

Lesson 04 Demo 01

Creating Interfaces and Multiple Implementation

Objective: To create multiple Interfaces

Tools required: Eclipse IDE

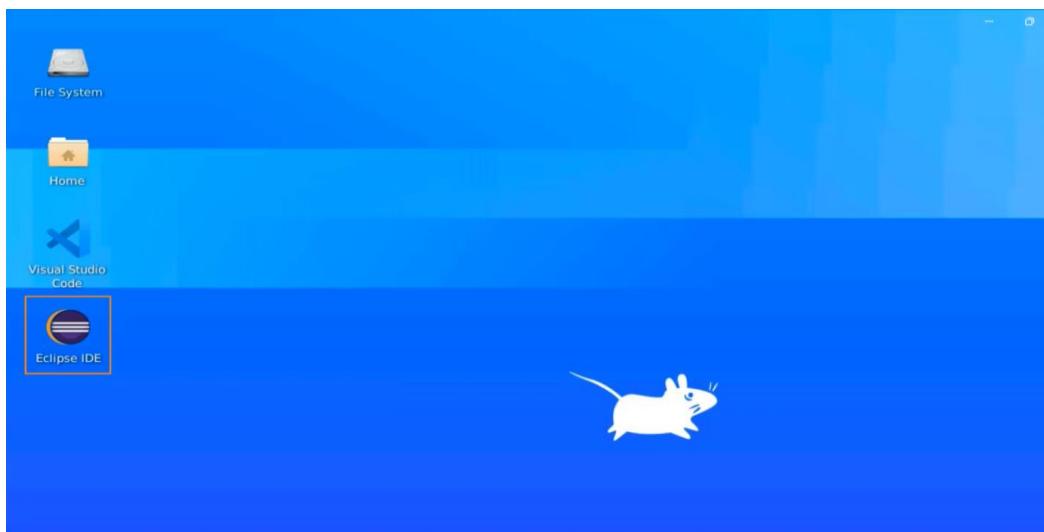
Prerequisites: None

Steps to be followed:

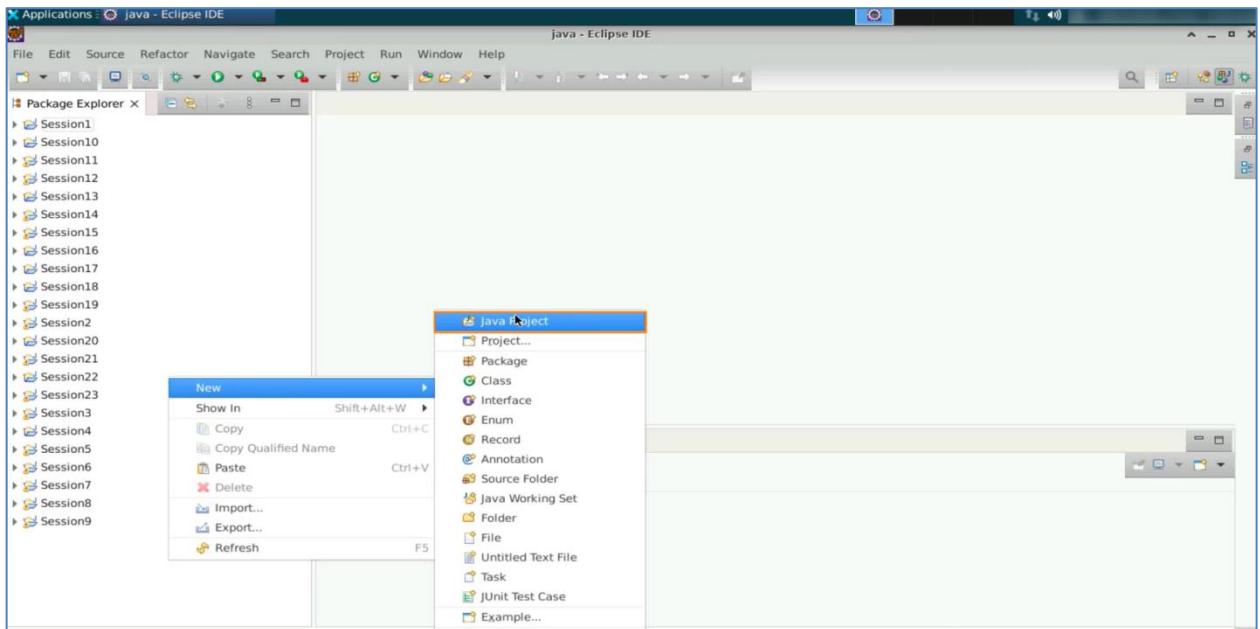
1. Implement the concept of interfaces
2. comprehend how interfaces work
3. Write polymorphic statements using interfaces
4. Implement how multiple inheritances work in interfaces
5. Link interfaces to classes and methods

Step 1: Implement the concept of interfaces

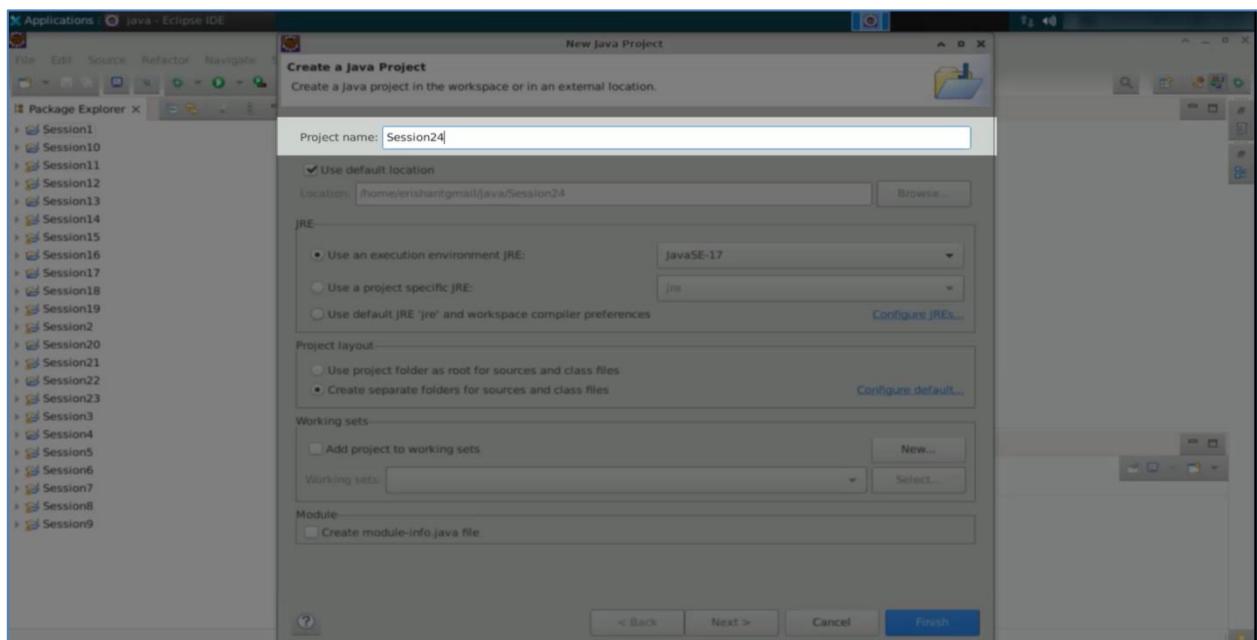
1.1 Open the Eclipse IDE



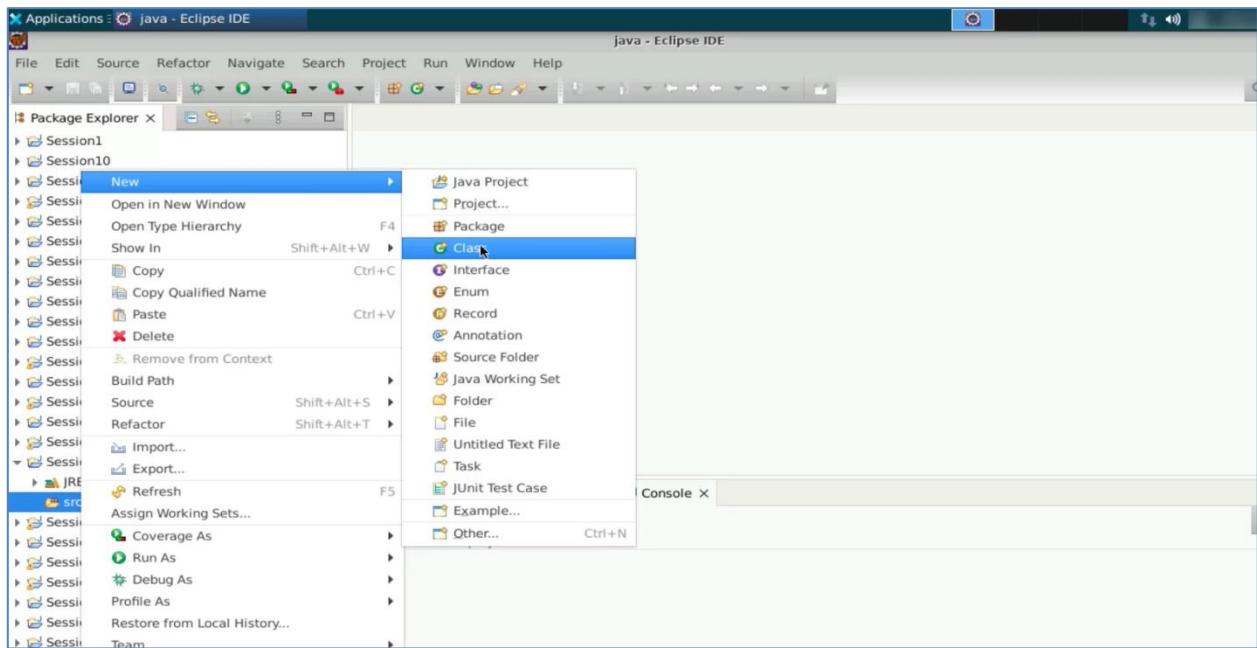
1.2 Select File, then New, and then Java project



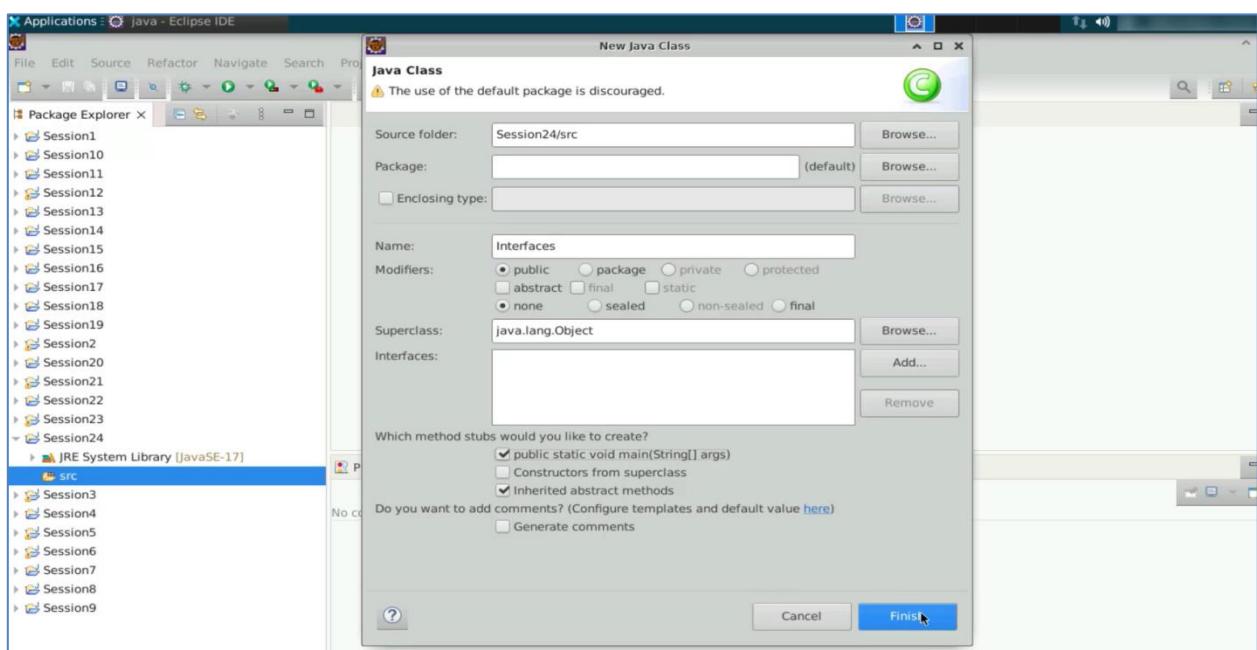
1.3 Name the project “Session24” and press Finish



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**



- 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 ×
1 abstract class PayTMPaymentGateway{
2
3 }
4
5 abstract class RazorPay{
6
7 }
8
9 abstract class PayPal{
10
11 }
12
13
14
15
16 public class Interfaces {
17
18     public static void main(String[] args) {
19
20
21     }
22
23 }
24
```

- 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 ×
1 abstract class PayTMPaymentGateway{
2
3 }
4
5 abstract class RazorPay{
6
7 }
8
9 abstract class PayPal{
10
11 }
12
13 // Multiple extends not supported by Java|
14 class MyPaymentPage extends PayTMPaymentGateway, RazorPay{
15
16 }
17
18 public class Interfaces {
19
20     public static void main(String[] args) {
21
22
23     }
24
25 }
26
```

- 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.

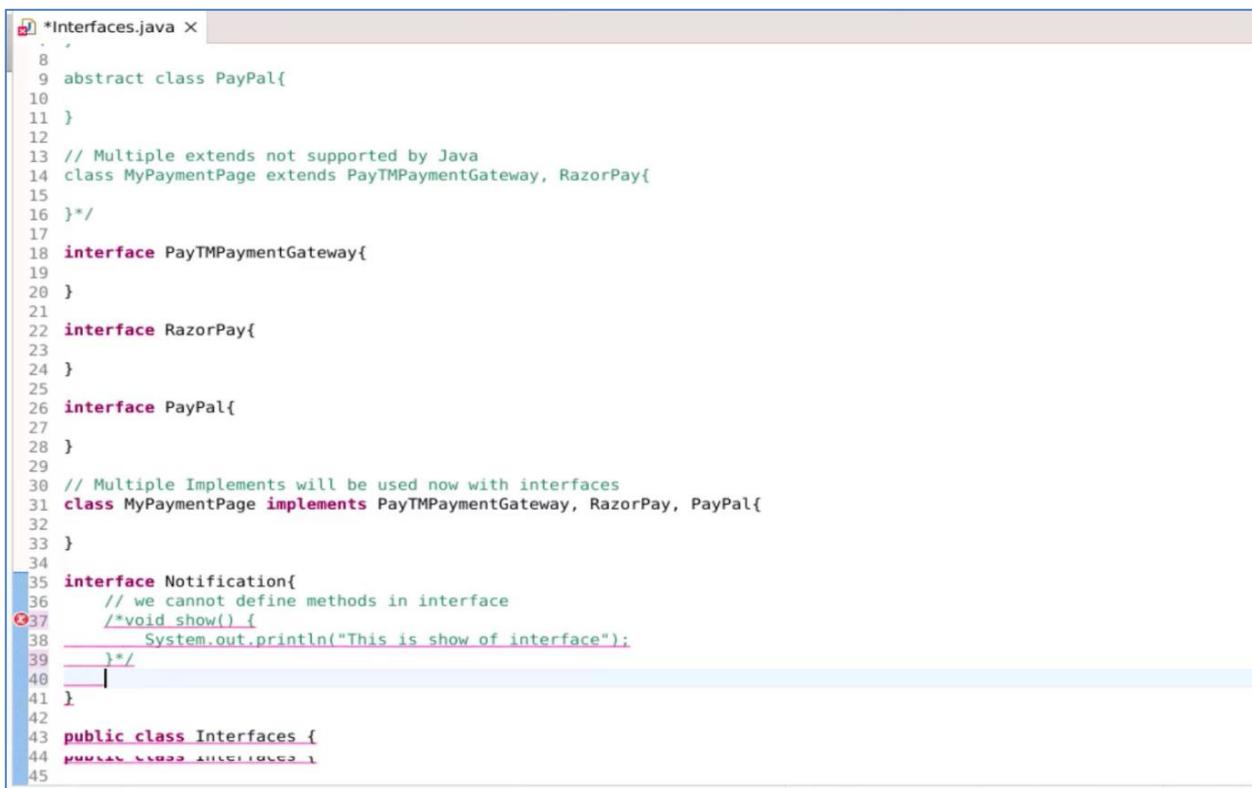
```
Interfaces.java X
1/*abstract class PayTMPaymentGateway{
2
3}
4
5 abstract class RazorPay{
6
7}
8
9 abstract class PayPal{
10
11}
12
13// Multiple extends not supported by Java
14class MyPaymentPage extends PayTMPaymentGateway, RazorPay{
15
16}*/
17
18interface PayTMPaymentGateway{
19
20}
21
22interface RazorPay{
23
24}
25
26interface PayPal{
27
28}
29
30// Multiple extends ot supported by Java
31class MyPaymentPage extends PayTMPaymentGateway, RazorPay{
32
33}
34
35public class Interfaces {
36
37    public static void main(String[] args) {
38
```

- 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.

```
1 Interfaces.java X
2 /*abstract class PayTMPaymentGateway{
3 }
4
5 abstract class RazorPay{
6 }
7
8 abstract class PayPal{
9 }
10
11 */
12
13 // Multiple extends not supported by Java
14 class MyPaymentPage extends PayTMPaymentGateway, RazorPay{
15
16 }*/
17
18 interface PayTMPaymentGateway{
19 }
20
21 interface RazorPay{
22 }
23
24
25 interface PayPal{
26 }
27
28
29
30 // Multiple Implements will be used now with interfaces
31 class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{
32
33 }
34
35 public class Interfaces {
36
37     public static void main(String[] args) {
38 }
```

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 X
8
9 abstract class PayPal{
10
11 }
12
13 // Multiple extends not supported by Java
14 class MyPaymentPage extends PayTMPaymentGateway, RazorPay{
15
16 */
17
18 interface PayTMPaymentGateway{
19
20 }
21
22 interface RazorPay{
23
24 }
25
26 interface PayPal{
27
28 }
29
30 // Multiple Implements will be used now with interfaces
31 class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{
32
33 }
34
35 interface Notification{
36     // we cannot define methods in interface
37     /*void show(){
38         System.out.println("This is show of interface");
39     }*/
40 }
41
42
43 public class Interfaces {
44     public static void main(String[] args) {
45 }
```

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.'

```
1 *Interfaces.java X
2
3
4
5
6
7
8  interface PayTMPaymentGateway{
9
10 }
11
12  interface RazorPay{
13
14 }
15
16  interface PayPal{
17
18 }
19
20
21 // Multiple Implements will be used now with interfaces
22 class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{
23
24 }
25
26
27
28
29
30
31
32
33
34
35  interface Notification{
36      // we cannot define methods in interface
37      /*void show() {
38          System.out.println("This is show of interface");
39      }*/
40
41      // We cannot have constructors in Interface
42      /*Notification(){
43
44      }*/
45
46
47 }
48
49 public class Interfaces {
50
51     public static void main(String[] args) {
52
53 }
```

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.

```
1 Interfaces.java X
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18 interface PayTMPaymentGateway{
19
20 }
21
22 interface RazorPay{
23
24 }
25
26 interface PayPal{
27
28 }
29
30 // Multiple Implements will be used now with interfaces
31 class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{
32
33 }
34
35 interface Notification{
36     // we cannot define methods in interface
37     /*void show() {
38         System.out.println("This is show of interface");
39     }*/
40
41     // We cannot have constructors in Interface
42     /*Notification(){
43
44     }*/
45
46     // public abstract void notifyMessage(String text)
47     void notifyMessage(String text); // public abstract -> method by default
48 }
49
50 public class Interfaces {
51
52     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

```
21 // Interfaces.java X
22 interface RazorPay{
23 }
24
25 interface PayPal{
26 }
27
28 // Multiple Implements will be used now with interfaces
29 class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{
30 }
31
32
33 }
34
35 interface Notification{
36     // we cannot define methods in interface
37     /*void show() {
38         System.out.println("This is show of interface");
39     }*/
40
41     // We cannot have constructors in Interface
42     /*Notification(){
43
44     }*/
45
46     // public abstract void notifyMessage(String text)
47     void notifyMessage(String text); // public abstract -> method by default
48 }
49
50 class User implements Notification{ // this is implementation of interface and not inheritance
51     I
52 }
53
54 public class Interfaces {
55
56     public static void main(String[] args) {
57
58 }
```

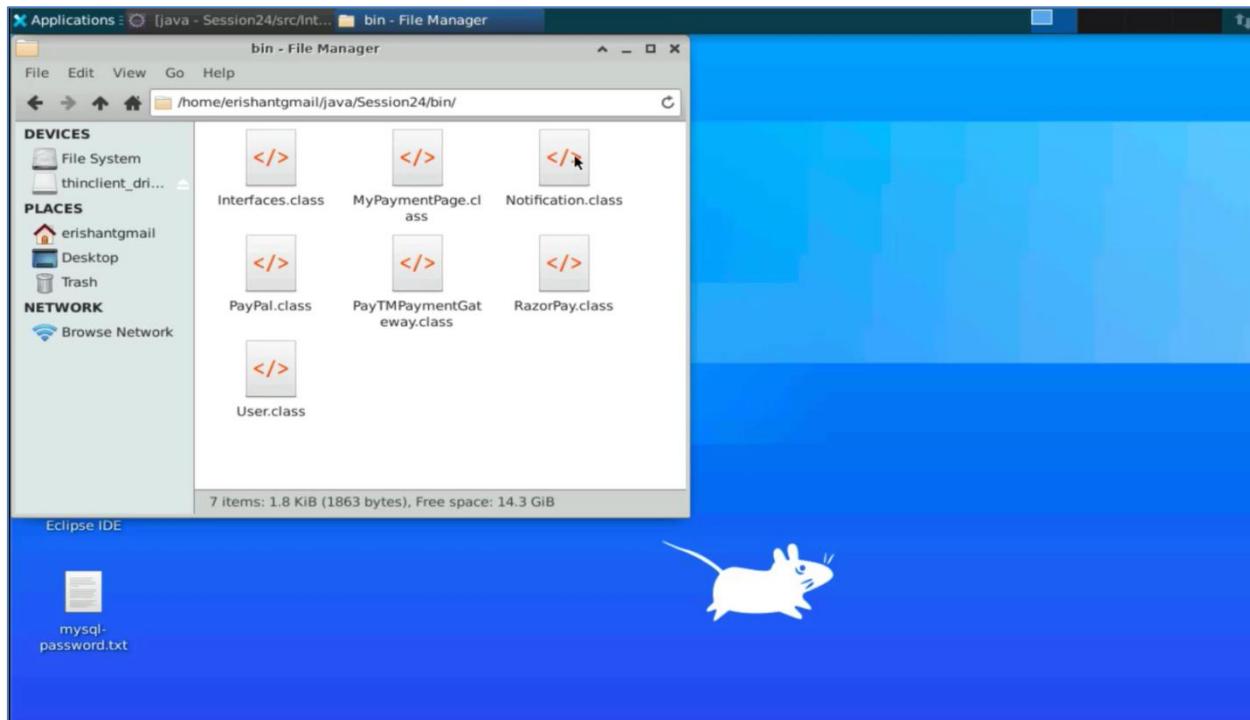
2.5 Interfaces contain abstract methods that must be defined by classes that implement the interface. The methods defined in the implementing classes must be declared as public, since the abstract methods in the interface are public by default. The user can print a new notification message with any text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

```
1  Interfaces.java X
21
22  interface RazorPay{
23
24  }
25
26  interface PayPal{
27
28  }
29
30 // Multiple Implements will be used now with interfaces
31 class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{
32
33 }
34
35 interface Notification{
36     // we cannot define methods in interface
37     /*void show() {
38         System.out.println("This is show of interface");
39     }*/
40
41     // We cannot have constructors in Interface
42     /*Notification(){
43
44     }*/
45
46     // public abstract void notifyMessage(String text)
47     void notifyMessage(String text); // public abstract -> method by default
48 }
49
50 class User implements Notification{ // this is implementation of interface and not inheritance
51
52     public void notifyMessage(String text) {
53
54     }
55 }
56
57 public class Interfaces {
58 }
```

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 X
21      I
22  interface RazorPay{
23
25
26  interface PayPal{
27
28 }
29
30 // Multiple Implements will be used now with interfaces
31 class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{
32
33 }
34
35 interface Notification{
36     // we cannot define methods in interface
37     /*void show() {
38         System.out.println("This is show of interface");
39     }*/
40
41     // We cannot have constructors in Interface
42     /*Notification(){
43
44     }*/
45
46     // public abstract void notifyMessage(String text)
47     void notifyMessage(String text); // public abstract -> method by default
48 }
49
50 class User implements Notification{ // this is implementation of interface and not inheritance
51
52     public void notifyMessage(String text) {
53         System.out.println("A new Notification Message: "+text);
54     }
55 }
56
57 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()

```
 25 *Interfaces.java X
26 
27 interface PayPal{
28 }
29
30 // Multiple Implements will be used now with interfaces
31 class MyPaymentPage implements PayTMPaymentGateway, RazorPay, PayPal{
32 }
33 }
34
35 interface Notification{
36     // we cannot define methods in interface
37     /*void show() {
38         System.out.println("This is show of interface");
39     }*/
40
41     // We cannot have constructors in Interface
42     /*Notification(){
43
44     }*/
45
46     // public abstract void notifyMessage(String text)
47     void notifyMessage(String text); // public abstract -> method by default
48 }
49
50 interface ElectricityBillPayment{
51     void billGenerated();
52     void payBillNotification();
53 }
54
55 class User implements Notification{ // this is implementation of interface and not inheritance
56
57     public void notifyMessage(String text) {
58         System.out.println("A new Notification Message: "+text);
59     }
60 }
61
62 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 X
39     }*/
40
41     // We cannot have constructors in Interface
42     /*Notification(){
43
44     }*/
45
46     // public abstract void notifyMessage(String text)
47     void notifyMessage(String text); // public abstract -> method by default
48 }
49
50 interface ElectricityBillPayment{
51     void billGenerated();
52     void payBillNotification();
53 }
54
55 class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance
56
57     public void notifyMessage(String text) {
58         System.out.println("A new Notification Message: "+text);
59     }
60
61     @Override
62     public void billGenerated() {
63
64     }
65
66
67     @Override
68     public void payBillNotification() {
69
70     }
71 }
72
73 public class Interfaces {
74
75     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 X
39     }/*
40
41     // We cannot have constructors in Interface
42     /*Notification(){
43
44     }*/
45
46     // public abstract void notifyMessage(String text)
47     void notifyMessage(String text); // public abstract -> method by default
48 }
49
50 interface ElectricityBillPayment{
51     void billGenerated();
52     void payBillNotification();
53 }
54
55 class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance
56
57     public void notifyMessage(String text) {
58         System.out.println("A new Notification Message: "+text);
59     }
60
61     @Override
62     public void billGenerated() {
63         System.out.println("Your Electricity bill is Generated. please pay in time");
64     }
65
66     @Override
67     public void payBillNotification() {
68         System.out.println("Thank you for payinh your bill");
69     }
70 }
71
72 public class Interfaces {
73
74     public static void main(String[] args) {
75 }
```

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 X
39     }*/
40
41     // We cannot have constructors in Interface
42     /*Notification(){
43
44     }*/
45
46     // public abstract void notifyMessage(String text)
47     void notifyMessage(String text); // public abstract -> method by default
48 }
49
50 interface ElectricityBillPayment{
51
52     int billAccountNumber = 154213012; // by default -> public static final int billAccountNumber = 154213012;
53
54     void billGenerated();
55     void payBillNotification();
56 }
57
58 class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance
59
60     public void notifyMessage(String text) {
61         System.out.println("A new Notification Message: "+text);
62     }
63
64     @Override
65     public void billGenerated() {
66         System.out.println("Your Electricity bill is Generated. please pay in time");
67     }
68
69     @Override
70     public void payBillNotification() {
71         System.out.println("Thank you for payinh your bill");
72     }
73 }
74
75 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 X
1  /*
2  * This Java code demonstrates the use of static variables in interfaces.
3  */
4
5  public class Interfaces {
6      public static void main(String[] args) {
7          System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
8      }
9  }
10
11 interface ElectricityBillPayment{
12     // public abstract void notifyMessage(String text)
13     void notifyMessage(String text); // public abstract -> method by default
14 }
15
16     int billAccountNumber = 154213012; // by default -> public static final int billAccountNumber = 154213012;
17
18     void billGenerated();
19     void payBillNotification();
20 }
21
22 class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance
23
24     public void notifyMessage(String text) {
25         System.out.println("A new Notification Message: "+text);
26     }
27
28     @Override
29     public void billGenerated() {
30         System.out.println("Your Electricity bill is Generated. please pay in time");
31     }
32
33     @Override
34     public void payBillNotification() {
35         System.out.println("Thank you for payinh your bill");
36     }
37 }
38
39 public class Interfaces {
40
41     public static void main(String[] args) {
42         System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
43     }
44 }
```

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.

```
1 Interfaces.java X
50     // public abstract void notifyMessage(String text)
51     void notifyMessage(String text); // public abstract -> method by default
52 }
53
54 interface ElectricityBillPayment{
55
56     int billAccountNumber = 154213012; // by default -> public static final int billAccountNumber = 154213012;
57
58     void billGenerated();
59     void payBillNotification();
60 }
61
62 class ElectricityApp{
63
64     ElectricityBillPayment payment;
65
66     void subscribeForBill(ElectricityBillPayment payment) {
67         this.payment = payment;
68     }
69 }
70
71 class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance
72
73     public void notifyMessage(String text) {
74         System.out.println("A new Notification Message: "+text);
75     }
76
77     @Override
78     public void billGenerated() {
79         System.out.println("Your Electricity bill is Generated. please pay in time");
80     }
81
82     @Override
83     public void payBillNotification() {
84         System.out.println("Thank you for payinh your bill");
85     }
86 }
```

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.

```
Interfaces.java X
56 }
57
58 class ElectricityApp{
59
60     ElectricityBillPayment payment;
61
62     void subscribeForBill(ElectricityBillPayment payment) {
63         this.payment = payment;
64     }
65 }
66
67 class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance
68
69     public void notifyMessage(String text) {
70         System.out.println("A new Notification Message: "+text);
71     }
72
73     @Override
74     public void billGenerated() {
75         System.out.println("Your Electricity bill is Generated. please pay in time");
76     }
77
78     @Override
79     public void payBillNotification() {
80         System.out.println("Thank you for payinh your bill");
81     }
82 }
83
84 public class Interfaces {
85
86     public static void main(String[] args) {
87
88         System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
89         ElectricityApp electricityApp = new ElectricityApp();
90         User john = new User();
91         electricityApp.subscribeForBill(john);
92     }
}
```

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 X
  63     this.payment = payment;
  64   }
  65 }
  66
  67 class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance
  68
  69@    public void notifyMessage(String text) {
  70        System.out.println("A new Notification Message: "+text);
  71    }
  72
  73@    @Override
  74    public void billGenerated() {
  75        System.out.println("Your Electricity bill is Generated. please pay in time");
  76    }
  77
  78@    @Override
  79    public void payBillNotification() {
  80        System.out.println("Thank you for payinh your bill");
  81    }
  82 }
  83
  84 public class Interfaces {
  85
  86@    public static void main(String[] args) {
  87
  88        // Polymorphic Statements with Interfaces
  89        // Reference variable of interface can refere to the Object/class implementing it :)
  90        Notification refl = new User();
  91        ElectricityBillPayment ref2 = new User();
  92
  93        System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
  94        ElectricityApp electricityApp = new ElectricityApp();
  95        User john = new User();
  96        electricityApp.subscribeForBill(john);
  97    }
  98 }
  99 }
```

- 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 X
54     void billGenerated();
55     void payBillNotification();
56 }
57
58 class ElectricityApp{
59
60     ElectricityBillPayment payment;
61
62     void subscribeForBill(ElectricityBillPayment payment) {
63         this.payment = payment;
64     }
65
66     void broadcastBills() {
67         payment.billGenerated();
68         payment.payBillNotification();
69     }
70 }
71
72 class User implements Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance
73
74     public void notifyMessage(String text) {
75         System.out.println("A new Notification Message: "+text);
76     }
77
78     @Override
79     public void billGenerated() {
80         System.out.println("Your Electricity bill is Generated. please pay in time");
81     }
82
83     @Override
84     public void payBillNotification() {
85         System.out.println("Thank you for payinh your bill");
86     }
87 }
88
89 public class Interfaces {
90
91     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.

The screenshot shows the Eclipse IDE interface with the following details:

- Left Panel (Code Editor):** Displays the `Interfaces.java` file. The code defines a class `User` that implements the `Notification` and `ElectricityBillPayment` interfaces. It overrides methods `notifyMessage`, `billGenerated`, and `payBillNotification`. The `main` method creates an instance of `User` and calls `broadcastBills`.
- Right Panel (Console):** Shows the terminal output of the application. It includes:
 - The bill account number: `Bill Account Number is: 154213012`
 - A notification message: `[User] Your Electricity bill is Generated. please pay in time`
 - A thank you message: `[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 X
59
60     ElectricityBillPayment payment;
61
62    void subscribeForBill(ElectricityBillPayment payment) {
63        this.payment = payment;
64    }
65
66    void broadcastBills() {
67        payment.billGenerated();
68        payment.payBillNotification();
69    }
70 }
71
72 // Inf as child form 2 interfaces : Inheritance -> Inheritance between interfaces
73 interface Inf extends INotification, ElectricityBillPayment{
74
75 }
76
77 class User implements Inf{ //Notification, ElectricityBillPayment{ // this is implementation of interface and not inheritance
78
79    public void notifyMessage(String text) {
80        System.out.println("A new Notification Message: "+text);
81    }
82
83    @Override
84    public void billGenerated() {
85        System.out.println("[User] Your Electricity bill is Generated. please pay in time");
86    }
87
88    @Override
89    public void payBillNotification() {
90        System.out.println("[User] Thank you for paying your bill");
91    }
92 }
93
94 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 X
1 public void notifyMessage(String text) {
2     System.out.println("A new Notification Message: "+text);
3 }
4
5 @Override
6 public void billGenerated() {
7     System.out.println("[User] Your Electricity bill is Generated. please pay in time");
8 }
9
10 @Override
11 public void payBillNotification() {
12     System.out.println("[User] Thank you for paying your bill");
13 }
14
15 interface MyInf{
16     void show();
17     System.out.println("This is show of MyInf");
18 }
19
20 public class Interfaces {
21
22     public static void main(String[] args) {
23
24         // Polymorphic Statements with Interfaces
25         // Reference variable of interface can refere to the Object/class implementing it :)
26         Notification ref1 = new User();
27         ElectricityBillPayment ref2 = new User();
28
29         System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
30         ElectricityApp electricityApp = new ElectricityApp();
31         User john = new User();
32         electricityApp.subscribeForBill(john);
33
34         electricityApp.broadcastBills();
35     }
36 }
```

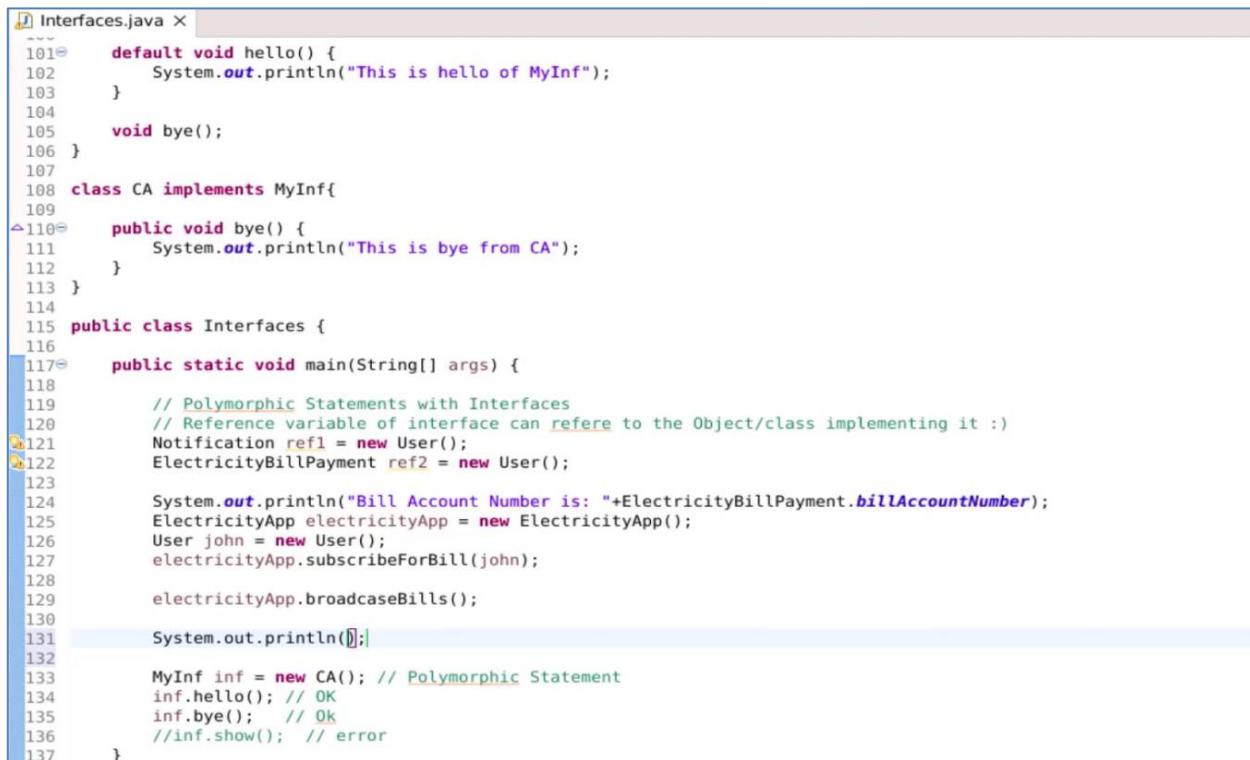
5.2 To define methods in an interface, there are two ways to do it. The first way is to create the method as static, which means that you mark the method as a static method inside your interface. This allows you to define the method. The second way is to create the method as default, which means that 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 X
1  public void notifyMessage(String text) {
2      System.out.println("A new Notification Message: "+text);
3  }
4
5  @Override
6  public void billGenerated() {
7      System.out.println("[User] Your Electricity bill is Generated. please pay in time");
8  }
9
10 @Override
11 public void payBillNotification() {
12     System.out.println("[User] Thank you for paying your bill");
13 }
14
15 // We can define the methods as static or default
16 interface MyInf{
17
18     static void show() {
19         System.out.println("This is show of MyInf");
20     }
21
22     default void hello() {
23         System.out.println("This is hello of MyInf");
24     }
25 }
26
27 public class Interfaces {
28
29     public static void main(String[] args) {
30
31         // Polymorphic Statements with Interfaces
32         // Reference variable of interface can refer to the Object/class implementing it :)
33         Notification ref1 = new User();
34         ElectricityBillPayment ref2 = new User();
35
36         System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
37     }
38 }
```

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 X
1  package com.simplilearn;
2
3  public interface MyInf {
4      public void notifyMessage(String text) {
5          System.out.println("A new Notification Message: "+text);
6      }
7
8      @Override
9      public void billGenerated() {
10         System.out.println("[User] Your Electricity bill is Generated. please pay in time");
11     }
12
13     @Override
14     public void payBillNotification() {
15         System.out.println("[User] Thank you for paying your bill");
16     }
17 }
18
19 // We can define the methods as static or default
20 interface MyInf{
21
22     static void show() {
23         System.out.println("This is show of MyInf");
24     }
25
26     default void hello() {
27         System.out.println("This is hello of MyInf");
28     }
29
30     void bye();
31 }
32
33 class CA implements MyInf{
34
35     @Override
36     public void bye() {
37         System.out.println("This is bye from CA");
38     }
39 }
40
41 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.



```
1 Interfaces.java X
2
3 101  default void hello() {
4      System.out.println("This is hello of MyInf");
5  }
6
7 105  void bye();
8 }
9
108 class CA implements MyInf{
109
110  public void bye() {
111      System.out.println("This is bye from CA");
112  }
113 }
114
115 public class Interfaces {
116
117  public static void main(String[] args) {
118
119      // Polymorphic Statements with Interfaces
120      // Reference variable of interface can refer to the Object/class implementing it :
121      Notification ref1 = new User();
122      ElectricityBillPayment ref2 = new User();
123
124      System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
125      ElectricityApp electricityApp = new ElectricityApp();
126      User john = new User();
127      electricityApp.subscribeForBill(john);
128
129      electricityApp.broadcastBills();
130
131      System.out.println();
132
133      MyInf inf = new CA(); // Polymorphic Statement
134      inf.hello(); // OK
135      inf.bye(); // OK
136      //inf.show(); // error
137  }
138 }
```

- 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.

The screenshot shows the Eclipse IDE interface with the Java editor open to a file named `Interfaces.java`. The code implements a `MyInf` interface with `hello()` and `bye()` methods. It also contains a `main` method that creates a `CA` object and prints various messages to the console. The console output shows the execution of the code, including the output of `hello()` and `bye()`.

```

101  default void hello() {
102      System.out.println("This is hello of MyInf");
103  }
104
105  void bye();
106 }
107
108 class CA implements MyInf{
109
110     public void bye() {
111         System.out.println("This is bye from CA");
112     }
113 }
114
115 public class Interfaces {
116
117     public static void main(String[] args) {
118
119         // Polymorphic Statements with Interfaces
120         // Reference variable of interface can refer to the Object/class implementing it :
121         Notification ref1 = new User();
122         ElectricityBillPayment ref2 = new User();
123
124         System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
125         ElectricityApp electricityApp = new ElectricityApp();
126         User john = new User();
127         electricityApp.subscribeForBill(john);
128
129         electricityApp.broadcastBills();
130
131         System.out.println();
132
133         MyInf inf = new CA(); // Polymorphic Statement
134         inf.hello(); // OK
135         inf.bye(); // Ok
136         //inf.show(); // error
137     }
138 }

```

- 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.

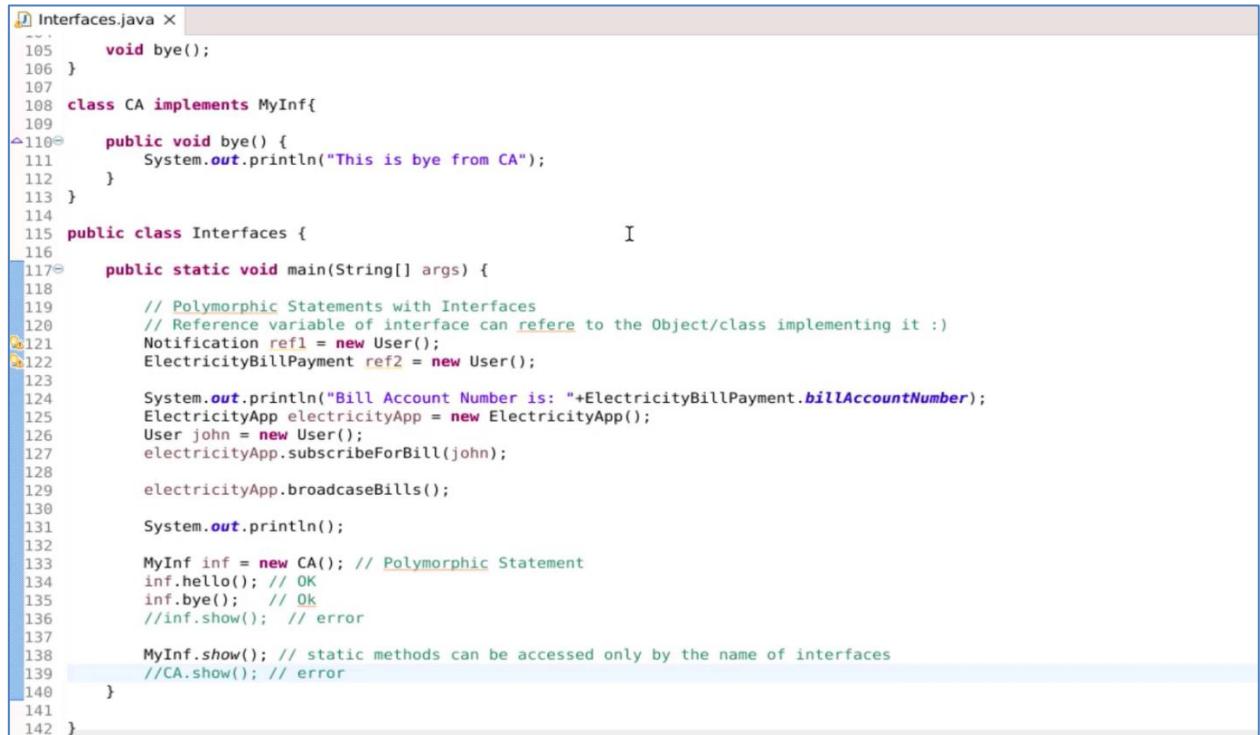
The screenshot shows the Eclipse IDE interface with the Java editor open to the same `Interfaces.java` file. A line of code has been added to the `main` method: `MyInf.show();`. This is highlighted in blue, indicating it is a syntax error. The console output shows the execution of the code, including the output of `hello()`, `bye()`, and the error message for the static method call.

```

105     void bye();
106 }
107
108 class CA implements MyInf{
109
110     public void bye() {
111         System.out.println("This is bye from CA");
112     }
113 }
114
115 public class Interfaces {
116
117     public static void main(String[] args) {
118
119         // Polymorphic Statements with Interfaces
120         // Reference variable of interface can refer to the Object/class implementing it :
121         Notification ref1 = new User();
122         ElectricityBillPayment ref2 = new User();
123
124         System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
125         ElectricityApp electricityApp = new ElectricityApp();
126         User john = new User();
127         electricityApp.subscribeForBill(john);
128
129         electricityApp.broadcastBills();
130
131         System.out.println();
132
133         MyInf inf = new CA(); // Polymorphic Statement
134         inf.hello(); // OK
135         inf.bye(); // Ok
136         //inf.show(); // error
137
138         MyInf.show(); // static methods can be accessed only by the name of interfaces
139     }
140 }

```

- 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.



```
105     void bye();
106 }
107
108 class CA implements MyInf{
109
110@   public void bye() {
111     System.out.println("This is bye from CA");
112   }
113 }
114
115 public class Interfaces {
116
117@   public static void main(String[] args) {
118
119     // Polymorphic Statements with Interfaces
120     // Reference variable of interface can refer to the Object/class implementing it :)
121     Notification ref1 = new User();
122     ElectricityBillPayment ref2 = new User();
123
124     System.out.println("Bill Account Number is: "+ElectricityBillPayment.billAccountNumber);
125     ElectricityApp electricityApp = new ElectricityApp();
126     User john = new User();
127     electricityApp.subscribeForBill(john);
128
129     electricityApp.broadcastBills();
130
131     System.out.println();
132
133     MyInf inf = new CA(); // Polymorphic Statement
134     inf.hello(); // OK
135     inf.bye(); // OK
136     //inf.show(); // error
137
138     MyInf.show(); // static methods can be accessed only by the name of interfaces
139     //CA.show(); // error
140   }
141 }
142 }
```

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 .