Lesson 06 Demo 07

Implementing Optional Class in Java

Objective: To implement the Optional class in Java to handle null values and avoid NullPointerExceptions

Tools Required: Eclipse IDE

Prerequisites: None

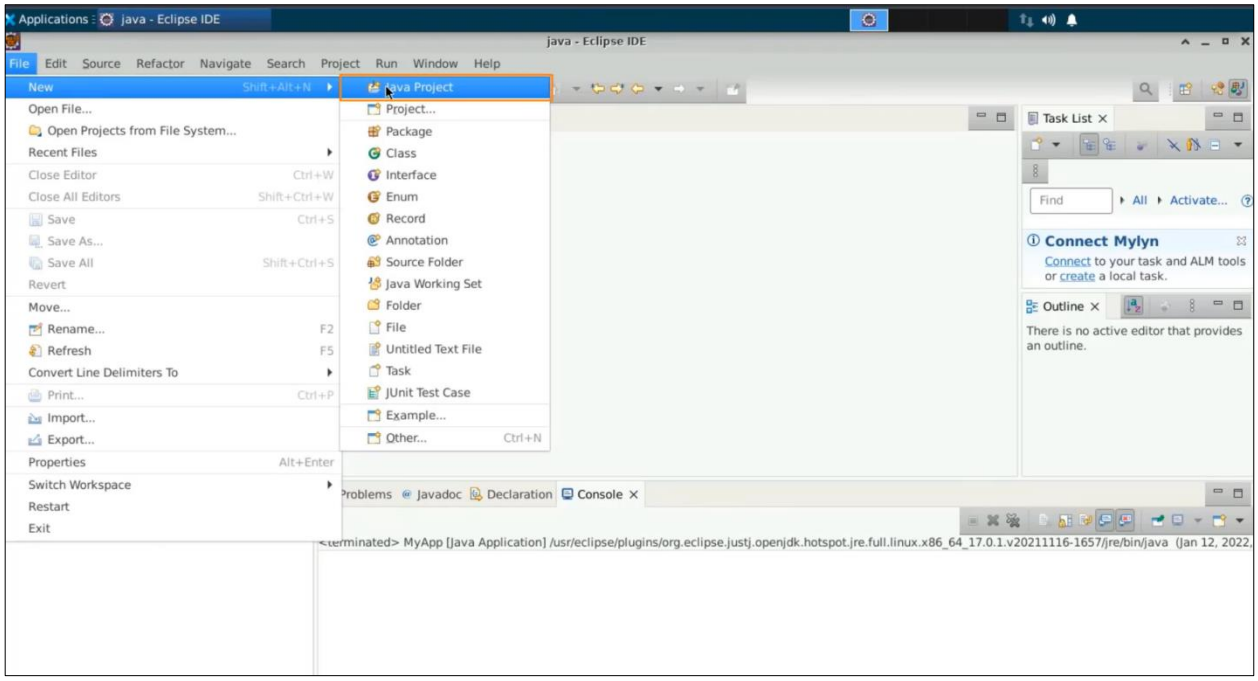
Steps to be followed:

1. Create a new java project
2. Create another constructor which will be parameterized with the values initialized to the inputs
3. Create users with example data and execute the code
4. Implement a safe execution through the optional class
5. Create another optional object and rerun the code

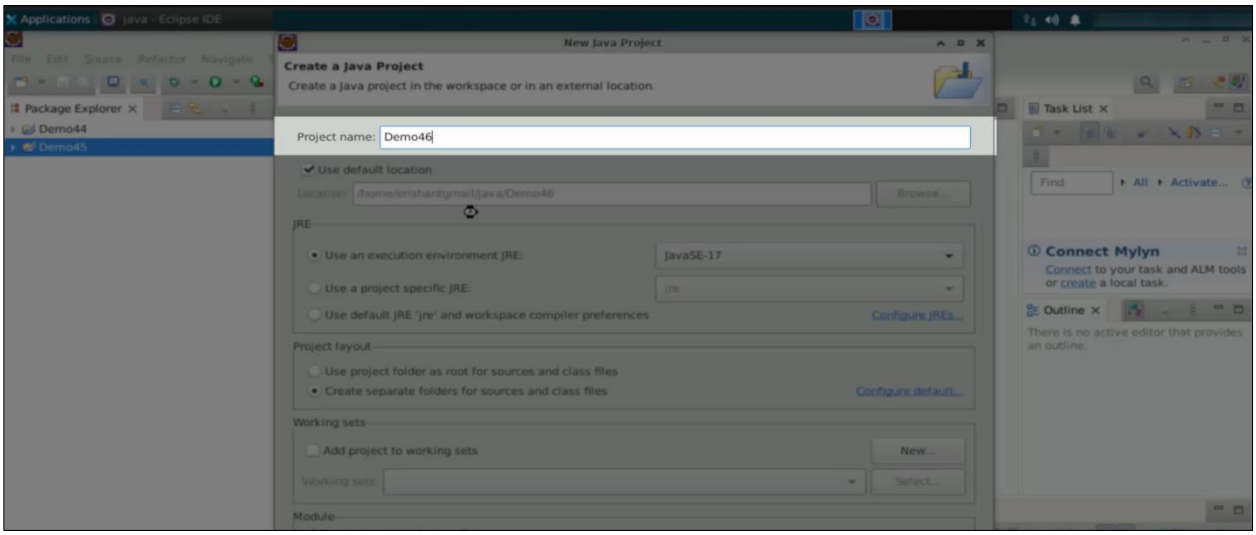
Step 1: Create a new java project

1.1 Open the **Eclipse IDE**

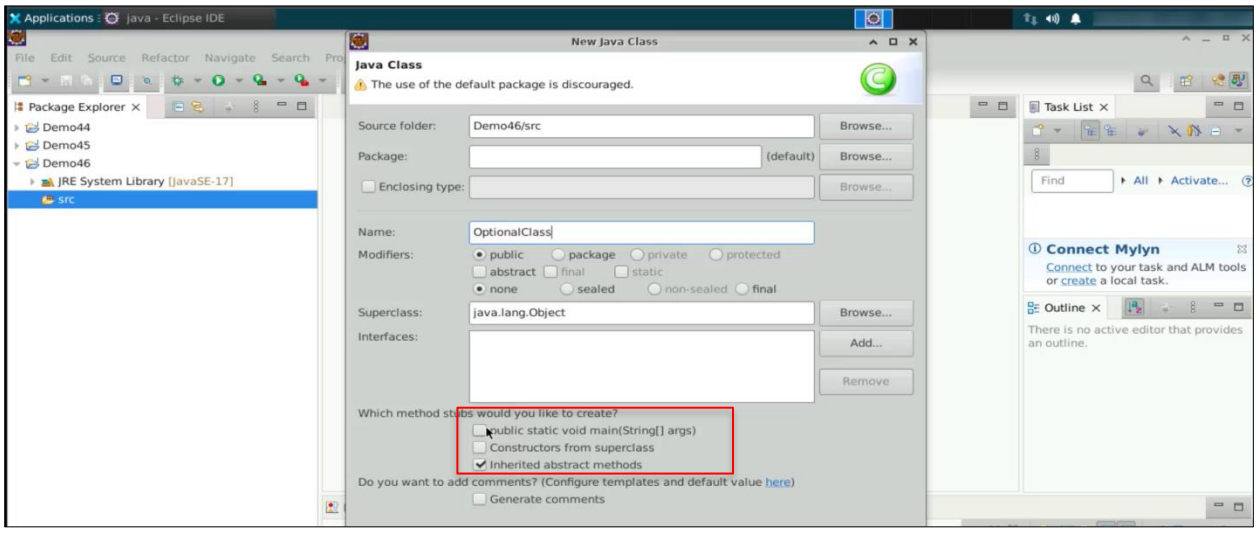
1.2 Select **File**, then **New,** and click **Java project**



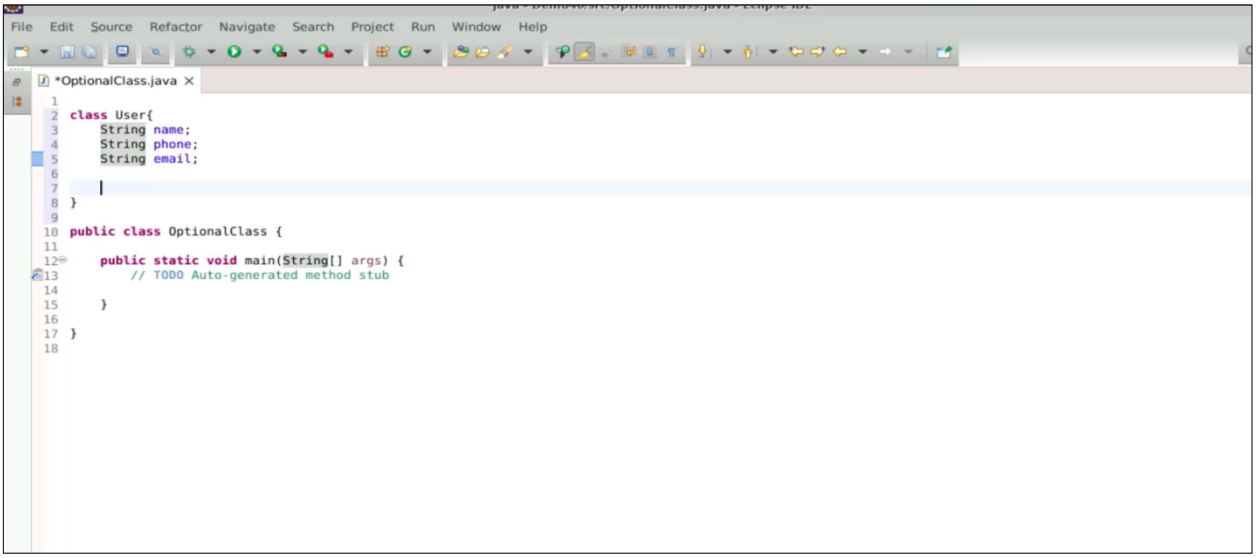
1.3 Name the project as **Demo46,** uncheck **Create a module-info.java file**, and click **Finish**



1.4 Right-click on the **Demo46** folder in the src directory, create a new class and name it **OptionalClass**, select the main method, and click **Finish**

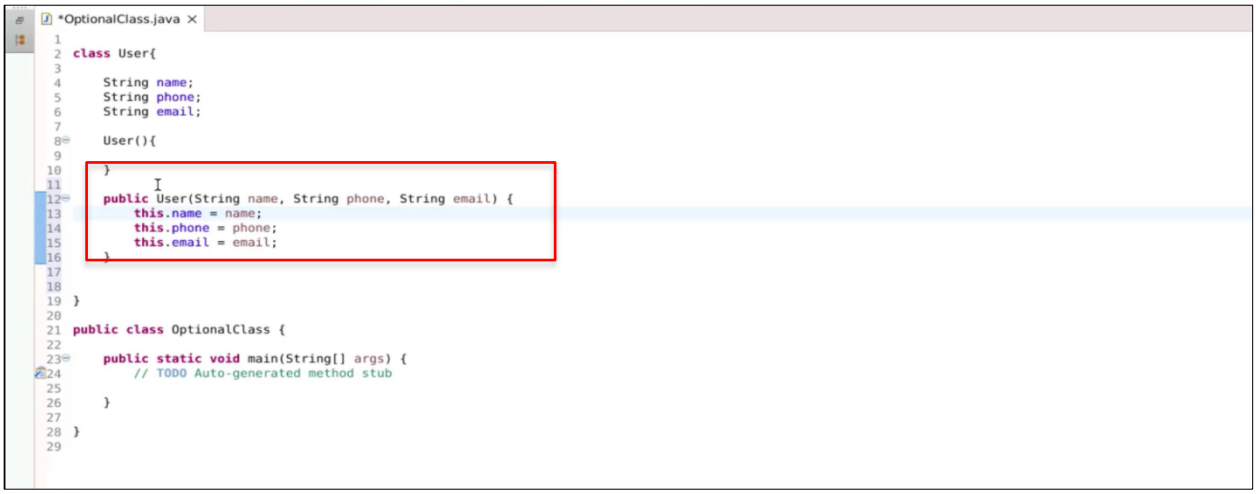


1.5 Create a class with the name **User** which includes attributes such as **name, phone**, and **email**



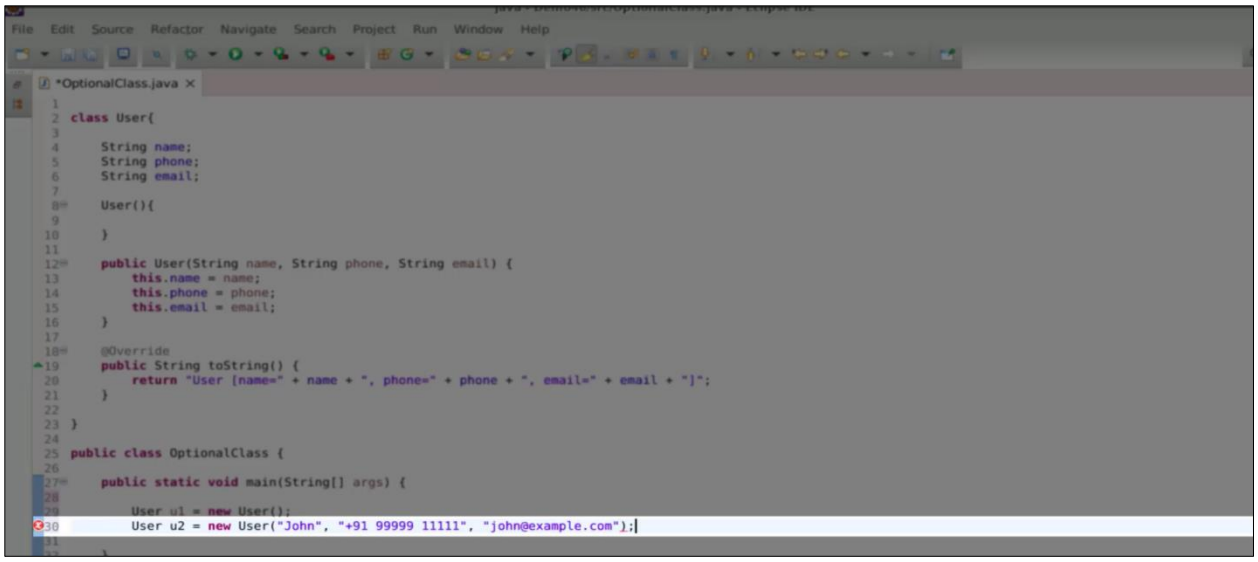
Step 2: Create another constructor which will be parameterized with the values initialized to the inputs

2.1 Create a default constructor for the **User** class without initializing the **name, phone**, and **email**. Then create another parameterized constructor with the values initialized to the inputs

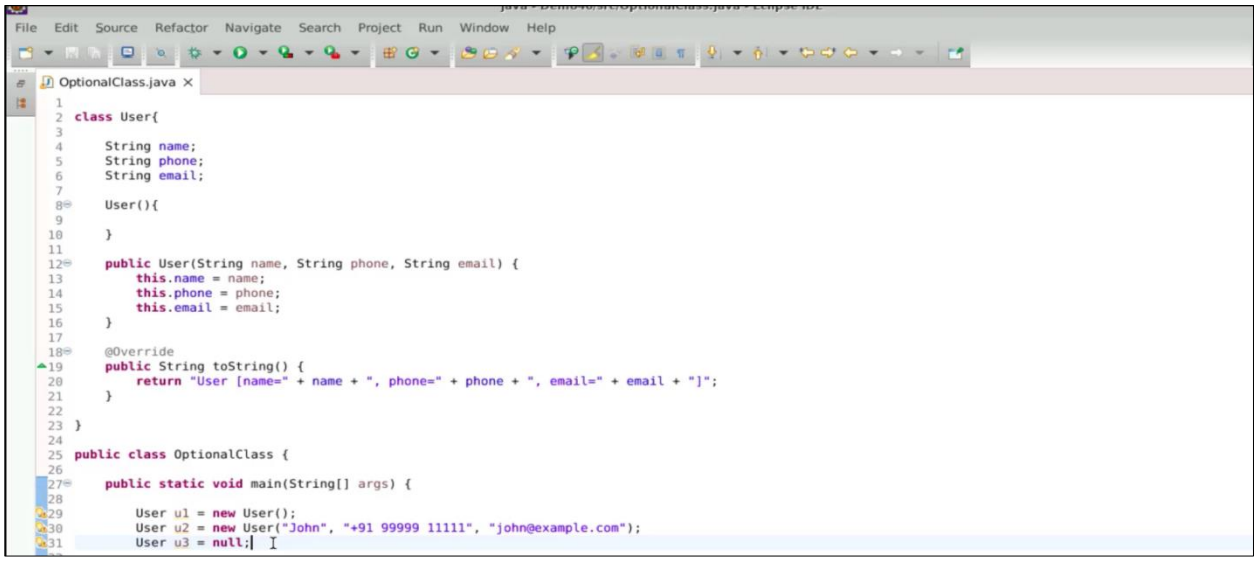


Step 3: Create users with example data and execute the code

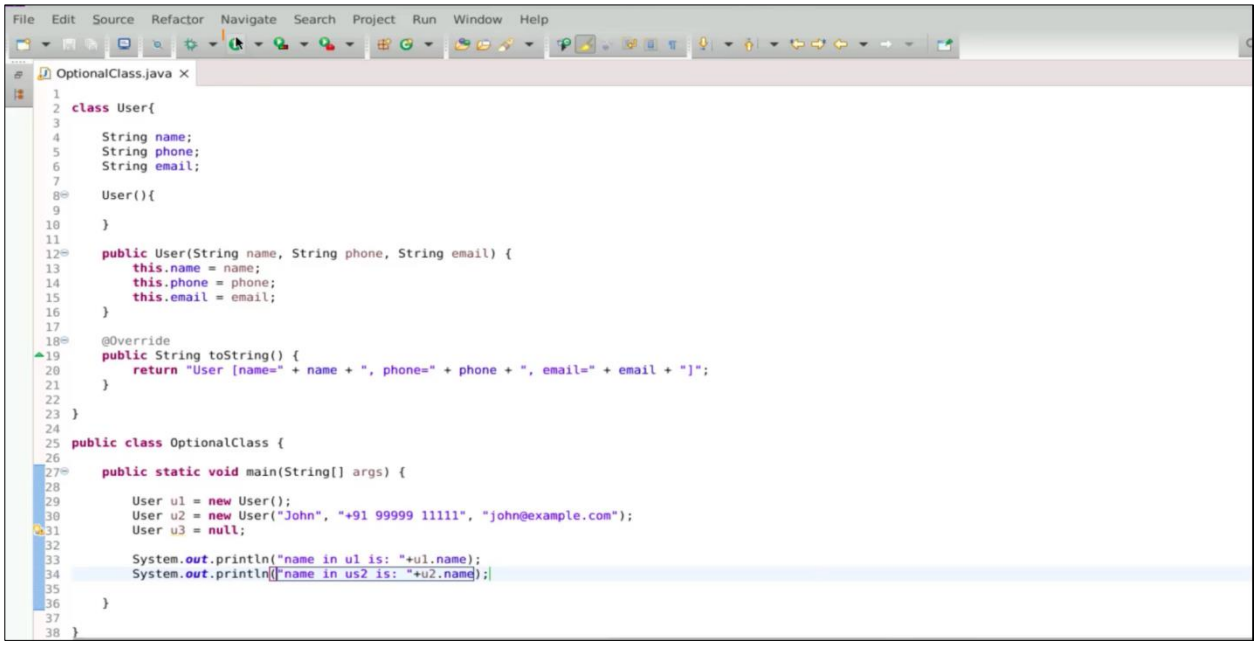
3.1 Create a new user object **u1** and assign it some example data. Similarly, create another User object **u2** with different example data



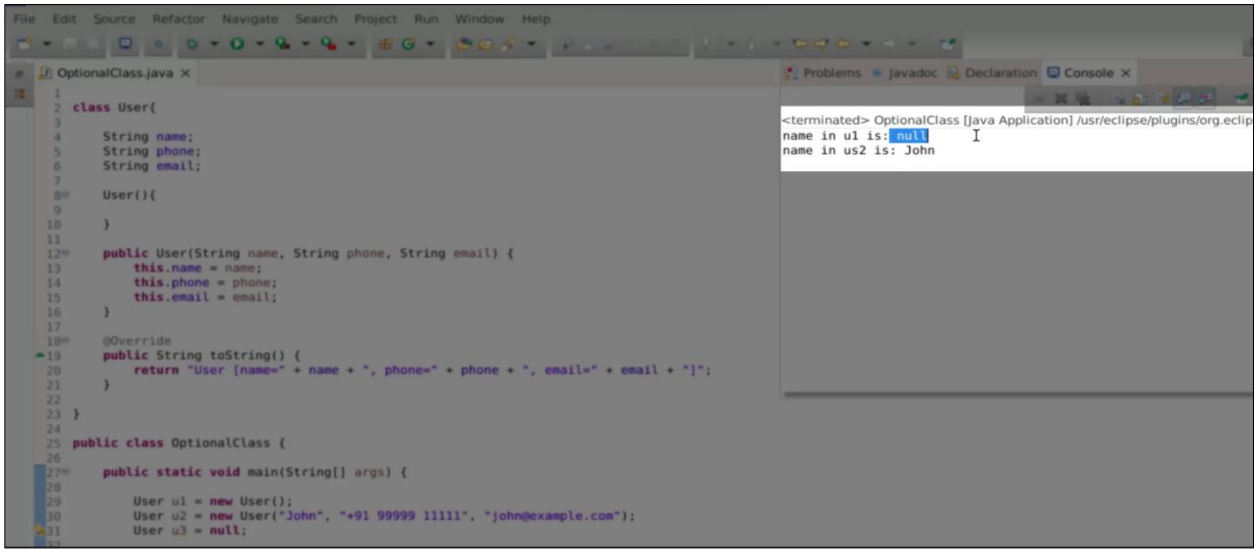
3.2 Create another user object **u3** and set it as null



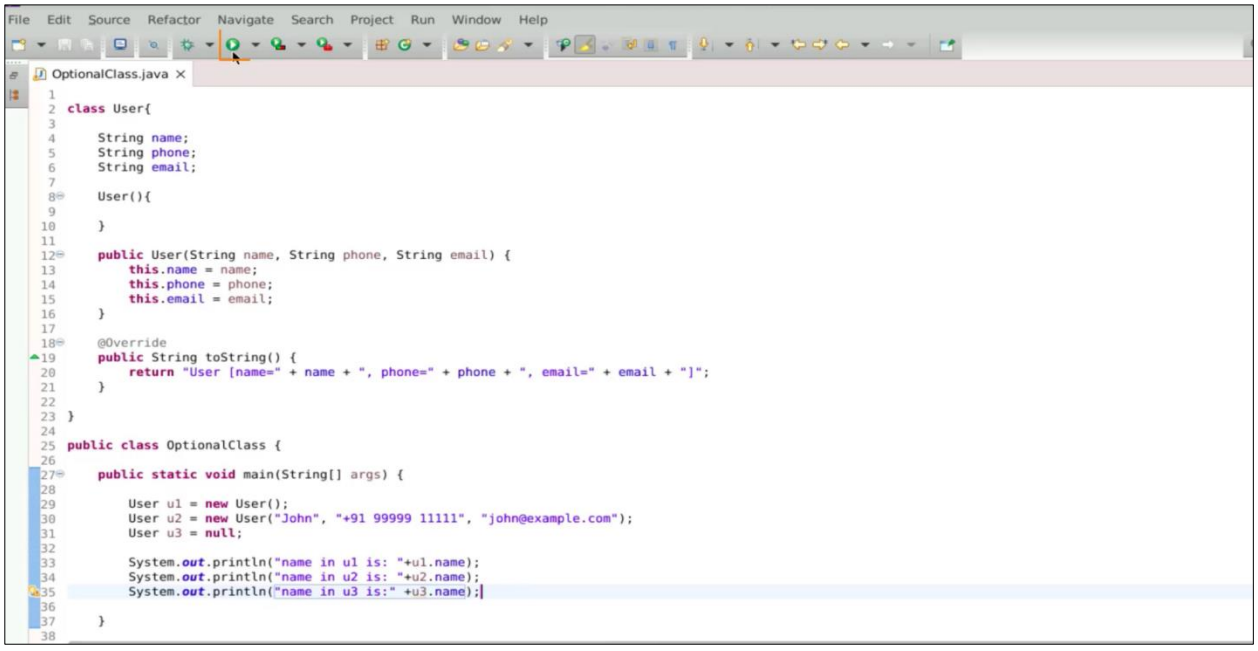
3.3 Print the name of **U1** using **System.out.println(U1.name)** and do the same for **U2**



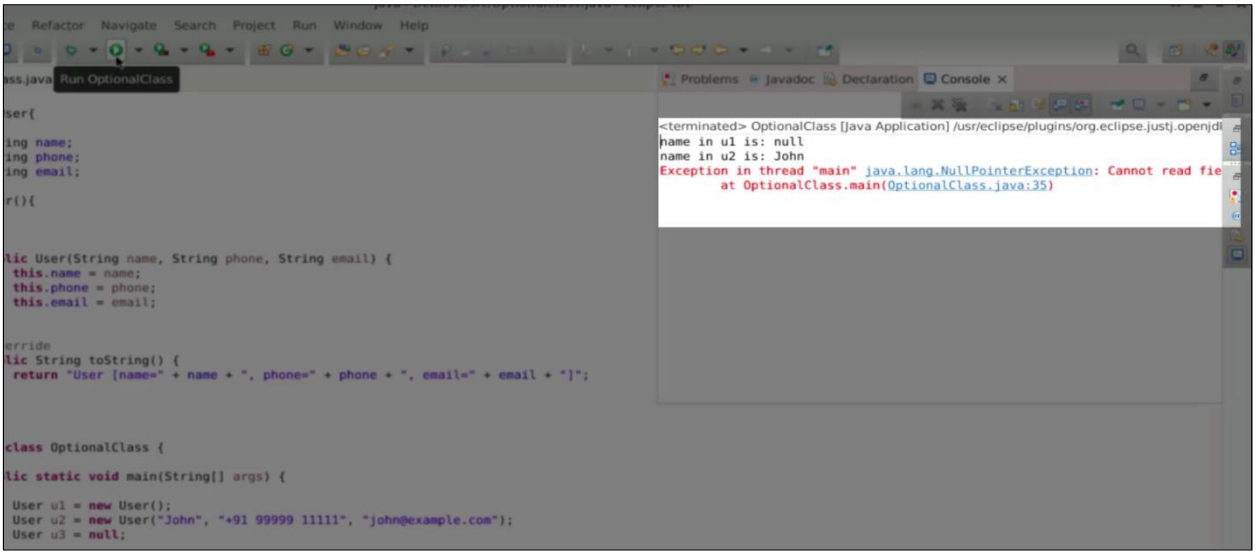
3.4 When you run the code, you will see that the name of **u1** is **null**, while the name of **u2** is displayed as **John**



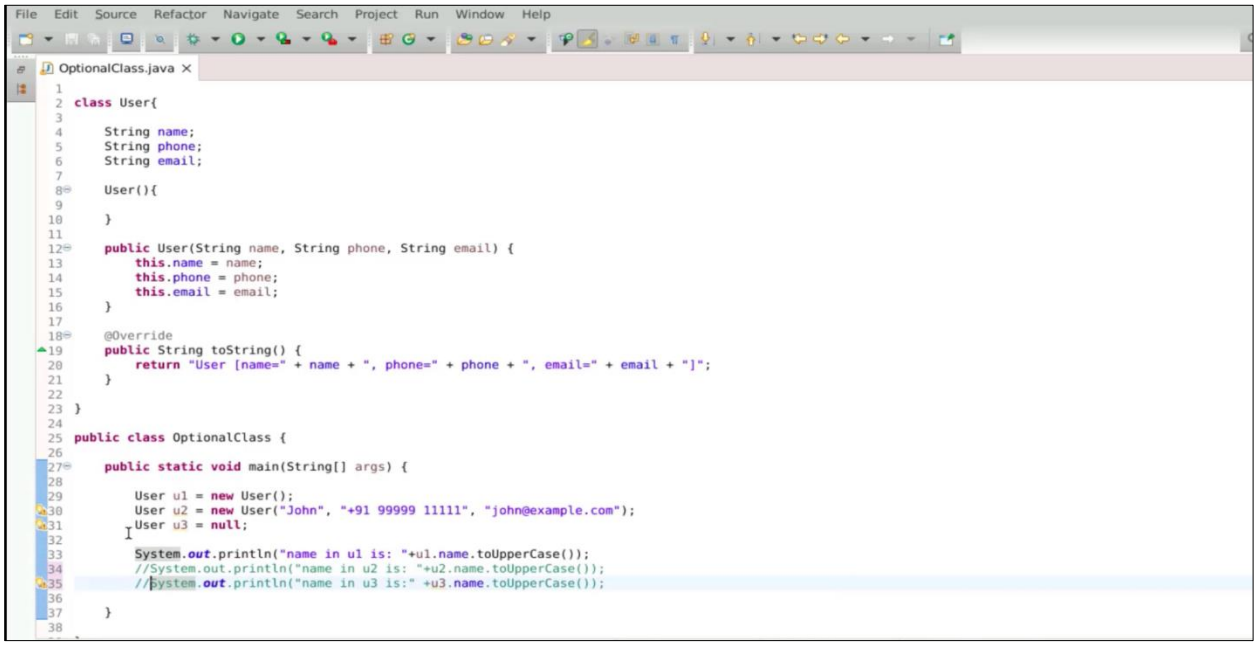
3.5 Print the name of **u3** using **System.out.println(u3.name)**. Although **u3** does not refer to any object, you would not get any compile-time errors because the reference variable is resolved at runtime



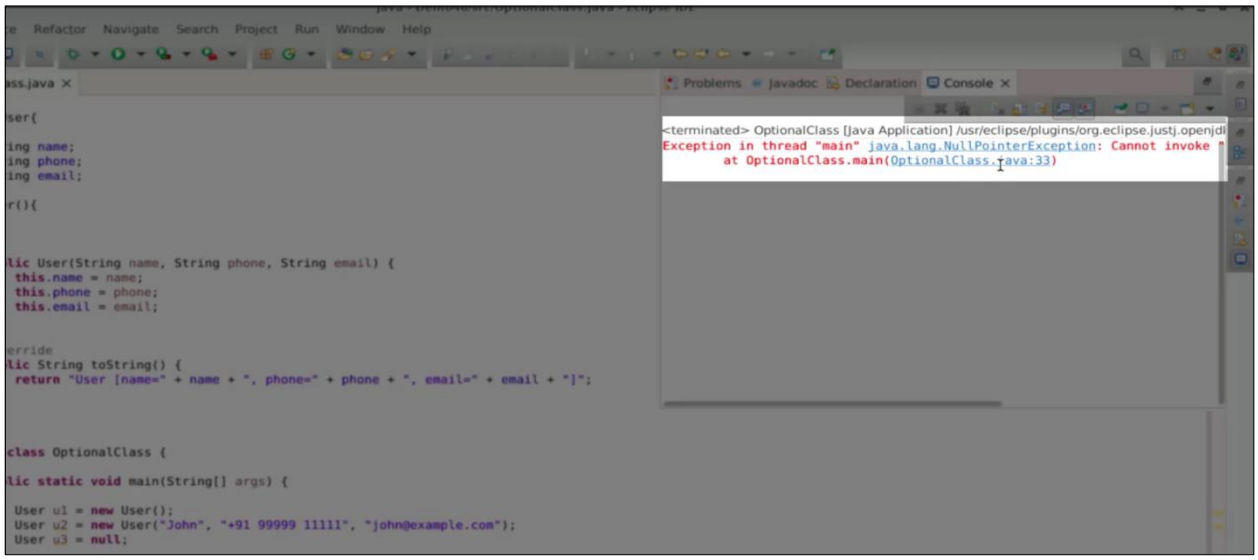
3.6 Running the program will result in an exception at line number 35 when trying to access the name in **u3**



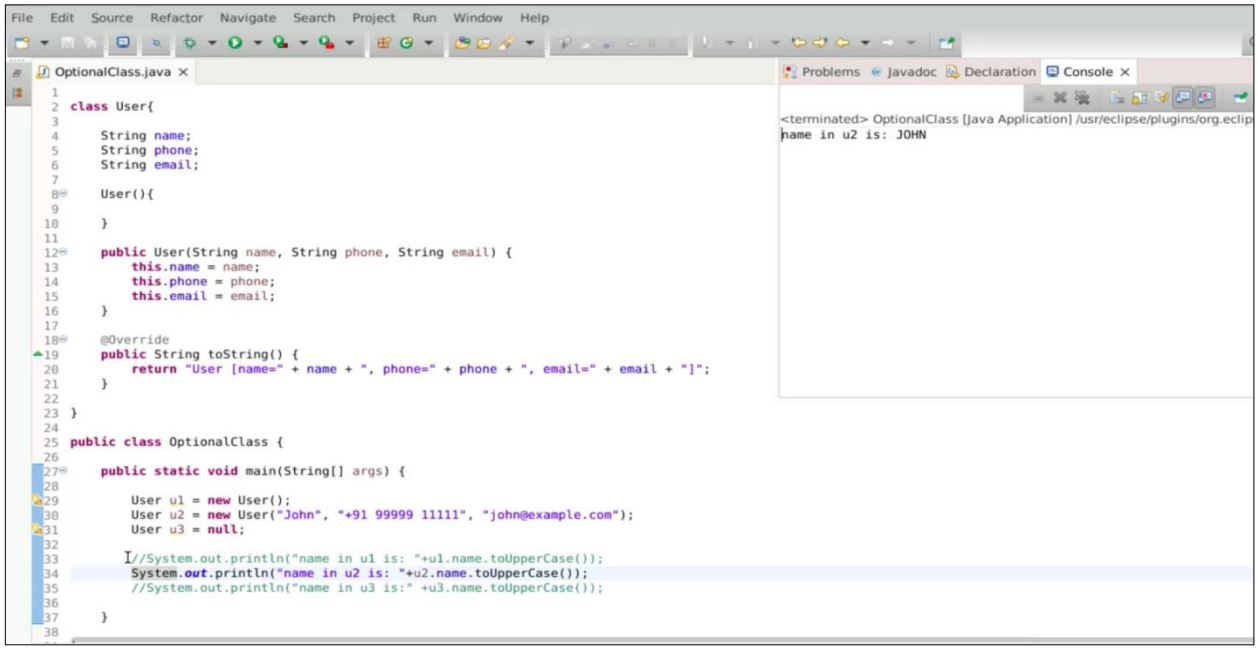
3.7 If you try to execute the **toUpperCase()** method on the **u1** name, you will get a null Pointer exception because **u1.name** is null



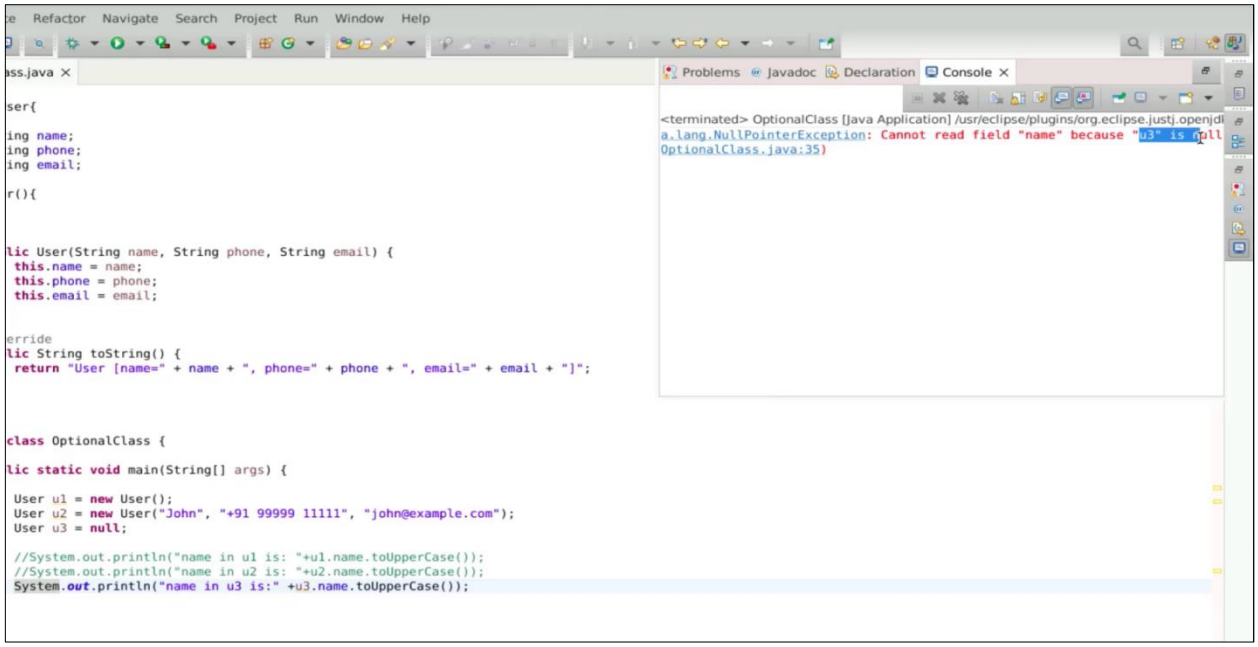
3.8 If you comment out these two lines and run the code, you will receive an error message stating that you cannot invoke the **toUpperCase()** method because **u1.name** is null



3.9 If you uncomment line number 34 and run the code, you will see that the output **name in u2 is: John,** and it is displayed in uppercase

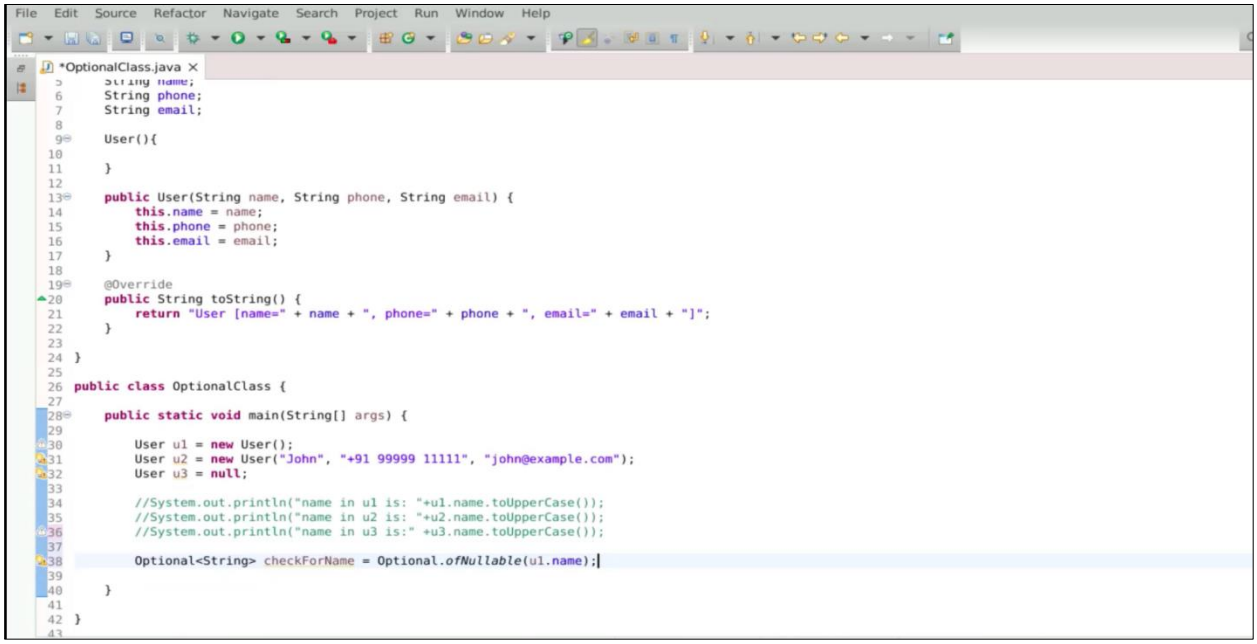


3.10 When trying to access the name of u3 using **u3.name.toUpperCase()**, it will crash because u3 itself is null, showing the error **Cannot read the field name as U3 is null**

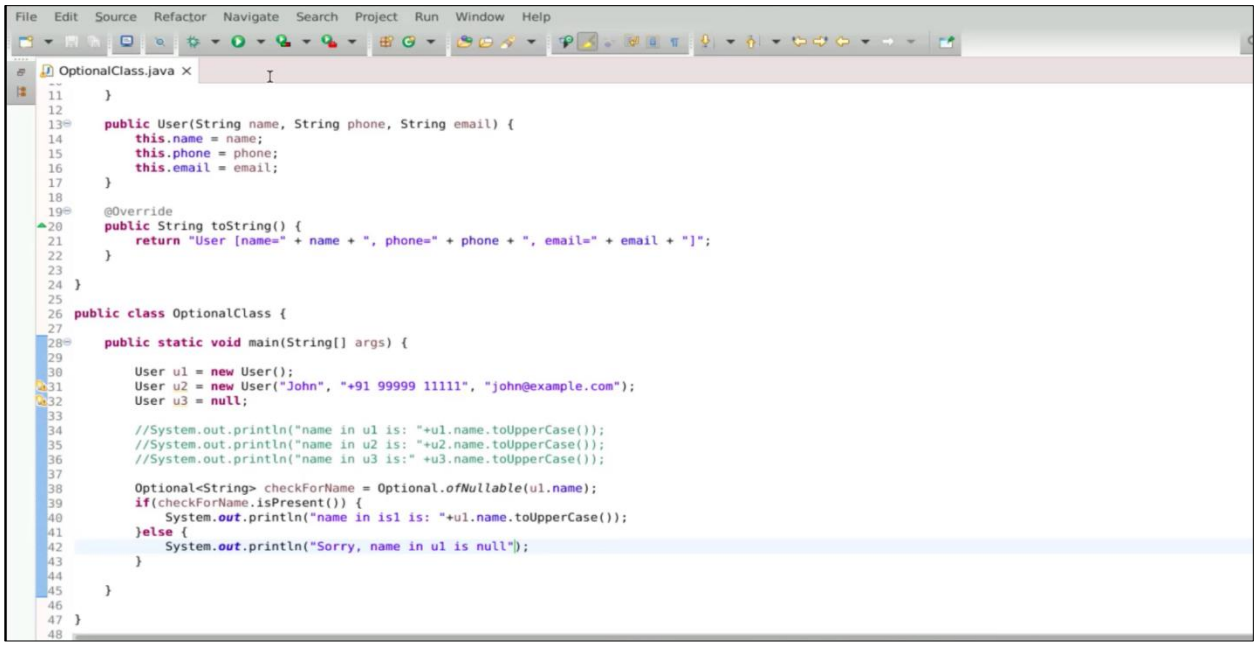


Step 4: Implement a safe execution through the optional class

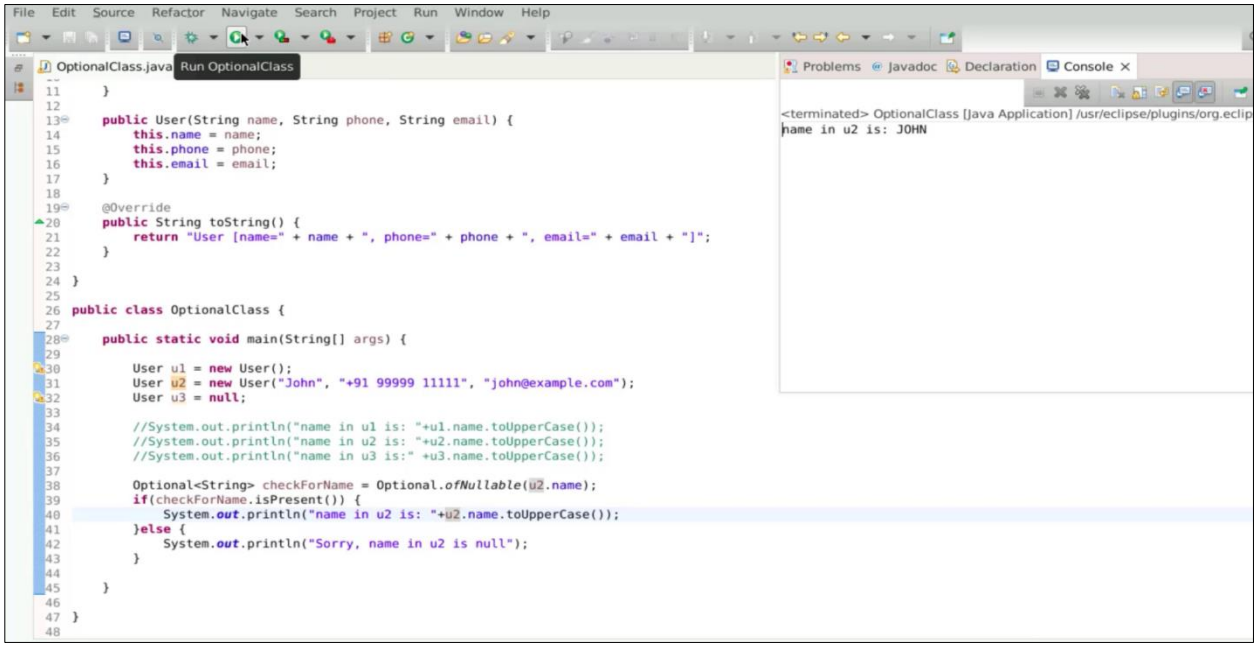
4.1 Create the **Optional** object for the name variable using **Optional.ofNullable(u1.name)**. The **ofNullable()** method checks whether the name in the **User** object is null or not



4.2 Use **isPresent** to check if the name is present. If it is present, print **name in u1 is** followed by **u1.name.toUpperCase().** Otherwise, print **Sorry, name in u1 is null**. This approach provides safer execution compared to the previous approach

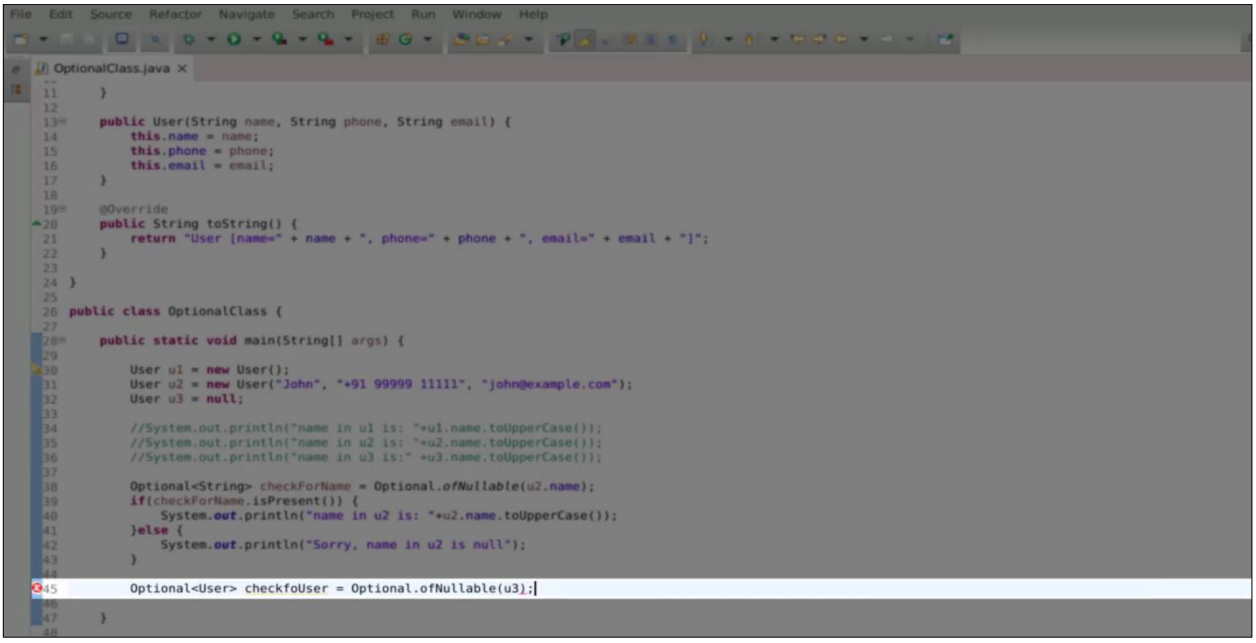


4.3 If you replace the name with **u2,** you will see that it prints **name in U2 is:JOHN**

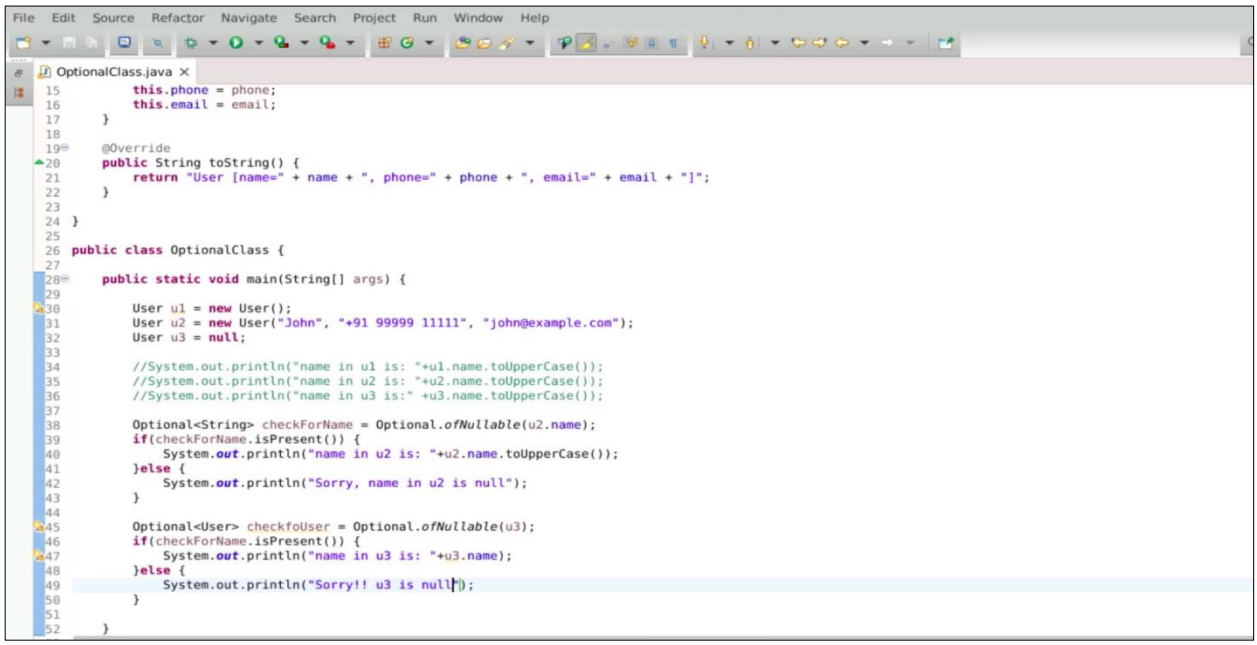


Step 5: Create another optional object and rerun the code

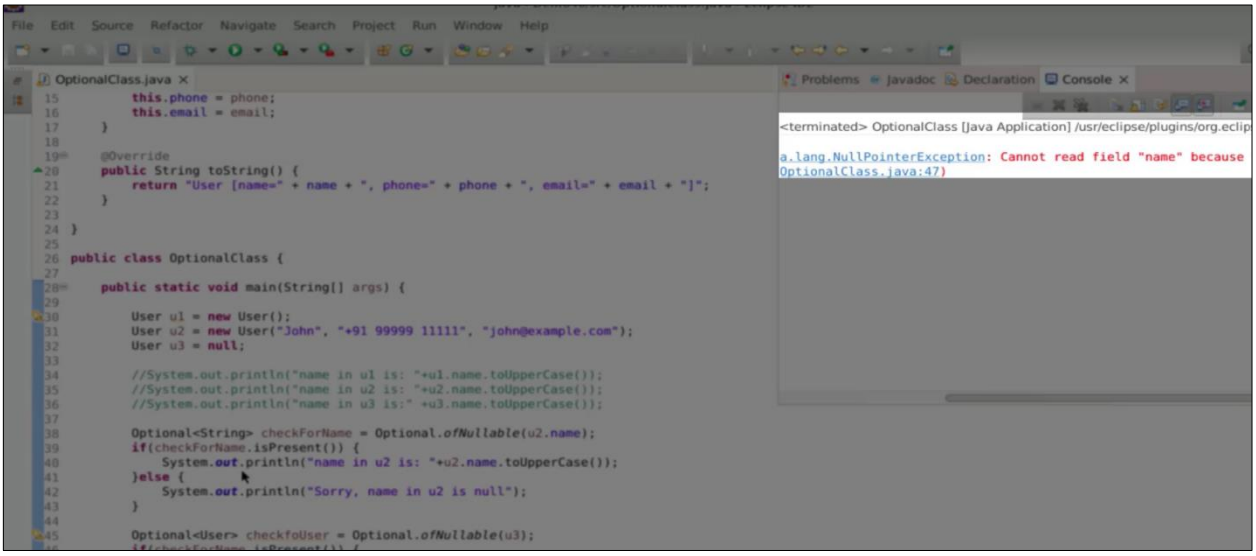
5.1 Now, create one more Optional object, but this time it will work specifically on the User object. Enter **Optional<User>** to define the type of the **Optional**. Check for the User object by using **Optional.ofNullable()** and pass **u3** inside the **ofNullable()** method



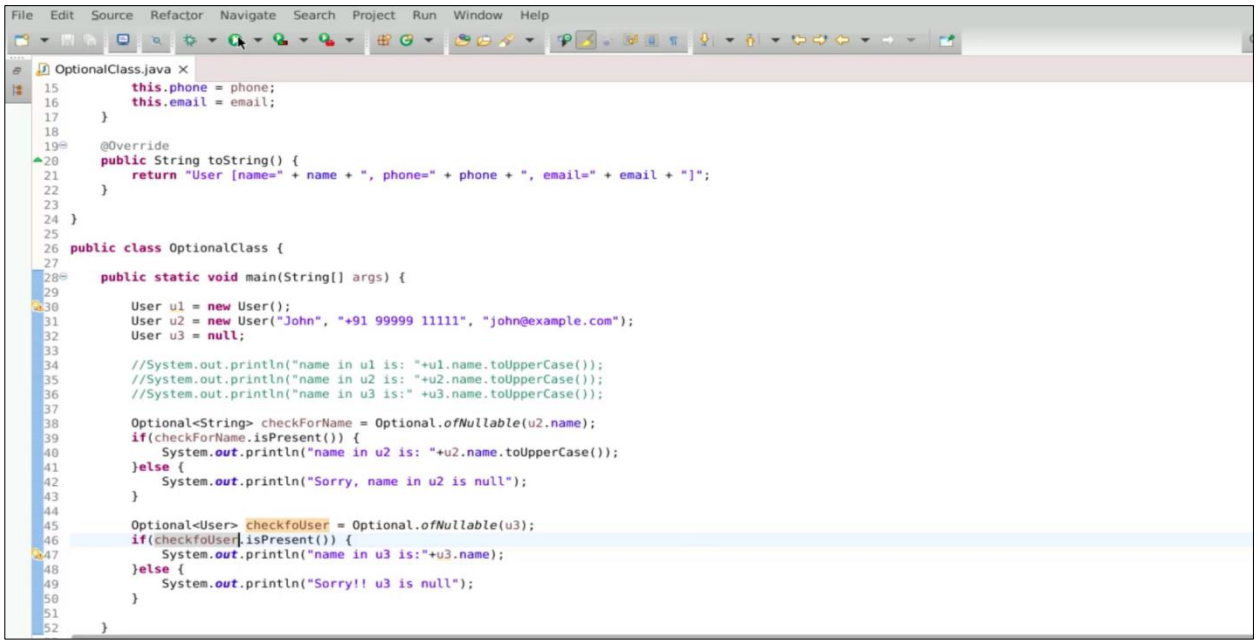
5.2 Next, type an **if** check for **user.isPresent().** This condition checks if there is a user object available and it is not null. If it is present, you can say **name in u3** is followed by **U3.name.** In the **else** case, type **Sorry, u3 is null**. This is a small implementation using Optional



5.3 When you run the code it shows: **Cannot read field name because U3 is null**

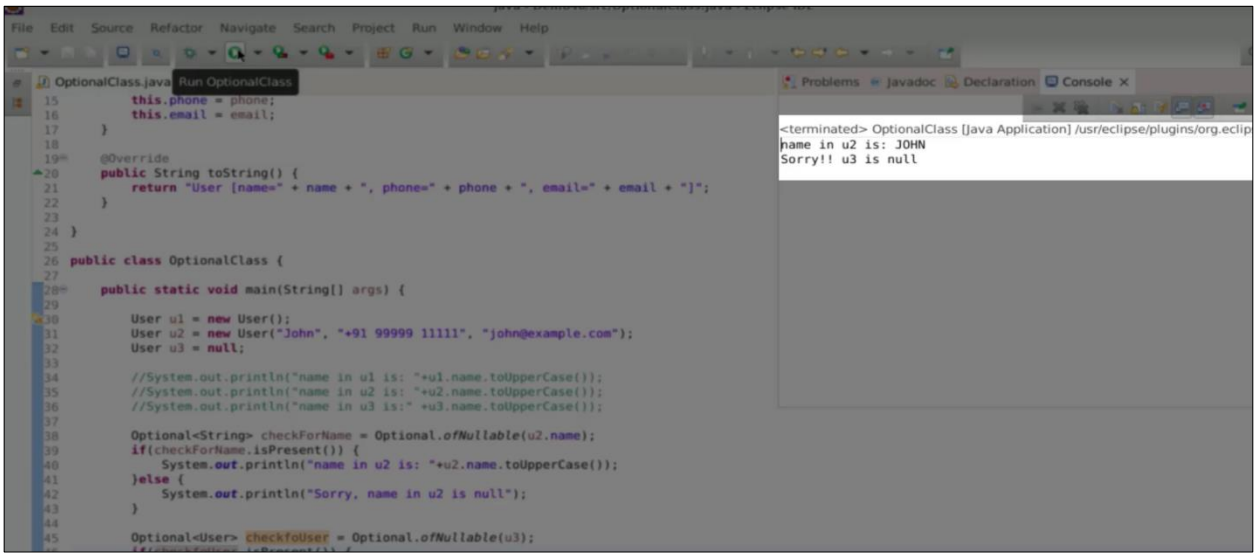


5.4 Here, change this to **checkforUser** instead of check for **name**



```
File Edit Source Refactor Navigate Search Project Run Window Help
OptionalClass.java ×
15          this.phone = phone;
16          this.email = email;
17      }
18
19      @Override
20      public String toString() {
21          return "User [name=" + name + ", phone=" + phone + ", email=" + email + "]";
22      }
23
24  }
25
26  public class OptionalClass {
27
28      public static void main(String[] args) {
29
30          User u1 = new User();
31          User u2 = new User("John", "+91 99999 11111", "john@example.com");
32          User u3 = null;
33
34          //System.out.println("name in u1 is: "+u1.name.toUpperCase());
35          //System.out.println("name in u2 is: "+u2.name.toUpperCase());
36          //System.out.println("name in u3 is:" +u3.name.toUpperCase());
37
38          Optional<String> checkForName = Optional.ofNullable(u2.name);
39          if(checkForName.isPresent()) {
40              System.out.println("name in u2 is: "+u2.name.toUpperCase());
41          }else {
42              System.out.println("Sorry, name in u2 is null");
43          }
44
45          Optional<User> checkfoUser = Optional.ofNullable(u3);
46          if(checkfoUser.isPresent()) {
47              System.out.println("name in u3 is:"+u3.name);
48          }else {
49              System.out.println("Sorry!! u3 is null");
50          }
51
52      }
```

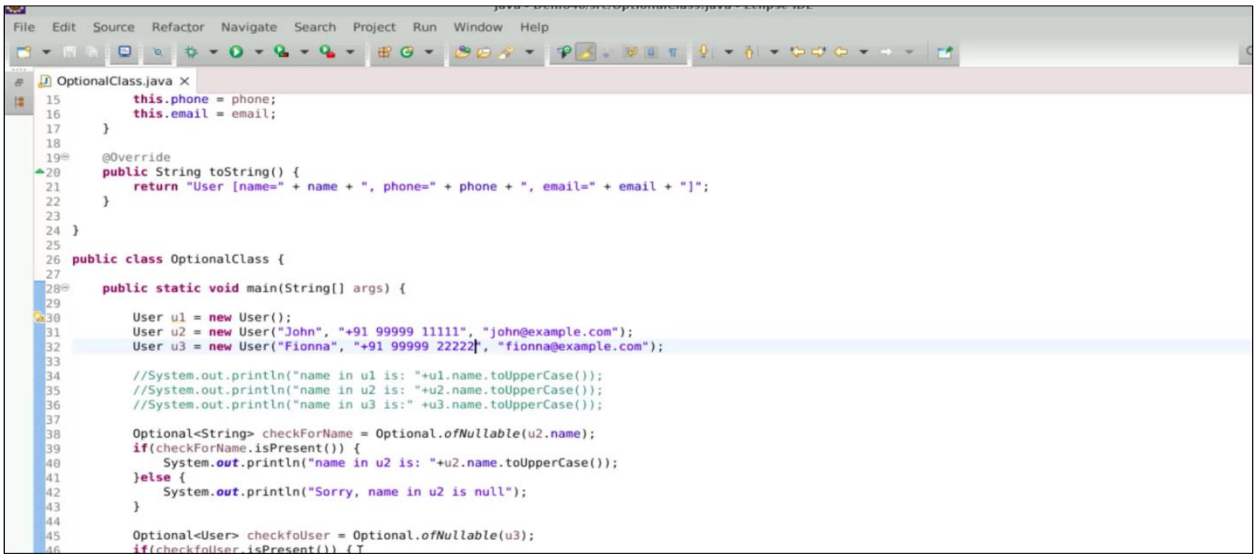
5.5 Now, re-run the code and here you can see it shows **Sorry!! u3 is null**



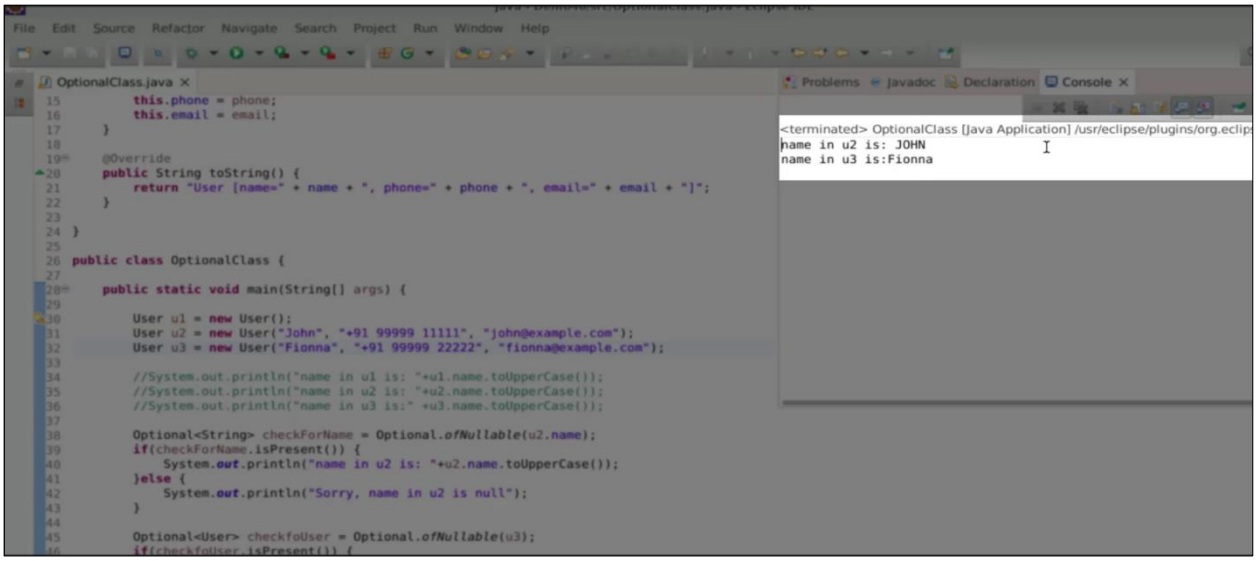
```
File Edit Source Refactor Navigate Search Project Run Window Help
OptionalClass.java  Run OptionalClass
Problems  Javadoc  Declaration  Console ×
<terminated> OptionalClass [Java Application] /usr/eclipse/plugins/org.eclip
name in u2 is: JOHN
Sorry!! u3 is null

15          this.phone = phone;
16          this.email = email;
17      }
18
19      @Override
20      public String toString() {
21          return "User [name=" + name + ", phone=" + phone + ", email=" + email + "]";
22      }
23
24  }
25
26  public class OptionalClass {
27
28      public static void main(String[] args) {
29
30          User u1 = new User();
31          User u2 = new User("John", "+91 99999 11111", "john@example.com");
32          User u3 = null;
33
34          //System.out.println("name in u1 is: "+u1.name.toUpperCase());
35          //System.out.println("name in u2 is: "+u2.name.toUpperCase());
36          //System.out.println("name in u3 is:" +u3.name.toUpperCase());
37
38          Optional<String> checkForName = Optional.ofNullable(u2.name);
39          if(checkForName.isPresent()) {
40              System.out.println("name in u2 is: "+u2.name.toUpperCase());
41          }else {
42              System.out.println("Sorry, name in u2 is null");
43          }
44
45          Optional<User> checkfoUser = Optional.ofNullable(u3);
```

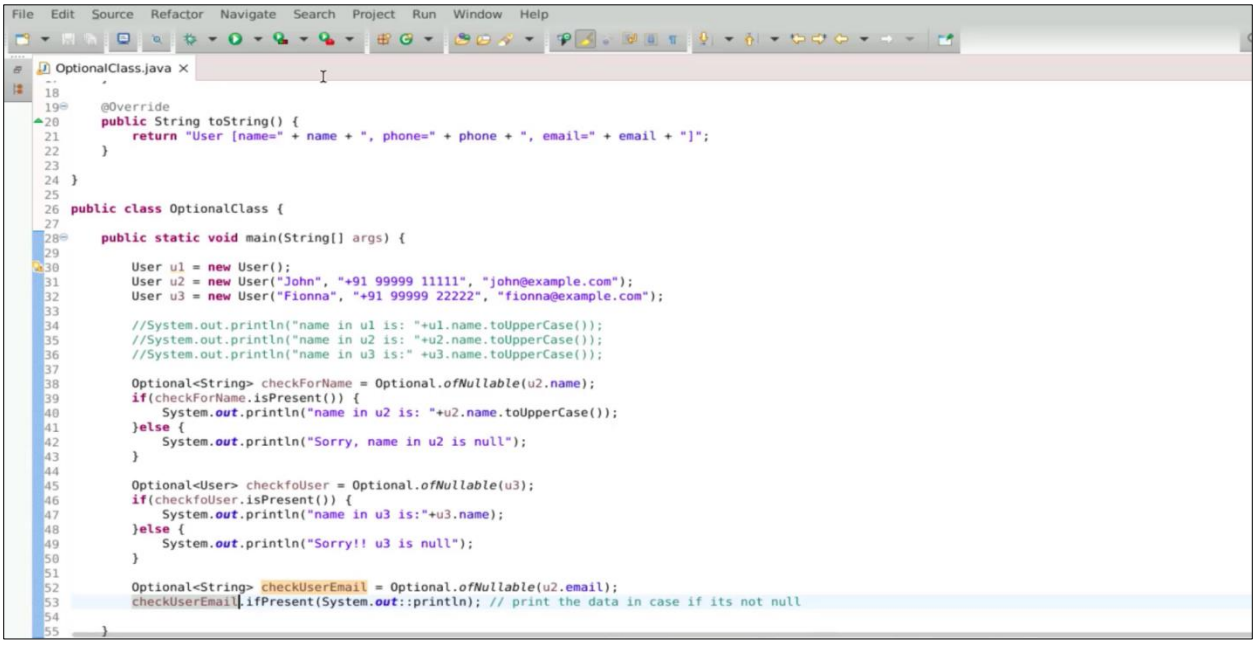
5.6 Create a new user object created and add the given data:
Name= **Fionna**, Email: **fionna@example.com,** and phone number



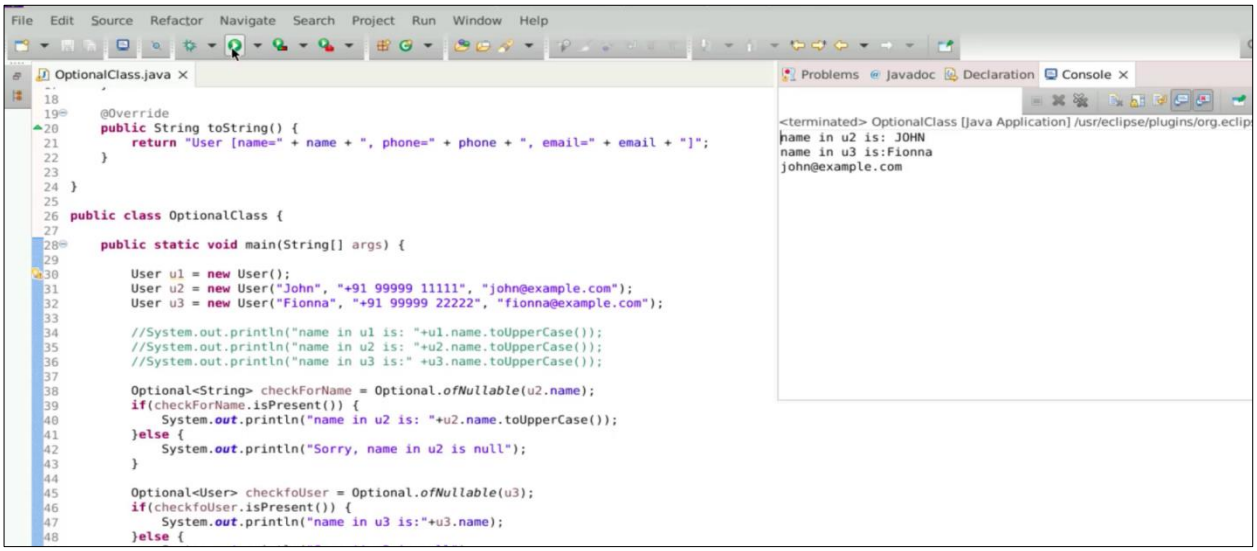
5.7 Now, since this is not null, when you run the program, it will give you the name. You can even check for the name within this method



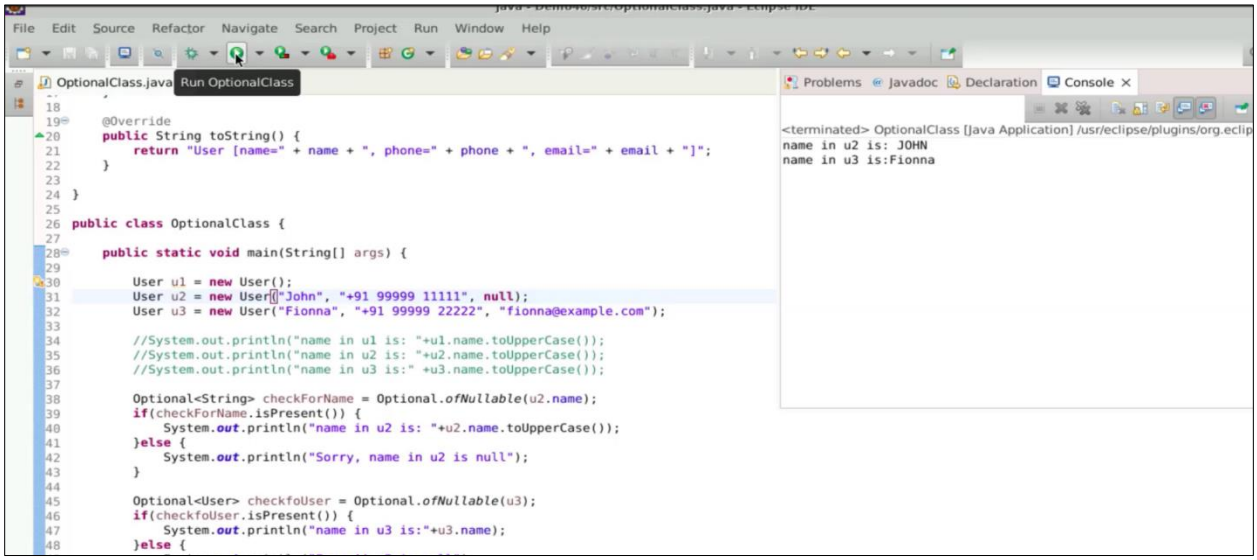
5.8 Enter **Optional<User>** and then check user by using **Optional.ofNullable()** and pass **u2** as the parameter. Next, for the **checkUser**, execute the **ifPresent()** method and pass a lambda function. Enter **ifPresent(System.out::println)** to print the data if it is not null. It's better to convert it to a string, so use **String.valueOf(checkUser.email.orElse(null))** to handle the email which may be null. Finally, print the data using **checkUser.email**. If the email is not null, it will be printed



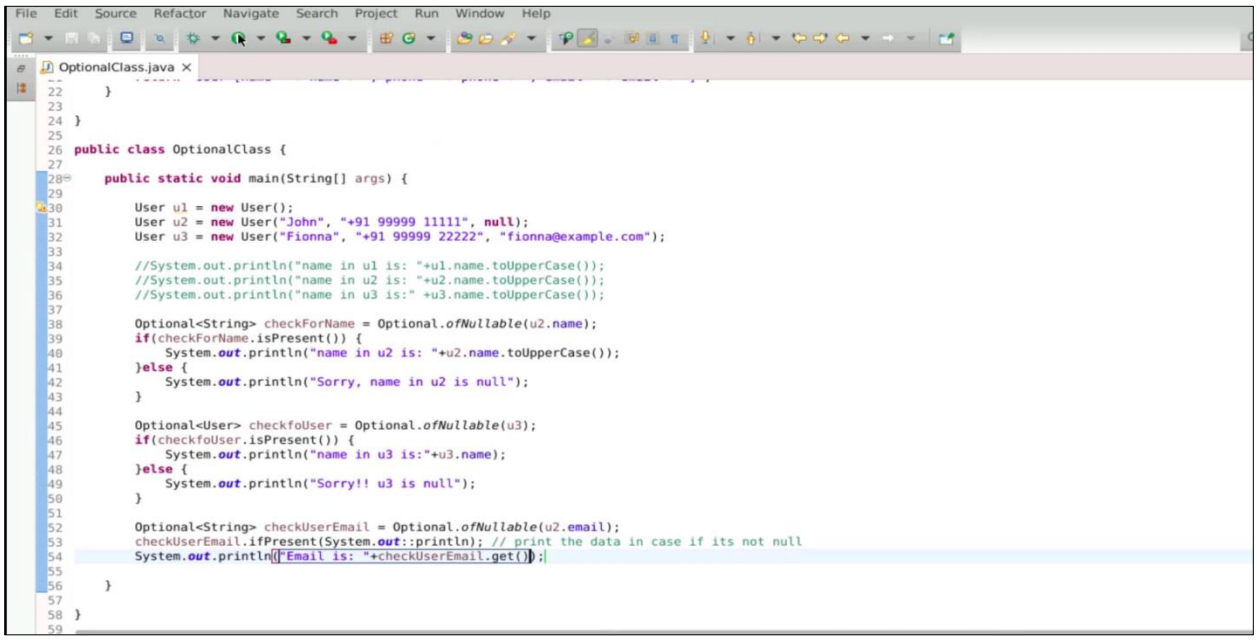
5.9 Run this, and you will see **john@example.com** printed



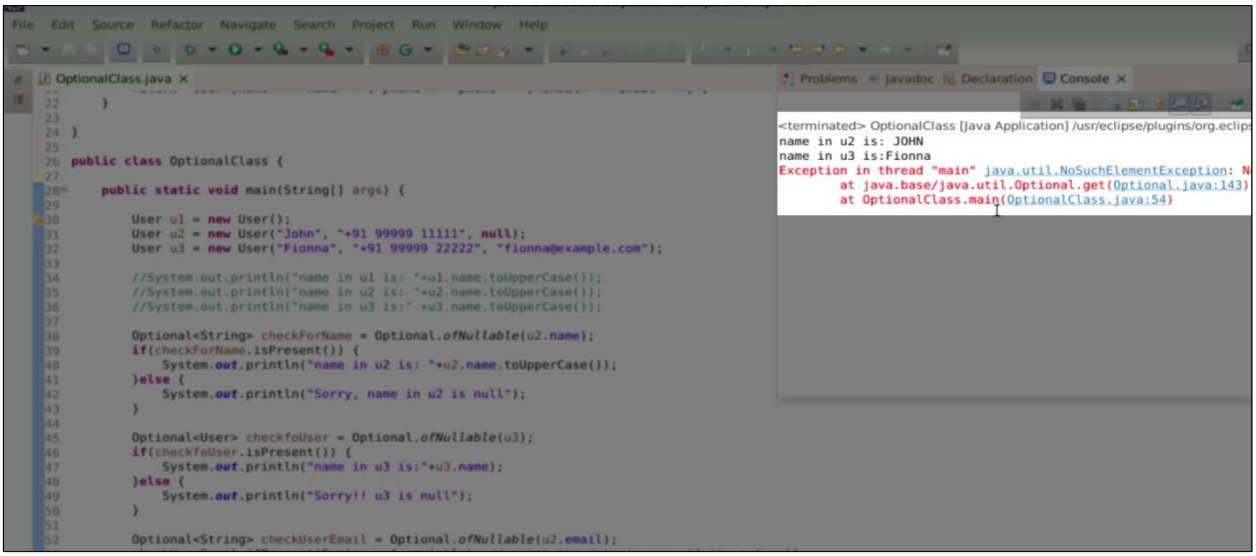
5.10 Let us consider that you have not supplied an email and have given **null** as the email for **u2**. When you run the code, you won't be able to see the email of the user printed. With this simple approach of method referencing you have printed your data



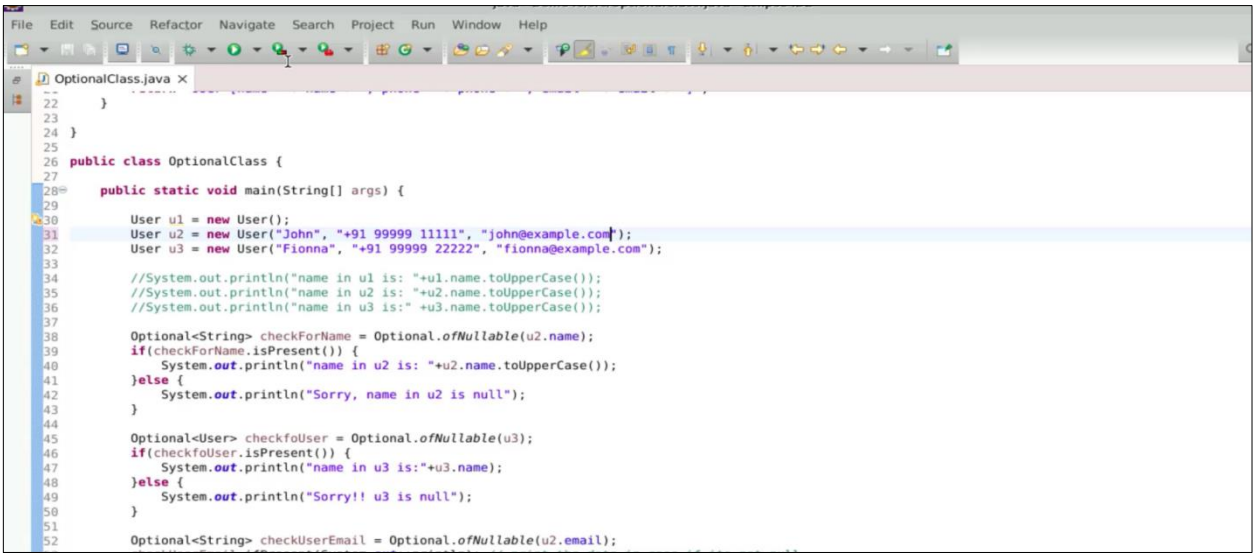
5.11 You can even use another method:
System.out.println("Email is: "+checkUser.email.get()) to extract the email data.
The **get()** method returns the optional email, which may be null or not



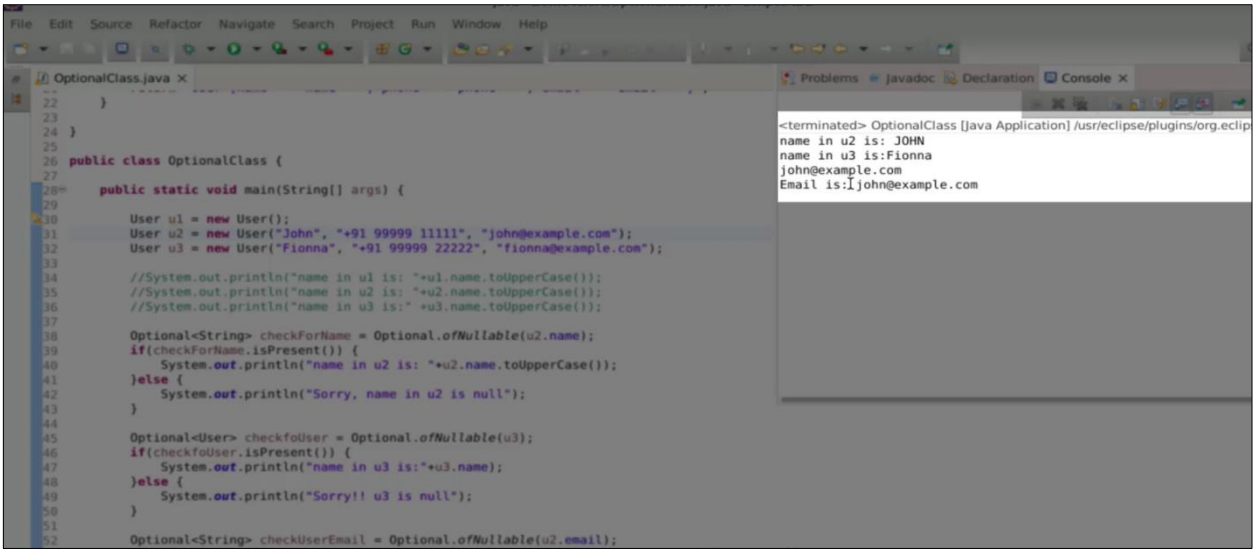
5.12 When you run the code it shows **no value present**



5.13 Add the email: **john@example.com**



5.14 Re-run the code, and you can see that with the **get()** method, you can extract the data. However, if it is null, it will crash



By following these steps, you have successfully implemented the Optional class in Java to handle null values and avoid NullPointerExceptions.