Lesson 04 Demo 07

Implementing Custom Exceptions

Objective: To implement custom exceptions in Java

Tools required: Eclipse IDE

Prerequisites: None

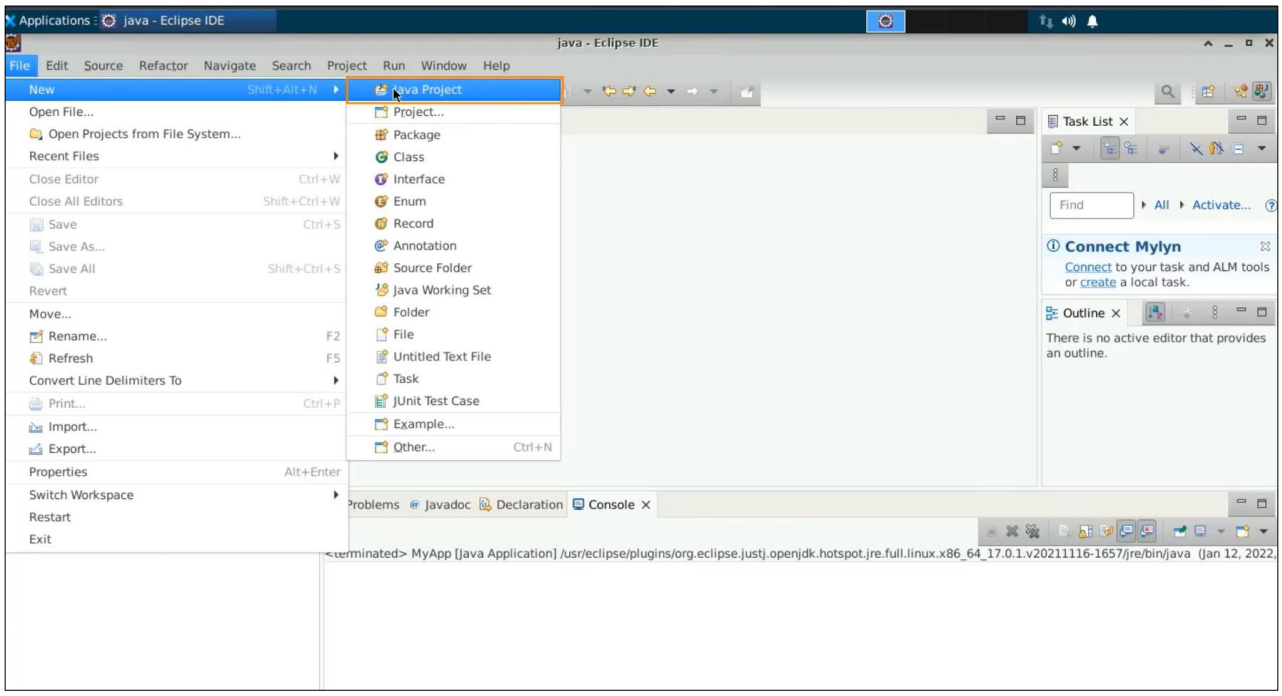
Steps to be followed:

1. Open IDE and create a new project
2. Implement exceptions in Java
3. Create the custom exception objects with example data
4. Compare checked and unchecked exceptions

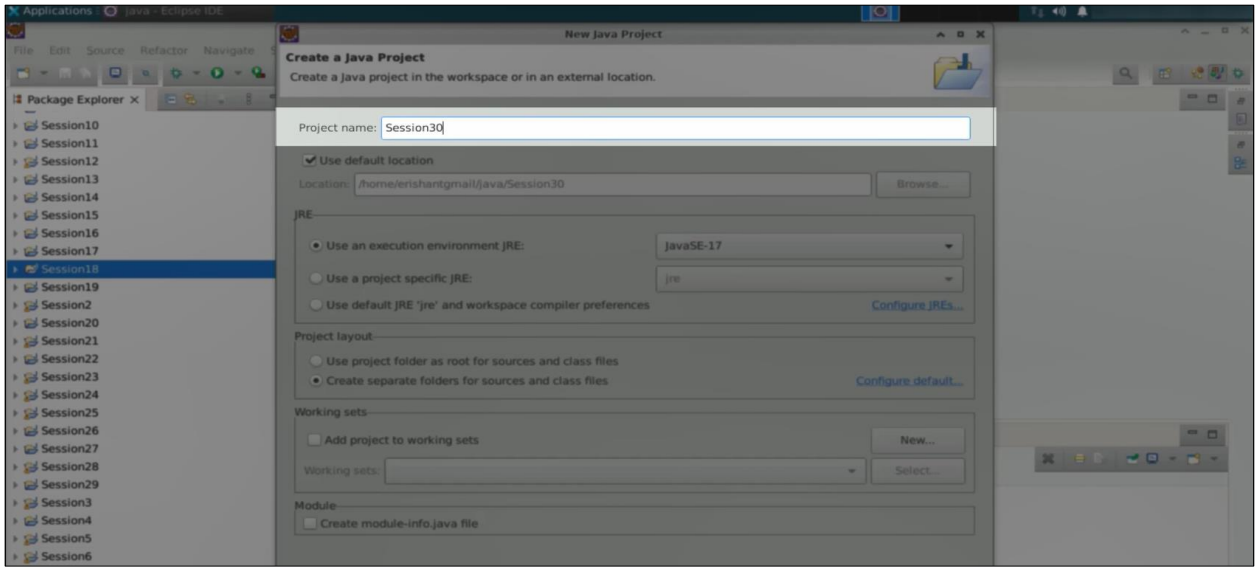
Step 1: Open IDE and create a new project

1.1 Open the **Eclipse IDE**

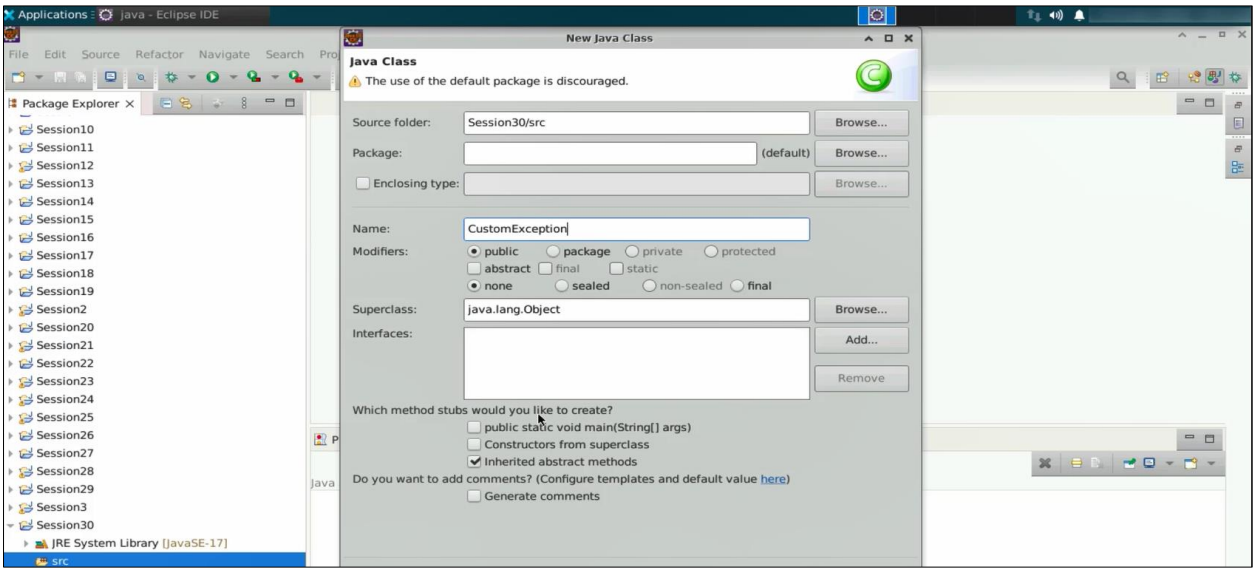
1.2 Select **File**, then **New,** and then **Java project**



1.3 Name the project **“Session30”,** uncheck **“Create a module info dot Java file”**, and press **Finish**

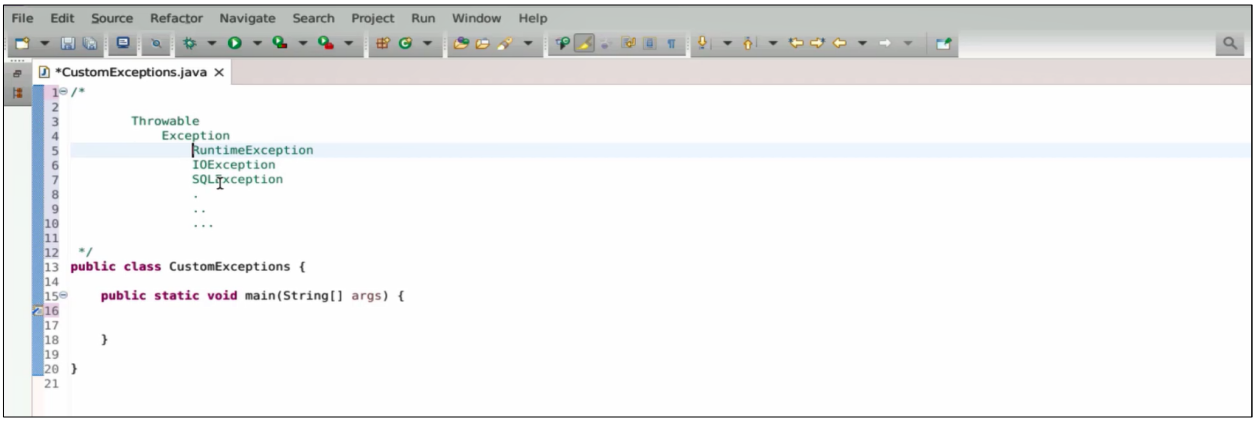


1.4 With a **Session30** on the src, do a right-click and create a **new class**. Name this class as an **CustomException**, then select the **main method,** and then select **finish.**

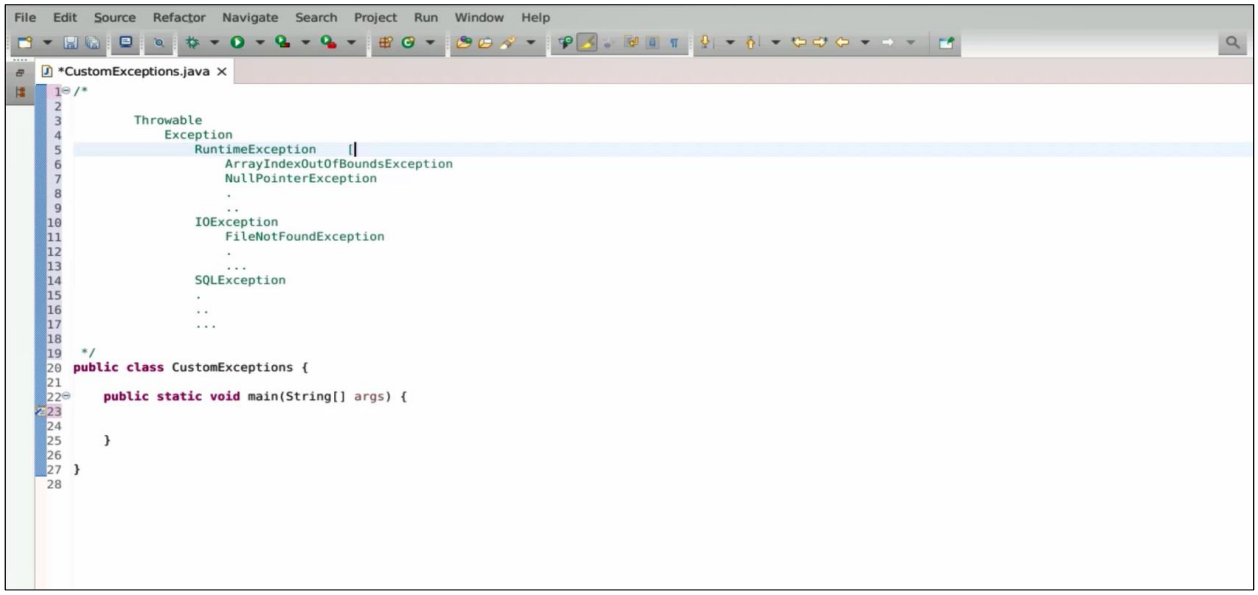


Step 2: Implement exceptions in Java

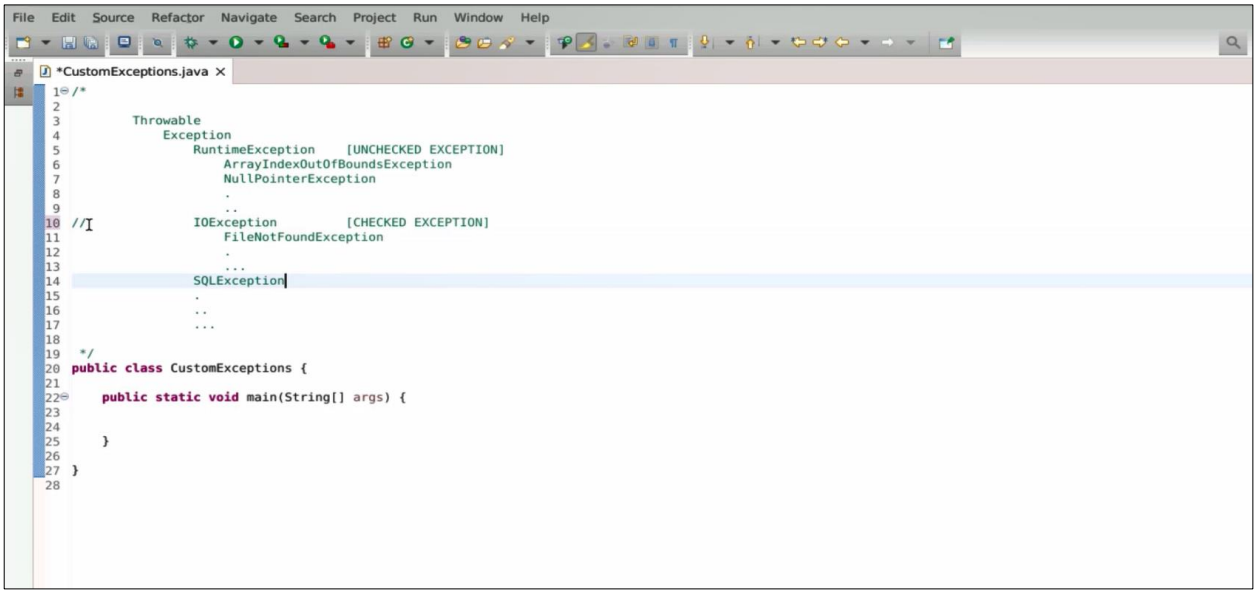
2.1 In Java, by default, all exceptions are children of the Exception class. The hierarchy is as follows: you have the Throwable class, then its child Exception, followed by RuntimeException, IOException, SQLException, and many more. The Exception class is the parent of all exceptions, making them all siblings in this hierarchy.



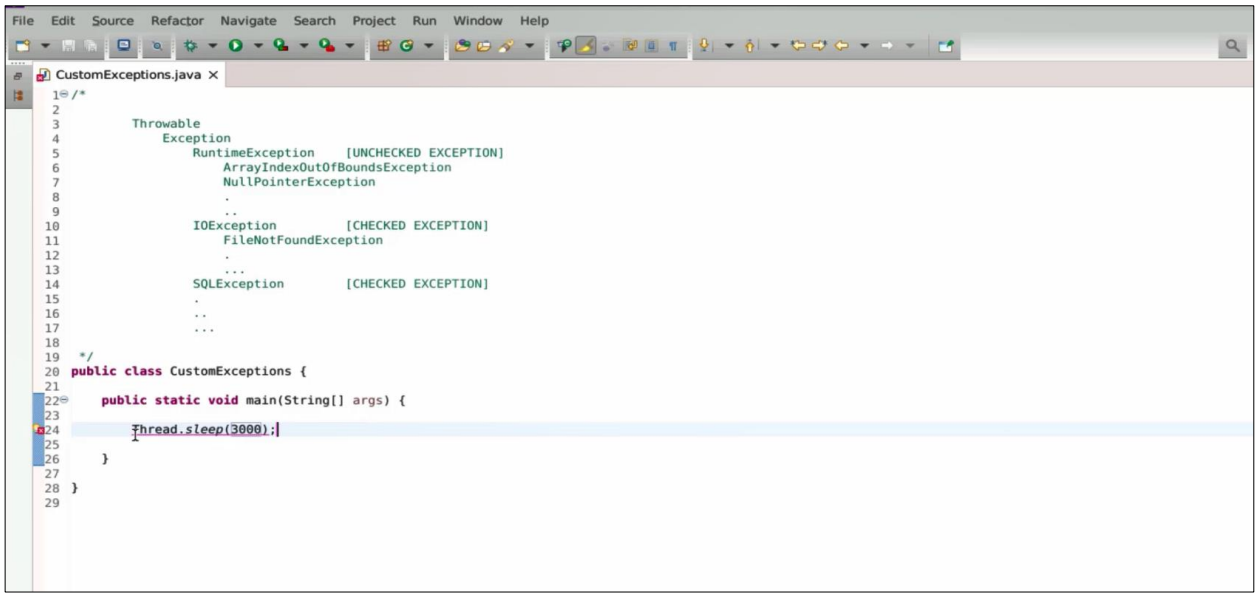
2.2 Now runtime exception is one class that further has the children-like array index out of bounds exception, then you have null pointer exception and many more. Even the IO exception can have a child, for example, file not found exception. It means that all the exceptions can have further children.



2.3 You need to understand in Java that runtime exceptions are referred to as unchecked exceptions, and IO exceptions are referred to as checked exceptions. That is, other than the runtime exception, all the exceptions are referred to as Checked exceptions.

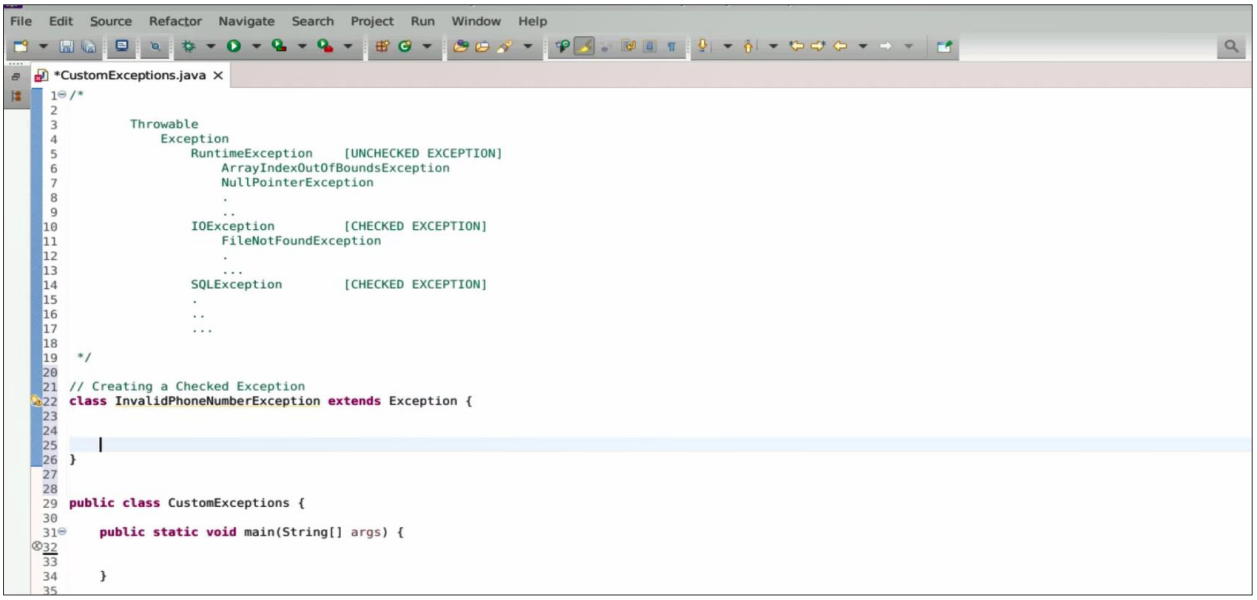


2.4 In Java, runtime exceptions are called unchecked exceptions, while IO exceptions are checked exceptions. Other than runtime exceptions, all are checked exceptions. For example, using Thread.sleep() causes a compile-time error showing an unhandled InterruptedException, requiring the code to be surrounded with try-catch. This shows a checked exception.

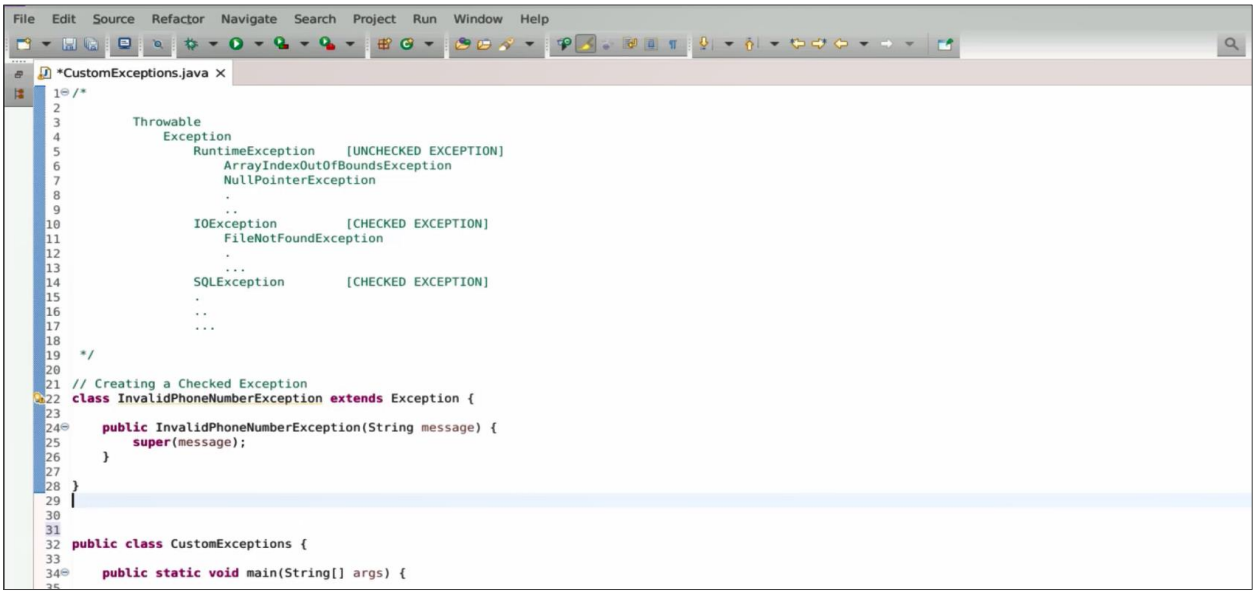


Step 3: Create the custom exception objects with example data

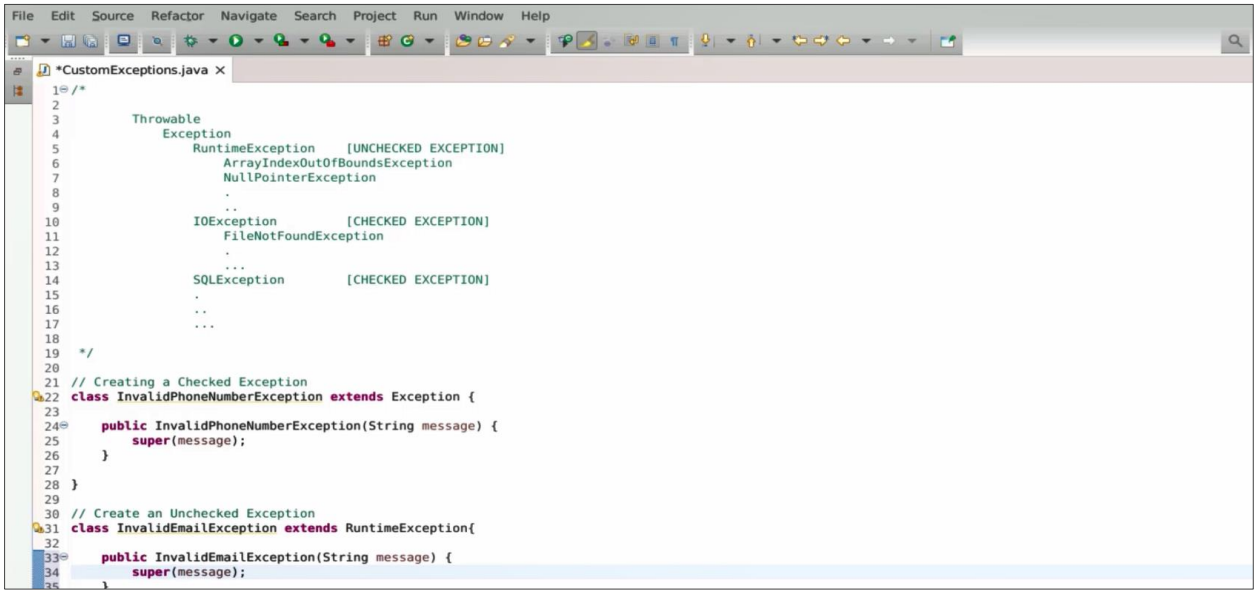
3.1 Moving ahead, your goal is to create the custom exception objects, you can create the custom exception objects like this, first, create a class called Invalid Phone number exception, which is the child of Exception, when you say that you will create a class with the inheritance, you are trying to create a relationship, the exception is the parent, and invalid phone number exception is the child.



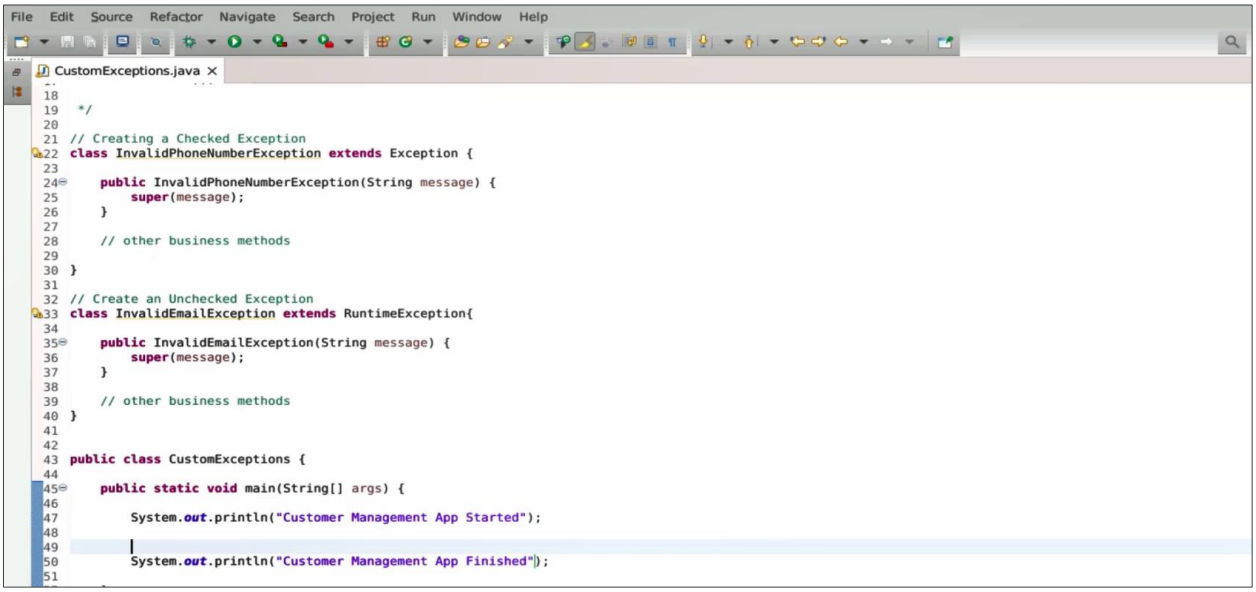
3.2 With your invalid phone number, you can create a constructor, which can take some message as input, and after that you can pass this message to the parent, that is, over here you will be passing this message. In the same way, you can create an unchecked exception.



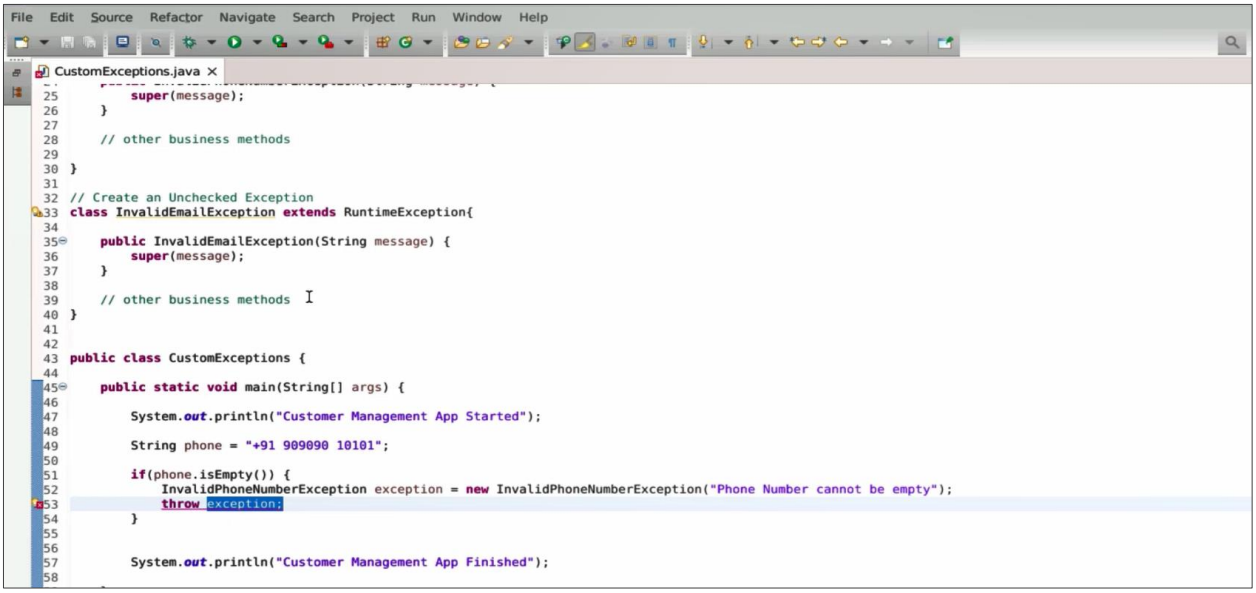
3.3 In order to create an unchecked exception, you need to create a class, for example, Invalid email exception which is the child of runtime exception. Now any child of this runtime exception class is known as an unchecked exception, hence the compiler will not prompt you to surround your code with try-catch if this exception is thrown, the rest structure remains the same. You got this message here and you can pass this message to the parent.



3.4 If you want you can even write other business methods, which is as per your logic. Next, print customer management App started, and then you can come here and write another statement which says customer management app Finished.

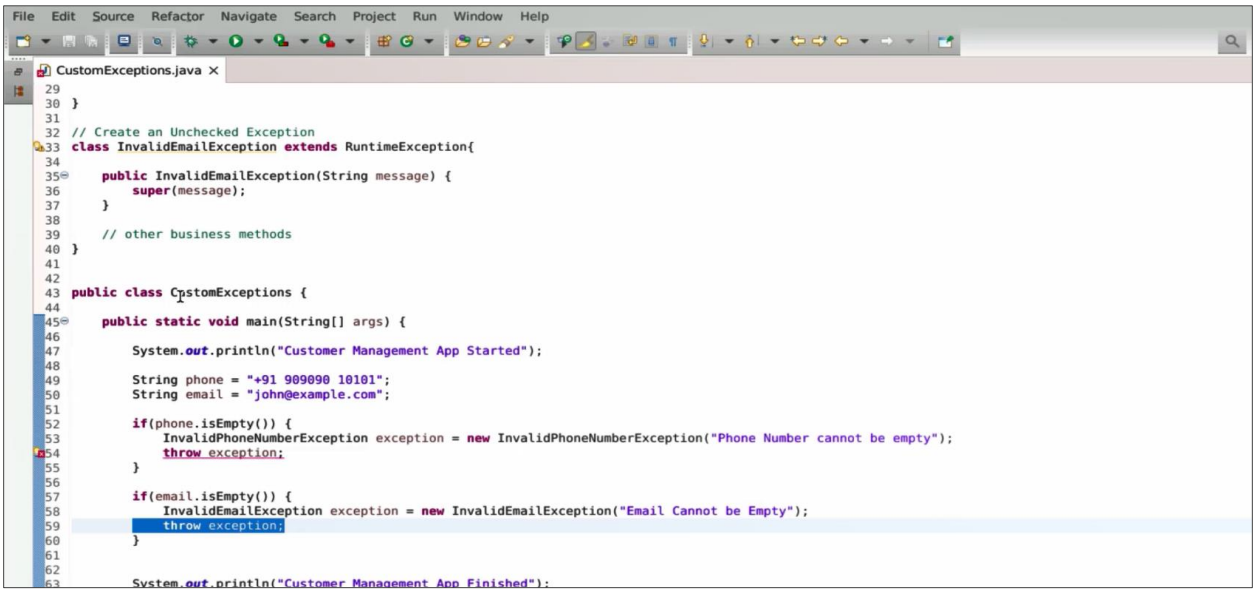


3.5 You can take one phone number like + 9 1 and some number. Write, If the phone is empty, if it's blank, you can create the object of this class called Invalid Phone number exception. So, let us see how, write invalid phone number exception, the exception is a new invalid Phone number exception and the message is phone number cannot be empty, this is the message which you have passed, and then you can use throw the exception.

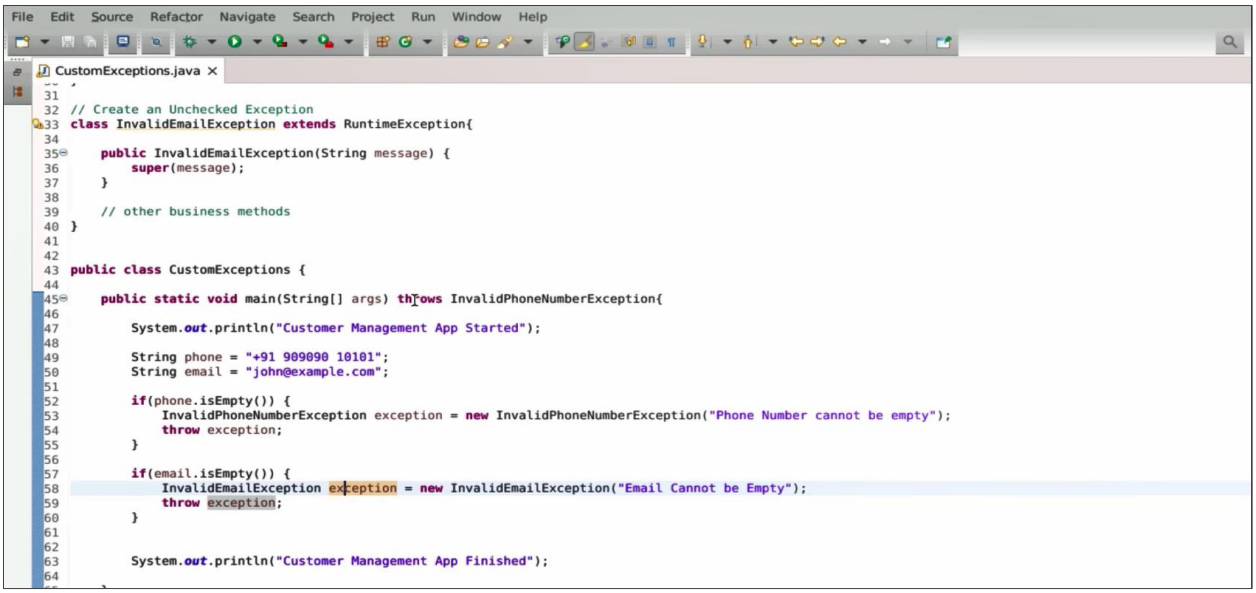


Step 4: Compare checked and unchecked exceptions

4.1 Let us use another concept called email, such as john@example.com. For the email, write an if statement to check if email.isEmpty(). If it is empty, create an InvalidEmailException object with the message "Email cannot be empty." You can have several other checks on the email. When you throw this exception, you will notice that for an unchecked exception, the compiler does not give an error message. However, for a checked exception, the compiler does give an error message.



4.2 When creating your custom exception, note that the compiler checks occur with checked exceptions. To demonstrate, create the main method and add the signature throws InvalidPhoneNumberException. You will see that no error occurs.



By following the above steps, you have successfully implemented custom exceptions in Java, thereby enhancing your application's clarity and robustness through tailored error handling.