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

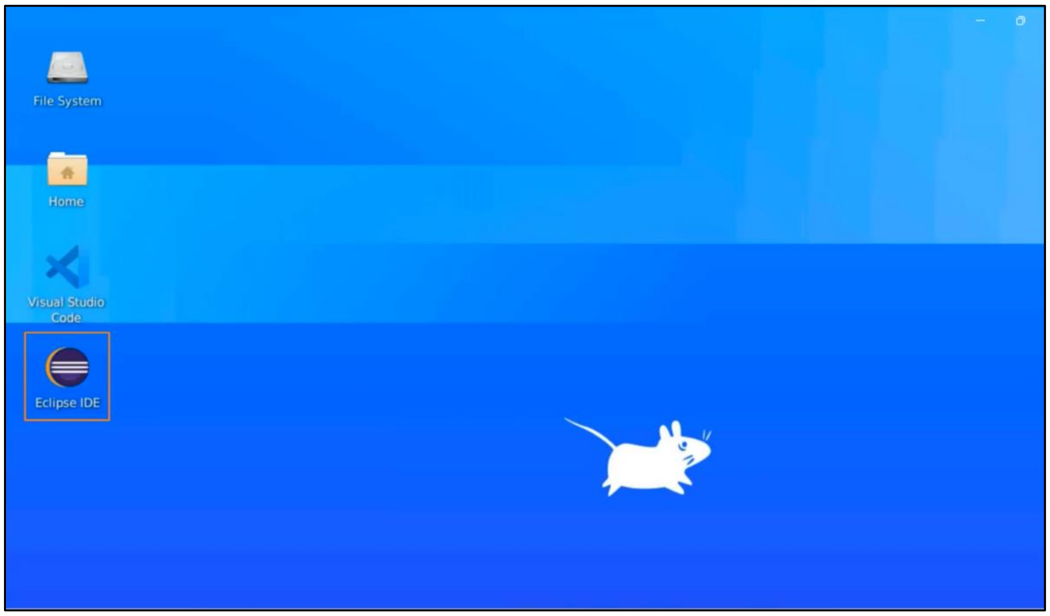
Prerequisites: None

Steps to be followed:

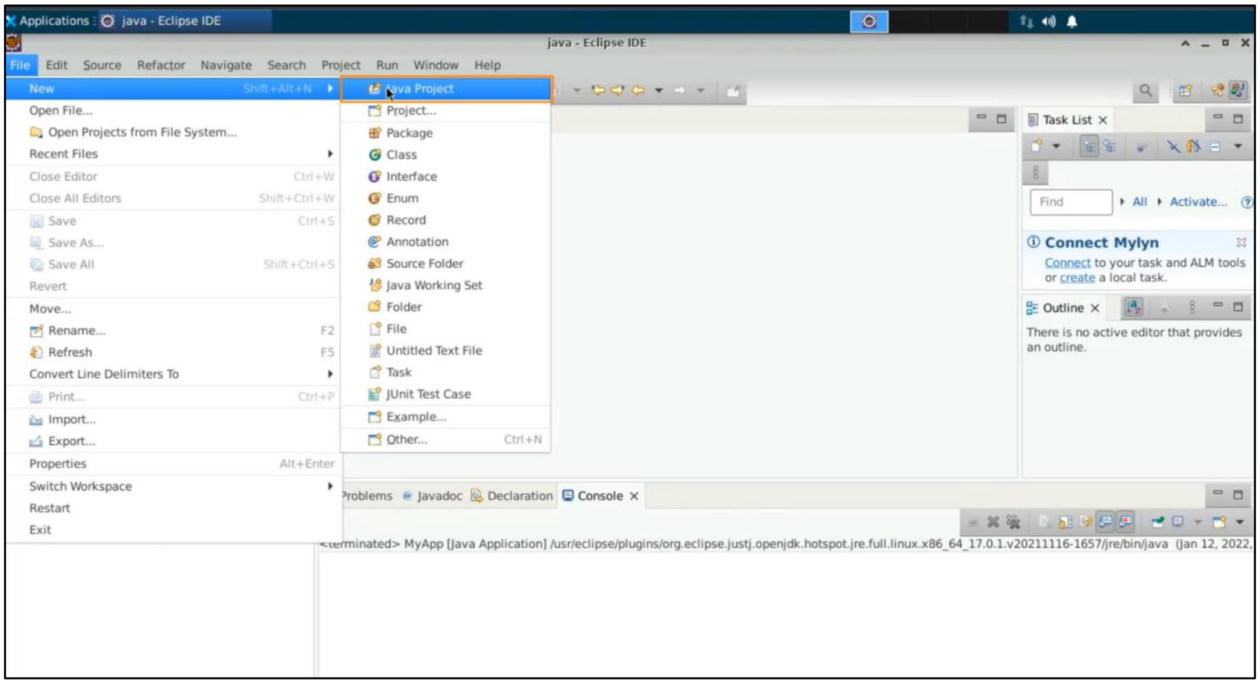
1. Open the Eclipse IDE and create a new Java project
2. Create a functional interface
3. Implement the interface
4. Use anonymous class for running an interface
5. Create a function, which is the implementation for the Lambda expression
6. Write another interface as login and execute the code with sample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

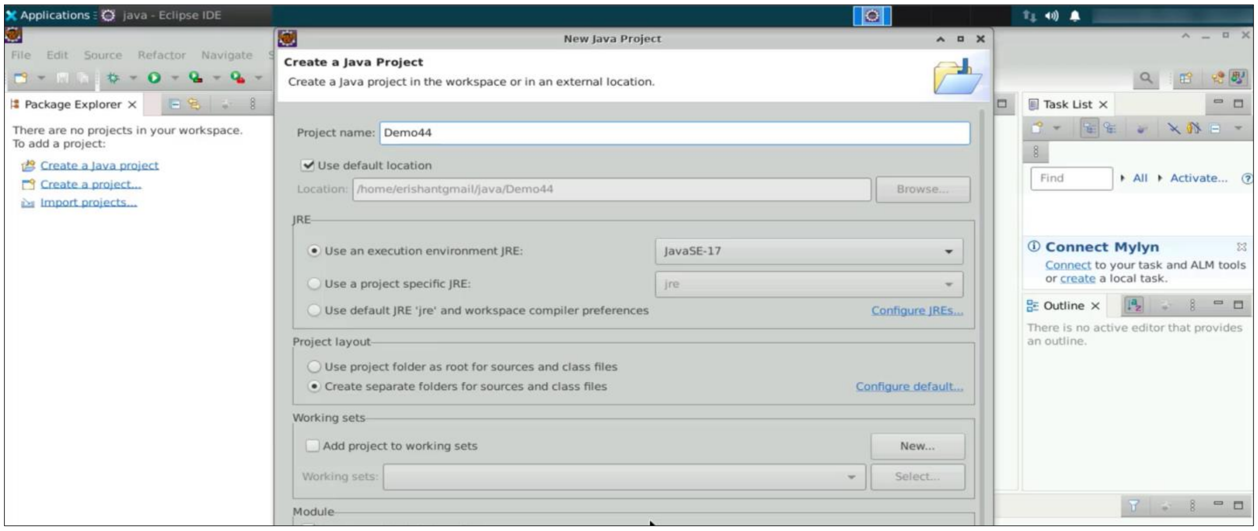


1.2 Select **File**, then **New,** and then **Java project**



```
<terminated> MyApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java (Jan 12, 2022,
```

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


Applications : java - Eclipse IDE
File Edit Source Refactor Navigate
Package Explorer
Create a Java project
Create a project...
Import projects...
Task List
Find
All
Activate...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
There is no active editor that provides an outline.

There are no projects in your workspace.
To add a project:

New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: Demo44

[✓] Use default location

Location: /home/erishantgmail/java/Demo44 Browse...

JRE

(●) Use an execution environment JRE: JavaSE-17

( ) Use a project specific JRE: jre

( ) Use default JRE 'jre' and workspace compiler preferences Configure JREs...

Project layout

( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files Configure default...

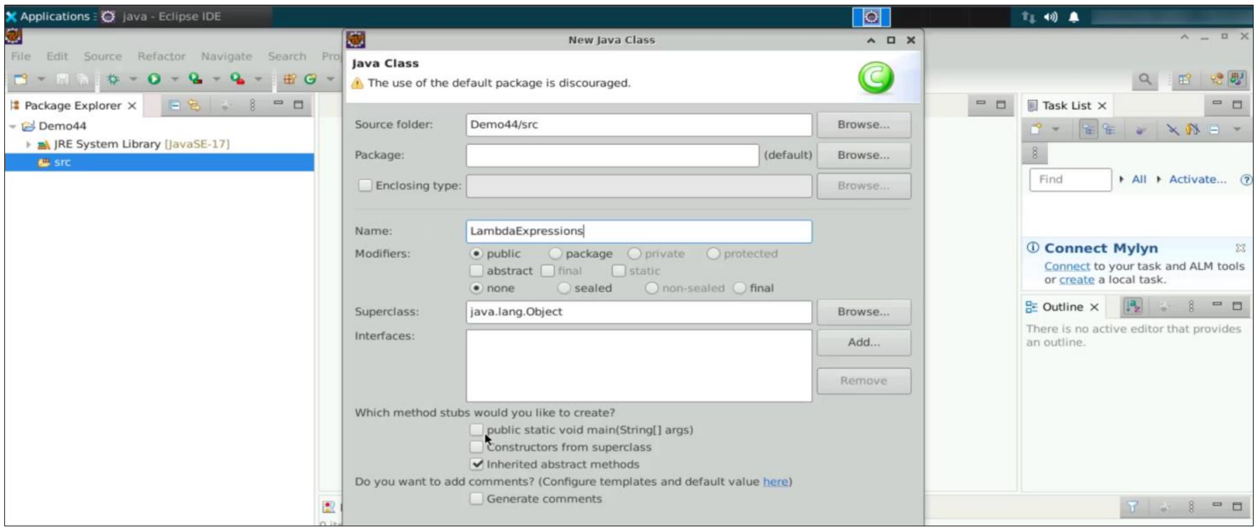
Working sets

[ ] Add project to working sets New...

Working sets: Select...

Module

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**

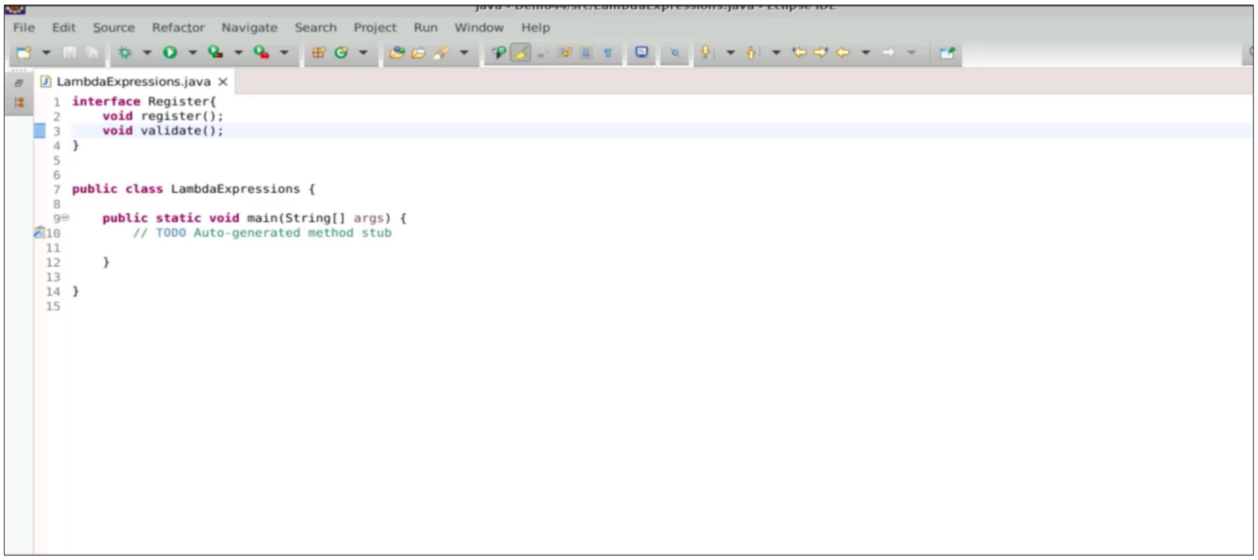

Applications : java - Eclipse IDE
File Edit Source Refactor Navigate Search Pro
Package Explorer
Demo44
JRE System Library [JavaSE-17]
src
New Java Class
Task List
Find
All
Activate...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
There is no active editor that provides an outline.

Java Class
The use of the default package is discouraged.

Source folder: Demo44/src
Package: (default)
Enclosing type:
Name: LambdaExpressions
Modifiers: [✓ public] package private protected
abstract final static
[✓ none] sealed non-sealed final
Superclass: java.lang.Object
Interfaces:
Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
☐ Generate comments

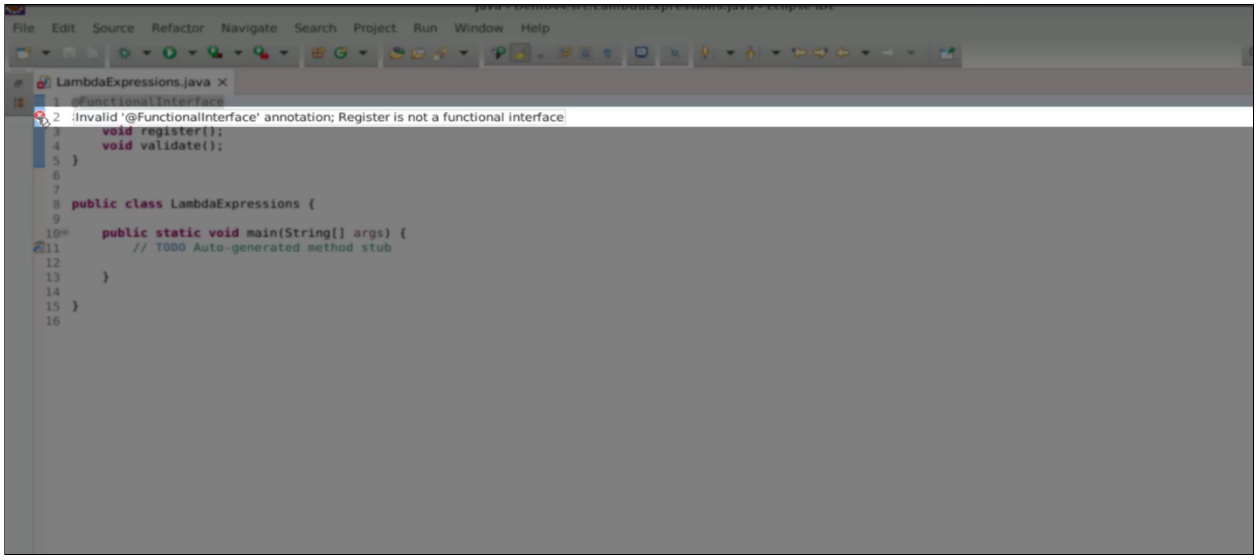
Step 2: Create a functional interface

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


File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java

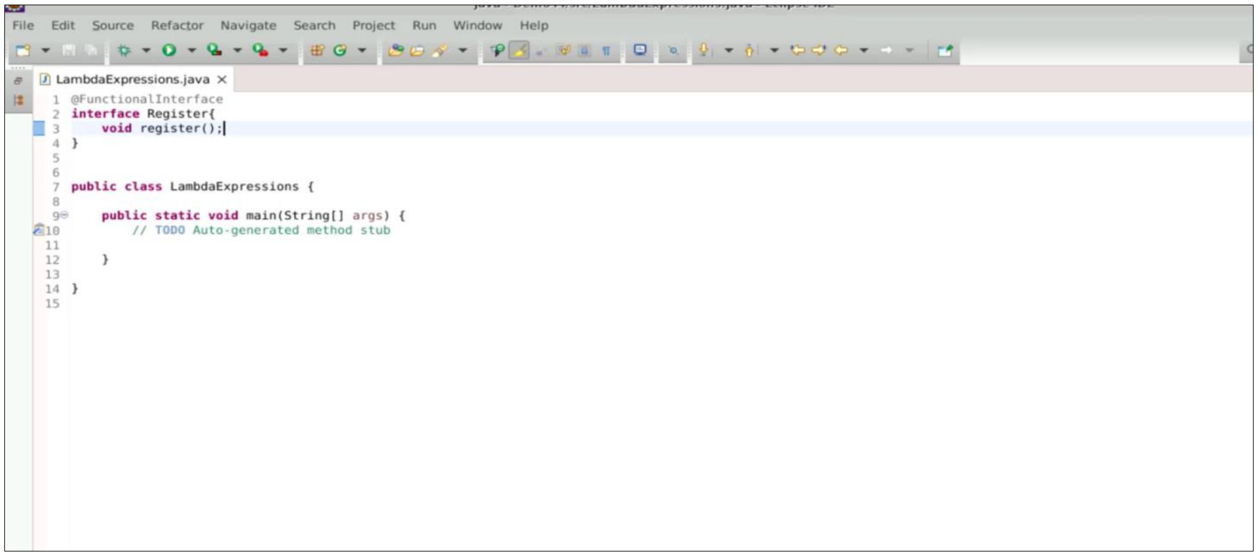
```java
interface Register{
    void register();
    void validate();
}


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


File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java

```
@FunctionalInterface
Invalid '@FunctionalInterface' annotation; Register is not a functional interface
    void register();
    void validate();
}


public class LambdaExpressions {

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

    }

}
```

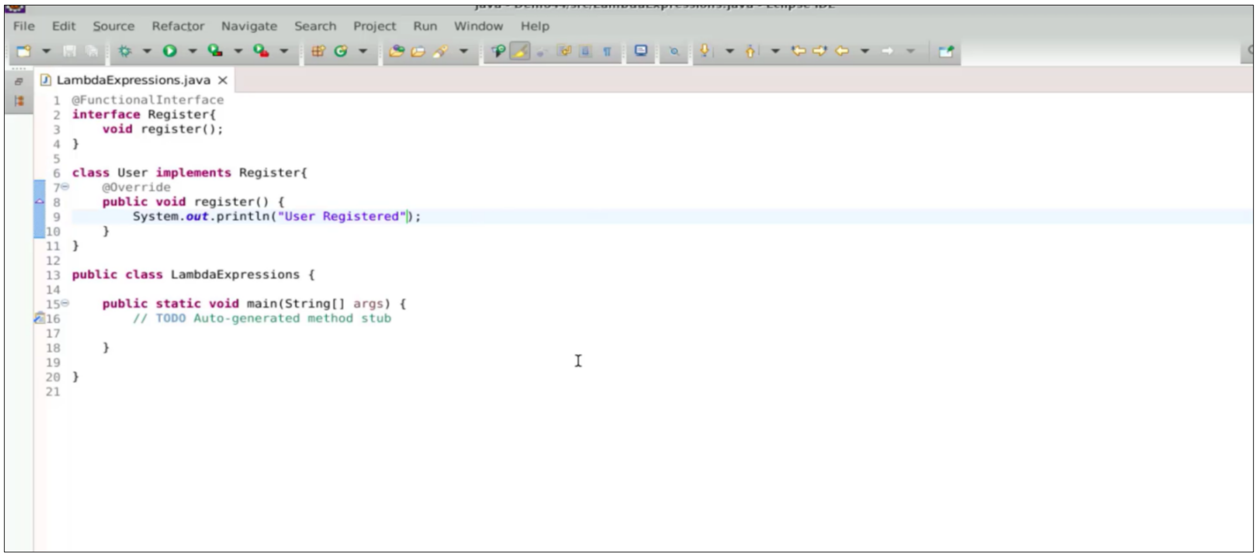
2.3 The moment you remove this **validate** method, you can notice that the error is gone. Functional interfaces are meant to hold only one single abstract method


File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java

```
@FunctionalInterface
interface Register{
    void register();
}


public class LambdaExpressions {

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

    }

}
```

Step 3: Implement the interface

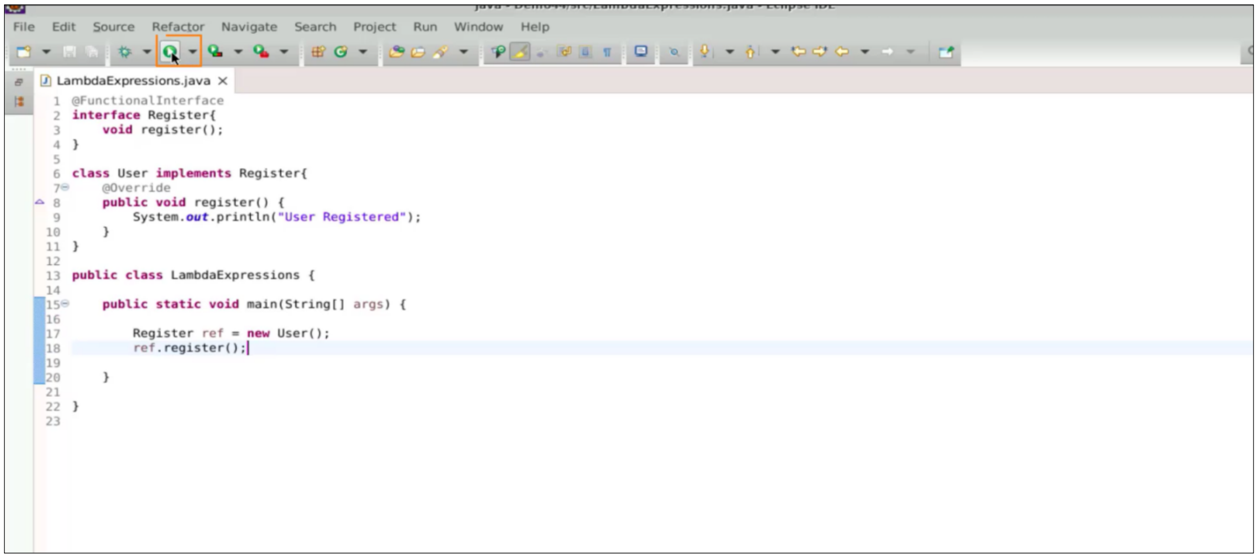
3.1 To work with interface implementation, create a class that implements it. For example, the **User** class can implement the **register** interface. Then, print **User registered** using **System.out.println**



File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java

```java
@FunctionalInterface
interface Register{
    void register();
}

class User implements Register{
    @Override
    public void register() {
        System.out.println("User Registered");
    }
}

public class LambdaExpressions {

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

    }

}
```

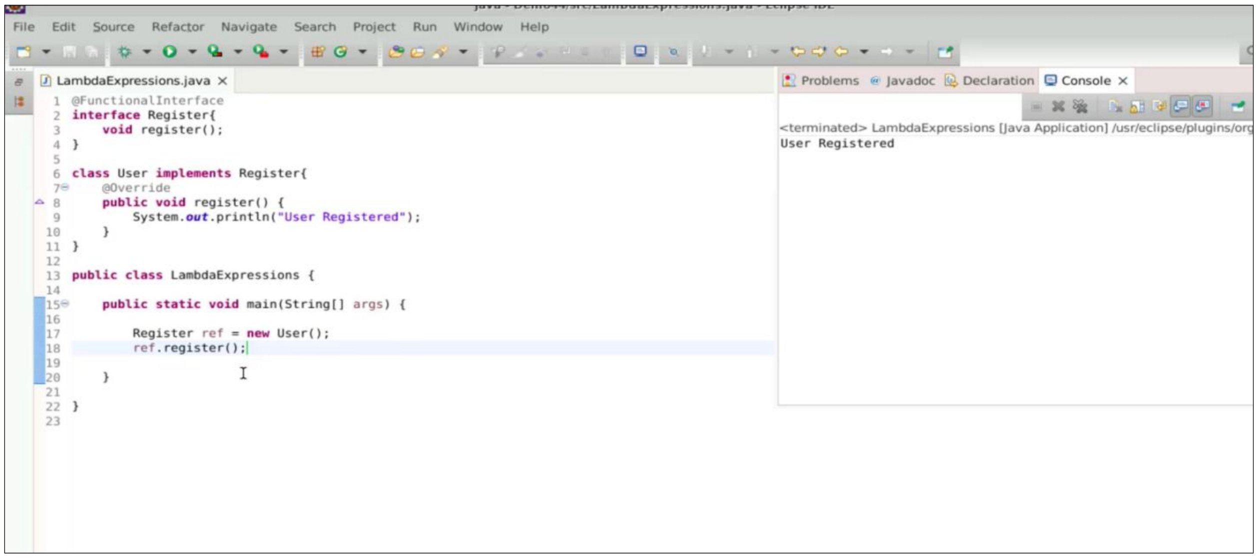
3.2 To achieve polymorphism, create a reference variable for the **register** method and execute it using the **user** object



File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java

```java
@FunctionalInterface
interface Register{
    void register();
}

class User implements Register{
    @Override
    public void register() {
        System.out.println("User Registered");
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        Register ref = new User();
        ref.register();

    }

}
```

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

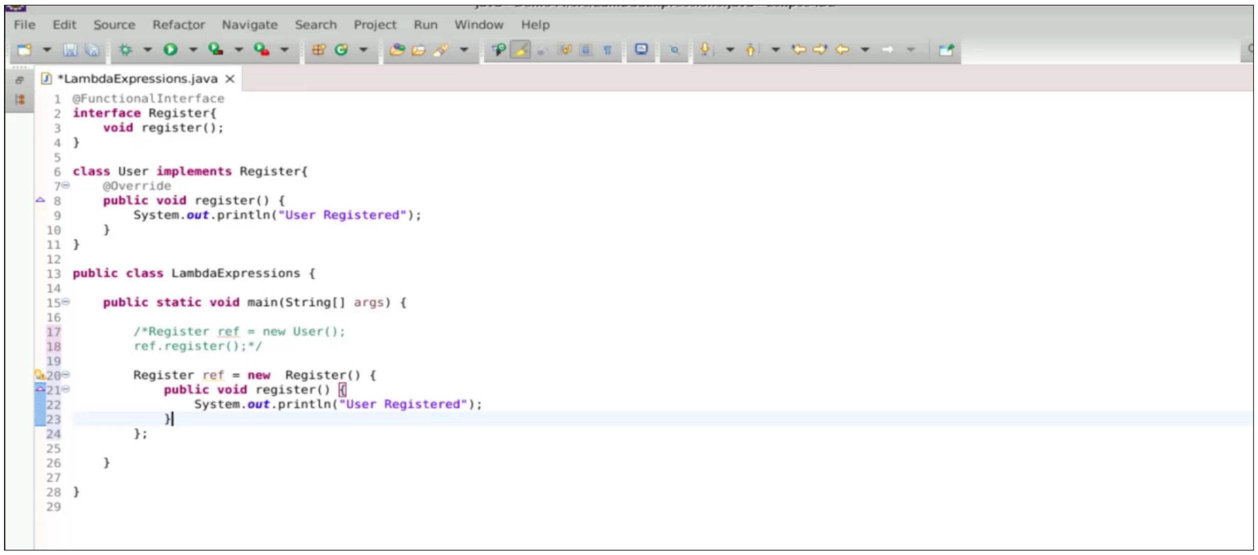


```
@FunctionalInterface
interface Register{
    void register();
}

class User implements Register{
    @Override
    public void register() {
        System.out.println("User Registered");
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        Register ref = new User();
        ref.register();

    }

}
```

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

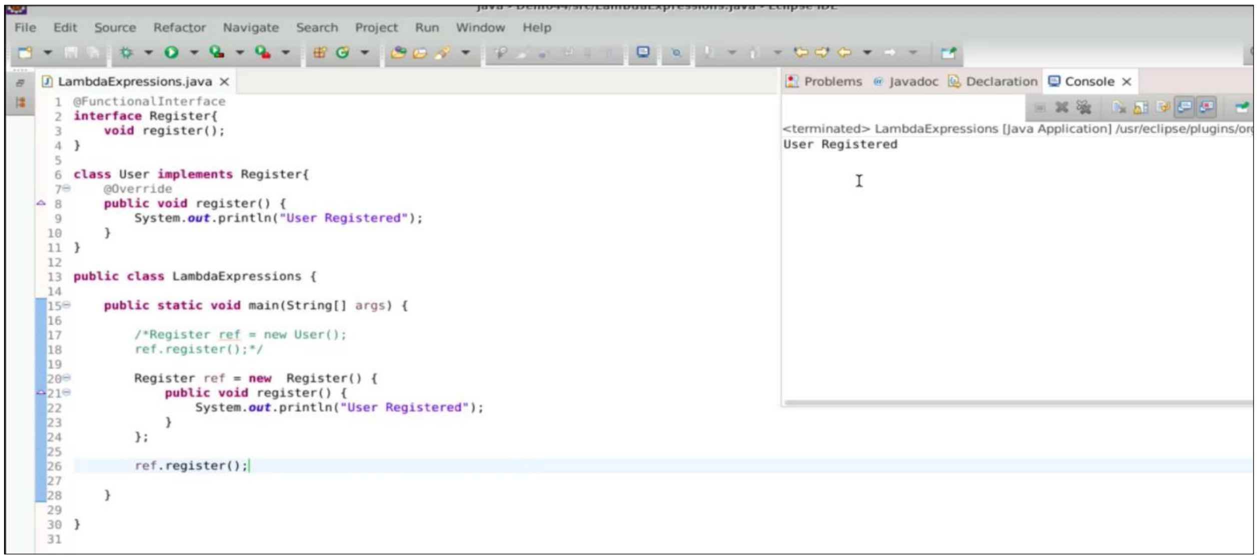
Step 4: Use anonymous class for running an interface

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



```
@FunctionalInterface
interface Register{
    void register();
}

class User implements Register{
    @Override
    public void register() {
        System.out.println("User Registered");
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

    }

}
```

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

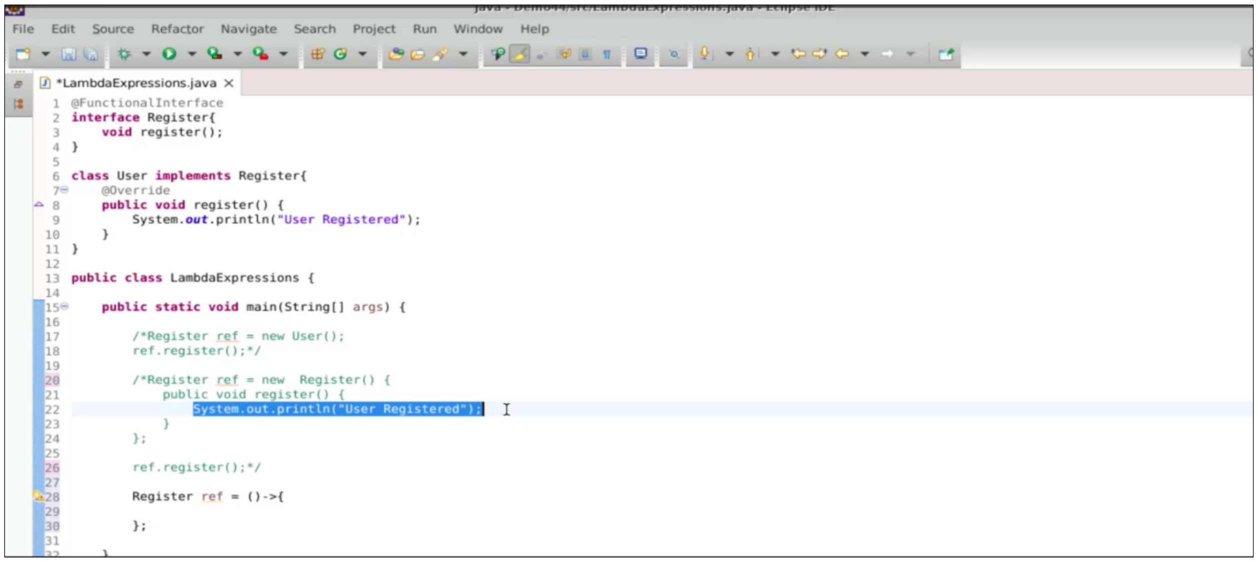


```
@FunctionalInterface
interface Register{
    void register();
}

class User implements Register{
    @Override
    public void register() {
        System.out.println("User Registered");
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();

    }

}
```

```
<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/or
User Registered
```

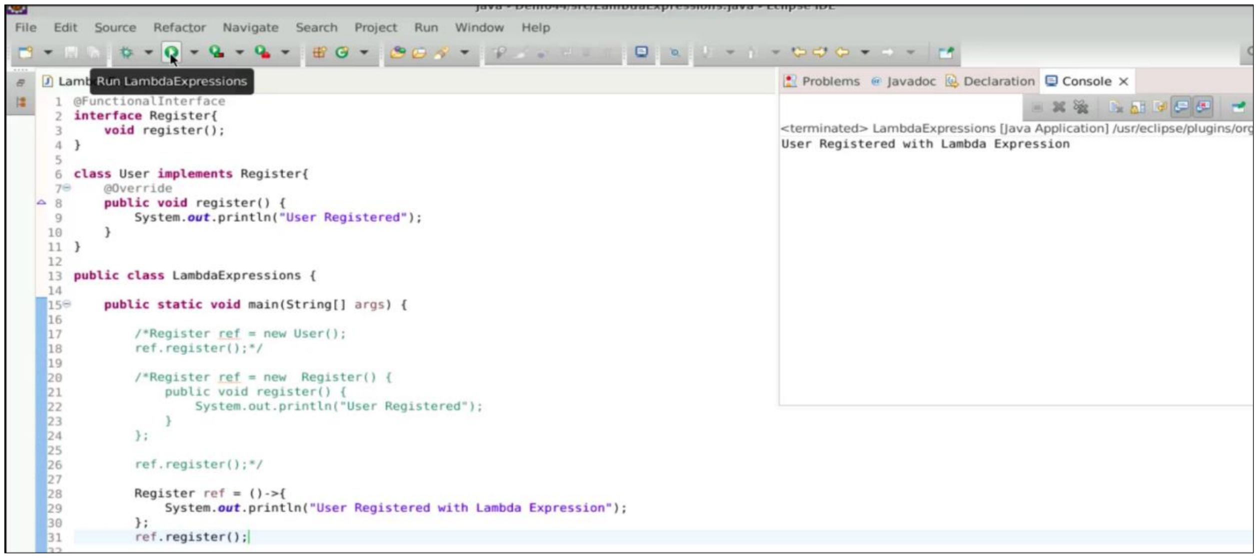
Step 5: Create a function, which is the implementation for the Lambda expression

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.



```
@FunctionalInterface
interface Register{
    void register();
}

class User implements Register{
    @Override
    public void register() {
        System.out.println("User Registered");
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/

        Register ref = ()->{

        };
```

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

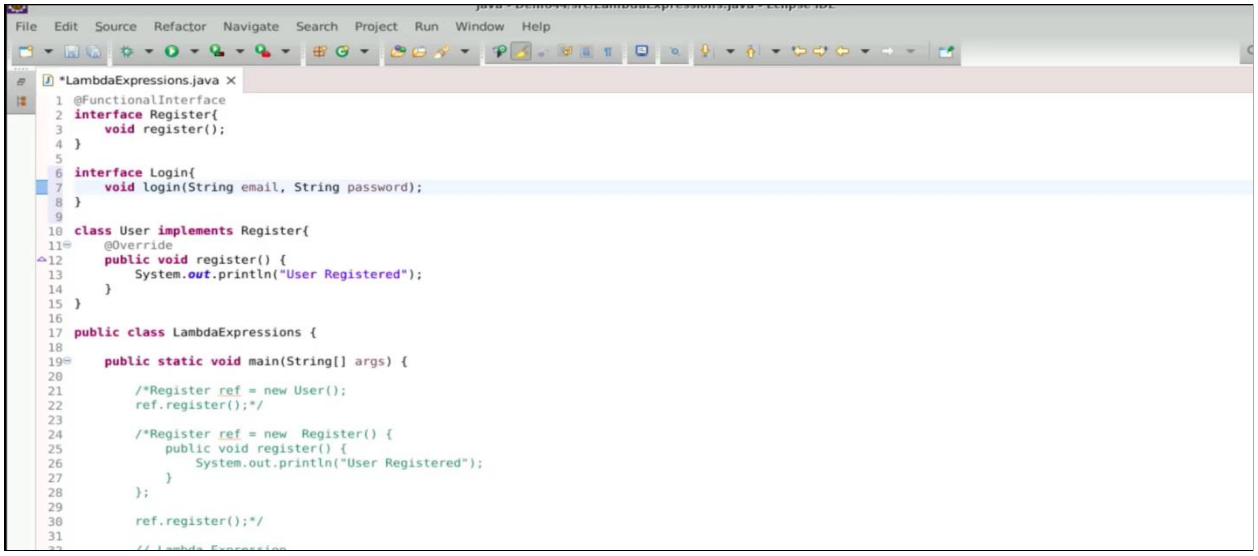


```java
@FunctionalInterface
interface Register{
    void register();
}

class User implements Register{
    @Override
    public void register() {
        System.out.println("User Registered");
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/

        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();
```

```
<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/org
User Registered with Lambda Expression
```

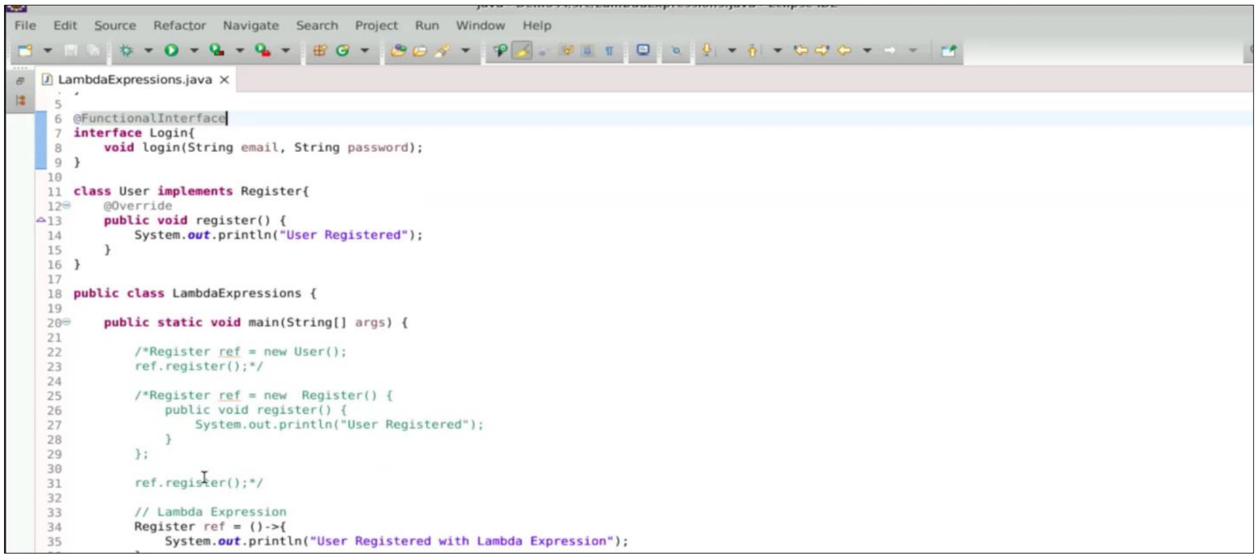
Step 6: Write another interface as login and execute the code with sample data

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



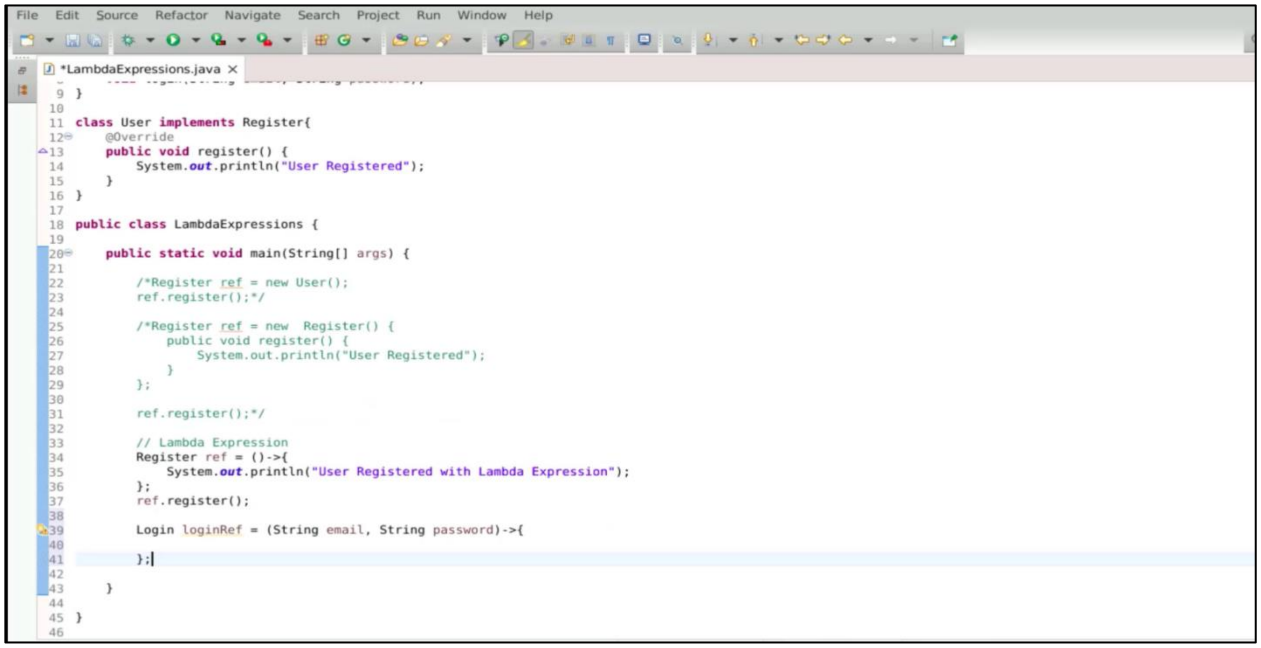
```java
@FunctionalInterface
interface Register{
    void register();
}

interface Login{
    void login(String email, String password);
}

class User implements Register{
    @Override
    public void register() {
        System.out.println("User Registered");
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/
```

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



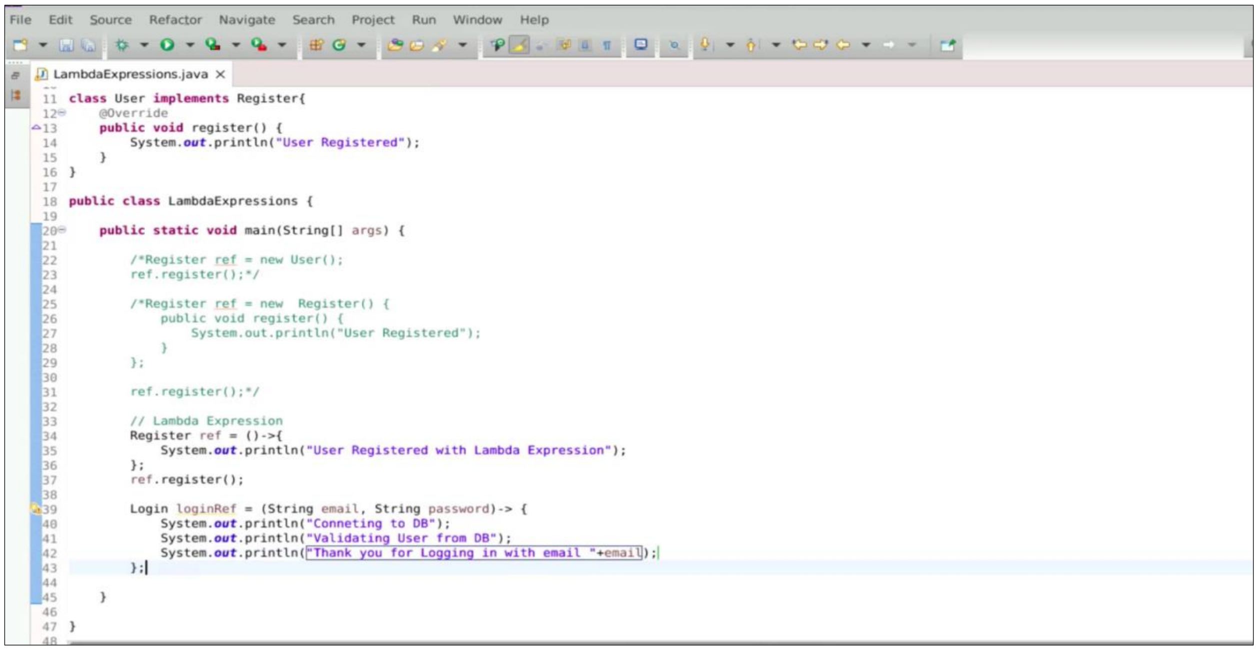
```
@FunctionalInterface
interface Login{
    void login(String email, String password);
}

class User implements Register{
    @Override
    public void register() {
        System.out.println("User Registered");
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/

        // Lambda Expression
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
```

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



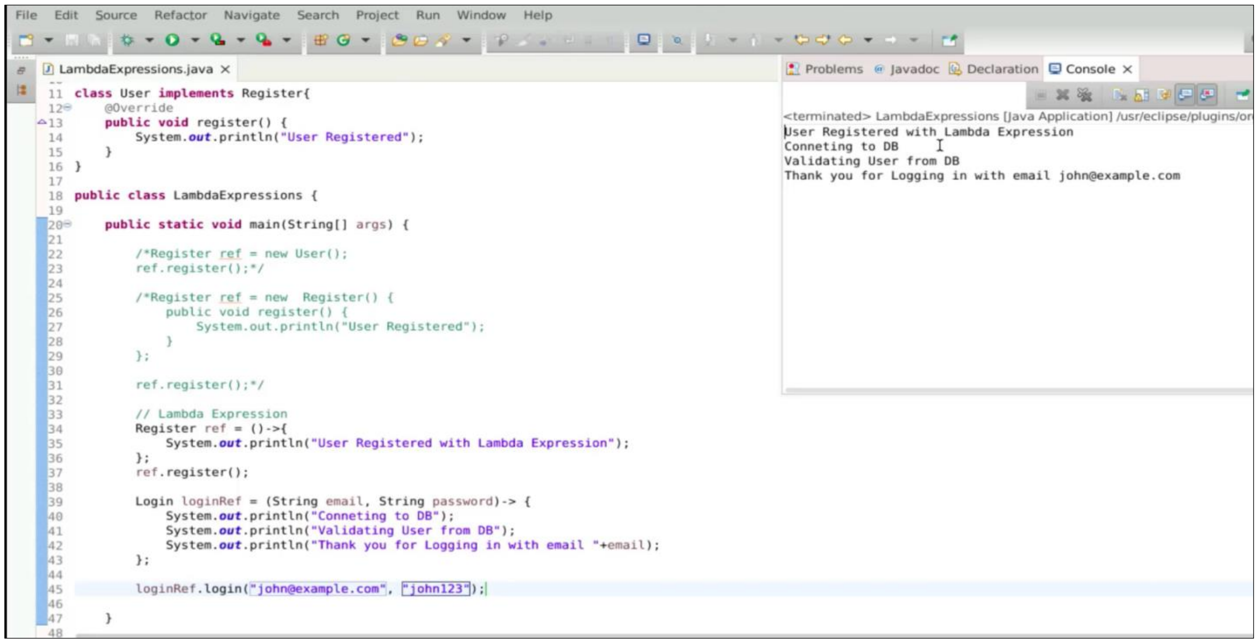
```
}

class User implements Register{
    @Override
    public void register() {
        System.out.println("User Registered");
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/

        // Lambda Expression
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        Login loginRef = (String email, String password)->{

        };

    }
}
```

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

```java
class User implements Register{
    @Override
    public void register() {
        System.out.println("User Registered");
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/

        // Lambda Expression
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        Login loginRef = (String email, String password)-> {
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            System.out.println("Thank you for Logging in with email "+email);
        };

    }
}
```

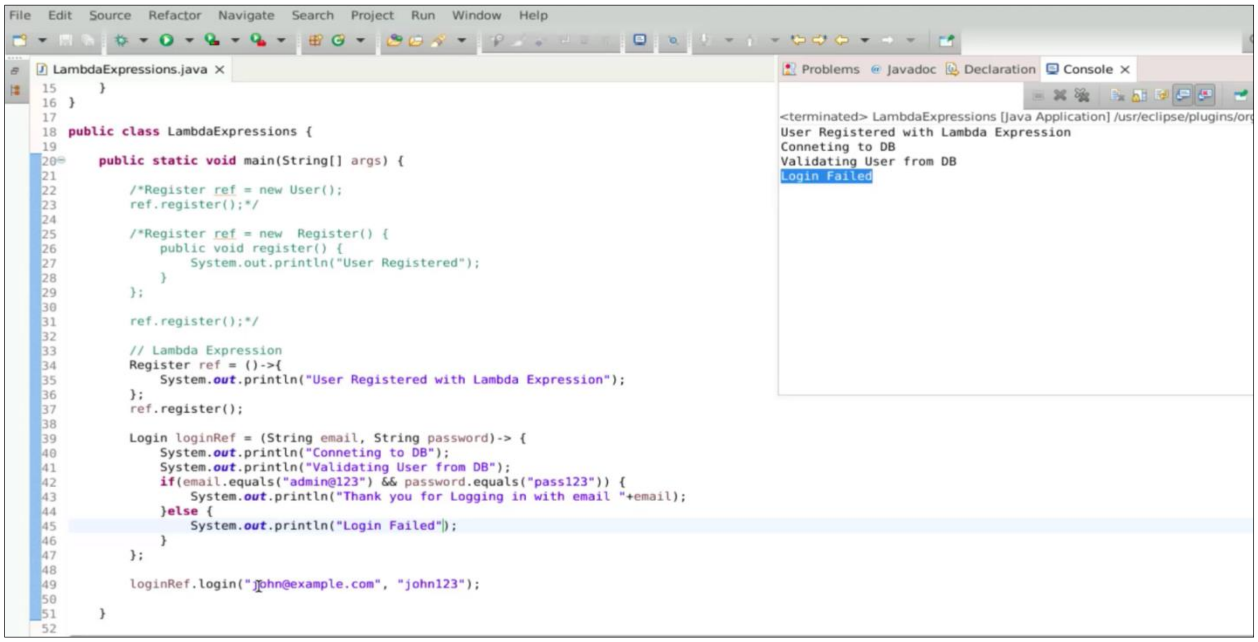
6.5 Execute the **login** method using the **Login** reference, providing the email **John@example.com** and password **john123** as parameters, allowing for the implementation of the entire logic within the method


File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java
Problems Javadoc Declaration Console

```java
class User implements Register{
    @Override
    public void register() {
        System.out.println("User Registered");
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/

        // Lambda Expression
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        Login loginRef = (String email, String password)-> {
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            System.out.println("Thank you for Logging in with email "+email);
        };

        loginRef.login("john@example.com", "john123");

    }
```

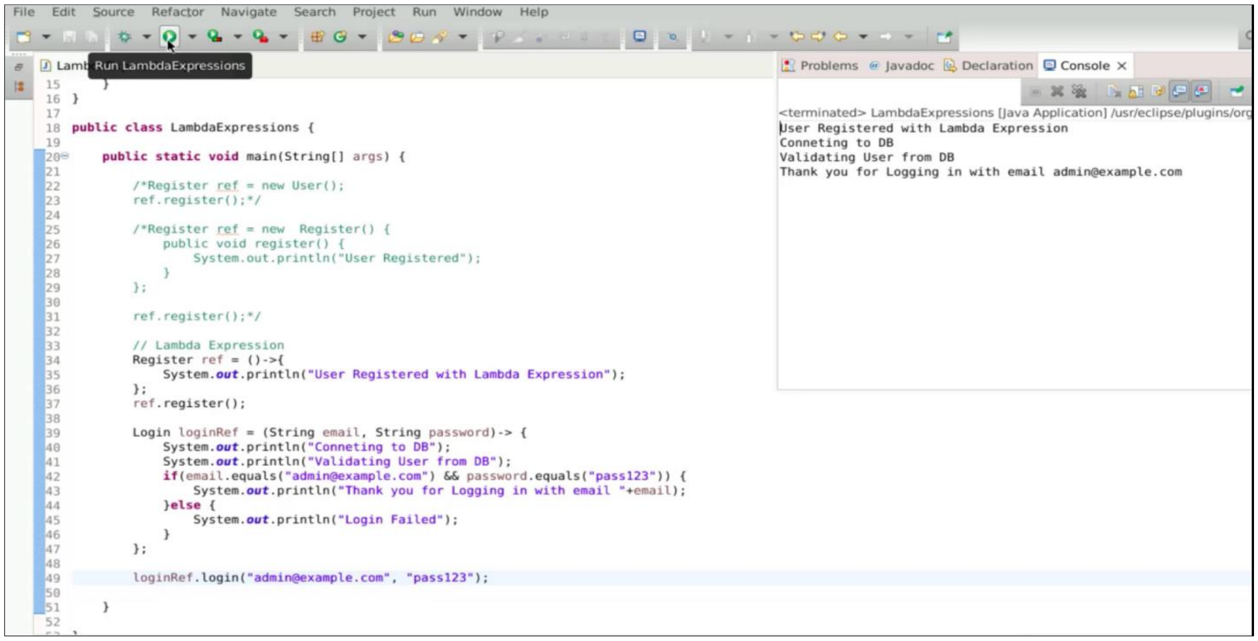
```
<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/or
User Registered with Lambda Expression
Conneting to DB
Validating User from DB
Thank you for Logging in with email john@example.com
```

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**


```
}
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/

        // Lambda Expression
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        Login loginRef = (String email, String password)-> {
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            if(email.equals("admin@123") && password.equals("pass123")) {
                System.out.println("Thank you for Logging in with email "+email);
            }else {
                System.out.println("Login Failed");
            }
        };

        loginRef.login("john@example.com", "john123");

    }
```

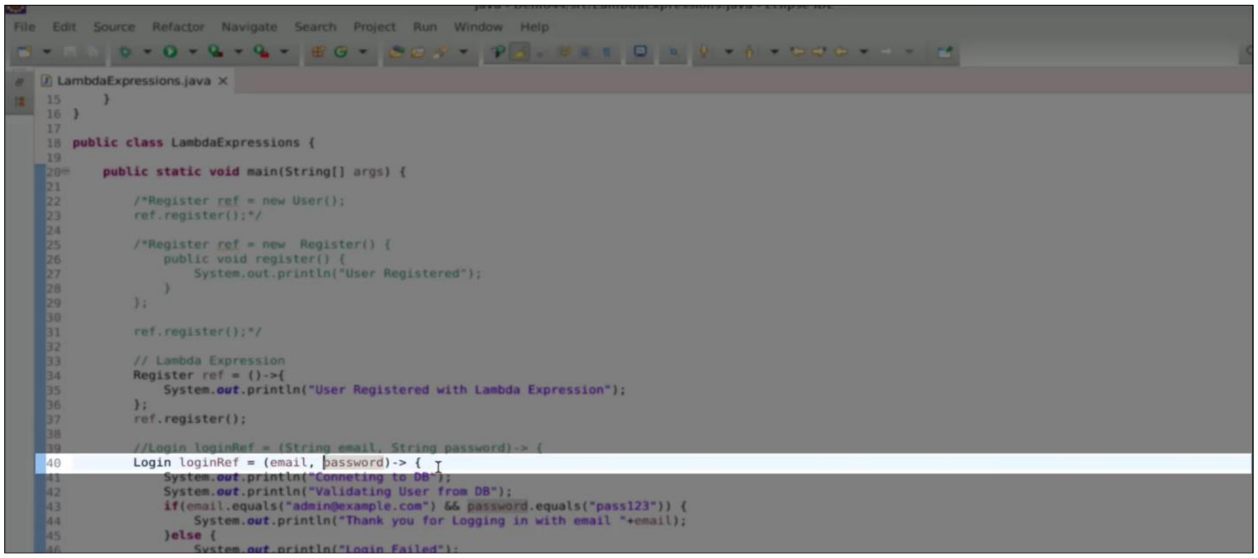
```
<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/or
User Registered with Lambda Expression
Conneting to DB
Validating User from DB
Login Failed
```

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


```
}
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/

        // Lambda Expression
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        Login loginRef = (String email, String password)-> {
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            if(email.equals("admin@example.com") && password.equals("pass123")) {
                System.out.println("Thank you for Logging in with email "+email);
            }else {
                System.out.println("Login Failed");
            }
        };

        loginRef.login("admin@example.com", "pass123");

    }
```

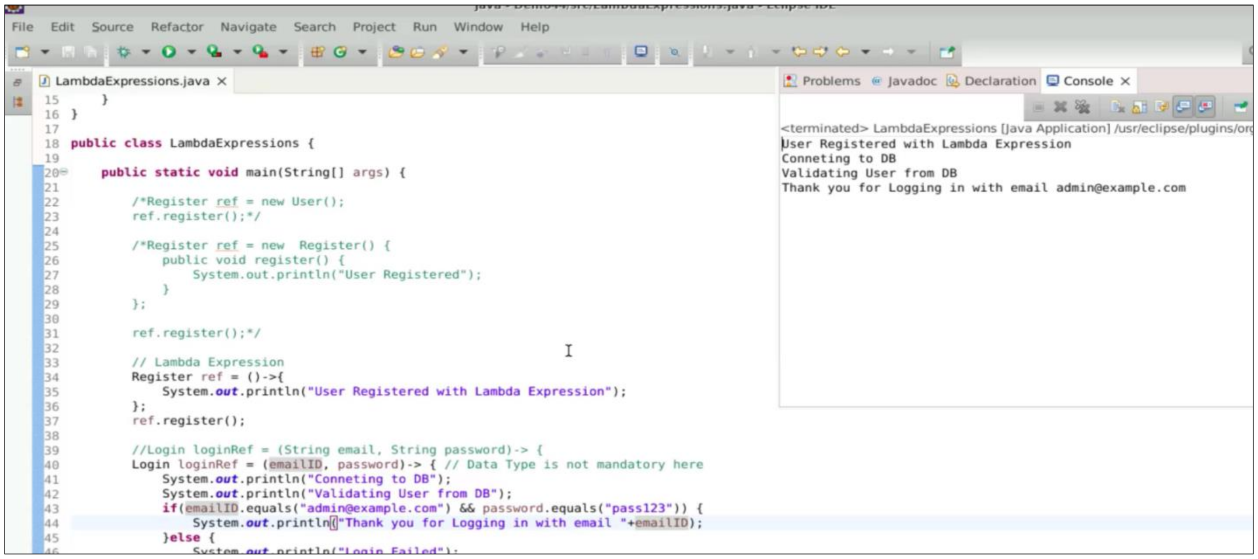
```
<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/or
User Registered with Lambda Expression
Conneting to DB
Validating User from DB
Thank you for Logging in with email admin@example.com
```

6.8 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



```java
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/

        // Lambda Expression
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        //Login loginRef = (String email, String password)-> {
        Login loginRef = (email, password)-> {
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            if(email.equals("admin@example.com") && password.equals("pass123")) {
                System.out.println("Thank you for Logging in with email "+email);
            }else {
                System.out.println("Login Failed");
```

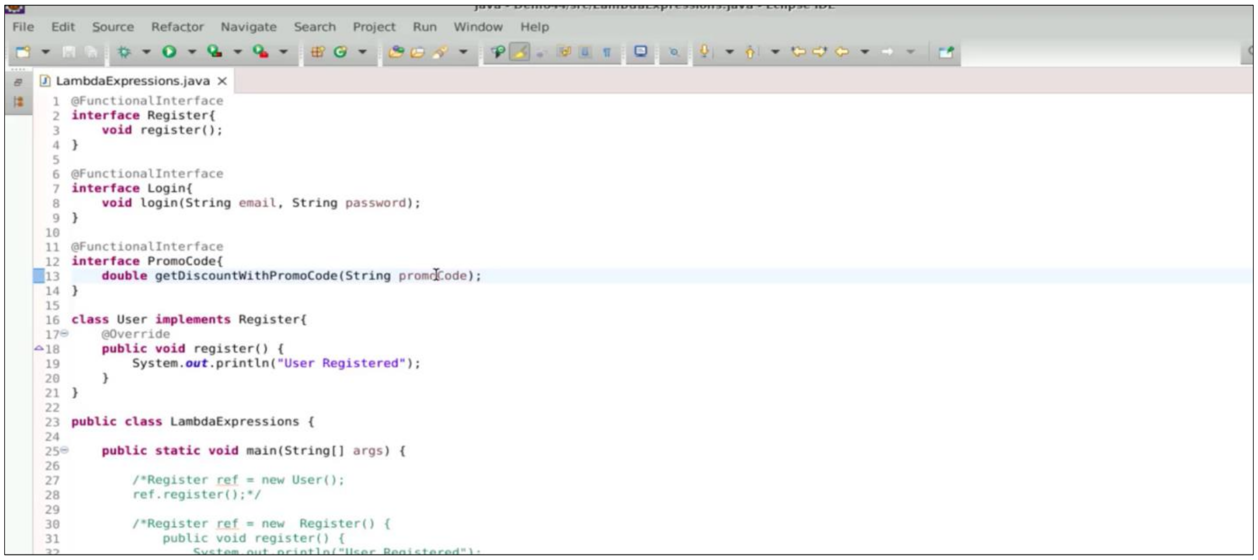
6.9 You can even change the parameter name, such as **emailID**. It is not mandatory to use the same parameter names



```java
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/

        // Lambda Expression
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        //Login loginRef = (String email, String password)-> {
        Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            if(emailID.equals("admin@example.com") && password.equals("pass123")) {
                System.out.println("Thank you for Logging in with email "+emailID);
            }else {
                System.out.println("Login Failed");
```

```
<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
```

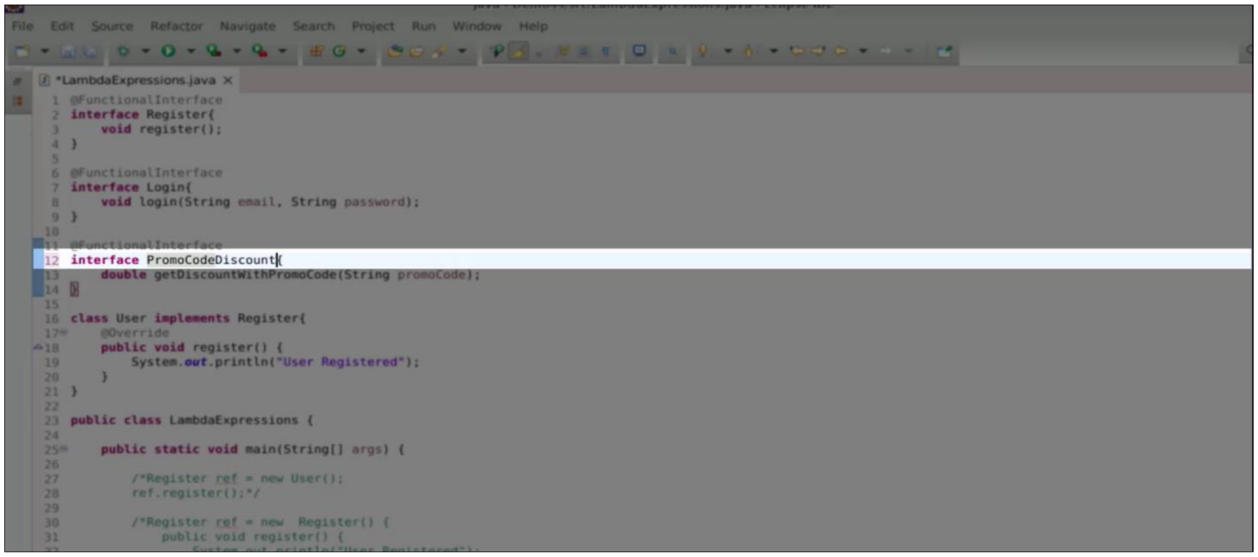
6.10 To create a lambda expression that returns data, define another functional interface called **PromoCode** with the annotation **@FunctionalInterface**. This interface will have a method named **getDiscount** that takes a promo code as input and returns a double value



File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java

```java
@FunctionalInterface
interface Register{
    void register();
}

@FunctionalInterface
interface Login{
    void login(String email, String password);
}

@FunctionalInterface
interface PromoCode{
    double getDiscountWithPromoCode(String promoCode);
}

class User implements Register{
    @Override
    public void register() {
        System.out.println("User Registered");
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
```

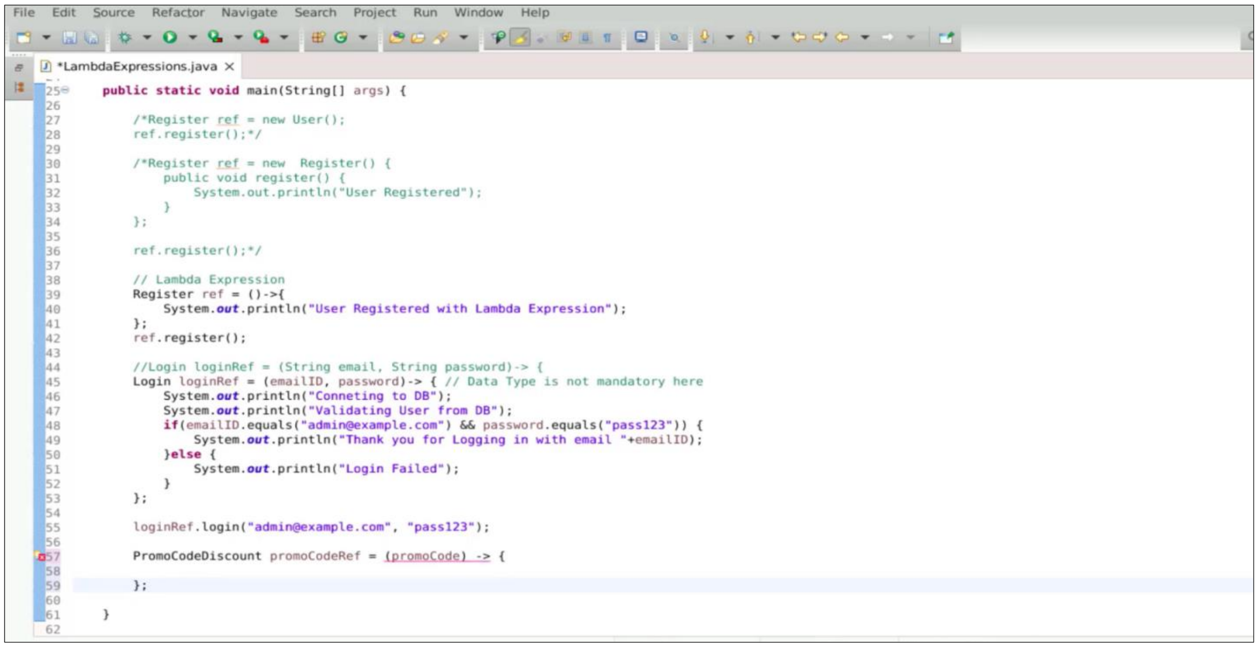
6.11 Based on the promo code, which you are going to pass as input, you will be giving one discount. This is one of the methods that has a return type. Let’s understand how to write a lambda expression for this functional interface. You have the interface known as PromoCode, hence write discounts



File Edit Source Refactor Navigate Search Project Run Window Help
*LambdaExpressions.java

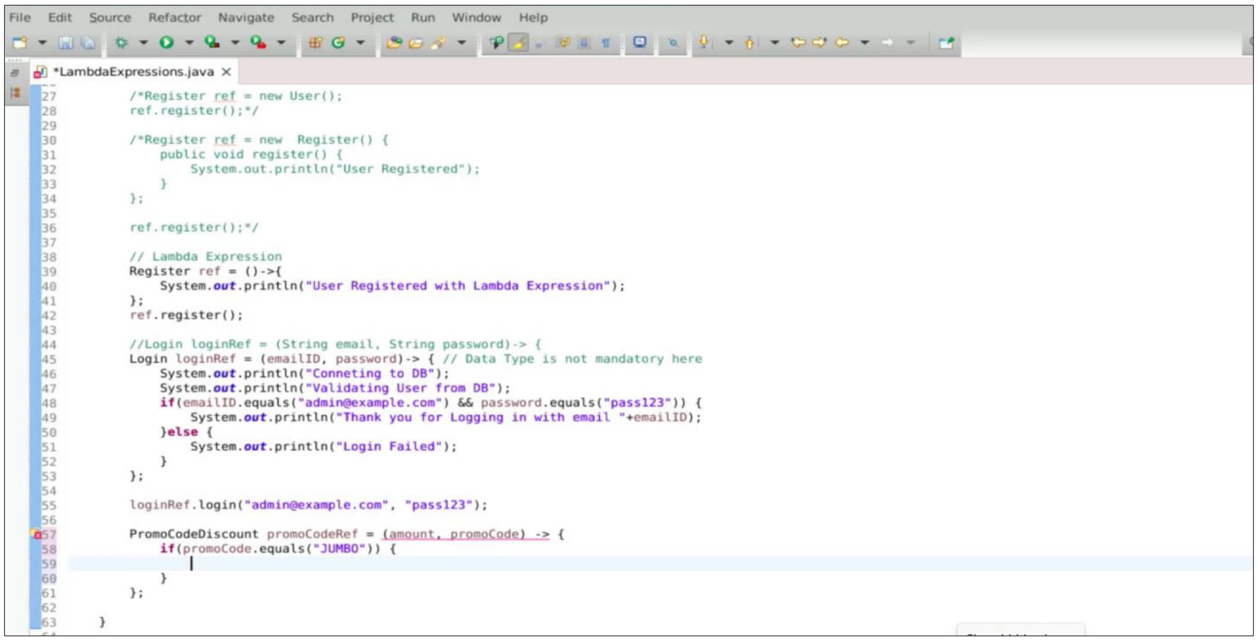
```java
@FunctionalInterface
interface Register{
    void register();
}

@FunctionalInterface
interface Login{
    void login(String email, String password);
}

@FunctionalInterface
interface PromoCodeDiscount{
    double getDiscountWithPromoCode(String promoCode);
}

class User implements Register{
    @Override
    public void register() {
        System.out.println("User Registered");
    }
}

public class LambdaExpressions {

    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
```

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

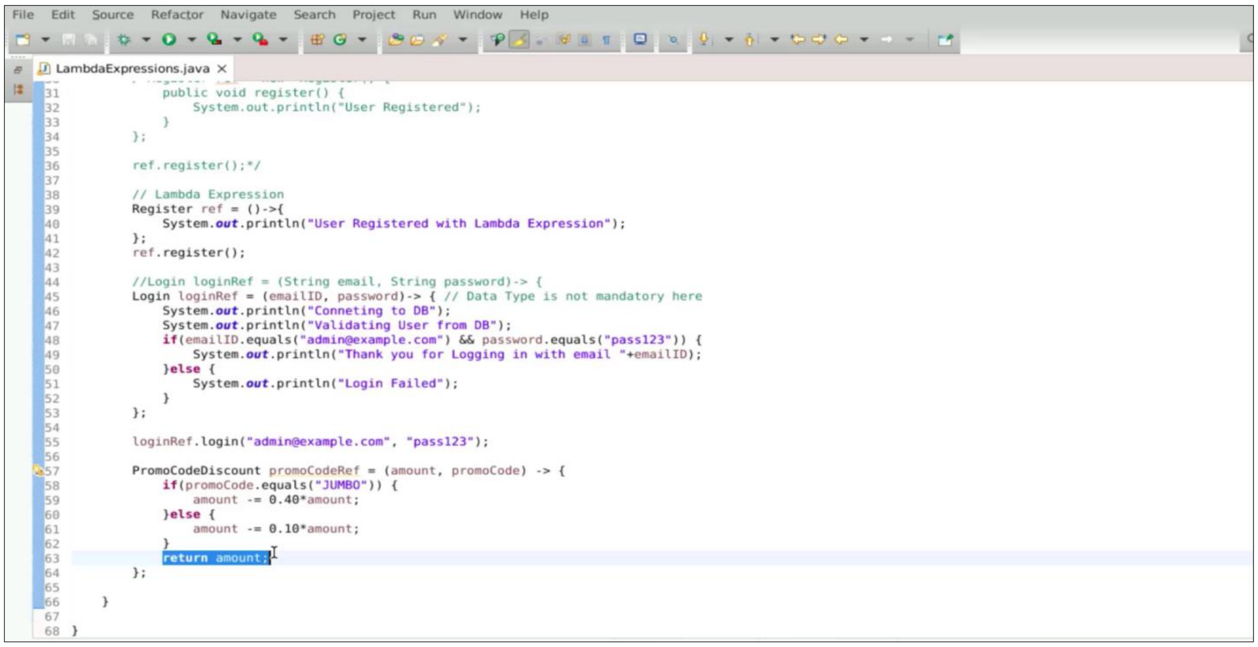
```
    public static void main(String[] args) {

        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/

        // Lambda Expression
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        //Login loginRef = (String email, String password)-> {
        Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            if(emailID.equals("admin@example.com") && password.equals("pass123")) {
                System.out.println("Thank you for Logging in with email "+emailID);
            }else {
                System.out.println("Login Failed");
            }
        };

        loginRef.login("admin@example.com", "pass123");

        PromoCodeDiscount promoCodeRef = (promoCode) -> {

        };

    }
```

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

```
        /*Register ref = new User();
        ref.register();*/

        /*Register ref = new  Register() {
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/

        // Lambda Expression
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        //Login loginRef = (String email, String password)-> {
        Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            if(emailID.equals("admin@example.com") && password.equals("pass123")) {
                System.out.println("Thank you for Logging in with email "+emailID);
            }else {
                System.out.println("Login Failed");
            }
        };

        loginRef.login("admin@example.com", "pass123");

        PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            if(promoCode.equals("JUMBO")) {

            }
        };

    }
```

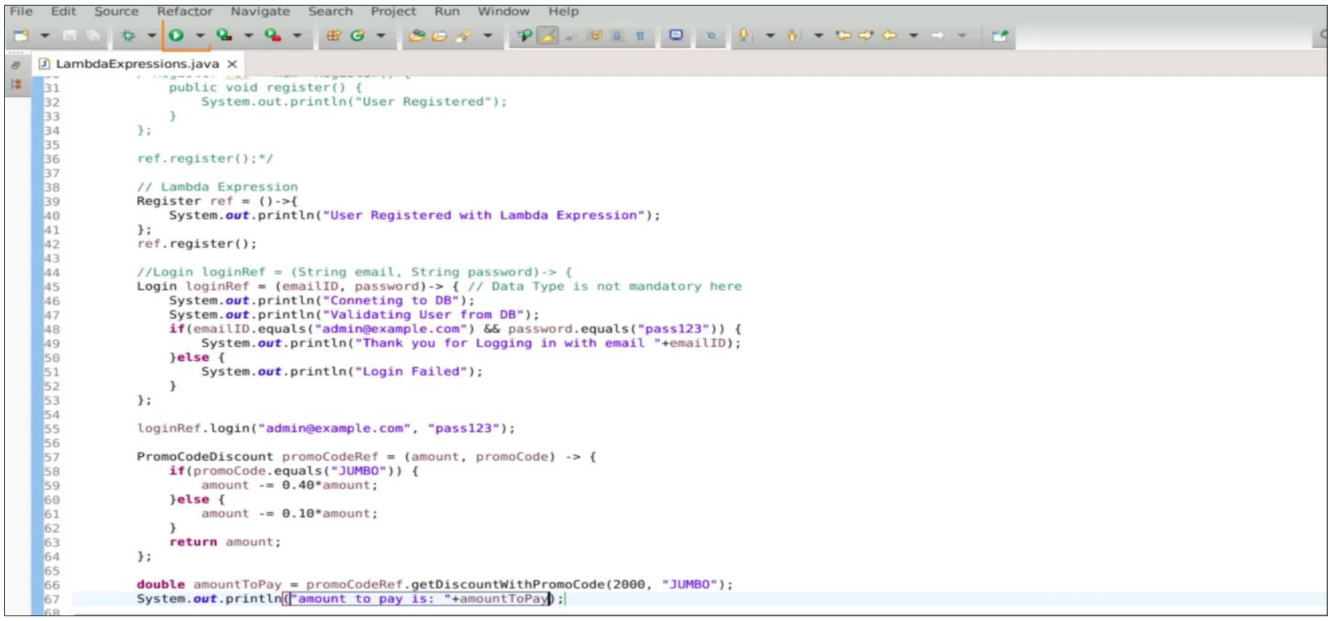
6.14 Calculate the updated amount by subtracting 40% for the Jumbo promo code, or apply a flat 10% off for other promo codes. Return the updated amount as the result.


File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java

```java
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/

        // Lambda Expression
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        //Login loginRef = (String email, String password)-> {
        Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            if(emailID.equals("admin@example.com") && password.equals("pass123")) {
                System.out.println("Thank you for Logging in with email "+emailID);
            }else {
                System.out.println("Login Failed");
            }
        };

        loginRef.login("admin@example.com", "pass123");

        PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            if(promoCode.equals("JUMBO")) {
                amount -= 0.40*amount;
            }else {
                amount -= 0.10*amount;
            }
            return amount;
        };

    }

}
```

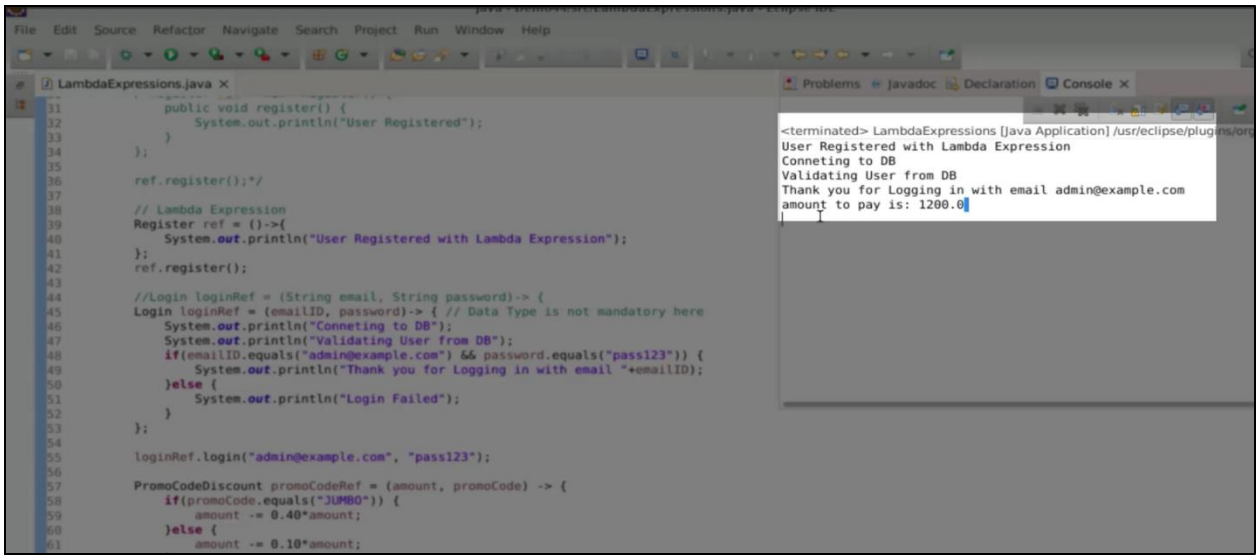
Step 7: Execute the Lambda expression with example data

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

```java
            public void register() {
                System.out.println("User Registered");
            }
        };

        ref.register();*/

        // Lambda Expression
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        //Login loginRef = (String email, String password)-> {
        Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            if(emailID.equals("admin@example.com") && password.equals("pass123")) {
                System.out.println("Thank you for Logging in with email "+emailID);
            }else {
                System.out.println("Login Failed");
            }
        };

        loginRef.login("admin@example.com", "pass123");

        PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            if(promoCode.equals("JUMBO")) {
                amount -= 0.40*amount;
            }else {
                amount -= 0.10*amount;
            }
            return amount;
        };

        double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "JUMBO");
        System.out.println("amount to pay is: "+amountToPay);
```

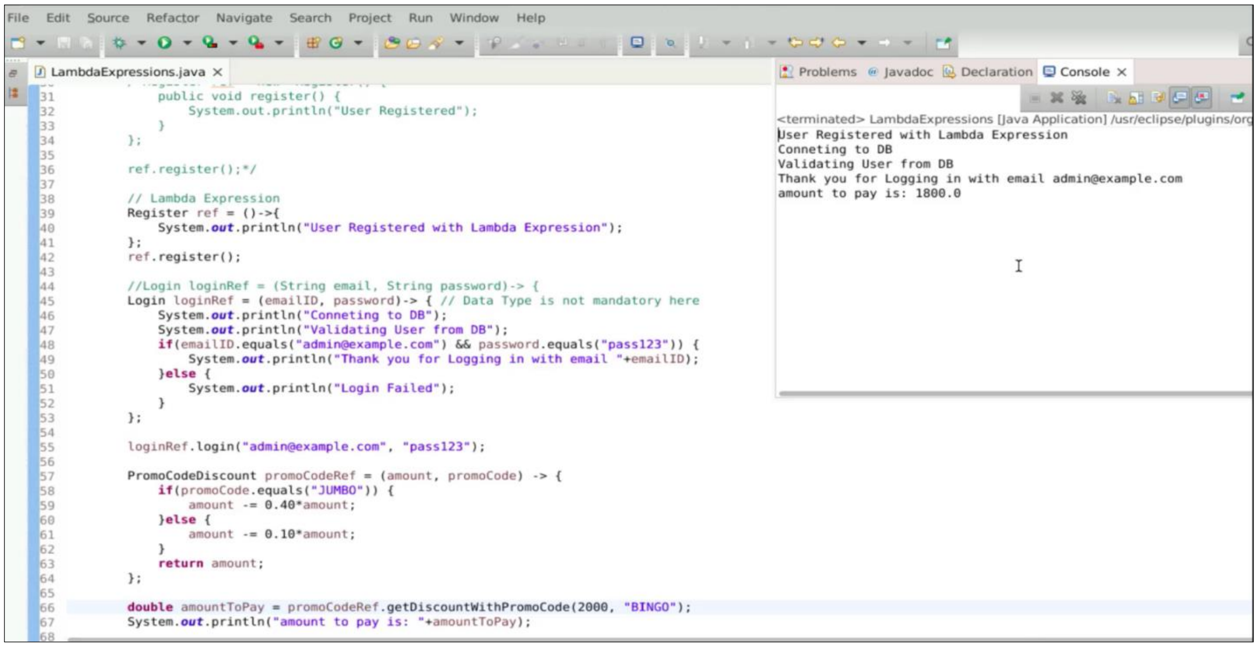
7.2 When you run the program, it shows the amount to pay is 1200



```java
        public void register() {
            System.out.println("User Registered");
        }
    };

    ref.register();*/

    // Lambda Expression
    Register ref = ()->{
        System.out.println("User Registered with Lambda Expression");
    };
    ref.register();

    //Login loginRef = (String email, String password)-> {
    Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
        System.out.println("Conneting to DB");
        System.out.println("Validating User from DB");
        if(emailID.equals("admin@example.com") && password.equals("pass123")) {
            System.out.println("Thank you for Logging in with email "+emailID);
        }else {
            System.out.println("Login Failed");
        }
    };

    loginRef.login("admin@example.com", "pass123");

    PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
        if(promoCode.equals("JUMBO")) {
            amount -= 0.40*amount;
        }else {
            amount -= 0.10*amount;
```

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

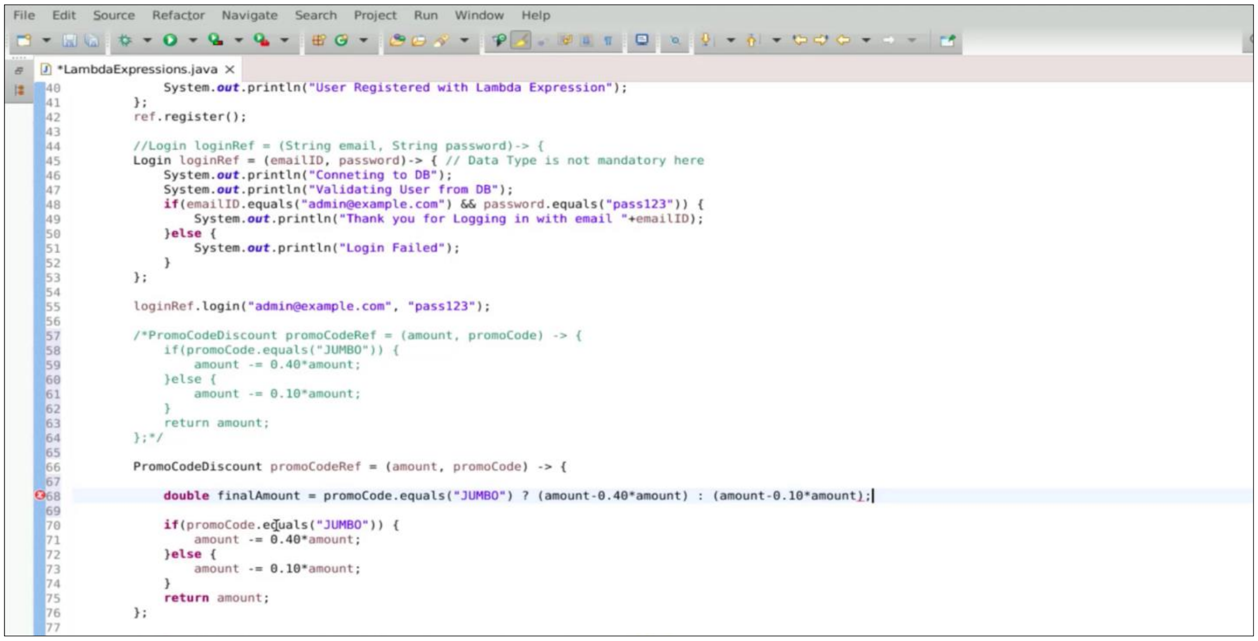
7.3 Let's assume that you replace it with bingo and rerun this program. Then it shows the amount to pay is 1800



```java
        public void register() {
            System.out.println("User Registered");
        }
    };

    ref.register();*/

    // Lambda Expression
    Register ref = ()->{
        System.out.println("User Registered with Lambda Expression");
    };
    ref.register();

    //Login loginRef = (String email, String password)-> {
    Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
        System.out.println("Conneting to DB");
        System.out.println("Validating User from DB");
        if(emailID.equals("admin@example.com") && password.equals("pass123")) {
            System.out.println("Thank you for Logging in with email "+emailID);
        }else {
            System.out.println("Login Failed");
        }
    };

    loginRef.login("admin@example.com", "pass123");

    PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
        if(promoCode.equals("JUMBO")) {
            amount -= 0.40*amount;
        }else {
            amount -= 0.10*amount;
        }
        return amount;
    };

    double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
    System.out.println("amount to pay is: "+amountToPay);
```

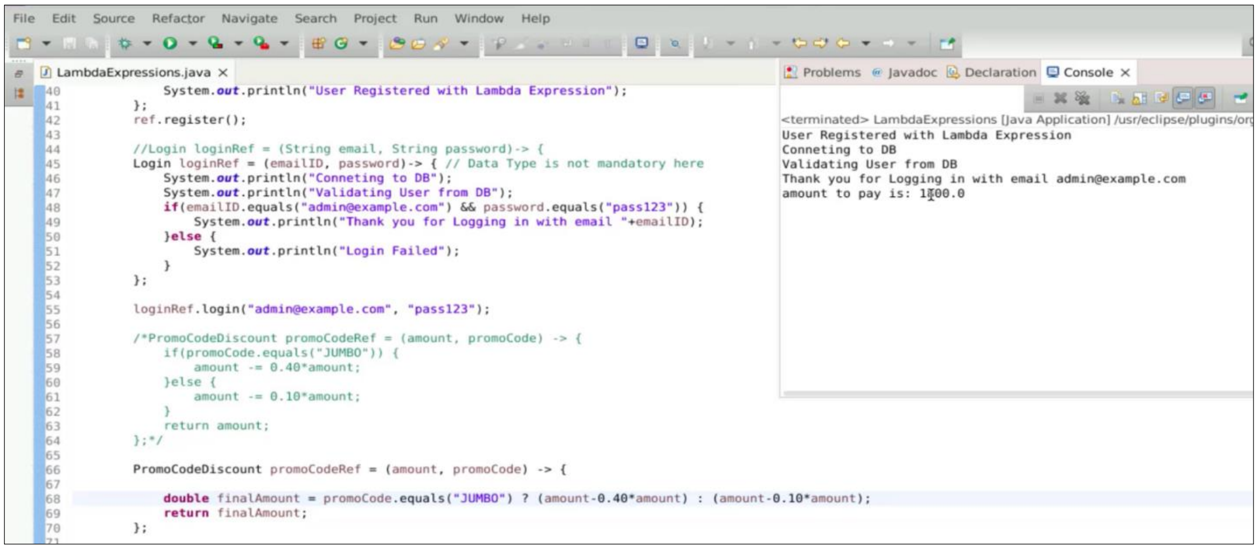
```
<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/org
User Registered with Lambda Expression
Conneting to DB
Validating User from DB
Thank you for Logging in with email admin@example.com
amount to pay is: 1800.0
```

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

```
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        //Login loginRef = (String email, String password)-> {
        Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            if(emailID.equals("admin@example.com") && password.equals("pass123")) {
                System.out.println("Thank you for Logging in with email "+emailID);
            }else {
                System.out.println("Login Failed");
            }
        };

        loginRef.login("admin@example.com", "pass123");

        /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            if(promoCode.equals("JUMBO")) {
                amount -= 0.40*amount;
            }else {
                amount -= 0.10*amount;
            }
            return amount;
        };*/

        PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {

            double finalAmount = promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);

            if(promoCode.equals("JUMBO")) {
                amount -= 0.40*amount;
            }else {
                amount -= 0.10*amount;
            }
            return amount;
        };
```

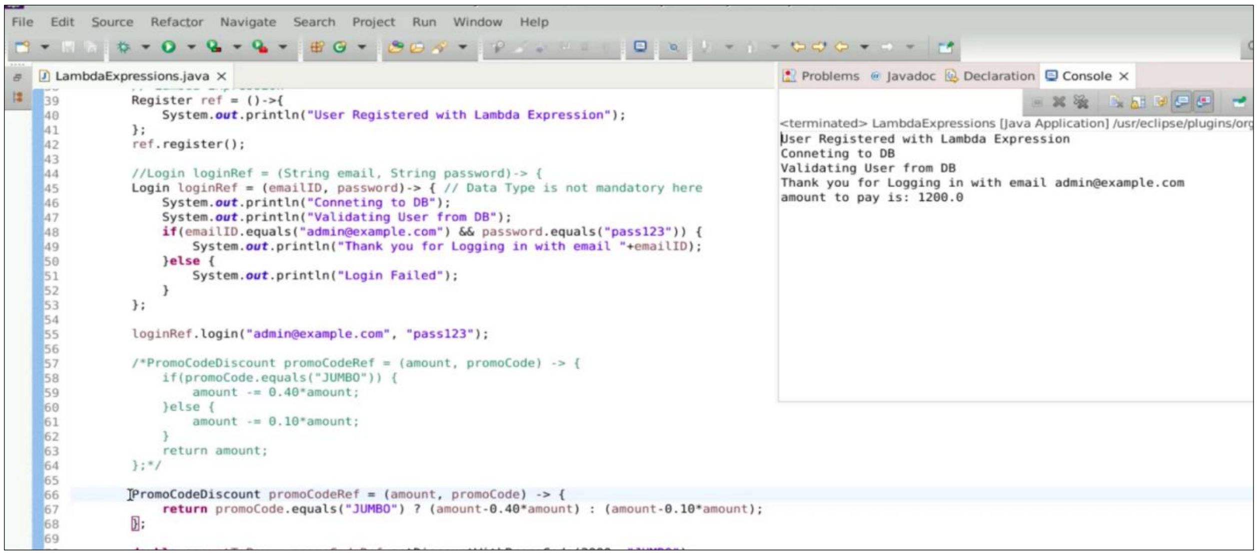
7.5 Conclude the optimized code by returning the **finalAmount**. This version utilizes the ternary operator to reduce the previous code and provides a 10% discount by default, while the jumbo promo code gives a 40% discount


File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java
Problems Javadoc Declaration Console

```
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        //Login loginRef = (String email, String password)-> {
        Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            if(emailID.equals("admin@example.com") && password.equals("pass123")) {
                System.out.println("Thank you for Logging in with email "+emailID);
            }else {
                System.out.println("Login Failed");
            }
        };

        loginRef.login("admin@example.com", "pass123");

        /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            if(promoCode.equals("JUMBO")) {
                amount -= 0.40*amount;
            }else {
                amount -= 0.10*amount;
            }
            return amount;
        };*/

        PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {

            double finalAmount = promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
            return finalAmount;
        };
```

```
<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
amount to pay is: 1800.0
```

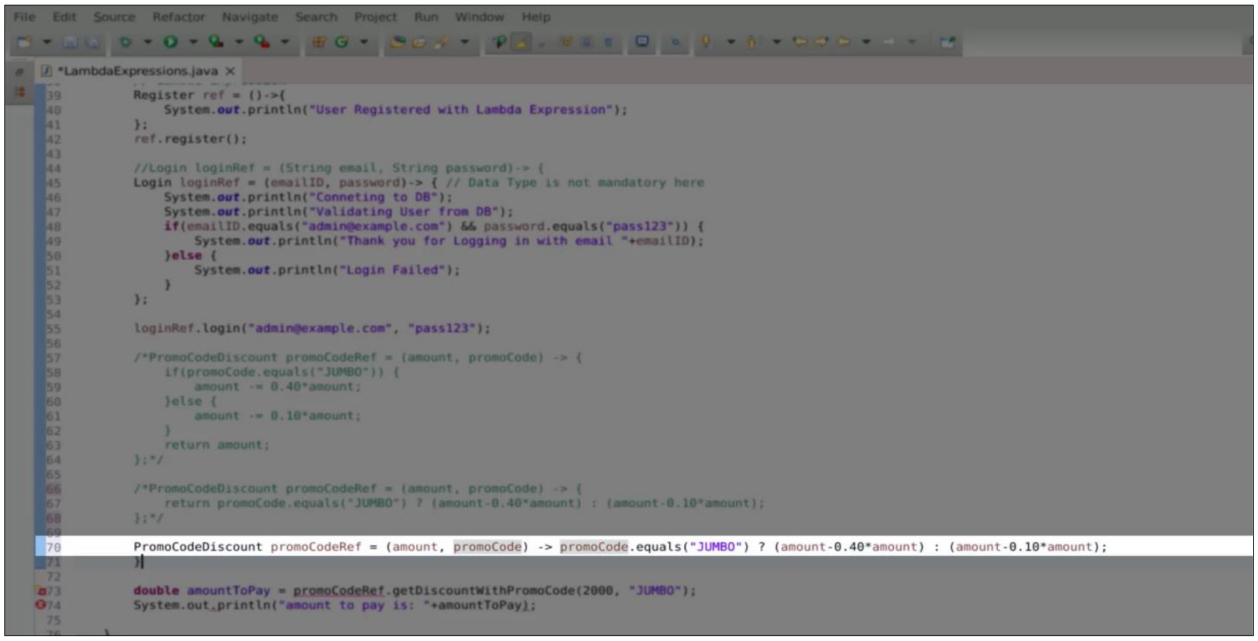
7.6 The benefit of using the ternary operator is that it allows for a more concise and compact Lambda expression implementation, resulting in code that works in the same way when executed



```
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        //Login loginRef = (String email, String password)-> {
        Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            if(emailID.equals("admin@example.com") && password.equals("pass123")) {
                System.out.println("Thank you for Logging in with email "+emailID);
            }else {
                System.out.println("Login Failed");
            }
        };

        loginRef.login("admin@example.com", "pass123");

        /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            if(promoCode.equals("JUMBO")) {
                amount -= 0.40*amount;
            }else {
                amount -= 0.10*amount;
            }
            return amount;
        };*/

        PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
        };
```

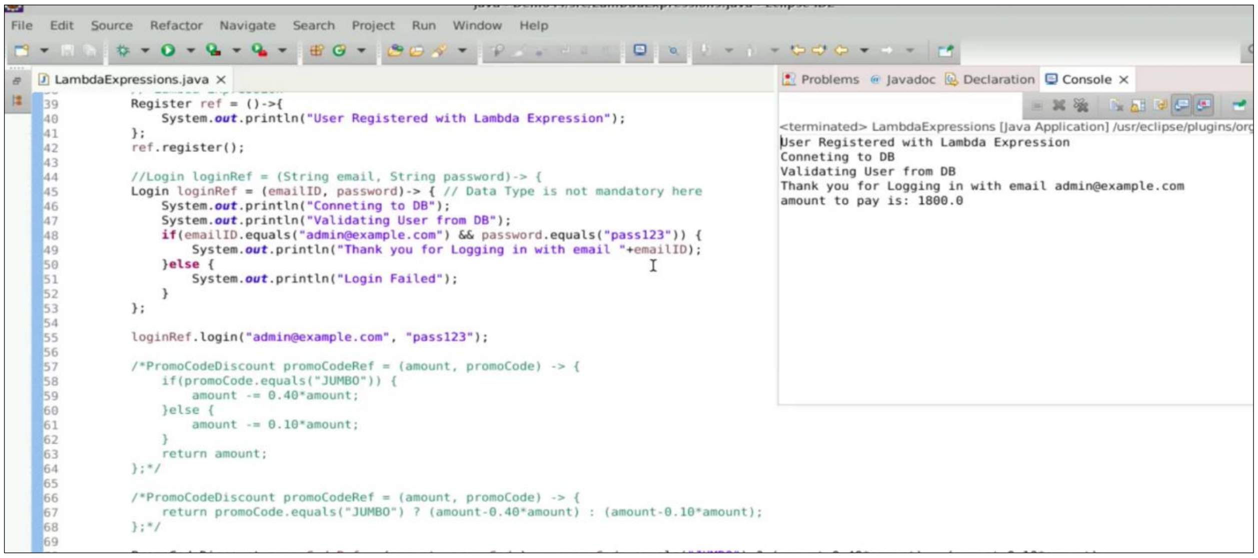
```
<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/org
User Registered with Lambda Expression
Conneting to DB
Validating User from DB
Thank you for Logging in with email admin@example.com
amount to pay is: 1200.0
```

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



```
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        //Login loginRef = (String email, String password)-> {
        Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            if(emailID.equals("admin@example.com") && password.equals("pass123")) {
                System.out.println("Thank you for Logging in with email "+emailID);
            }else {
                System.out.println("Login Failed");
            }
        };

        loginRef.login("admin@example.com", "pass123");

        /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            if(promoCode.equals("JUMBO")) {
                amount -= 0.40*amount;
            }else {
                amount -= 0.10*amount;
            }
            return amount;
        };*/

        /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
        };*/

        PromoCodeDiscount promoCodeRef = (amount, promoCode) -> promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
        }

        double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "JUMBO");
        System.out.println("amount to pay is: "+amountToPay);
```

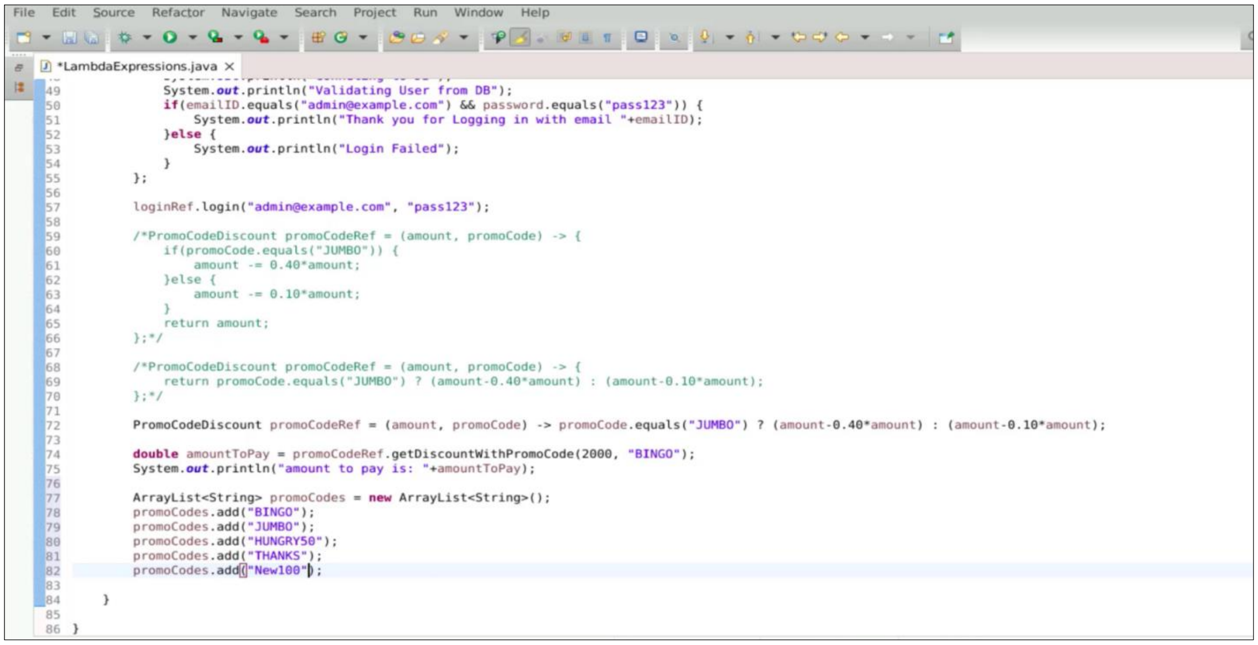
7.8 Run the same code, and you will see the correct output for jumbo and bingo. You will notice that it will provide a lesser discount. Hence, this is what lambda expressions are all about


File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java
Problems Javadoc Declaration Console

```
        Register ref = ()->{
            System.out.println("User Registered with Lambda Expression");
        };
        ref.register();

        //Login loginRef = (String email, String password)-> {
        Login loginRef = (emailID, password)-> { // Data Type is not mandatory here
            System.out.println("Conneting to DB");
            System.out.println("Validating User from DB");
            if(emailID.equals("admin@example.com") && password.equals("pass123")) {
                System.out.println("Thank you for Logging in with email "+emailID);
            }else {
                System.out.println("Login Failed");
            }
        };

        loginRef.login("admin@example.com", "pass123");

        /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            if(promoCode.equals("JUMBO")) {
                amount -= 0.40*amount;
            }else {
                amount -= 0.10*amount;
            }
            return amount;
        };*/

        /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
        };*/
```

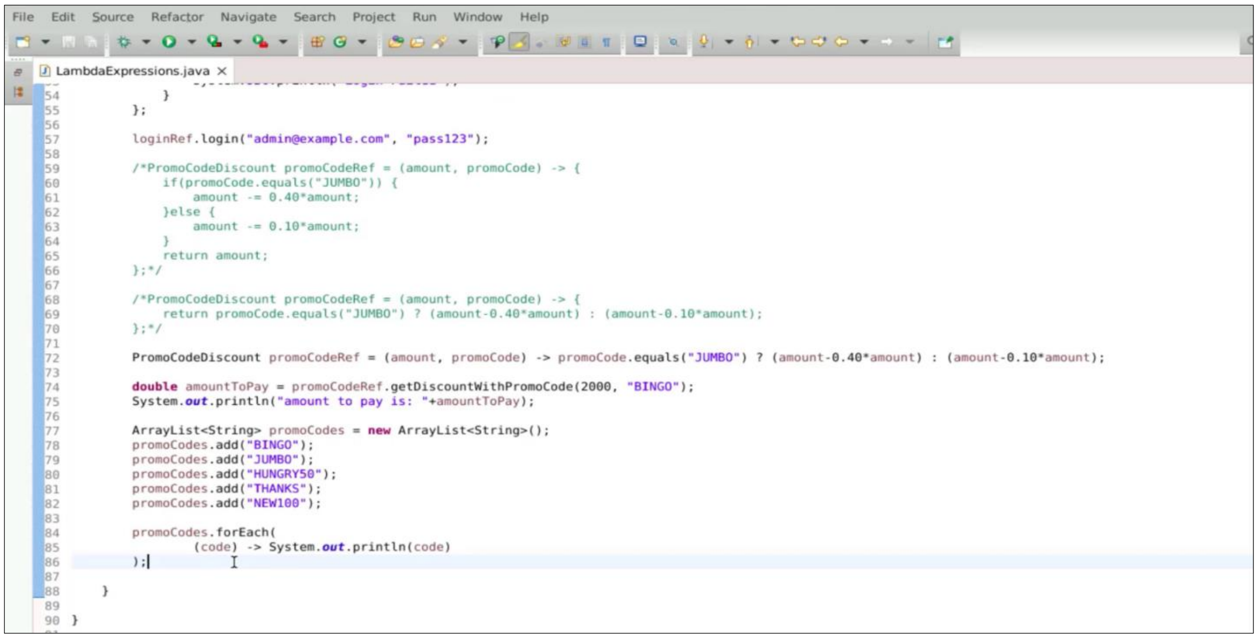
```
<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/org
User Registered with Lambda Expression
Conneting to DB
Validating User from DB
Thank you for Logging in with email admin@example.com
amount to pay is: 1800.0
```

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

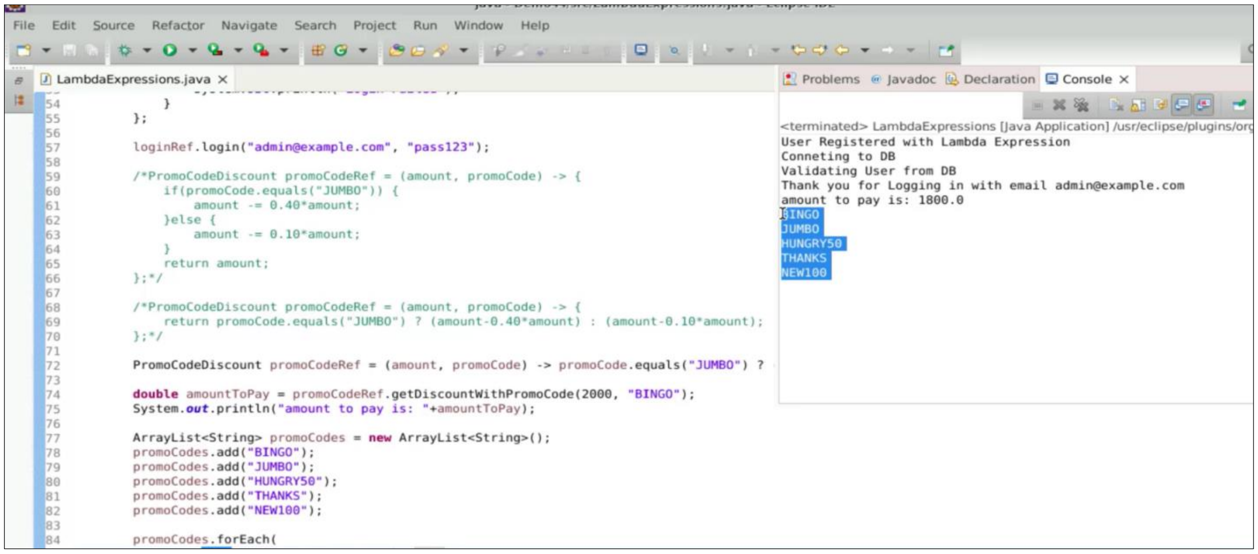
```
            System.out.println("Validating User from DB");
            if(emailID.equals("admin@example.com") && password.equals("pass123")) {
                System.out.println("Thank you for Logging in with email "+emailID);
            }else {
                System.out.println("Login Failed");
            }
        };

        loginRef.login("admin@example.com", "pass123");

        /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            if(promoCode.equals("JUMBO")) {
                amount -= 0.40*amount;
            }else {
                amount -= 0.10*amount;
            }
            return amount;
        };*/

        /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
        };*/

        PromoCodeDiscount promoCodeRef = (amount, promoCode) -> promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);

        double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
        System.out.println("amount to pay is: "+amountToPay);

        ArrayList<String> promoCodes = new ArrayList<String>();
        promoCodes.add("BINGO");
        promoCodes.add("JUMBO");
        promoCodes.add("HUNGRY50");
        promoCodes.add("THANKS");
        promoCodes.add("New100");

    }

}
```

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

```java
            }
        };

        loginRef.login("admin@example.com", "pass123");

        /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            if(promoCode.equals("JUMBO")) {
                amount -= 0.40*amount;
            }else {
                amount -= 0.10*amount;
            }
            return amount;
        };*/

        /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
        };*/

        PromoCodeDiscount promoCodeRef = (amount, promoCode) -> promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);

        double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
        System.out.println("amount to pay is: "+amountToPay);

        ArrayList<String> promoCodes = new ArrayList<String>();
        promoCodes.add("BINGO");
        promoCodes.add("JUMBO");
        promoCodes.add("HUNGRY50");
        promoCodes.add("THANKS");
        promoCodes.add("NEW100");

        promoCodes.forEach(
                (code) -> System.out.println(code)
        );

    }

}
```

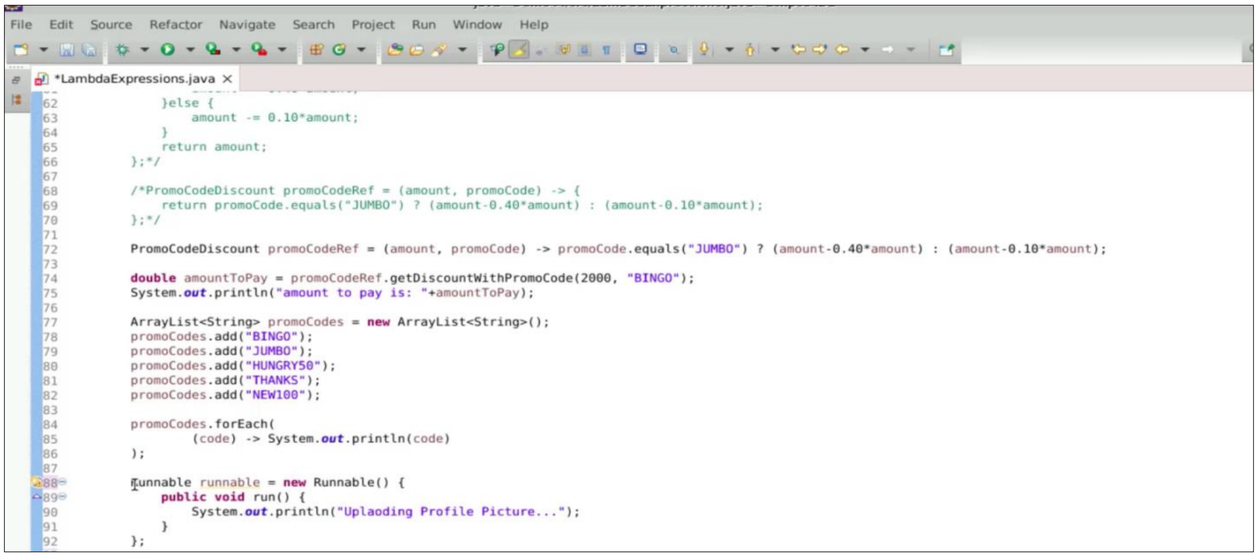
7.11 Run this code and that is how you can iterate through all your collections. Lambda expressions can be very useful when working in a multithreaded environment


File Edit Source Refactor Navigate Search Project Run Window Help
LambdaExpressions.java
Problems Javadoc Declaration Console

```java
            }
        };

        loginRef.login("admin@example.com", "pass123");

        /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            if(promoCode.equals("JUMBO")) {
                amount -= 0.40*amount;
            }else {
                amount -= 0.10*amount;
            }
            return amount;
        };*/

        /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
        };*/

        PromoCodeDiscount promoCodeRef = (amount, promoCode) -> promoCode.equals("JUMBO") ?

        double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
        System.out.println("amount to pay is: "+amountToPay);

        ArrayList<String> promoCodes = new ArrayList<String>();
        promoCodes.add("BINGO");
        promoCodes.add("JUMBO");
        promoCodes.add("HUNGRY50");
        promoCodes.add("THANKS");
        promoCodes.add("NEW100");

        promoCodes.forEach(
```

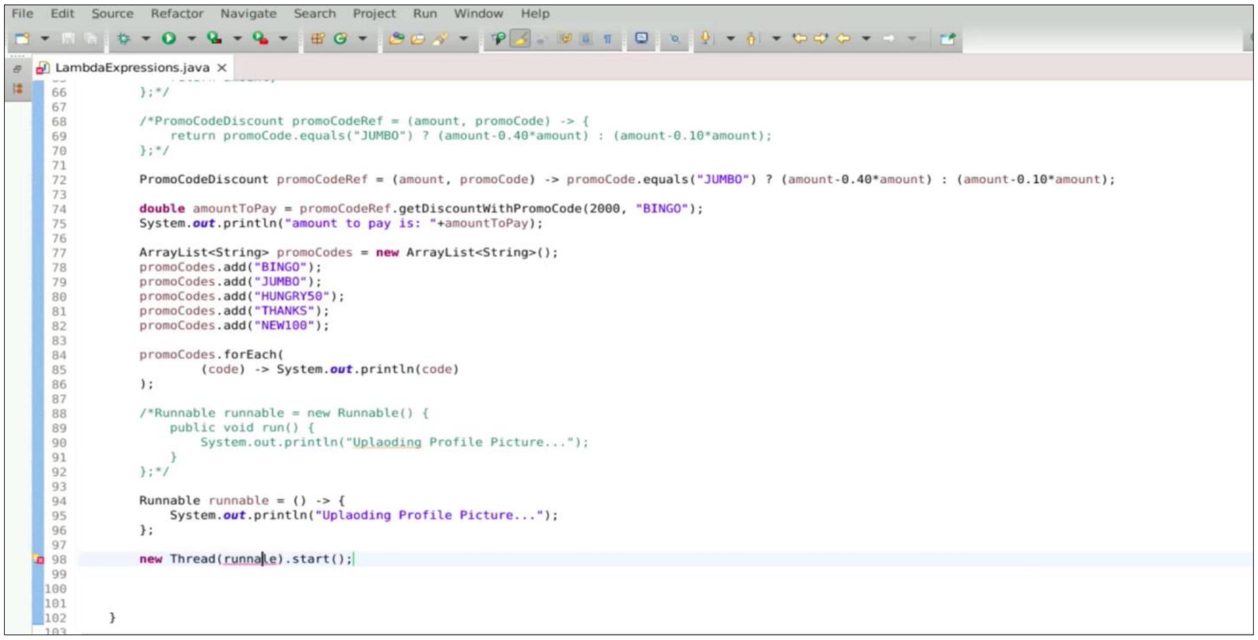
```
<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/org
User Registered with Lambda Expression
Conneting to DB
Validating User from DB
Thank you for Logging in with email admin@example.com
amount to pay is: 1800.0
BINGO
JUMBO
HUNGRY50
THANKS
NEW100
```

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

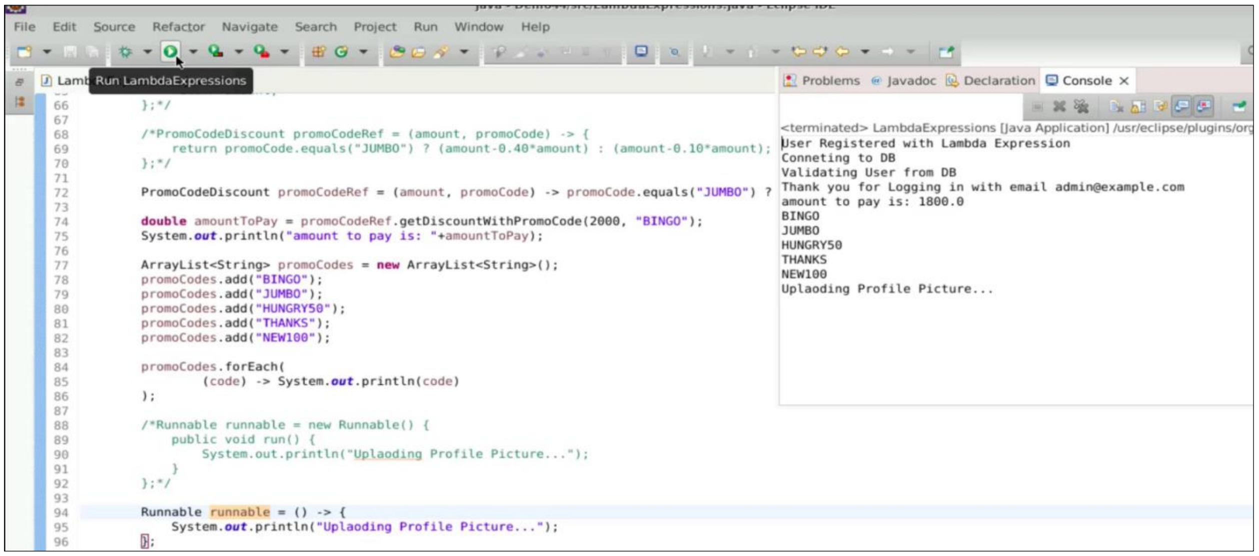
```
        }else {
            amount -= 0.10*amount;
        }
        return amount;
    };*/

    /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
        return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
    };*/

    PromoCodeDiscount promoCodeRef = (amount, promoCode) -> promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);

    double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
    System.out.println("amount to pay is: "+amountToPay);

    ArrayList<String> promoCodes = new ArrayList<String>();
    promoCodes.add("BINGO");
    promoCodes.add("JUMBO");
    promoCodes.add("HUNGRY50");
    promoCodes.add("THANKS");
    promoCodes.add("NEW100");

    promoCodes.forEach(
            (code) -> System.out.println(code)
    );

    Runnable runnable = new Runnable() {
        public void run() {
            System.out.println("Uplaoding Profile Picture...");
        }
    };
```

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

```
    };*/

    /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
        return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
    };*/

    PromoCodeDiscount promoCodeRef = (amount, promoCode) -> promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);

    double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
    System.out.println("amount to pay is: "+amountToPay);

    ArrayList<String> promoCodes = new ArrayList<String>();
    promoCodes.add("BINGO");
    promoCodes.add("JUMBO");
    promoCodes.add("HUNGRY50");
    promoCodes.add("THANKS");
    promoCodes.add("NEW100");

    promoCodes.forEach(
            (code) -> System.out.println(code)
    );

    /*Runnable runnable = new Runnable() {
        public void run() {
            System.out.println("Uplaoding Profile Picture...");
        }
    };*/

    Runnable runnable = () -> {
        System.out.println("Uplaoding Profile Picture...");
    };

    new Thread(runnable).start();



}
```

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



```java
        };*/

        /*PromoCodeDiscount promoCodeRef = (amount, promoCode) -> {
            return promoCode.equals("JUMBO") ? (amount-0.40*amount) : (amount-0.10*amount);
        };*/

        PromoCodeDiscount promoCodeRef = (amount, promoCode) -> promoCode.equals("JUMBO") ?

        double amountToPay = promoCodeRef.getDiscountWithPromoCode(2000, "BINGO");
        System.out.println("amount to pay is: "+amountToPay);

        ArrayList<String> promoCodes = new ArrayList<String>();
        promoCodes.add("BINGO");
        promoCodes.add("JUMBO");
        promoCodes.add("HUNGRY50");
        promoCodes.add("THANKS");
        promoCodes.add("NEW100");

        promoCodes.forEach(
                (code) -> System.out.println(code)
        );

        /*Runnable runnable = new Runnable() {
            public void run() {
                System.out.println("Uplaoding Profile Picture...");
            }
        };*/

        Runnable runnable = () -> {
            System.out.println("Uplaoding Profile Picture...");
        };
```

```
<terminated> LambdaExpressions [Java Application] /usr/eclipse/plugins/org
User Registered with Lambda Expression
Conneting 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
Uplaoding Profile Picture...
```

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