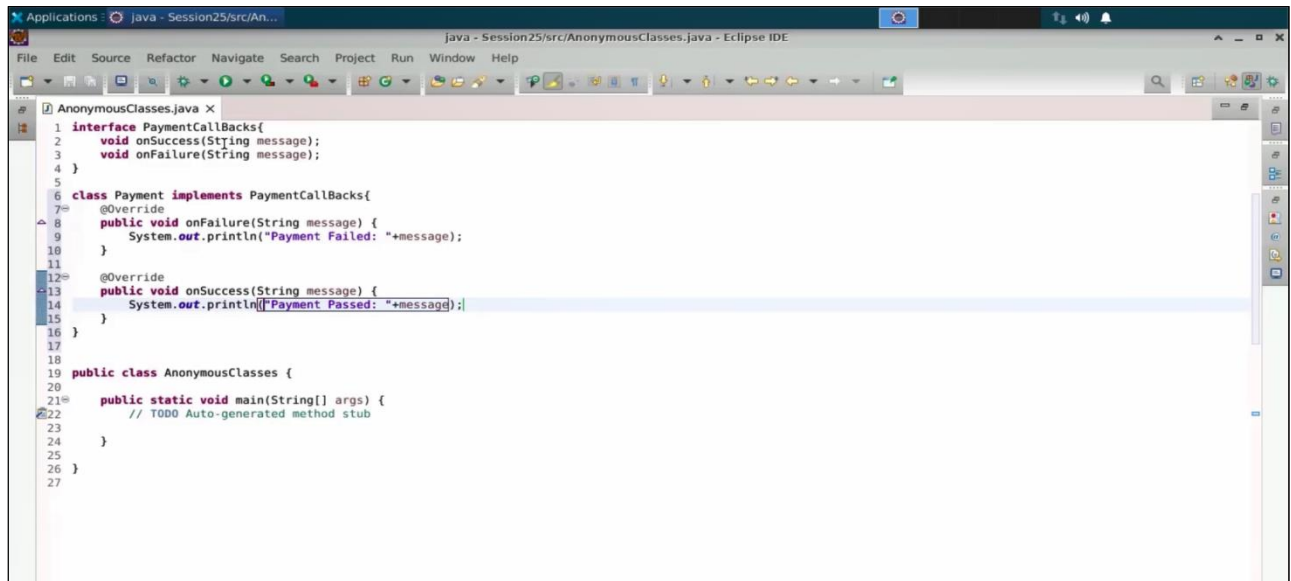
```

1 interface PaymentCallbacks{
2     void onSuccess(String message);
3     void onFailure(String message);
4 }
5
6 class Payment implements PaymentCallbacks{
7     @Override
8     public void onFailure(String message) {
9     }
10
11     @Override
12     public void onSuccess(String message) {
13     }
14 }
15
16
17
18 public class AnonymousClasses {
19
20     public static void main(String[] args) {
21         // TODO Auto-generated method stub
22     }
23 }
24
25
26
27

```

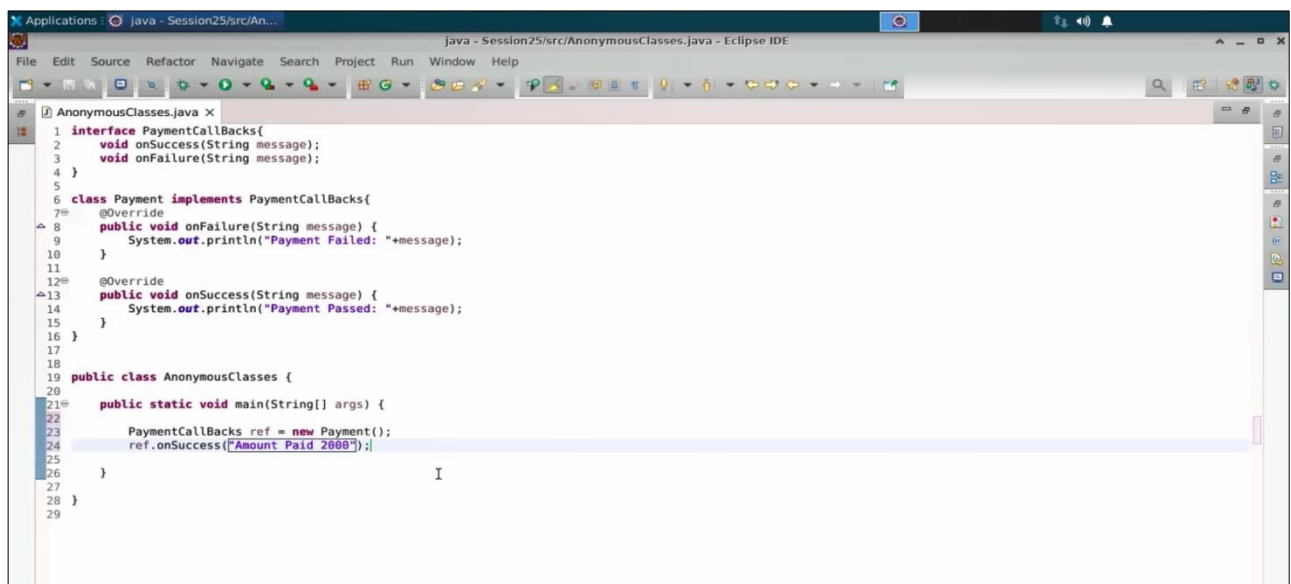
Step 3: Execute the code and print the message accordingly

- 3.1 Let us print down payment failed along with the message. Next, write as payment passed and along with it, also print the message.



```
1 interface PaymentCallbacks{
2     void onSuccess(String message);
3     void onFailure(String message);
4 }
5
6 class Payment implements PaymentCallbacks{
7     @Override
8     public void onFailure(String message) {
9         System.out.println("Payment Failed: "+message);
10    }
11
12    @Override
13    public void onSuccess(String message) {
14        System.out.println("Payment Passed: "+message);
15    }
16 }
17
18 public class AnonymousClasses {
19
20     public static void main(String[] args) {
21         // TODO Auto-generated method stub
22     }
23 }
24
25
26
27
```

- 3.2 For the polymorphic statement, the payment callbacks can have a reference variable that can refer to the object of payment. And here you can use the reference variable to execute the method called on success and let us write the amount paid as 2000.



```
1 interface PaymentCallbacks{
2     void onSuccess(String message);
3     void onFailure(String message);
4 }
5
6 class Payment implements PaymentCallbacks{
7     @Override
8     public void onFailure(String message) {
9         System.out.println("Payment Failed: "+message);
10    }
11
12    @Override
13    public void onSuccess(String message) {
14        System.out.println("Payment Passed: "+message);
15    }
16 }
17
18 public class AnonymousClasses {
19
20     public static void main(String[] args) {
21
22         PaymentCallbacks ref = new Payment();
23         ref.onSuccess("Amount Paid 2000");
24     }
25 }
26
27
28
29
```

3.3 Hence, when you run the code, it shows payment passed and amount as 2000 like a message.

```

1 interface PaymentCallbacks{
2     void onSuccess(String message);
3     void onFailure(String message);
4 }
5
6 class Payment implements PaymentCallbacks{
7     @Override
8     public void onFailure(String message) {
9         System.out.println("Payment Failed: "+message);
10    }
11
12    @Override
13    public void onSuccess(String message) {
14        System.out.println("Payment Passed: "+message);
15    }
16 }
17
18 public class AnonymousClasses {
19
20     public static void main(String[] args) {
21
22         PaymentCallbacks ref = new Payment();
23         ref.onSuccess("Amount Paid 2000");
24     }
25 }
26
27
28
29

```

```

<terminated> AnonymousClasses [Java Application] /usr/eclipse/plugins/org.eclipse.justj.o
Payment Passed: Amount Paid 2000

```

3.4 Similarly, you can write `reference.onFailure` with the message "Amount 2000 not processed." Thus, you have executed both methods. You only need a single object of `Payment`. This is how the use case goes.

```

1 interface PaymentCallbacks{
2     void onSuccess(String message);
3     void onFailure(String message);
4 }
5
6 class Payment implements PaymentCallbacks{
7     @Override
8     public void onFailure(String message) {
9         System.out.println("Payment Failed: "+message);
10    }
11
12    @Override
13    public void onSuccess(String message) {
14        System.out.println("Payment Passed: "+message);
15    }
16 }
17
18 public class AnonymousClasses {
19
20     public static void main(String[] args) {
21
22         PaymentCallbacks ref = new Payment();
23         ref.onSuccess("Amount Paid 2000");
24         ref.onFailure("Amount 2000 not processed");
25     }
26 }
27
28
29

```

```

<terminated> AnonymousClasses [Java Application] /usr/eclipse/plugins/org.eclipse.justj.o
Payment Passed: Amount Paid 2000
Payment Failed: Amount 2000 not processed

```

3.5 If there is only one single object of payment and you do not need multiple objects of payment. In such kind of situation, you can eliminate, this entire class. Then you can come here and give as the interface payment callbacks create a reference variable as a new payment callback. This is like you are creating an object.

```

1 interface PaymentCallBacks{
2     void onSuccess(String message);
3     void onFailure(String message);
4 }
5
6 /*class Payment implements PaymentCallBacks{
7     @Override
8     public void onFailure(String message) {
9         System.out.println("Payment Failed: "+message);
10    }
11
12    @Override
13    public void onSuccess(String message) {
14        System.out.println("Payment Passed: "+message);
15    }
16 }*/
17
18
19 public class AnonymousClasses {
20
21     public static void main(String[] args) {
22
23         //PaymentCallBacks ref = new Payment();
24         //ref.onSuccess("Amount Paid 2000");
25         //ref.onFailure("Amount 2000 not processed");
26
27         PaymentCallBacks ref = new PaymentCallBacks();
28     }
29
30 }
31

```

3.6 With this kind of syntax, you can use these two brackets and come down here. This is one of the syntaxes where you are trying to create an anonymous class. If you notice here, you are getting an error message that says that on success and on failure are not implemented.

```

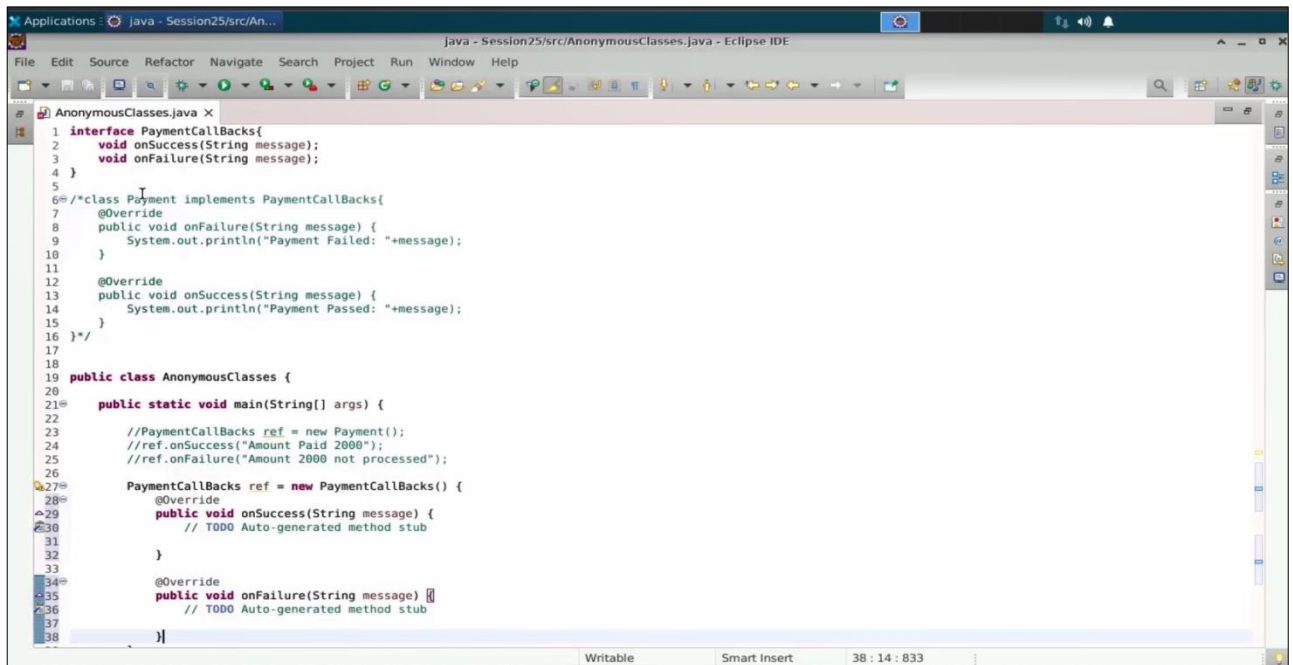
1 interface PaymentCallBacks{
2     void onSuccess(String message);
3     void onFailure(String message);
4 }
5
6 /*class Payment implements PaymentCallBacks{
7     @Override
8     public void onFailure(String message) {
9         System.out.println("Payment Failed: "+message);
10    }
11
12    @Override
13    public void onSuccess(String message) {
14        System.out.println("Payment Passed: "+message);
15    }
16 }*/
17
18
19 public class AnonymousClasses {
20
21     public static void main(String[] args) {
22
23         //PaymentCallBacks ref = new Payment();
24         //ref.onSuccess("Amount Paid 2000");
25         //ref.onFailure("Amount 2000 not processed");
26
27         PaymentCallBacks ref = new PaymentCallBacks();
28     }
29
30 }
31

```

Multiple markers at this line
- The type new PaymentCallBacks(){} must implement the inherited abstract method PaymentCallBacks.onSuccess(String)
- The type new PaymentCallBacks(){} must implement the inherited abstract method PaymentCallBacks.onFailure(String)

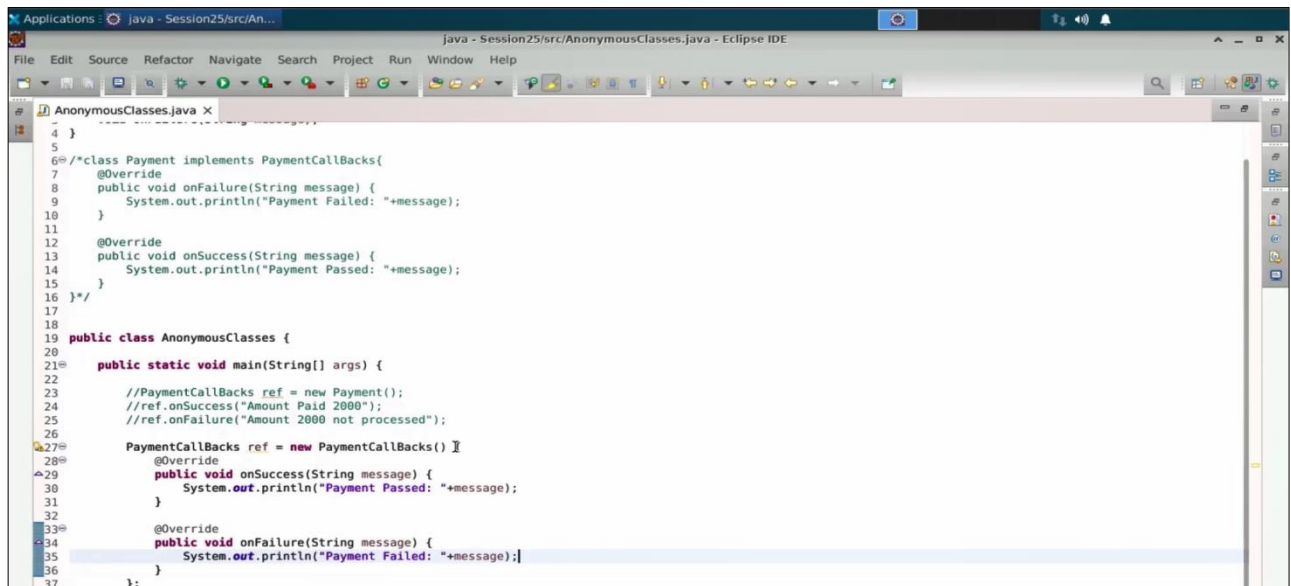
Step 4: Override the methods called on success and failure

- 4.1 Let us come here and override the method called on success. And then override another method called on failure. Hence, here you are with no more errors.



```
1 interface PaymentCallbacks{
2     void onSuccess(String message);
3     void onFailure(String message);
4 }
5
6 /*class Payment implements PaymentCallbacks{
7     @Override
8     public void onFailure(String message) {
9         System.out.println("Payment Failed: "+message);
10    }
11 }
12
13 @Override
14 public void onSuccess(String message) {
15     System.out.println("Payment Passed: "+message);
16 }
17 }*/
18
19 public class AnonymousClasses {
20
21     public static void main(String[] args) {
22
23         //PaymentCallbacks ref = new Payment();
24         //ref.onSuccess("Amount Paid 2000");
25         //ref.onFailure("Amount 2000 not processed");
26
27         PaymentCallbacks ref = new PaymentCallbacks() {
28             @Override
29             public void onSuccess(String message) {
30                 // TODO Auto-generated method stub
31             }
32
33             @Override
34             public void onFailure(String message) {
35                 // TODO Auto-generated method stub
36             }
37         }
38     }
39 }
```

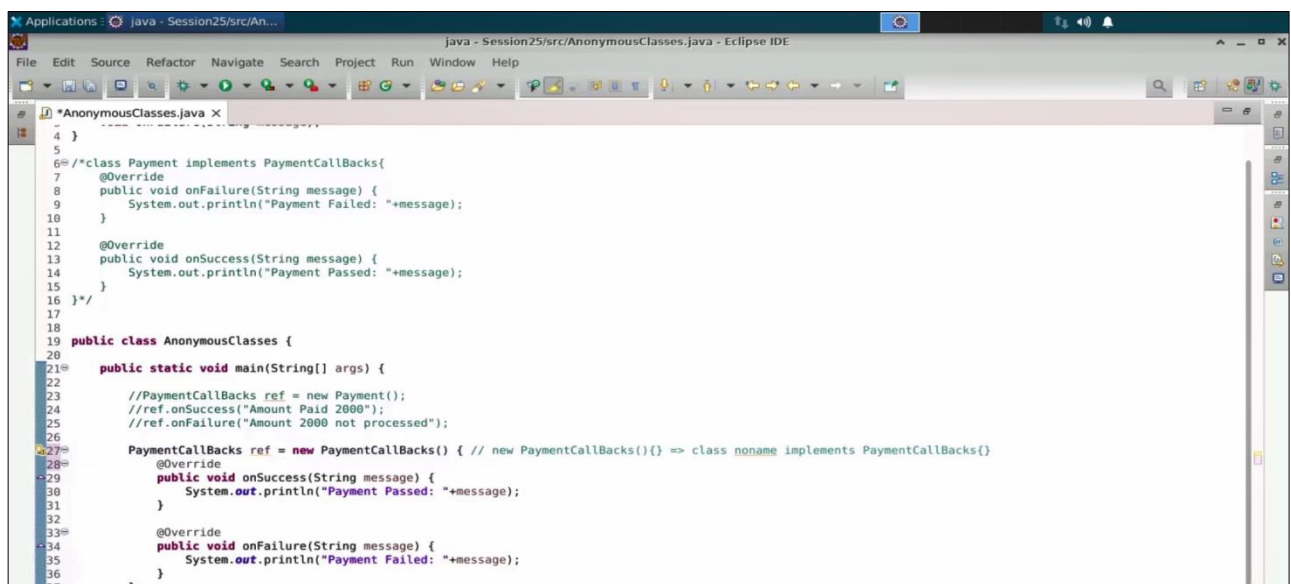

4.2 Come here and write as the payment failed, then the message. Then, this is payment passed and this can be given as payment failed. What you observe is that you have created one of the classes which starts here and finishes here with the semicolon at the end. This is known as an anonymous class. It has no name. And you got the definitions of two methods created.



```

1  }
2  }
3  }
4  }
5  }
6  /**class Payment implements PaymentCallbacks{
7  @Override
8  public void onFailure(String message) {
9      System.out.println("Payment Failed: "+message);
10 }
11 }
12 @Override
13 public void onSuccess(String message) {
14     System.out.println("Payment Passed: "+message);
15 }
16 }*/
17
18
19 public class AnonymousClasses {
20
21     public static void main(String[] args) {
22
23         //PaymentCallbacks ref = new Payment();
24         //ref.onSuccess("Amount Paid 2000");
25         //ref.onFailure("Amount 2000 not processed");
26
27         PaymentCallbacks ref = new PaymentCallbacks() {
28             @Override
29             public void onSuccess(String message) {
30                 System.out.println("Payment Passed: "+message);
31             }
32
33             @Override
34             public void onFailure(String message) {
35                 System.out.println("Payment Failed: "+message);
36             }
37         };
    
```

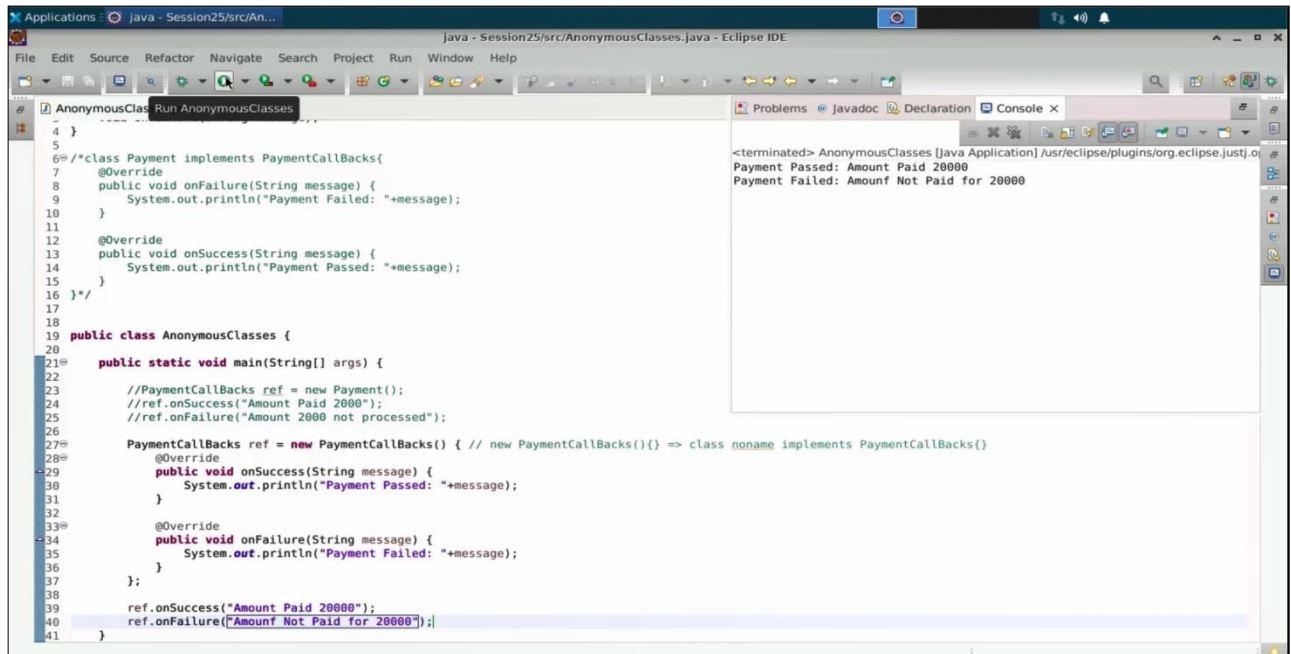
4.3 This new payment callback is an anonymous class, meaning there is a class with no name implementing your payment callbacks. The new payment callbacks are replaced with this anonymous class implementing the payment callback. Thus, a new class is created at runtime, an object of that class is instantiated, and the reference to that object is stored in this reference variable.



```

1  }
2  }
3  }
4  }
5  }
6  /**class Payment implements PaymentCallbacks{
7  @Override
8  public void onFailure(String message) {
9      System.out.println("Payment Failed: "+message);
10 }
11 }
12 @Override
13 public void onSuccess(String message) {
14     System.out.println("Payment Passed: "+message);
15 }
16 }*/
17
18
19 public class AnonymousClasses {
20
21     public static void main(String[] args) {
22
23         //PaymentCallbacks ref = new Payment();
24         //ref.onSuccess("Amount Paid 2000");
25         //ref.onFailure("Amount 2000 not processed");
26
27         PaymentCallbacks ref = new PaymentCallbacks() { // new PaymentCallbacks(){} => class noname implements PaymentCallbacks{}
28             @Override
29             public void onSuccess(String message) {
30                 System.out.println("Payment Passed: "+message);
31             }
32
33             @Override
34             public void onFailure(String message) {
35                 System.out.println("Payment Failed: "+message);
36             }
37         };
    
```

- 4.4 You can use the same reference variable and execute the success and the failure. Hence, The amount paid is 20000 and then you can give as on failure as well which says amount not paid for 20,000. Run the code here. You get the output coming as payment passes and the payment fails.



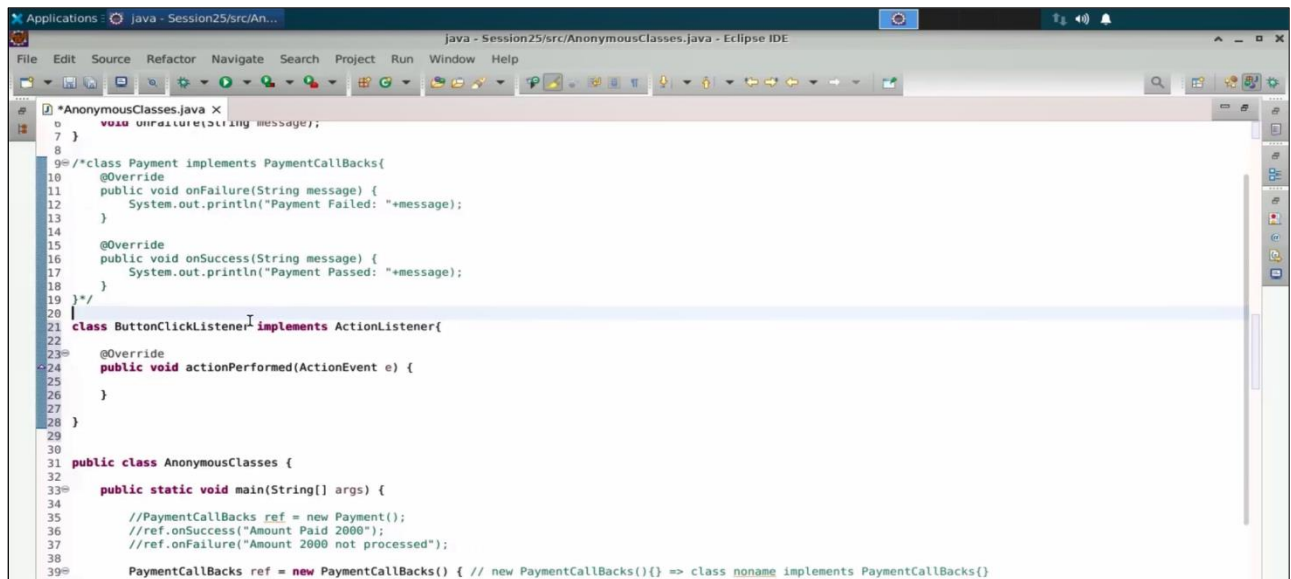
The screenshot displays the Eclipse IDE interface. The main editor window shows the source code for `AnonymousClasses.java`. The code defines a `Payment` class that implements `PaymentCallbacks` and a `AnonymousClasses` class with a `main` method. The `main` method creates a `Payment` object and calls `onSuccess` and `onFailure` methods with specific messages. The console window on the right shows the output of the program, which is:

```
<terminated> AnonymousClasses [Java Application] /usr/eclipse/plugins/org.eclipse.justj.o
Payment Passed: Amount Paid 20000
Payment Failed: Amount Not Paid for 20000
```

```
4 }
5
6 /**class Payment implements PaymentCallbacks{
7  @Override
8  public void onFailure(String message) {
9      System.out.println("Payment Failed: "+message);
10 }
11
12 @Override
13 public void onSuccess(String message) {
14     System.out.println("Payment Passed: "+message);
15 }
16 }*/
17
18
19 public class AnonymousClasses {
20
21     public static void main(String[] args) {
22
23         //PaymentCallbacks ref = new Payment();
24         //ref.onSuccess("Amount Paid 2000");
25         //ref.onFailure("Amount 2000 not processed");
26
27         PaymentCallbacks ref = new PaymentCallbacks() { // new PaymentCallbacks(){} => class noname implements PaymentCallbacks{}
28             @Override
29             public void onSuccess(String message) {
30                 System.out.println("Payment Passed: "+message);
31             }
32
33             @Override
34             public void onFailure(String message) {
35                 System.out.println("Payment Failed: "+message);
36             }
37         };
38
39         ref.onSuccess("Amount Paid 20000");
40         ref.onFailure("Amount Not Paid for 20000");
41     }
42 }
```

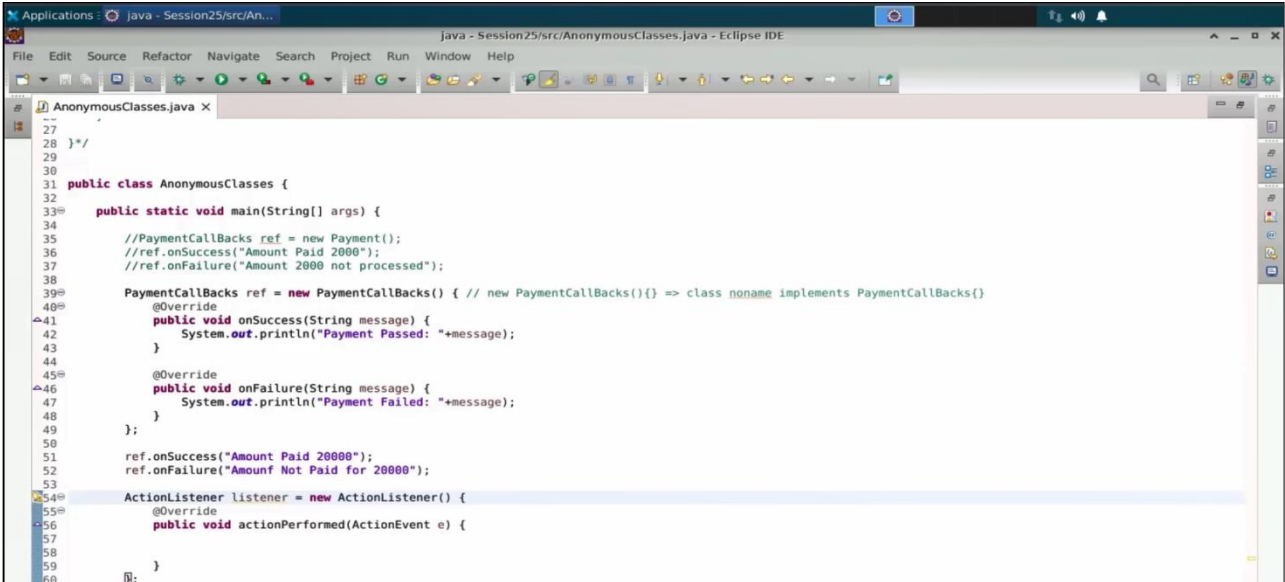
Step 5: Use anonymous classes

- 5.1 You can use anonymous classes in various scenarios, such as in the graphical user interfaces (GUIs) of Java. For instance, there's an interface known as `ActionListener` that allows you to control buttons and detect when they are clicked. The `ActionListener` interface has a single method called `actionPerformed`. If you create a class called `ButtonClickListener`, it will implement `ActionListener`. By adding the unimplemented method, `actionPerformed`, you can handle the button click events.



```
1  *AnonymousClasses.java x
2  1  void onFailure(String message);
3  2  }
4  3  }
5  4  }
6  5  }
7  6  }
8  7  }
9  8  /**class Payment implements PaymentCallbacks{
10 9  @Override
11 10 public void onFailure(String message) {
12 11     System.out.println("Payment Failed: "+message);
13 12 }
14 13 }
15 14 @Override
16 15 public void onSuccess(String message) {
17 16     System.out.println("Payment Passed: "+message);
18 17 }
19 18 }
20 19 }
21 20 class ButtonClickListener implements ActionListener{
22 21 @Override
23 22 public void actionPerformed(ActionEvent e) {
24 23 }
25 24 }
26 25 }
27 26 }
28 27 }
29 28 }
30 29 }
31 30 public class AnonymousClasses {
32 31 }
33 32 public static void main(String[] args) {
34 33 }
35 34 //PaymentCallbacks ref = new Payment();
36 35 //ref.onSuccess("Amount Paid 2000");
37 36 //ref.onFailure("Amount 2000 not processed");
38 37 }
39 38 PaymentCallbacks ref = new PaymentCallbacks() { // new PaymentCallbacks(){} => class noname implements PaymentCallbacks()
40 39 }
```

- 5.2 You must come up and create a separate class that contains only one single method implementation and then use the object of this class. Rather than writing a class, you can always come back and write as there is a listener, which is going to be an object of an anonymous class with the action performed.



```
Applications : Java - Session25/src/An...
Java - Session25/src/AnonymousClasses.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

AnonymousClasses.java x
27
28 */
29
30
31 public class AnonymousClasses {
32
33     public static void main(String[] args) {
34
35         //PaymentCallBacks ref = new Payment();
36         //ref.onSuccess("Amount Paid 2000");
37         //ref.onFailure("Amount 2000 not processed");
38
39         PaymentCallBacks ref = new PaymentCallBacks() { // new PaymentCallBacks() => class noname implements PaymentCallBacks{}
40             @Override
41             public void onSuccess(String message) {
42                 System.out.println("Payment Passed: "+message);
43             }
44
45             @Override
46             public void onFailure(String message) {
47                 System.out.println("Payment Failed: "+message);
48             }
49         };
50
51         ref.onSuccess("Amount Paid 20000");
52         ref.onFailure("Amount Not Paid for 20000");
53
54         ActionListener listener = new ActionListener() {
55             @Override
56             public void actionPerformed(ActionEvent e) {
57
58             }
59         };
60     }
```

By following the above steps, you have successfully implemented anonymous classes, allowing you to create concise and efficient instances of classes.