Lesson 03 Demo 01

Designing the User and Product Objects in OOPs

Objective: Implementation of designing the user and product object in object-oriented programming

Tools required: Eclipse IDE

Prerequisites: None

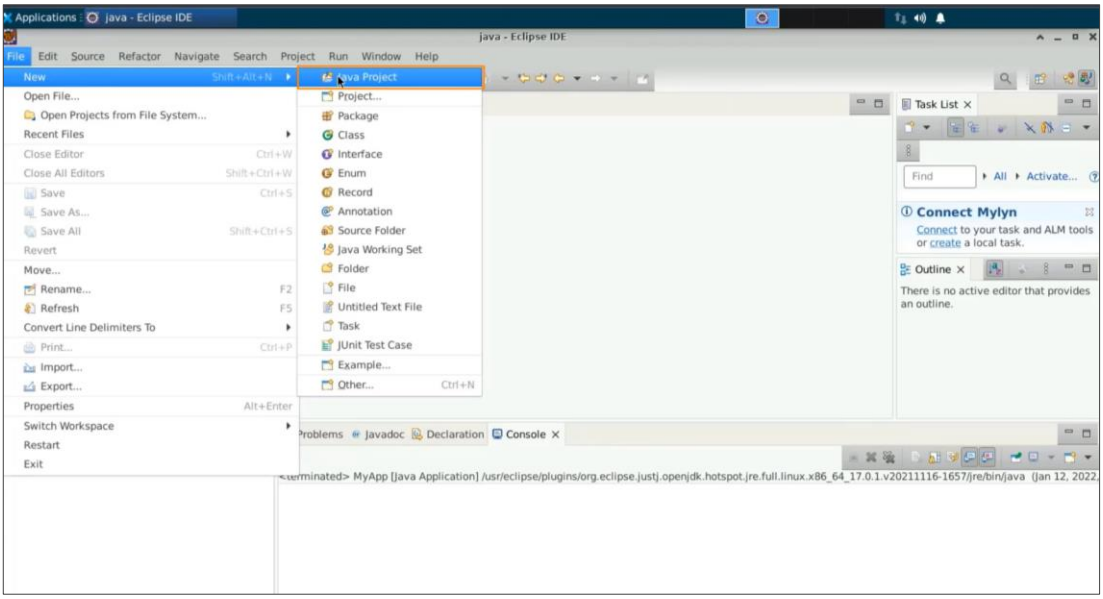
Steps to be followed:

1. Open the IDE and Use a one-dimensional array with suitable examples
2. Create a new class called OOPS
3. Use the sample e-store application to add objects
4. Identify the data in the attributes
5. Create a real object in the memory
6. Assign hash codes to the objects
7. Implement a read operation on the user object
8. Perform operations on the object
9. Access the attribute and print the details
10. Run the code and populate the data
11. Select the default constructor from the source, then generate the constructor with fields
12. Create an array with a specific size
13. Implement a two-dimensional array with suitable examples

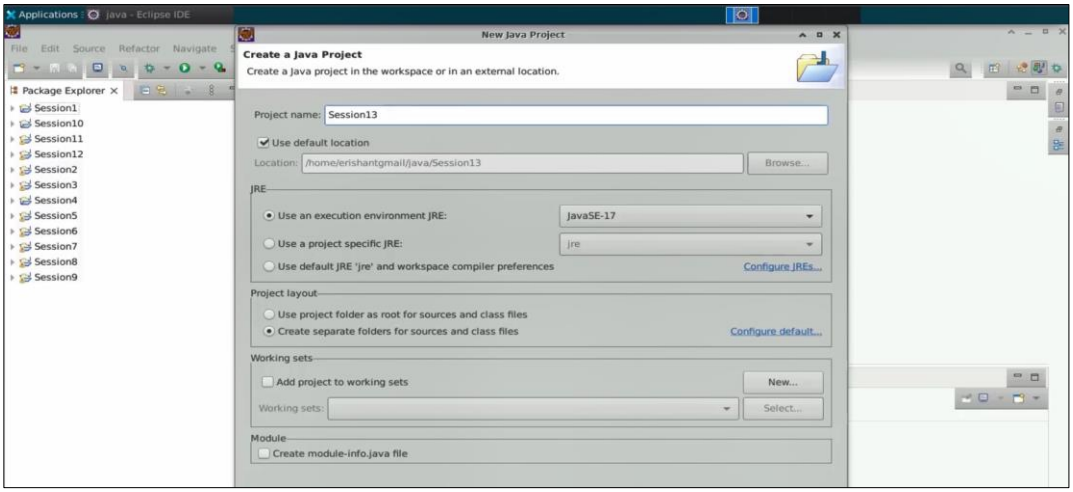
Step 1: Open the IDE and use a one-dimensional array with suitable examples

1.1 Open the **Eclipse IDE**

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

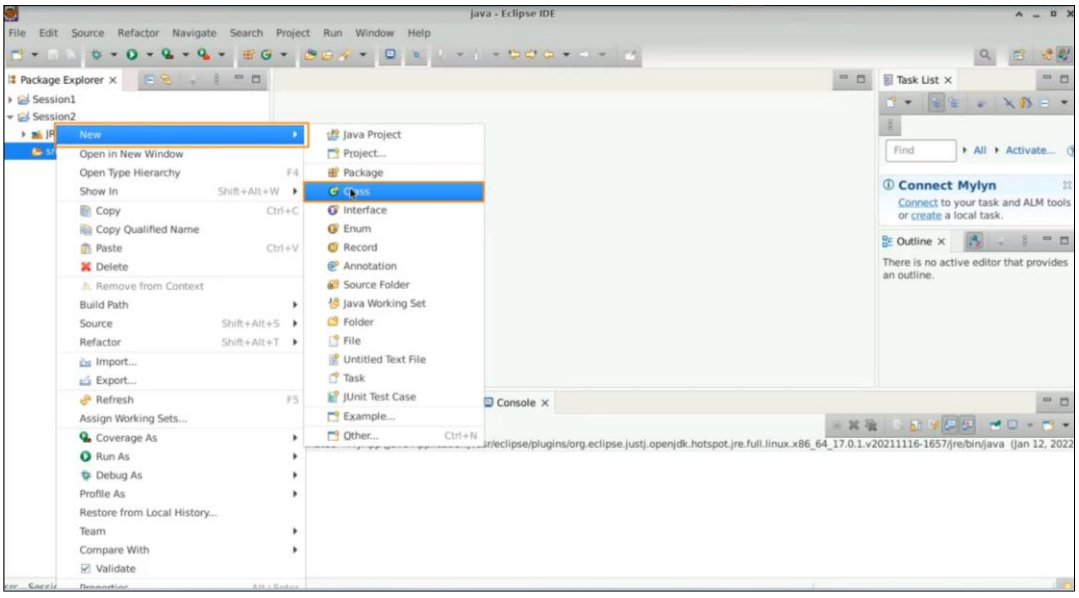


1.3 Name the project **“Session14”,** uncheck **“Create a module info dot Java file”**, and press **Finish**

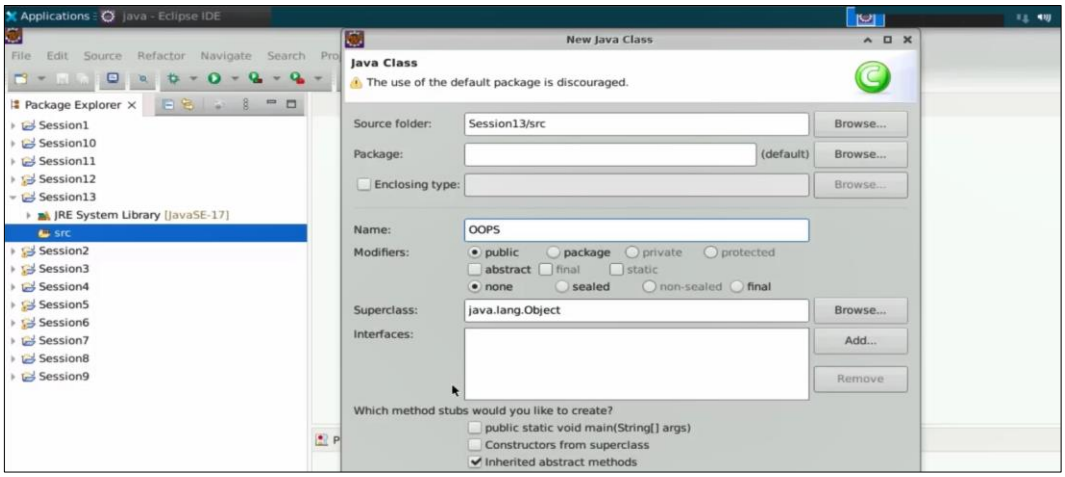


Step 2: Create a new class called OOPS

2.1 With a **Session13** on the src, do a right-click and create a **new class**

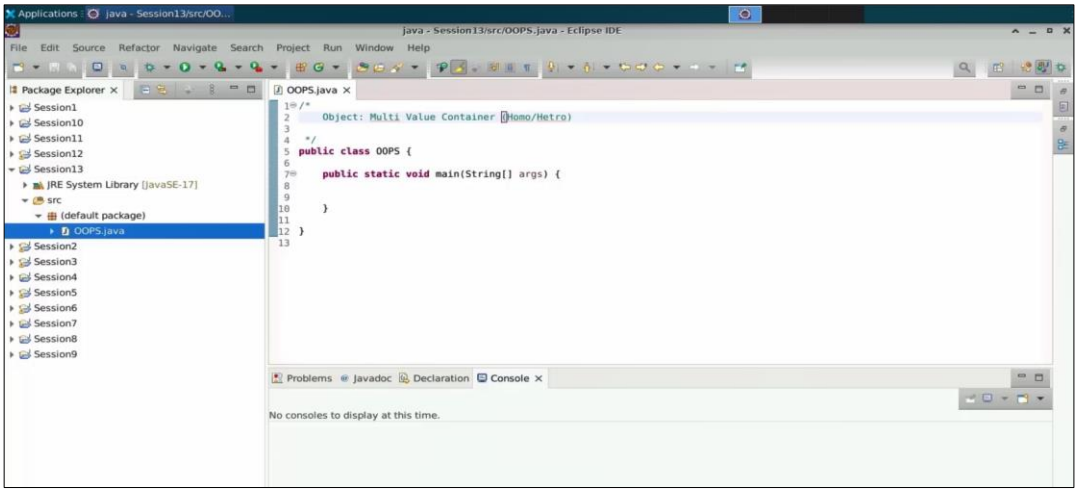


2.2 Name this class as an **OOPS**, then select the **main method,** and then select **finish**

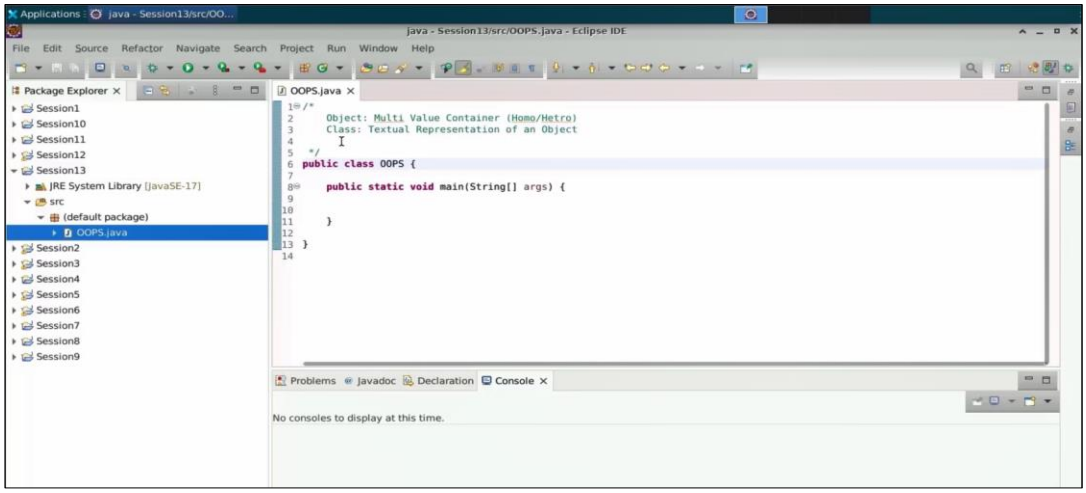


Step 3: Use the sample e-store application to add objects

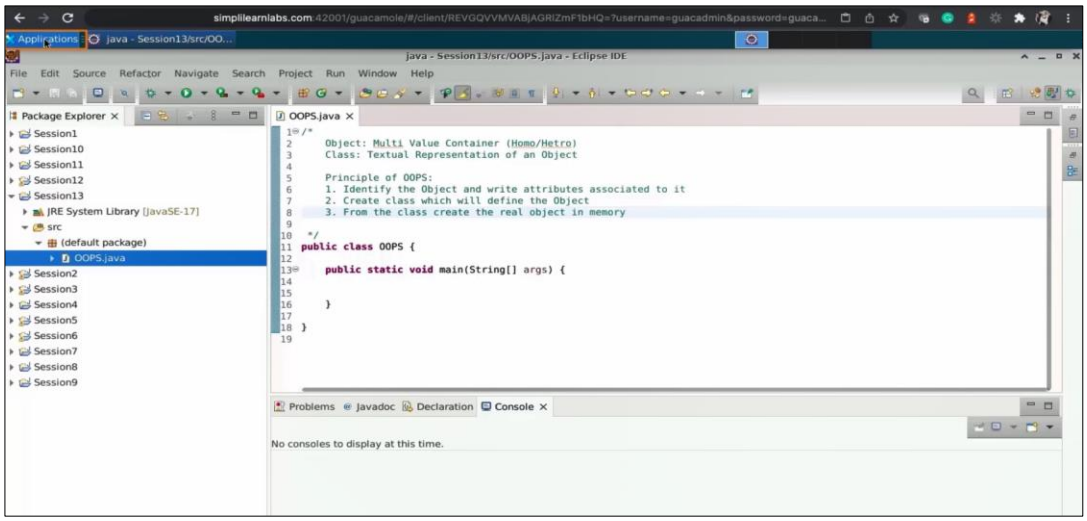
3.1 In order to understand object-oriented programming structure, there are two important key concepts linked to it. The fundamental thing here is an object, which is a multi-value container which can hold a lot of data. It can be homogeneous or it can be heterogenous, which means it can store the data of the same type or of different type. Hence, the object is a user defined data type



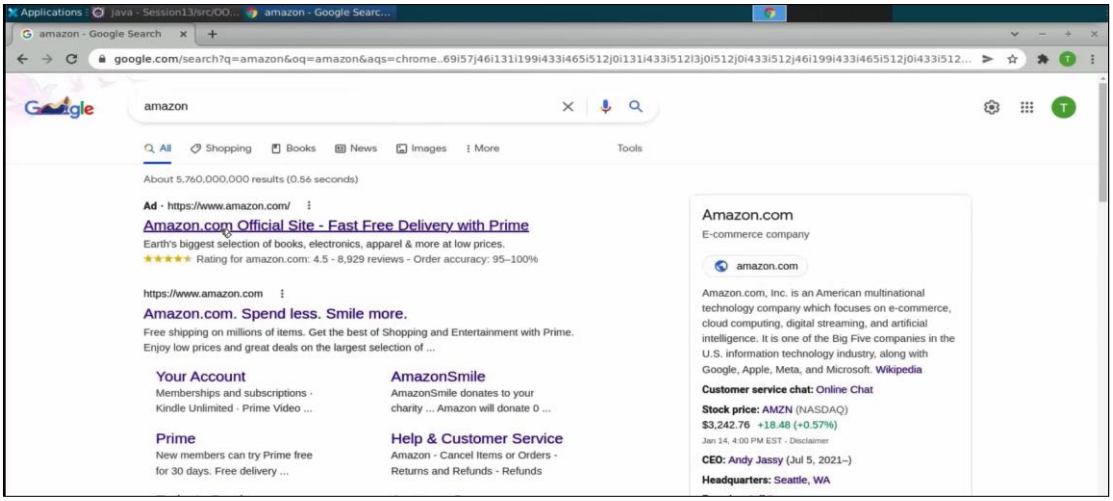
3.2 As a developer, you can write a class that basically is a textual representation of an object. It can also be defined as a blueprint that defines the structure of an object



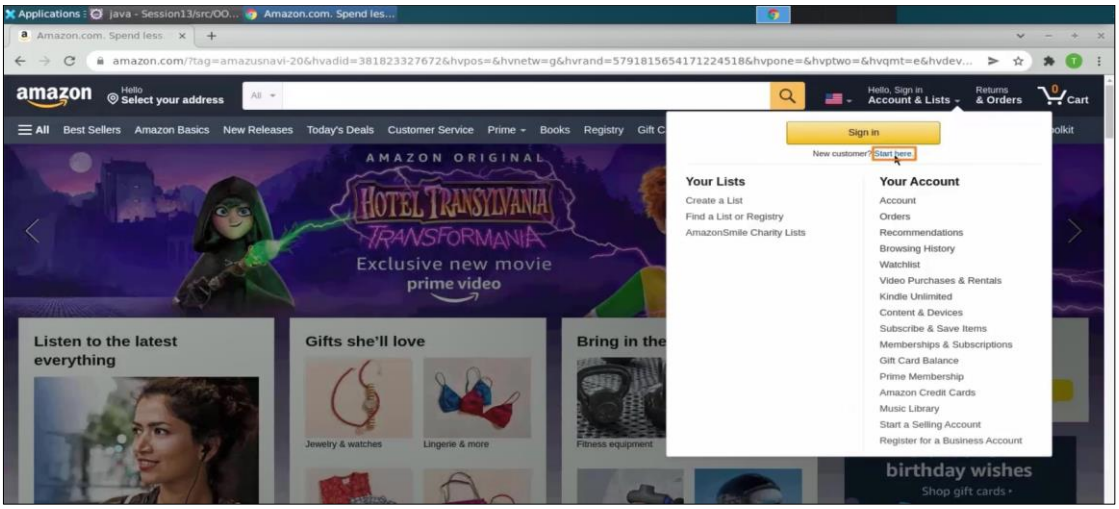
3.3 For the object, let us understand the principle of OOPS. First thing is to identify the object and write the attributes associated with it. Number two, create a class that will define the object, and number three from the class create the real object in the memory



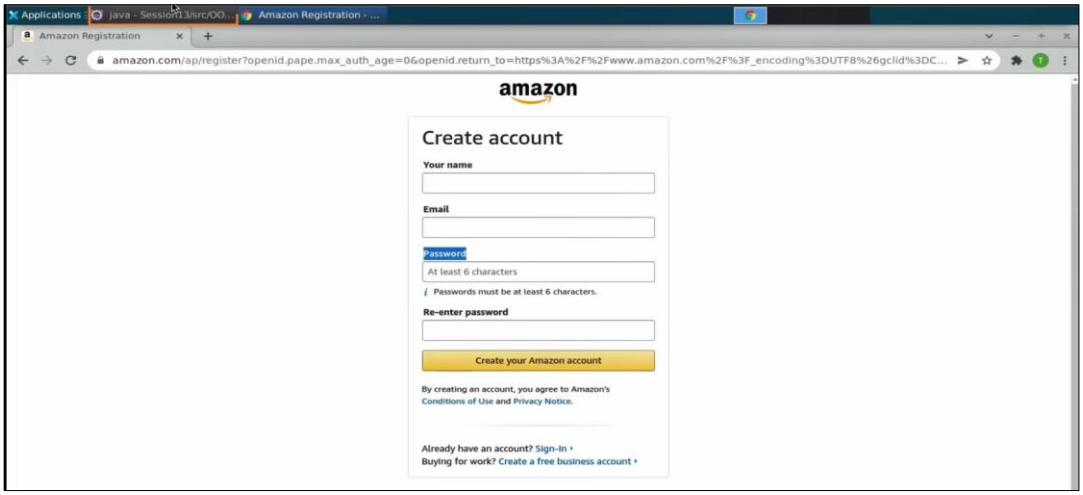
3.4 Let us open the sample e-store application and open the browser window and search for Amazon and open Amazon.com



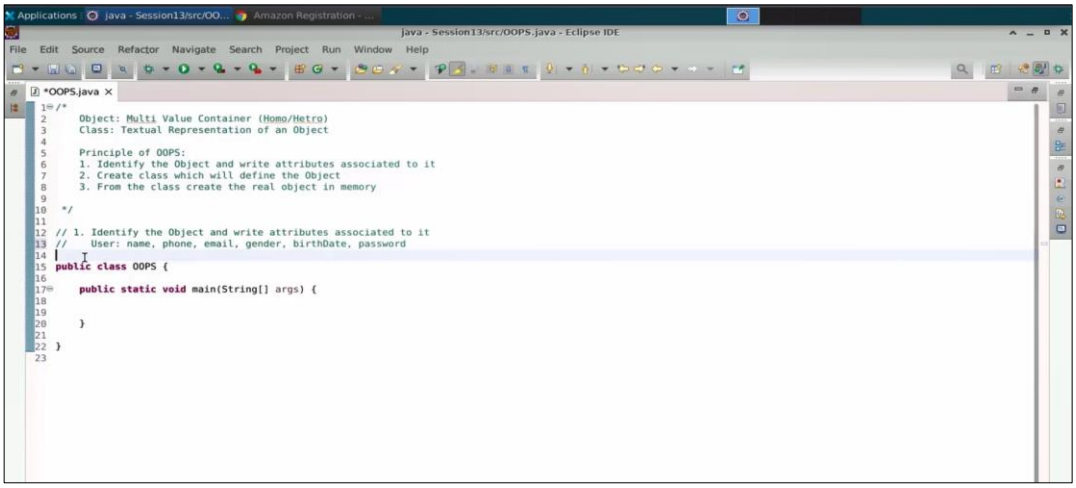
3.5 Whenever you open up any e-commerce platform, there is an option called to sign in or an option for new customers, start here



3.6 When you click on start here, what you observe is that in order to create a user account, the user has to have a name, email, and password associated with the account. Hence, the user is an object, where the user would have a name, email, and password and that is the identification for your object

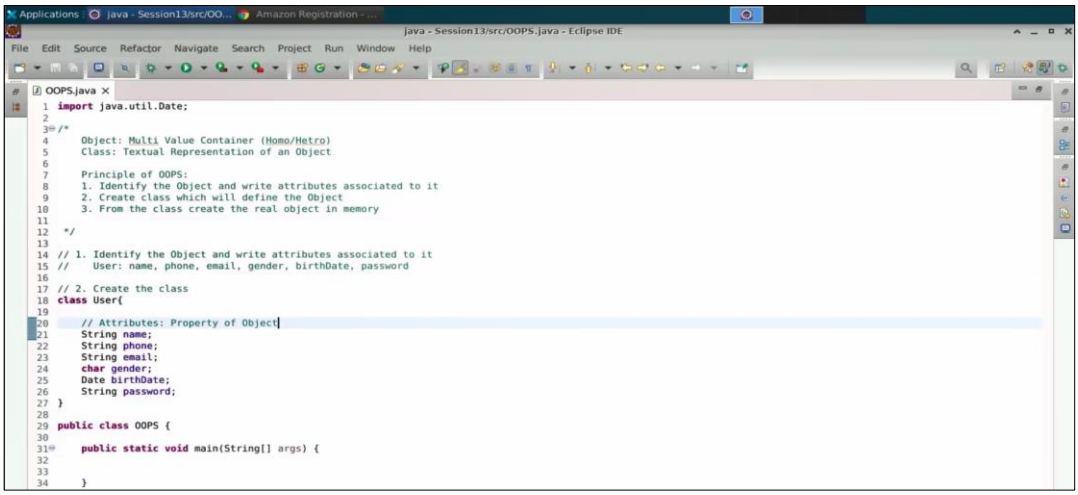


3.7 Let us use the same use case of Amazon or any other E-commerce platform where you can have the user object. Your first goal is to identify the object. Here the object is identified as a user, hence the user has a name and then adds a few more details like the Phone, email, gender, birth date, and password associated with the user

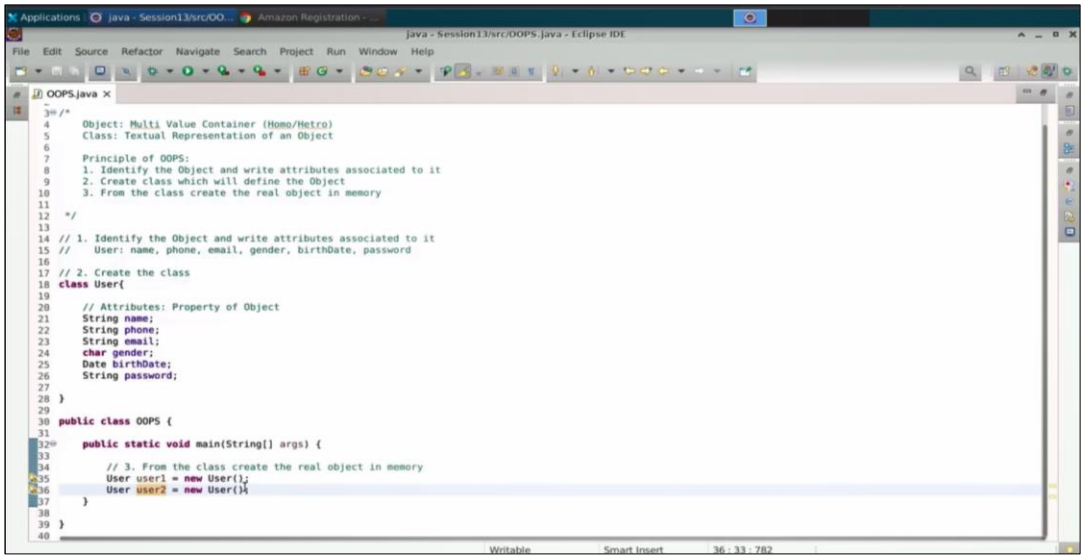


Step 4: Identify the data in the attributes

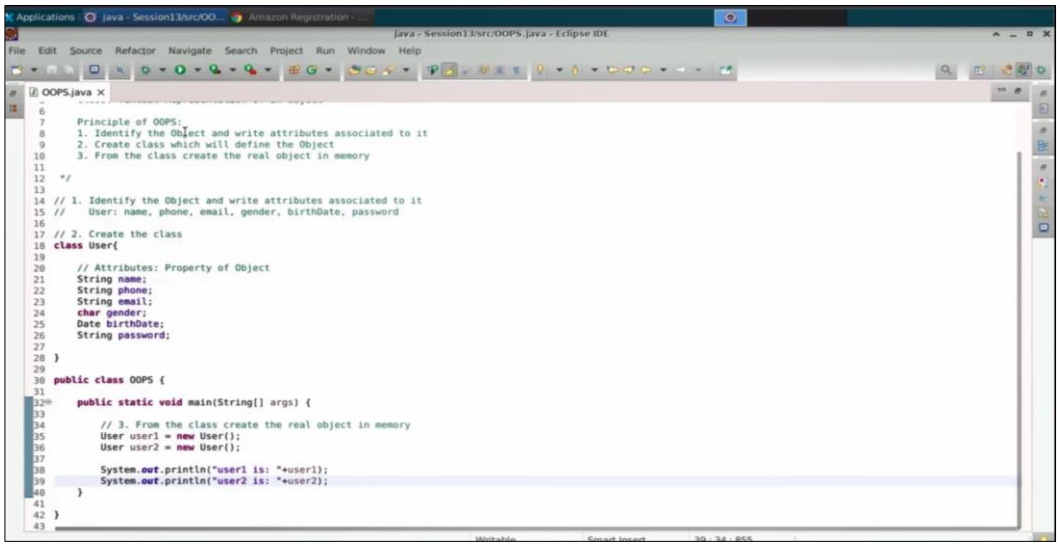
4.1 The second step is to create the class. Let us write class user and then start the bracket and close the bracket. Now you need to add the data inside as your attributes. Next, identify the type of these attributes, where the name will be string type, so is your phone number, the email also goes as a string, and for the birthdate, you can use **Java.util.date**. Then for the password, you can take the data type as a string



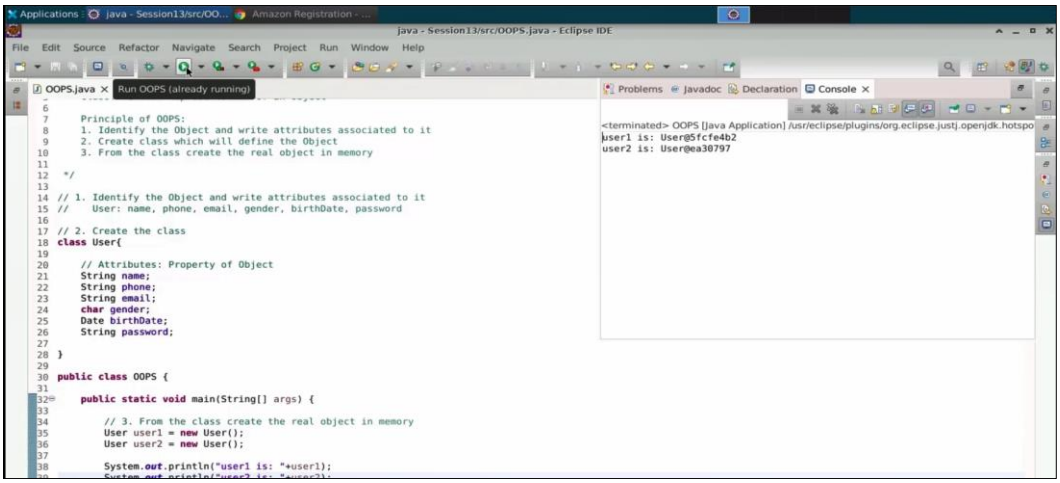
4.2 If you want to associate some values with these attributes. Hence, step number three, where you will create the real object in the memory. Let us use the class name, and then create one of the reference variables. Assign a new operator, then again the class name, and add a parenthesis and a semi-colon. Next, let us write **user 2** as a new user and whenever you will create another object, then **user 2** is the second reference variable.



4.3 Let us print out both these references. Write **System.out.println("User 1 is " + user1);**. In the same way, print out **user2** by writing **System.out.println("User 2 is " + user2);**

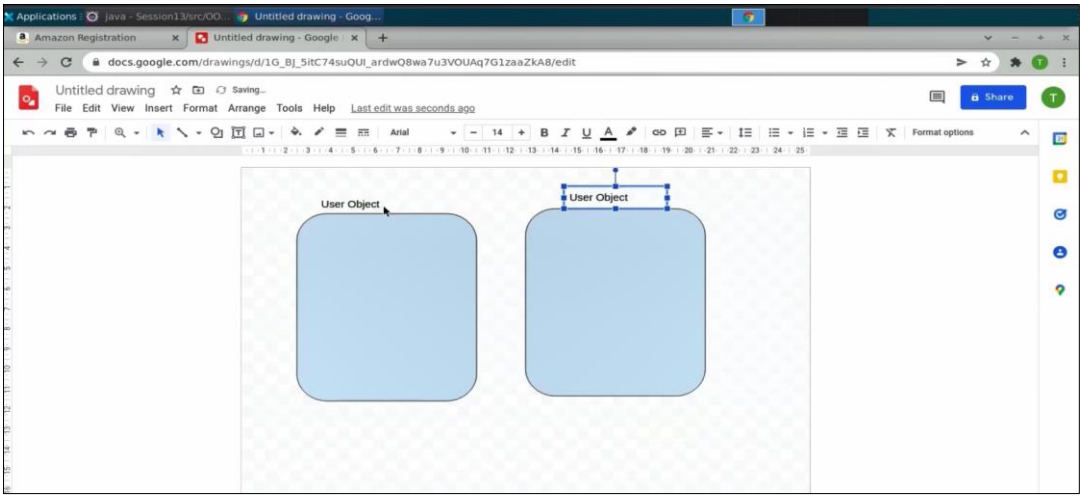


4.4 Run this program and you will observe that user one is one of the user objects at some hash codes. And user 2 is referring to the other user at a different hash code. Hence, there are two of these objects available in the memory

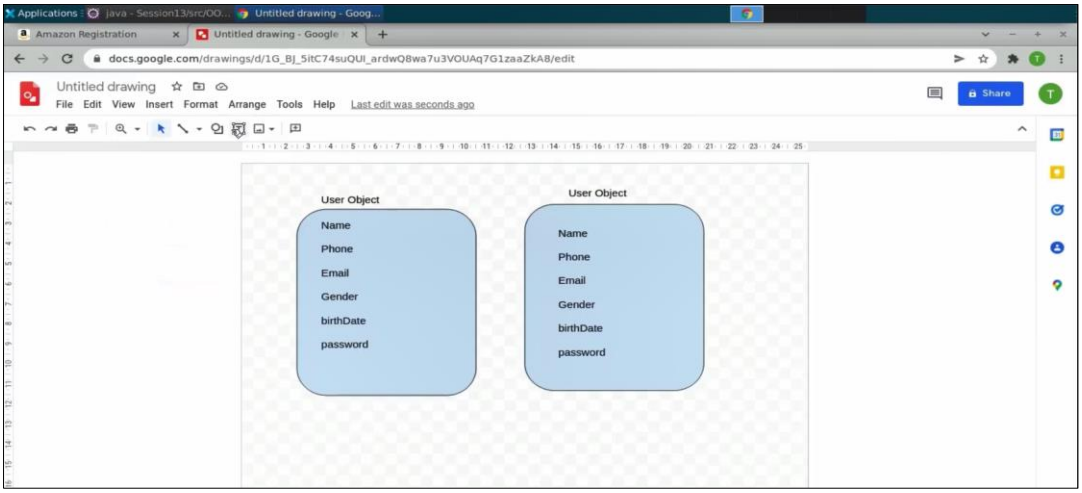


Step 5: Create a real object in the memory

5.1 Open Google Draw and draw these two objects. Both objects are named user objects

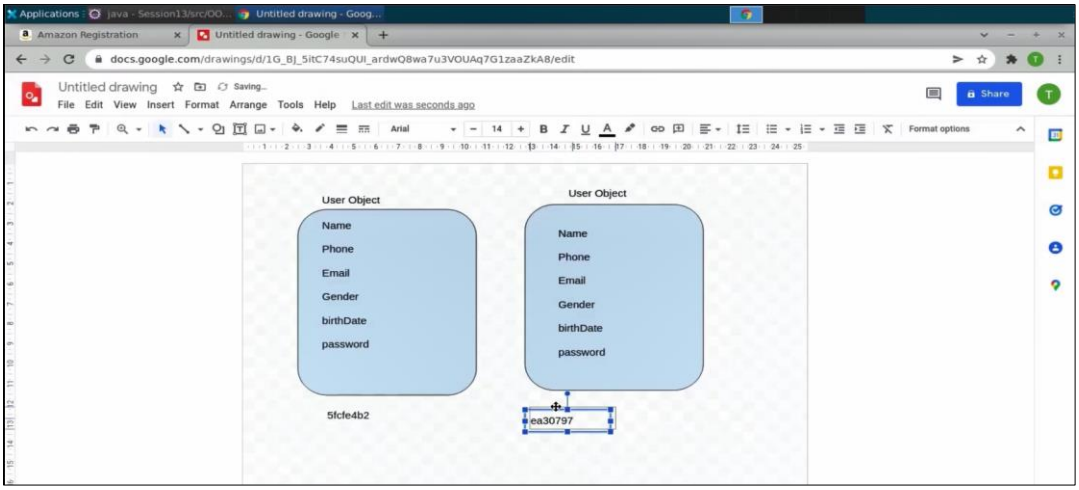


5.2 Inside the object, you will have the attributes which are the name, the phone, email, gender, birth date, and then the password

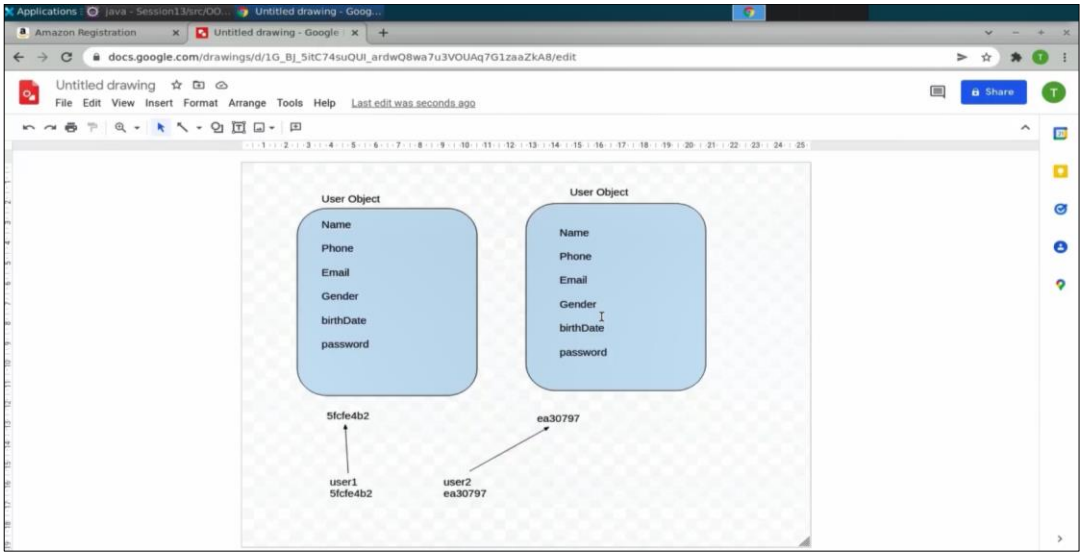


Step 6: Assign hash codes to the objects

6.1 As you have created these two objects, copy this first hash code associated with the first object and paste it here and the same way for the second hash code for the second object

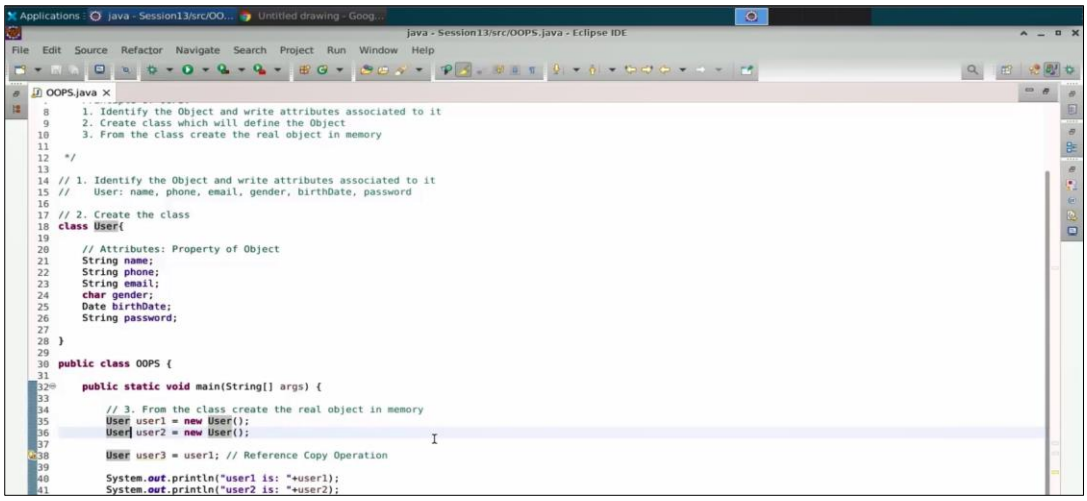


6.2 User one and user two are the reference variables, and these users as reference variable has these hash codes. Thus, the reference variables point out as these, which means, these reference variables refer to these two objects

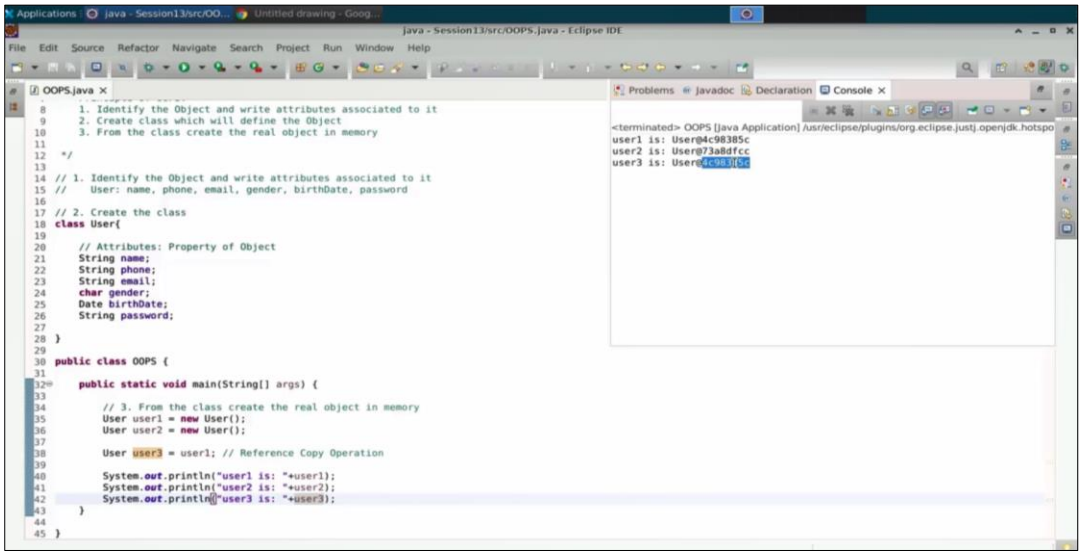


Step 7: Implement a read operation on the user object

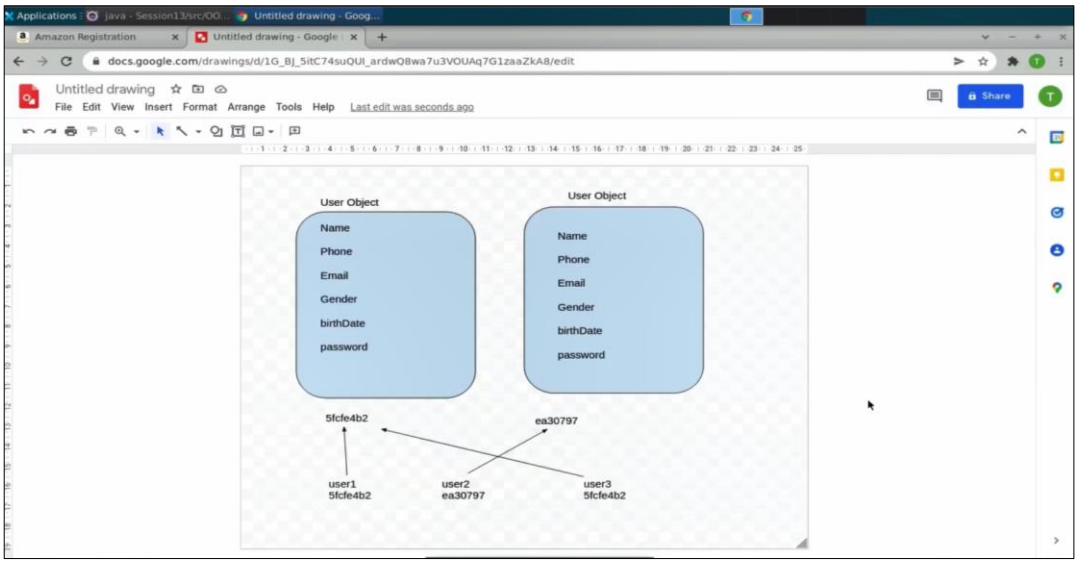
7.1 In the next approach, if you want to give a user, user three is using one. What you did here is a copy operation and this is known as a reference copy operation. It means, there are only two objects, **User 3** is not an object. **User 1**, **user 2**, and **user 3** are the reference variables



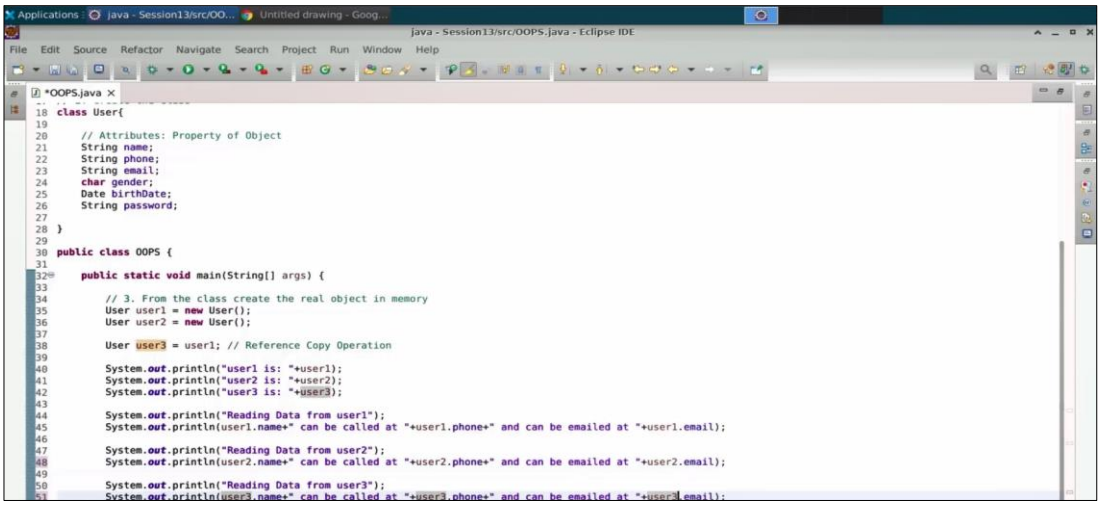
7.2 Now if you print **user 3**, you will notice that user one and user three have the same hash codes. Every time you run the program, hash codes may be different, so do not get confused on the part that why the hash codes are coming differently now



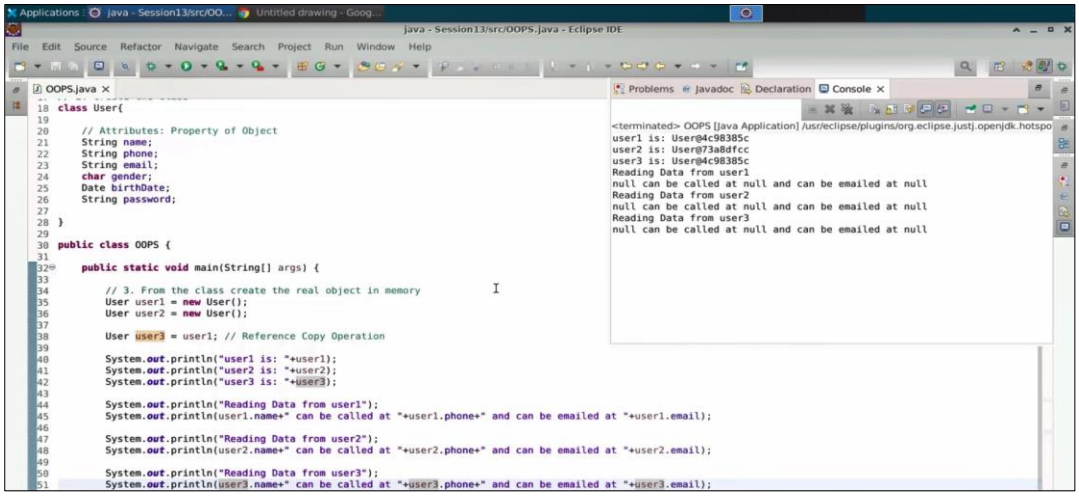
7.3 **User 1** has the same hash code as **user 3**, hence **user 1** and **user 3** are kind of different references, but pointing to the same object



7.4 You have the object constructed, but the objects do not have data. Hence, let us first try to perform a read operation on the user object. Write print("**Reading data from user1**"), and you can read that with the help of your reference variable. Add a dot operator and access your attributes. For example, **user1.name** can be concatenated with **user1.phone** and **user1.email**. Do the same with the other objects

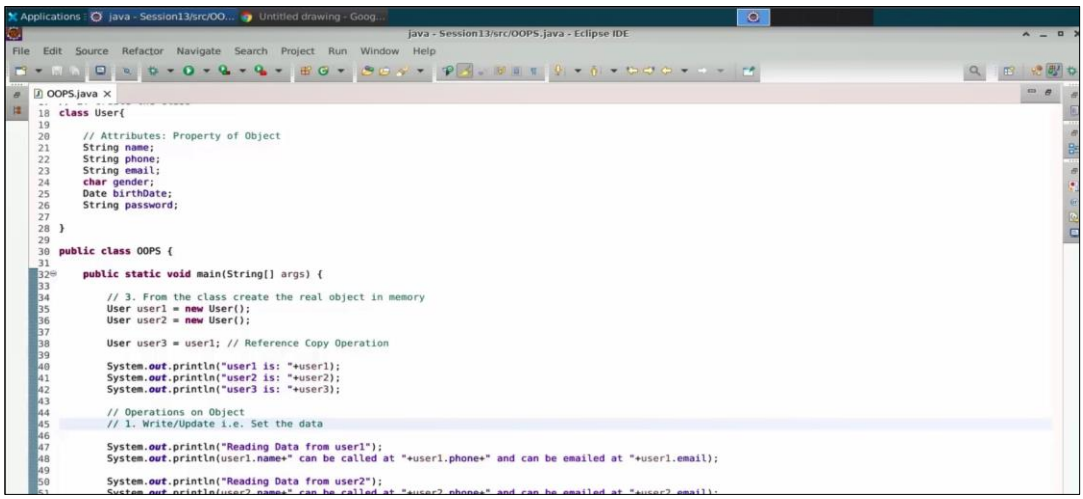


7.5 whenever you execute this program, the data inside the object is coming as null. Hence, It is all empty, which means null can be called at null and can be emailed at null

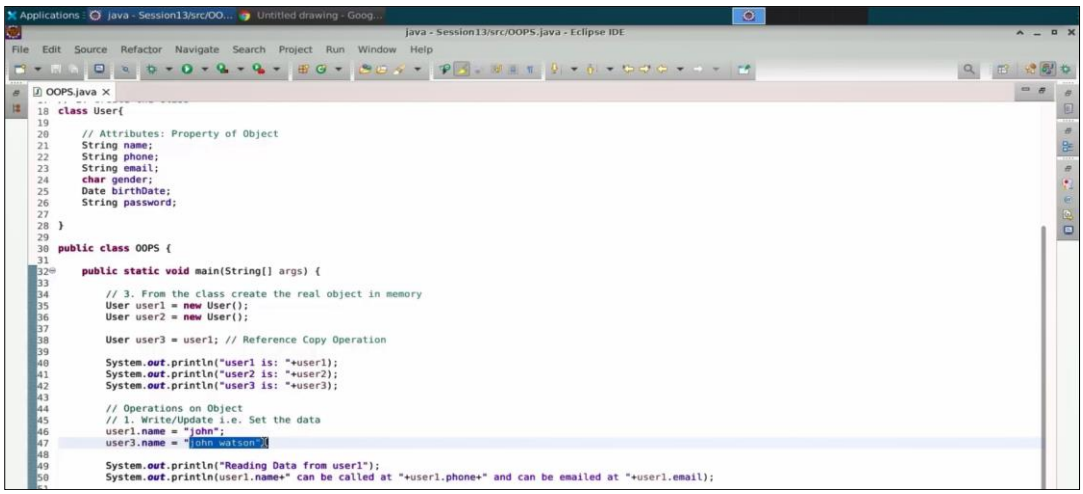


Step 8: Perform operations on the object

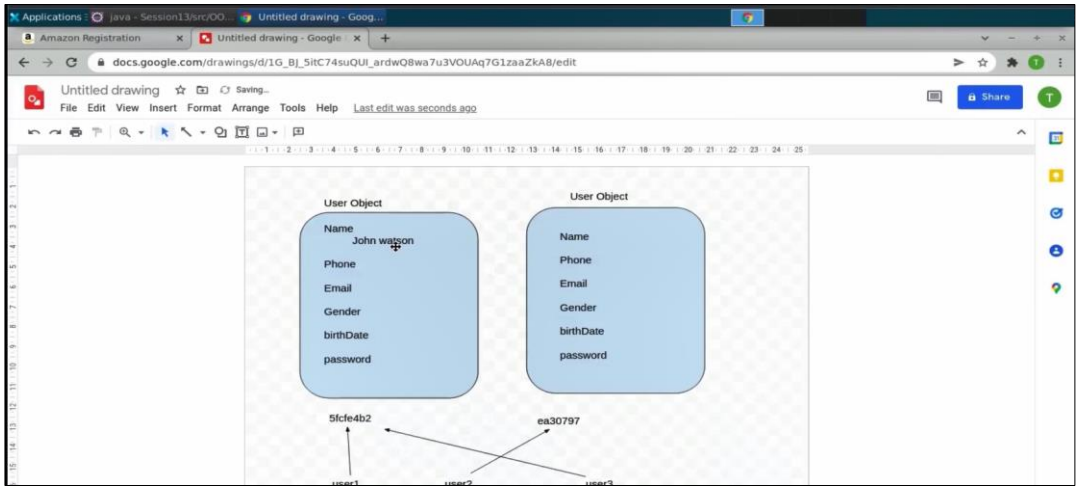
8.1 Moving ahead, you need to perform operations on the object, which means writing or updating the data. The set operation is a combination of write and update, which means it will override the data once you have updated the value



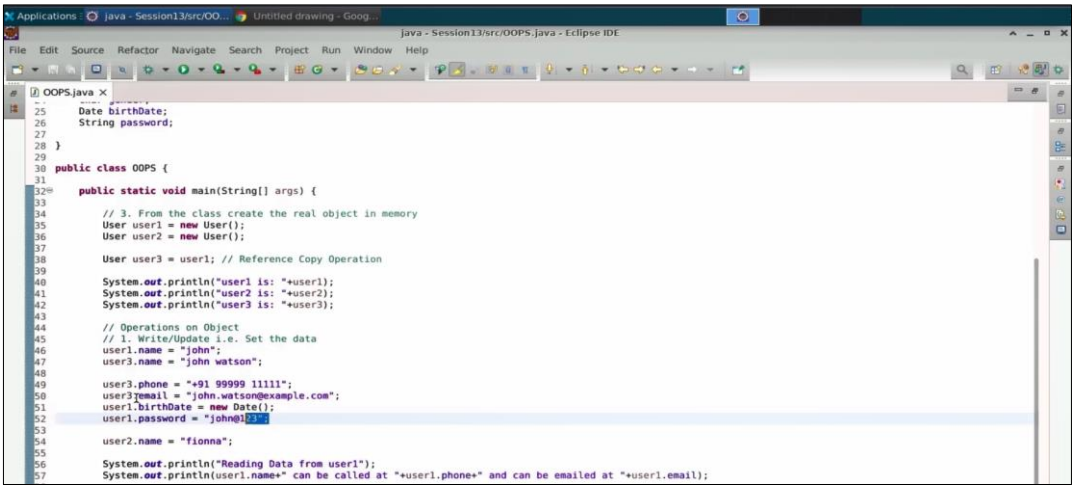
8.2 Let us write **user1.name**, which can be "**John**", and this can be either written or updated. In the next line, you can write **user3.name** and update the value to "**John Watson**"



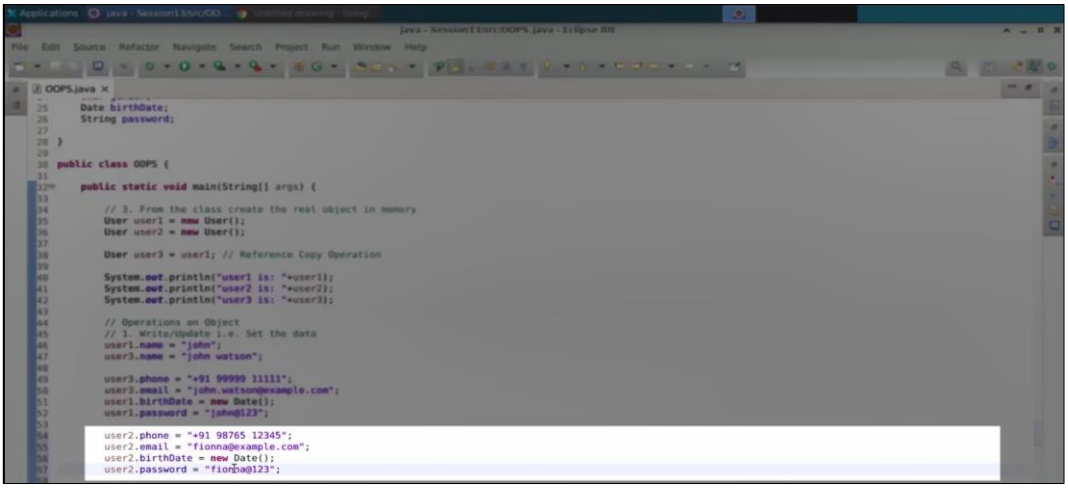
8.3 Hence, when it comes to the object here, the name is **John Watson**, and this is the record that would be added to the attribute name. Similarly, let us write **user2.name** as **Fionna**



8.4 As you have added the name, let us add the other details. Let us write **user3.phone** as +91 and a number. Then **user3.email** as **John.Watson@example.com** and **user1.birthdate** as a new Date object from **java.util**. Then **user1.password** goes like **john@123**. These are the values that will get populated for your first object

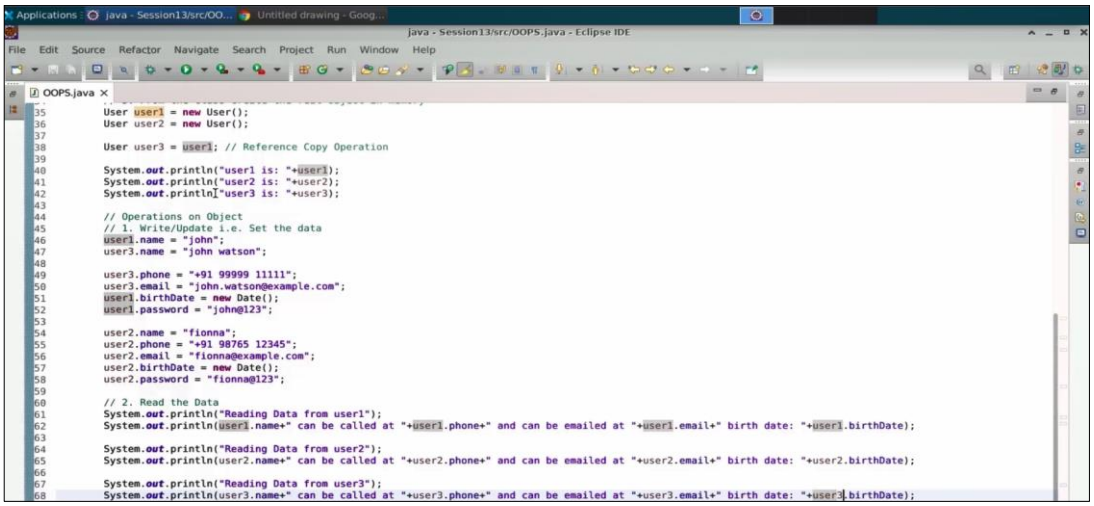


8.5 Let us also feed similar details into **user2**. Let us give the phone number, then write as **fionna@example.com** and the password as **fionna@123**

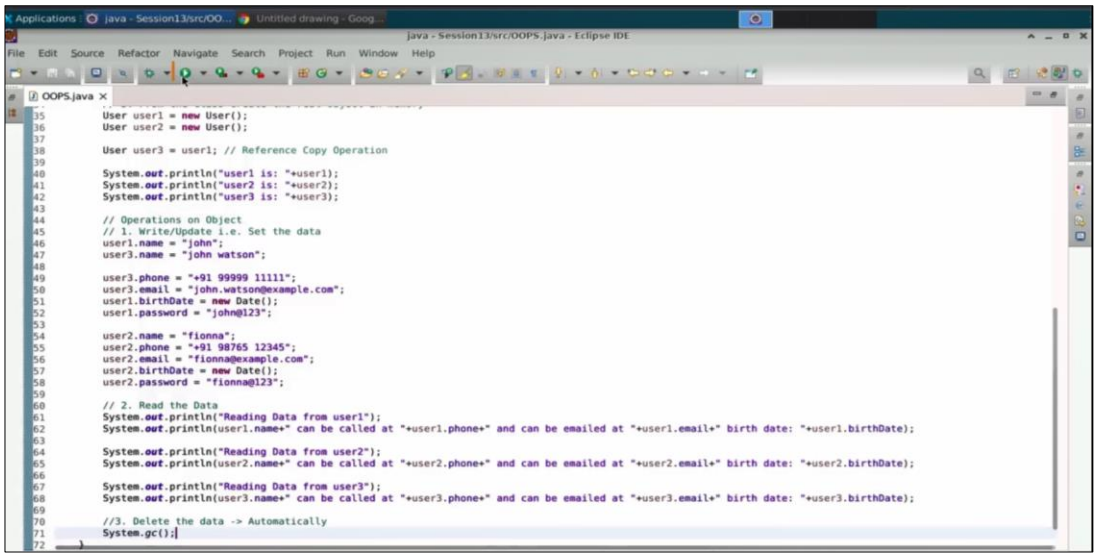


Step 9: Access the attribute and print the details

9.1 In The second operation called **Read the Data**, which means with the object's reference, you are accessing the attribute and it is printing the details

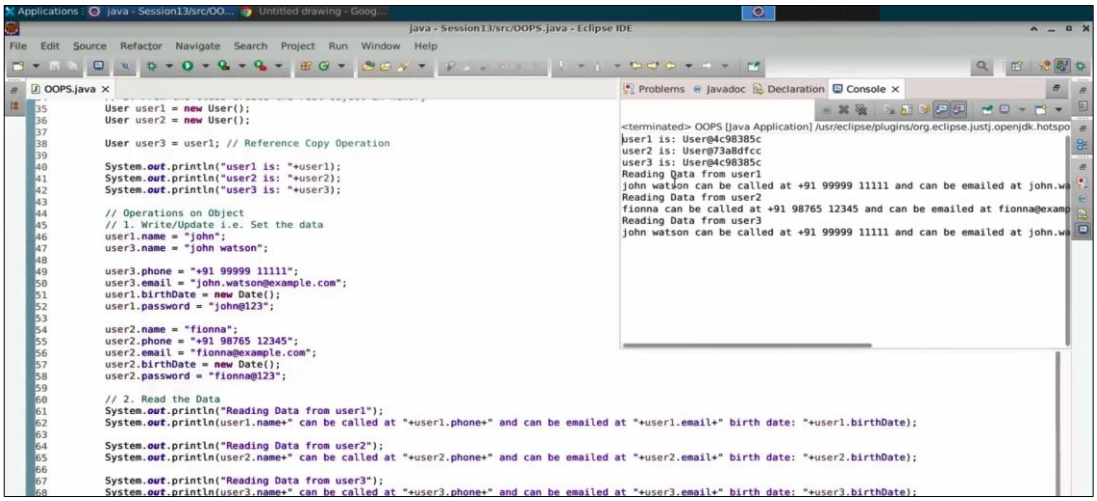


9.2 The third operation is '**delete the data**,' which happens automatically. Java has a garbage collector that runs periodically, looks for unused objects, and deletes them from memory. However, you can also write it as **System.gc()**, which is an explicit call to the garbage collector that can eliminate or remove unused objects from memory with a single command

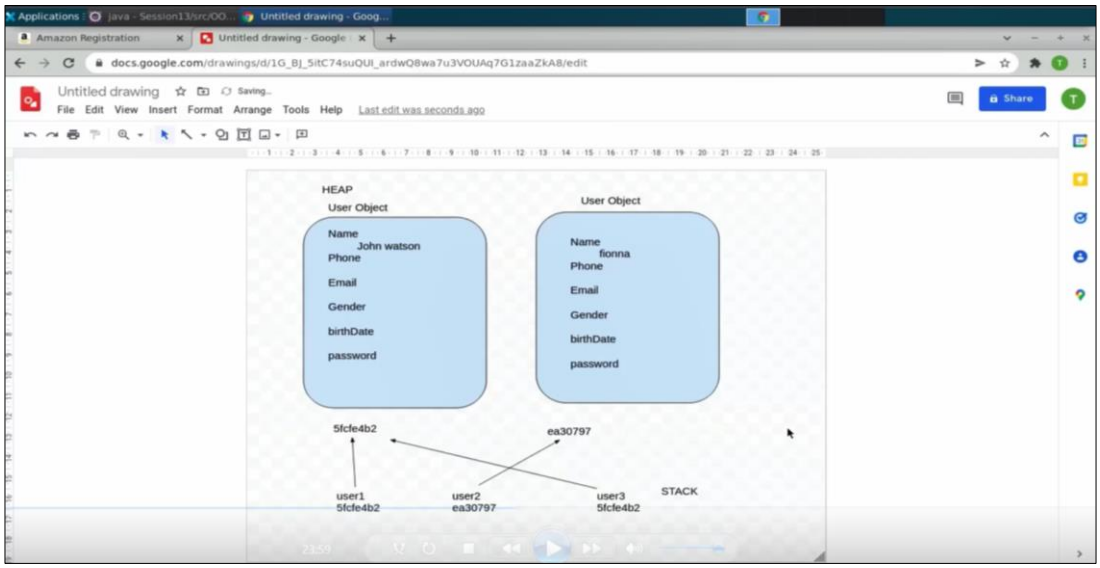


Step 10: Run the code and populate the data

10.1 Let us run this code and what you will observe is while reading the data from user one, the data has been populated over here. And the rest of the details coming in, with the birthday coming as the entire data. One thing that you need to note is for user one and for user three all the details are going to be the same because they are not different objects

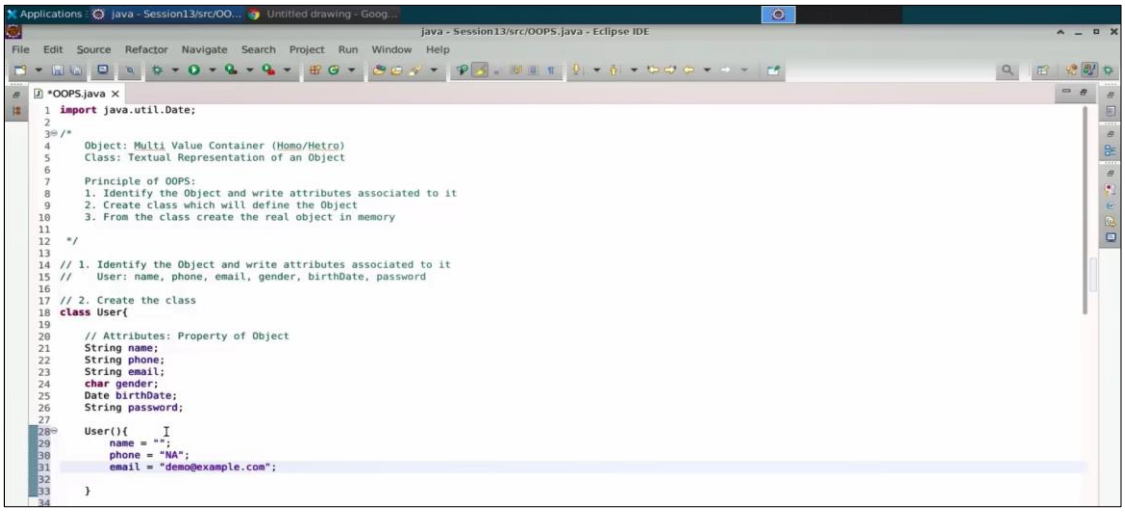


10.2 The objects get constructed in the area known as the heap of your ram area whereas these reference variables would be available in the stack region allocated to the main method

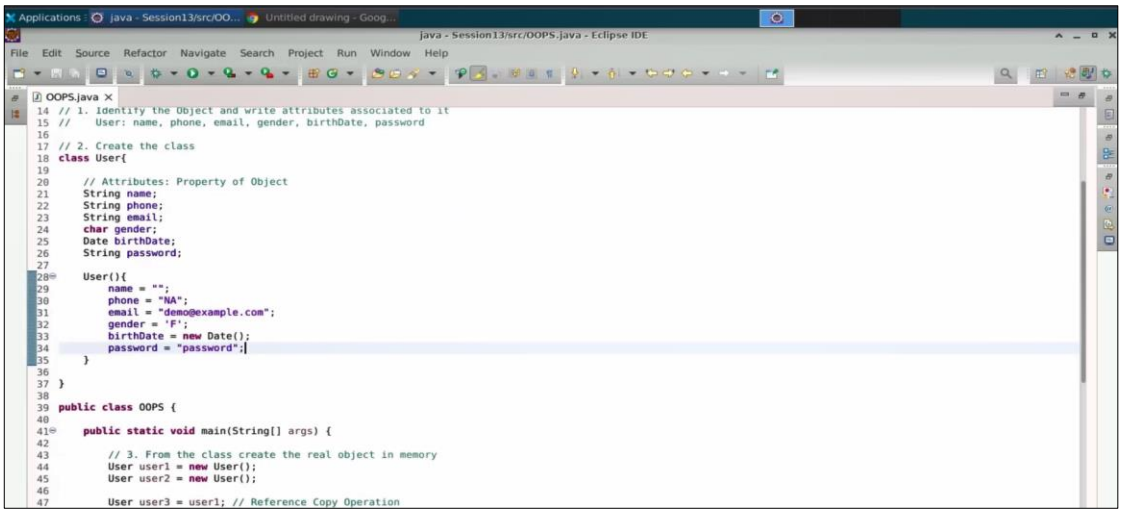


Step 11: Select the default constructor from the source, then generate the constructor with fields

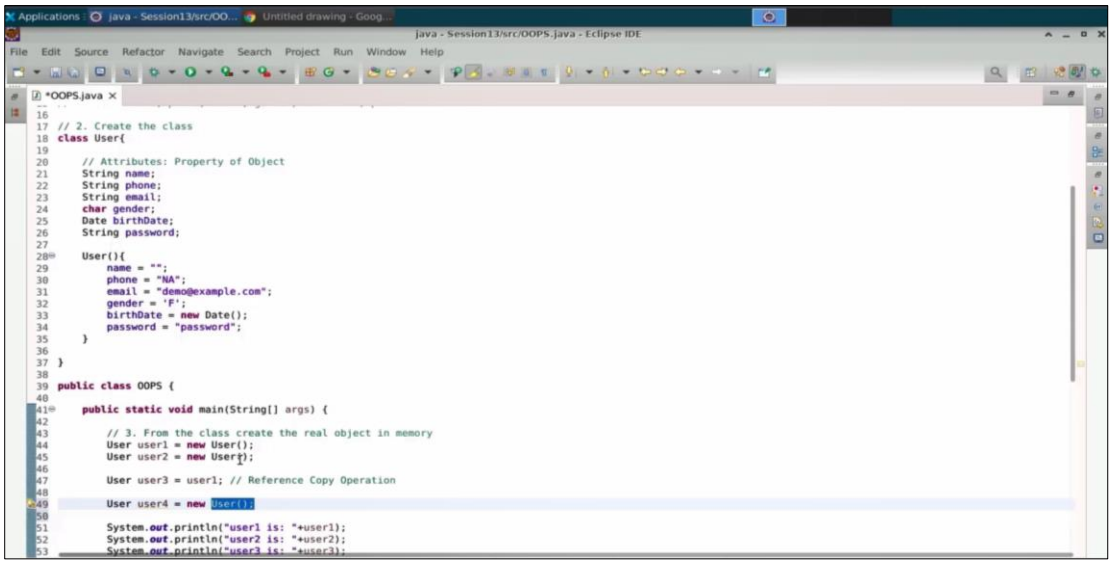
11.1 Now, write a constructor which is a method inside your class, having the same name as that of the class name. You can give as the name is not available, which is the default value of the variable name. Phone is not available or you can even initialize it to empty, so rather than null, you can give it as not available. Then the email goes as **demo@example.com**.



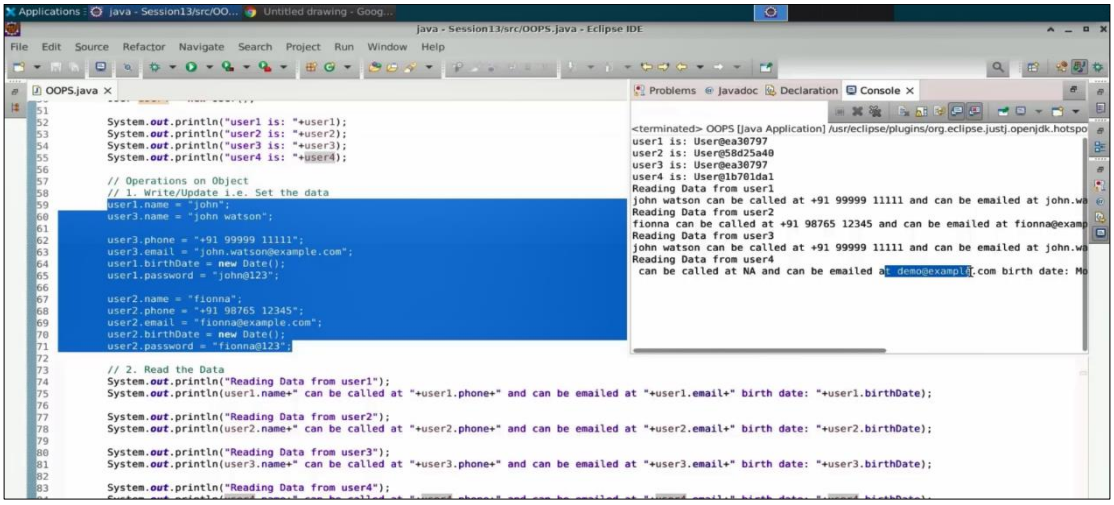
11.2 Let us give the gender by default which is Female. Then the birthdate is a new date object from the java dot util. The password is by default password. These are some of the details, which will be automatically associated



11.3 Whenever you create any user object, remember in order to create an object, you need to write new and then use your class name and add parenthesis. Note, that this is basically execution to the default constructor

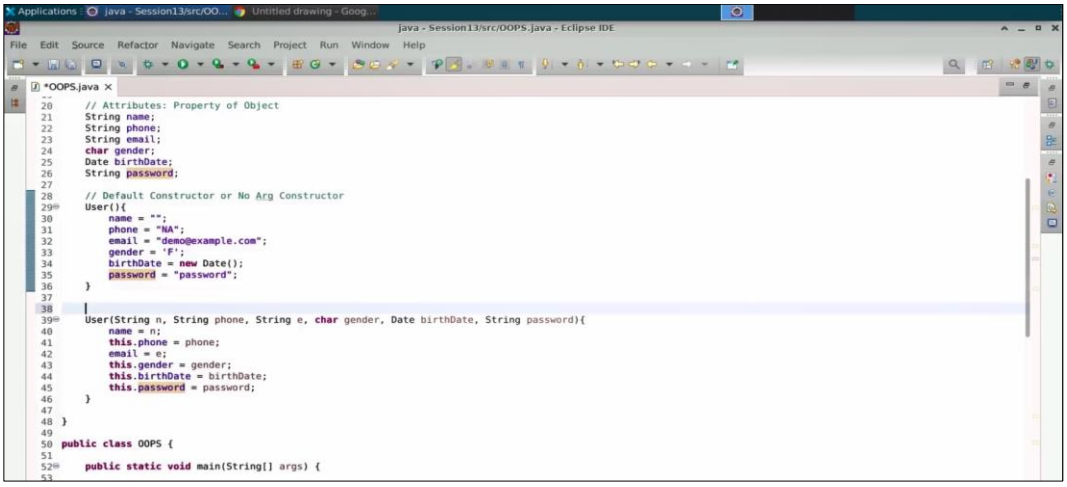


11.4 Let us create a **user4** and print it and the same way let us print the data from the **user4**. You will not be writing any data in **user4** as you have written the data in **user1, 2, and 3** as just a reference copy. Then, when you run your code, it will show empty can be called at NA, and can be emailed at **demo@example.com** and birthdate in the default manner

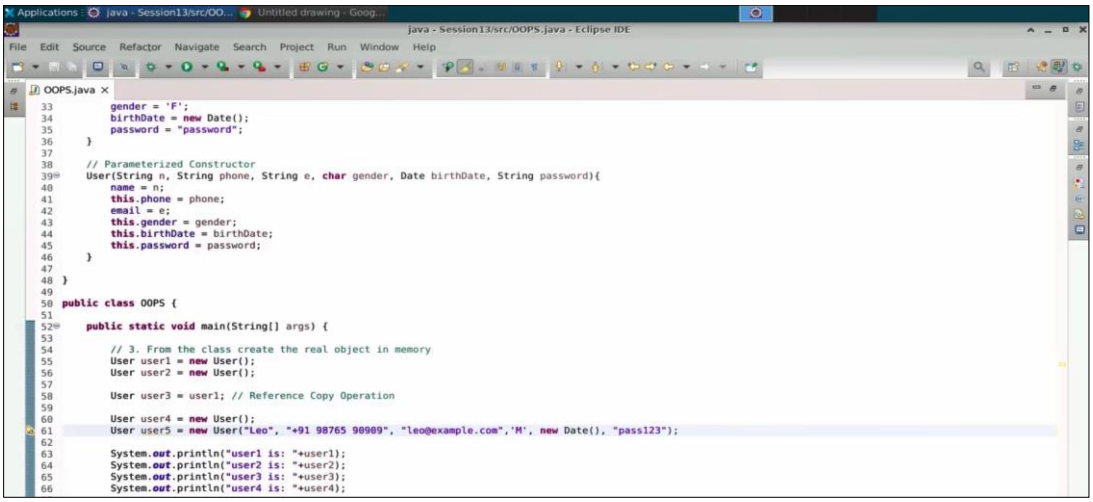


Step 12: Create an array with a specific size

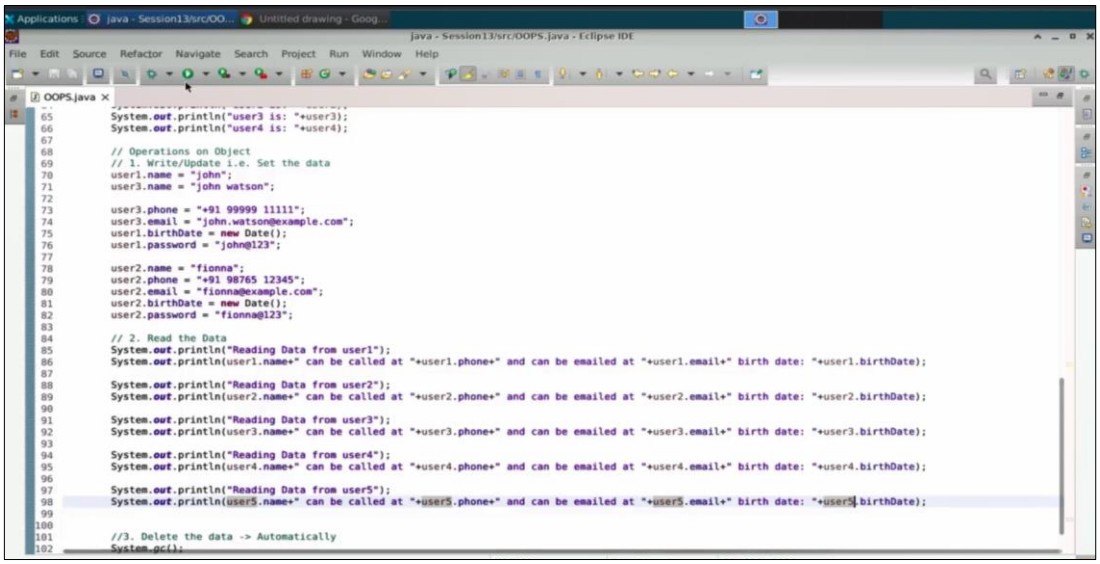
12.1 You can also write a constructor with inputs, and to do that, there are two ways. One way is to write **String n** and then copy this **n** into the name. You can then give the second input as **String phone**. Here, there is confusion: when you assign phone to **phone**, it creates ambiguity. To resolve this, you need to use a variable called **this**, which refers to the input to the constructor. Next, write **char gender**, then **Date birthDate**, and finally a String called **password**



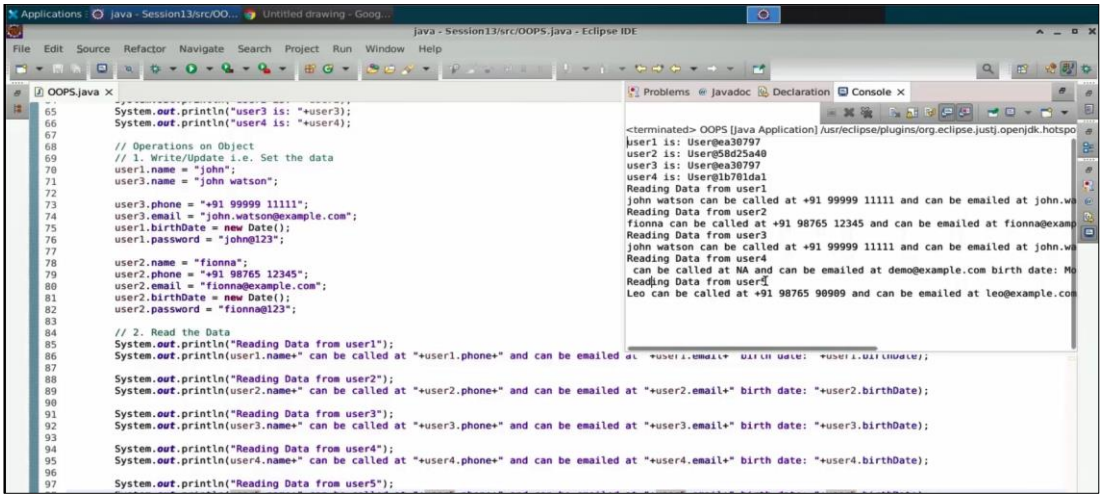
12.2 Next, let us create another user object called **user5**, which is a new user and this time you can pass the data. Let us write the name, **Leo**. A phone number is a phone number and then the gender. Then the birthdate, you can give a **new date** object here and then the **password**. Next, the email, which is like your third input, so let us give it as **Leo@example.com**.



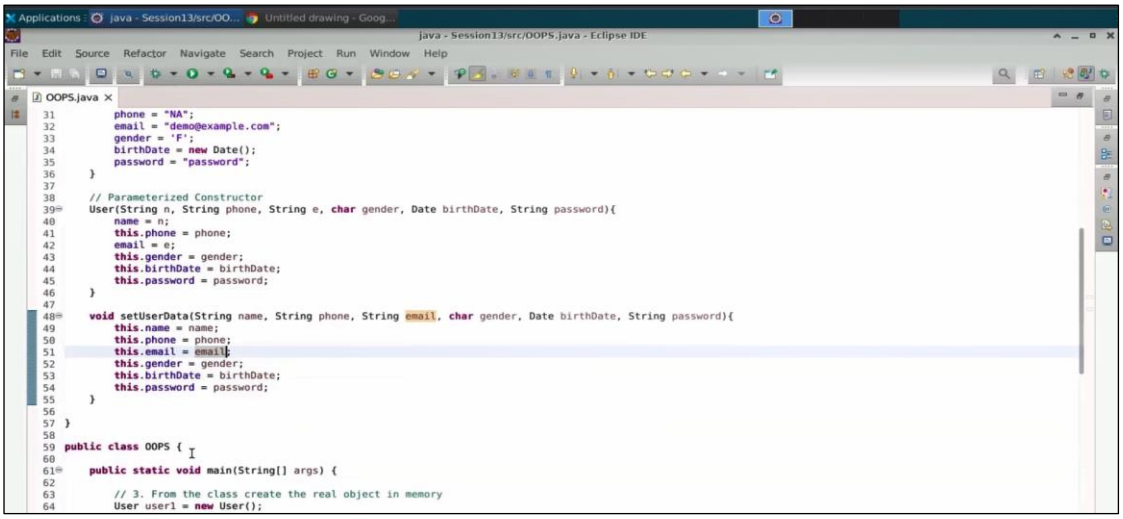
12.3 Let us print the data for **User5**



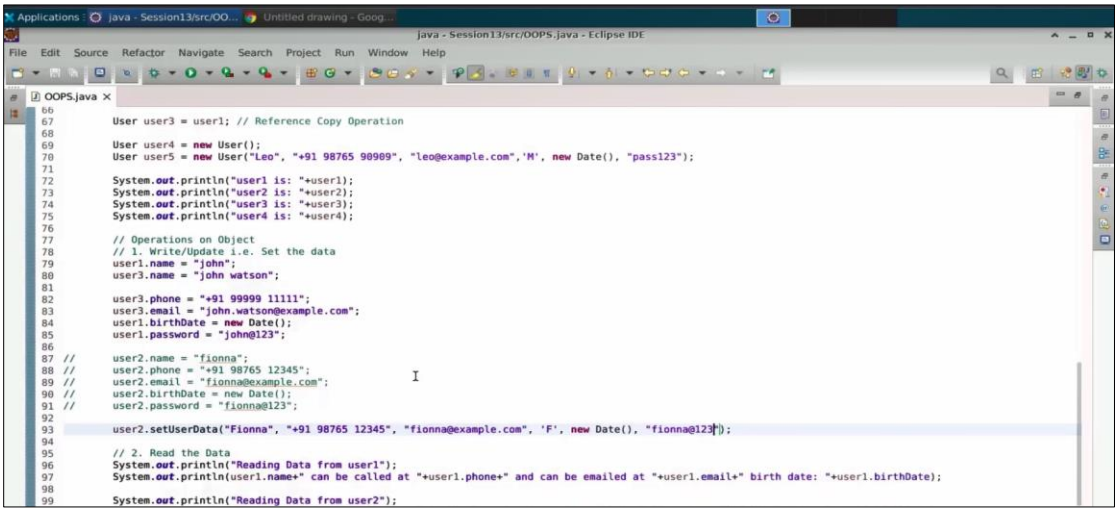
12.4 Run it and Here you are with the data in user five, which shows Leo can be called and email and accordingly the entire data coming in



12.5 Let us write void and create a method called **setUserData.** For the user data, you can take all the inputs that are present in the constructor. To have a better approach, let us write this as **this.name = name** and then **this.email = email**

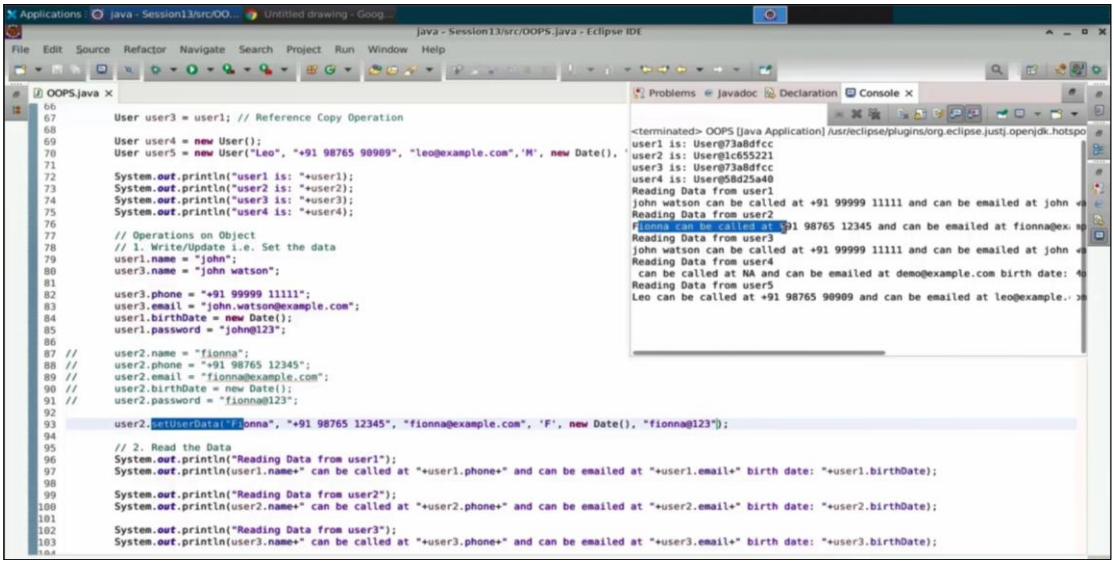


12.6 Instead of writing **user2.setAllDetails(),** you can just write **user2.setUserData()**, where the name is set to **Fionna**, then the **phone number**, and then the **email**. For a **new date** object, you can even manipulate the date objects and work on customized date objects

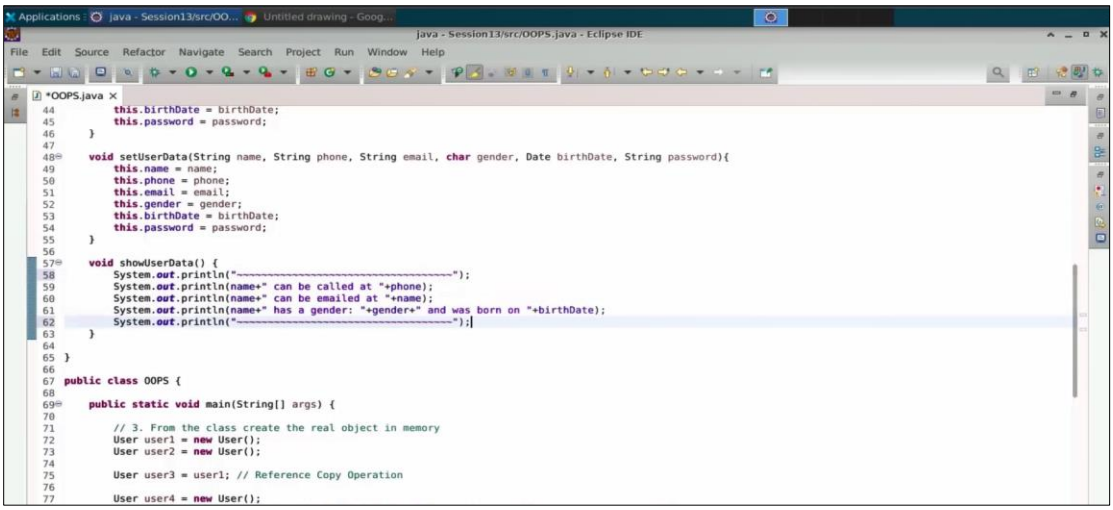


Step 13: Implement a two-dimensional array with suitable examples

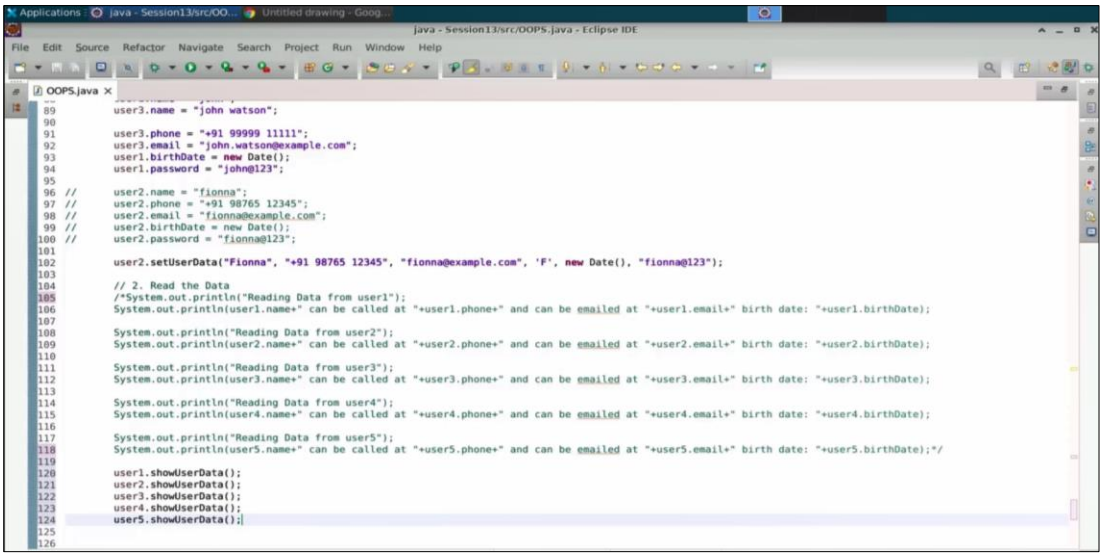
13.1 Run the code, and you will notice that wherever you are writing these five lines of code, you are done with a single line of code and then the data is available in the object. Let us try to find a solution where you need to read the data



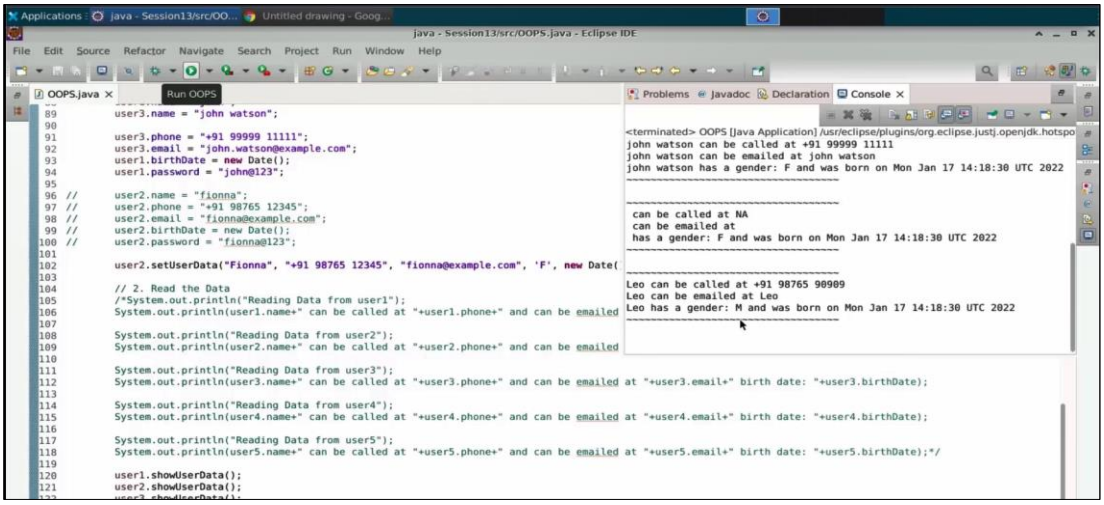
13.2 For reading the data, you can write a method called **showUserData** and then print out the name, followed by the phone number. Then, you can print the email, followed by the name. Next, print the gender and the birth date. The password will not be shown. You can make this prettier by using this basic representation and adding an empty print line at the end



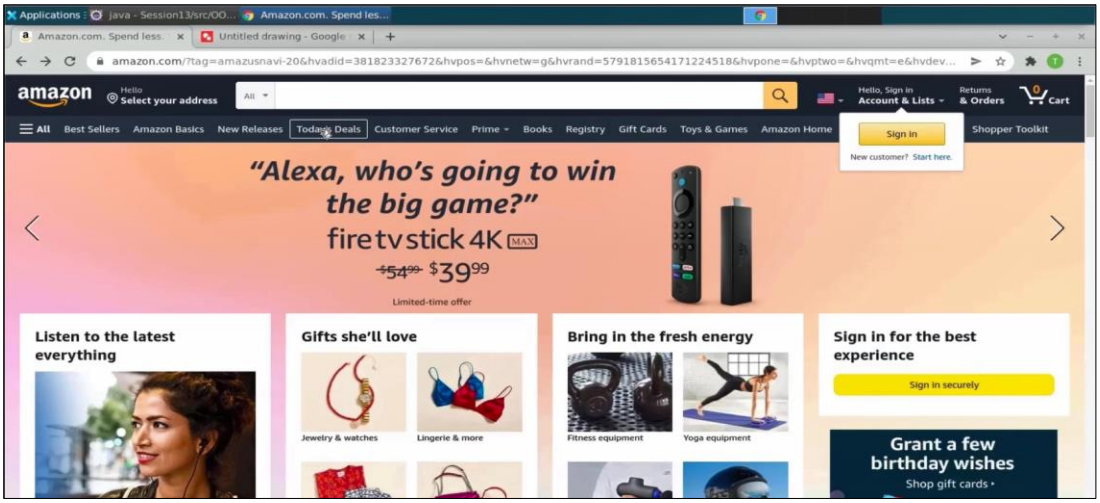
13.3 Moving ahead, rather than showing the data in this manner, you can just write **user1.showUserData()**, **user2.showUserData()**, **user3.showUserData()**, **user4.showUserData()**, **and user5.showUserData().** This way, you have five objects



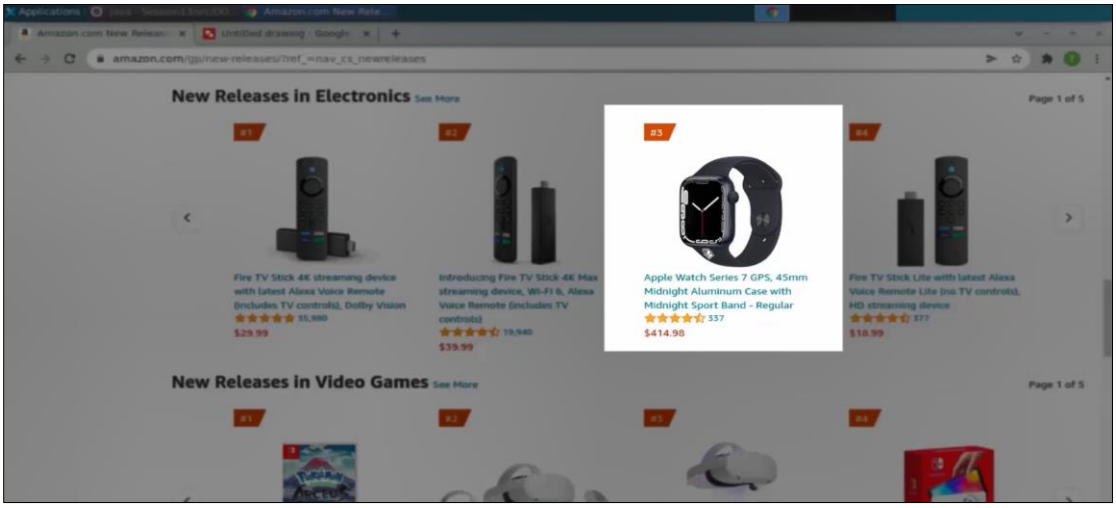
13.4 Run the code, and here a beautiful representation of all five objects is shown. It has John Watson, then Fionna, and John Watson again, as user one and user three are different references but they are pointing to the same object



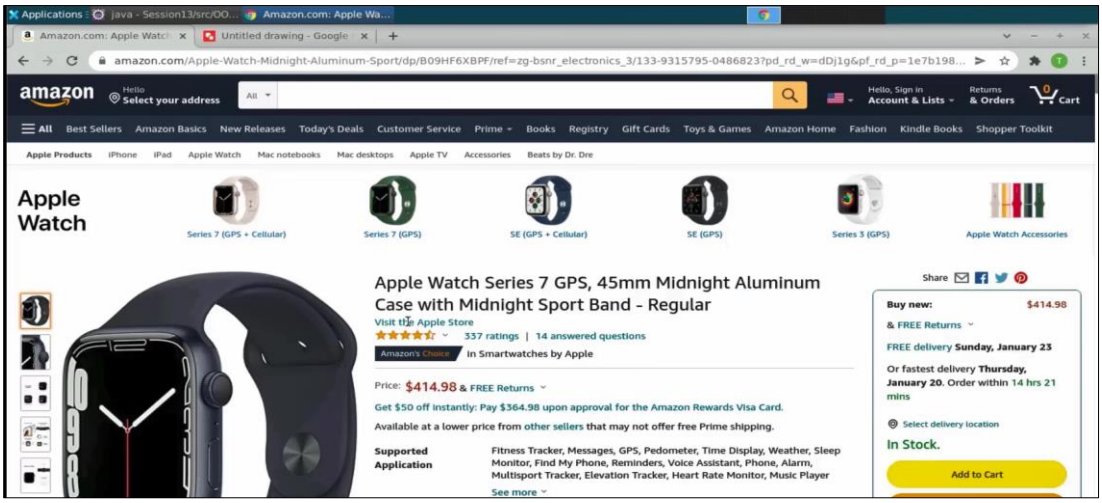
13.5 Moving ahead, you will now see how to design the product object with the same use case on the e-commerce platform. For that, let us go to the e-commerce web application on Amazon. Here, let us click on some new releases, and you can see some products listed



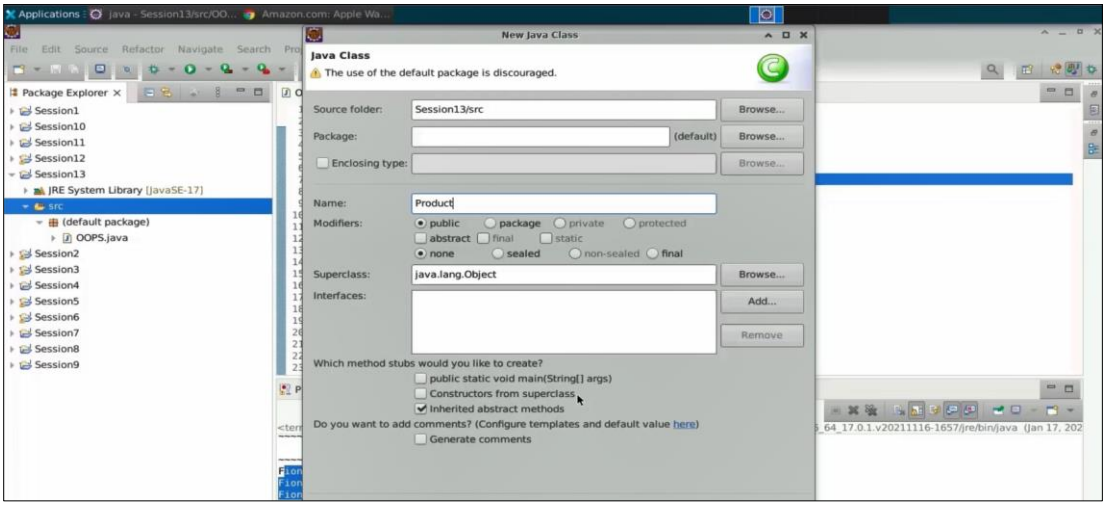
13.6 You can click on one of the products called the Apple Watch series.



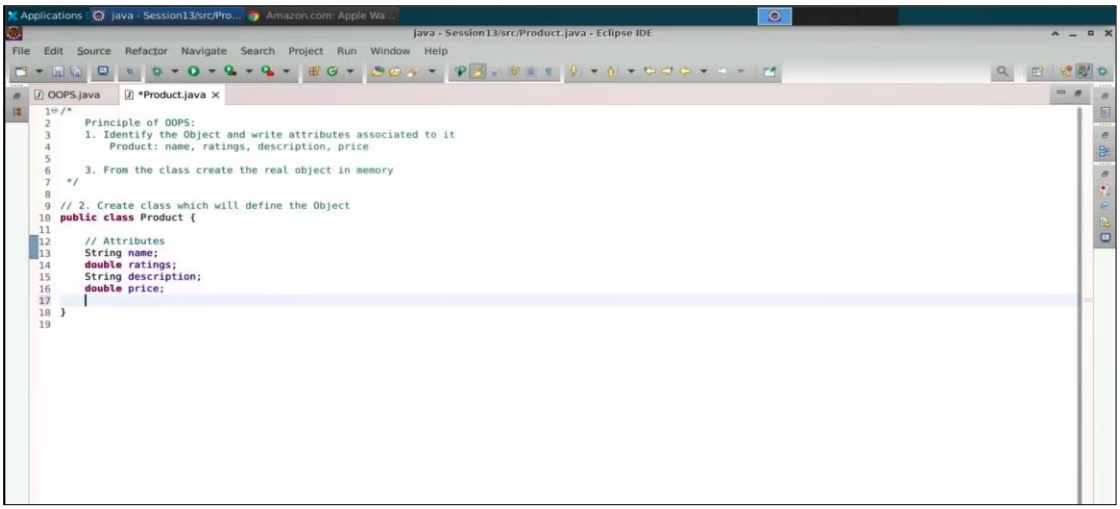
13.7 This is how a product object is described, it has a name or a title, and it has some readings and moving ahead it has a description and then the price associated with the product object



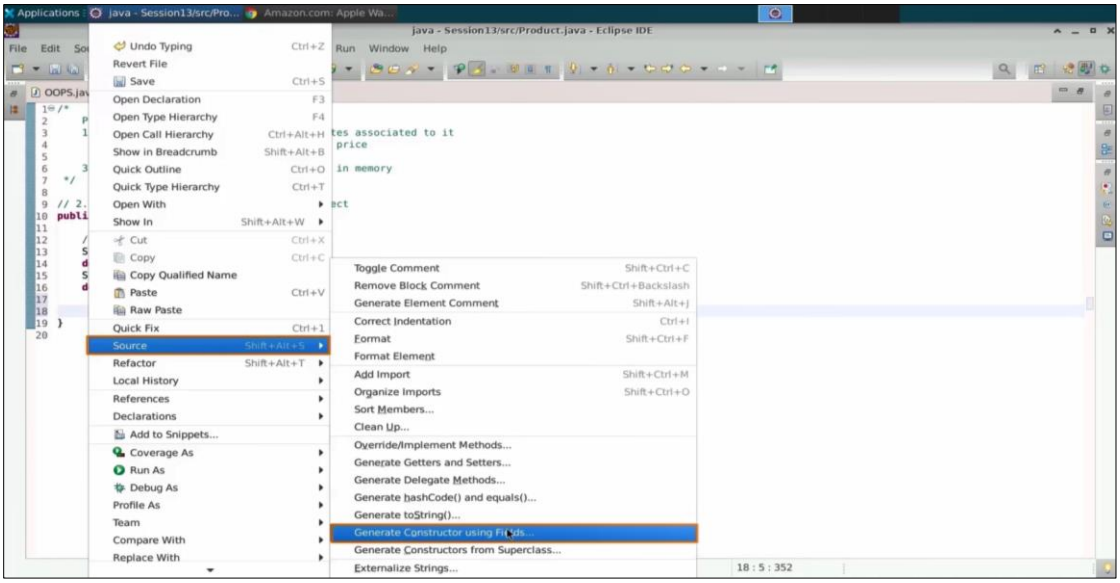
13.8 Let us also follow the same philosophy, create a new class, and name the class itself as **product** and you will not be using the create method here, it is just one class by the name of the product which you will be creating separately as a separate program



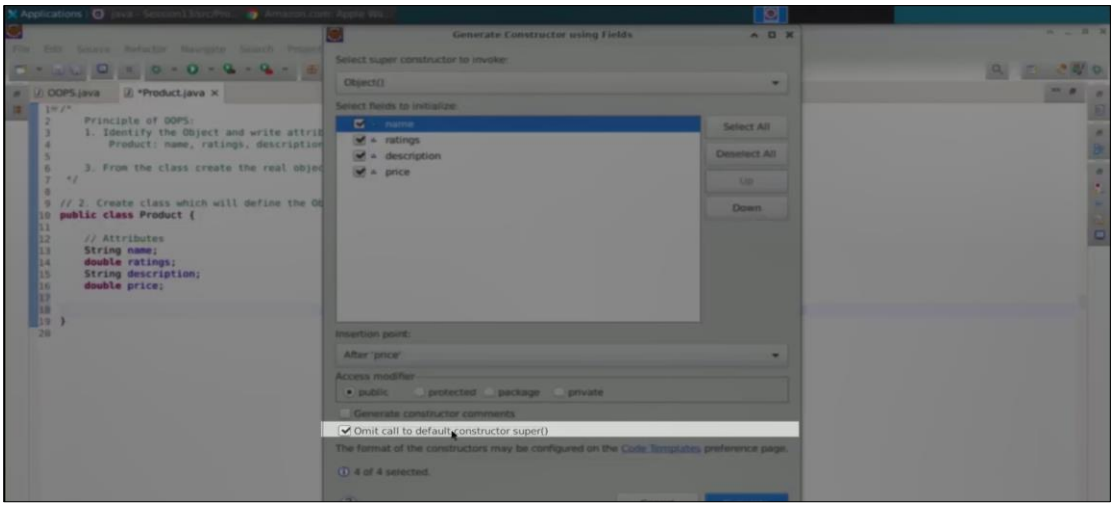
13.9 For the product object, the first key step is to identify the object and list the attributes associated with it. So, the object should be named **Product**, and it should have a name, ratings, description, and price. Let us now define these attributes. The name is going to be a string. The ratings can be a double or a floating-point number, the choice is yours. Then, you have the description as a string. And the price can again be doubled



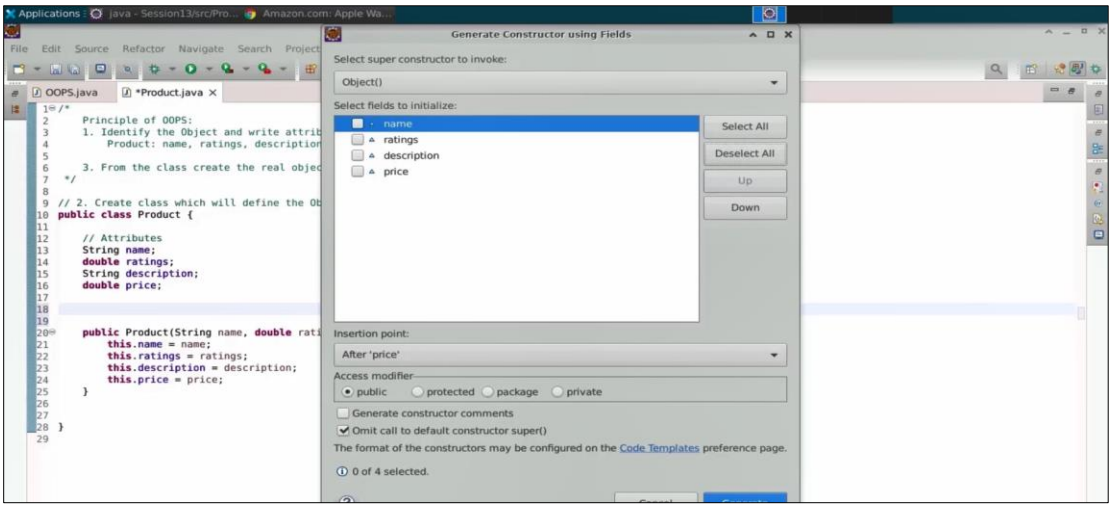
13.10 In order to make things a bit simpler, you can do right click and select **source > generate the constructor using the fields**



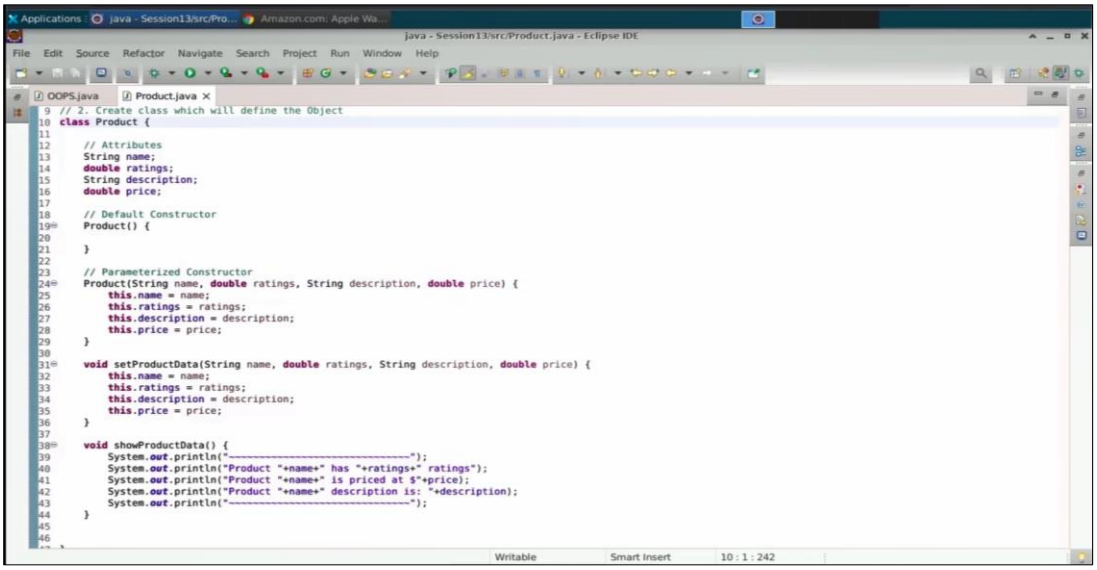
13.11 You will select, the **omit Call to the default constructor**, which is a super construction which is basically concerning inheritance. Select all these fields and click **generate** and you will notice that you are getting a parameterized constructor in action



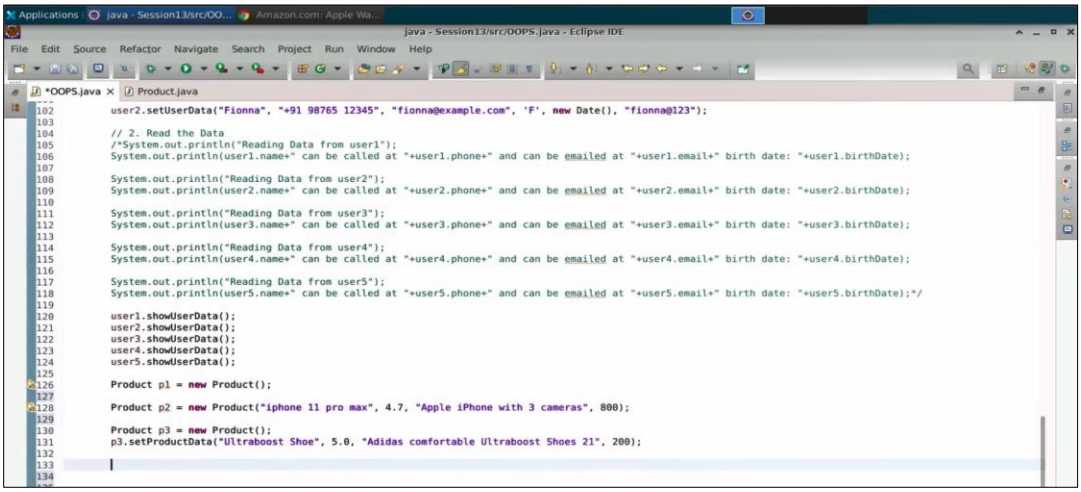
13.12 Let us try to come here and select the **default constructor** which goes like source, then generate the constructor using the fields. You can **deselect all** and select **generate** and here you are with your default constructor



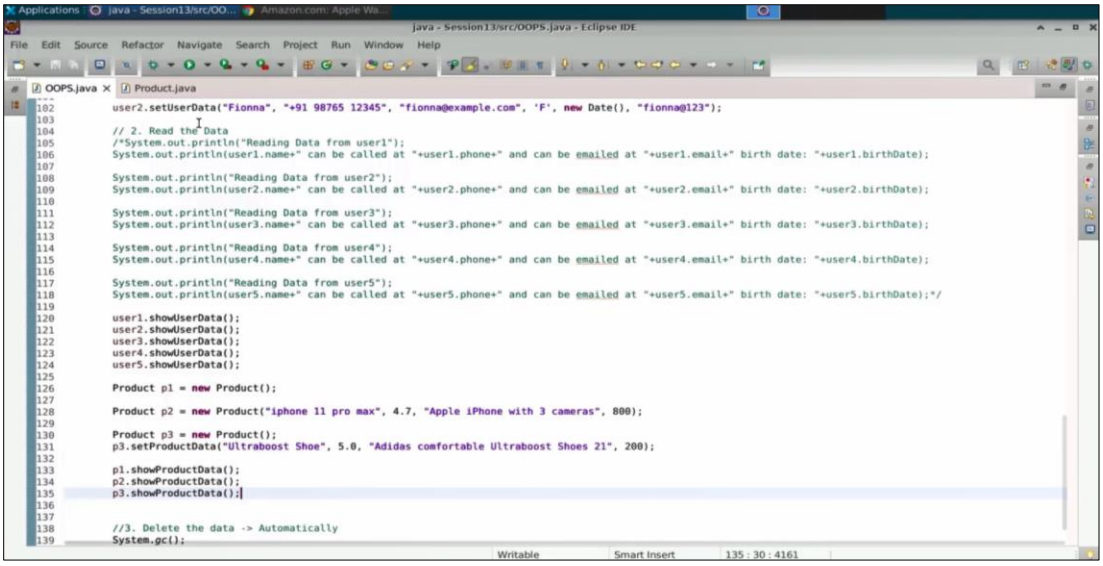
13.13 Next, let us create a method called **setProductData** (or any name of your choice). You can copy the entire definition of the parameterized constructor and use it within the setter method. Similarly, create a method called **showProductData**. In this method, write **Product** followed by the **name**, then "**has ratings**" and the **ratings**. Then write **Product** followed by the **name**, "**is priced at $**" and the **price**. Finally, write **Product** followed by the **name**, "**description is**" and the **description**



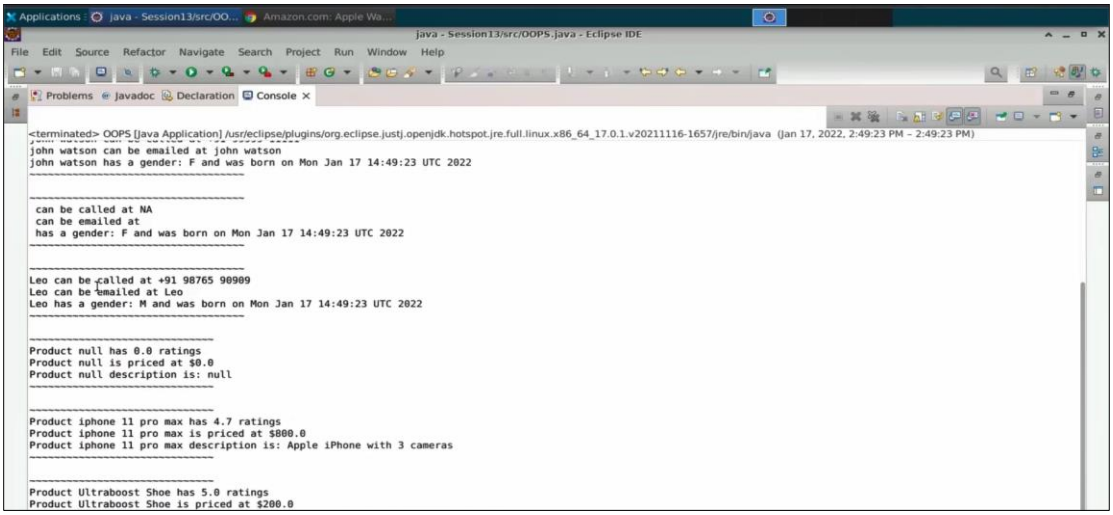
13.14 Come back to **oops.java** and create the product objects. Let us write **Product P1** as a **new product** with the default configuration. Then, write **Product P2** as a **new product**, which will be created through the parameterized constructor. Next, create **P3** as a **new product** for which you can set the data later. To set the product data, you can write **ultraBoost Shoe**, rating as 5.0, description as **'Adidas comfortable ultra-boost shoes, 21'**, and the price as **200**



13.15 Let us understand how to see the data in the object. You can execute the method called **showProductData** on all three objects, where **P1**, **P2**, and **P3** are not objects themselves but references to the objects. In **Product.java**, you need to write an empty print line in the **showProductData** method to create a better-distinguished output



13.16 Run the code, and if you notice here, the default product is priced at zero, has zero ratings, and has a null name with a null description. When you create a parameterized constructor, these are the details that are included in the object. Using the **setData** method, you can also have the data populated



By following the above steps, you have successfully Designed the User and Product Objects in Object-oriented programming.