Lesson 02 Demo 04

Overloading Methods in Java

Objective: To depict how to overload methods in Java

Tools required: Eclipse IDE

Prerequisites: None

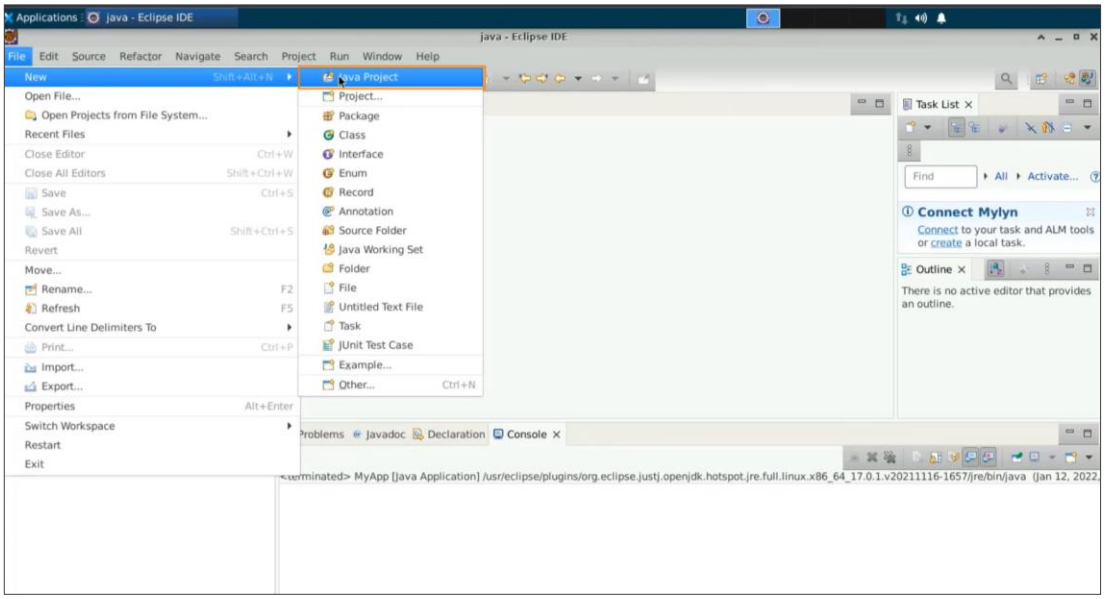
Steps to be followed:

1. Write and implement method overloading rules in Java
2. Implement compiler time polymorphism
3. Test the overloaded methods
4. Run the code and execute the result
5. Write methods for authentication

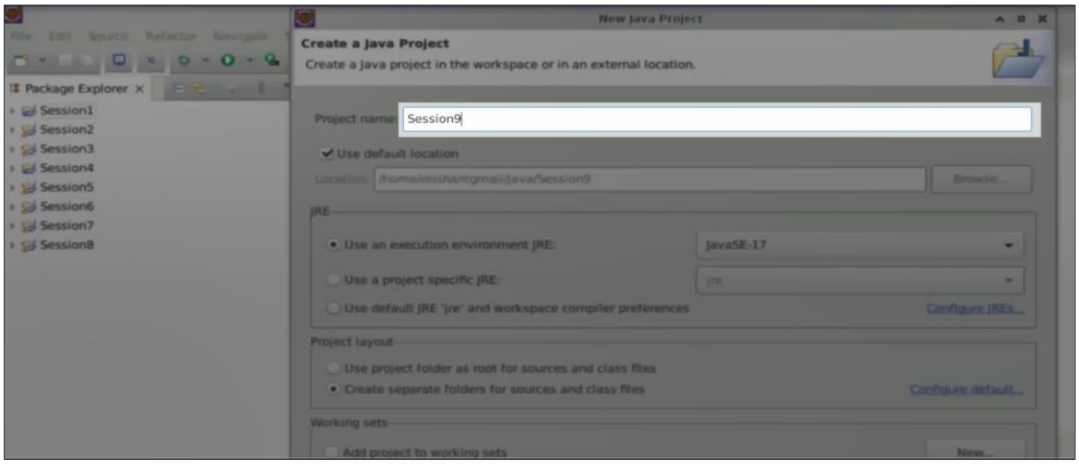
Step 1: Write and implement method overloading rules in Java

1.1 Open the **Eclipse IDE**

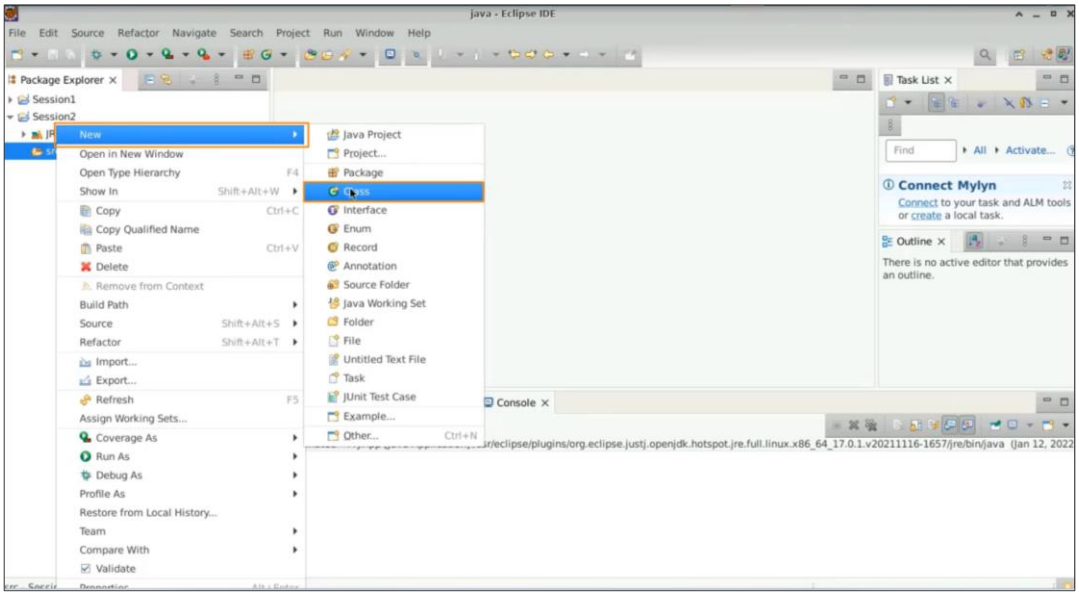
1.2. Select **File**, then **New,** and then **Java project**



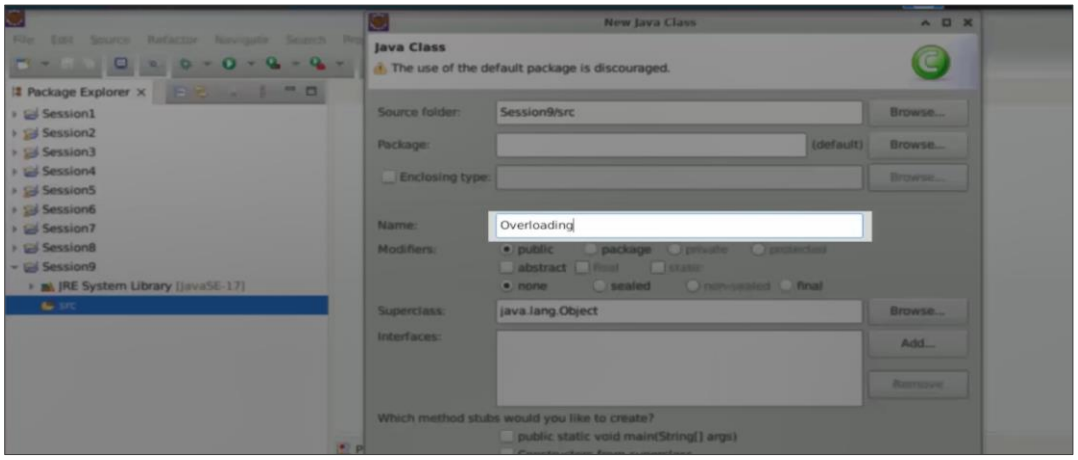
1.3 Name the project **“Session9”,** uncheck **“Create a module info dot Java file”**, and press **Finish**



1.4 With a **Session9** on the src, do a right-click and create a **new class**

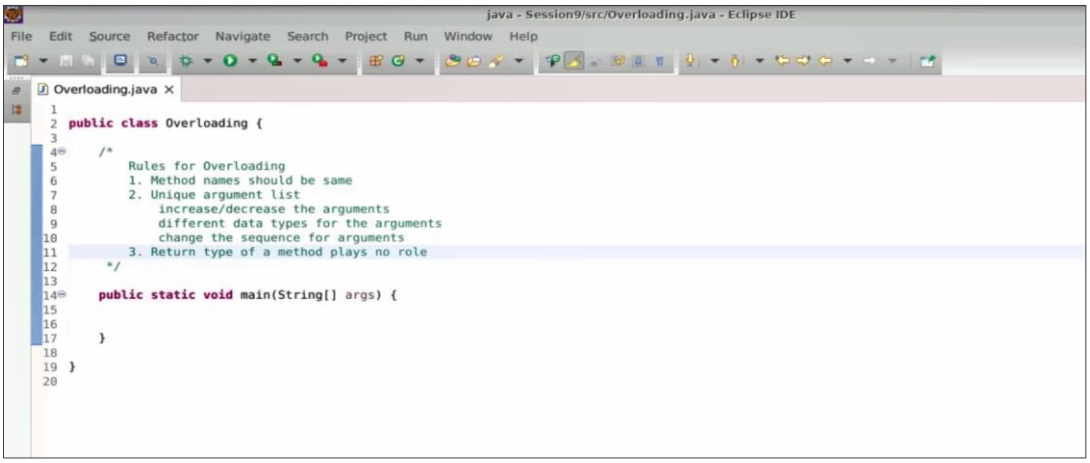


1.5 Name this class as an **Overloading**, then select the **main method,** and then select **finish**

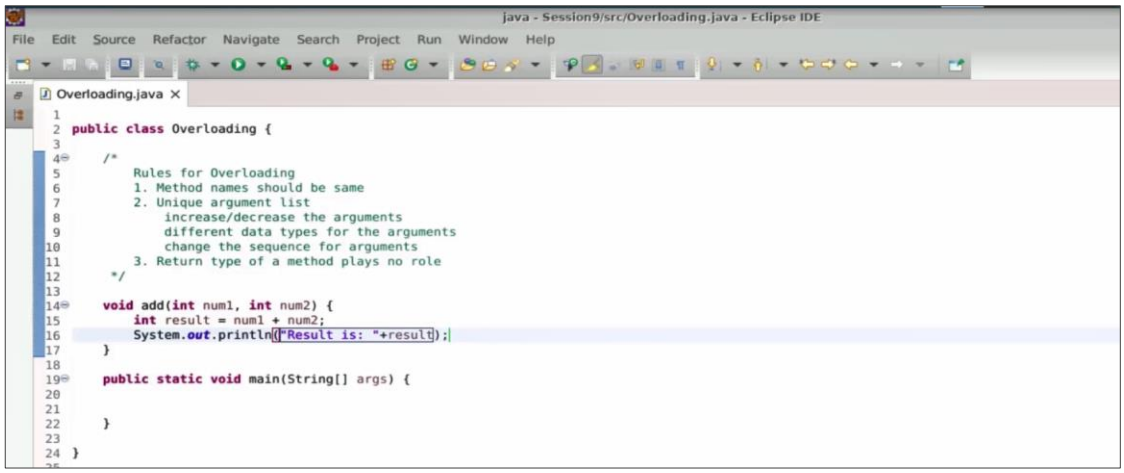


Step 2: Implement compiler time polymorphism

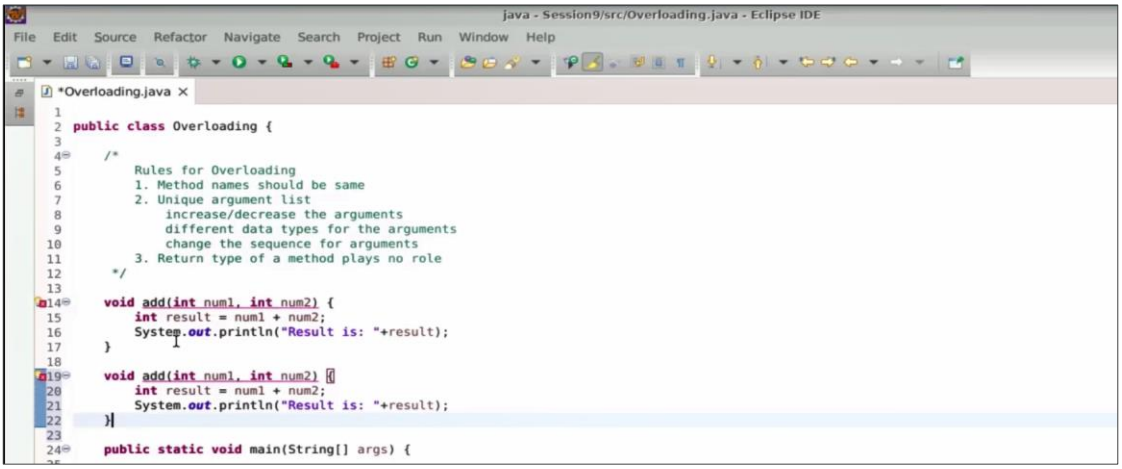
2.1 To implement overloading, you need to follow certain rules. The very first rule is, that the method name should be the same, the second rule will be a Unique Argument List, which means you can increase or decrease the arguments. Then you can have different Data types for the arguments, then you can even change the sequence for arguments, and the third rule is the return type of a method Plays no role



2.2 Create a method called add that takes two integer inputs, **integer1** and **integer2**. Write the equation **result = integer1 + integer2**. Then, print **Result is:** followed by the value of result

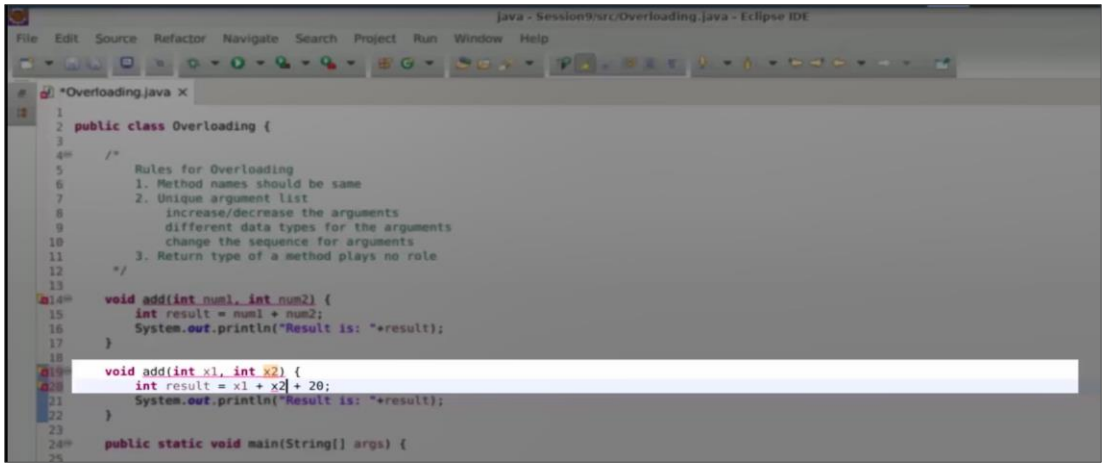


2.3 If in case you try to create the same method with the same name and the same inputs, you will get an error, hence there cannot be ambiguity for a method. Hence, you can have a method overload

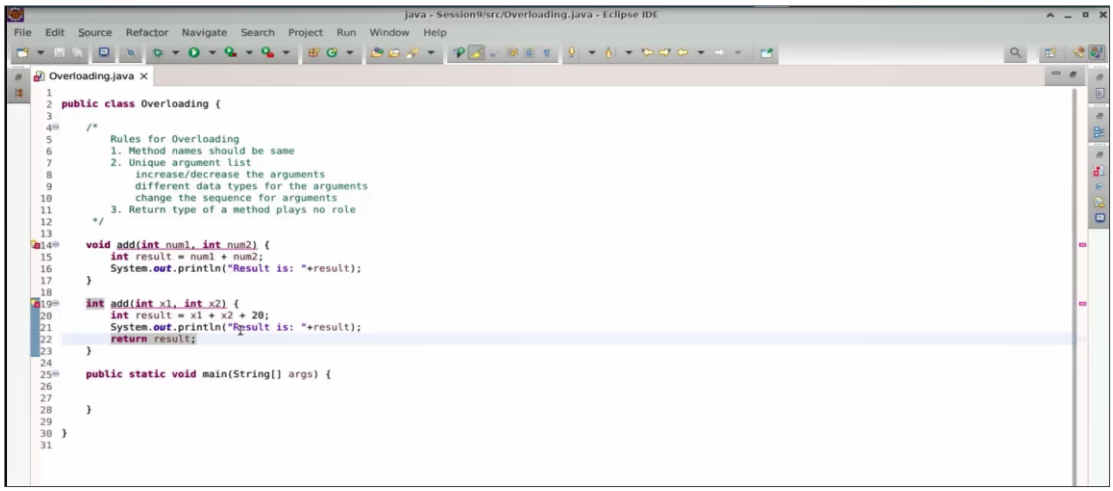


Step 3: Test the overloaded methods

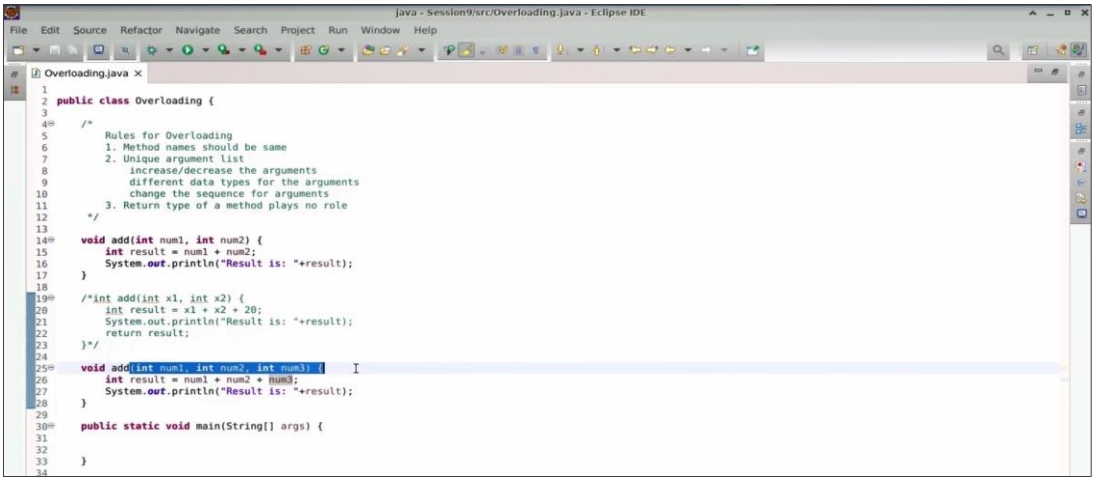
3.1 Now, let us use x1 and x2 as the variables, and define the result as x1 + x2 + 20 to ensure it would not have any impact. The uniqueness of the method comes from the list of arguments or inputs you are trying to pass. If you try to execute this method, it would be confusing whether the two integers should be passed as number1, number2, or x1, x2



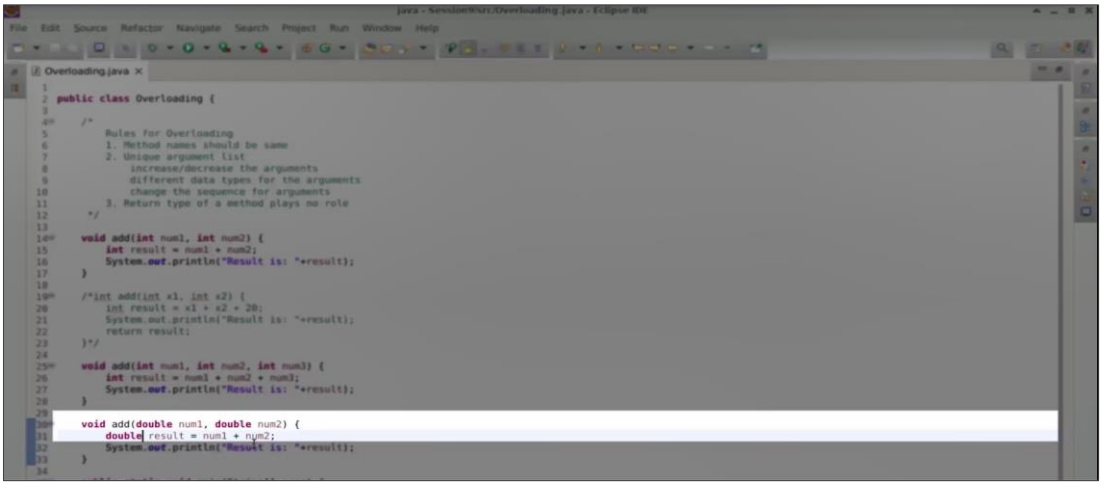
3.2 Now in a scenario, let the Return type be an integer, where you will write Return the result. This will also not be a solution for overloading



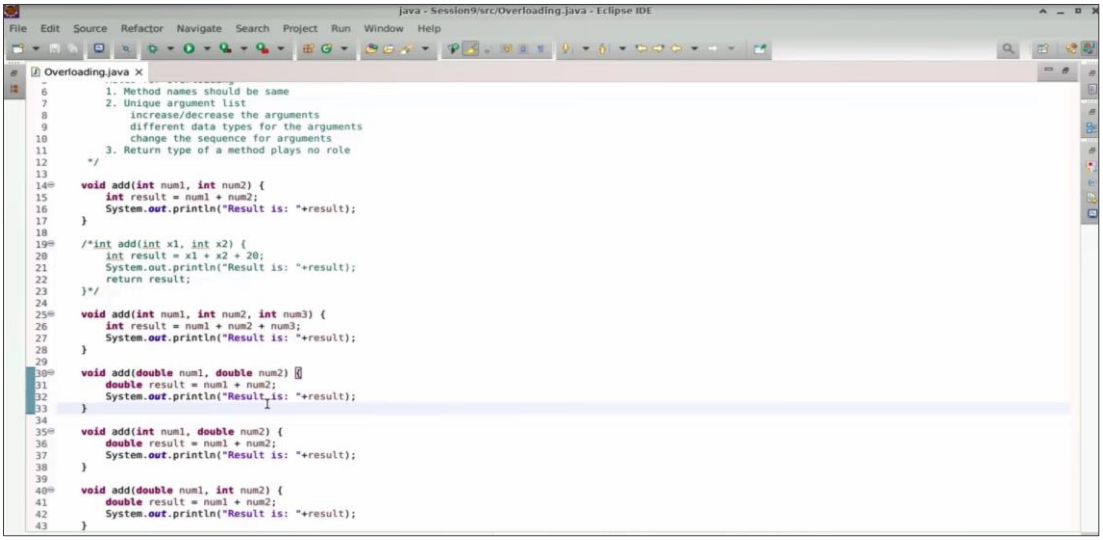
3.3 Write integer number 3. With another integer number 3, you can have a different unique list. The method name is the same, and this is where overloading has been implemented. This is known as compile-time polymorphism, where the compiler will distinguish based on the method execution call, which definition is supposed to be executed



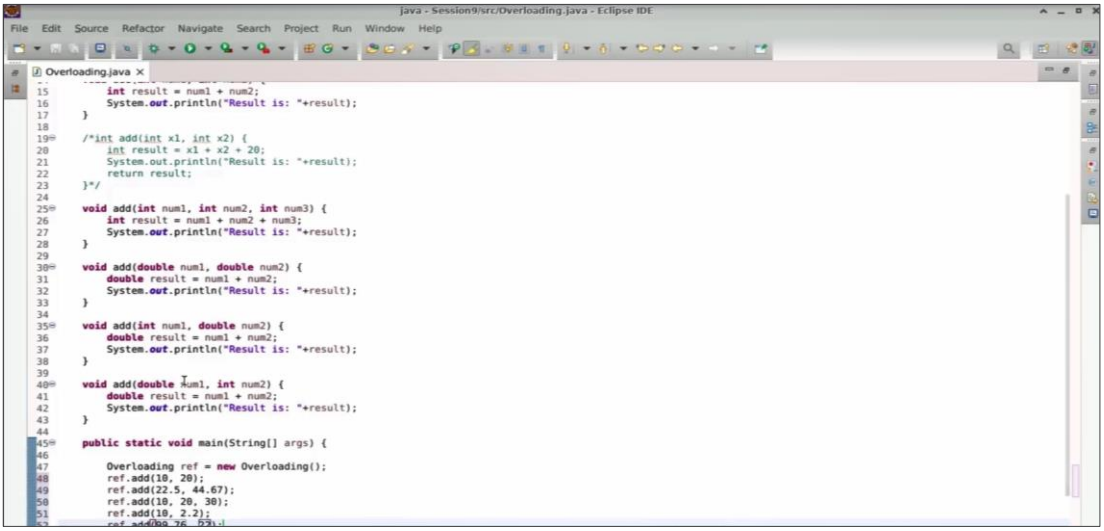
3.4 Instead of adding the numbers with integers, you can add the numbers as doubles, and this will also help us to implement the method overloading. That is the same name with the different inputs



3.5 Now, for the sequence of the arguments, there can be an integer number1 and a double number2, and then you can change the sequence to double number1 and integer number2

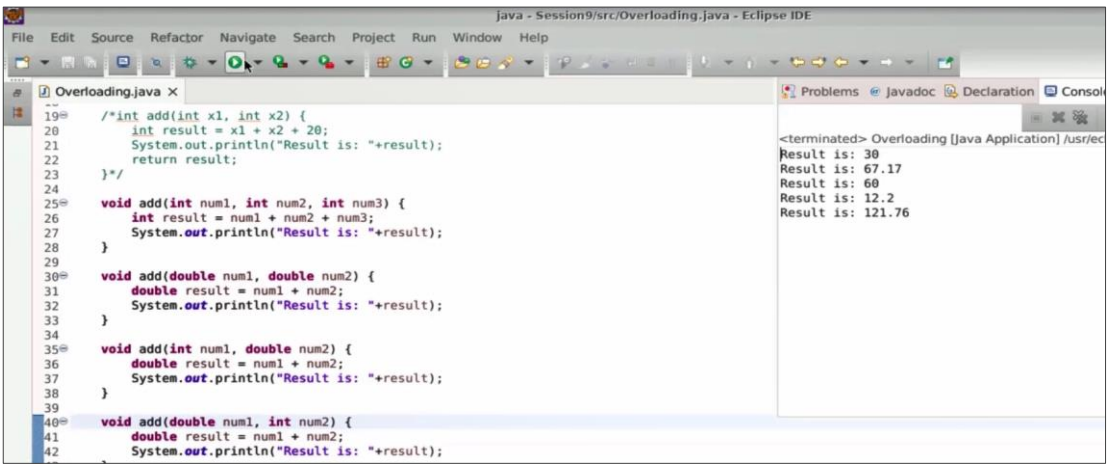


3.6 Create a reference variable and use method overloading. Write add, and inside the add method, you can pass two integers to add two numbers. With the same reference, you can call add to pass two doubles or floating-point numbers. Then with your reference, you can write **add(10, 20, 30)**. With the same reference, you can add one integer and the next one as a double. You can even use **reference.add(double, int)**

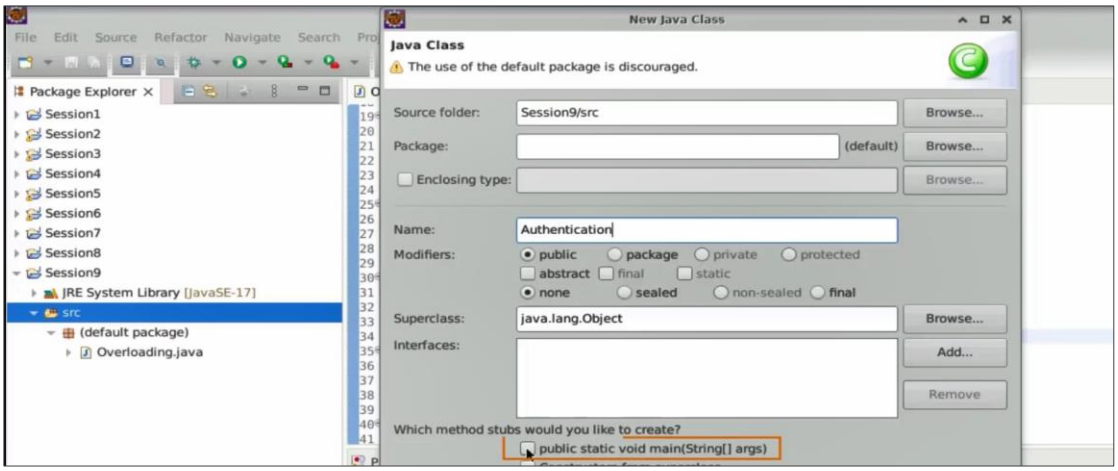


Step 4: Run the code and execute the result

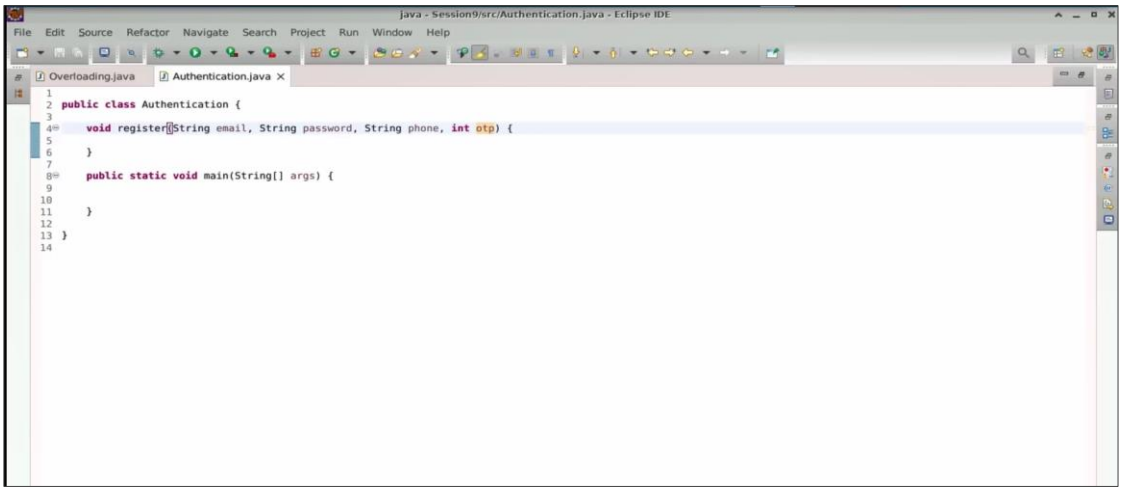
4.1 Run this code, you can now see that all of the methods are executed and you are able to compute the result



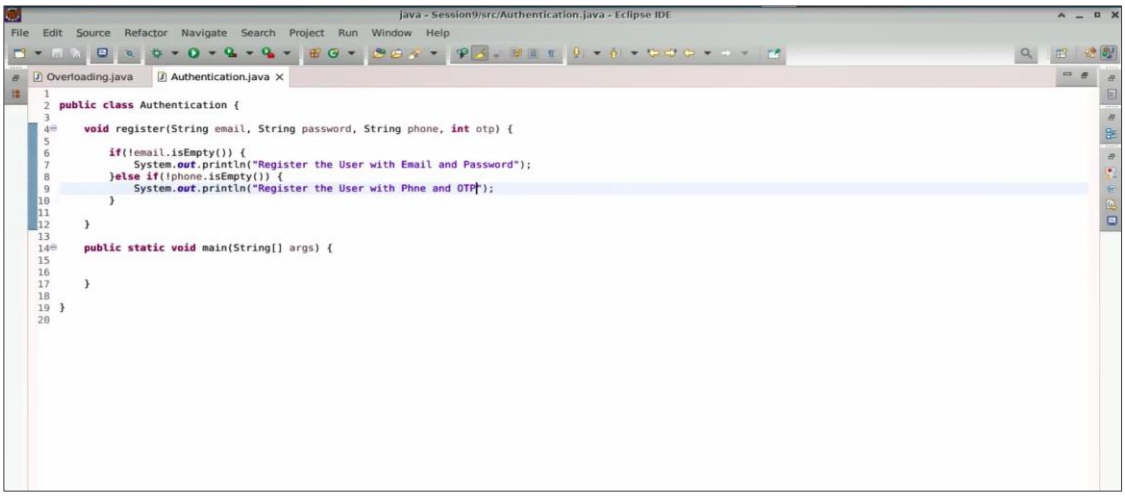
4.2 Create a new class and name this as authentication, with the main method.



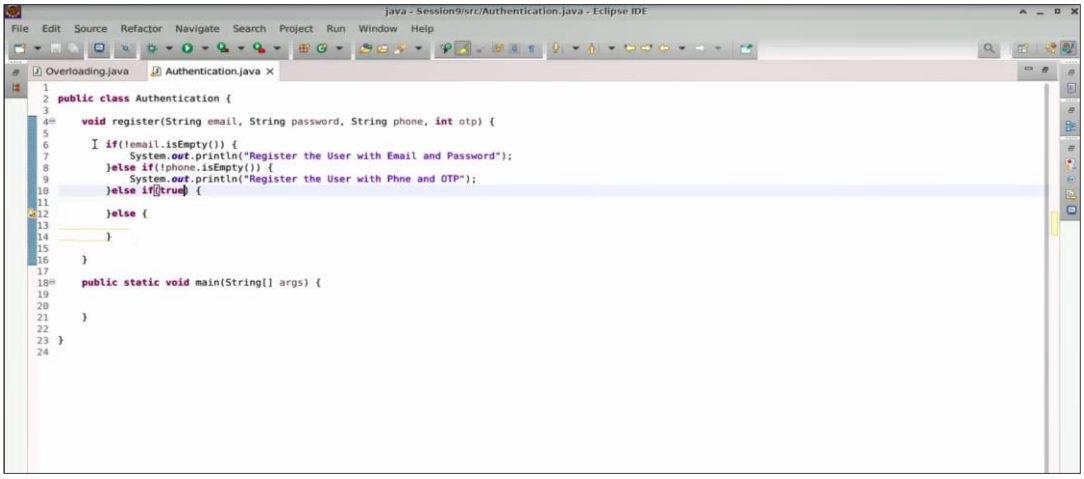
4.3 If you want the user to register in your application, create a method called register. For the registration, there can be several inputs. The first input is email, and the second input is password. Next, you will pass one more input for example, this input is a phone number and an integer called OTP



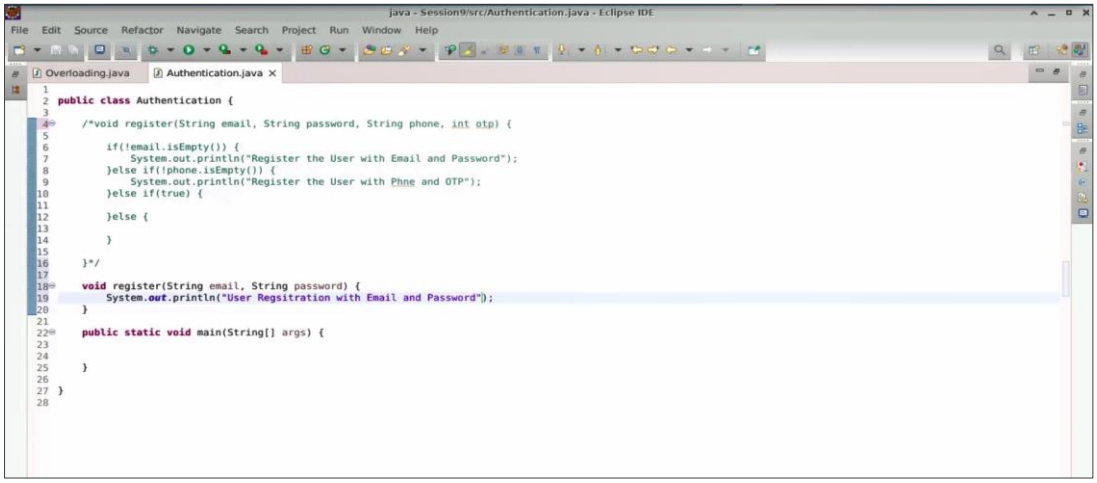
4.4 Add a simple check: if the email is not empty using **(!email.isEmpty())**, then register the user with an email and password. Create an else-if case: if the phone number is not empty, register the user with the phone number and OTP



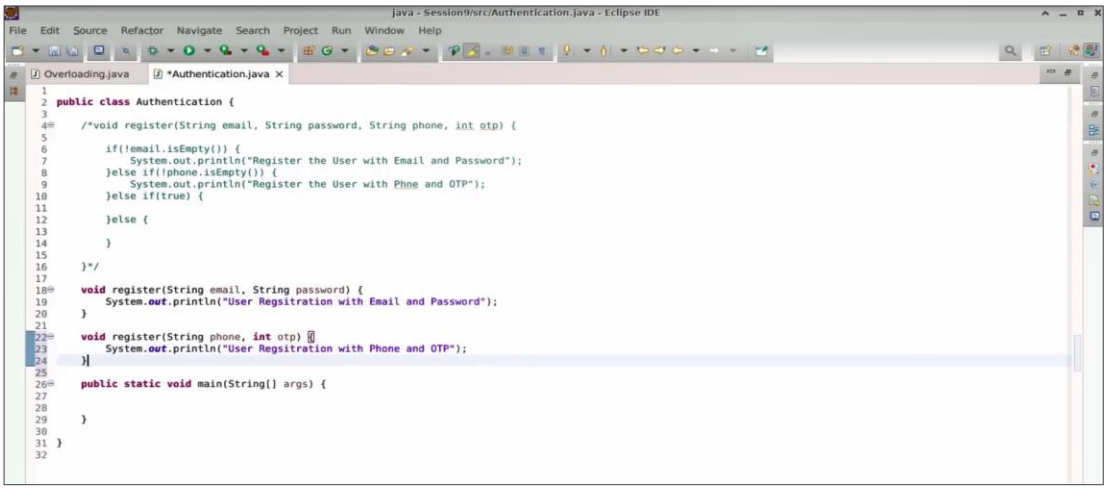
4.5 In case you need to add more registration methods, you need to have a lot of other else If structures, and finally an else. So you can break down this method with the help of overloading



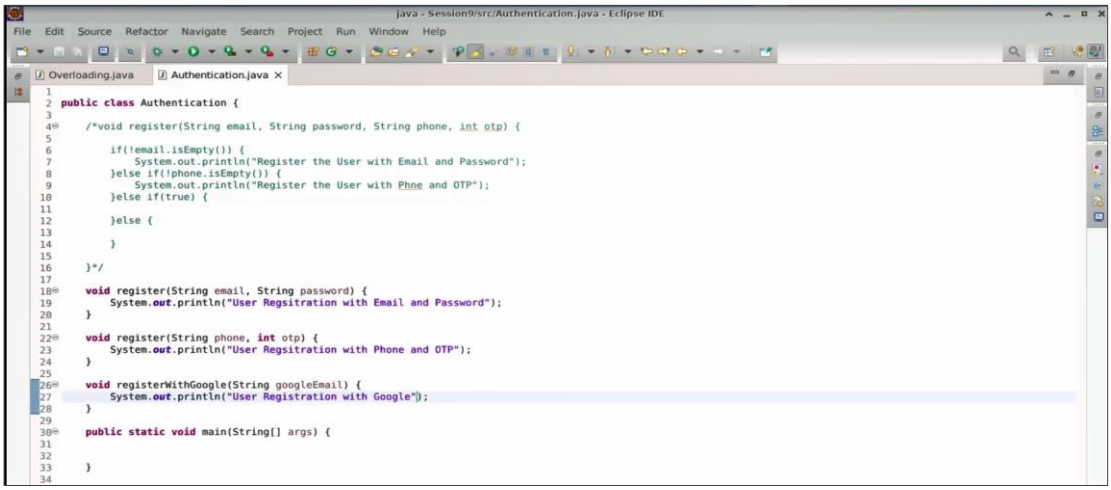
4.6 You can write void register, and you can take the first thing as email and the second thing as a password. Next, print down the user registration with email and password



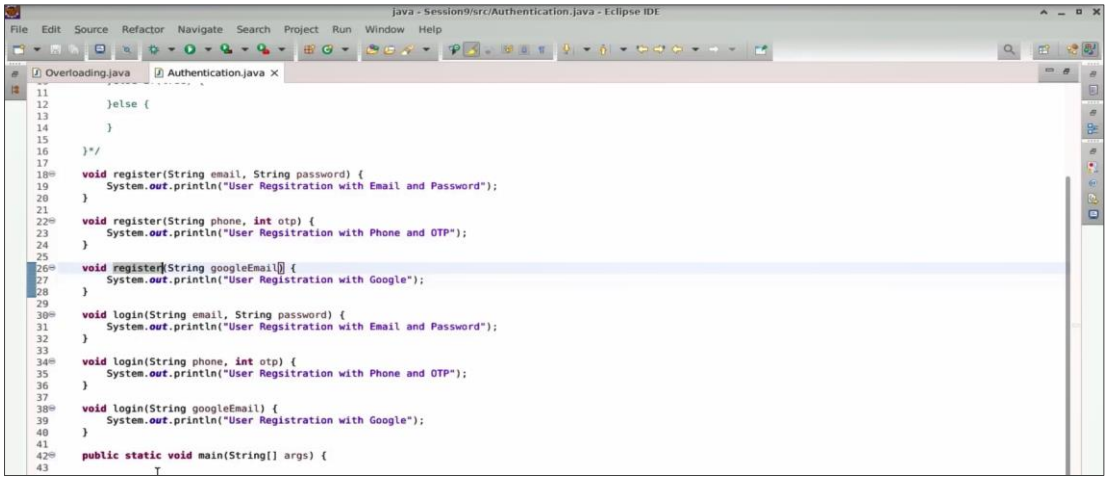
4.7 The other one can be your string phone number and an integer OTP that is a One-time password. Here, user registration with phone and OTP



4.8 Next, you can even Register with Google, where you will take one Google email as input and print down this as user registration with Google. And you can have different ways to do registration for a user

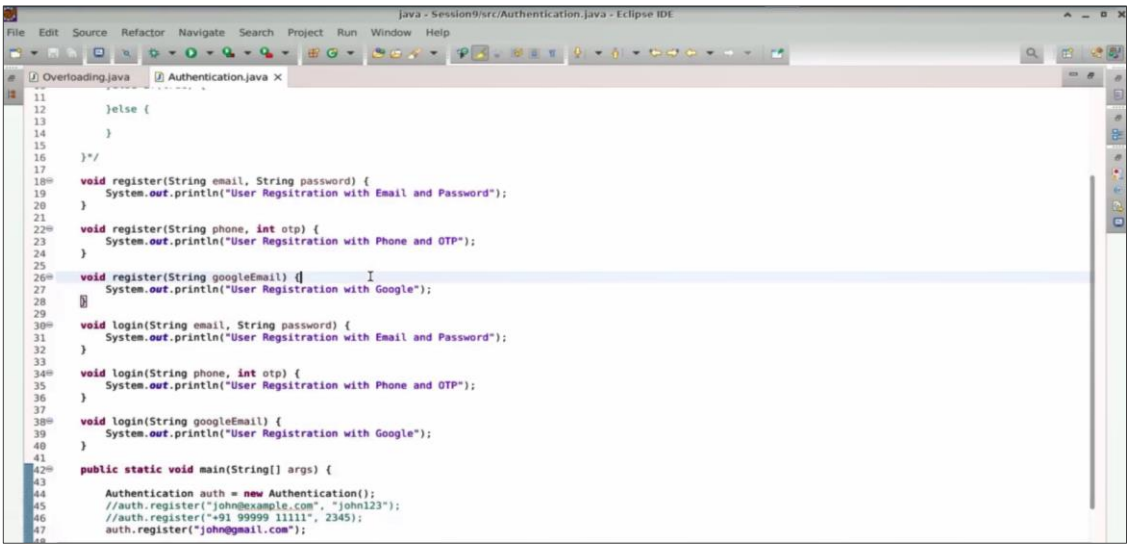


4.9 Like the way you got registration methods, similarly, you can create log-in methods, so you can log in the user with email and password, log in the user with the phone and OTP, and also, you can even log in as a user with a Google account

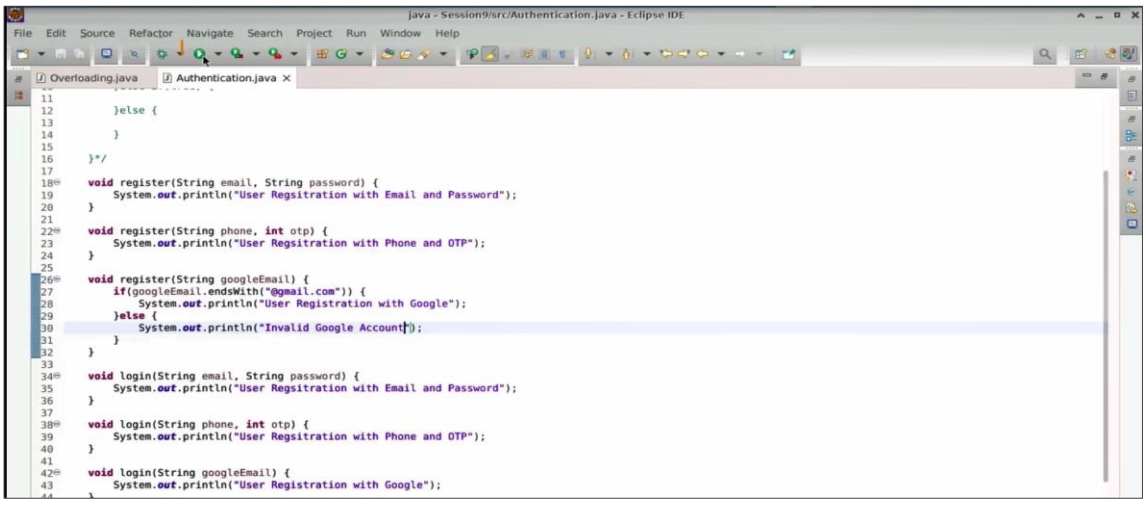


Step 5: Write methods for authentication

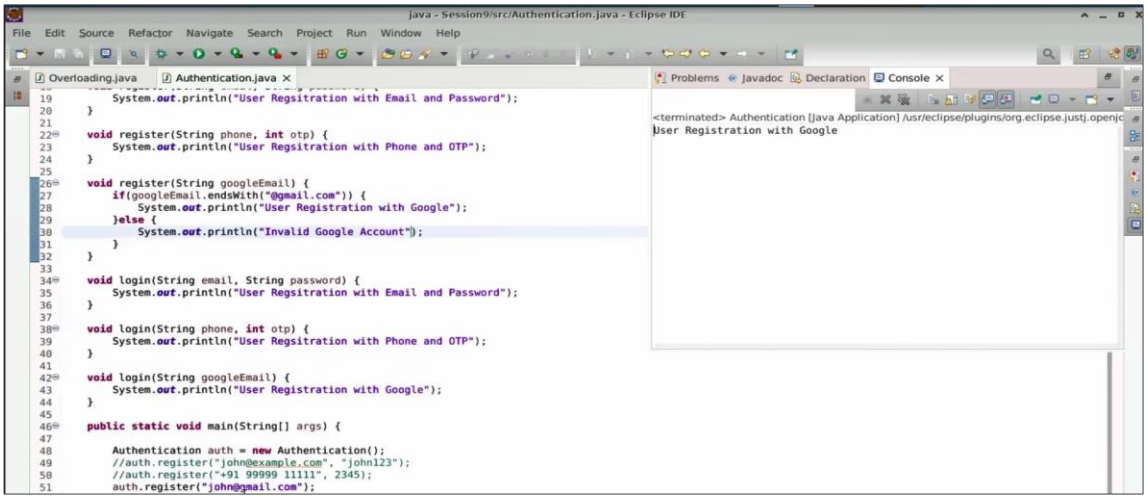
5.1 The benefit is very simple, you have now divided your complex logic, and you have created different methods, and the user can anytime say auth dot register, which is a reference variable to your authentication object



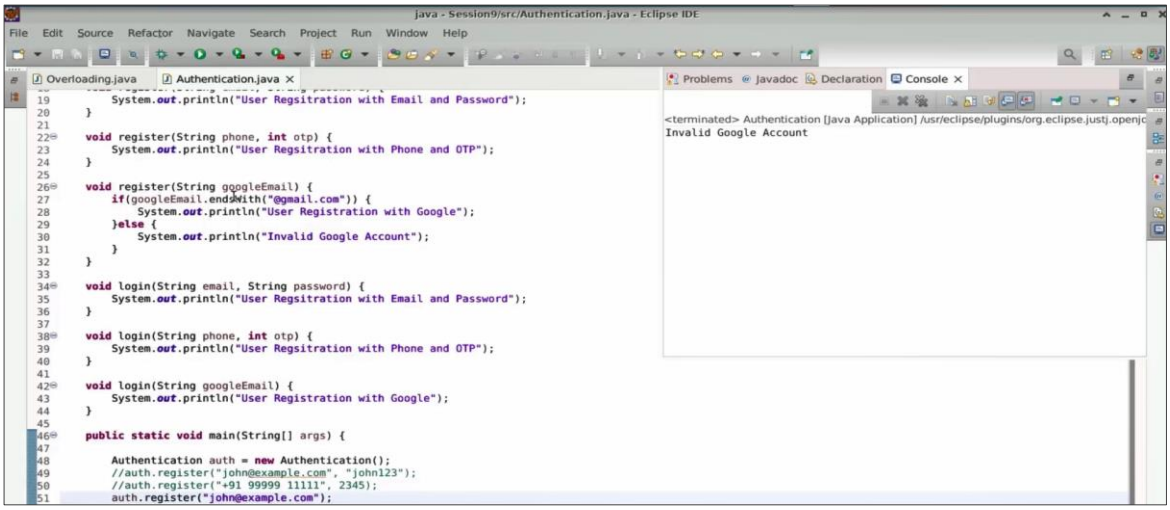
5.2 The logic can be written inside these methods. For example, if your googleEmail.endsWith (“@gmail.com”), only then you will register with Google, and in the else case, you can do some other processing



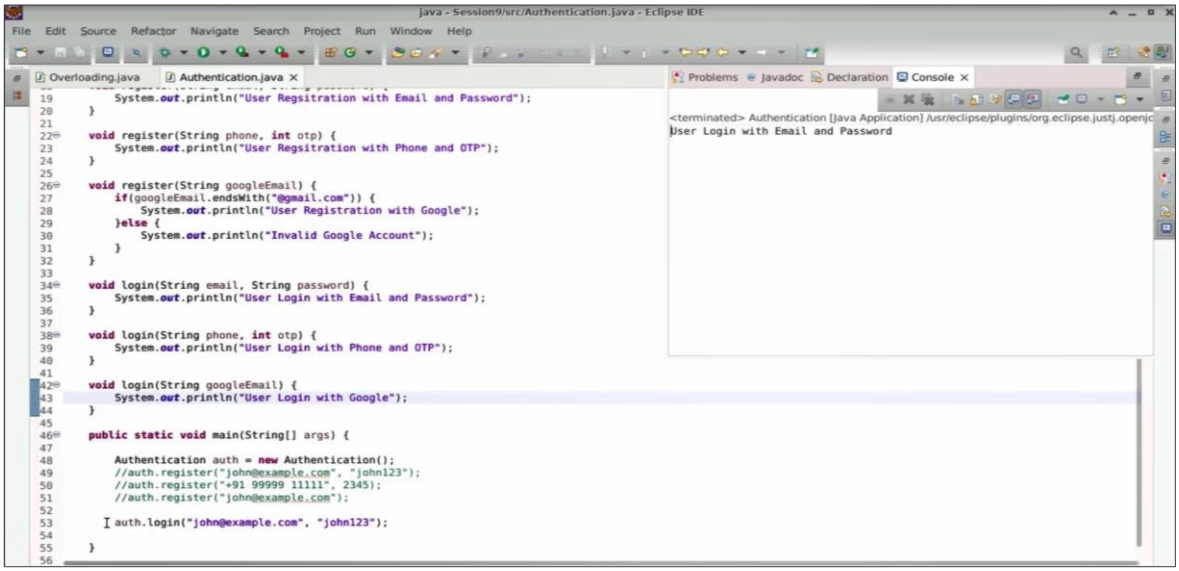
5.3 Run the code, it states user registration with Google



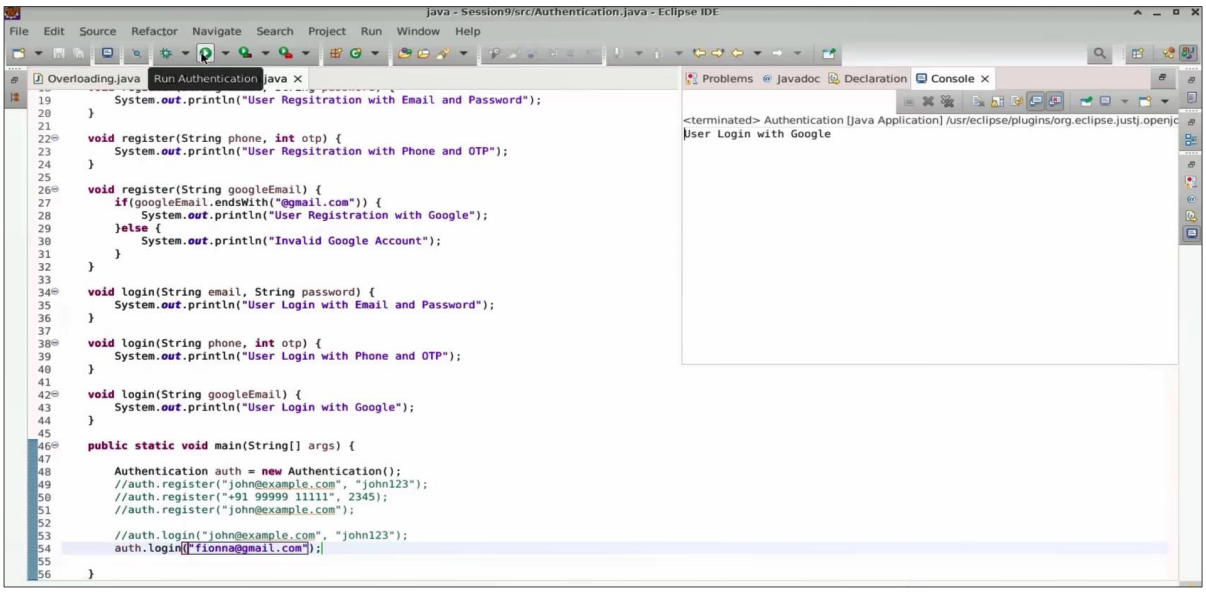
5.4 But if you use **John@example.com**, it states Invalid Google Account because you are still using the same register method, which takes the Google email as input



5.5 Now, with these different ways to do registration, you can also use the method called log-in, with the help of which you will execute your login method. Change all this as user log-in. Run this code. It states User login with email and password.



5.6 If you use **auth.login** and mention **fionna@gmail.com** when you log in, it will state that the user logs in with Google



By following the above steps, you have successfully depicted how to overload methods in Java.