

## Lesson 06 Demo 06

### Implementing Method References in Java

**Objective:** To implement method references in Java, including creating functional interfaces and executing code with example data.

**Tools Required:** Eclipse IDE

**Prerequisites:** None

Steps to be followed:

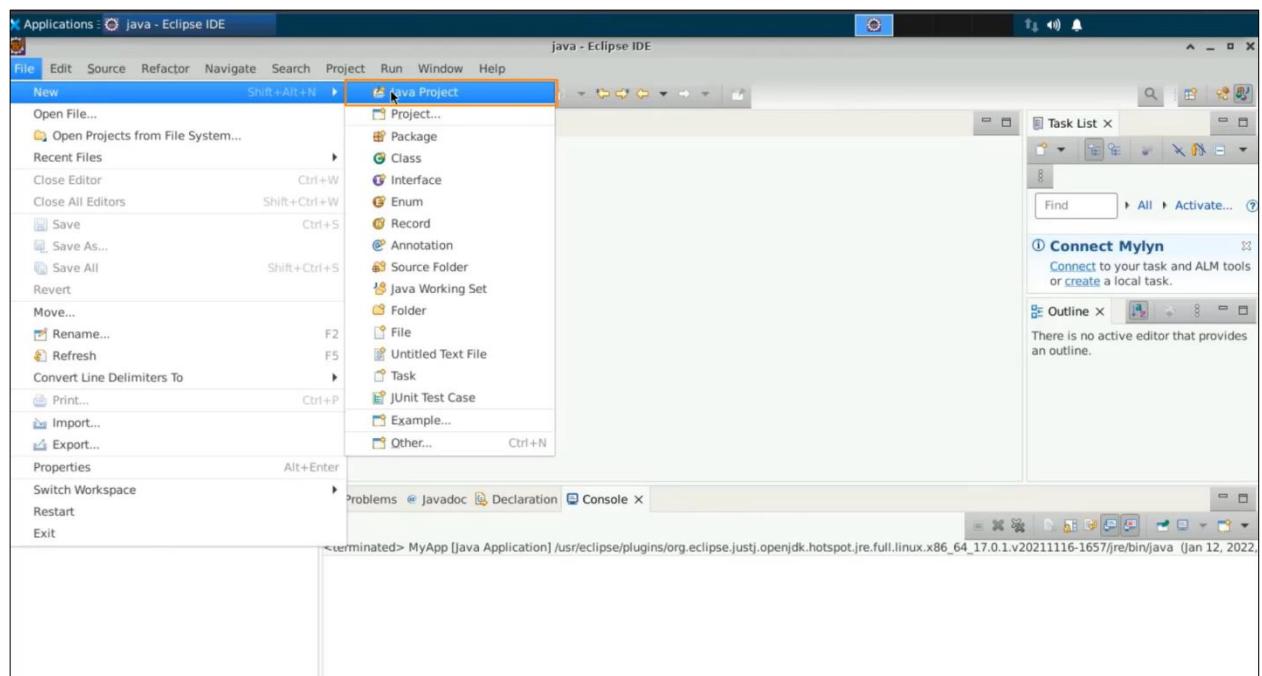
1. Create a Java project
2. Create a functional interface and a class for a static void registered user
3. Create Lambda expressions
4. Create a reference to the interface and execute the code
5. Execute the log in reference
6. Create methods that can do a return, and execute the code with example data
7. Write the reference variable notification and execute the code

## Step 1: Create a Java project

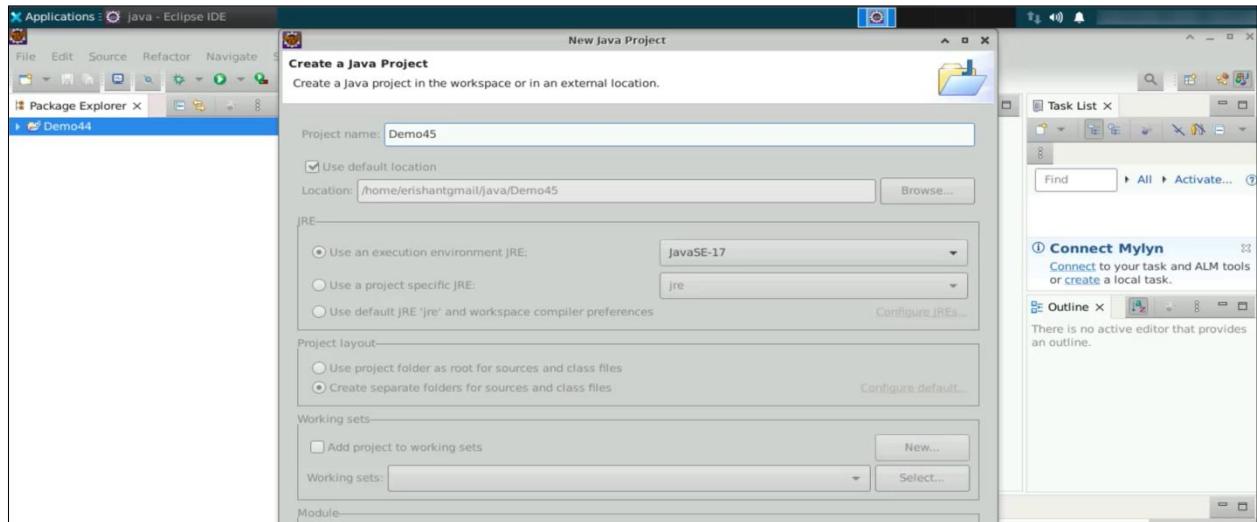
### 1.1 Open the Eclipse IDE



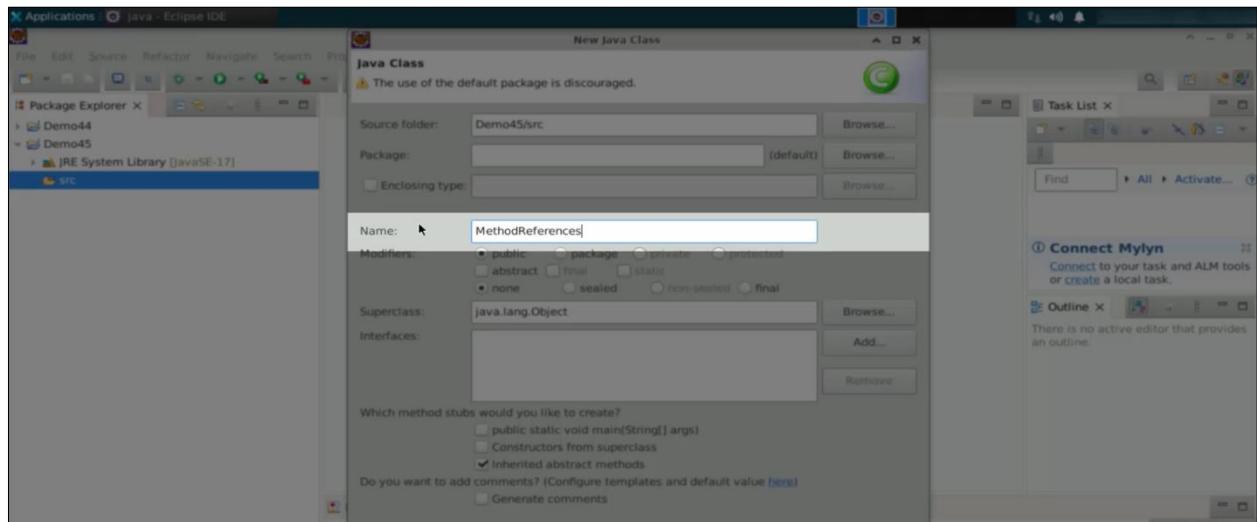
### 1.2 Select File, then New, and click Java project



1.3 Enter the name of the project as "**Demo45**", uncheck "**Create a module-info.java file**", and click **Finish**

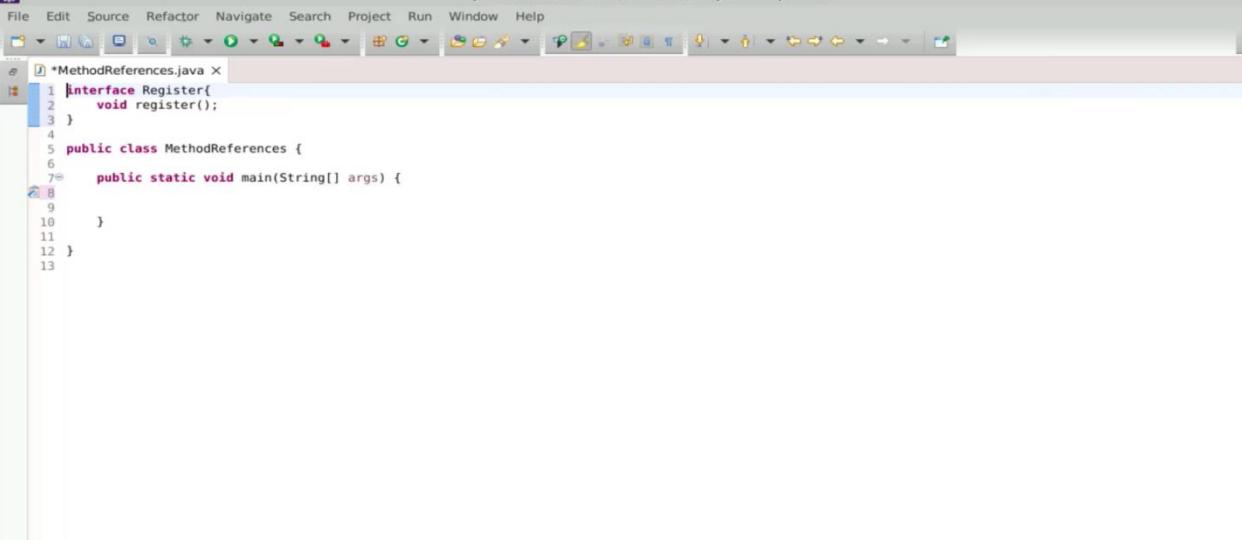


1.4 With **Demo45** selected in the **src** folder, right-click and create a new class. Name this class **MethodReferences**, then select the main method, and click **Finish**



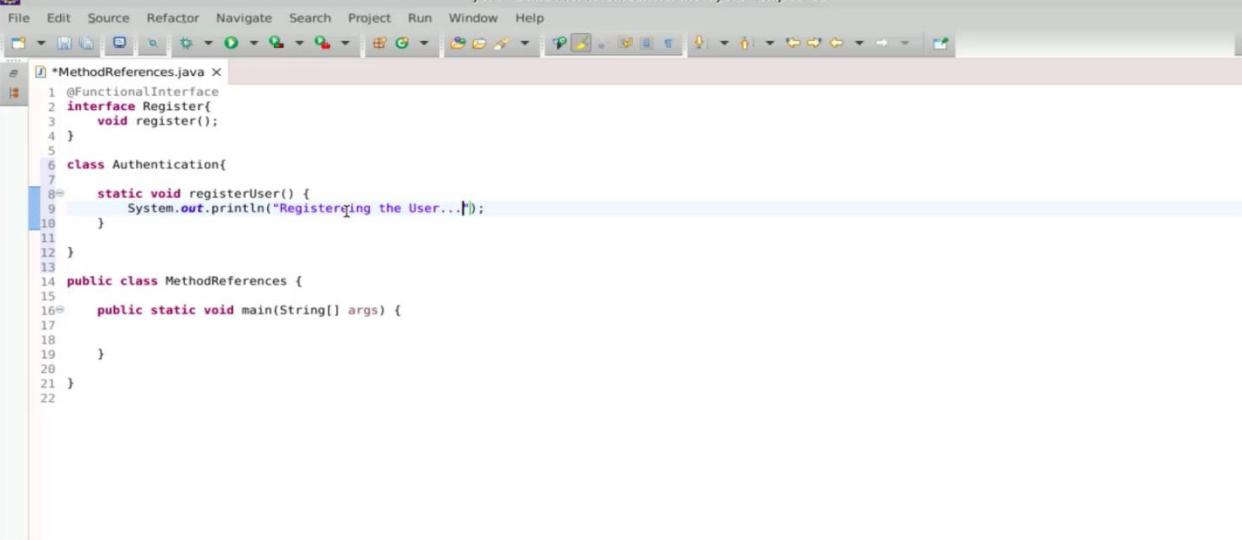
## Step 2: Create a functional interface and a class for a static void registered user

2.1 Create a functional interface called **RegisteredInterface**. Annotate it with the **@FunctionalInterface** annotation. Add a method with the name register



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

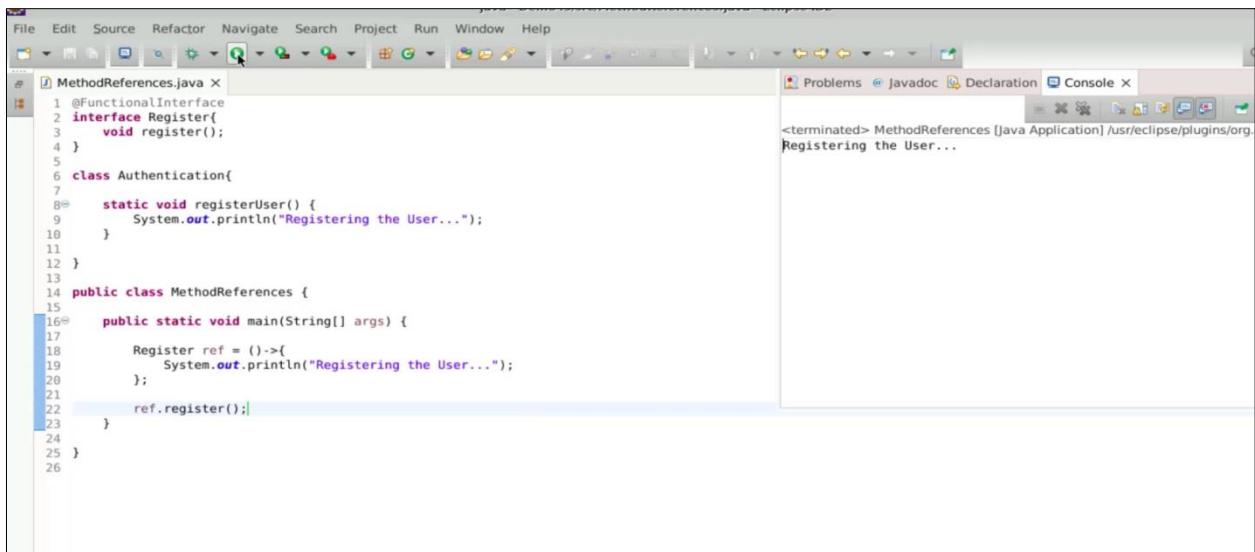
2.2 Create a class and name it Authentication Class. Implement a static void method and name it **registeredUser**, which prints Registering the user



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

## Step 3: Create Lambda expressions

3.1 In the main method, for the **register** method, you will create lambda expressions. This means that you can have the method definition **registerUser**, and then execute this method with the name **register** on the object. This is how lambda expressions are used, as you have done earlier



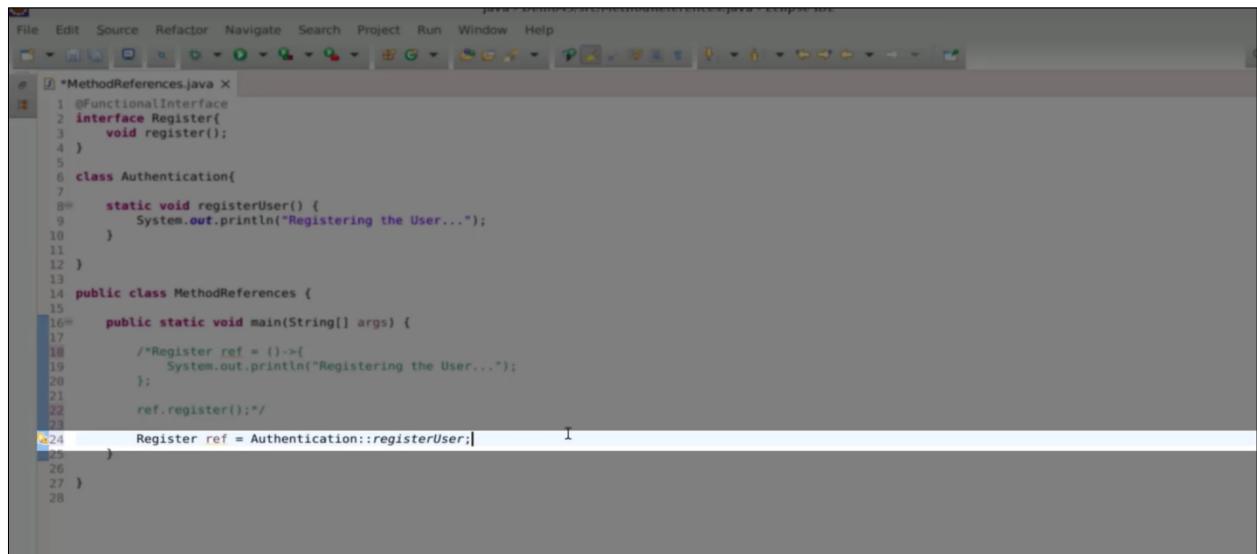
```
File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferences.java X
1 @FunctionalInterface
2 interface Register{
3     void register();
4 }
5
6 class Authentication{
7
8     static void registerUser() {
9         System.out.println("Registering the User...");
10    }
11
12 }
13
14 public class MethodReferences {
15
16     public static void main(String[] args) {
17
18         Register ref = ()->{
19             System.out.println("Registering the User...");
20         };
21
22         ref.register();
23     }
24
25 }
26
```

Problems Javadoc Declaration Console X

<terminated> MethodReferences [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.core.prefs Registering the User...

## Step 4: Create a reference to the interface and execute the code

4.1 Next, replace the lambda expression with a static method reference. Create the reference to the interface and then with the class **AuthenticationClass** using the method reference operator (`::`). Assign the reference of the **registerUser** method. This is static method referencing.

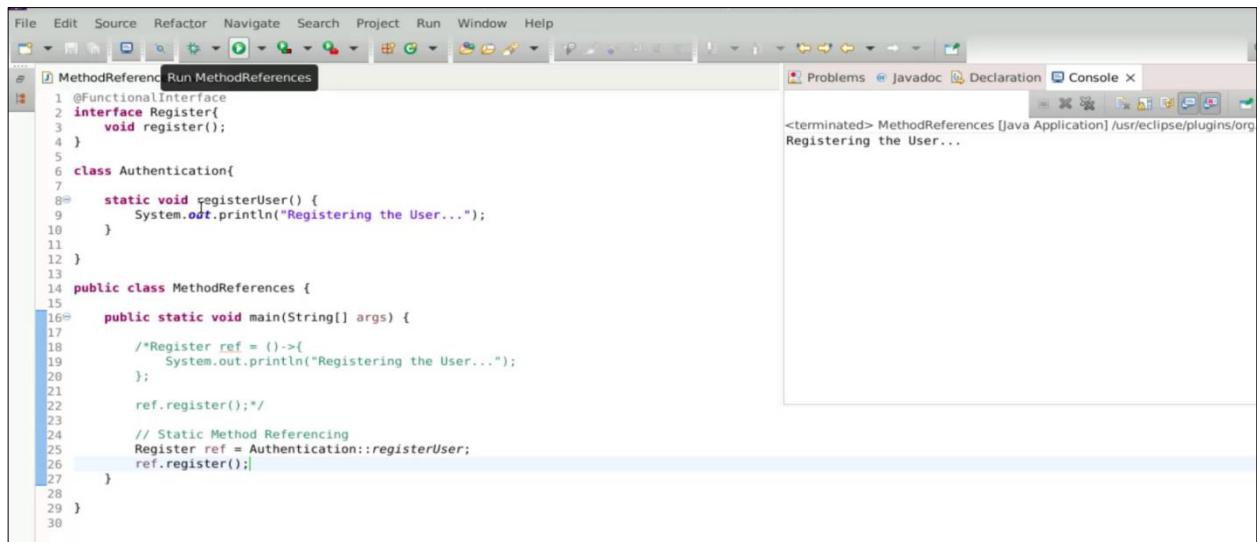


```

File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferences.java ×
1 @FunctionalInterface
2 interface Register{
3     void register();
4 }
5
6 class Authentication{
7
8     static void registerUser() {
9         System.out.println("Registering the User...");
10    }
11
12 }
13
14 public class MethodReferences {
15
16     public static void main(String[] args) {
17
18         /*Register ref = ()->{
19             System.out.println("Registering the User...");
20         };
21
22         ref.register();*/
23
24         Register ref = Authentication::registerUser;
25
26     }
27 }
28
29
30

```

### 4.2 Execute the `register` method



```

File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferenceRun MethodReferences ×
MethodReferences [Java Application] /usr/eclipse/plugins/org
Registering the User...

```

```

File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferenceRun MethodReferences ×
MethodReferences [Java Application] /usr/eclipse/plugins/org
Registering the User...

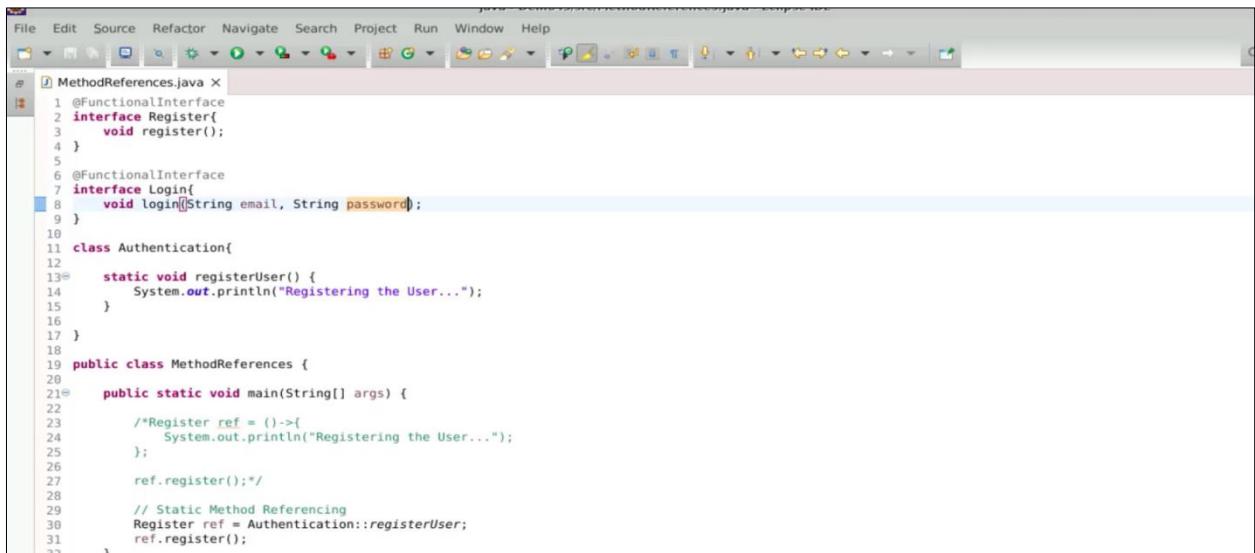
```

```

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

```

4.3 If you want an interface with input parameters, create an interface and name it **Login** with a **logIn** method that takes two string parameters: email and password

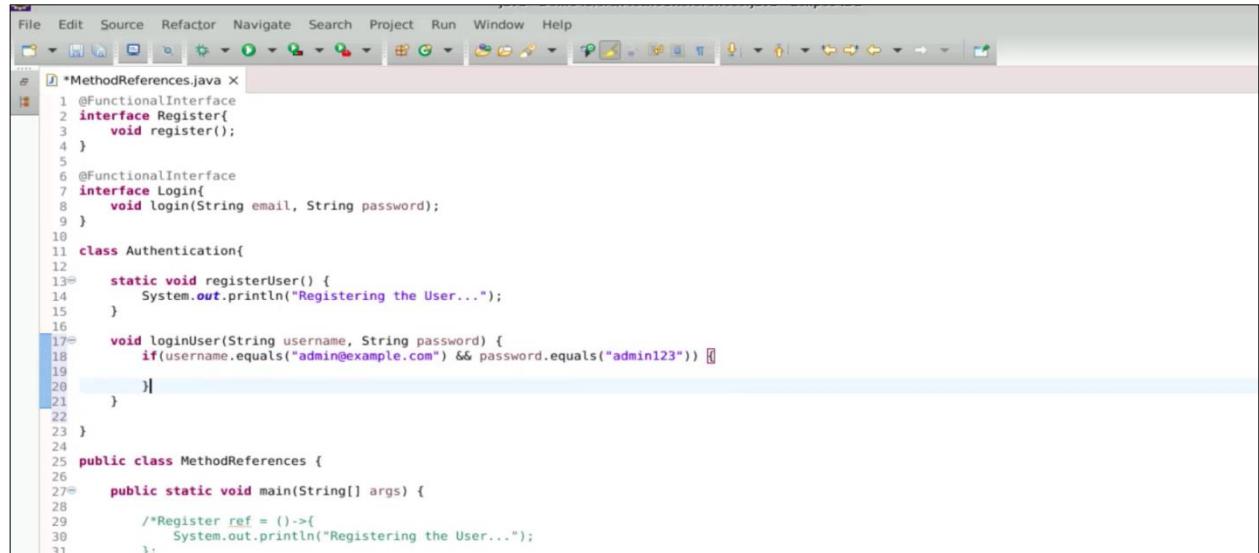


```

File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferences.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 class Authentication{
12
13    static void registerUser() {
14        System.out.println("Registering the User...");
15    }
16
17 }
18
19 public class MethodReferences {
20
21    public static void main(String[] args) {
22
23        /*Register ref = ()->{
24            System.out.println("Registering the User...");
25        };
26
27        ref.register();*/
28
29        // Static Method Referencing
30        Register ref = Authentication::registerUser;
31        ref.register();
32    }

```

4.4 In the **AuthenticationClass**, create a non-static method called **logInUser** that takes a username and password as inputs



```

File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferences.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 class Authentication{
12
13    static void registerUser() {
14        System.out.println("Registering the User...");
15    }
16
17    void logInUser(String username, String password) {
18        if(username.equals("admin@example.com") && password.equals("admin123")) {
19
20        }
21    }
22
23 }
24
25 public class MethodReferences {
26
27    public static void main(String[] args) {
28
29        /*Register ref = ()->{
30            System.out.println("Registering the User...");
31        };*/
32    }

```

4.5 In the else block, you can print **login is successful** or **login failed**. This is the method definition you created for the **logInUser** method in the **AuthenticationClass**

```

File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferences.java X
6 @FunctionalInterface
7 interface Login{
8     void login(String email, String password);
9 }
10 class Authentication{
11     static void registerUser() {
12         System.out.println("Registering the User...");
13     }
14     void logInUser(String username, String password) {
15         if(username.equals("admin@example.com") && password.equals("admin123")) {
16             System.out.println("Log In Successfull..");
17         }else {
18             System.out.println("Log In Failed");
19         }
20     }
21 }
22 public class MethodReferences {
23     public static void main(String[] args) {
24         /*Register ref = ()->{
25             System.out.println("Registering the User...");
26         };
27         ref.register();*/
28     }
29 }
30 
```

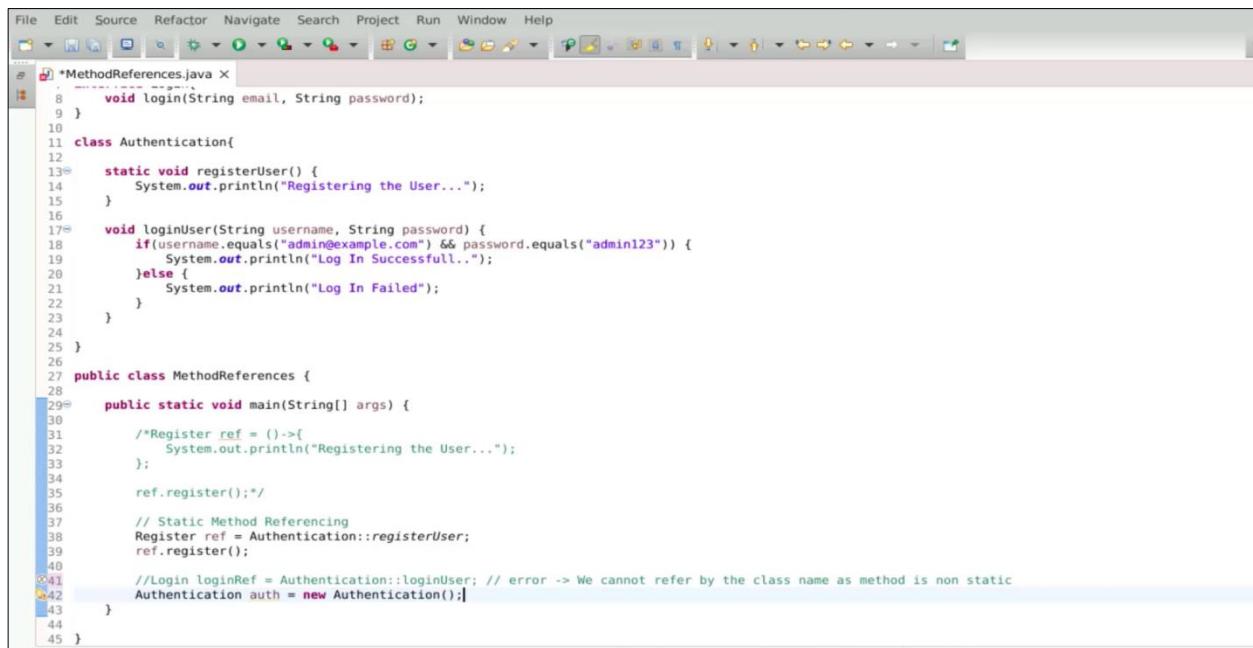
4.6 Create a reference variable **loginRef** to refer to the **logInUser** method of the **Authentication** class. You need to use the object reference to access non-static methods

```

File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferences.java X
7 interface Login{
8     void login(String email, String password);
9 }
10 class Authentication{
11     static void registerUser() {
12         System.out.println("Registering the User...");
13     }
14     void logInUser(String username, String password) {
15         if(username.equals("admin@example.com") && password.equals("admin123")) {
16             System.out.println("Log In Successfull..");
17         }else {
18             System.out.println("Log In Failed");
19         }
20     }
21 }
22 public class MethodReferences {
23     public static void main(String[] args) {
24         /*Register ref = ()->{
25             System.out.println("Registering the User...");
26         };
27         ref.register();*/
28     }
29 }
30 
```

4.7 Create an object of the **Authentication** class, as shown:

```
Authentication auth = new Authentication()
```

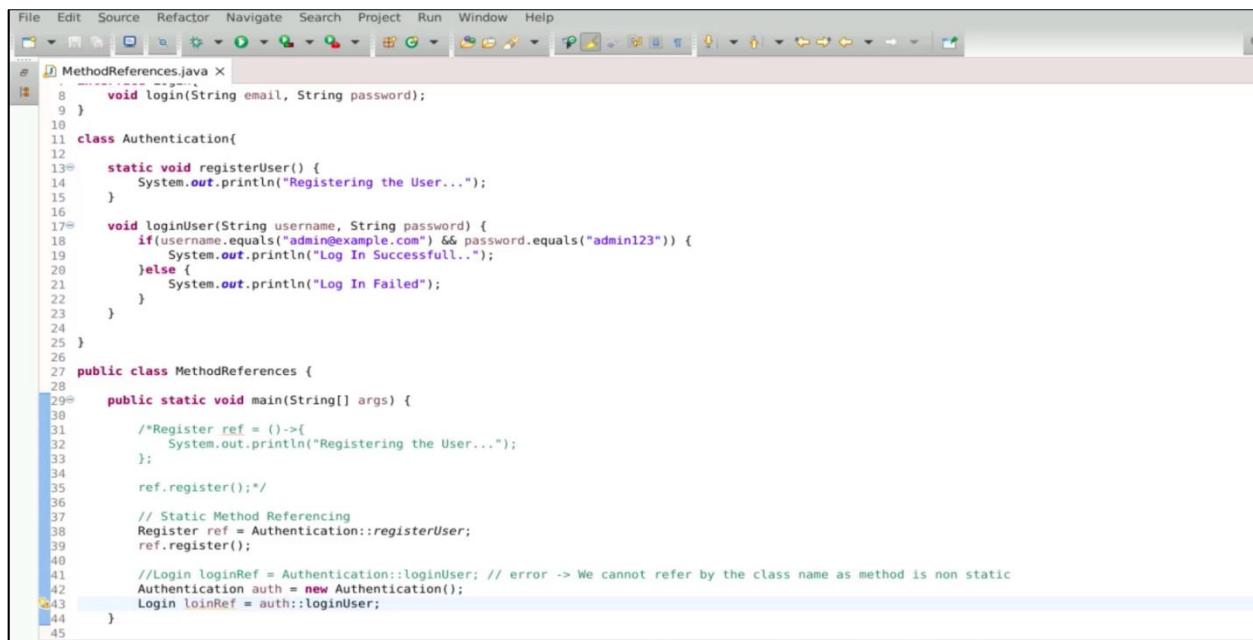


```

File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferences.java X
8     void login(String email, String password);
9 }
10
11 class Authentication{
12
13     static void registerUser() {
14         System.out.println("Registering the User...");
15     }
16
17     void loginUser(String username, String password) {
18         if(username.equals("admin@example.com") && password.equals("admin123")) {
19             System.out.println("Log In Successful..");
20         }else {
21             System.out.println("Log In Failed");
22         }
23     }
24
25 }
26
27 public class MethodReferences {
28
29     public static void main(String[] args) {
30
31         /*Register ref = ()->{
32             System.out.println("Registering the User...");
33         };
34
35         ref.register();*/
36
37         // Static Method Referencing
38         Register ref = Authentication::registerUser;
39         ref.register();
40
41         //Login loginRef = Authentication::loginUser; // error -> We cannot refer by the class name as method is non static
42         Authentication auth = new Authentication();
43
44     }
45 }

```

4.8 Now that you have created the object of the **AuthenticationClass**, you can proceed with method referencing



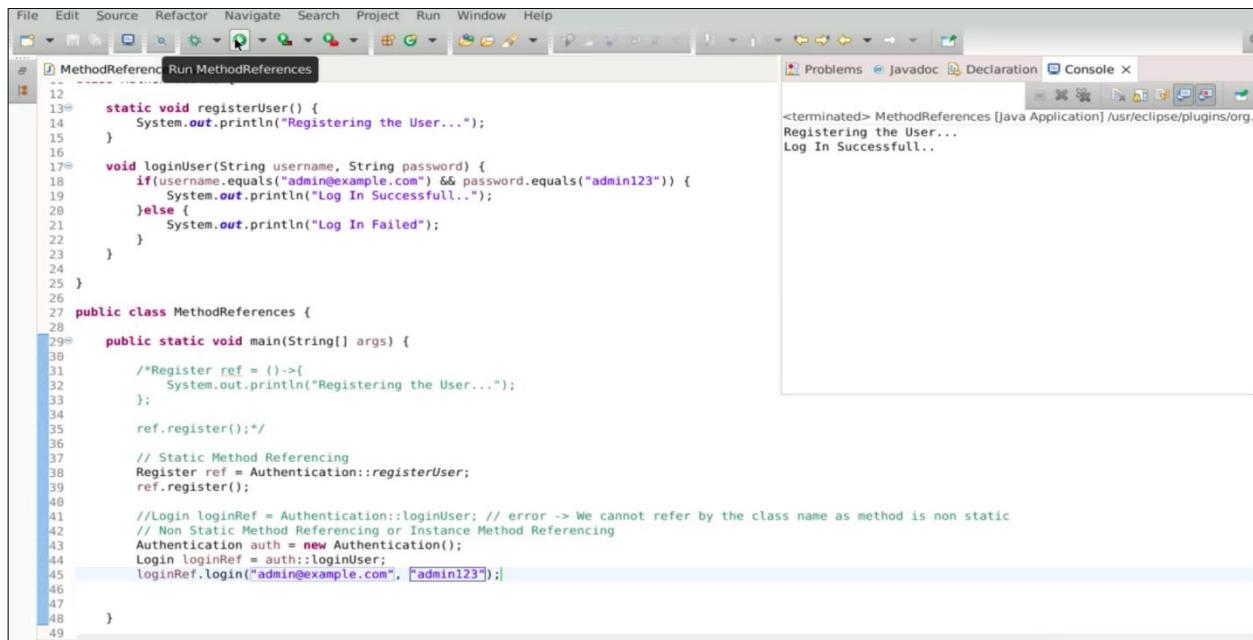
```

File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferences.java X
8     void login(String email, String password);
9 }
10
11 class Authentication{
12
13     static void registerUser() {
14         System.out.println("Registering the User...");
15     }
16
17     void loginUser(String username, String password) {
18         if(username.equals("admin@example.com") && password.equals("admin123")) {
19             System.out.println("Log In Successful..");
20         }else {
21             System.out.println("Log In Failed");
22         }
23     }
24
25 }
26
27 public class MethodReferences {
28
29     public static void main(String[] args) {
30
31         /*Register ref = ()->{
32             System.out.println("Registering the User...");
33         };
34
35         ref.register();*/
36
37         // Static Method Referencing
38         Register ref = Authentication::registerUser;
39         ref.register();
40
41         //Login loginRef = Authentication::loginUser; // error -> We cannot refer by the class name as method is non static
42         Authentication auth = new Authentication();
43         Login loginRef = auth::loginUser;
44     }
45 }

```

## Step 5: Execute the log in reference

5.1 Execute **loginRef**. On the login reference, execute the method **login**. Pass the email **admin@example.com** and the password **admin123**. You will observe that the login is successful



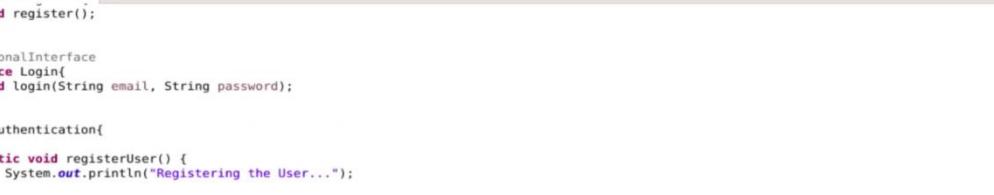
The screenshot shows the Eclipse IDE interface with the following details:

- File Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard icons for file operations like Open, Save, Print, etc.
- Left Margin:** Shows line numbers from 12 to 49.
- Code Editor:** Displays Java code for `MethodReferences` class. It includes static methods `registerUser` and `loginUser`, and a main method that creates a `Register` reference and calls `register` and `login` methods.
- Console View:** Shows the output of the application's execution:

```
<terminated> MethodReferences [java Application] /usr/eclipse/plugins/org.eclips...  
Registering the User...  
Log In Successfull..
```

## **Step 6: Create methods that can return values, and execute the code with example data**

6.1 Create a functional interface called Taxes. Add a method with the name **getFinalAmount** that takes an amount as input. Implement a method with the name **getAmountToPay** in the Taxes interface, which returns the amount plus 18% of the amount



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

```
File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferences.java X
3     void register();
4 }
5
6 @FunctionalInterface
7 interface Login{
8     void login(String email, String password);
9 } I
10 class Authentication{
11
12     static void registerUser() {
13         System.out.println("Registering the User...");
14     }
15
16     void loginUser(String username, String password) {
17         if(username.equals("admin@example.com") && password.equals("admin123")) {
18             System.out.println("Log In Successfull..");
19         }else {
20             System.out.println("Log In Failed");
21         }
22     }
23 }
24
25     double getFinalAmount(int amount) {
26         return amount+(0.18*amount);
27     }
28
29 }
30
31 public class MethodReferences {
32
33     public static void main(String[] args) {
```

6.2 Modify the `getAmountToPay` method to take a double as the input instead of an integer.

```
MethodReferences.java X
1  package com.example;
2
3  public class MethodReferences {
4      void register();
5
6      @FunctionalInterface
7      interface Login{
8          void login(String email, String password);
9      }
10     interface Taxes{
11         double getAmountToPay(double amount);
12     }
13 }
14
15 class Authentication{
16
17     static void registerUser() {
18         System.out.println("Registering the User...");
19     }
20
21     void loginUser(String username, String password) {
22         if(username.equals("admin@example.com") && password.equals("admin123")) {
23             System.out.println("Log In Successfull..");
24         }else {
25             System.out.println("Log In Failed");
26         }
27     }
28
29     double getFinalAmount(double amount) {
30         return amount+(0.18*amount);
31     }
32
33 }
```

6.3 Create a reference variable of type Taxes. Assign the reference of the `getFinalAmount` method to the reference variable. Use `tRef.getAmountToPay(1200)` to calculate the final amount to pay after applying 18% taxes to 1200

```

File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferences.java X
22     if(username.equals("admin@example.com") && password.equals("admin123")) {
23         System.out.println("Log In Successfull..");
24     }else {
25         System.out.println("Log In Failed");
26     }
27 }
28
29=     double getFinalAmount(double amount) {
30         return amount+(0.18*amount);
31     }
32
33 }
34
35 public class MethodReferences {
36
37=     public static void main(String[] args) {
38
39         /*Register ref = ()->{
40             System.out.println("Registering the User...");
41         };
42
43         ref.register();*/
44
45         // Static Method Referencing
46         Register ref = Authentication::registerUser;
47         ref.register();
48
49         //Login loginRef = Authentication::loginUser; // error -> We cannot refer by the class name as method is non static
50         // Non Static Method Referencing or Instance Method Referencing
51         Authentication auth = new Authentication();
52         Login loginRef = auth::loginUser;
53         loginRef.login("admin@example.com", "admin123");
54
55         Taxes tRef = auth::getFinalAmount;
56         System.out.println("Final Amount to Pay for 1200 is: "+tRef.getAmountToPay(1200));
57
58     }
59 }

```

6.4 When you run the code, it should display **1416** as the final amount after applying taxes

```

File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferences.java X
22     if(username.equals("admin@example.com") && password.equals("admin123")) {
23         System.out.println("Log In Successfull..");
24     }else {
25         System.out.println("Log In Failed");
26     }
27 }
28
29=     double getFinalAmount(double amount) {
30         return amount+(0.18*amount);
31     }
32
33 }
34
35 public class MethodReferences {
36
37=     public static void main(String[] args) {
38
39         /*Register ref = ()->{
40             System.out.println("Registering the User...");
41         };
42
43         ref.register();*/
44
45         // Static Method Referencing
46         Register ref = Authentication::registerUser;
47         ref.register();
48
49         //Login loginRef = Authentication::loginUser; // error -> We cannot refer by the class name as method is non static
50         // Non Static Method Referencing or Instance Method Referencing
51         Authentication auth = new Authentication();
52         Login loginRef = auth::loginUser;
53
54         System.out.println("Registering the User...");
55         System.out.println("Log In Successfull..");
56         System.out.println("Final Amount to Pay for 1200 is: 1416.0");
57
58     }
59 }

```

## 6.5 Another way to do method referencing is by creating an object of the **Authentication** class and immediately referencing the **getFinalAmount** method of the Taxes interface

```

1  package com.simplilearn;
2
3  public class MethodReferences {
4
5      public static void main(String[] args) {
6
7          /*Register ref = ()->{
8              System.out.println("Registering the User...");9          };
10         ref.register();*/
11
12         // Static Method Referencing
13         Register ref = Authentication::registerUser;
14         ref.register();
15
16         //Login loginRef = Authentication::loginUser; // error -> We cannot refer by the class name as method is non static
17         // Non Static Method Referencing or Instance Method Referencing
18         Authentication auth = new Authentication();
19         Login loginRef = auth::loginUser;
20         loginRef.login("admin@example.com", "admin123");
21
22         //Taxes tRef = auth::getFinalAmount;
23         Taxes tRef = new Authentication()::getFinalAmount; // Method Referencing on non static-instance method when we create the object
24         System.out.println("Final Amount to Pay for 1200 is: "+tRef.getAmountToPay(1200));
25     }
26 }

```

## 6.6 The **BookMovieTicketTask** includes a static method with the name **bookTicket**. The method prints messages related to movie ticket booking

```

1  package com.simplilearn;
2
3  public class MethodReferences {
4
5      public static void main(String[] args) {
6
7          /*Register ref = ()->{
8              System.out.println("Registering the User...");9          };
10         ref.register();*/
11
12         // Static Method Referencing
13         Register ref = Authentication::registerUser;
14         ref.register();
15
16         //Login loginRef = Authentication::loginUser; // error -> We cannot refer by the class name as method is non static
17         // Non Static Method Referencing or Instance Method Referencing
18         Authentication auth = new Authentication();
19         BookMovieTicketTask bookTicketTask = auth::bookTicket;
20         bookTicketTask.bookTicket();
21
22         //Taxes tRef = auth::getFinalAmount;
23     }
24 }

```

6.7 Create a Runnable object and assign the **bookTicket** method to it. Create a new thread object, pass the Runnable object as a parameter, and start the thread

```

File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferences.java X I
34
35 class BookMovieTicketTask{
36     static void bookTicket() {
37         System.out.println("1. Please Pay 200");
38         System.out.println("2. Ticket for the Movie Avengers Generated with Seat no.1 in row B");
39         System.out.println("3. Email Sent");
40     }
41 }
42
43 public class MethodReferences {
44
45     public static void main(String[] args) {
46
47         /*Register ref = ()->{
48             System.out.println("Registering the User...");
49         };
50
51         ref.register();*/
52
53         // Static Method Referencing
54         Register ref = Authentication::registerUser;
55         ref.register();
56
57         //Login loginRef = Authentication::loginUser; // error -> We cannot refer by the class name as method is non static
58         // Non Static Method Referencing or Instance Method Referencing
59         Authentication auth = new Authentication();
60         Login loginRef = auth::loginUser;
61         loginRef.login("admin@example.com", "admin123");
62
63         //Taxes tRef = auth::getFinalAmount;
64         Taxes tRef = new Authentication()::getFinalAmount; // Method Referencing on non static-instance method when we create the object
65         System.out.println("Final Amount to Pay for 1200 is: "+tRef.getAmountToPay(1200));
66
67         Runnable runnable = BookMovieTicketTask::bookTicket;
68         new Thread(runnable).start();
69     }
70 }
71

```

6.8 When you run the program, it will asynchronously execute the **bookTicket** method and display the corresponding messages

```

File Refactor Navigate Search Project Run Window Help
MethodReferences.java X Problems Javadoc Declaration Console X
bookMovieTicketTask{
    static void bookTicket() {
        System.out.println("1. Please Pay 200");
        System.out.println("2. Ticket for the Movie Avengers Generated with Seat no.1 in row B");
        System.out.println("3. Email Sent");
    }
}

public class MethodReferences {
    public static void main(String[] args) {
        /*Register ref = ()->{
            System.out.println("Registering the User...");
        };
        ref.register();*/
        // Static Method Referencing
        Register ref = Authentication::registerUser;
        ref.register();
        //Login loginRef = Authentication::loginUser; // error -> We cannot refer by the class name as method is non static
        // Non Static Method Referencing or Instance Method Referencing
        Authentication auth = new Authentication();
        Login loginRef = auth::loginUser;
        loginRef.login("admin@example.com", "admin123");
        //Taxes tRef = auth::getFinalAmount;
        Taxes tRef = new Authentication()::getFinalAmount; // Method Referencing on non static-instance method when we create the object
    }
}

```

Output in Console:

```

<terminated> MethodReferences [java Application] /usr/eclipse/plugins/org.eclipse.justj.op
Registering the User...
Log In Successfull.
Final Amount to Pay for 1200 is: 1416.0
1. Please Pay 200
2. Ticket for the Movie Avengers Generated with Seat no.1 in row B
3. Email Sent

```

## 6.9 The **Message** class includes a constructor that takes a string parameter

```

File Edit Source Refactor Navigate Search Project Run Window Help
*MethodReferences.java X
34
35 class BookMovieTicketTask{
36     static void bookTicket() {
37         System.out.println("1. Please Pay 200");
38         System.out.println("2. Ticket for the Movie Avengers Generated with Seat no.1 in row B");
39         System.out.println("3. Email Sent");
40     }
41 }
42
43 class Message{
44     Message(String text, String phone){
45         System.out.println("Thank You");
46         System.out.println("You text has been sent to "+phone);
47     }
48 }
49
50 public class MethodReferences {
51
52     public static void main(String[] args) {
53
54         /*Register ref = ()->{
55             System.out.println("Registering the User...");
56         };
57
58         ref.register();*/
59
60         // Static Method Referencing
61         Register ref = Authentication::registerUser;
62         ref.register();
63
64         //Login loginRef = Authentication::loginUser; // error -> We cannot refer by the class name as method is non static
65
66     }
67 }

```

## Step 7: Write the reference variable notification and execute the code

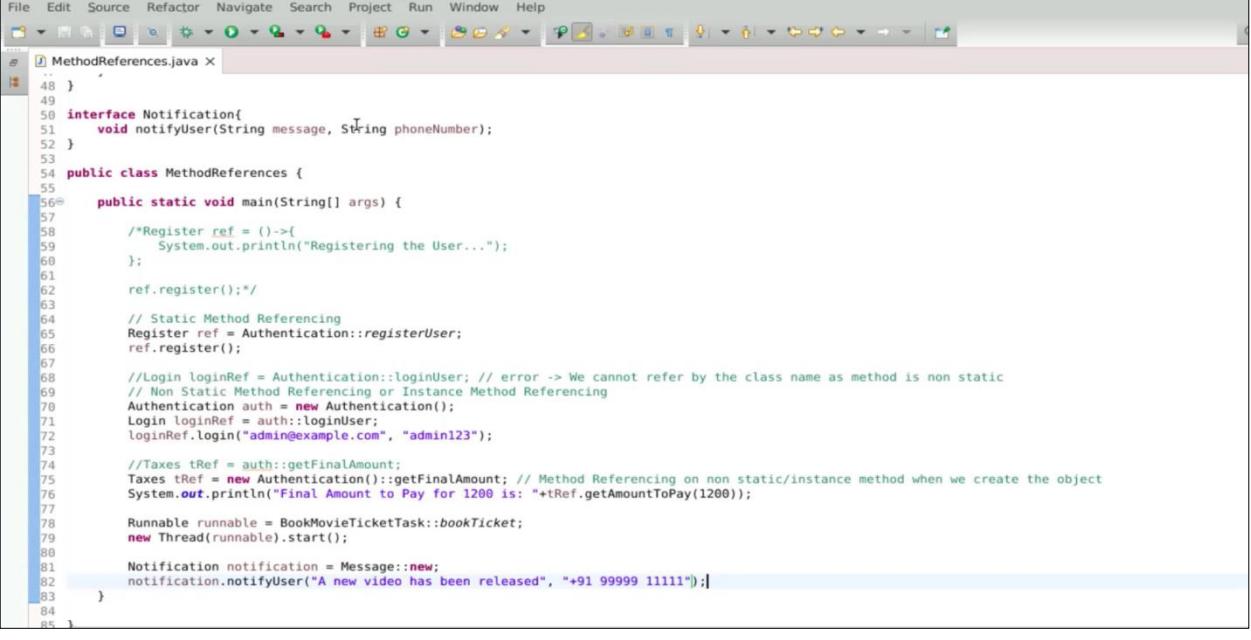
7.1 Create an interface called **Notification** with the **notifyUser** method, which takes a **Message** object and a phone number as input.

```

File Edit Source Refactor Navigate Search Project Run Window Help
*MethodReferences.java X
46         System.out.println("You text has been sent to "+phone);
47     }
48 }
49
50 interface Notification{
51     void notifyUser(String message, String phoneNumber);
52 }
53
54 public class MethodReferences {
55
56     public static void main(String[] args) {
57
58         /*Register ref = ()->{
59             System.out.println("Registering the User...");
60         };
61
62         ref.register();*/
63
64         // Static Method Referencing
65         Register ref = Authentication::registerUser;
66         ref.register();
67
68         //Login loginRef = Authentication::loginUser; // error -> We cannot refer by the class name as method is non static
69         // Non Static Method Referencing or Instance Method Referencing
70         Authentication auth = new Authentication();
71         Login loginRef = auth::loginUser;
72         loginRef.login("admin@example.com", "admin123");
73
74         //Taxes tRef = auth::getFinalAmount;
75         Taxes tRef = new Authentication()::getFinalAmount; // Method Referencing on non static-instance method when we create the object
76         System.out.println("Final Amount to Pay for 1200 is: "+tRef.getAmountToPay(1200));
77
78         Runnable runnable = BookMovieTicketTask::bookTicket;
79         new Thread(runnable).start();
80
81     Notification notification = Message::new;
82 }
83

```

7.2 Create a reference variable notification of type **Notification** and assign it the reference of the **Message** constructor using the **new** keyword. Use the **notifyUser** method to send a notification with the message **A new video has been released** and a phone number



The screenshot shows a Java code editor with the file "MethodReferences.java" open. The code demonstrates various method references and their usage. The code includes imports for java.util.List, java.util.ArrayList, java.util.Arrays, and java.util.stream.Collectors. It defines an interface Notification with a notifyUser method. A class MethodReferences contains a main method. Inside the main method, it registers a ref using a lambda expression, prints a registration message, and then calls its register method. It also shows static method referencing where a Register ref is created from Authentication::registerUser and then called. It attempts to use a non-static method loginUser directly from the class Login, which fails. It then creates a Taxes tRef from the static method getFinalAmount in the Authentication class and uses it to print a final amount. Finally, it creates a notification using the Message constructor with a new keyword and sends a notification message.

```
File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferences.java X
48 }
49
50 interface Notification{
51     void notifyUser(String message, String phoneNumber);
52 }
53
54 public class MethodReferences {
55
56     public static void main(String[] args) {
57
58         /*Register ref = ()->{
59             System.out.println("Registering the User...");
60         };
61
62         ref.register();*/
63
64         // Static Method Referencing
65         Register ref = Authentication::registerUser;
66         ref.register();
67
68         //Login loginRef = Authentication::loginUser; // error -> We cannot refer by the class name as method is non static
69         // Non Static Method Referencing or Instance Method Referencing
70         Authentication auth = new Authentication();
71         Login loginRef = auth::loginUser;
72         loginRef.login("admin@example.com", "admin123");
73
74         //Taxes tRef = auth::getFinalAmount;
75         Taxes tRef = new Authentication():getFinalAmount; // Method Referencing on non static-instance method when we create the object
76         System.out.println("Final Amount to Pay for 1200 is: "+tRef.getAmountToPay(1200));
77
78         Runnable runnable = BookMovieTicketTask::bookTicket;
79         new Thread(runnable).start();
80
81         Notification notification = Message::new;
82         notification.notifyUser("A new video has been released", "+91 99999 11111");
83     }
84
85 }
```

7.3 When you run the code, it will display the message as shown:

The screenshot shows the Eclipse IDE interface. On the left, the code editor displays `MethodReferences.java` with Java code demonstrating method references. On the right, the `Console` tab shows the application's output. The output includes a registration message, a login confirmation, a payment instruction, a movie ticket booking, an email sent notification, and a final message indicating the text has been sent.

```
File Edit Source Refactor Navigate Search Project Run Window Help
MethodReferences.java X
48 }
49
50 interface Notification{
51     void notifyUser(String message, String phoneNumber);
52 }
53
54 public class MethodReferences {
55
56     public static void main(String[] args) {
57
58         /*Register ref = ()->{
59             System.out.println("Registering the User...");
60         };
61
62         ref.register();*/
63
64         // Static Method Referencing
65         Register ref = Authentication::registerUser;
66         ref.register();
67
68         //Login loginRef = Authentication::loginUser; // error -> We cannot refer by the class name
69         // Non Static Method Referencing or Instance Method Referencing
70         Authentication auth = new Authentication();
71         Login loginRef = auth::loginUser;
72         loginRef.login("admin@example.com", "admin123");
73
74         //Taxes tRef = auth::getFinalAmount;
75         Taxes tRef = new Authentication()::getFinalAmount; // Method Referencing on non static(instance) method when we create the object
76         System.out.println("Final Amount to Pay for 1200 is: "+tRef.getAmountToPay(1200));
77
78         Runnable runnable = BookMovieTicketTask::bookTicket;
```

<terminated> MethodReferences [Java Application] /usr/eclipse/plugins/org  
Registering the User...  
Log In Successfull..  
Final Amount to Pay for 1200 is: 1416.0  
1. Please Pay 200  
2. Ticket for the Movie Avengers Generated with Seat no.1 in row 1  
3. Email Sent  
Thank You  
You text has been sent to +91 99999 1111.

By following these steps, you have successfully implemented method references in Java, including creating functional interfaces and executing code with example data.