

Lesson 04 Demo 02

Implementing Anonymous Classes

Objective: To demonstrate the implementation of anonymous classes

Tools required: Eclipse IDE

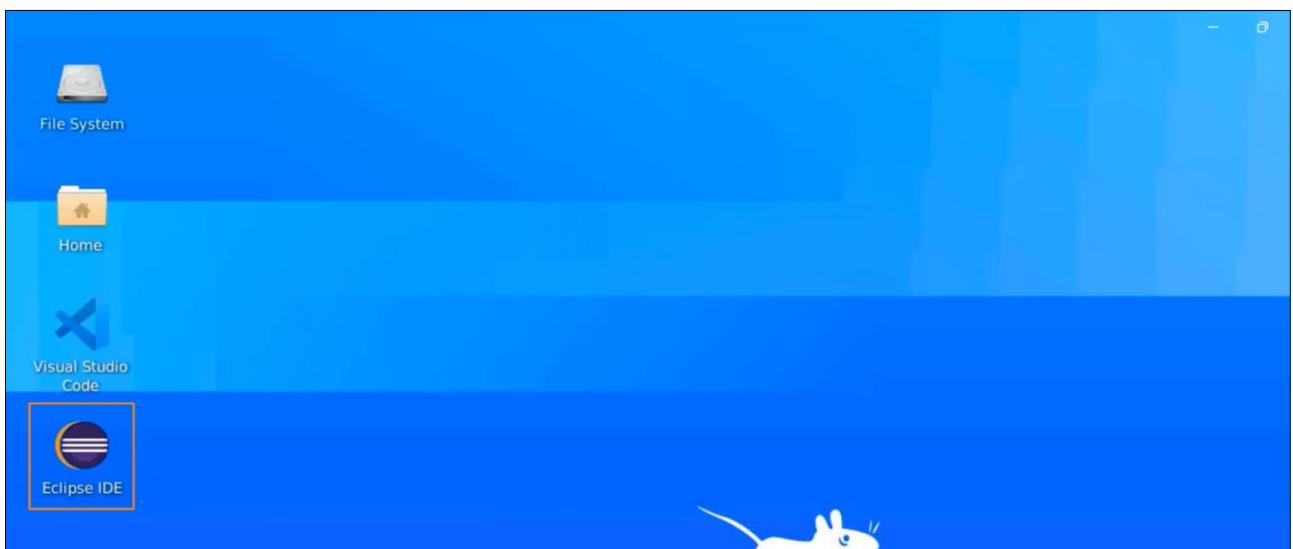
Prerequisites: None

Steps to be followed:

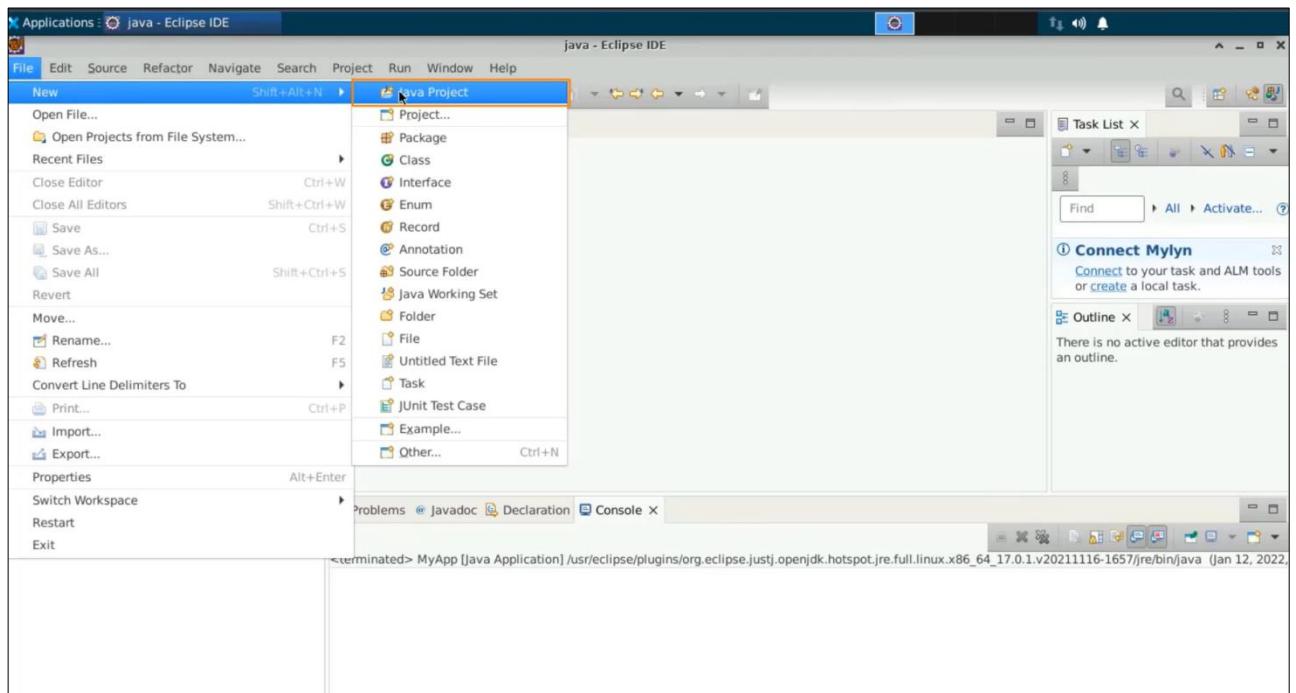
1. Open the IDE and create a new project
2. Write an interface and execute the code with example data
3. Execute the code and print the message accordingly
4. Override the methods called on success and failure
5. Use anonymous classes

Step 1: Open IDE and create a new project

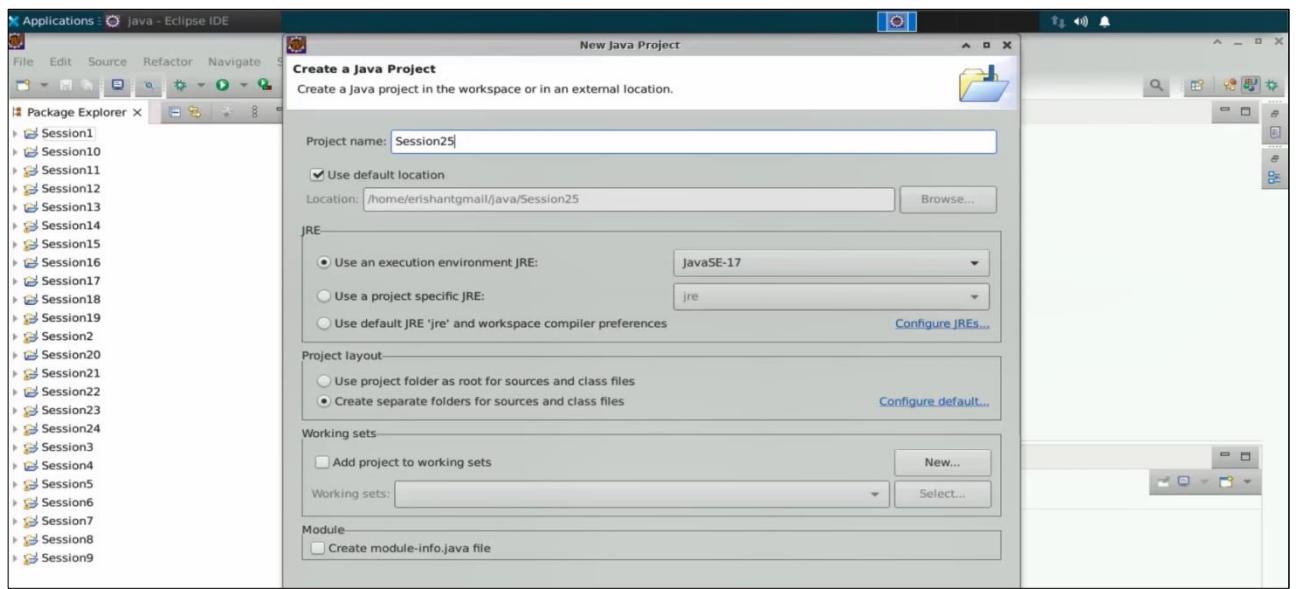
1.1 Open the Eclipse IDE



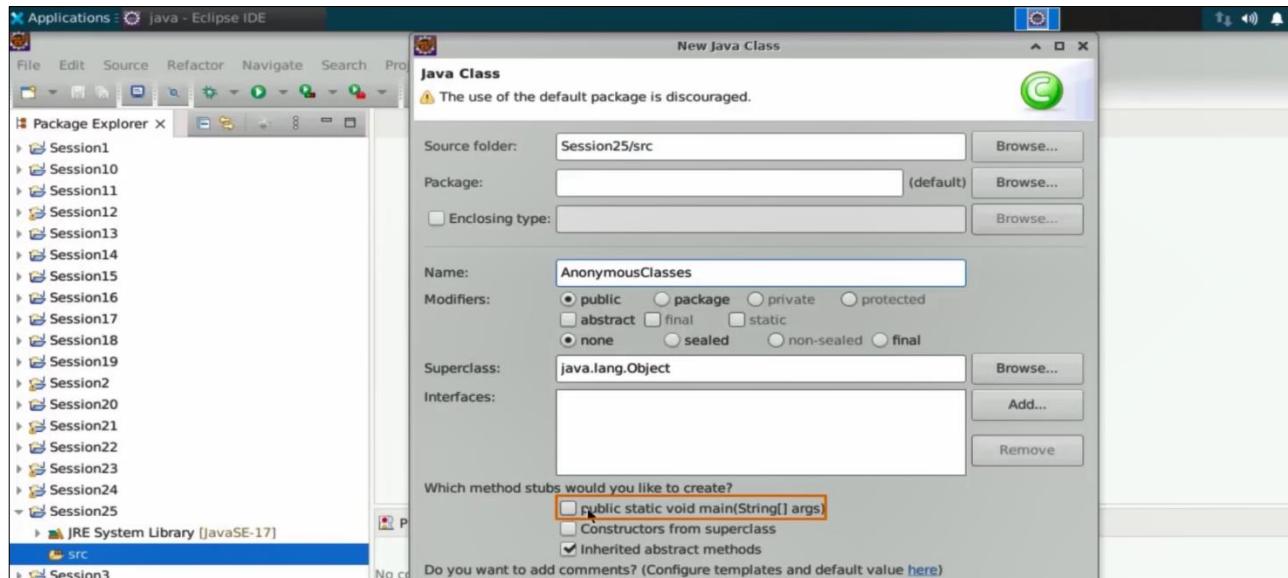
1.2 Select File, then New, and then Java project



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.

```

1 interface Payment{
2     default void pay(int amount) {
3     }
4 }
5 }
6 }
7 }
8 public class AnonymousClasses {
9
10    public static void main(String[] args) {
11        // TODO Auto-generated method stub
12    }
13 }
14 }
15 }
16 }

```

- 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.

```

Applications : java - Session25/src/AnonymousClasses.java
File Edit Source Refactor Navigate Search Project Run Window Help
AnonymousClasses.java X
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.

```

Applications : java - Session25/src/AnonymousClasses.java
File Edit Source Refactor Navigate Search Project Run Window Help
AnonymousClasses.java X
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("Failure: " + message);
10    }
11
12     @Override
13     public void onSuccess(String message) {
14         System.out.println("Success: " + message);
15    }
16 }
17
18
19 public class AnonymousClasses {
20
21     public static void main(String[] args) {
22         // TODO Auto-generated method stub
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.

```

Applications : Java - Session25/src/An...
File Edit Source Refactor Navigate Search Project Run Window Help
AnonymousClasses.java X
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         // TODO Auto-generated method stub
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.

```

Applications : Java - Session25/src/An...
File Edit Source Refactor Navigate Search Project Run Window Help
AnonymousClasses.java X
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
26     }
27
28 }
29

```

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

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Applications : java - Session25/src/AnonymousClasses.java - Eclipse IDE
- Left Panel (Editor):** The code for `AnonymousClasses.java` is displayed. It defines an interface `PaymentCallBacks` with two methods: `onSuccess` and `onFailure`. It also defines a class `Payment` that implements `PaymentCallBacks`. The `main` method creates an instance of `Payment` and calls its `onSuccess` method with the argument "Amount Paid 2000".
- Right Panel (Console):** The output window shows the results of the program execution. It displays the message "Payment Passed: Amount Paid 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
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
26     }
27
28 }
29

```

- 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.

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Applications : java - Session25/src/AnonymousClasses.java - Eclipse IDE
- Left Panel (Editor):** The code for `AnonymousClasses.java` is displayed. It is identical to the previous screenshot, except the `ref.onFailure` line is also uncommented, showing "Amount 2000 not processed".
- Right Panel (Console):** The output window shows the results of the program execution. It displays the message "Payment Passed: Amount Paid 2000" followed by "Payment Failed: Amount 2000 not processed".

```

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     }
28 }
29

```

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.

```

  Applications  java - Session25/src/An...
  File Edit Source Refactor Navigate Search Project Run Window Help
  AnonymousClasses.java X
  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.

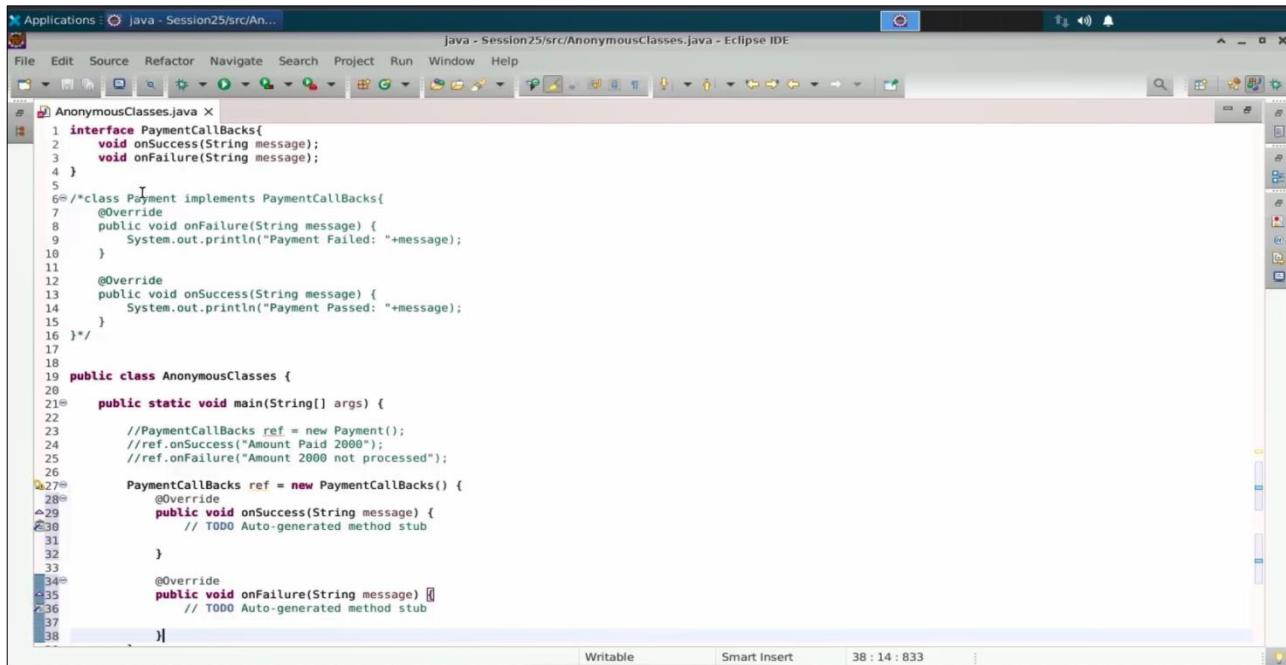
```

  Applications  java - Session25/src/An...
  File Edit Source Refactor Navigate Search Project Run Window Help
  AnonymousClasses.java X
  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         Multiple markers at this line
 28         - The type new PaymentCallBacks(){} must implement the inherited abstract method PaymentCallBacks.onSuccess
 29         (String)
 30         - The type new PaymentCallBacks(){} must implement the inherited abstract method PaymentCallBacks.onFailure
 31         (String)
 32     }
 33 }

```

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.



The screenshot shows the Eclipse IDE interface with the title bar "Java - Session25/src/AnonymousClasses.java - Eclipse IDE". The main window displays the Java code for "AnonymousClasses.java". The code defines an interface "PaymentCallBacks" with two methods: "onSuccess" and "onFailure". It then implements this interface in the "AnonymousClasses" class. The "onSuccess" method prints "Payment Passed: <message>". The "onFailure" method prints "Payment Failed: <message>". Lines 27 and 34 show the creation of a new instance of "PaymentCallBacks" and its assignment to the variable "ref". Lines 29 and 35 show the overriding of the "onSuccess" and "onFailure" methods respectively. The code is annotated with TODO comments indicating auto-generated stubs.

```
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             @Override
29             public void onSuccess(String message) {
30                 // TODO Auto-generated method stub
31
32             }
33
34             @Override
35             public void onFailure(String message) {
36                 // TODO Auto-generated method stub
37
38         }
39     }
40 }
```

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.

```

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

```

Applications : java - Session25/src/AnonymousClasses.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
AnonymousClasses.java X
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 shows the Eclipse IDE interface with a Java file named 'AnonymousClasses.java' open. The code implements a Payment interface with onSuccess and onFailure methods, and defines an AnonymousClass AnonymousClasses that creates a PaymentCallbacks object. The code is as follows:

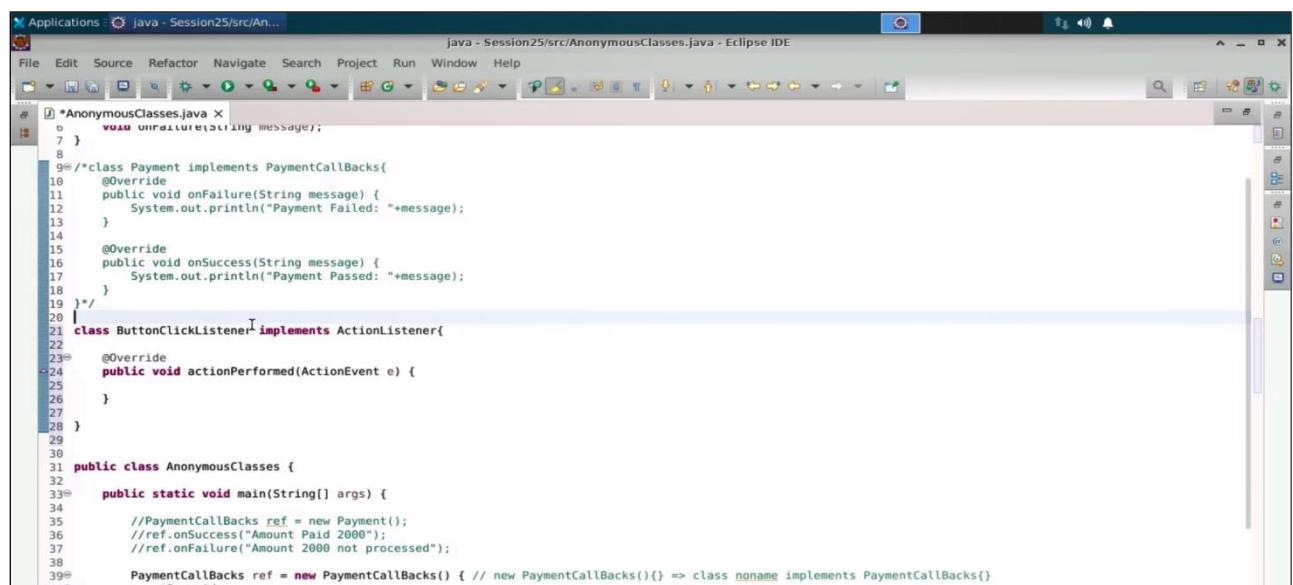
```
# 1 AnonymousClass Run AnonymousClasses
2
3
4 }
5
6 /*class Payment implements PaymentCallBacks{
7     @Override
8     public void onFailure(String message) {
9         System.out.println("Payment Failed: "+message);
10    }
11
12     @Override
13     public void onSuccess(String message) {
14         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             };
38
39             ref.onSuccess("Amount Paid 20000");
40             ref.onFailure("Amount Not Paid for 20000");
41     }
42 }
```

The 'Console' tab in the Eclipse interface shows the output of the program:

```
<terminated> AnonymousClasses [Java Application] /usr/eclipse/plugins/org.eclipse.justj/oracle/jdk1.8.0_131/bin/java -jar /usr/eclipse/plugins/com.simplilearn.essentialjava-1.0.0.jar
Payment Passed: Amount Paid 20000
Payment Failed: Amount Not Paid for 20000
```

Step 5: Use anonymous classes

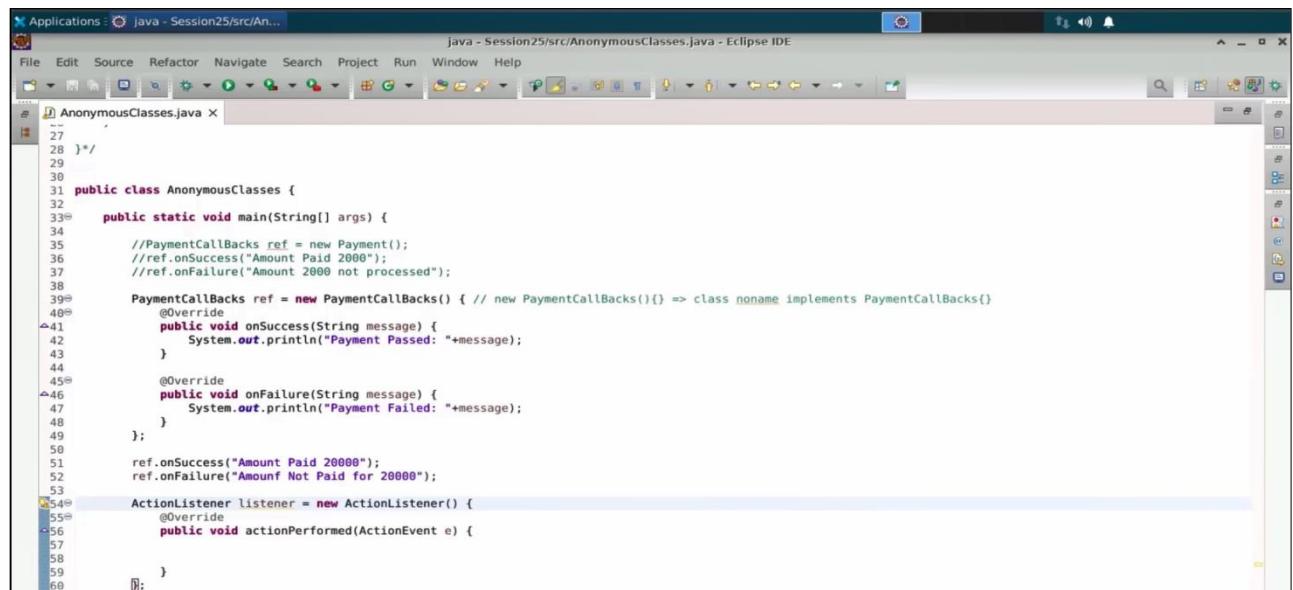
- 5.1 You can use anonymous classes in various scenarios, such as in the graphical user interfaces (GUIs) of Java. For instance, there's an interface known as `ActionListener` that allows you to control buttons and detect when they are clicked. The `ActionListener` interface has a single method called `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.



The screenshot shows the Eclipse IDE interface with a Java file named "AnonymousClasses.java" open. The code implements an anonymous class for the ActionListener interface:

```
* *AnonymousClasses.java *
  1  package com.simplilearn;
  2
  3  import java.awt.event.ActionListener;
  4
  5  public class AnonymousClasses {
  6      public static void main(String[] args) {
  7          //PaymentCallBacks ref = new Payment();
  8          //ref.onSuccess("Amount Paid 2000");
  9          //ref.onFailure("Amount 2000 not processed");
 10
 11          PaymentCallBacks ref = new PaymentCallBacks() { // new PaymentCallBacks(){} => class noname implements PaymentCallBacks{
 12              @Override
 13              public void onFailure(String message) {
 14                  System.out.println("Payment Failed: "+message);
 15              }
 16
 17              @Override
 18              public void onSuccess(String message) {
 19                  System.out.println("Payment Passed: "+message);
 20              }
 21          };
 22
 23          ref.addActionListener(new ButtonClickListener());
 24
 25          ref.actionPerformed(null);
 26      }
 27  }
```

- 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.



The screenshot shows the Eclipse IDE interface with a Java file named "AnonymousClasses.java" open in the editor. The code implements an anonymous inner class for a PaymentCallbacks interface, which overrides the onSuccess and onFailure methods to print messages to System.out. It also creates an ActionListener anonymous class that overrides actionPerformed.

```
Applications : java - Session25/src/An...
File Edit Source Refactor Navigate Search Project Run Window Help
AnonymousClasses.java X
1 /*
2  */
3
4 public class AnonymousClasses {
5
6     public static void main(String[] args) {
7
8         PaymentCallbacks ref = new Payment() {
9             @Override
10            public void onSuccess(String message) {
11                System.out.println("Payment Passed: "+message);
12            }
13
14            @Override
15            public void onFailure(String message) {
16                System.out.println("Payment Failed: "+message);
17            }
18        };
19
20        ref.onSuccess("Amount Paid 2000");
21        ref.onFailure("Amount 2000 not processed");
22
23        ActionListener listener = new ActionListener() {
24            @Override
25            public void actionPerformed(ActionEvent e) {
26
27            }
28        };
29
30    }
31
32 }
```

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