Lesson 04 Demo 01

Creating Interfaces and Multiple Implementation

Objective: To create multiple Interfaces

Tools required: Eclipse IDE

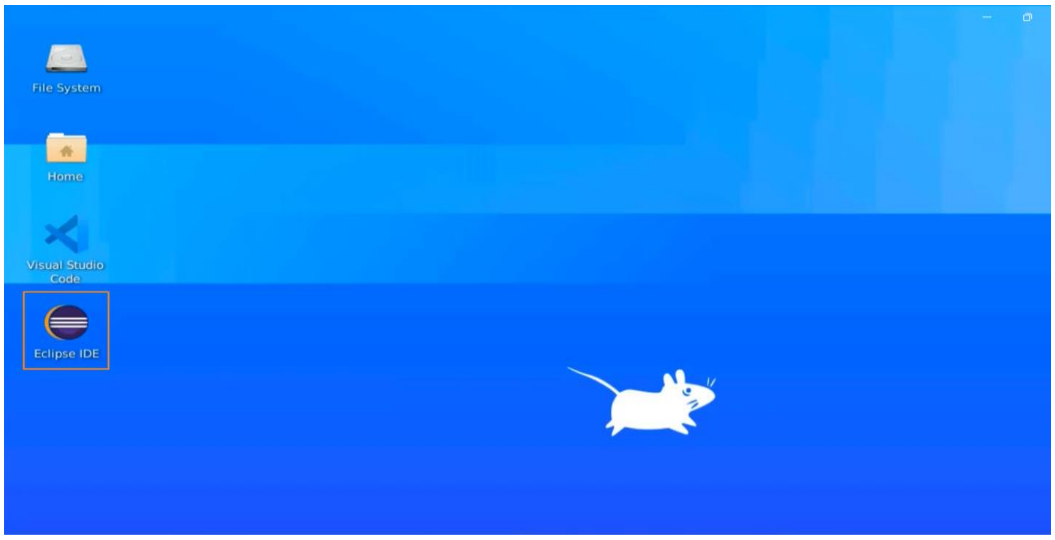
Prerequisites: None

Steps to be followed:

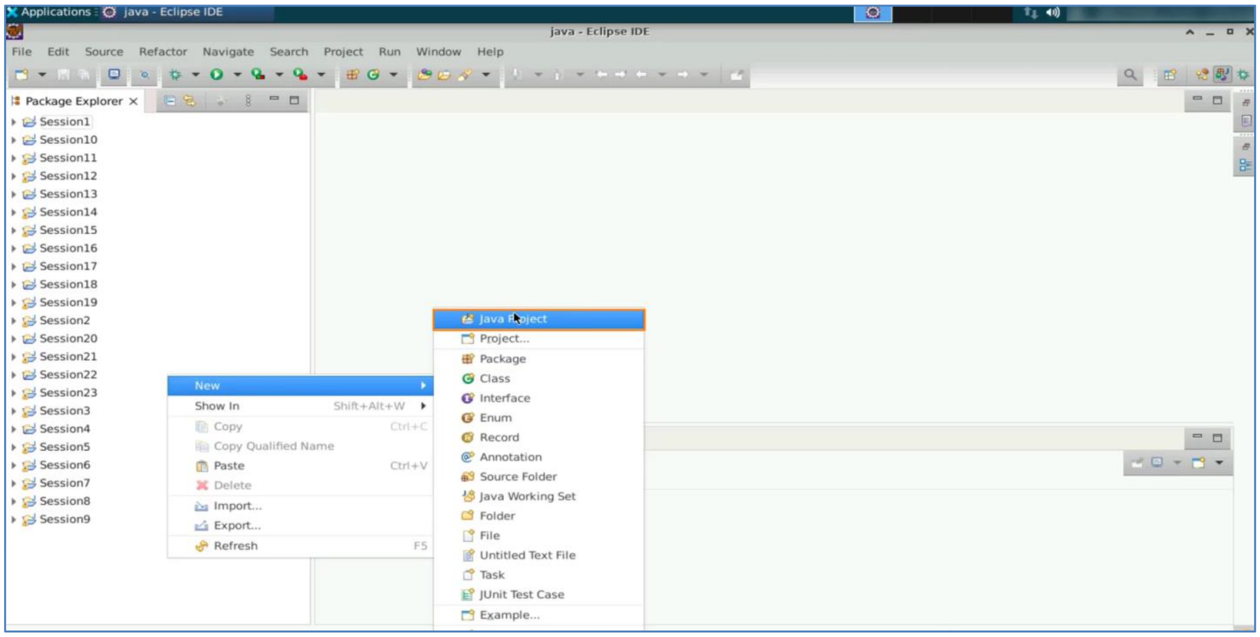
1. Implement the concept of interfaces
2. comprehend how interfaces work
3. Write polymorphic statements using interfaces
4. Implement how multiple inheritances work in interfaces
5. Link interfaces to classes and methods

Step 1: Implement the concept of interfaces

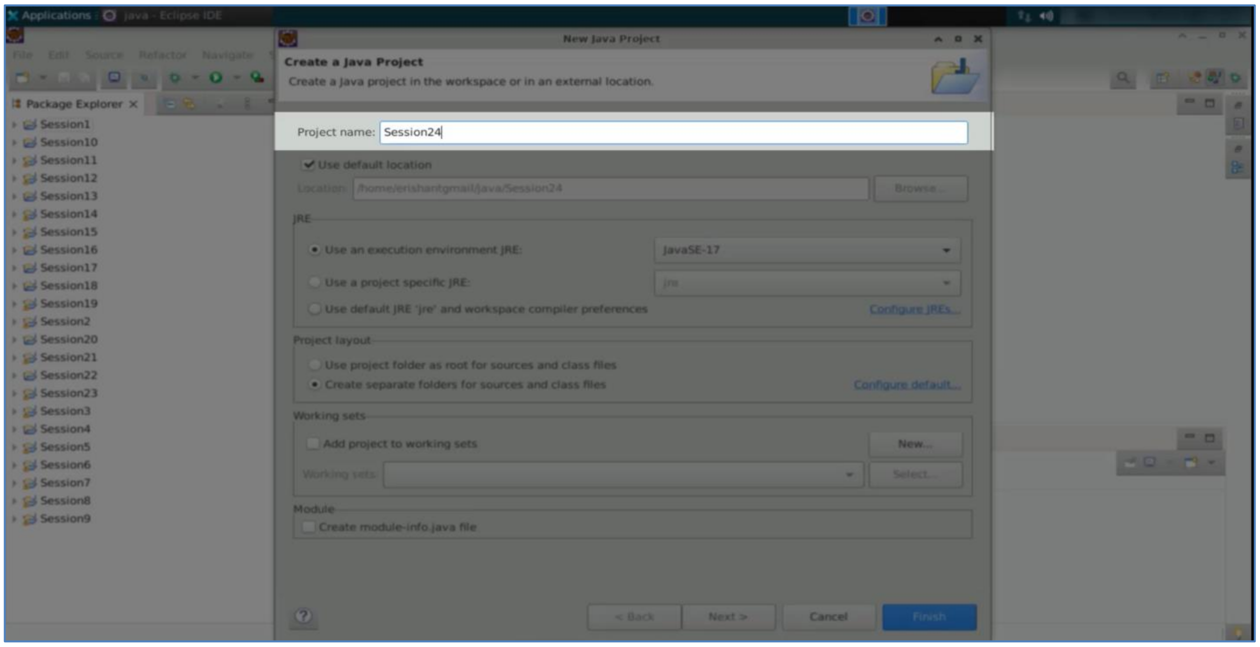
1.1 Open the **Eclipse IDE**



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



1.3 Name the project **"Session24"** and press **Finish**


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: Session24

[✓] Use default location

Location: /home/erishantgmail/java/Session24

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

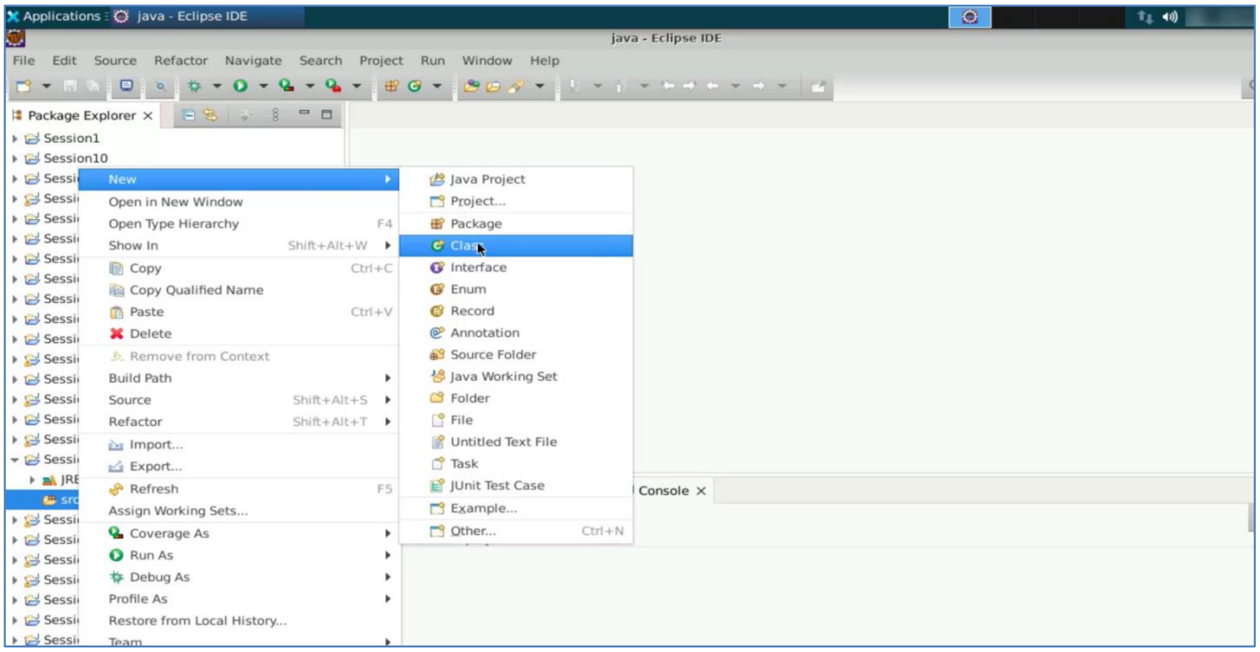
Working sets

[ ] Add project to working sets

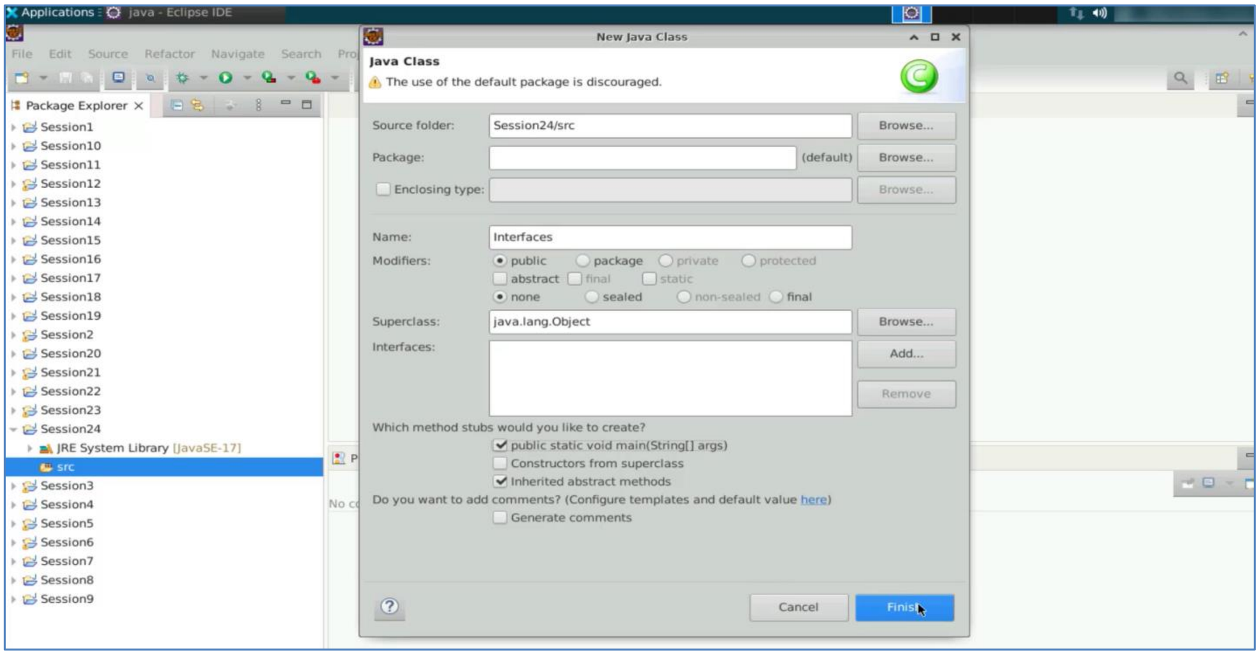
Module

[ ] Create module-info.java file

1.4 With a **Session24** on the src, do a right-click and create a **new class**



1.5 Name this class as an **Interface**, then click on check box “public static void main(String[] args)”**,** and then select **finish**



New Java Class

Java Class

The use of the default package is discouraged.

Source folder: Session24/src

Package: (default)

Enclosing type:

Name: Interfaces

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)

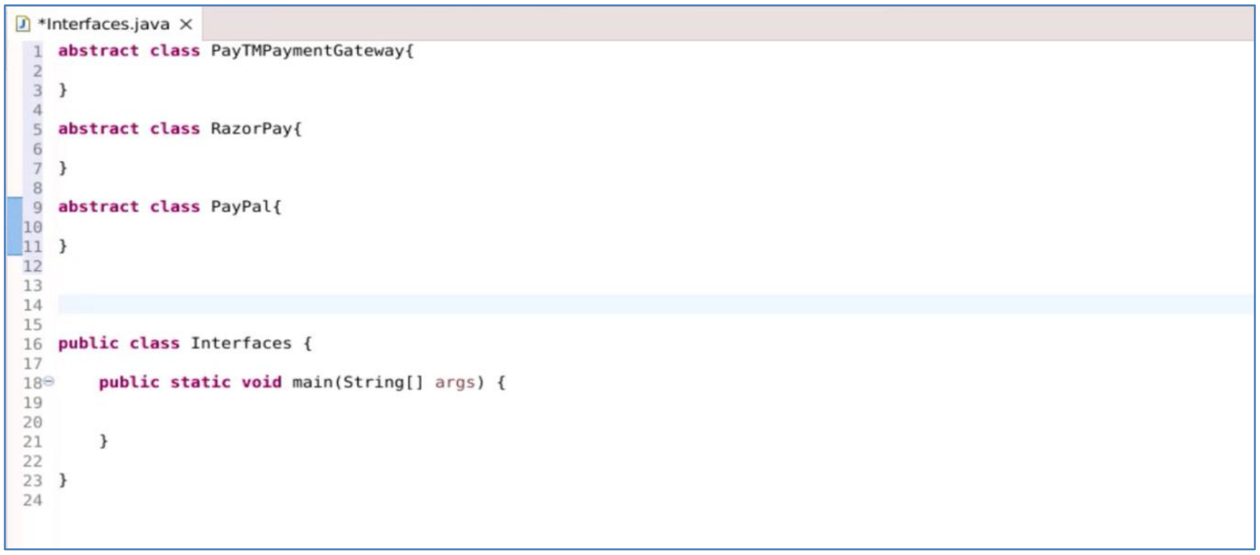
[ ] Constructors from superclass

[✓] Inherited abstract methods

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

[ ] Generate comments

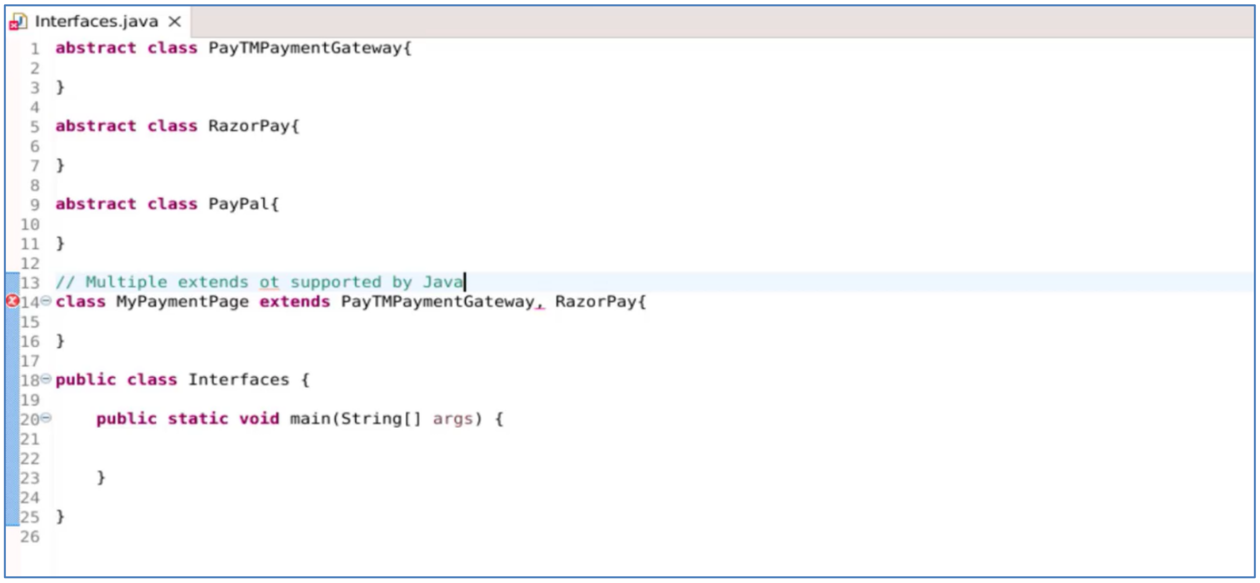
1.6 Create an interface called PayTMPaymentGateway. This interface will define the methods that any PayTM payment gateway class must implement. Create another interface called Razorpay. This interface will also define the methods that any Razorpay class must implement. Similarly, create an interface called PayPal for the PayPal payment gateway.



*Interfaces.java

```java
abstract class PayTMPaymentGateway{

}

abstract class RazorPay{

}

abstract class PayPal{

}



public class Interfaces {

    public static void main(String[] args) {


    }

}
```

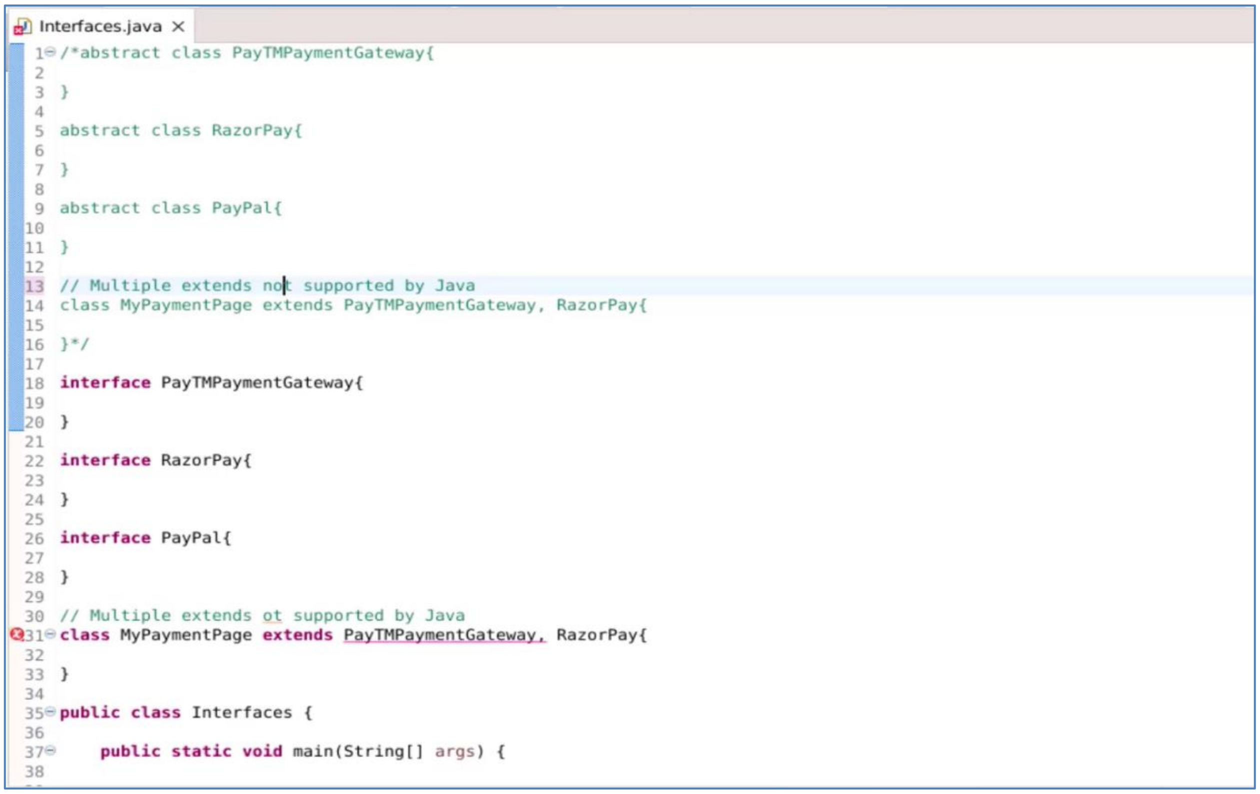
1.7 Define a class called MyPaymentPage, which will be the model for the payment page. Note: You can only extend one class, so if you want to create a payment structure, you need to write a class for it. You can extend the payment gateway from PayTM, but not from Razorpay due to this limitation. Java does not support multiple inheritance, so you cannot implement abstraction using runtime polymorphism with multiple classes.



Interfaces.java

```java
abstract class PayTMPaymentGateway{

}

abstract class RazorPay{

}

abstract class PayPal{

}

// Multiple extends ot supported by Java
class MyPaymentPage extends PayTMPaymentGateway, RazorPay{

}

public class Interfaces {

    public static void main(String[] args) {


    }

}
```

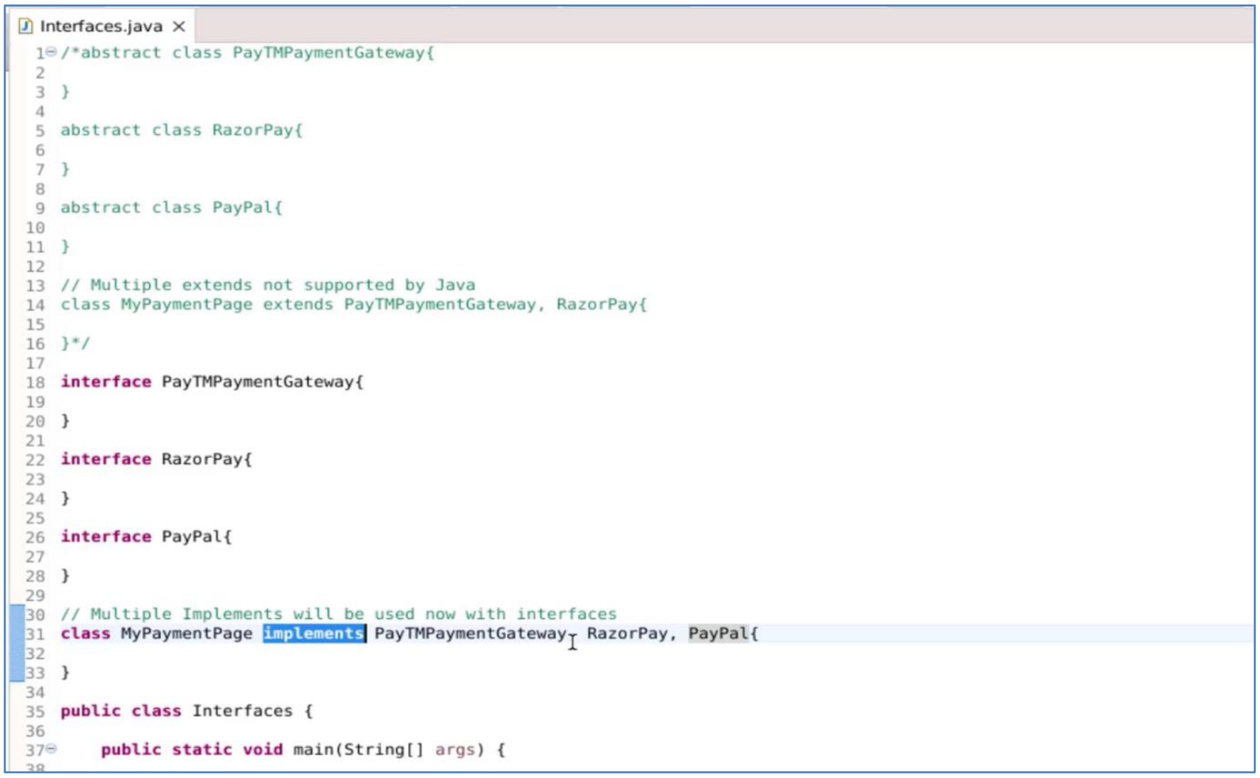
1.8 Instead of extending the PaymentGateway class, have MyPaymentPage implement the PaymentGateway interface using the keyword 'implements'. You can also have MyPaymentPage implement the Razorpay and PayPal interfaces, if needed, using the same 'implements' keyword. Now, what is an interface? It is exactly like an abstract class; you can simply consider that for whatever purpose the abstract class was being used, you can use an interface for the same purpose.



Interfaces.java ×

```java
/*abstract class PayTMPaymentGateway{

}

abstract class RazorPay{

}

abstract class PayPal{

}

// Multiple extends not supported by Java
class MyPaymentPage extends PayTMPaymentGateway, RazorPay{

}*/

interface PayTMPaymentGateway{

}

interface RazorPay{

}

interface PayPal{

}

// Multiple extends ot supported by Java
class MyPaymentPage extends PayTMPaymentGateway, RazorPay{

}

public class Interfaces {

    public static void main(String[] args) {

```

1.9 Now, implement the methods of each interface in the MyPaymentPage class. You can create as many payment gateway classes as you want if they implement the PaymentGateway interface. Similarly, you can create as many Razorpay or PayPal classes as you want if they implement their respective interfaces. By using interfaces, you can achieve the same level of abstraction as with abstract classes, but with the added benefit of being able to implement multiple interfaces in a single class.

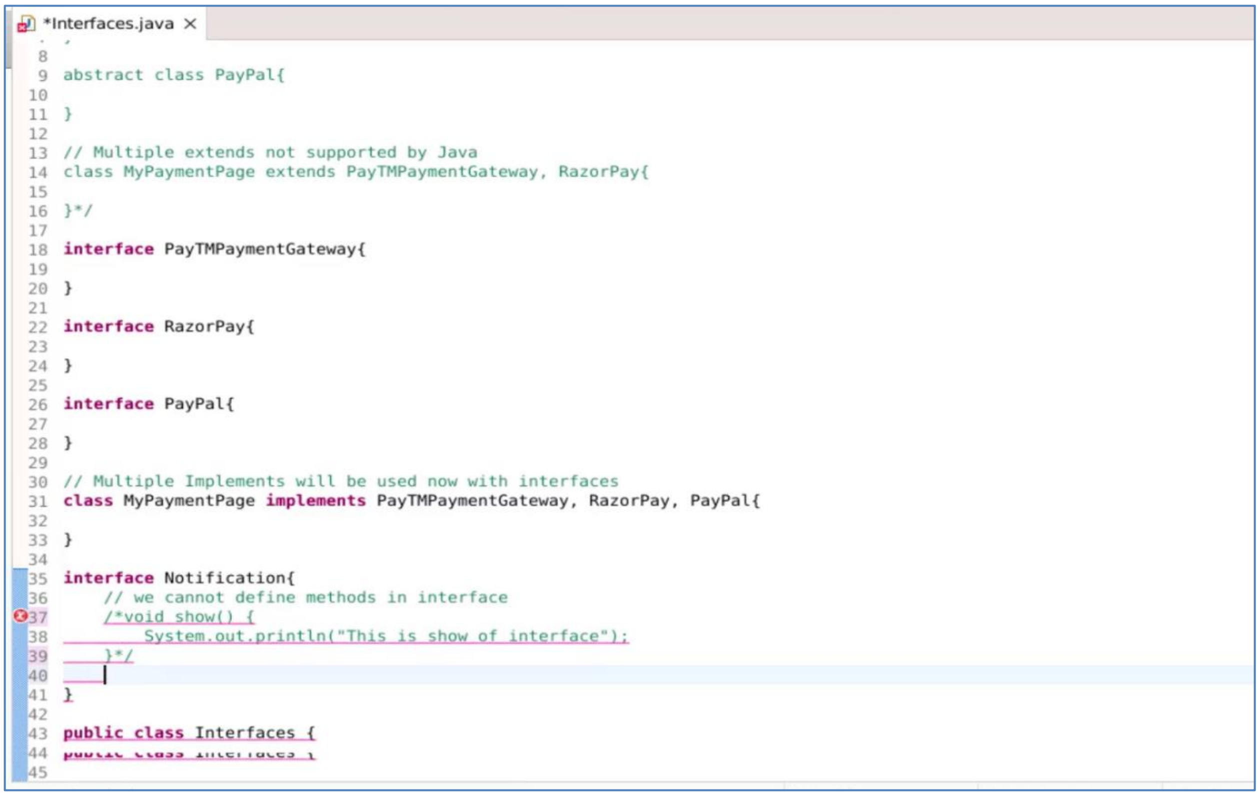


Interfaces.java

```
/*abstract class PayTMPaymentGateway{

}

abstract class RazorPay{

}

abstract class PayPal{

}

// Multiple extends not supported by Java
class MyPaymentPage extends PayTMPaymentGateway, RazorPay{

}*/

interface PayTMPaymentGateway{

}

interface RazorPay{

}

interface PayPal{

}

// Multiple Implements will be used now with interfaces
class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{

}

public class Interfaces {

    public static void main(String[] args) {
```

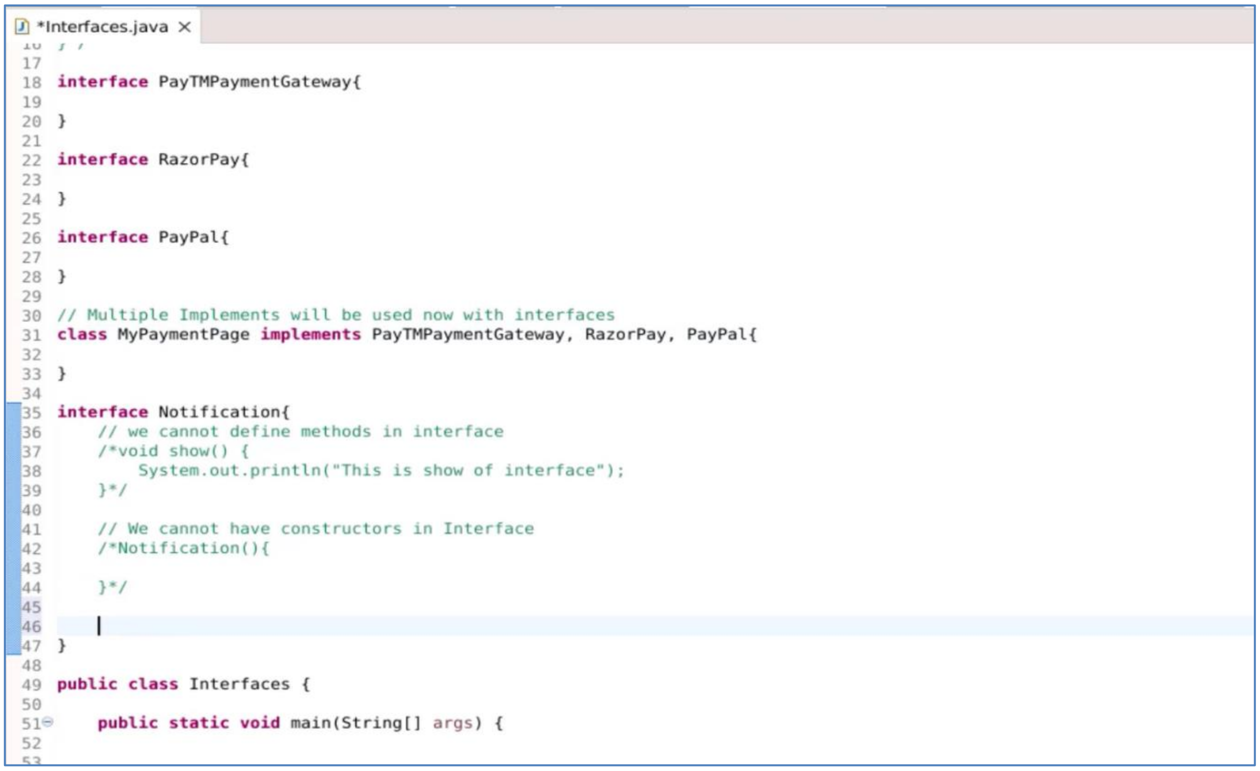
Step 2: Comprehend how interfaces work

2.1 Start by creating an interface for notifications. Attempt to define a method within the interface but observe that it results in an error. This is because methods cannot be defined within an interface. Print a message that indicates interfaces do not allow method definitions, such as 'This is a demonstration of an interface. Note that we cannot define methods within interfaces, which is why attempting to do so results in an error.


*Interfaces.java

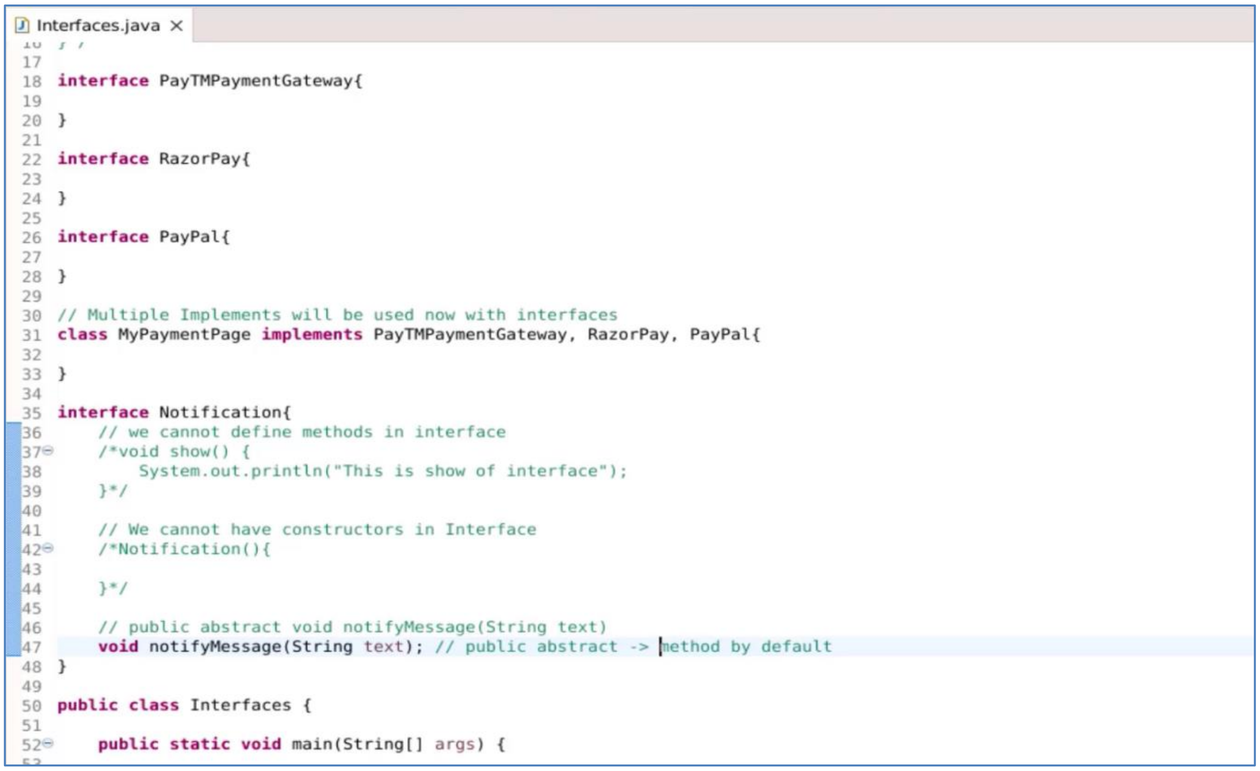
```
abstract class PayPal{

}

// Multiple extends not supported by Java
class MyPaymentPage extends PayTMPaymentGateway, RazorPay{

}*/

interface PayTMPaymentGateway{

}

interface RazorPay{

}

interface PayPal{

}

// Multiple Implements will be used now with interfaces
class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{

}

interface Notification{
    // we cannot define methods in interface
    /*void show() {
        System.out.println("This is show of interface");
    }*/
    
}

public class Interfaces {
```

2.2 Attempt to create a constructor for the interface but observe that this is also not allowed. Interfaces cannot have constructors, so attempting
to create one will result in an error. Print a message that indicates interfaces cannot have constructors, such as 'It is important to note that interfaces cannot have constructors. This means that neither you nor the runtime environment can create an object of an interface.


*Interfaces.java

```java

interface PayTMPaymentGateway{

}

interface RazorPay{

}

interface PayPal{

}

// Multiple Implements will be used now with interfaces
class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{

}

interface Notification{
    // we cannot define methods in interface
    /*void show() {
        System.out.println("This is show of interface");
    }*/

    // We cannot have constructors in Interface
    /*Notification(){

    }*/


}

public class Interfaces {

    public static void main(String[] args) {

```

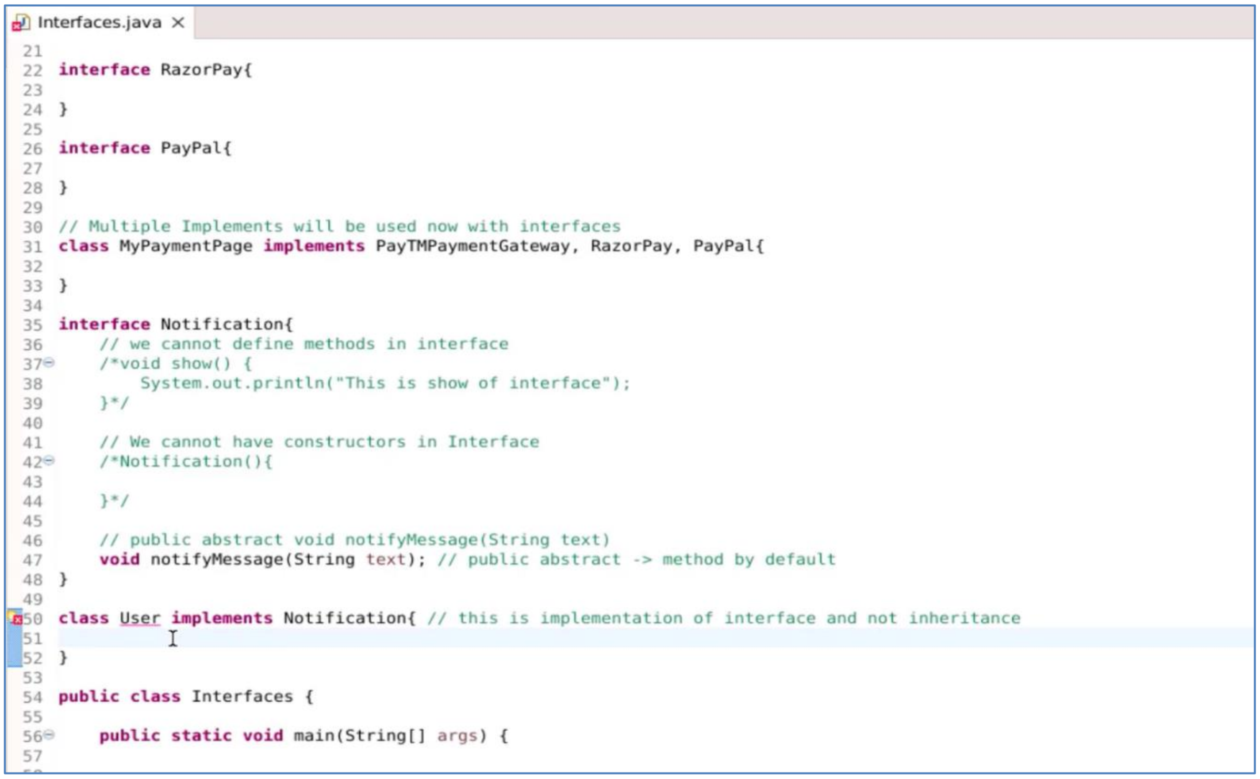
2.3 Declare an abstract method within the interface that accepts a string parameter called **text**. By default, this method will be public, abstract, and void. You can simply write **void notifyMessage(String text);** without including the access modifier or abstract keyword, as these are both **implied.Remember** that interfaces can have multiple abstract methods declared within them but cannot define any concrete methods. This means that any class that implements the interface must provide an implementation for all its abstract methods.



Interfaces.java

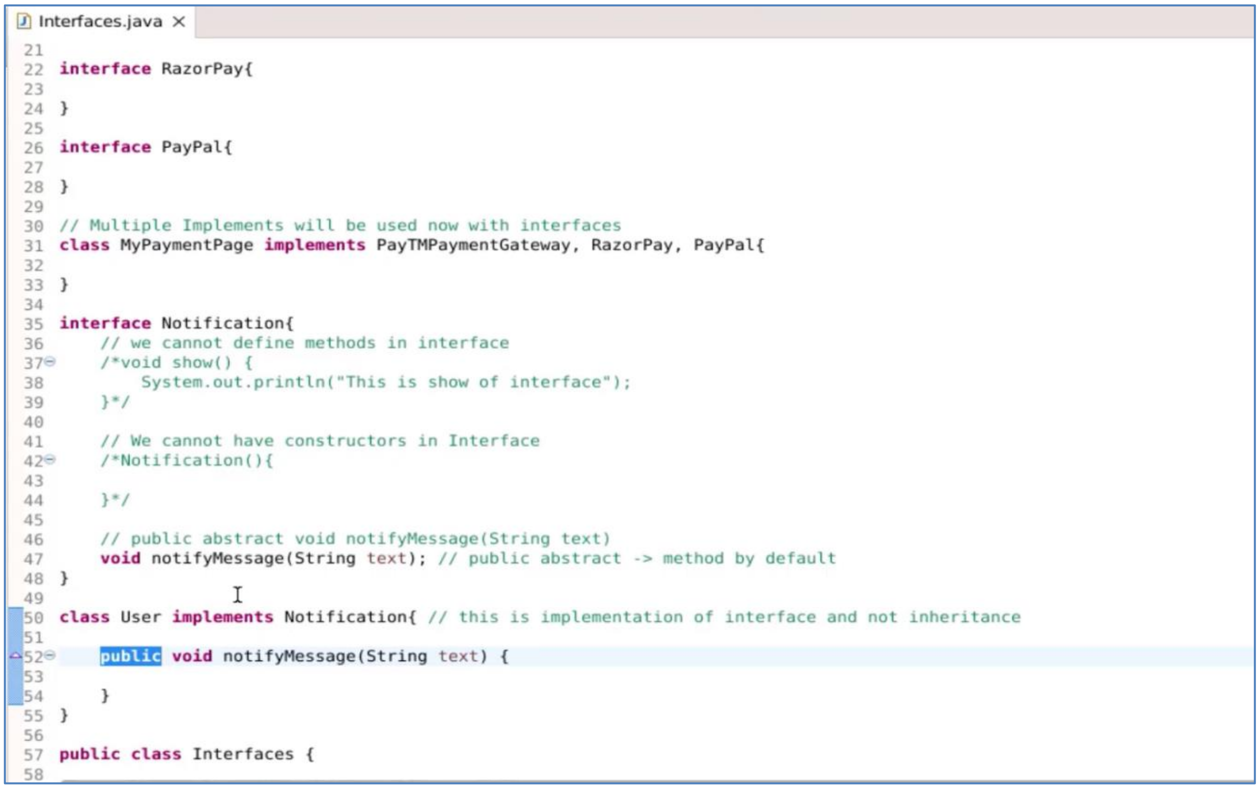
```java

interface PayTMPaymentGateway{

}

interface RazorPay{

}

interface PayPal{

}

// Multiple Implements will be used now with interfaces
class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{

}

interface Notification{
    // we cannot define methods in interface
    /*void show() {
        System.out.println("This is show of interface");
    }*/

    // We cannot have constructors in Interface
    /*Notification(){

    }*/

    // public abstract void notifyMessage(String text)
    void notifyMessage(String text); // public abstract -> method by default
}

public class Interfaces {

    public static void main(String[] args) {
```

2.4 To implement a notification interface, the user does not need to inherit from it. Instead, the user will implement the interface by defining its methods. Inheritance is a different concept from interface implementation, where inheritance creates a parent-child relationship. However, interfaces do not have constructors, which means objects of interfaces cannot be created



```

interface RazorPay{

}

interface PayPal{

}

// Multiple Implements will be used now with interfaces
class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{

}

interface Notification{
    // we cannot define methods in interface
    /*void show() {
        System.out.println("This is show of interface");
    }*/

    // We cannot have constructors in Interface
    /*Notification(){

    }*/

    // public abstract void notifyMessage(String text)
    void notifyMessage(String text); // public abstract -> method by default
}

class User implements Notification{ // this is implementation of interface and not inheritance

}

public class Interfaces {

    public static void main(String[] args) {

```

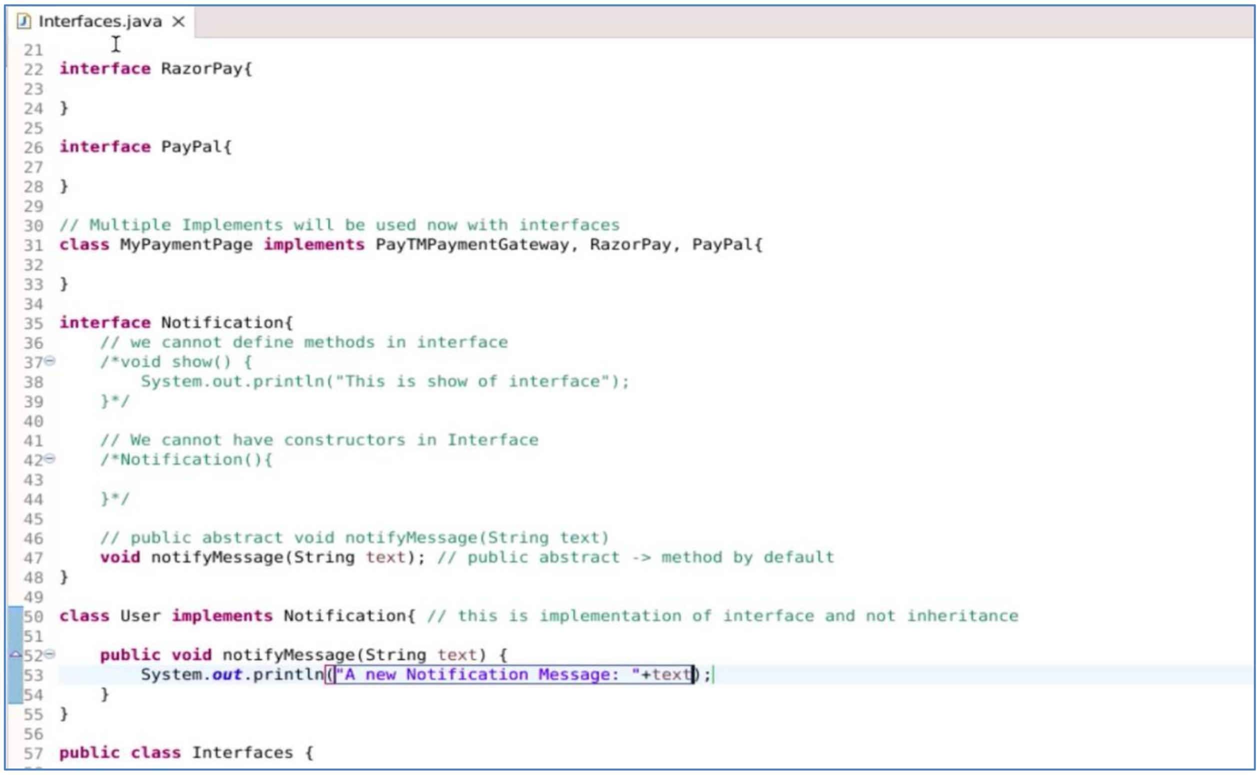
2.5 Interfaces contain abstract methods that must be defined by classes that implement the interface. The methods defined in the implementing classes must be declared as public, since the abstract methods in the interface are public by default. The user can print a new notification message with any text by implementing the notification interface.



Interfaces.java

```java

interface RazorPay{

}

interface PayPal{

}

// Multiple Implements will be used now with interfaces
class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{

}

interface Notification{
    // we cannot define methods in interface
    /*void show() {
        System.out.println("This is show of interface");
    }*/

    // We cannot have constructors in Interface
    /*Notification(){

    }*/

    // public abstract void notifyMessage(String text)
    void notifyMessage(String text); // public abstract -> method by default
}

class User implements Notification{ // this is implementation of interface and not inheritance

    public void notifyMessage(String text) {

    }
}

public class Interfaces {

```

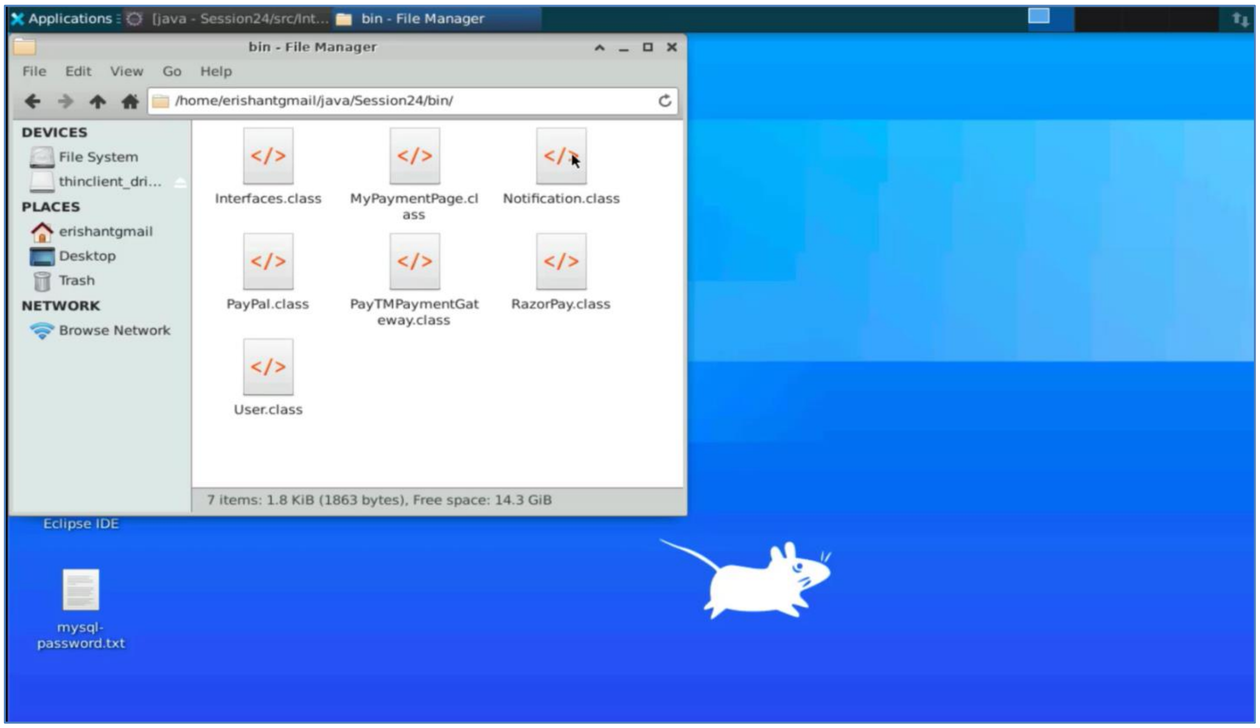
2.6 The user can print a new notification message with any text by implementing the notification interface. Interfaces cannot be treated like classes, and even when they are compiled, they generate byte codes.



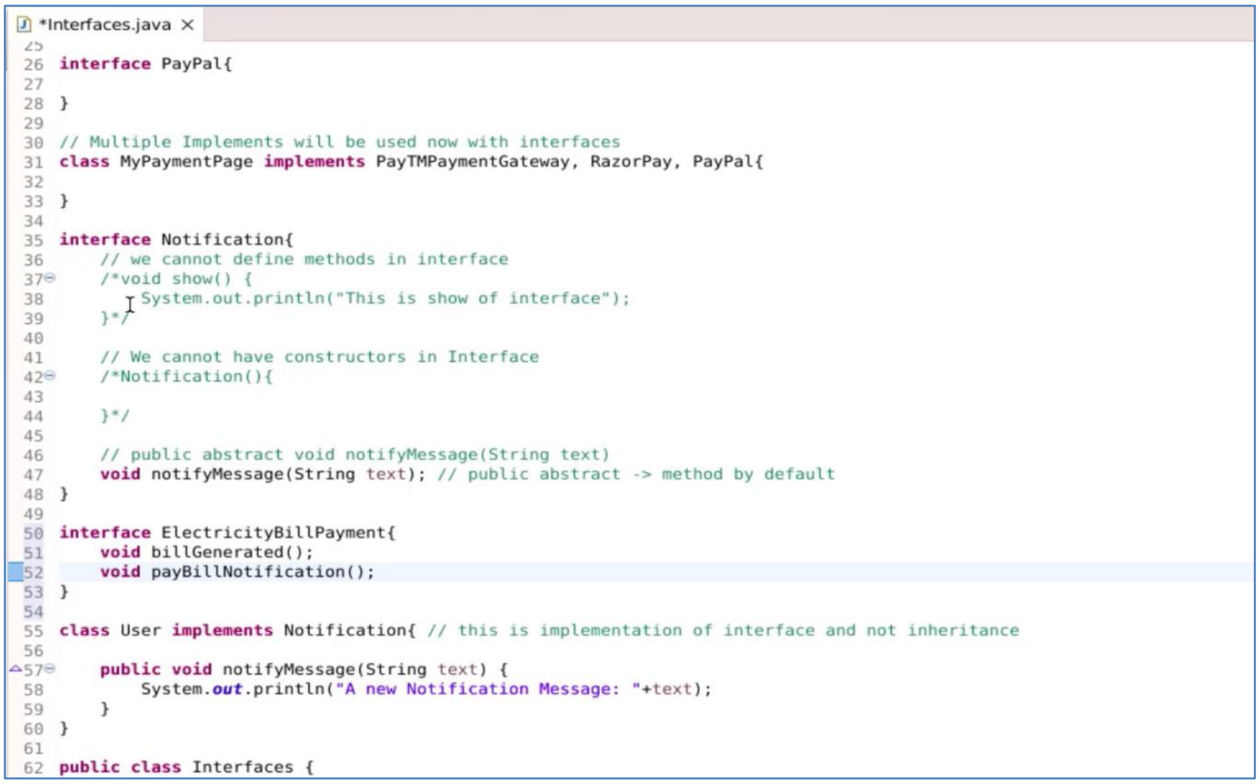
Interfaces.java

```

interface RazorPay{

}

interface PayPal{

}

// Multiple Implements will be used now with interfaces
class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{

}

interface Notification{
    // we cannot define methods in interface
    /*void show() {
        System.out.println("This is show of interface");
    }*/

    // We cannot have constructors in Interface
    /*Notification(){

    }*/

    // public abstract void notifyMessage(String text)
    void notifyMessage(String text); // public abstract -> method by default
}

class User implements Notification{ // this is implementation of interface and not inheritance

    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }
}

public class Interfaces {
```

2.7 To see the byte codes generated for interfaces, go to **Home -> java -> session 24 -> bin** directory.



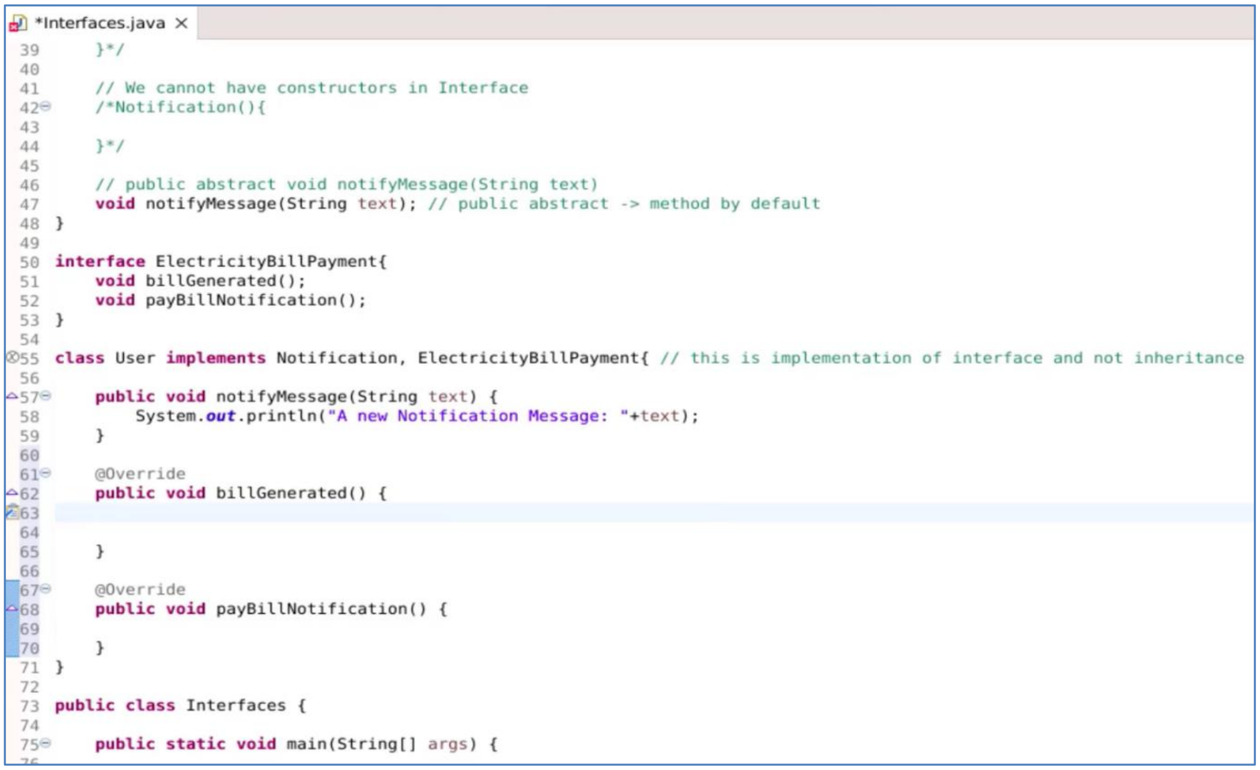
2.8 Now, Let Us define the **ElectricityBillPayment** interface with two abstract methods: **void billGenerated()** and **void payBillNotification()**



*Interfaces.java

```java

interface PayPal{

}

// Multiple Implements will be used now with interfaces
class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{

}

interface Notification{
    // we cannot define methods in interface
    /*void show() {
        System.out.println("This is show of interface");
    }*/

    // We cannot have constructors in Interface
    /*Notification(){

    }*/

    // public abstract void notifyMessage(String text)
    void notifyMessage(String text); // public abstract -> method by default
}

interface ElectricityBillPayment{
    void billGenerated();
    void payBillNotification();
}

class User implements Notification{ // this is implementation of interface and not inheritance

    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }
}

public class Interfaces {
```

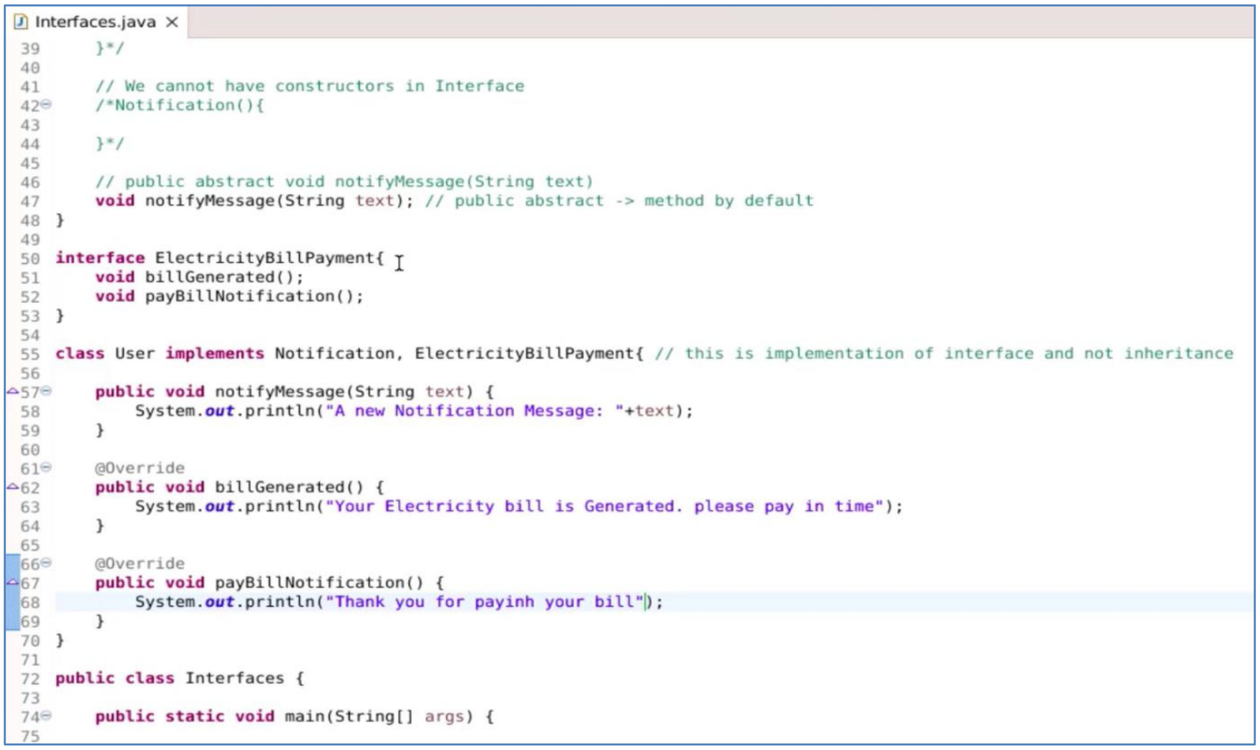
2.9 User implements the Electricity Bill Payment interface. Whenever the electricity bill is generated, the user is notified. The user pays the electricity bill. The user implements the Pay Bill Notification interface. Once the electricity bill is paid, a notification is sent to the user.



*Interfaces.java

```
    }*/

    // We cannot have constructors in Interface
    /*Notification(){

    }*/

    // public abstract void notifyMessage(String text)
    void notifyMessage(String text); // public abstract -> method by default
}

interface ElectricityBillPayment{
    void billGenerated();
    void payBillNotification();
}

class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance

    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }

    @Override
    public void billGenerated() {


    }

    @Override
    public void payBillNotification() {

    }
}

public class Interfaces {

    public static void main(String[] args) {
```

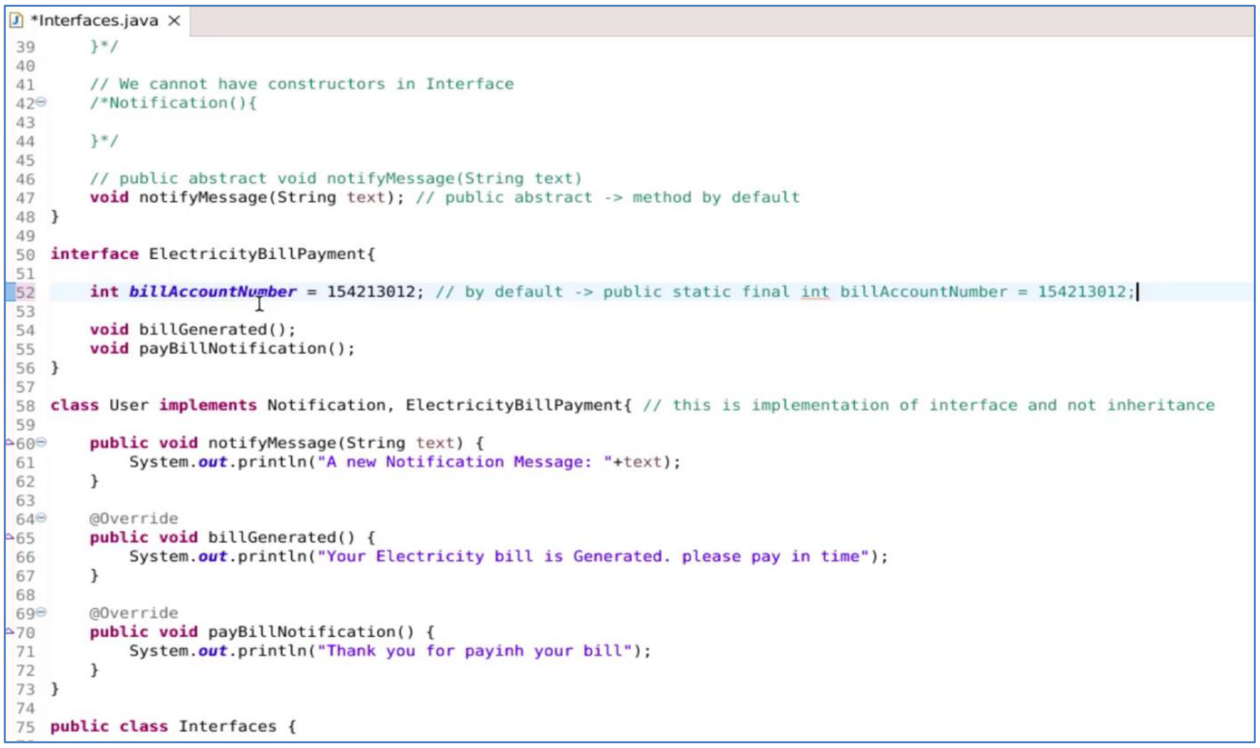
2.10 First, the electricity bill is generated. Next, a message is printed requesting the customer to pay the bill on time. The message is **Please pay your bill on time**. The payment notification will be displayed using the system.out.println. The message will be **Thank you for paying your bill**. Additionally, the interface can have variables, but these variables are constants



Interfaces.java

```java
    }*/

    // We cannot have constructors in Interface
    /*Notification(){

    }*/

    // public abstract void notifyMessage(String text)
    void notifyMessage(String text); // public abstract -> method by default
}

interface ElectricityBillPayment{
    void billGenerated();
    void payBillNotification();
}

class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance

    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }

    @Override
    public void billGenerated() {
        System.out.println("Your Electricity bill is Generated. please pay in time");
    }

    @Override
    public void payBillNotification() {
        System.out.println("Thank you for payinh your bill");
    }
}

public class Interfaces {

    public static void main(String[] args) {

```

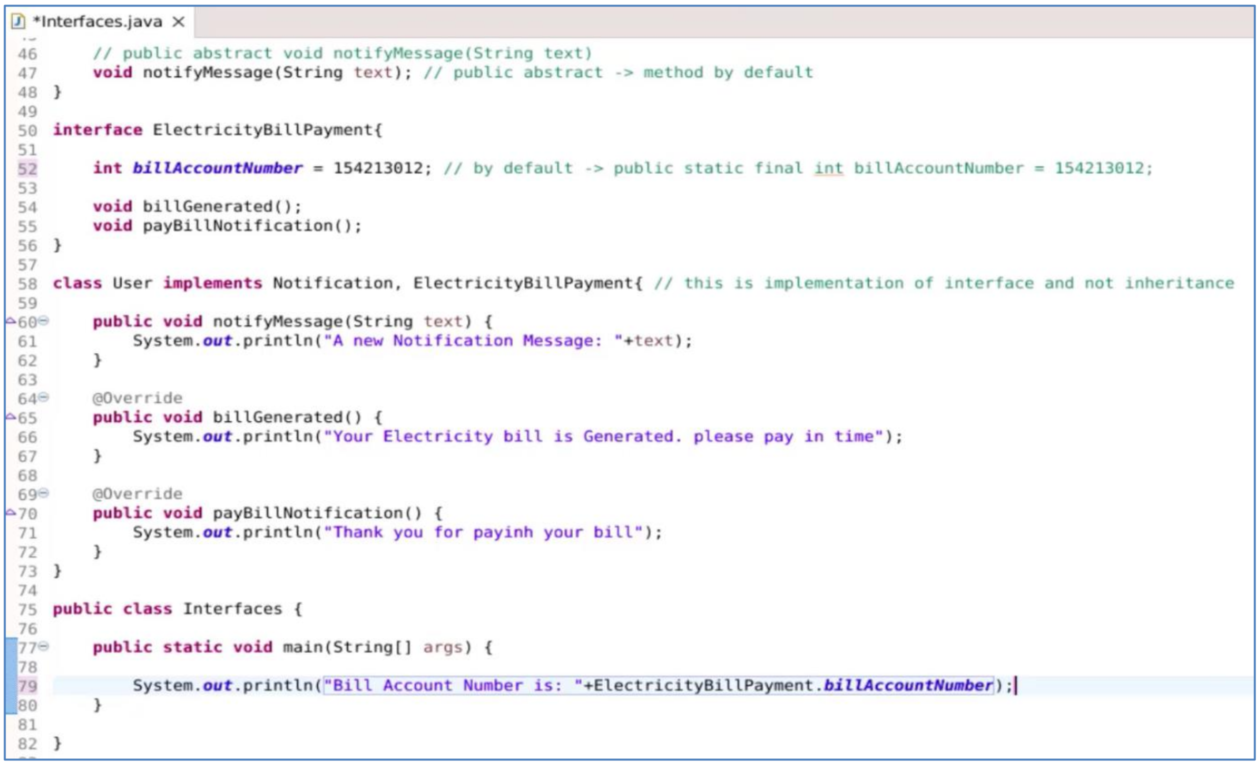
2.11 Create a variable named **Bill account number** and assign it a value, say **154213012**. Since it is a final variable, it cannot be manipulated later.



*Interfaces.java

```java
    }*/

    // We cannot have constructors in Interface
    /*Notification(){

    }*/

    // public abstract void notifyMessage(String text)
    void notifyMessage(String text); // public abstract -> method by default
}

interface ElectricityBillPayment{

    int billAccountNumber = 154213012; // by default -> public static final int billAccountNumber = 154213012;

    void billGenerated();
    void payBillNotification();
}

class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance

    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }

    @Override
    public void billGenerated() {
        System.out.println("Your Electricity bill is Generated. please pay in time");
    }

    @Override
    public void payBillNotification() {
        System.out.println("Thank you for payinh your bill");
    }
}

public class Interfaces {
```

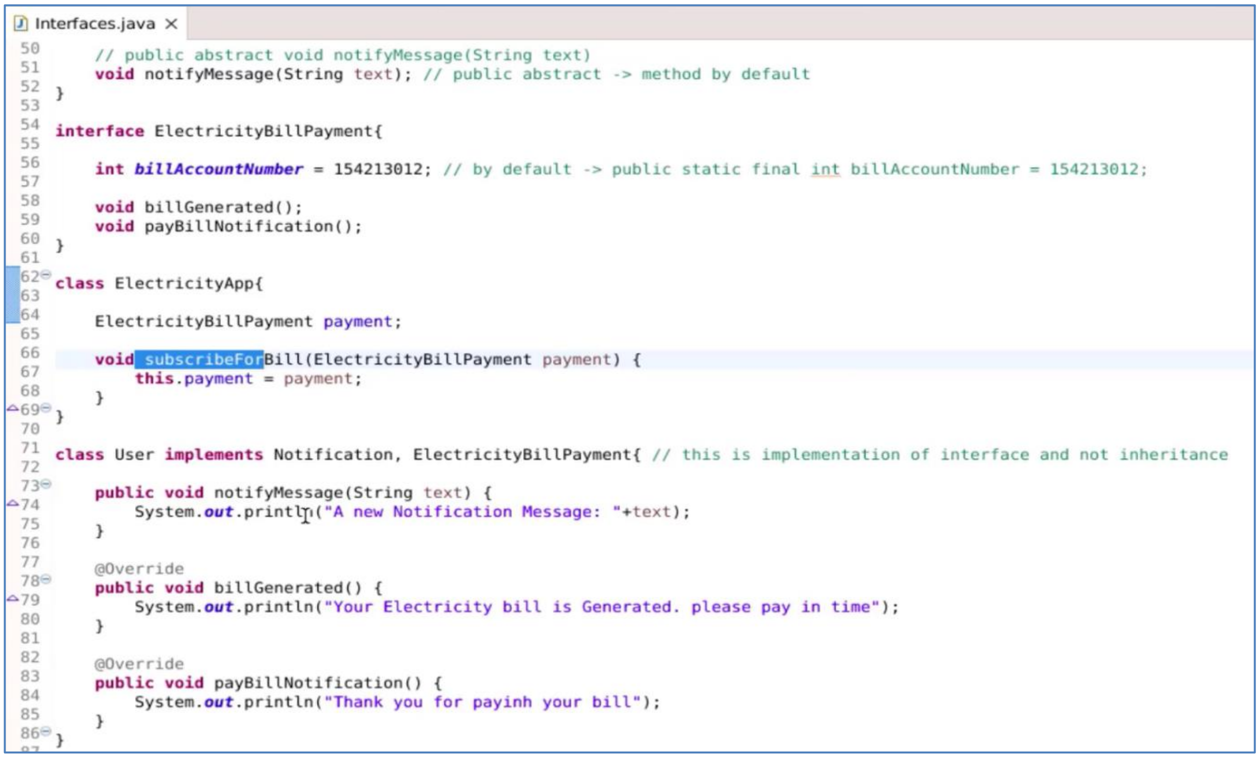
2.12 To access the variable, use the interface name and the dot operator. Since it is a static variable, it can be accessed with the interface name.



*Interfaces.java

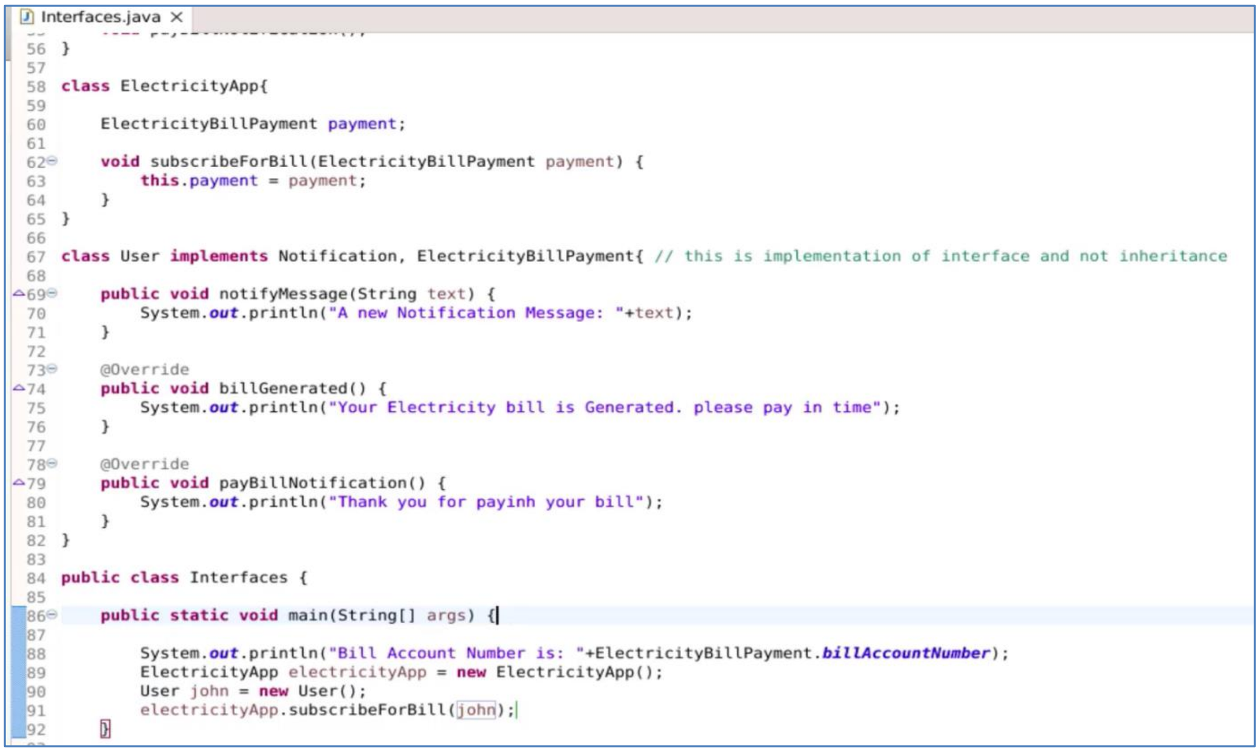
```java
    // public abstract void notifyMessage(String text)
    void notifyMessage(String text); // public abstract -> method by default
}

interface ElectricityBillPayment{

    int billAccountNumber = 154213012; // by default -> public static final int billAccountNumber = 154213012;

    void billGenerated();
    void payBillNotification();
}

class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance

    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }

    @Override
    public void billGenerated() {
        System.out.println("Your Electricity bill is Generated. please pay in time");
    }

    @Override
    public void payBillNotification() {
        System.out.println("Thank you for payinh your bill");
    }
}

public class Interfaces {

    public static void main(String[] args) {

        System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
    }

}
```

2.13 Create a class named **ElectricityApp** with a method named **subscribeForBill** and another method named **payment**. Declare a variable called **payment** in the class and assign it to the value of the method parameter.



```
    // public abstract void notifyMessage(String text)
    void notifyMessage(String text); // public abstract -> method by default
}

interface ElectricityBillPayment{

    int billAccountNumber = 154213012; // by default -> public static final int billAccountNumber = 154213012;

    void billGenerated();
    void payBillNotification();
}

class ElectricityApp{

    ElectricityBillPayment payment;

    void subscribeForBill(ElectricityBillPayment payment) {
        this.payment = payment;
    }
}

class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance

    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }

    @Override
    public void billGenerated() {
        System.out.println("Your Electricity bill is Generated. please pay in time");
    }

    @Override
    public void payBillNotification() {
        System.out.println("Thank you for payinh your bill");
    }
}
```

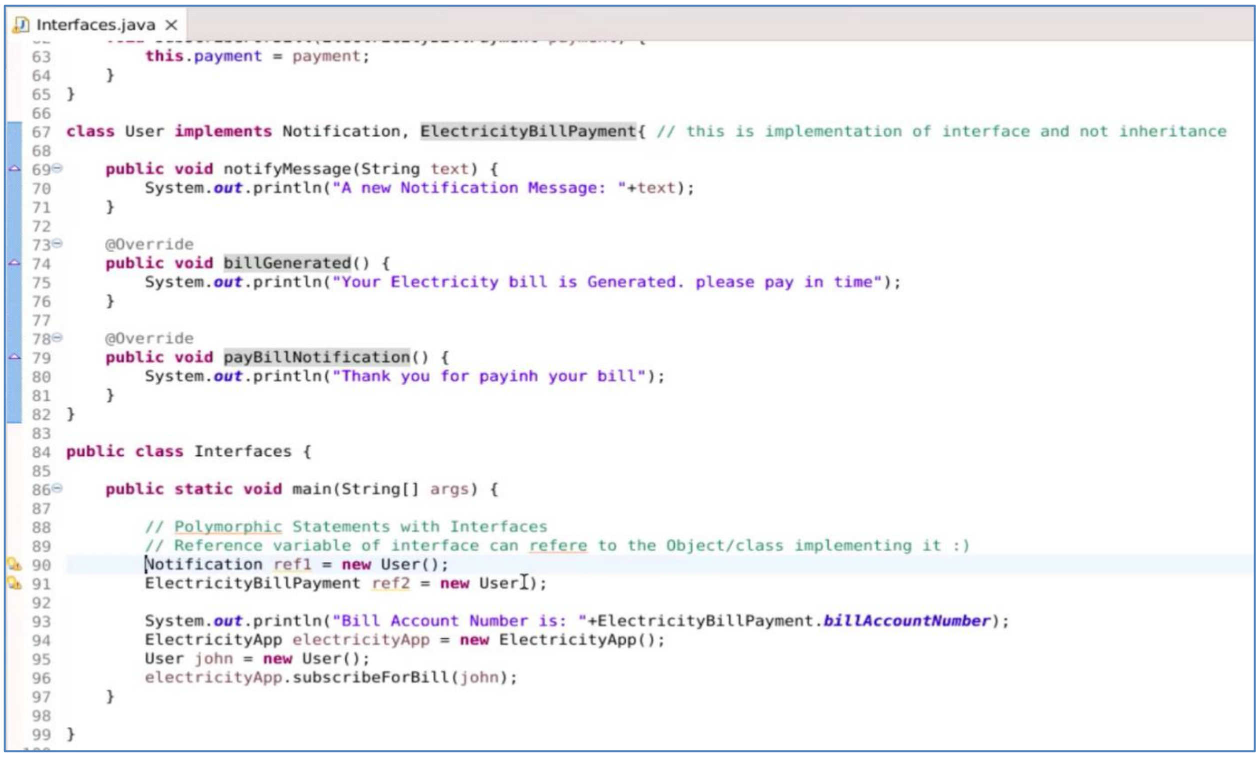
2.14 Create an object of the **ElectricityApp** class and a user named **John**. Then call the **subscribeForBill** method of the **ElectricityApp** object to subscribe John for the bill.



```java
}

class ElectricityApp{

    ElectricityBillPayment payment;

    void subscribeForBill(ElectricityBillPayment payment) {
        this.payment = payment;
    }
}

class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance

    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }

    @Override
    public void billGenerated() {
        System.out.println("Your Electricity bill is Generated. please pay in time");
    }

    @Override
    public void payBillNotification() {
        System.out.println("Thank you for payinh your bill");
    }
}

public class Interfaces {

    public static void main(String[] args) {

        System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
        ElectricityApp electricityApp = new ElectricityApp();
        User john = new User();
        electricityApp.subscribeForBill(john);
    }
```

Step 3: How to write polymorphic statements using interfaces.

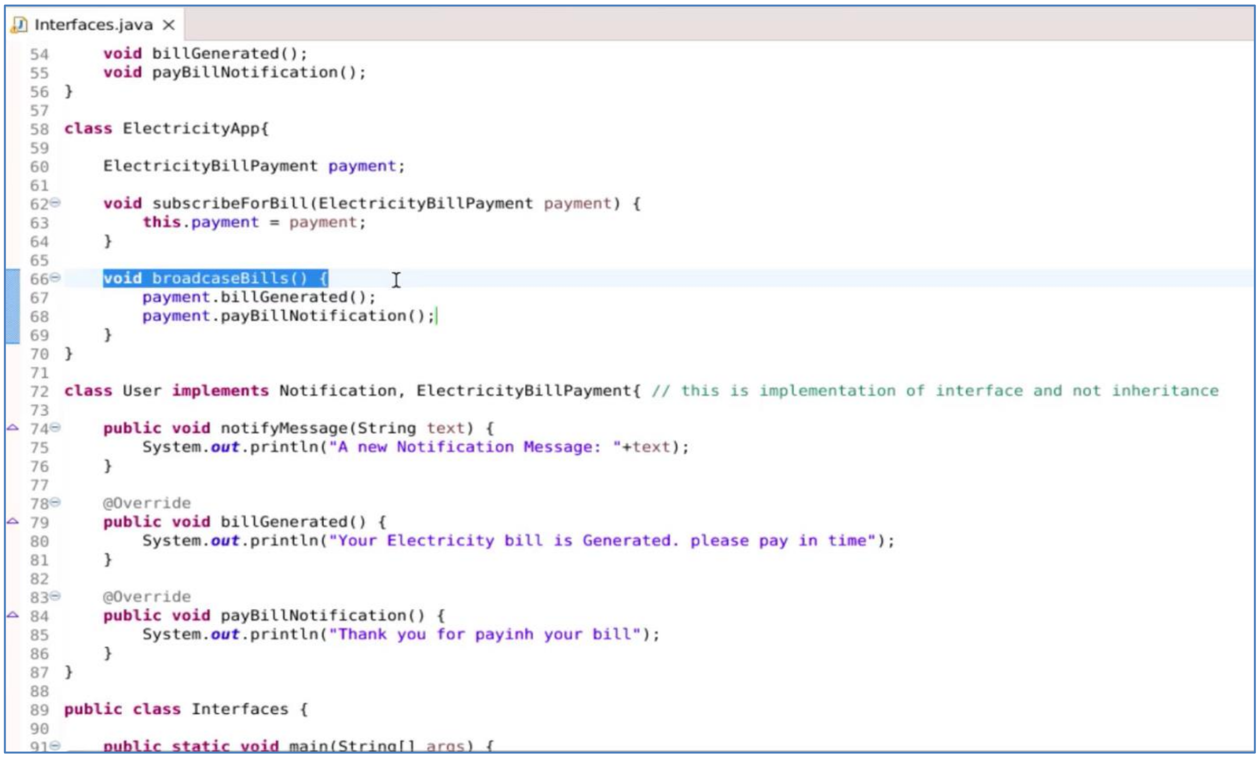
3.1 Create an interface called **Notification**. Create a new user object and assign it to a reference variable. Create another reference variable named "payment" which can also refer to a new user object. This is a polymorphic statement because the reference variable of an interface can refer to the object/class implementing it. Implement the **Notification** and **ElectricityBillPayment** interfaces in the User class. This makes the User class capable of sending notifications and paying electricity bills. Subscribe John to the electricity bill payment by copying John's details into the **payment** reference variable.



Interfaces.java

```
        this.payment = payment;
    }
}

class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance

    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }

    @Override
    public void billGenerated() {
        System.out.println("Your Electricity bill is Generated. please pay in time");
    }

    @Override
    public void payBillNotification() {
        System.out.println("Thank you for payinh your bill");
    }
}

public class Interfaces {

    public static void main(String[] args) {

        // Polymorphic Statements with Interfaces
        // Reference variable of interface can refere to the Object/class implementing it :)
        Notification ref1 = new User();
        ElectricityBillPayment ref2 = new User();

        System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
        ElectricityApp electricityApp = new ElectricityApp();
        User john = new User();
        electricityApp.subscribeForBill(john);
    }

}
```

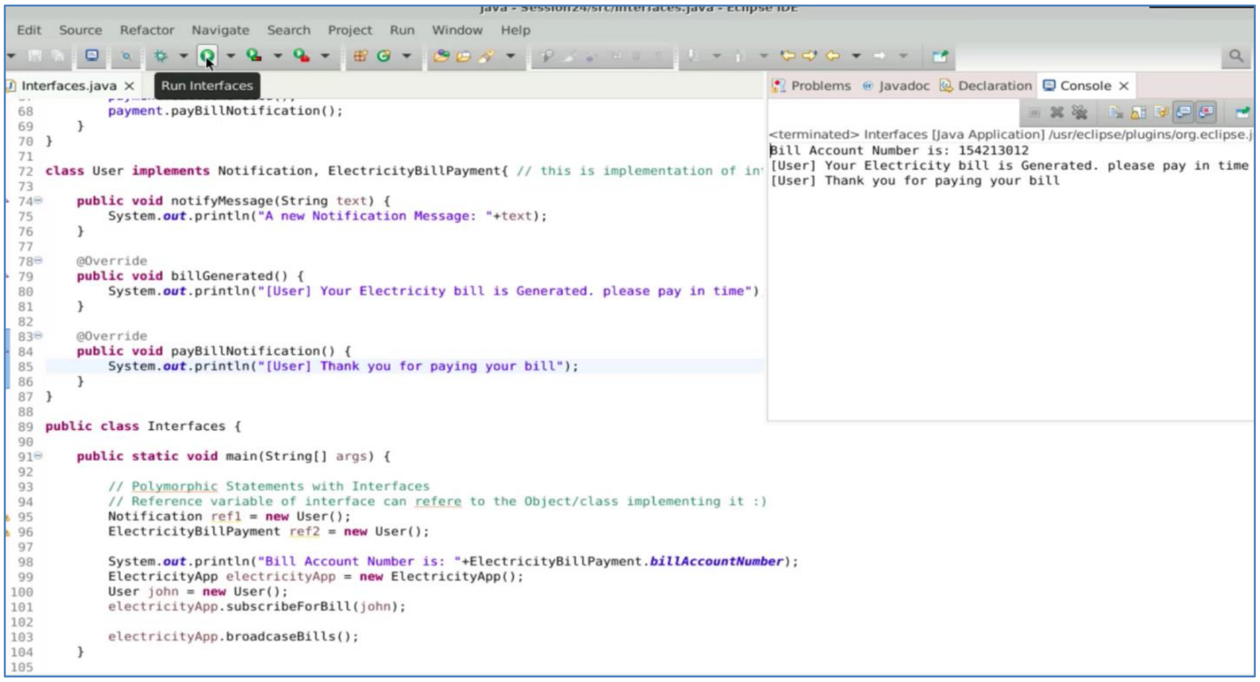
3.2 Create a method called **broadcastBills** which executes the **billGenerated** method and uses the **payBillNotification** method. Execute the **broadcastBills** method when bills are generated. This will notify the user object that their bill has been generated and they should pay it on time.



Interfaces.java

```
    void billGenerated();
    void payBillNotification();
}

class ElectricityApp{

    ElectricityBillPayment payment;

    void subscribeForBill(ElectricityBillPayment payment) {
        this.payment = payment;
    }

    void broadcaseBills() {
        payment.billGenerated();
        payment.payBillNotification();
    }
}

class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance

    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }

    @Override
    public void billGenerated() {
        System.out.println("Your Electricity bill is Generated. please pay in time");
    }

    @Override
    public void payBillNotification() {
        System.out.println("Thank you for payinh your bill");
    }
}

public class Interfaces {

    public static void main(String[] args) {
```

3.3 Use the Electricity app to execute the **broadcastBills** method. The user will be notified that their electricity bill has been generated and they should pay it on time. This message will be sent from the user. Abstraction is no longer limited to abstract classes because interfaces can be implemented by multiple classes.

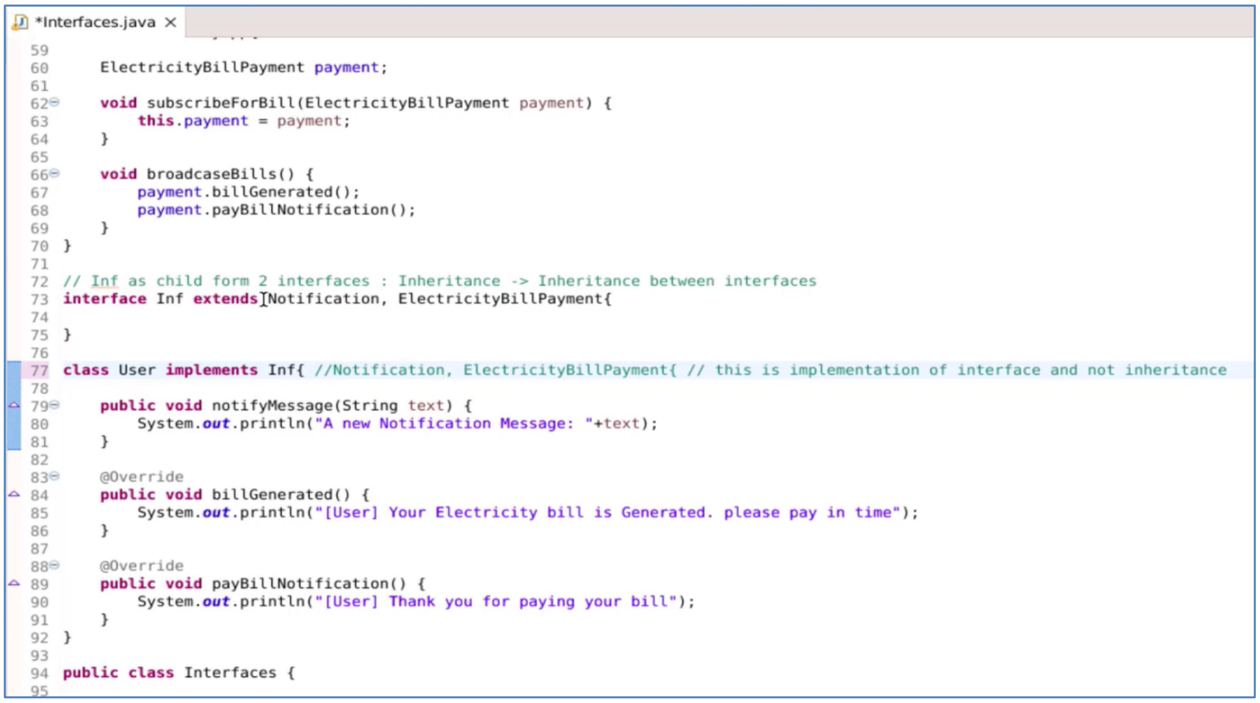

java - Session24/src/Interfaces.java - Eclipse IDE
Edit Source Refactor Navigate Search Project Run Window Help
Interfaces.java
Run Interfaces
Problems Javadoc Declaration Console

```java
        payment.payBillNotification();
    }
}

class User implements Notification, ElectricityBillPayment{ // this is implementation of in

    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }

    @Override
    public void billGenerated() {
        System.out.println("[User] Your Electricity bill is Generated. please pay in time")
    }

    @Override
    public void payBillNotification() {
        System.out.println("[User] Thank you for paying your bill");
    }
}

public class Interfaces {

    public static void main(String[] args) {

        // Polymorphic Statements with Interfaces
        // Reference variable of interface can refere to the Object/class implementing it :)
        Notification ref1 = new User();
        ElectricityBillPayment ref2 = new User();

        System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
        ElectricityApp electricityApp = new ElectricityApp();
        User john = new User();
        electricityApp.subscribeForBill(john);

        electricityApp.broadcaseBills();
    }

```

```
<terminated> Interfaces [Java Application] /usr/eclipse/plugins/org.eclipse.j
Bill Account Number is: 154213012
[User] Your Electricity bill is Generated. please pay in time
[User] Thank you for paying your bill
```

Step 4: Implement how multiple inheritances work in interfaces.

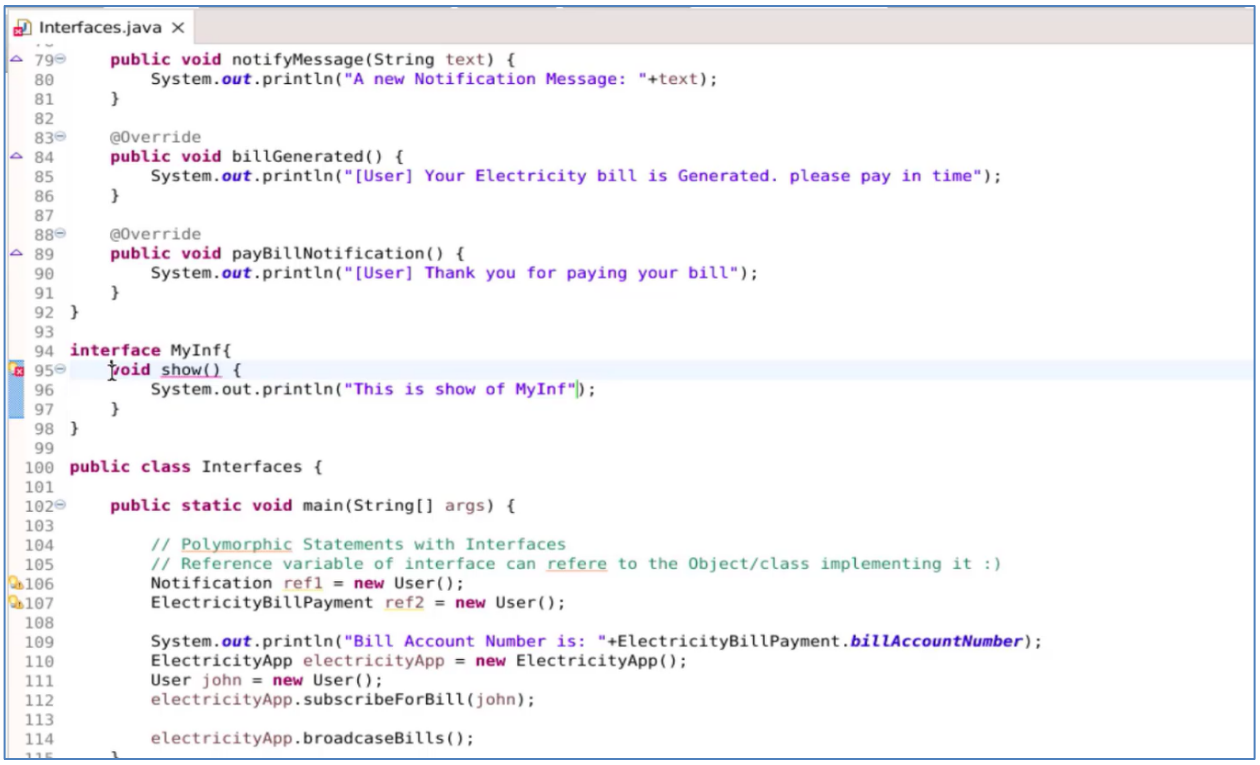
4.1 Define an interface called `Notification` and `ElectricityBillPayment` with their method signatures. Define an interface called `INF` that extends both `Notification` and `ElectricityBillPayment`. The `INF` interface now inherits the methods from both these interfaces. The benefit of implementing the `INF` interface is that you can execute all three methods from a reference variable of this interface. By using this interface, you can implement multiple interfaces in a single implementation. Interface inheritance differs from class inheritance because interfaces can be inherited in multiple ways using the `extends` keyword. With this implementation, you can access all the methods defined in both `Notification` and `ElectricityBillPayment` interfaces through the reference variable of the `INF` interface.


*Interfaces.java

```

    ElectricityBillPayment payment;

    void subscribeForBill(ElectricityBillPayment payment) {
        this.payment = payment;
    }

    void broadcaseBills() {
        payment.billGenerated();
        payment.payBillNotification();
    }
}

// Inf as child form 2 interfaces : Inheritance -> Inheritance between interfaces
interface Inf extends Notification, ElectricityBillPayment{

}

class User implements Inf{ //Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance

    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }

    @Override
    public void billGenerated() {
        System.out.println("[User] Your Electricity bill is Generated. please pay in time");
    }

    @Override
    public void payBillNotification() {
        System.out.println("[User] Thank you for paying your bill");
    }
}

public class Interfaces {

```

Step 5: How to link interfaces to classes and methods.

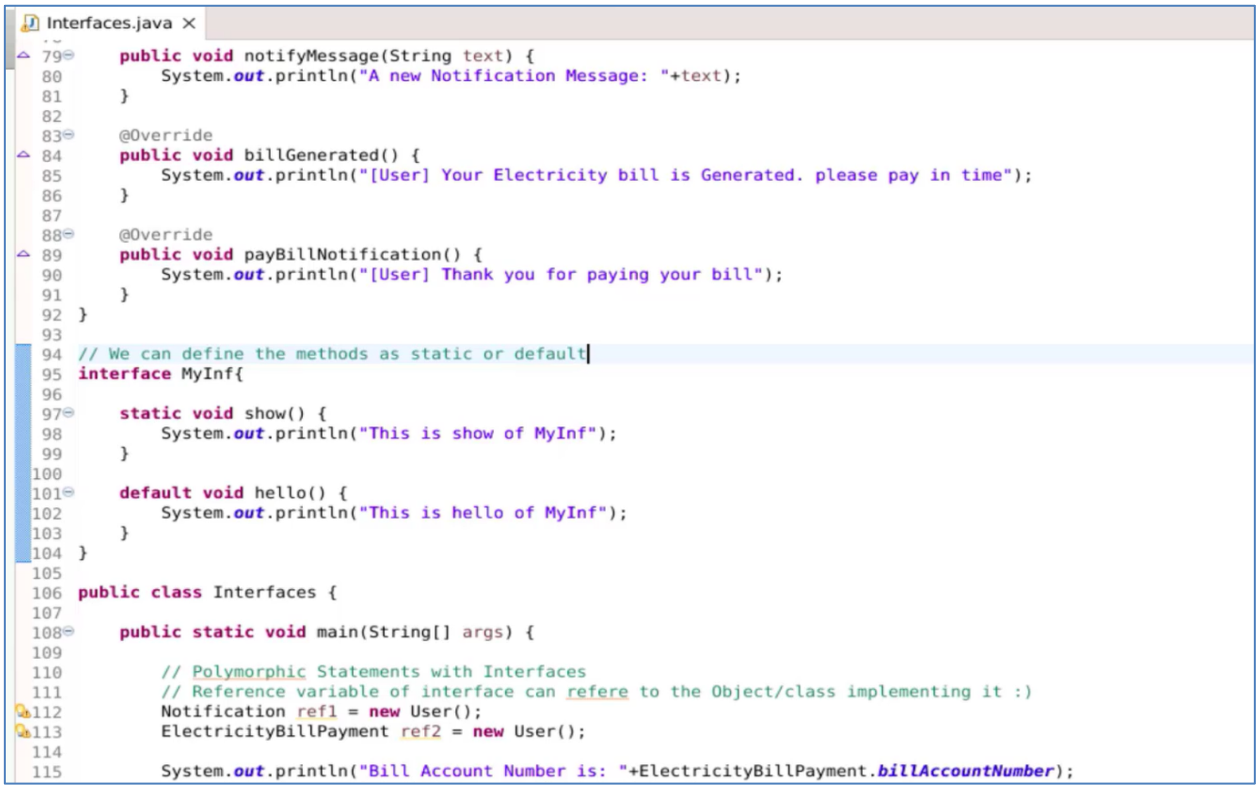
5.1 If you create an interface called My INF and define a method called **show** that prints out **This is show of my INF.** However, this resulted in an error because interfaces can only define method signatures, not implementations.



Interfaces.java

```java
    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }

    @Override
    public void billGenerated() {
        System.out.println("[User] Your Electricity bill is Generated. please pay in time");
    }

    @Override
    public void payBillNotification() {
        System.out.println("[User] Thank you for paying your bill");
    }
}

interface MyInf{
    void show() {
        System.out.println("This is show of MyInf");
    }
}

public class Interfaces {

    public static void main(String[] args) {

        // Polymorphic Statements with Interfaces
        // Reference variable of interface can refere to the Object/class implementing it :)
        Notification ref1 = new User();
        ElectricityBillPayment ref2 = new User();

        System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
        ElectricityApp electricityApp = new ElectricityApp();
        User john = new User();
        electricityApp.subscribeForBill(john);

        electricityApp.broadcaseBills();
```

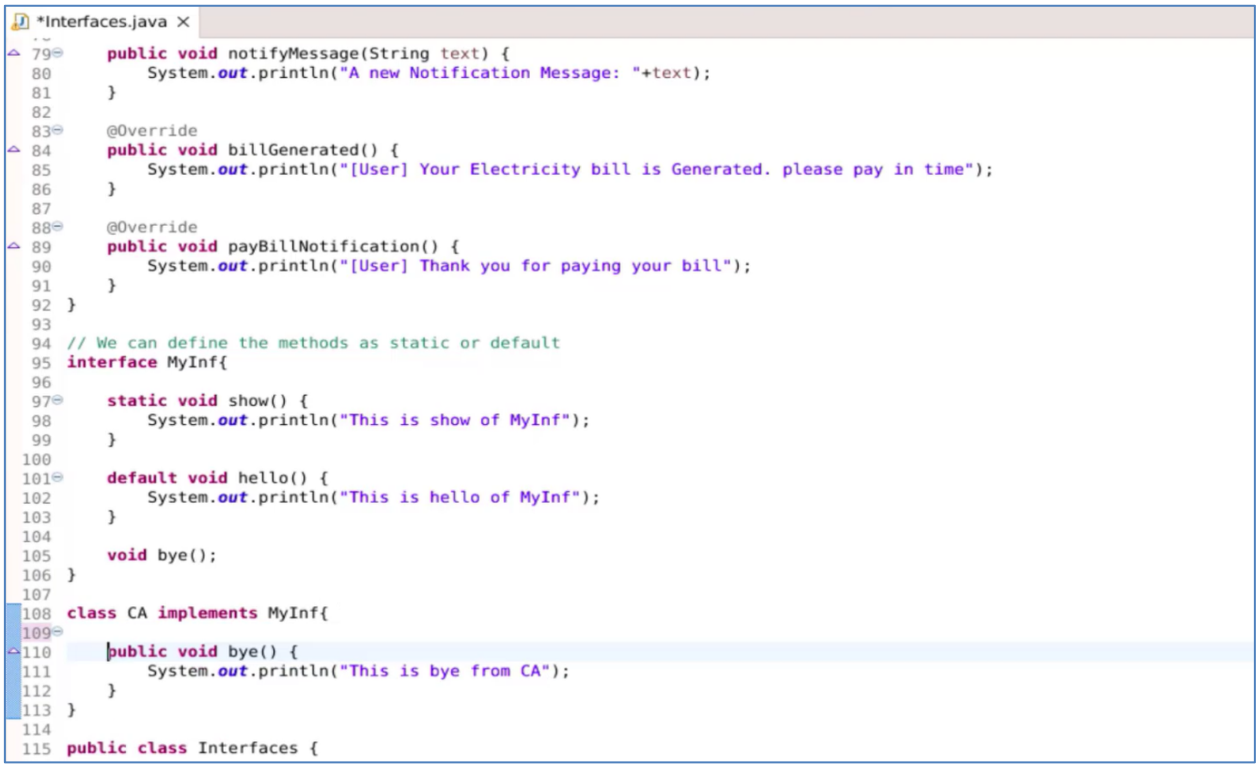
5.2 To define methods in an interface, there are two ways to do it. The first way is to create the method as static, which means that you mark the method as a static method inside your interface. This allows you to define the method. The second way is to create the method as default, which means that you add the keyword "default" in front of the method. This allows you to define the method as well. You cannot create non-static methods in an interface because there are no objects involved in an interface. Interfaces are purely for abstraction.



Interfaces.java

```java
    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }

    @Override
    public void billGenerated() {
        System.out.println("[User] Your Electricity bill is Generated. please pay in time");
    }

    @Override
    public void payBillNotification() {
        System.out.println("[User] Thank you for paying your bill");
    }
}

// We can define the methods as static or default
interface MyInf{

    static void show() {
        System.out.println("This is show of MyInf");
    }

    default void hello() {
        System.out.println("This is hello of MyInf");
    }
}

public class Interfaces {

    public static void main(String[] args) {

        // Polymorphic Statements with Interfaces
        // Reference variable of interface can refere to the Object/class implementing it :)
        Notification ref1 = new User();
        ElectricityBillPayment ref2 = new User();

        System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
```

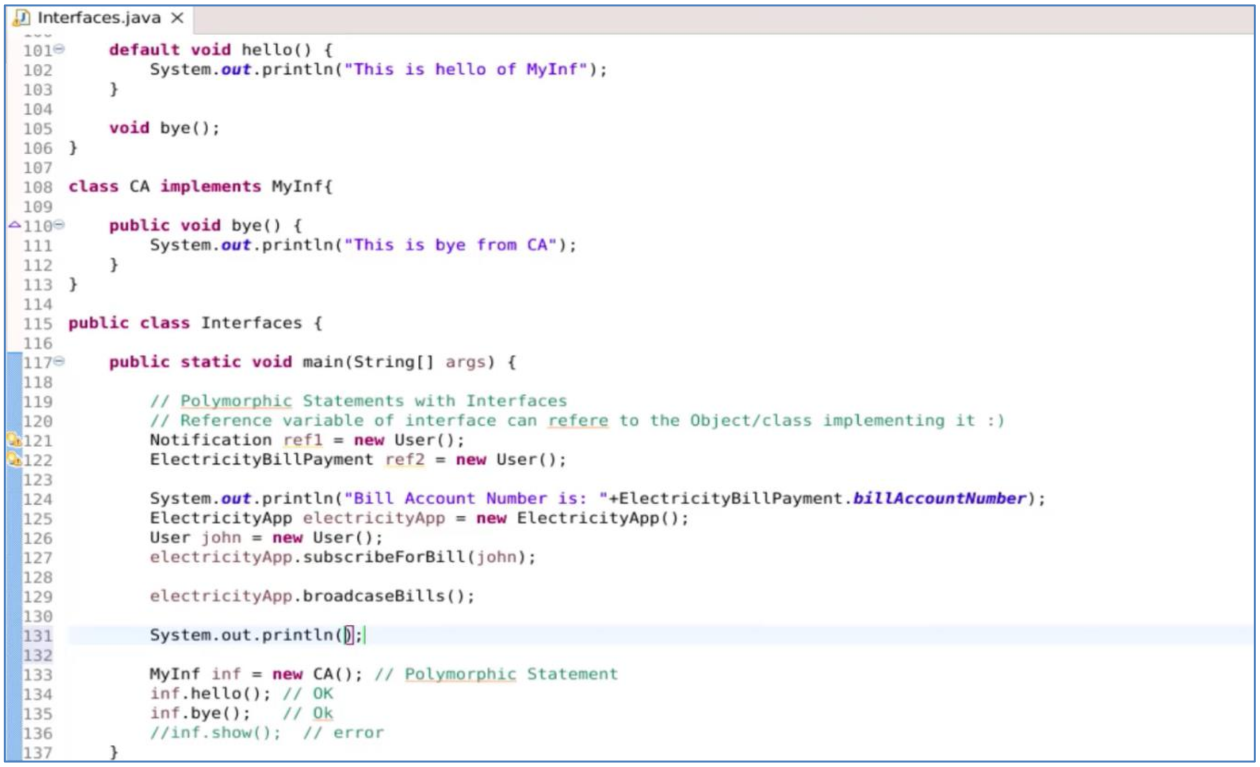
5.3 Created a class called **CA** that implements **My INF**. You also added a public and abstract method called **Bye** to **CA**. You can now print out **This is bye from CA**.



*Interfaces.java

```
    public void notifyMessage(String text) {
        System.out.println("A new Notification Message: "+text);
    }

    @Override
    public void billGenerated() {
        System.out.println("[User] Your Electricity bill is Generated. please pay in time");
    }

    @Override
    public void payBillNotification() {
        System.out.println("[User] Thank you for paying your bill");
    }
}

// We can define the methods as static or default
interface MyInf{

    static void show() {
        System.out.println("This is show of MyInf");
    }

    default void hello() {
        System.out.println("This is hello of MyInf");
    }

    void bye();
}

class CA implements MyInf{

    public void bye() {
        System.out.println("This is bye from CA");
    }
}

public class Interfaces {
```

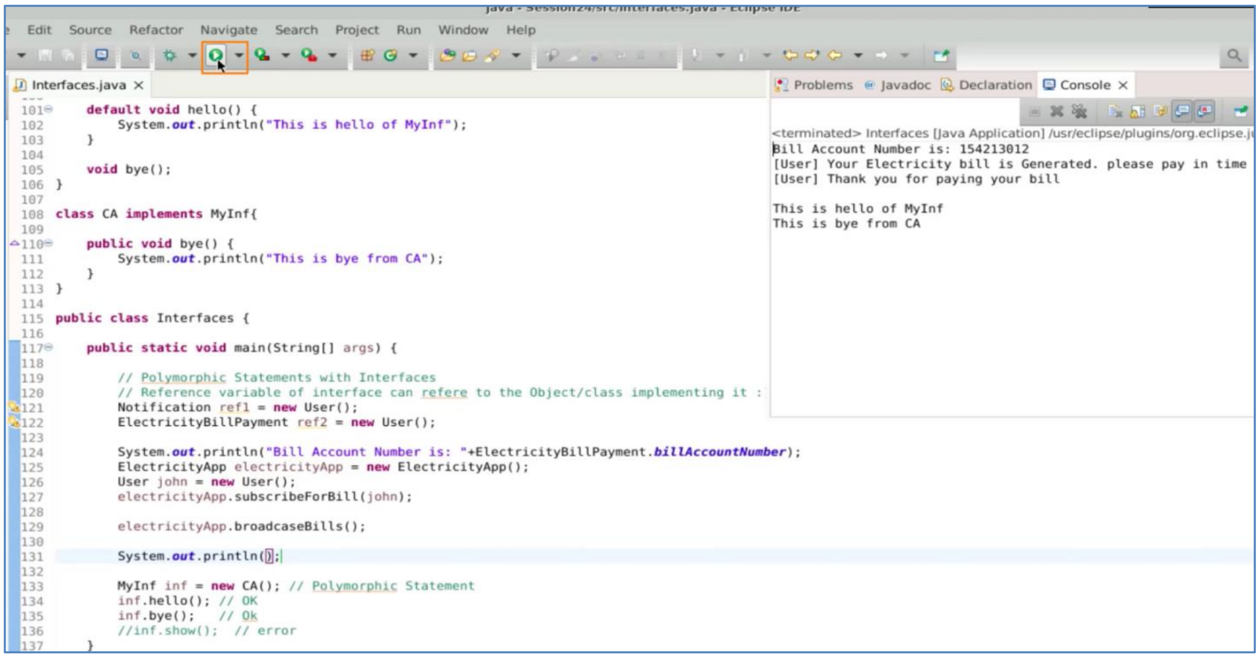
5.4 Define an interface reference variable called `INF` that can point to an object of `CA`. With `INF`, you can execute the `hello` and `bye` methods. However, you cannot execute the `show` method with the same `INF` reference variable because it is a static method. You can access default methods or abstract methods with the reference variable of the interface written as a polymorphic statement.



Interfaces.java

```
    default void hello() {
        System.out.println("This is hello of MyInf");
    }

    void bye();
}

class CA implements MyInf{

    public void bye() {
        System.out.println("This is bye from CA");
    }
}

public class Interfaces {

    public static void main(String[] args) {

        // Polymorphic Statements with Interfaces
        // Reference variable of interface can refere to the Object/class implementing it :)
        Notification ref1 = new User();
        ElectricityBillPayment ref2 = new User();

        System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
        ElectricityApp electricityApp = new ElectricityApp();
        User john = new User();
        electricityApp.subscribeForBill(john);

        electricityApp.broadcaseBills();

        System.out.println();

        MyInf inf = new CA(); // Polymorphic Statement
        inf.hello(); // OK
        inf.bye();   // Ok
        //inf.show();  // error
    }
```

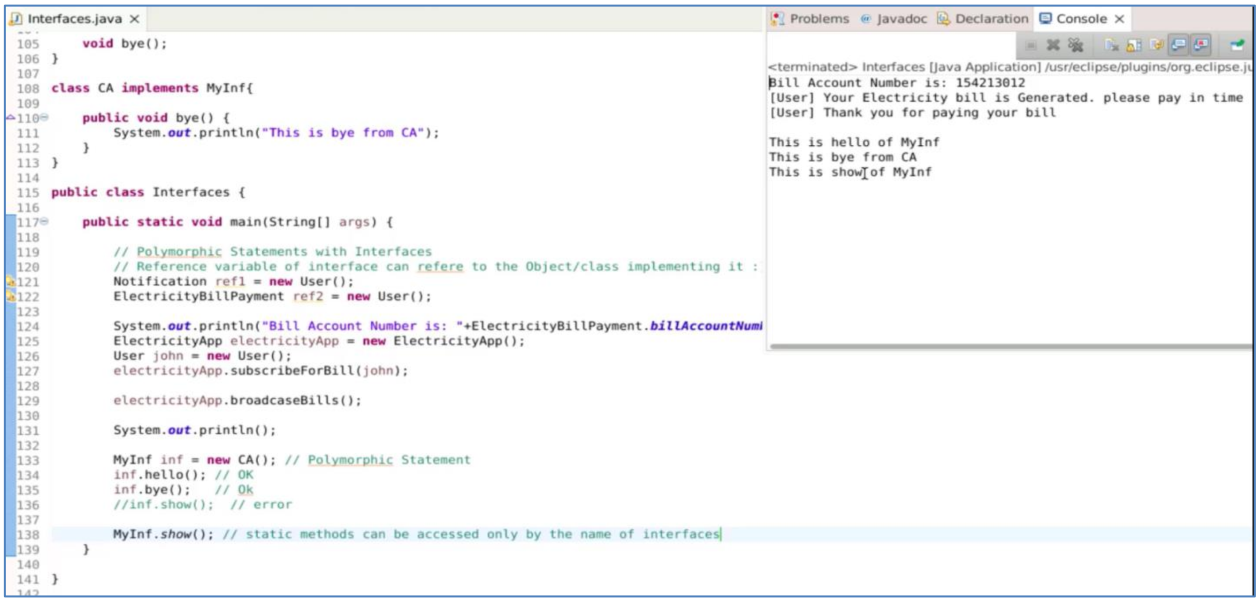
5.5 When you run the code, the console will display **Hello of my INF** followed by **Bye from CA**, which is accessible and clear. However, if you try to access the static method **show** by typing **INF.show()**, it will result in an error.


java - Session24/src/Interfaces.java - Eclipse IDE
Edit Source Refactor Navigate Search Project Run Window Help
Interfaces.java
Problems Javadoc Declaration Console

```
    default void hello() {
        System.out.println("This is hello of MyInf");
    }

    void bye();
}

class CA implements MyInf{

    public void bye() {
        System.out.println("This is bye from CA");
    }
}

public class Interfaces {

    public static void main(String[] args) {

        // Polymorphic Statements with Interfaces
        // Reference variable of interface can refere to the Object/class implementing it :
        Notification ref1 = new User();
        ElectricityBillPayment ref2 = new User();

        System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
        ElectricityApp electricityApp = new ElectricityApp();
        User john = new User();
        electricityApp.subscribeForBill(john);

        electricityApp.broadcaseBills();

        System.out.println();

        MyInf inf = new CA(); // Polymorphic Statement
        inf.hello(); // OK
        inf.bye();   // Ok
        //inf.show();  // error
    }
```

```
<terminated> Interfaces [Java Application] /usr/eclipse/plugins/org.eclipse.j
Bill Account Number is: 154213012
[User] Your Electricity bill is Generated. please pay in time
[User] Thank you for paying your bill

This is hello of MyInf
This is bye from CA
```

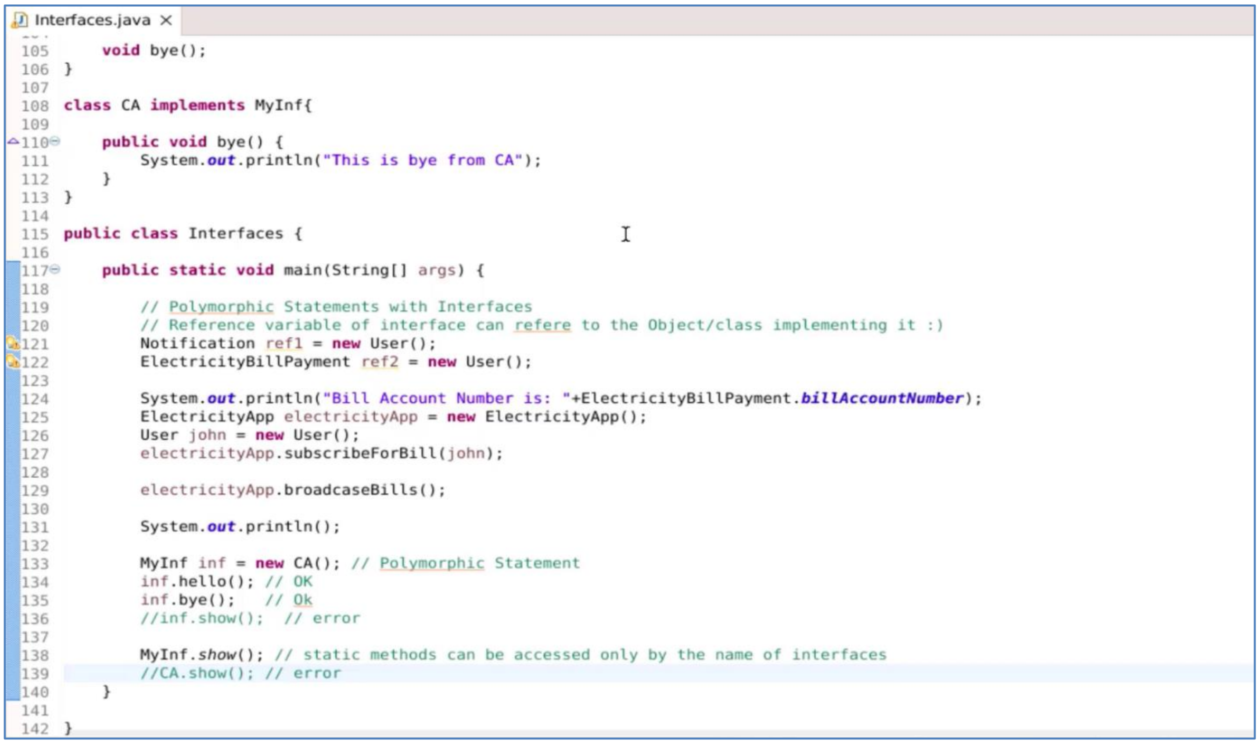
5.6 Static methods can only be accessed through the name of the interface, so the correct syntax would be **MyInf.show()**. When you run the code again it will display **show of myINF** in the console.



```
    void bye();
}

class CA implements MyInf{

    public void bye() {
        System.out.println("This is bye from CA");
    }
}

public class Interfaces {

    public static void main(String[] args) {

        // Polymorphic Statements with Interfaces
        // Reference variable of interface can refere to the Object/class implementing it :
        Notification ref1 = new User();
        ElectricityBillPayment ref2 = new User();

        System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNum
        ElectricityApp electricityApp = new ElectricityApp();
        User john = new User();
        electricityApp.subscribeForBill(john);

        electricityApp.broadcaseBills();

        System.out.println();

        MyInf inf = new CA(); // Polymorphic Statement
        inf.hello(); // OK
        inf.bye();   // Ok
        //inf.show();  // error

        MyInf.show(); // static methods can be accessed only by the name of interfaces
    }

}
```

```
<terminated> Interfaces [Java Application] /usr/eclipse/plugins/org.eclipse.j
Bill Account Number is: 154213012
[User] Your Electricity bill is Generated. please pay in time
[User] Thank you for paying your bill

This is hello of MyInf
This is bye from CA
This is show of MyInf
```

5.7 On the other hand, attempting to access the method **show** using **CA.show()** will result in an error. This is because an interface is not a parent or superclass of a class that implements it, but rather a set of rules or guidelines for implementation.



Interfaces.java

```
    void bye();
}

class CA implements MyInf{

    public void bye() {
        System.out.println("This is bye from CA");
    }
}

public class Interfaces {

    public static void main(String[] args) {

        // Polymorphic Statements with Interfaces
        // Reference variable of interface can refere to the Object/class implementing it :)
        Notification ref1 = new User();
        ElectricityBillPayment ref2 = new User();

        System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
        ElectricityApp electricityApp = new ElectricityApp();
        User john = new User();
        electricityApp.subscribeForBill(john);

        electricityApp.broadcaseBills();

        System.out.println();

        MyInf inf = new CA(); // Polymorphic Statement
        inf.hello(); // OK
        inf.bye();   // Ok
        //inf.show();  // error

        MyInf.show(); // static methods can be accessed only by the name of interfaces
        //CA.show(); // error
    }

}
```

By using the following steps, you have successfully created multiple interfaces and implemented their abstract methods, allowing for modular, flexible, and reusable code structures in your application.