Lesson 03 Demo 05

Implementing Overloading and Overriding

Objective: Differentiating and implementing the concept of Overloading and Overriding in Java

Tools required: Eclipse IDE

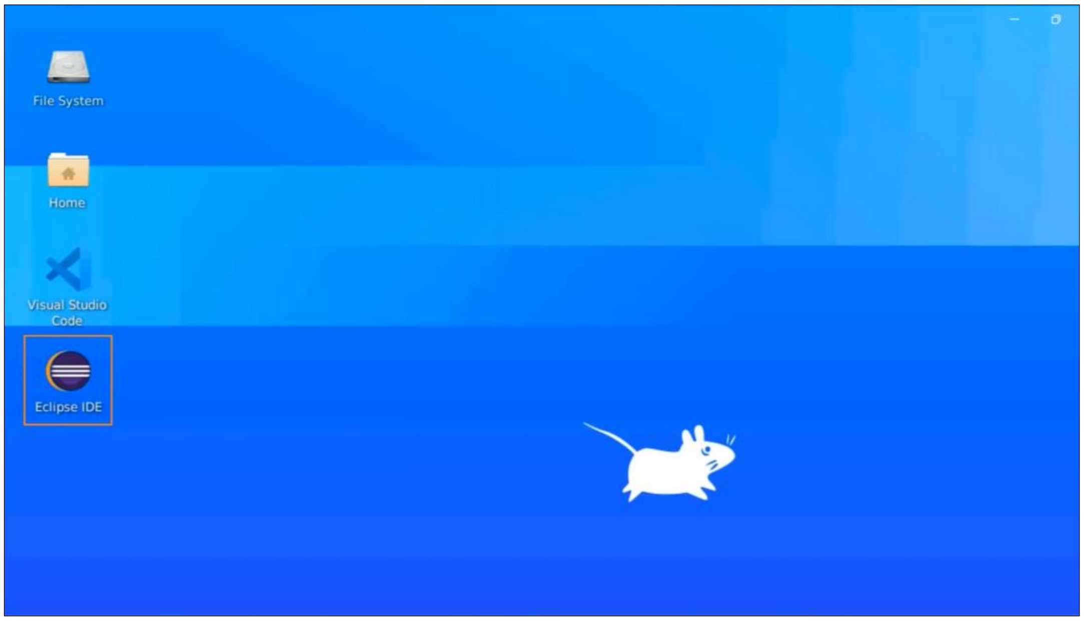
Prerequisites: None

Steps to be followed:

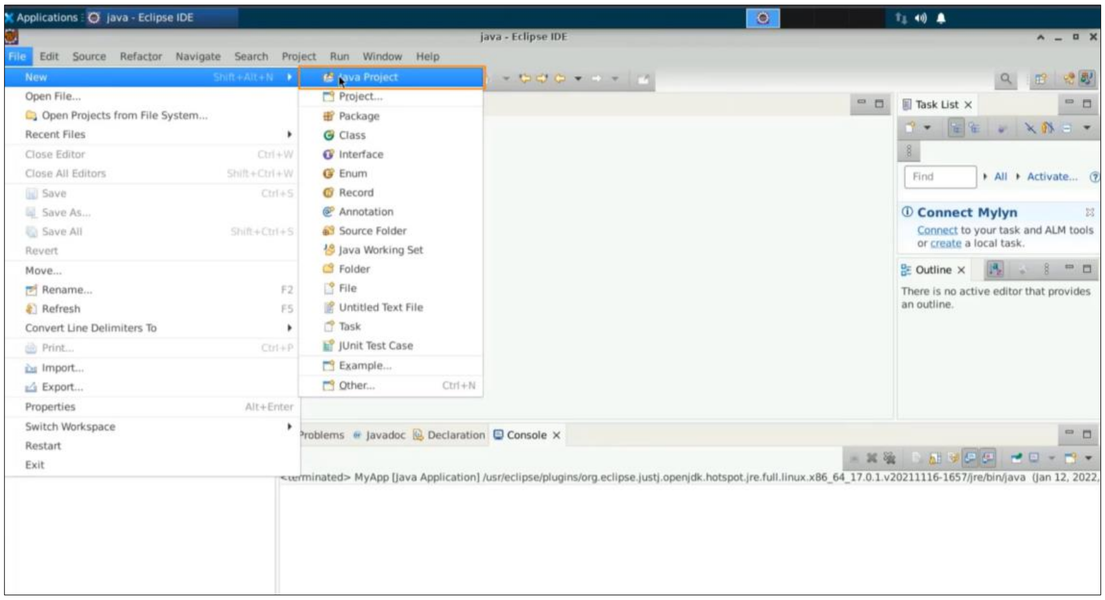
1. Create a class called Overloading Vs Overriding, followed by selecting the main method
2. Use the class called authentication with a login method
3. Overload the method with unique inputs
4. Use example scenarios for overloading and overriding

Step 1: Create a class called Overloading Vs Overriding, followed by selecting the main method

1.1 Open the **Eclipse IDE**

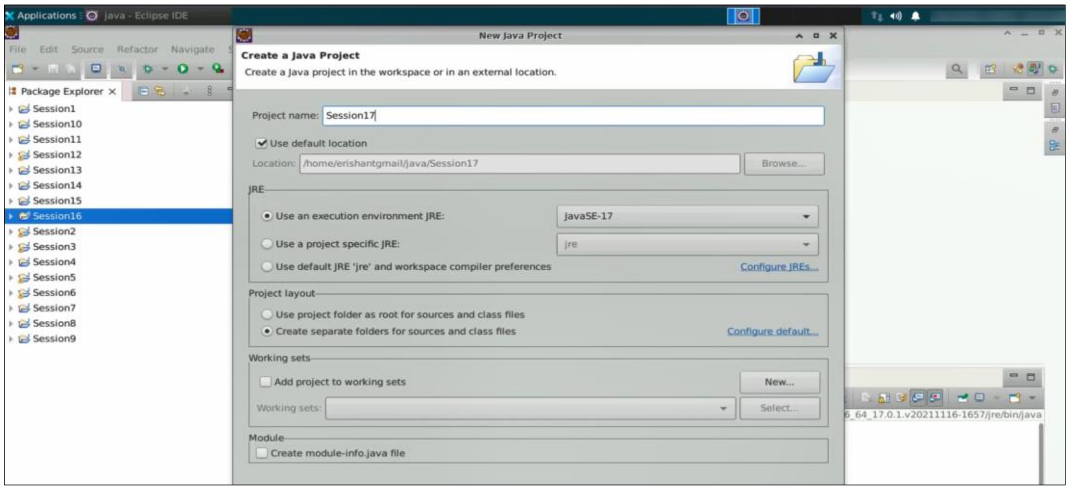


1.2. Select **File**, then **New,** and then **Java project**



```
<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 **“Session17”,** uncheck **“Create a module info dot Java file”**, and press **Finish**



Project name: Session17

☑ Use default location

Location: /home/erishantgmail/java/Session17

JRE

● Use an execution environment JRE: JavaSE-17

○ Use a project specific JRE: jre

○ Use default JRE 'jre' and workspace compiler preferences

Project layout

○ Use project folder as root for sources and class files

● Create separate folders for sources and class files

Working sets

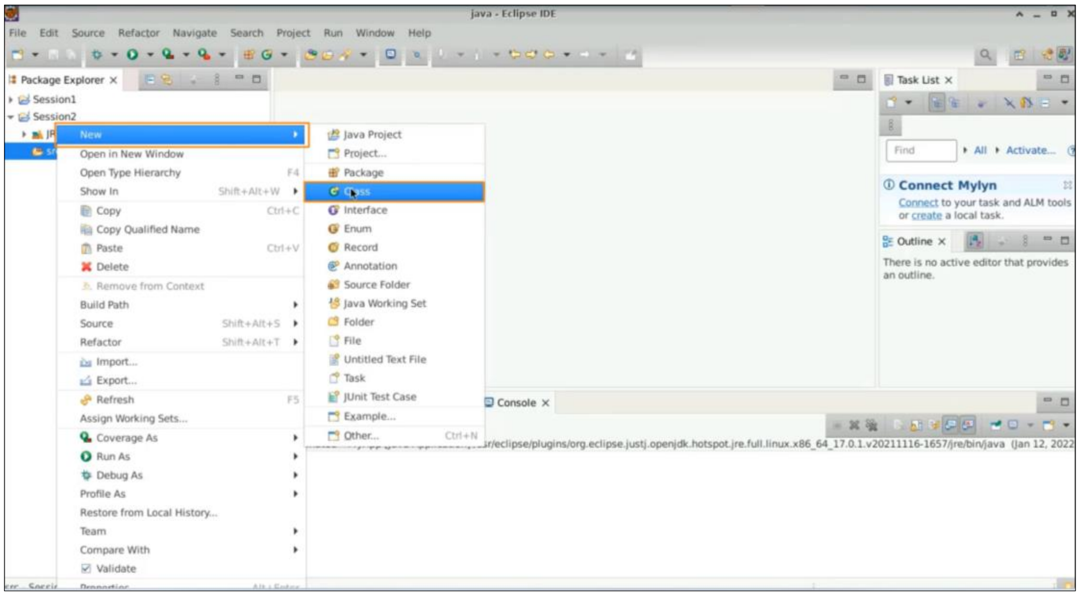
☐ Add project to working sets

Working sets:

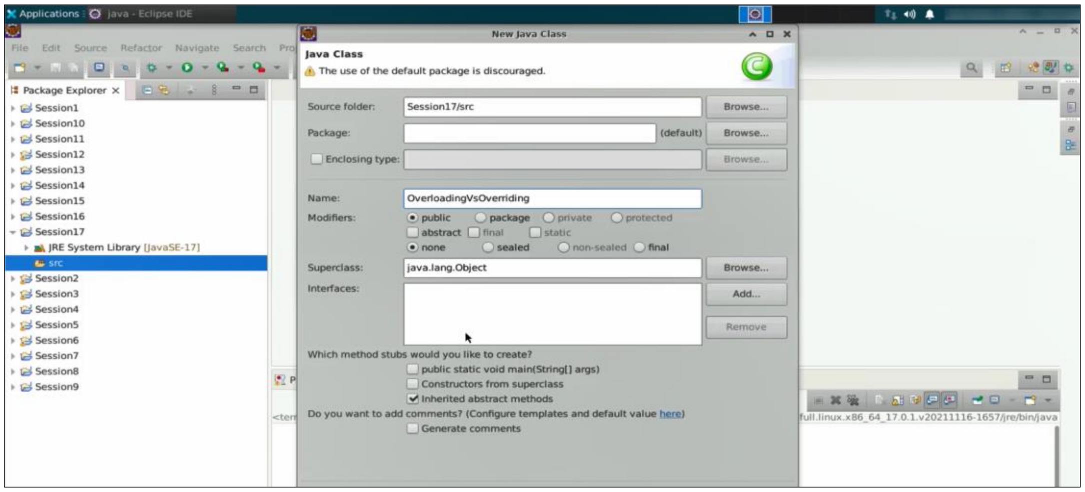
Module

☐ Create module-info.java file

1.4 With a **Session17** on the src, do a right-click and create a **new class**



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



The use of the default package is discouraged.

Source folder: Session17/src

Package: (default)

Enclosing type:

Name: OverloadingVsOverriding

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

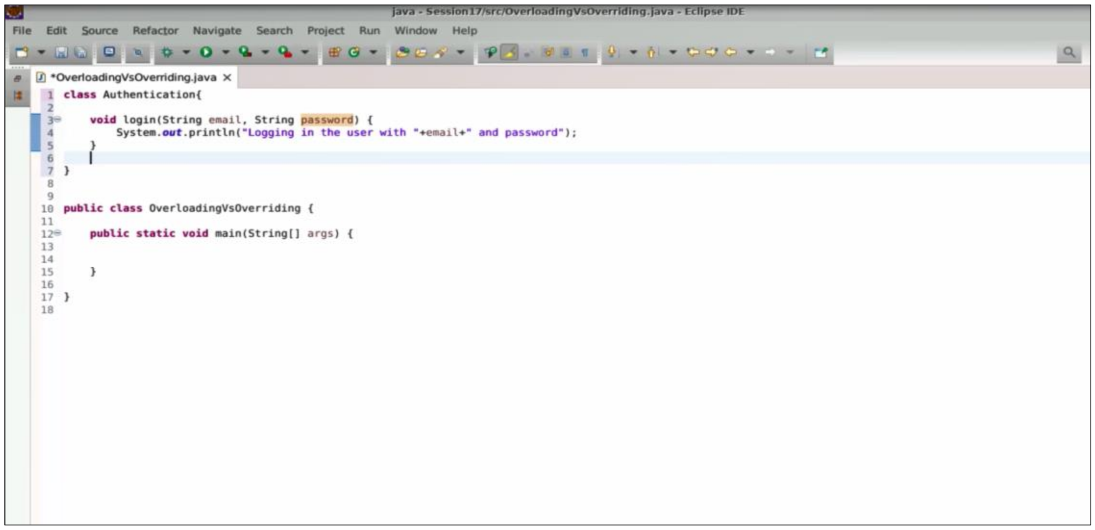
[✓ Inherited abstract methods]

Do you want to add comments? (Configure templates and default value here)

Generate comments

Step 2: Use the class called authentication with a login method

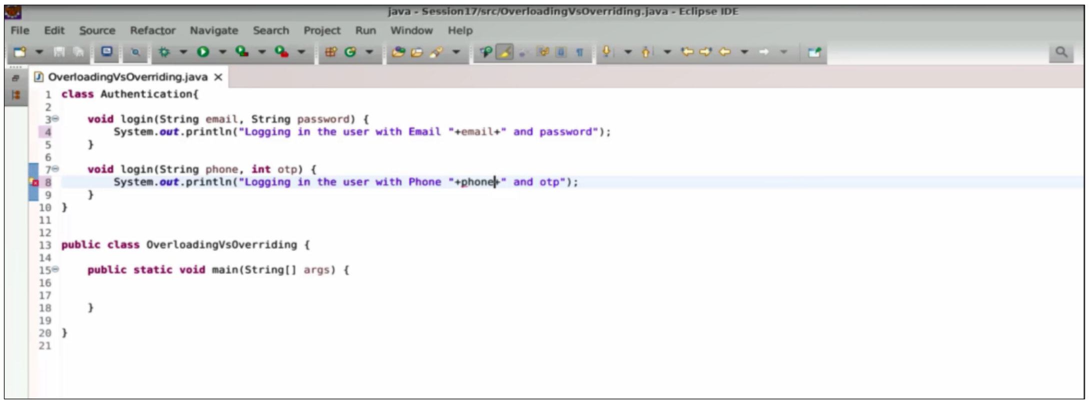
2.1 When it comes to overloading, you will have the same class that can contain methods with different signatures. For example, there is a class called Authentication. In this class, you have a login method that can take an email and a password as input. Next, write **Logging in the user with email and password**. You are not going to print the password; you will just mention it here


java - Session17/src/OverloadingVsOverriding.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*OverloadingVsOverriding.java

```java
class Authentication{

    void login(String email, String password) {
        System.out.println("Logging in the user with "+email+" and password");
    }
    
}


public class OverloadingVsOverriding {

    public static void main(String[] args) {


    }
}
```

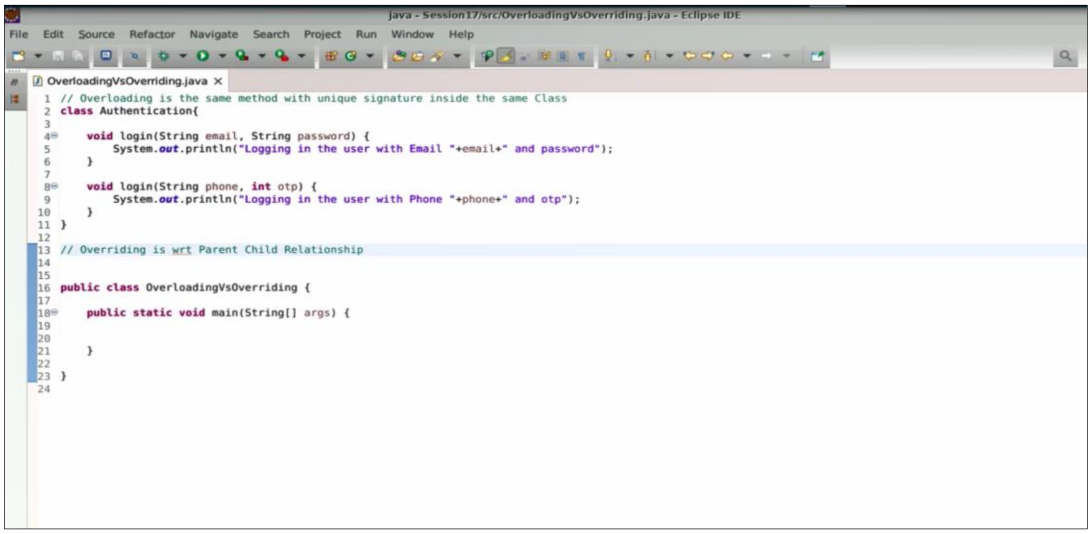
Step 3: Overload the method with unique inputs

3.1 If you want, you can overload this method, which means you are going to have the same method name again. But you need to make your inputs to go as unique. Hence logging in the user with phone and otp


java - Session17/src/OverloadingVsOverriding.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
OverloadingVsOverriding.java

```java
class Authentication{

    void login(String email, String password) {
        System.out.println("Logging in the user with Email "+email+" and password");
    }

    void login(String phone, int otp) {
        System.out.println("Logging in the user with Phone "+phone+" and otp");
    }
}


public class OverloadingVsOverriding {

    public static void main(String[] args) {


    }
}
```

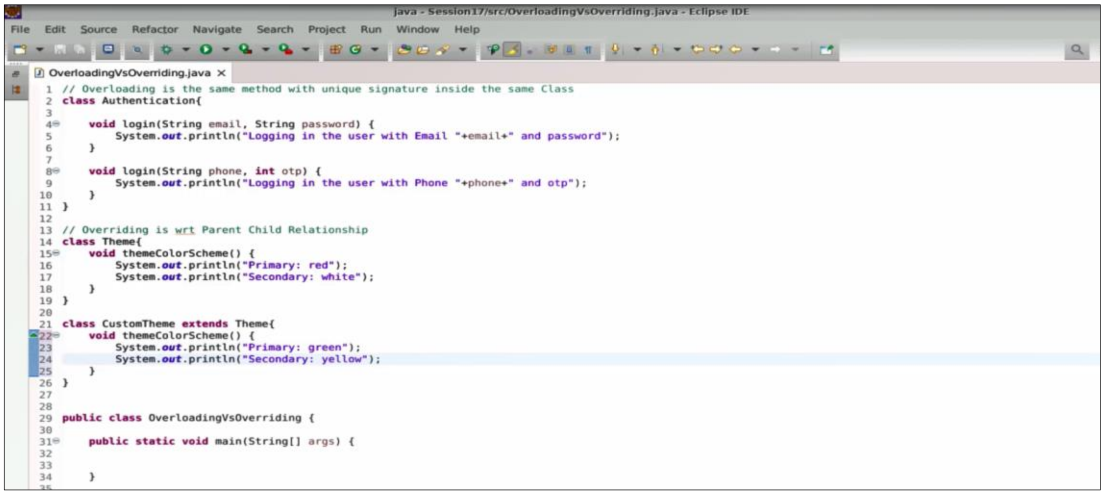
3.2 When it comes to overloading, you have the same method name which is repeatedly used inside the same class, which is the primary rule. Overloading is the same method with a unique signature, inside the same class. When it comes to the overriding, it is with respect to parent-child relationship


java - Session17/src/OverloadingVsOverriding.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
OverloadingVsOverriding.java

```java
// Overloading is the same method with unique signature inside the same Class
class Authentication{

    void login(String email, String password) {
        System.out.println("Logging in the user with Email "+email+" and password");
    }

    void login(String phone, int otp) {
        System.out.println("Logging in the user with Phone "+phone+" and otp");
    }
}

// Overriding is wrt Parent Child Relationship


public class OverloadingVsOverriding {

    public static void main(String[] args) {


    }
}
```

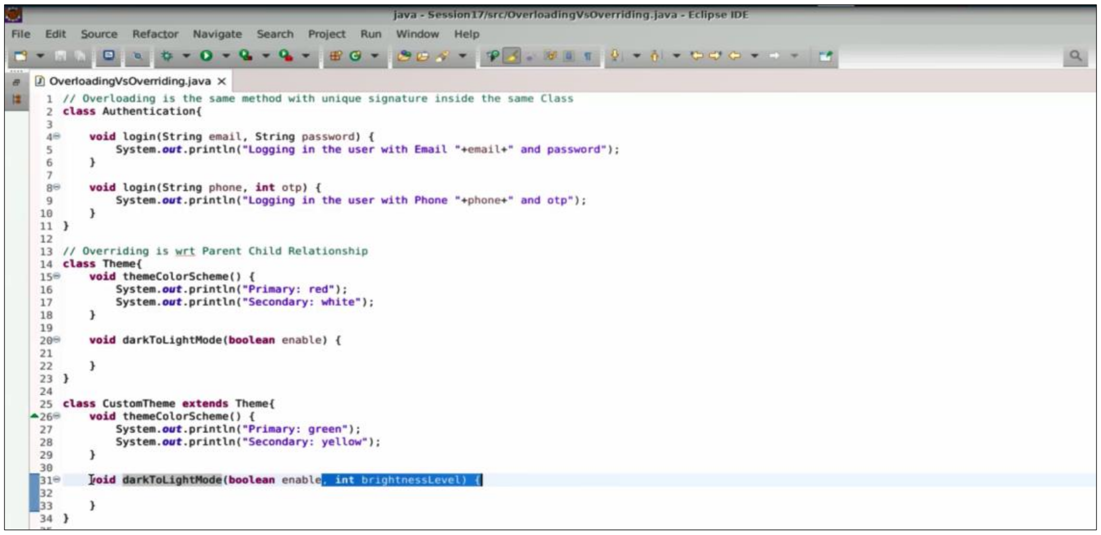
Step 4: Use Example scenarios for overloading and overriding

4.1 There is a parent class called Theme and a child class called **CustomTheme**. For overriding, let us write a method called void **themeColorScheme**. In the **themeColorScheme**, you can specify that the primary color is red and the secondary color is white. The same method with the same signature has to be defined in the child class. Here, the definition can be different, such as setting the primary color to green and the secondary color to yellow


java - Session17/src/OverloadingVsOverriding.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
OverloadingVsOverriding.java

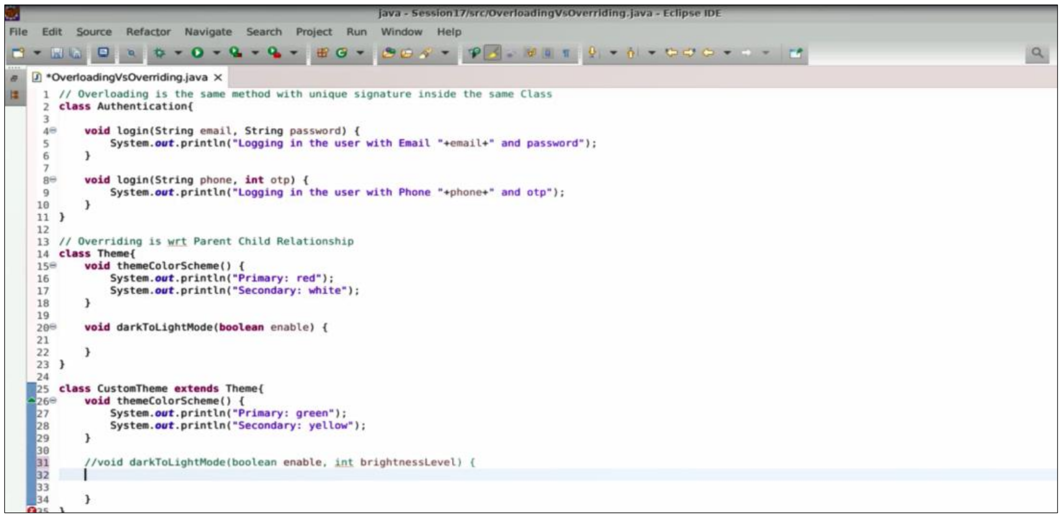
```java
// Overloading is the same method with unique signature inside the same Class
class Authentication{

    void login(String email, String password) {
        System.out.println("Logging in the user with Email "+email+" and password");
    }

    void login(String phone, int otp) {
        System.out.println("Logging in the user with Phone "+phone+" and otp");
    }
}

// Overriding is wrt Parent Child Relationship
class Theme{
    void themeColorScheme() {
        System.out.println("Primary: red");
        System.out.println("Secondary: white");
    }
}

class CustomTheme extends Theme{
    void themeColorScheme() {
        System.out.println("Primary: green");
        System.out.println("Secondary: yellow");
    }
}


public class OverloadingVsOverriding {

    public static void main(String[] args) {


    }
```

4.2 Let us take up one more example. There is this theme that says dark to light mode. It takes one input, Boolean enabled. Create the method in the child, but this method takes another input, hence Boolean enables integer brightness level


java - Session17/src/OverloadingVsOverriding.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
OverloadingVsOverriding.java

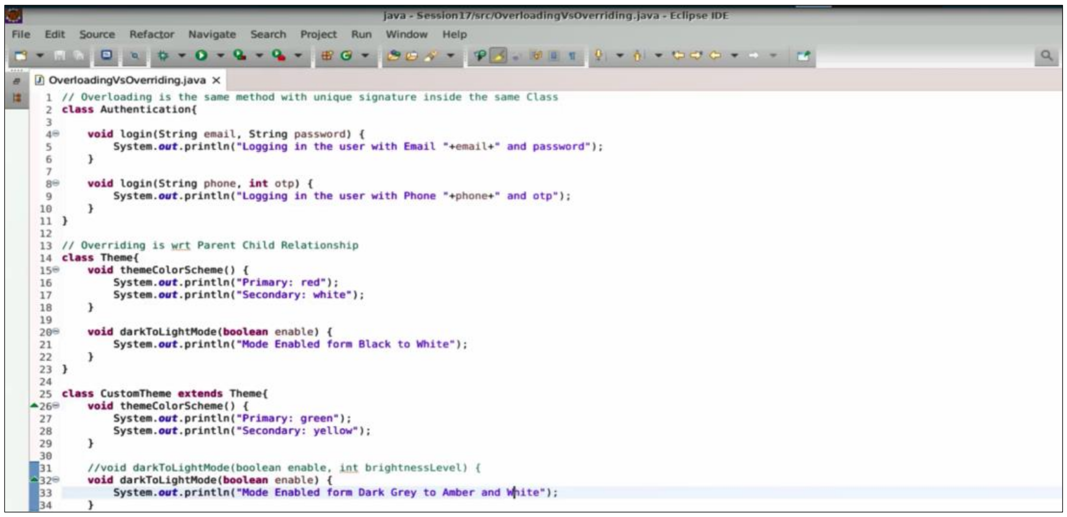
```java
// Overloading is the same method with unique signature inside the same Class
class Authentication{

    void login(String email, String password) {
        System.out.println("Logging in the user with Email "+email+" and password");
    }

    void login(String phone, int otp) {
        System.out.println("Logging in the user with Phone "+phone+" and otp");
    }
}

// Overriding is wrt Parent Child Relationship
class Theme{
    void themeColorScheme() {
        System.out.println("Primary: red");
        System.out.println("Secondary: white");
    }

    void darkToLightMode(boolean enable) {

    }
}

class CustomTheme extends Theme{
    void themeColorScheme() {
        System.out.println("Primary: green");
        System.out.println("Secondary: yellow");
    }

    void darkToLightMode(boolean enable, int brightnessLevel) {

    }
}
```

4.3 This is not overriding. Overriding means that the method name and signature must be the same in both the parent and child classes. For overriding and overloading, you will have different scenarios


java - Session17/src/OverloadingVsOverriding.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*OverloadingVsOverriding.java

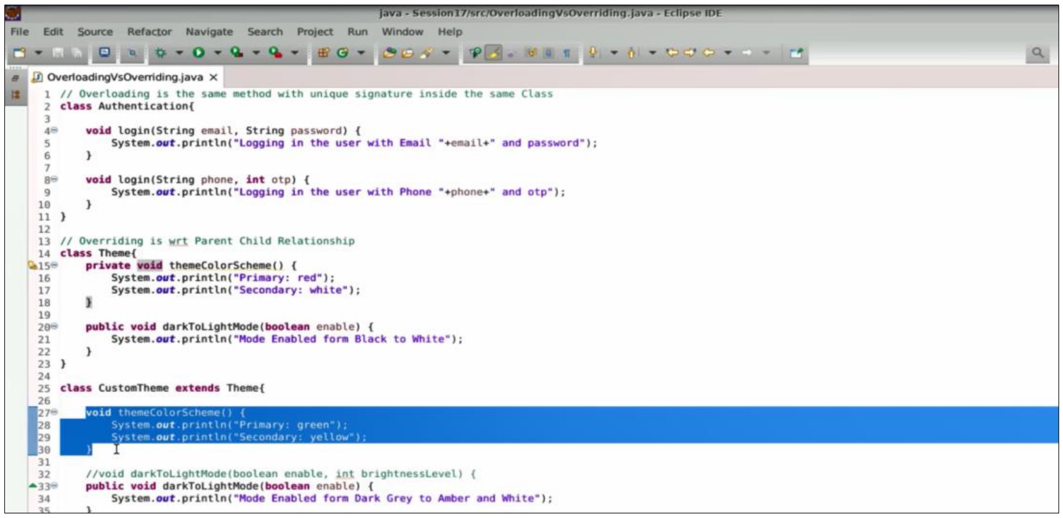
```java
// Overloading is the same method with unique signature inside the same Class
class Authentication{

    void login(String email, String password) {
        System.out.println("Logging in the user with Email "+email+" and password");
    }

    void login(String phone, int otp) {
        System.out.println("Logging in the user with Phone "+phone+" and otp");
    }
}

// Overriding is wrt Parent Child Relationship
class Theme{
    void themeColorScheme() {
        System.out.println("Primary: red");
        System.out.println("Secondary: white");
    }

    void darkToLightMode(boolean enable) {

    }
}

class CustomTheme extends Theme{
    void themeColorScheme() {
        System.out.println("Primary: green");
        System.out.println("Secondary: yellow");
    }

    //void darkToLightMode(boolean enable, int brightnessLevel) {



    }
}
```

4.4 If you want to enable dark to light mode in the parent class, you can specify the mode as enabled from black to white. However, in the child class, the mode can be enabled from dark gray to amber and white. The definitions can be different, but the signature must be exactly the same


java - Session17/src/OverloadingVsOverriding.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
OverloadingVsOverriding.java

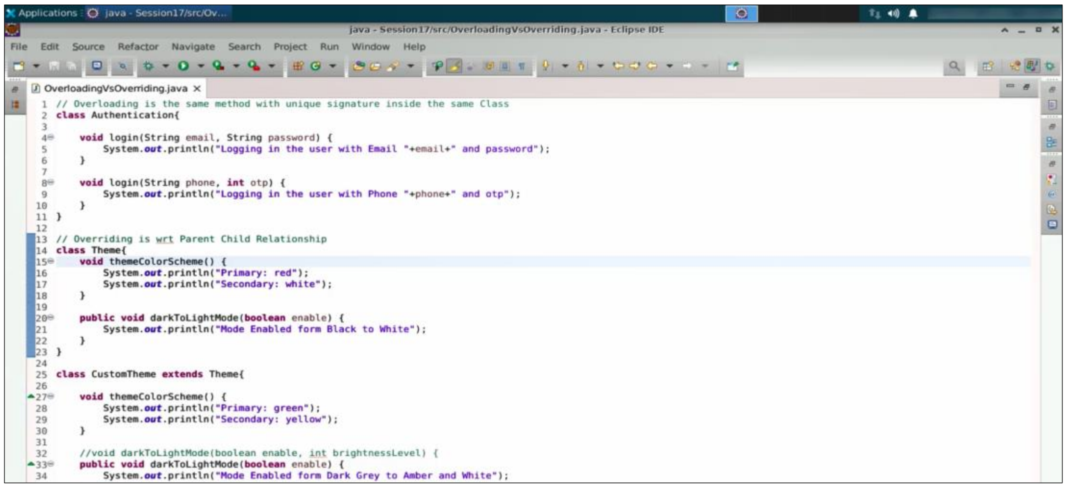
```java
// Overloading is the same method with unique signature inside the same Class
class Authentication{

    void login(String email, String password) {
        System.out.println("Logging in the user with Email "+email+" and password");
    }

    void login(String phone, int otp) {
        System.out.println("Logging in the user with Phone "+phone+" and otp");
    }
}

// Overriding is wrt Parent Child Relationship
class Theme{
    void themeColorScheme() {
        System.out.println("Primary: red");
        System.out.println("Secondary: white");
    }

    void darkToLightMode(boolean enable) {
        System.out.println("Mode Enabled form Black to White");
    }
}

class CustomTheme extends Theme{
    void themeColorScheme() {
        System.out.println("Primary: green");
        System.out.println("Secondary: yellow");
    }

    //void darkToLightMode(boolean enable, int brightnessLevel) {
    void darkToLightMode(boolean enable) {
        System.out.println("Mode Enabled form Dark Grey to Amber and White");
    }
```

4.5 If the parent method is marked as public, the child can override it. However, overriding comes with some constraints. If you want to override a public method, the access level in the child class needs to be the same or higher. When you make it public, you will be able to override it. If you mark your method as private in the parent, defining the theme color scheme in the child is not considered overriding because private methods are limited only to the parent class


java - Session17/src/OverloadingVsOverriding.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
OverloadingVsOverriding.java

```java
// Overloading is the same method with unique signature inside the same Class
class Authentication{

    void login(String email, String password) {
        System.out.println("Logging in the user with Email "+email+" and password");
    }

    void login(String phone, int otp) {
        System.out.println("Logging in the user with Phone "+phone+" and otp");
    }
}

// Overriding is wrt Parent Child Relationship
class Theme{
    private void themeColorScheme() {
        System.out.println("Primary: red");
        System.out.println("Secondary: white");
    }

    public void darkToLightMode(boolean enable) {
        System.out.println("Mode Enabled form Black to White");
    }
}

class CustomTheme extends Theme{

    void themeColorScheme() {
        System.out.println("Primary: green");
        System.out.println("Secondary: yellow");
    }

    //void darkToLightMode(boolean enable, int brightnessLevel) {
    public void darkToLightMode(boolean enable) {
        System.out.println("Mode Enabled form Dark Grey to Amber and White");
    }
```

4.6 Overloading and overriding are two different concepts. Overloading is linked with having the same method name with different signatures in the same class, whereas overriding involves different classes, such as a parent-child relationship, with the same method signature. One rule for overriding is that the access level of the child method should be the same as or higher than that of the parent; only then can overriding be done


Applications
java - Session17/src/Ov...
java - Session17/src/OverloadingVsOverriding.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
OverloadingVsOverriding.java

```java
// Overloading is the same method with unique signature inside the same Class
class Authentication{

    void login(String email, String password) {
        System.out.println("Logging in the user with Email "+email+" and password");
    }

    void login(String phone, int otp) {
        System.out.println("Logging in the user with Phone "+phone+" and otp");
    }
}

// Overriding is wrt Parent Child Relationship
class Theme{
    void themeColorScheme() {
        System.out.println("Primary: red");
        System.out.println("Secondary: white");
    }

    public void darkToLightMode(boolean enable) {
        System.out.println("Mode Enabled form Black to White");
    }
}

class CustomTheme extends Theme{

    void themeColorScheme() {
        System.out.println("Primary: green");
        System.out.println("Secondary: yellow");
    }

    //void darkToLightMode(boolean enable, int brightnessLevel) {
    public void darkToLightMode(boolean enable) {
        System.out.println("Mode Enabled form Dark Grey to Amber and White");
```

By following the above steps, you have successfully differentiated and implemented the concepts of overloading and overriding in Java.