

Lesson 06 Demo 05

Utilizing Lambda Expressions in Java

Objective: To implement Lambda expressions in Java along with the creation of functional interfaces

Tools Required: Eclipse IDE

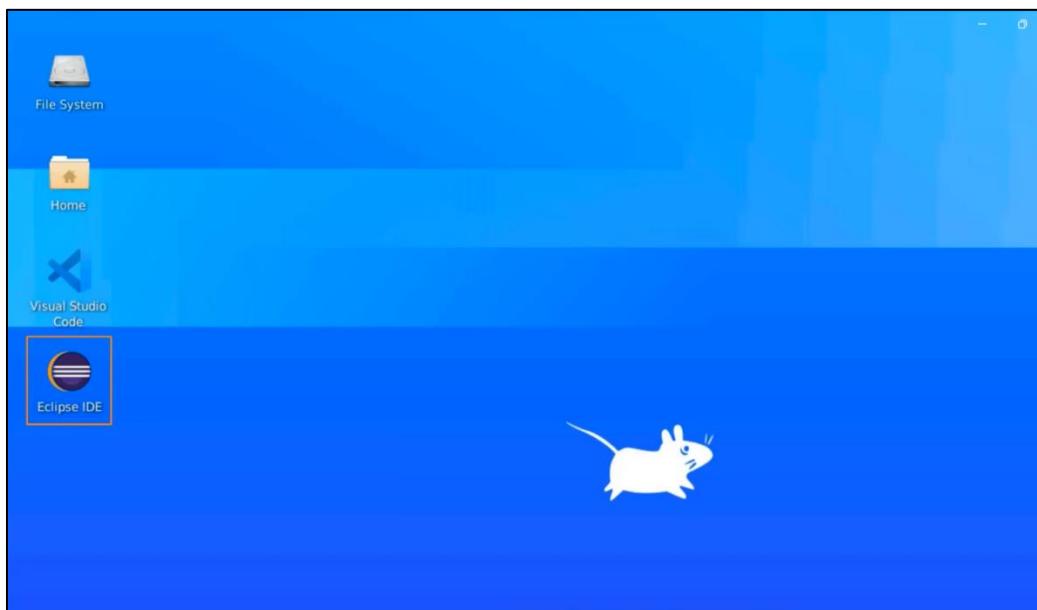
Prerequisites: None

Steps to be followed:

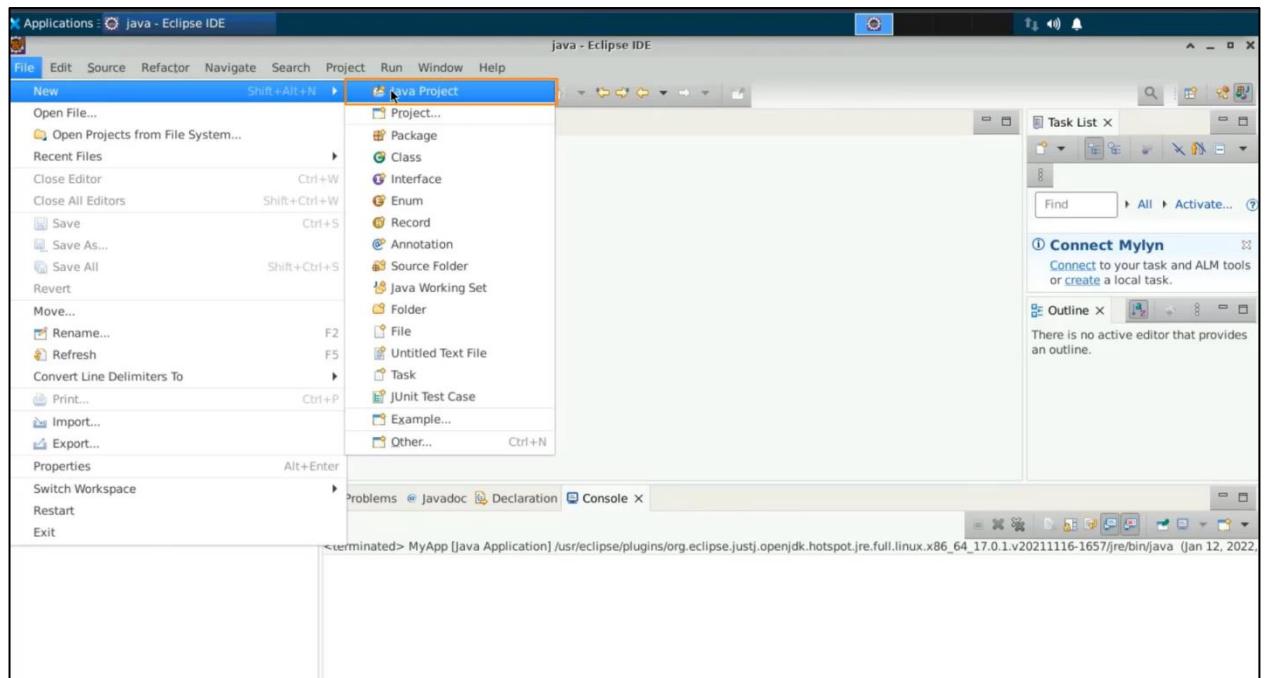
1. Open the Eclipse IDE and create a new Java project
2. Create a functional interface
3. Implement the interface
4. Use anonymous class for running an interface
5. Create a function, which is the implementation for the Lambda expression
6. Write another interface as login and execute the code with sample data
7. Execute the Lambda expression with example data

Step 1: Open the Eclipse IDE and create a new Java project

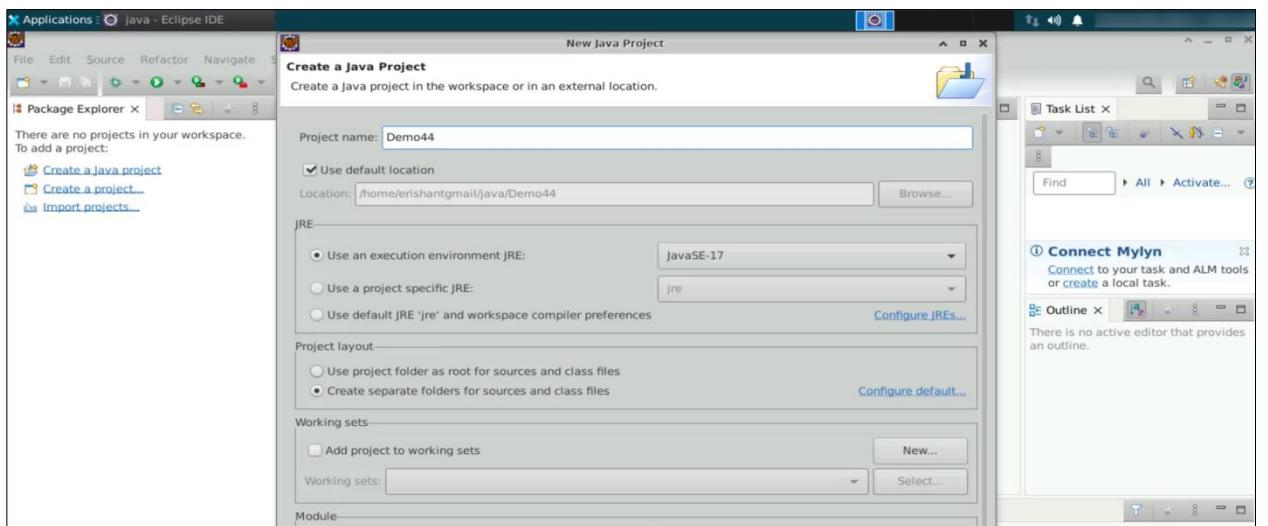
1.1 Open the Eclipse IDE



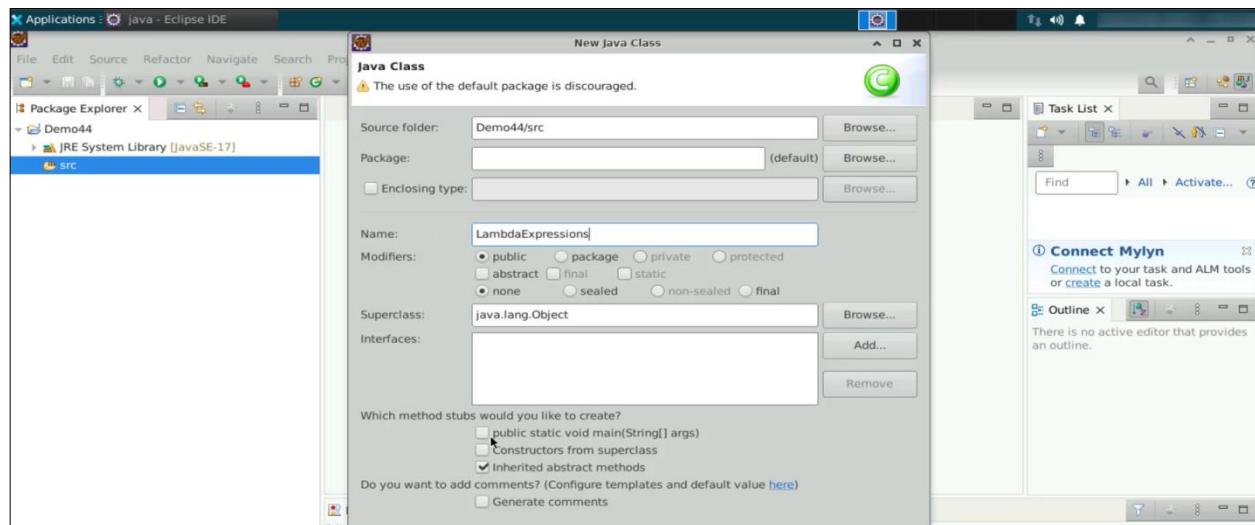
1.2 Select File, then New, and then Java project



1.3 Name the project **Demo44**, uncheck **Create a module-info.java file**, and press **Finish**



1.4 With **Demo44** selected in the **src** folder, right-click and create a new class. Name this class **LambdaExpressions**, then select the **main** method, and then select **Finish**



Step 2: Create a functional interface

2.1 Create a functional interface called **Register** with a method called **validate** for declaring abstract methods. The rule for a functional interface is to have only one abstract method

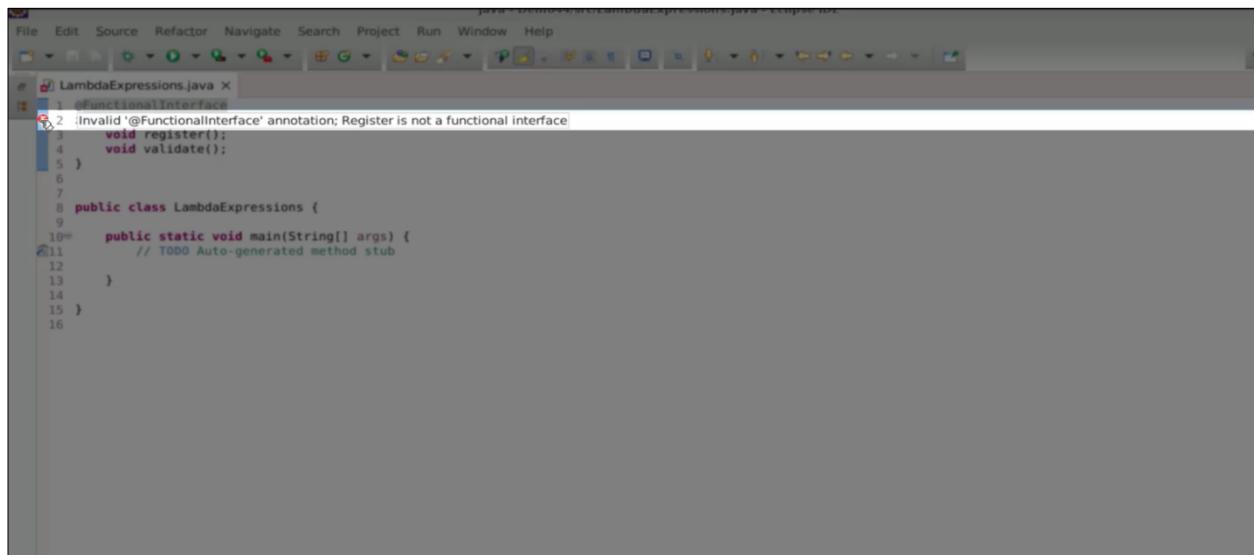
```

interface Register{
    void register();
    void validate();
}

public class LambdaExpressions {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }
}

```

2.2 To ensure your interface is a functional interface, use the **@FunctionalInterface** annotation. The error you encountered indicates that **Register** is not a valid annotation for this interface

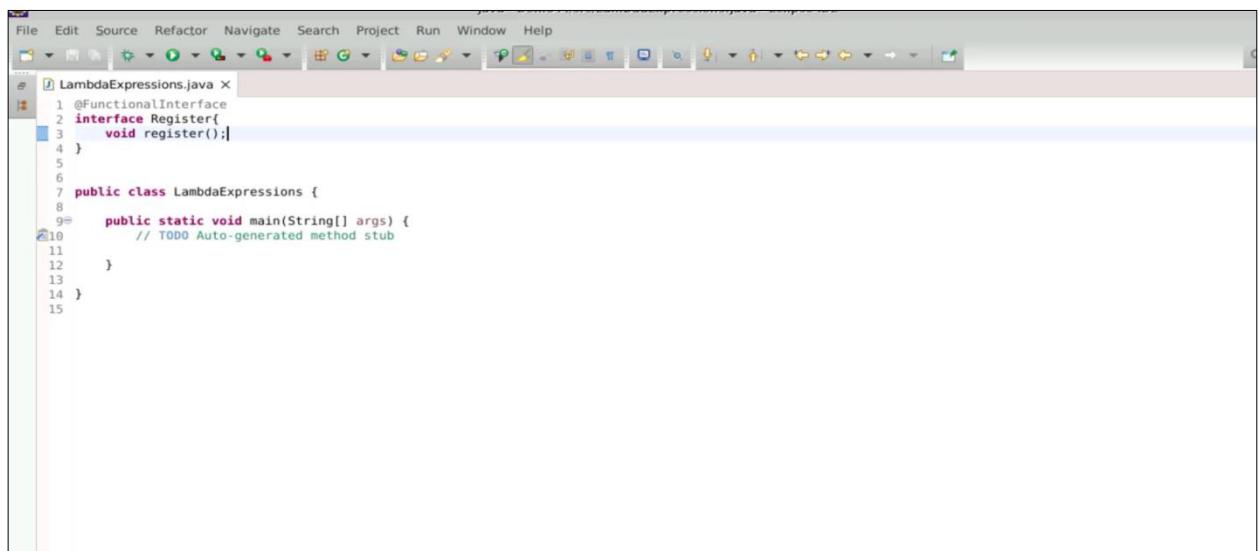


The screenshot shows a Java code editor with the following code:

```
File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
1 @FunctionalInterface
2 interface Register{
3     void register();
4 }
5
6
7 public class LambdaExpressions {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11     }
12 }
13
14 }
15
16
```

A red error indicator is present on line 2, highlighting the word "Register". A tooltip or status message above the line reads: "Invalid '@FunctionalInterface' annotation; Register is not a valid annotation".

2.3 The moment you remove this **validate** method, you can notice that the error is gone. Functional interfaces are meant to hold only one single abstract method



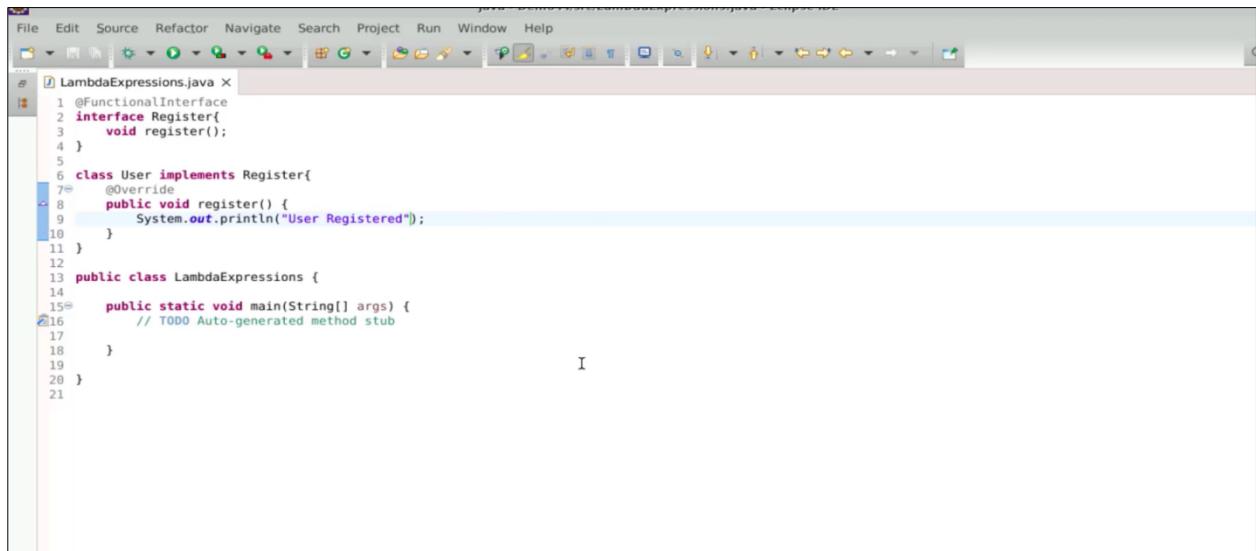
The screenshot shows the same Java code as before, but the "validate" method has been removed from the interface. The code now looks like this:

```
File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
1 @FunctionalInterface
2 interface Register{
3     void register();
4 }
5
6
7 public class LambdaExpressions {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11     }
12 }
13
14 }
15
```

The error message is no longer present, indicating that the code is valid.

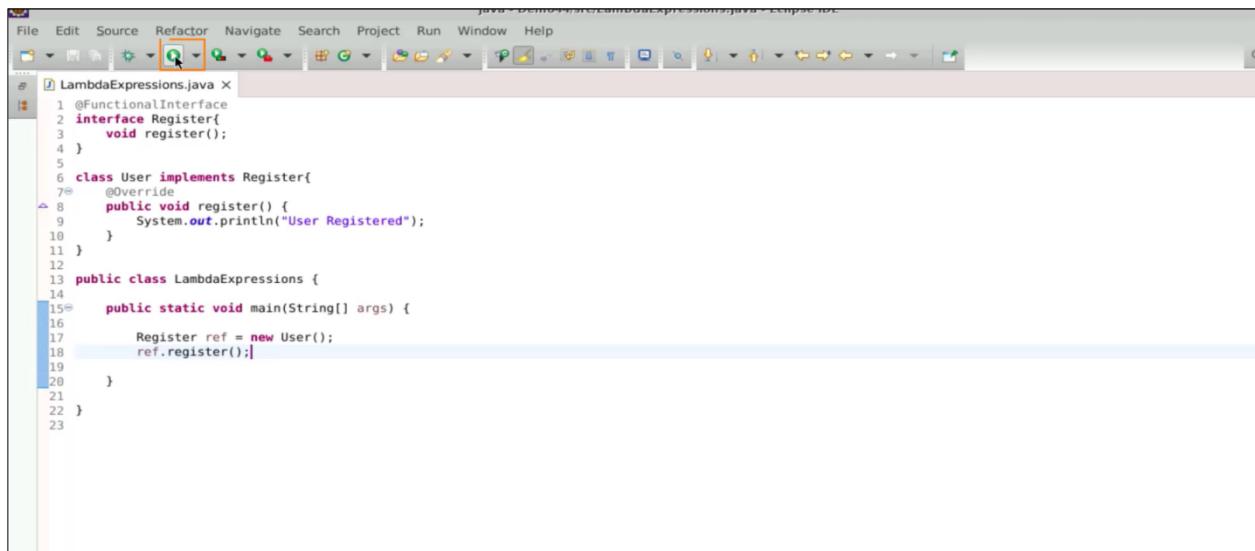
Step 3: Implement the interface

3.1 To work with interface implementation, create a class that implements it. For example, the **User** class can implement the **register** interface. Then, print **User registered** using **System.out.println**



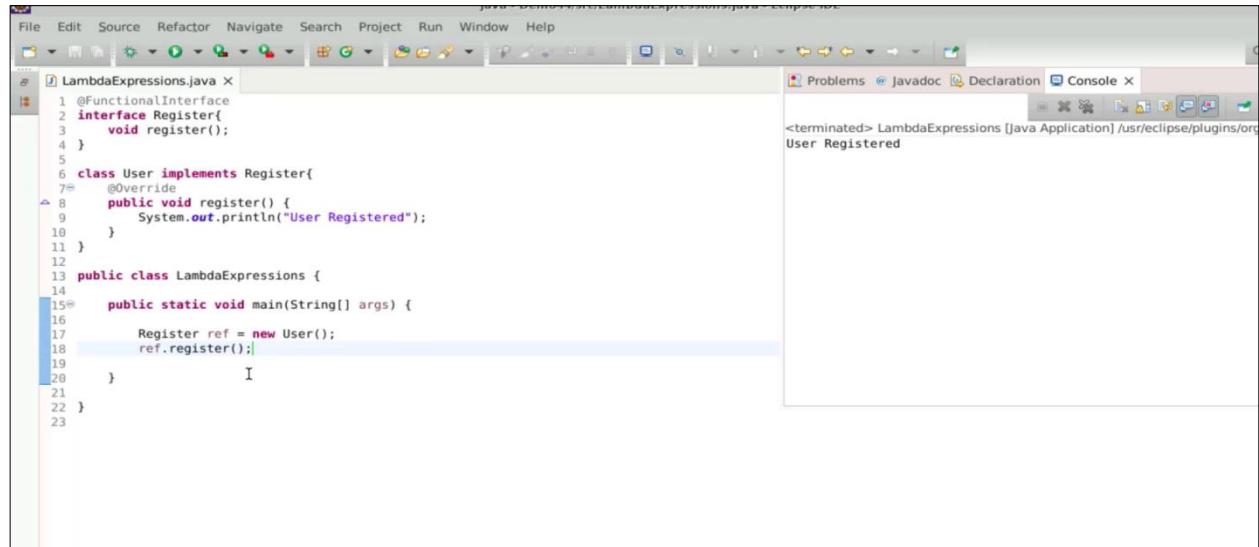
```
File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
1 @FunctionalInterface
2 interface Register{
3     void register();
4 }
5
6 class User implements Register{
7     @Override
8     public void register() {
9         System.out.println("User Registered");
10    }
11 }
12
13 public class LambdaExpressions {
14
15     public static void main(String[] args) {
16         // TODO Auto-generated method stub
17     }
18 }
19
20 }
21
```

3.2 To achieve polymorphism, create a reference variable for the **register** method and execute it using the **user** object



```
File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
1 @FunctionalInterface
2 interface Register{
3     void register();
4 }
5
6 class User implements Register{
7     @Override
8     public void register() {
9         System.out.println("User Registered");
10    }
11 }
12
13 public class LambdaExpressions {
14
15     public static void main(String[] args) {
16
17         Register ref = new User();
18         ref.register();
19
20     }
21
22 }
23
```

3.3 This is a general way of working with the interface and the implementation. However, there is another way known as anonymous classes, through which you can implement this interface



```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
1 @FunctionalInterface
2 interface Register{
3     void register();
4 }
5
6 class User implements Register{
7     @Override
8     public void register() {
9         System.out.println("User Registered");
10    }
11 }
12
13 public class LambdaExpressions {
14
15     public static void main(String[] args) {
16
17         Register ref = new User();
18         ref.register();
19
20     }
21
22 }
23

```

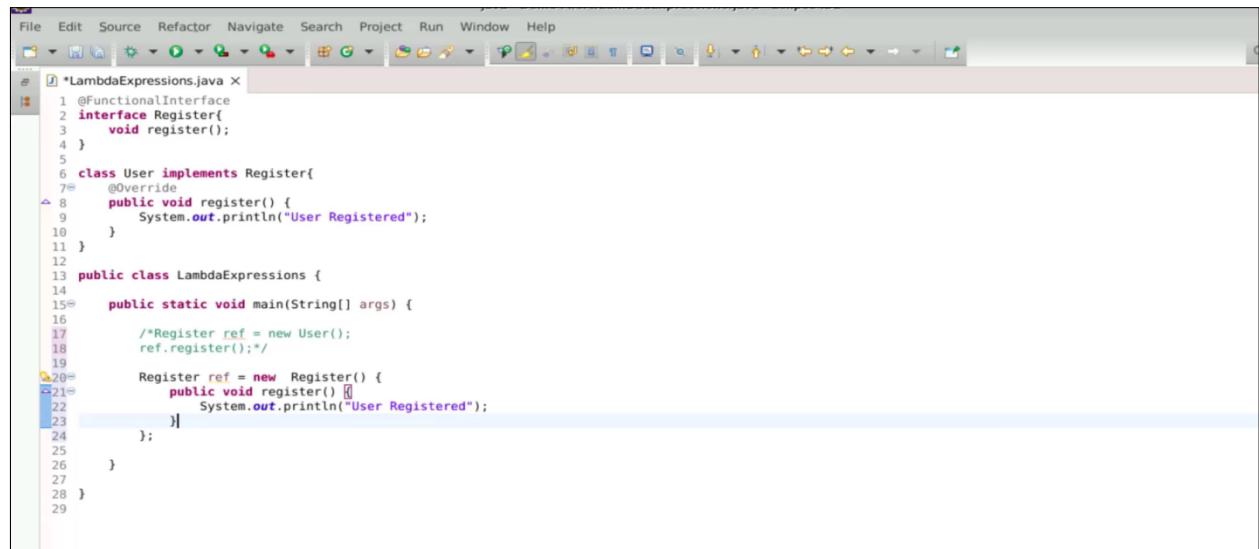
Problems Javadoc Declaration Console X

<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/org

User Registered

Step 4: Use anonymous class for running an interface

4.1 To avoid creating a separate class, use an anonymous class where the **register** reference is assigned to a new instance of **register**. Define the **register** method within the anonymous class



```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
1 @FunctionalInterface
2 interface Register{
3     void register();
4 }
5
6 class User implements Register{
7     @Override
8     public void register() {
9         System.out.println("User Registered");
10    }
11 }
12
13 public class LambdaExpressions {
14
15     public static void main(String[] args) {
16
17         /*Register ref = new User();
18         ref.register();*/
19
20         Register ref = new Register() {
21             public void register() {
22                 System.out.println("User Registered");
23             }
24         };
25
26     }
27
28 }
29

```

4.2 Execute the **register** method using the reference variable of the anonymous class to achieve the same output

The screenshot shows the Eclipse IDE interface with the code editor containing `LambdaExpressions.java`. The code implements a `Register` functional interface and uses both anonymous classes and lambda expressions to implement the `register` method. The output window shows the message "User Registered".

```

1  @FunctionalInterface
2  interface Register{
3      void register();
4  }
5
6  class User implements Register{
7      @Override
8      public void register() {
9          System.out.println("User Registered");
10     }
11 }
12
13 public class LambdaExpressions {
14
15     public static void main(String[] args) {
16
17         /*Register ref = new User();
18         ref.register();*/
19
20         Register ref = new Register() {
21             public void register() {
22                 System.out.println("User Registered");
23             }
24         };
25
26         ref.register();
27
28     }
29
30 }
31

```

Step 5: Create a function, which is the implementation for the Lambda expression

5.1 Use a Lambda expression with a simple syntax to implement the functional interface by creating a **register** function with the desired definition using the arrow operator.

The screenshot shows the Eclipse IDE interface with the code editor containing `LambdaExpressions.java`. The code is identical to the previous one, but the second implementation of the `register` method uses a lambda expression (`() ->`) instead of an anonymous class. The output window shows the message "User Registered".

```

1  @FunctionalInterface
2  interface Register{
3      void register();
4  }
5
6  class User implements Register{
7      @Override
8      public void register() {
9          System.out.println("User Registered");
10     }
11 }
12
13 public class LambdaExpressions {
14
15     public static void main(String[] args) {
16
17         /*Register ref = new User();
18         ref.register();*/
19
20         /*Register ref = new Register() {
21             public void register() {
22                 System.out.println("User Registered");
23             }
24         };*/
25
26         ref.register();
27
28         Register ref = () ->{
29
30     };
31

```

5.2 Print **User registered** using the Lambda expression implementation for the functional interface, providing the same output as before. Lambda expressions offer a simpler and easier implementation compared to creating a class and using polymorphic statements

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaRun LambdaExpressions
1 @FunctionalInterface
2 interface Register{
3     void register();
4 }
5
6 class User implements Register{
7     @Override
8     public void register() {
9         System.out.println("User Registered");
10    }
11 }
12
13 public class LambdaExpressions {
14
15     public static void main(String[] args) {
16
17         /*Register ref = new User();
18         ref.register();*/
19
20         /*Register ref = new Register() {
21             public void register() {
22                 System.out.println("User Registered");
23             }
24         };
25
26         ref.register();*/
27
28         Register ref = ()->{
29             System.out.println("User Registered with Lambda Expression");
30         };
31         ref.register();
32     }
33 }

```

<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/org

User Registered with Lambda Expression

Step 6: Write another interface as login and execute the code with sample data

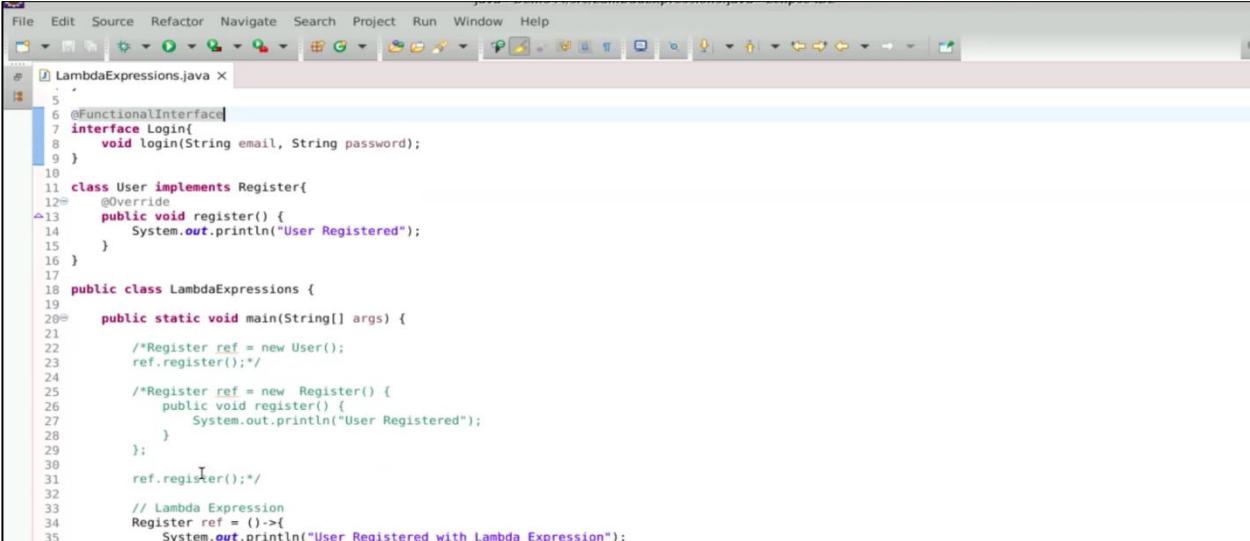
6.1 Let us write another interface called **Login**. Let us write a method named **login**, which takes the email and the password as input

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java *
1 @FunctionalInterface
2 interface Register{
3     void register();
4 }
5
6 interface Login{
7     void login(String email, String password);
8 }
9
10 class User implements Register{
11     @Override
12     public void register() {
13         System.out.println("User Registered");
14     }
15 }
16
17 public class LambdaExpressions {
18
19     public static void main(String[] args) {
20
21         /*Register ref = new User();
22         ref.register();*/
23
24         /*Register ref = new Register() {
25             public void register() {
26                 System.out.println("User Registered");
27             }
28         };
29
30         ref.register();*/
31
32     }
33 }

```

6.2 The **login** interface is marked as a functional interface to enforce the restriction of having no additional abstract methods

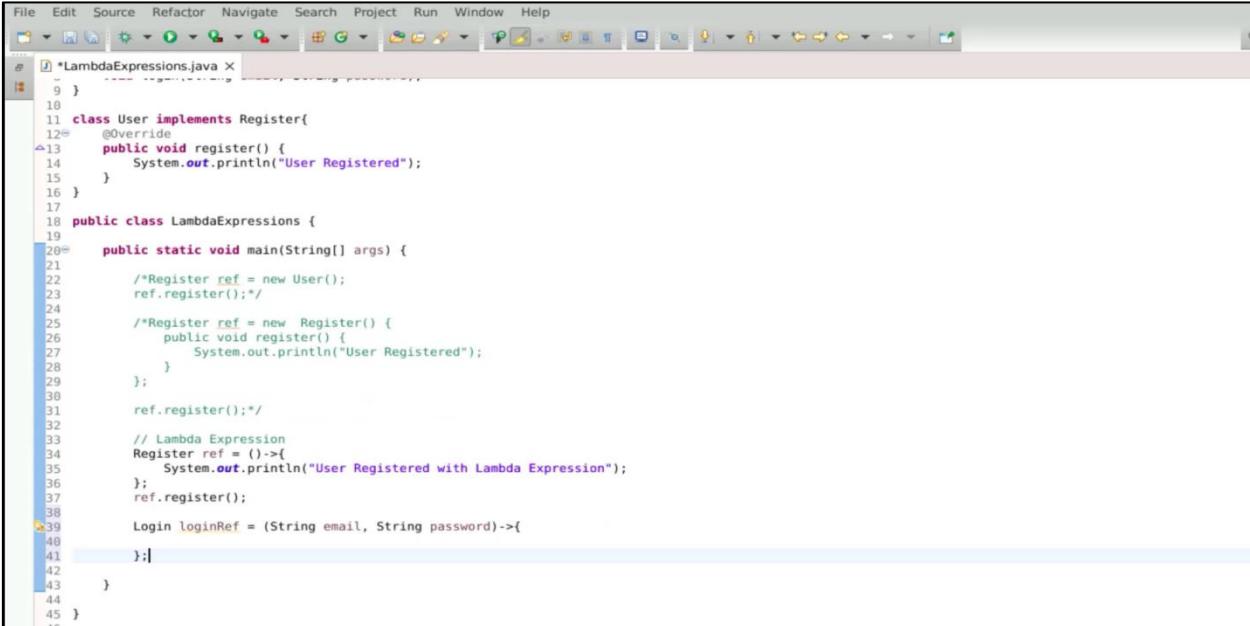


```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
5
6 @FunctionalInterface
7 interface Login{
8     void login(String email, String password);
9 }
10
11 class User implements Register{
12     @Override
13     public void register() {
14         System.out.println("User Registered");
15     }
16 }
17
18 public class LambdaExpressions {
19
20     public static void main(String[] args) {
21
22         /*Register ref = new User();
23         ref.register();*/
24
25         /*Register ref = new Register() {
26             public void register() {
27                 System.out.println("User Registered");
28             }
29         };
30         ref.register();*/
31
32         // Lambda Expression
33         Register ref = ()->{
34             System.out.println("User Registered with Lambda Expression");
35         };
36         ref.register();
37
38         Login loginRef = (String email, String password)->{
39
40         };
41
42     }
43
44 }
45

```

6.3 Define the implementation for the **login** function using a Lambda expression, providing parameters for the email and password strings, and allowing multiple instructions within the expression



```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
5
6
7
8
9
10
11 class User implements Register{
12     @Override
13     public void register() {
14         System.out.println("User Registered");
15     }
16 }
17
18 public class LambdaExpressions {
19
20     public static void main(String[] args) {
21
22         /*Register ref = new User();
23         ref.register();*/
24
25         /*Register ref = new Register() {
26             public void register() {
27                 System.out.println("User Registered");
28             }
29         };
30         ref.register();*/
31
32         // Lambda Expression
33         Register ref = ()->{
34             System.out.println("User Registered with Lambda Expression");
35         };
36         ref.register();
37
38         Login loginRef = (String email, String password)->{
39             System.out.println("User Registered with Lambda Expression");
40         };
41
42     }
43
44 }
45

```

6.4 In the Lambda expression, you can define a complete method implementation, such as connecting to the database, validating the user from the database, and printing a message thanking the user for logging in with the specified email

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
11 class User implements Register{
12     @Override
13     public void register() {
14         System.out.println("User Registered");
15     }
16 }
17
18 public class LambdaExpressions {
19
20     public static void main(String[] args) {
21
22         /*Register ref = new User();
23         ref.register();*/
24
25         /*Register ref = new Register() {
26             public void register() {
27                 System.out.println("User Registered");
28             }
29         };
30
31         ref.register();*/
32
33         // Lambda Expression
34         Register ref = ()->{
35             System.out.println("User Registered with Lambda Expression");
36         };
37         ref.register();
38
39         Login loginRef = (String email, String password)-> {
40             System.out.println("Connecting to DB");
41             System.out.println("Validating User from DB");
42             System.out.println("Thank you for Logging in with email "+email);
43         };
44
45     }
46 }
47
48

```

6.5 Execute the **login** method using the **Login** reference, providing the email **John@example.com** and password **john123** as parameters, allowing for the implementation of the entire logic within the method

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
11 class User implements Register{
12     @Override
13     public void register() {
14         System.out.println("User Registered");
15     }
16 }
17
18 public class LambdaExpressions {
19
20     public static void main(String[] args) {
21
22         /*Register ref = new User();
23         ref.register();*/
24
25         /*Register ref = new Register() {
26             public void register() {
27                 System.out.println("User Registered");
28             }
29         };
30
31         ref.register();*/
32
33         // Lambda Expression
34         Register ref = ()->{
35             System.out.println("User Registered with Lambda Expression");
36         };
37         ref.register();
38
39         Login loginRef = (String email, String password)-> {
40             System.out.println("Connecting to DB");
41             System.out.println("Validating User from DB");
42             System.out.println("Thank you for Logging in with email "+email);
43         };
44
45         loginRef.login("john@example.com", "john123");
46
47     }
48

```

Console output:

```

<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/or
User Registered with Lambda Expression
Connecting to DB
Validating User from DB
Thank you for Logging in with email john@example.com

```

6.6 Implement a conditional logic where if the email is **admin@123** and the password is **pass123**, the login is successful; otherwise, it is a failed login. When running the code, it will display **login failed**

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
15 }
16 }
17
18 public class LambdaExpressions {
19
20     public static void main(String[] args) {
21
22         /*Register ref = new User();
23         ref.register();*/
24
25         /*Register ref = new Register() {
26             public void register() {
27                 System.out.println("User Registered");
28             }
29         };
30
31         ref.register();*/
32
33         // Lambda Expression
34         Register ref = ()->{
35             System.out.println("User Registered with Lambda Expression");
36         };
37         ref.register();
38
39         Login loginRef = (String email, String password)-> {
40             System.out.println("Connecting to DB");
41             System.out.println("Validating User from DB");
42             if(email.equals("admin@123") && password.equals("pass123")) {
43                 System.out.println("Thank you for Logging in with email "+email);
44             }else {
45                 System.out.println("Login Failed");
46             }
47         };
48
49         loginRef.login("John@example.com", "john123");
50
51     }
52 }
```

Problems Javadoc Declaration Console X

<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/org
User Registered with Lambda Expression
Connecting to DB
Validating User from DB
Login Failed

6.7 But if you use **admin@example.com** and **pass123**, the lambda expression method can be written as a complete business method, and it is not restricted to work with only one basic part

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaRunLambdaExpressions X
15 }
16 }
17
18 public class LambdaExpressions {
19
20     public static void main(String[] args) {
21
22         /*Register ref = new User();
23         ref.register();*/
24
25         /*Register ref = new Register() {
26             public void register() {
27                 System.out.println("User Registered");
28             }
29         };
30
31         ref.register();*/
32
33         // Lambda Expression
34         Register ref = ()->{
35             System.out.println("User Registered with Lambda Expression");
36         };
37         ref.register();
38
39         Login loginRef = (String email, String password)-> {
40             System.out.println("Connecting to DB");
41             System.out.println("Validating User from DB");
42             if(email.equals("admin@example.com") && password.equals("pass123")) {
43                 System.out.println("Thank you for Logging in with email "+email);
44             }else {
45                 System.out.println("Login Failed");
46             }
47         };
48
49         loginRef.login("admin@example.com", "pass123");
50
51     }
52 }
```

Problems Javadoc Declaration Console X

<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/org
User Registered with Lambda Expression
Connecting to DB
Validating User from DB
Thank you for Logging in with email admin@example.com

6.8 In Lambda expressions, the data types of the parameters can be omitted, allowing you to directly use variables like **email** and **password** without explicitly specifying their data types



The screenshot shows the IntelliJ IDEA interface with the file `LambdaExpressions.java` open. The code implements a `Register` interface with a `register` method that prints "User Registered". It also demonstrates a lambda expression for the same method. Additionally, it includes a `Login` class with a static method that prints connection and validation messages, checks email and password, and prints a thank you message or login failed message.

```
# LambdaExpressions.java X
15 }
16 }
17
18 public class LambdaExpressions {
19
20     public static void main(String[] args) {
21
22         /*Register ref = new User();
23         ref.register();*/
24
25         /*Register ref = new Register() {
26             public void register() {
27                 System.out.println("User Registered");
28             }
29         };
30
31         ref.register();*/
32
33         // Lambda Expression
34         Register ref = ()->{
35             System.out.println("User Registered with Lambda Expression");
36         };
37         ref.register();
38
39         //Login loginRef = (String email, String password)-> {
40         Login loginRef = (email, password)-> {
41             System.out.println("Connecting to DB");
42             System.out.println("Validating User from DB");
43             if(email.equals("admin@example.com") && password.equals("pass123")) {
44                 System.out.println("Thank you for Logging in with email "+email);
45             }else {
46                 System.out.println("Login Failed");
47             }
48         };
49     }
50 }
```

6.9 You can even change the parameter name, such as **emailID**. It is not mandatory to use the same parameter names

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

- File Menu:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- LambdaExpressions.java Content:** The code demonstrates various Java 8 features like static imports, lambda expressions, and method references. It includes a main method that registers a user and prints a welcome message.
- Console Output:** The output window shows the application's log:

```
<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/or
User Registered with Lambda Expression
Conneting to DB
Validating User from DB
Thank you for Logging in with email admin@example.com
```

- 6.10 To create a lambda expression that returns data, define another functional interface called **PromoCode** with the annotation **@FunctionalInterface**. This interface will have a method named **getDiscount** that takes a promo code as input and returns a double value

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
1 @FunctionalInterface
2 interface Register{
3     void register();
4 }
5
6 @FunctionalInterface
7 interface Login{
8     void login(String email, String password);
9 }
10
11 @FunctionalInterface
12 interface PromoCode{
13     double getDiscountWithPromoCode(String promoCode);
14 }
15
16 class User implements Register{
17     @Override
18     public void register() {
19         System.out.println("User Registered");
20     }
21 }
22
23 public class LambdaExpressions {
24
25     public static void main(String[] args) {
26
27         /*Register ref = new User();
28         ref.register();*/
29
30         /*Register ref = new Register() {
31             public void register() {
32                 System.out.println("User Registered");
33             }
34         };
35         ref.register();*/
36
37     }
38 }

```

- 6.11 Based on the promo code, which you are going to pass as input, you will be giving one discount. This is one of the methods that has a return type. Let's understand how to write a lambda expression for this functional interface. You have the interface known as **PromoCode**, hence write discounts

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
1 @FunctionalInterface
2 interface Register{
3     void register();
4 }
5
6 @FunctionalInterface
7 interface Login{
8     void login(String email, String password);
9 }
10
11 @FunctionalInterface
12 interface PromoCodeDiscount{
13     double getDiscountWithPromoCode(String promoCode);
14 }
15
16 class User implements Register{
17     @Override
18     public void register() {
19         System.out.println("User Registered");
20     }
21 }
22
23 public class LambdaExpressions {
24
25     public static void main(String[] args) {
26
27         /*Register ref = new User();
28         ref.register();*/
29
30         /*Register ref = new Register() {
31             public void register() {
32                 System.out.println("User Registered");
33             }
34         };
35         ref.register();*/
36
37     }
38 }

```

- 6.12 Define the **PromoCode** reference as a Lambda expression that takes a promo code as input and ensure an implicit return statement is present to automatically return a value without explicitly specifying the data type

```

File Edit Source Refactor Navigate Search Project Run Window Help
*LambdaExpressions.java X
25*  public static void main(String[] args) {
26      /*Register ref = new User();
27      ref.register();*/
28
29      /*Register ref = new Register() {
30          public void register() {
31              System.out.println("User Registered");
32          }
33      };
34
35      ref.register();*/
36
37      // Lambda Expression
38      Register ref = ()->{
39          System.out.println("User Registered with Lambda Expression");
40      };
41      ref.register();
42
43
44      //Login loginRef = (String email, String password)-> {
45      Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
46          System.out.println("Connecting to DB");
47          System.out.println("Validating User from DB");
48          if(emailID.equals("admin@example.com") && password.equals("pass123")) {
49              System.out.println("Thank you for Logging in with email "+emailID);
50          }else {
51              System.out.println("Login Failed");
52          }
53      };
54
55      loginRef.login("admin@example.com", "pass123");
56
57      PromoCodeDiscount promoCodeRef = (promoCode) -> {
58
59      };
60
61  }
62

```

- 6.13 Take two inputs, amount and promo code, and check if the promo code is equal to Jumbo. If true, update the amount with a flat 40% off

```

File Edit Source Refactor Navigate Search Project Run Window Help
*LambdaExpressions.java X
27      /*Register ref = new User();
28      ref.register();*/
29
30      /*Register ref = new Register() {
31          public void register() {
32              System.out.println("User Registered");
33          }
34      };
35
36      ref.register();*/
37
38      // Lambda Expression
39      Register ref = ()->{
40          System.out.println("User Registered with Lambda Expression");
41      };
42      ref.register();
43
44      //Login loginRef = (String email, String password)-> {
45      Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
46          System.out.println("Connecting to DB");
47          System.out.println("Validating User from DB");
48          if(emailID.equals("admin@example.com") && password.equals("pass123")) {
49              System.out.println("Thank you for Logging in with email "+emailID);
50          }else {
51              System.out.println("Login Failed");
52          }
53      };
54
55      loginRef.login("admin@example.com", "pass123");
56
57      PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
58          if(promoCode.equals("JUMBO")) {
59              |
60          }
61      };
62

```

- 6.14 Calculate the updated amount by subtracting 40% for the Jumbo promo code, or apply a flat 10% off for other promo codes. Return the updated amount as the result.

```

File Edit Source Refactor Navigate Search Project Run Window Help
# LambdaExpressions.java X
31     public void register() {
32         System.out.println("User Registered");
33     };
34
35     ref.register();*/
36
37     // Lambda Expression
38     Register ref = ()->{
39         System.out.println("User Registered with Lambda Expression");
40     };
41     ref.register();
42
43     //Login loginRef = (String email, String password)-> {
44     Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
45         System.out.println("Conneting to DB");
46         System.out.println("Validating User from DB");
47         if(emailID.equals("admin@example.com") && password.equals("pass123")) {
48             System.out.println("Thank you for Logging in with email "+emailID);
49         }else {
50             System.out.println("Login Failed");
51         }
52     };
53
54     loginRef.login("admin@example.com", "pass123");
55
56     PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
57         if(promoCode.equals("JUMBO")) {
58             amount -= 0.40*amount;
59         }else {
60             amount -= 0.10*amount;
61         }
62         return amount;
63     };
64
65
66 }
67
68

```

Step 7: Execute the Lambda expression with example data

- 7.1 Execute the Lambda expression by assigning the result to the variable **amountToPay** using the **promoCodeReference.getDiscount** method with the promo code **jumbo** for an amount of two thousand. Print **Amount to pay** concatenated with the value of the **amountToPay** variable

```

File Edit Source Refactor Navigate Search Project Run Window Help
# LambdaExpressions.java X
31     public void register() {
32         System.out.println("User Registered");
33     };
34
35     ref.register();*/
36
37     // Lambda Expression
38     Register ref = ()->{
39         System.out.println("User Registered with Lambda Expression");
40     };
41     ref.register();
42
43     //Login loginRef = (String email, String password)-> {
44     Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
45         System.out.println("Conneting to DB");
46         System.out.println("Validating User from DB");
47         if(emailID.equals("admin@example.com") && password.equals("pass123")) {
48             System.out.println("Thank you for Logging in with email "+emailID);
49         }else {
50             System.out.println("Login Failed");
51         }
52     };
53
54     loginRef.login("admin@example.com", "pass123");
55
56     PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
57         if(promoCode.equals("JUMBO")) {
58             amount -= 0.40*amount;
59         }else {
60             amount -= 0.10*amount;
61         }
62         return amount;
63     };
64
65
66     double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "JUMBO");
67     System.out.println("Amount to pay is: "+amountToPay);
68
69

```

7.2 When you run the program, it shows the amount to pay is 1200

The screenshot shows the Eclipse IDE interface with the LambdaExpressions.java file open in the editor. The code implements a registration and login process using Java 8 features like lambdas and method references. In the console, the output shows the user registered, the login attempt, and the final amount to pay, which is 1200.0.

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
31     public void register() {
32         System.out.println("User Registered");
33     }
34     );
35     );
36     );
37     );
38     // Lambda Expression
39     Register ref = ()->{
40         System.out.println("User Registered with Lambda Expression");
41     };
42     ref.register();
43     );
44     );
45     Login loginRef = (String email, String password)-> {
46         Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
47             System.out.println("Connecting to DB");
48             System.out.println("Validating User from DB");
49             if(emailID.equals("admin@example.com") && password.equals("pass123")) {
50                 System.out.println("Thank you for Logging in with email "+emailID);
51             }else {
52                 System.out.println("Login Failed");
53             }
54         };
55         loginRef.login("admin@example.com", "pass123");
56     );
57     PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
58         if(promoCode.equals("JUMBO")) {
59             amount -= 0.40*amount;
60         }else {
61             amount -= 0.10*amount;
62         }
63     );
64     );
65     double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
66     System.out.println("amount to pay is: "+amountToPay);
67

```

Console Output:

```

<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/org
User Registered with Lambda Expression
Connecting to DB
Validating User from DB
Thank you for Logging in with email admin@example.com
amount to pay is: 1200.0

```

7.3 Let's assume that you replace it with bingo and rerun this program. Then it shows the amount to pay is 1800

The screenshot shows the Eclipse IDE interface with the LambdaExpressions.java file open in the editor. The code is identical to the previous version, except the promotional code 'JUMBO' has been replaced with 'BINGO'. In the console, the output shows the user registered, the login attempt, and the final amount to pay, which is 1800.0.

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
31     public void register() {
32         System.out.println("User Registered");
33     }
34     );
35     );
36     );
37     );
38     // Lambda Expression
39     Register ref = ()->{
40         System.out.println("User Registered with Lambda Expression");
41     };
42     ref.register();
43     );
44     );
45     Login loginRef = (String email, String password)-> {
46         Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
47             System.out.println("Connecting to DB");
48             System.out.println("Validating User from DB");
49             if(emailID.equals("admin@example.com") && password.equals("pass123")) {
50                 System.out.println("Thank you for Logging in with email "+emailID);
51             }else {
52                 System.out.println("Login Failed");
53             }
54         };
55         loginRef.login("admin@example.com", "pass123");
56     );
57     PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
58         if(promoCode.equals("BINGO")) {
59             amount -= 0.40*amount;
60         }else {
61             amount -= 0.10*amount;
62         }
63         return amount;
64     );
65     );
66     double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
67     System.out.println("amount to pay is: "+amountToPay);
68

```

Console Output:

```

<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/org
User Registered with Lambda Expression
Connecting to DB
Validating User from DB
Thank you for Logging in with email admin@example.com
amount to pay is: 1800.0

```

7.4 Optimize the code using the ternary operator: Calculate the **finalAmount** based on whether the promo code is jumbo or not, returning the amount minus 40% for jumbo or amount minus 10% otherwise, resulting in a single line of code

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java
40     System.out.println("User Registered with Lambda Expression");
41 };
42 ref.register();
43
44 //Login loginRef = (String email, String password)-> {
45 Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
46     System.out.println("Connecting to DB");
47     System.out.println("Validating User from DB");
48     if(emailID.equals("admin@example.com") && password.equals("pass123")) {
49         System.out.println("Thank you for Logging in with email "+emailID);
50     }else {
51         System.out.println("Login Failed");
52     }
53 }
54
55 loginRef.login("admin@example.com", "pass123");
56
57 /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
58     if(promoCode.equals("JUMBO")) {
59         amount -= 0.40*amount;
60     }else {
61         amount -= 0.10*amount;
62     }
63     return amount;
64 };*/
65
66 PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
67
68     double finalAmount = promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
69
70     if(promoCode.equals("JUMBO")) {
71         amount -= 0.40*amount;
72     }else {
73         amount -= 0.10*amount;
74     }
75     return amount;
76 };
77

```

7.5 Conclude the optimized code by returning the **finalAmount**. This version utilizes the ternary operator to reduce the previous code and provides a 10% discount by default, while the jumbo promo code gives a 40% discount

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java
40     System.out.println("User Registered with Lambda Expression");
41 };
42 ref.register();
43
44 //Login loginRef = (String email, String password)-> {
45 Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
46     System.out.println("Connecting to DB");
47     System.out.println("Validating User from DB");
48     if(emailID.equals("admin@example.com") && password.equals("pass123")) {
49         System.out.println("Thank you for Logging in with email "+emailID);
50     }else {
51         System.out.println("Login Failed");
52     }
53 }
54
55 loginRef.login("admin@example.com", "pass123");
56
57 /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
58     if(promoCode.equals("JUMBO")) {
59         amount -= 0.40*amount;
60     }else {
61         amount -= 0.10*amount;
62     }
63     return amount;
64 };*/
65
66 PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
67
68     double finalAmount = promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
69     return finalAmount;
70 };
71

```

7.6 The benefit of using the ternary operator is that it allows for a more concise and compact Lambda expression implementation, resulting in code that works in the same way when executed

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java | Problems Javadoc Declaration Console
39     Register ref = ()->{
40         System.out.println("User Registered with Lambda Expression");
41     };
42     ref.register();
43
44     //Login loginRef = (String email, String password)-> {
45     Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
46         System.out.println("Connecting to DB");
47         System.out.println("Validating User from DB");
48         if(emailID.equals("admin@example.com") && password.equals("pass123")) {
49             System.out.println("Thank you for Logging in with email "+emailID);
50         }else {
51             System.out.println("Login Failed");
52         }
53     };
54
55     loginRef.login("admin@example.com", "pass123");
56
57     /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
58         if(promoCode.equals("JUMBO")) {
59             amount -= 0.40*amount;
60         }else {
61             amount -= 0.10*amount;
62         }
63         return amount;
64     }*/
65
66     PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
67         return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
68     };
69
70
71
72
73     double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "JUMBO");
74     System.out.println("amount to pay is: "+amountToPay);
75

```

<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/or
User Registered with Lambda Expression
Connecting to DB
Validating User from DB
Thank you for Logging in with email admin@example.com
amount to pay is: 1200.0

7.7 Now finally, let us make this lambda expression even simpler. Copy this part and paste it here. This will be directly the return statement, demonstrating how you can have a single line of code for the lambda expression

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java | Problems Javadoc Declaration Console
39     Register ref = ()->{
40         System.out.println("User Registered with Lambda Expression");
41     };
42     ref.register();
43
44     //Login loginRef = (String email, String password)-> {
45     Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
46         System.out.println("Connecting to DB");
47         System.out.println("Validating User from DB");
48         if(emailID.equals("admin@example.com") && password.equals("pass123")) {
49             System.out.println("Thank you for Logging in with email "+emailID);
50         }else {
51             System.out.println("Login Failed");
52         }
53     };
54
55     loginRef.login("admin@example.com", "pass123");
56
57     /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
58         if(promoCode.equals("JUMBO")) {
59             amount -= 0.40*amount;
60         }else {
61             amount -= 0.10*amount;
62         }
63         return amount;
64     }*/
65
66     PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
67         return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
68     };
69
70     PromoCodeDiscount promoCodeRef = (amount, promoCode) -> promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
71
72
73     double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "JUMBO");
74     System.out.println("amount to pay is: "+amountToPay);
75

```

7.8 Run the same code, and you will see the correct output for jumbo and bingo. You will notice that it will provide a lesser discount. Hence, this is what lambda expressions are all about

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
39     Register ref = ()->{
40         System.out.println("User Registered with Lambda Expression");
41     };
42     ref.register();
43
44     //Login loginRef = (String email, String password)-> {
45     Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
46         System.out.println("Connecting to DB");
47         System.out.println("Validating User from DB");
48         if(emailID.equals("admin@example.com") && password.equals("pass123")) {
49             System.out.println("Thank you for Logging in with email "+emailID);
50         } else {
51             System.out.println("Login Failed");
52         }
53     };
54
55     loginRef.login("admin@example.com", "pass123");
56
57     /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
58         if(promoCode.equals("JUMBO")) {
59             amount -= 0.40*amount;
60         } else {
61             amount -= 0.10*amount;
62         }
63         return amount;
64     };*/
65
66     /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
67         return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
68     };*/

```

Problems Javadoc Declaration Console X

<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/org

User Registered with Lambda Expression
Connecting to DB
Validating User from DB
Thank you for Logging in with email admin@example.com
amount to pay is: 1800.0

7.9 Let us understand one of the examples of lambda expressions with the collection framework. Assume you have an **ArrayList** of type String and this is promo codes as a new **ArrayList**. Inside the promo codes, add BINGO. Then add JUMBO, HUNGRY50, THANKS, and New100

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
49     System.out.println("Validating User from DB");
50     if(emailID.equals("admin@example.com") && password.equals("pass123")) {
51         System.out.println("Thank you for Logging in with email "+emailID);
52     } else {
53         System.out.println("Login Failed");
54     }
55
56     loginRef.login("admin@example.com", "pass123");
57
58     /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
59         if(promoCode.equals("JUMBO")) {
60             amount -= 0.40*amount;
61         } else {
62             amount -= 0.10*amount;
63         }
64         return amount;
65     };*/
66
67     /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
68         return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
69     };*/
70
71     PromoCodeDiscount promoCodeRef = (amount, promoCode) -> promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
72
73     double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
74     System.out.println("amount to pay is: "+amountToPay);
75
76     ArrayList<String> promoCodes = new ArrayList<String>();
77     promoCodes.add("BINGO");
78     promoCodes.add("JUMBO");
79     promoCodes.add("HUNGRY50");
80     promoCodes.add("THANKS");
81     promoCodes.add("New100");
82
83 }
84
85
86 }

```

- 7.10 Java 8 introduces the usage of lambda expressions as a feature, such as with the **forEach** method on a list of promo codes, where a consumer lambda expression can be passed to print each code using **System.out.println**.

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
54     }
55   };
56 
57   loginRef.login("admin@example.com", "pass123");
58 
59   /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
60     if(promoCode.equals("JUMBO")) {
61       amount -= 0.40*amount;
62     }else {
63       amount -= 0.10*amount;
64     }
65     return amount;
66  };*/
67 
68   /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
69     return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
70  };*/
71 
72   PromoCodeDiscount promoCodeRef = (amount, promoCode) -> promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
73 
74   double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
75   System.out.println("amount to pay is: "+amountToPay);
76 
77   ArrayList<String> promoCodes = new ArrayList<String>();
78   promoCodes.add("BINGO");
79   promoCodes.add("JUMBO");
80   promoCodes.add("HUNGRY50");
81   promoCodes.add("THANKS");
82   promoCodes.add("NEW100");
83 
84   promoCodes.forEach(
85     (code) -> System.out.println(code)
86   );
87 
88 }
89 
90 }

```

- 7.11 Run this code and that is how you can iterate through all your collections. Lambda expressions can be very useful when working in a multithreaded environment

```

File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java X
54     }
55   };
56 
57   loginRef.login("admin@example.com", "pass123");
58 
59   /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
60     if(promoCode.equals("JUMBO")) {
61       amount -= 0.40*amount;
62     }else {
63       amount -= 0.10*amount;
64     }
65     return amount;
66  };*/
67 
68   /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
69     return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
70  };*/
71 
72   PromoCodeDiscount promoCodeRef = (amount, promoCode) -> promoCode.equals("JUMBO") ?
73 
74   double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
75   System.out.println("amount to pay is: "+amountToPay);
76 
77   ArrayList<String> promoCodes = new ArrayList<String>();
78   promoCodes.add("BINGO");
79   promoCodes.add("JUMBO");
80   promoCodes.add("HUNGRY50");
81   promoCodes.add("THANKS");
82   promoCodes.add("NEW100");
83 
84   promoCodes.forEach(
85     (code) -> System.out.println(code)
86   );
87 
88 }
89 
90 }

```

<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/or
User Registered with Lambda Expression
Connecting to DB
Validating User from DB
Thank you for Logging in with email admin@example.com
amount to pay is: 1800.0
BINGO
JUMBO
HUNGRY50
THANKS
NEW100

- 7.12 The general way of writing a thread is as shown: you can write as Runnable, the reference is a new instance of Runnable, and then using the anonymous class concept, you will be overriding the run method. Here, you will be uploading a profile picture. This is one of the general ways to create a thread

```

File Edit Source Refactor Navigate Search Project Run Window Help
* LambdaExpressions.java X
62         }else {
63             amount -= 0.10*amount;
64         }
65     return amount;
66 }
67 */
68 /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
69     return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
70 }*/
71 PromoCodeDiscount promoCodeRef = (amount, promoCode) -> promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
72
73 double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
74 System.out.println("amount to pay is: "+amountToPay);
75
76 ArrayList<String> promoCodes = new ArrayList<String>();
77 promoCodes.add("BINGO");
78 promoCodes.add("JUMBO");
79 promoCodes.add("HUNGRY50");
80 promoCodes.add("THANKS");
81 promoCodes.add("NEW100");
82
83 promoCodes.forEach(
84     (code) -> System.out.println(code)
85 );
86
87 /*Runnable runnable = new Runnable() {
88     public void run() {
89         System.out.println("Uploading Profile Picture...");
90     }
91 };
92 */
93
94 Runnable runnable = () -> {
95     System.out.println("Uploading Profile Picture...");
96 };
97
98 new Thread(runnable).start();
99
100
101
102
103

```

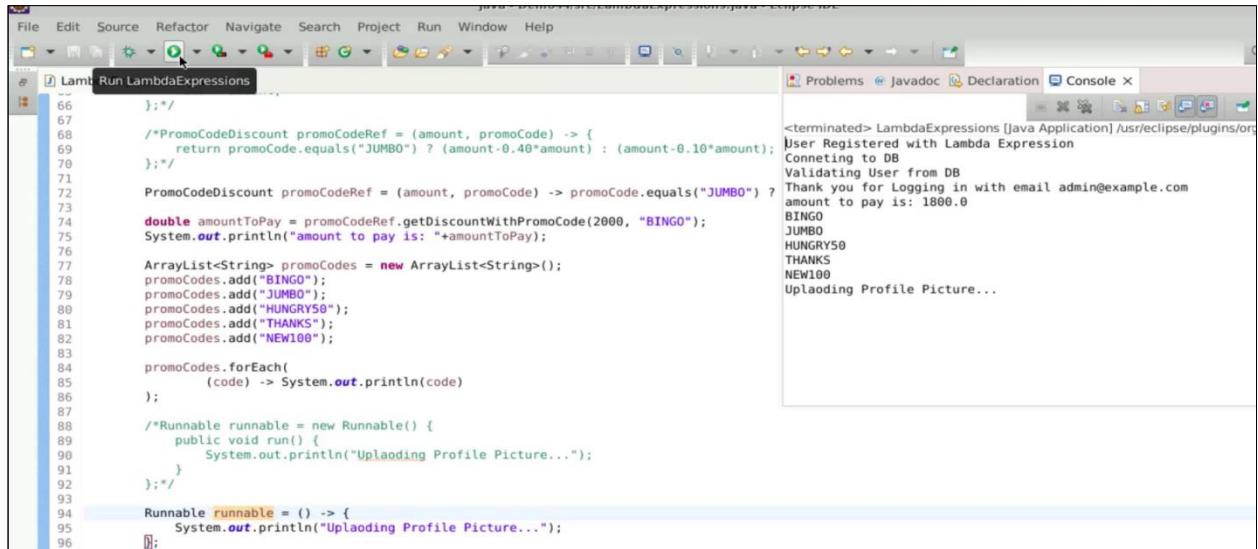
- 7.13 To convert the code to a lambda expression, define a runnable lambda expression with the implementation of the run method, printing and uploading profile picture. Then create a new thread object with this runnable as input

```

File Edit Source Refactor Navigate Search Project Run Window Help
* LambdaExpressions.java X
66 */
67
68 /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
69     return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
70 }*/
71 PromoCodeDiscount promoCodeRef = (amount, promoCode) -> promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
72
73 double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
74 System.out.println("amount to pay is: "+amountToPay);
75
76 ArrayList<String> promoCodes = new ArrayList<String>();
77 promoCodes.add("BINGO");
78 promoCodes.add("JUMBO");
79 promoCodes.add("HUNGRY50");
80 promoCodes.add("THANKS");
81 promoCodes.add("NEW100");
82
83 promoCodes.forEach(
84     (code) -> System.out.println(code)
85 );
86
87 /*Runnable runnable = new Runnable() {
88     public void run() {
89         System.out.println("Uploading Profile Picture...");
90     }
91 };
92 */
93
94 Runnable runnable = () -> {
95     System.out.println("Uploading Profile Picture...");
96 };
97
98 new Thread(runnable).start();
99
100
101
102
103

```

7.14 Thus, using lambda expressions can change the way you code, ensuring that lambda expressions are used for implementing functional interfaces. You can't create a lambda expression for an interface with multiple methods



The screenshot shows the Eclipse IDE interface with a Java file named `LambdaExpressions.java` open. The code demonstrates the use of lambda expressions for functional interfaces. The console output shows the execution results.

```
File Edit Source Refactor Navigate Search Project Run Window Help
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console X
<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/org
User Registered with Lambda Expression
Connecting to DB
Validating User from DB
Thank you for Logging in with email admin@example.com
amount to pay is: 1800.0
BINGO
JUMBO
HUNGRY50
THANKS
NEW100
Uploading Profile Picture...

```

```
66     };*/
67
68     /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
69         return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
70     };*/
71
72     PromoCodeDiscount promoCodeRef = (amount, promoCode) -> promoCode.equals("JUMBO") ? 
73
73     double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
74     System.out.println("amount to pay is: "+amountToPay);
75
76
77     ArrayList<String> promoCodes = new ArrayList<String>();
78     promoCodes.add("BINGO");
79     promoCodes.add("JUMBO");
80     promoCodes.add("HUNGRY50");
81     promoCodes.add("THANKS");
82     promoCodes.add("NEW100");
83
84     promoCodes.forEach(
85         (code) -> System.out.println(code)
86     );
87
88     /*Runnable runnable = new Runnable() {
89         public void run() {
90             System.out.println("Uploading Profile Picture...");
91         }
92     };*/
93
94     Runnable runnable = () -> {
95         System.out.println("Uploading Profile Picture...");
96     };

```

By following these steps, you have successfully implemented Lambda expressions in Java along with the creation of functional interfaces.