

Lesson 04 Demo 03

Implementing Abstraction with Interfaces

Objective: To implement abstraction using interfaces

Tools required: Eclipse IDE

Prerequisites: None

Steps to be followed:

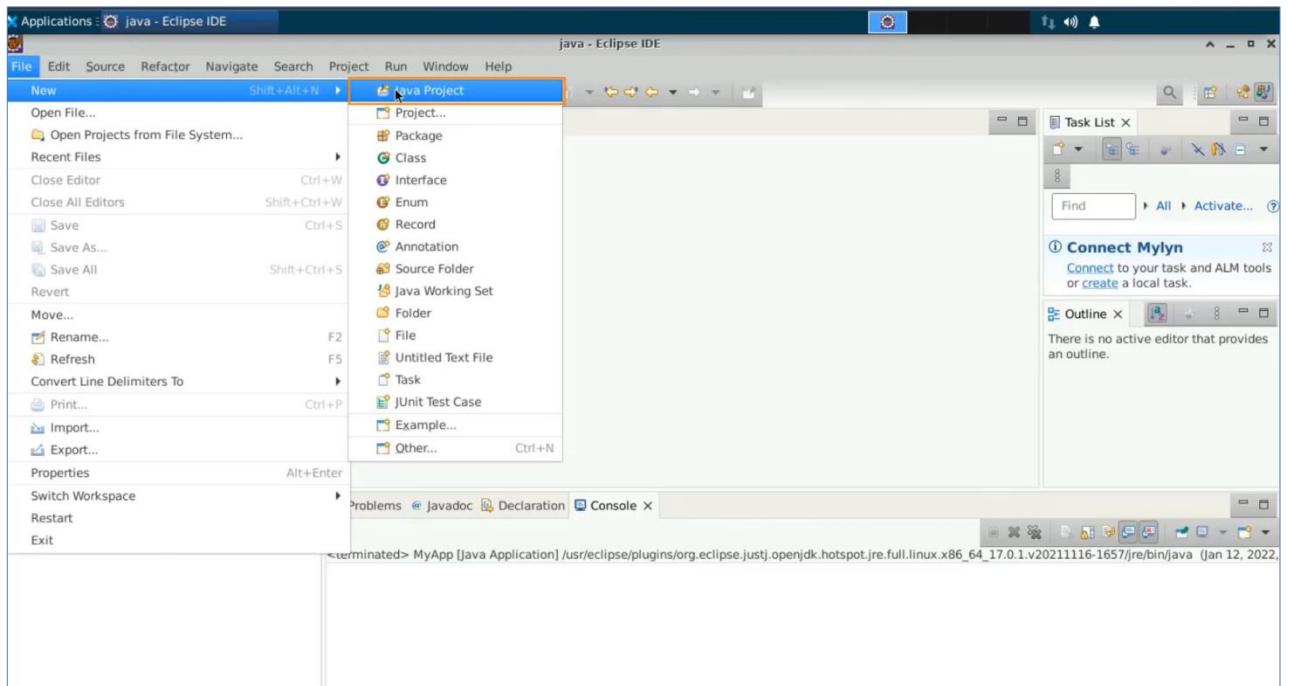
1. Open the IDE and create a new project
2. Consider a class marked as abstract and two methods, each for failure and success
3. Execute the code with example data
4. Make the abstract class as an interface
5. Implement the polymorphic statement and execute the code

Step 1: Open IDE and create a new project

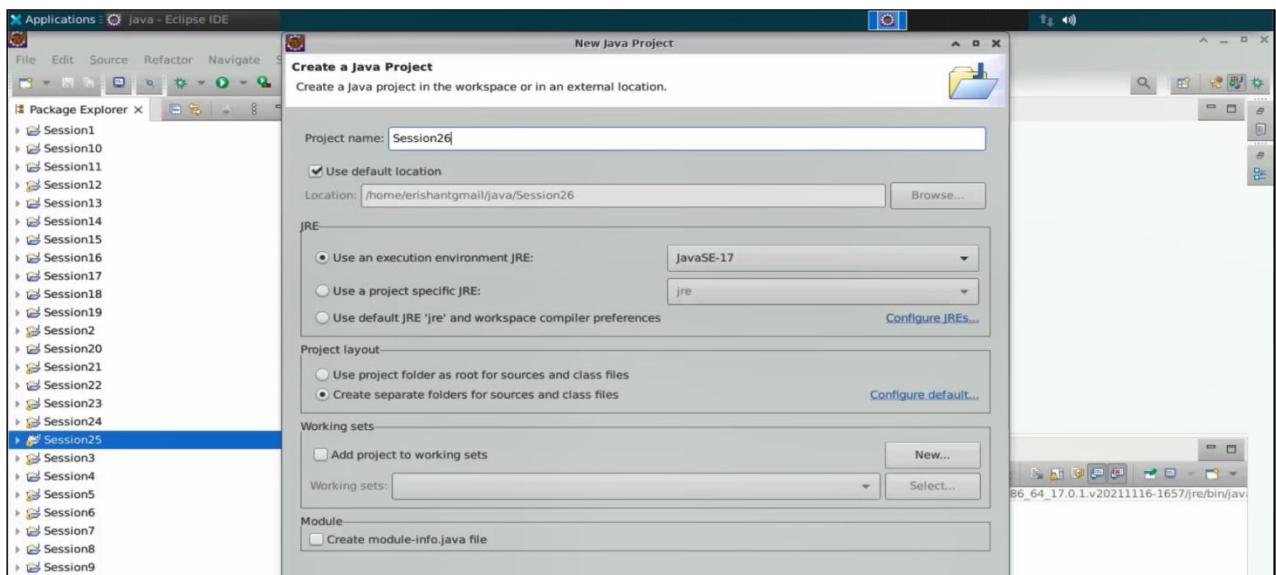
1.1 Open the Eclipse IDE



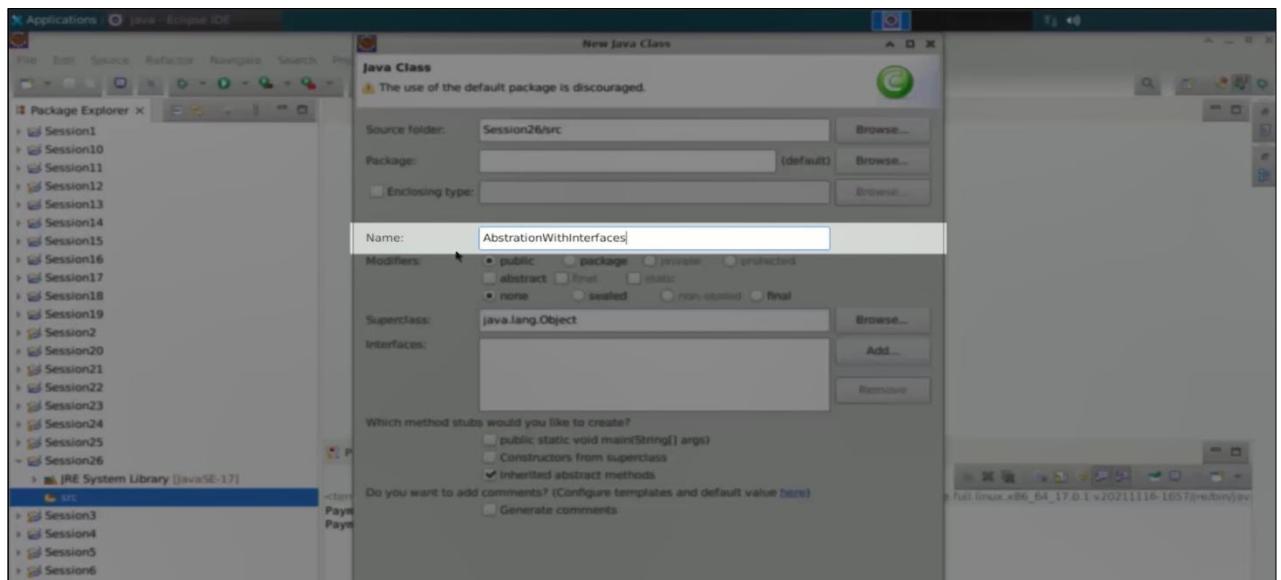
1.2 Select File, then New, and then Java project



1.3 Name the project “Session26”, uncheck “Create module info.java file”, and press Finish



- 1.4 With a **Session26** on the src, do a right-click and create a **new class**. Name this class as an **AbstractionWithInterfaces**, then select the **main method**, and then select **finish**.



Step 2: Consider a class marked as abstract and two methods, each for failure and success

- 2.1 Consider that there is one of the class and this is marked as an abstract class. This abstract class is GooglePay. You can pay from Google, and there is a function called pay, which takes one input as amount. In the pay from google, you can write as Google pay, amount paid, and mention the amount paid.

```

1 abstract class GooglePay{
2
3     void payFromGoogle(int amount) {
4         System.out.println("GooglePay Amount Paid: "+amount);
5     }
6 }
7
8 public class AbstractionWithInterfaces {
9
10    public static void main(String[] args) {
11
12
13
14    }
15
16 }
17

```

- 2.2 In the same abstract class GooglePay, create 2 methods. One is the success method. And write abstract void on failure, for a case when the amount would have been failed. Let's write int as code and string message, that will be two of the inputs now. You can give an Integer code as the first input and the string message as the second input.

```

1 abstract class GooglePay{
2
3    void payFromGoogle(int amount) {
4        System.out.println("[GooglePay] Amount Paid: "+amount);
5    }
6
7    abstract void onSuccess(int code, String message);
8    abstract void onFailure(int code, String message);
9
10
11 public class AbstractionWithInterfaces {
12
13    public static void main(String[] args) {
14
15
16    }
17
18 }
19
20

```

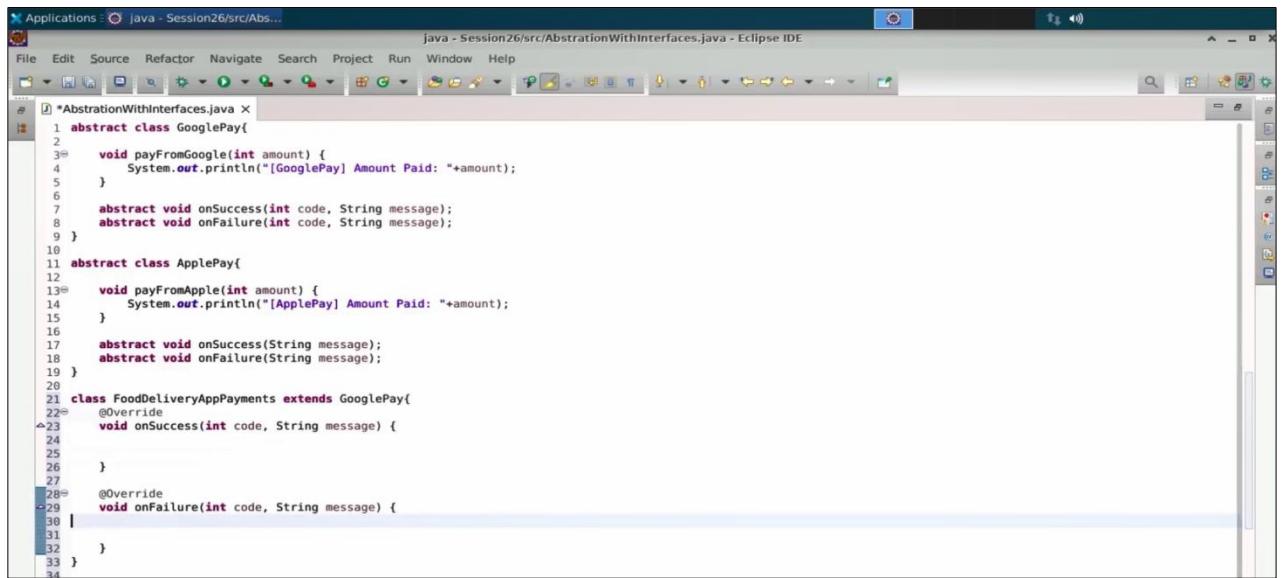
- 2.3 Similar to Google Pay, you can create an identical class called Apple Pay. The Apple Pay class includes a method called payFromApple. This method will use System.out.println to display a message stating "Apple Pay" along with the amount paid.

```

1 abstract class GooglePay{
2
3    void payFromGoogle(int amount) {
4        System.out.println("[GooglePay] Amount Paid: "+amount);
5    }
6
7    abstract void onSuccess(int code, String message);
8    abstract void onFailure(int code, String message);
9
10
11 abstract class ApplePay{
12
13    void payFromApple(int amount) {
14        System.out.println("[ApplePay] Amount Paid: "+amount);
15    }
16
17    abstract void onSuccess(String message);
18    abstract void onFailure(String message);
19
20
21 public class AbstractionWithInterfaces {
22
23    public static void main(String[] args) {
24
25
26    }
27
28 }

```

- 2.4 Create a food delivery app with a payment module called FoodDeliveryAppPayments that extends Google Pay, defining success and failure methods as per the Google method with the code and message.



The screenshot shows the Eclipse IDE interface with a Java file named "AbstractionWithInterfaces.java" open in the editor. The code defines two abstract classes, GooglePay and ApplePay, and a concrete class, FoodDeliveryAppPayments, which implements GooglePay. The code includes annotations like @Override and System.out.println statements.

```
# *AbstractionWithInterfaces.java *
1 abstract class GooglePay{
2
3    void payFromGoogle(int amount) {
4        System.out.println("[GooglePay] Amount Paid: "+amount);
5    }
6
7    abstract void onSuccess(int code, String message);
8    abstract void onFailure(int code, String message);
9 }
10
11 abstract class ApplePay{
12
13    void payFromApple(int amount) {
14        System.out.println("[ApplePay] Amount Paid: "+amount);
15    }
16
17    abstract void onSuccess(String message);
18    abstract void onFailure(String message);
19 }
20
21 class FoodDeliveryAppPayments extends GooglePay{
22
23     @Override
24     void onSuccess(int code, String message) {
25
26     }
27
28     @Override
29     void onFailure(int code, String message) {
30
31
32     }
33 }
34
```

Step 3: Execute the code with example data

3.1 Once the payment is made, Google will execute the success method, sending code 101 and the message "transaction success." Print "FDA payment received; we will deliver the order soon" in the food delivery app to handle this success case.

The screenshot shows the Eclipse IDE interface with the title bar "java - Session26/src/AbstractionWithInterfaces.java - Eclipse IDE". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has various icons for file operations like Open, Save, Cut, Copy, Paste, Find, etc. The left sidebar shows the package structure with "AbstractionWithInterfaces.java X" selected. The main editor area contains the following Java code:

```
1 abstract class GooglePay{
2
3     void payFromGoogle(int amount) {
4         System.out.println("[GooglePay] Amount Paid: "+amount);
5         onSuccess(101, "Transaction Success");
6     }
7
8     abstract void onSuccess(int code, String message);
9     abstract void onFailure(int code, String message);
10 }
11
12 abstract class ApplePay{
13
14     void payFromApple(int amount) {
15         System.out.println("[ApplePay] Amount Paid: "+amount);
16     }
17
18     abstract void onSuccess(String message);
19     abstract void onFailure(String message);
20 }
21
22 class FoodDeliveryAppPayments extends GooglePay{
23
24     @Override
25     void onSuccess(int code, String message) {
26         System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+" "+message);
27     }
28     I
29     @Override
30     void onFailure(int code, String message) {
31
32     }
33
34 }
```

3.2 Same way, if something goes wrong, it can be changed as payment failed over here. And here you can give as, please try again, with the code and the message. Later in the execution phase, you will create the object of your food delivery app payments.

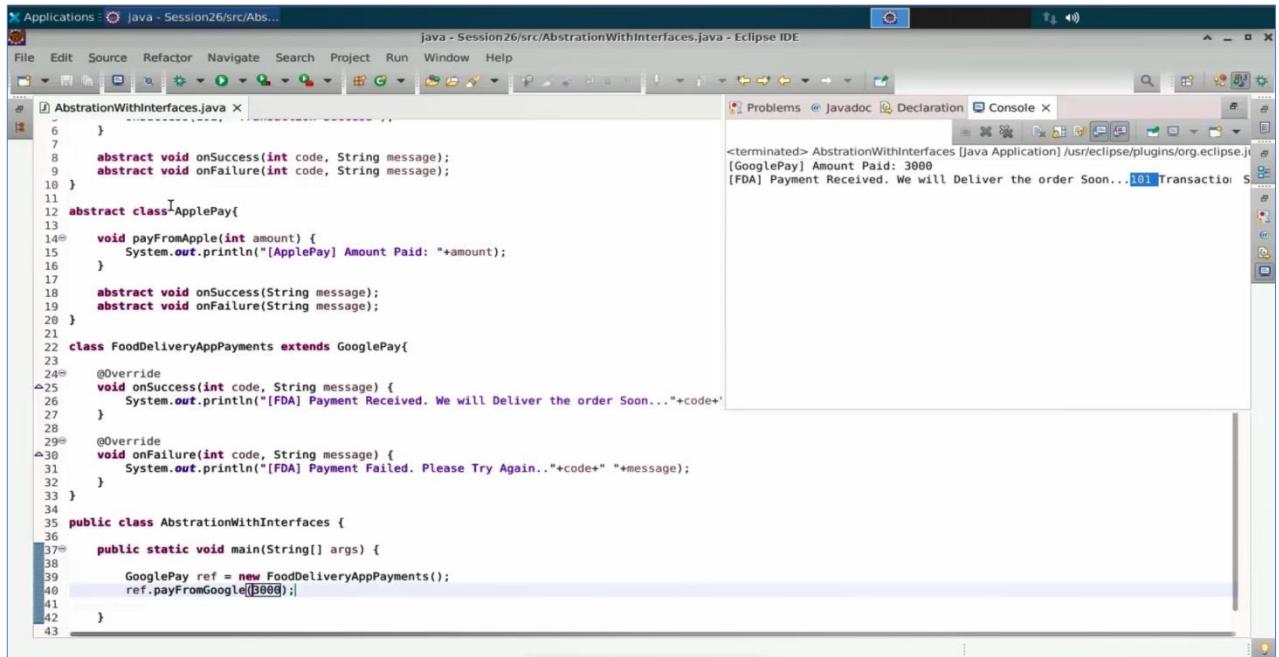
The screenshot shows the Eclipse IDE interface with the title bar "java - Session26/src/AbstractionWithInterfaces.java - Eclipse IDE". The code editor displays the following Java code:

```
File Edit Source Refactor Navigate Search Project Run Window Help

*AbstractionWithInterfaces.java X
1
2
3
4
5
6
7
8     abstract void onSuccess(int code, String message);
9     abstract void onFailure(int code, String message);
10 }
11
12 abstract class ApplePay{
13
14     void payFromApple(int amount) {
15         System.out.println("[ApplePay] Amount Paid: "+amount);
16     }
17
18     abstract void onSuccess(String message);
19     abstract void onFailure(String message);
20 }
21
22 class FoodDeliveryAppPayments extends GooglePay{
23
24     @Override
25     void onSuccess(int code, String message) {
26         System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+" "+message);
27     }
28
29     @Override
30     void onFailure(int code, String message) {
31         System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
32     }
33 }
34
35 public class AbstractionWithInterfaces {
36
37     public static void main(String[] args) {
38
39

```

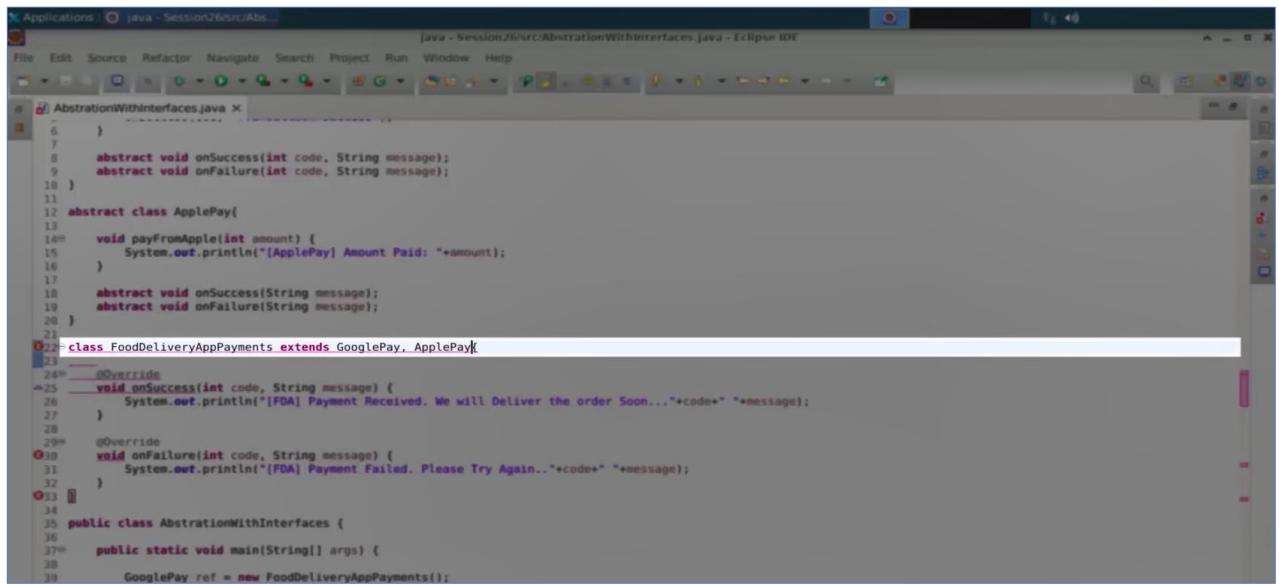
3.3 Let us write a polymorphic statement. Suppose you are using Google Pay as a reference for this food delivery app payment. Then, you can execute this method and write it as pay from Google, with the amount as 3000. Run the code. The Google Pay amount is paid, and you receive a message inside your food delivery app payment object, which says that payment has been received. You also get a code and a message from Google.



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

- Title Bar:** Applications - Java - Session26/src/Abs...
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Toolbar:** Standard Eclipse toolbar icons.
- Left Panel:** Shows the file "AbstractionWithInterfaces.java" is open.
- Code Editor:** Displays the Java code for the `AbstractionWithInterfaces` class. It includes an abstract base class `GooglePay` with methods `onSuccess` and `onFailure`, and an abstract class `ApplePay` with a method `payFromApple`. A concrete class `FoodDeliveryAppPayments` extends `GooglePay` and overrides both methods to print messages to `System.out`.
- Console View:** Shows the output of the application's execution. It includes the command `<terminated> AbstractionWithInterfaces [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.core/lib/jdt-launcher`, the message `[GooglePay] Amount Paid: 3000`, and the message `[FDA] Payment Received. We will Deliver the order Soon... Transaction S...`.

3.4 Sometimes, as a developer, you may wish to integrate Apple Pay as well. This is where abstraction comes into play. If you implement Apple Pay using abstraction, you are not required to use multiple inheritance. Abstraction with interfaces will eliminate this challenge.

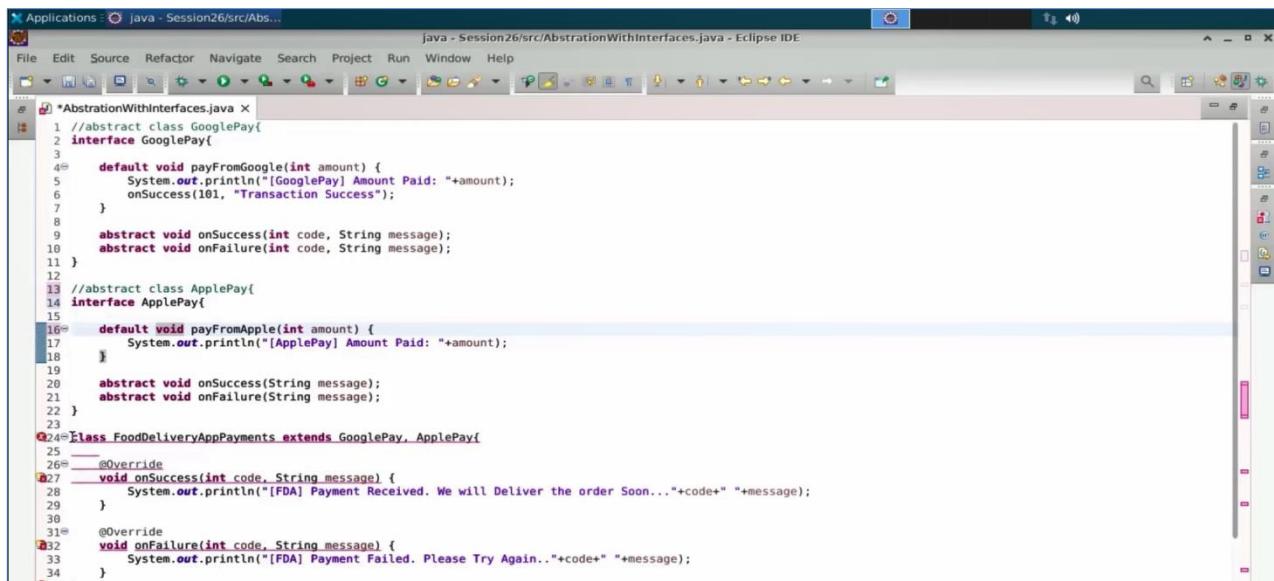


The screenshot shows the Eclipse IDE interface with a Java file named 'AbstractionWithInterfaces.java' open. The code implements abstraction using interfaces and inheritance:

```
File Edit Source Refactor Navigate Search Project Run Window Help
AbstractionWithInterfaces.java X
1
2
3
4
5
6
7     abstract void onSuccess(int code, String message);
8     abstract void onFailure(int code, String message);
9 }
10 }
11 abstract class ApplePay{
12
13     void payFromApple(int amount) {
14         System.out.println("[ApplePay] Amount Paid: "+amount);
15     }
16
17     abstract void onSuccess(String message);
18     abstract void onFailure(String message);
19 }
20
21
22 class FoodDeliveryAppPayments extends GooglePay, ApplePay{
23
24     @Override
25     void onSuccess(int code, String message) {
26         System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+" "+message);
27     }
28
29     @Override
30     void onFailure(int code, String message) {
31         System.out.println("[FDA] Payment Failed.. Please Try Again.."+code+" "+message);
32     }
33 }
34
35 public class AbstractionWithInterfaces {
36
37     public static void main(String[] args) {
38
39         GooglePay ref = new FoodDeliveryAppPayments();
40
41     }
42 }
```

Step 4: Make the abstract class as an interface

4.1 Now, you will convert this abstract class into an interface, which can be named Google Pay. Since interfaces cannot have method definitions, you can mark the methods as default. Similarly, convert the abstract class Apple Pay into an interface called Apple Pay and mark the methods as default.



The screenshot shows the Eclipse IDE interface with the title bar "java - Session26/src/AbstractionWithInterfaces.java - Eclipse IDE". The code editor displays the following Java code:

```
* AbstractionWithInterfaces.java *
1 //abstract class GooglePay{
2 interface GooglePay{
3
4     default void payFromGoogle(int amount) {
5         System.out.println("[GooglePay] Amount Paid: "+amount);
6         onSuccess(101, "Transaction Success");
7     }
8
9     abstract void onSuccess(int code, String message);
10    abstract void onFailure(int code, String message);
11 }
12
13 //abstract class ApplePay{
14 interface ApplePay{
15
16     default void payFromApple(int amount) {
17         System.out.println("[ApplePay] Amount Paid: "+amount);
18     }
19
20     abstract void onSuccess(String message);
21     abstract void onFailure(String message);
22 }
23
24 Class FoodDeliveryAppPayments extends GooglePay, ApplePay{
25
26     @Override
27     void onSuccess(int code, String message) {
28         System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+" "+message);
29     }
30
31     @Override
32     void onFailure(int code, String message) {
33         System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
34     }
}
```

4.2 Instead of your class stating that it wants to perform an extension now, this will cause an error because multiple inheritance is not supported. There is a class called FoodDeliveryAppPayments. You can write it as implements GooglePay, and as you can see, there is no change other than the fact that if you have a method, you do not need to mark it as abstract.

```

1 //abstract class GooglePay{
2 interface GooglePay{
3
4     default void payFromGoogle(int amount) {
5         System.out.println("[GooglePay] Amount Paid: "+amount);
6         onSuccess(101, "Transaction Success");
7     }
8
9     abstract void onSuccess(int code, String message);
10    abstract void onFailure(int code, String message);
11 }
12
13 //abstract class ApplePay{
14 interface ApplePay{
15
16     default void payFromApple(int amount) {
17         System.out.println("[ApplePay] Amount Paid: "+amount);
18     }
19
20     abstract void onSuccess(String message);
21     abstract void onFailure(String message);
22 }
23
24 //class FoodDeliveryAppPayments extends GooglePay, ApplePay{ // error as multiple extends not supported
25 class FoodDeliveryAppPayments implements GooglePay{
26
27     @Override
28     void onSuccess(int code, String message) {
29         System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+" "+message);
30     }
31
32     @Override
33     void onFailure(int code, String message) {
34         System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
35 }

```

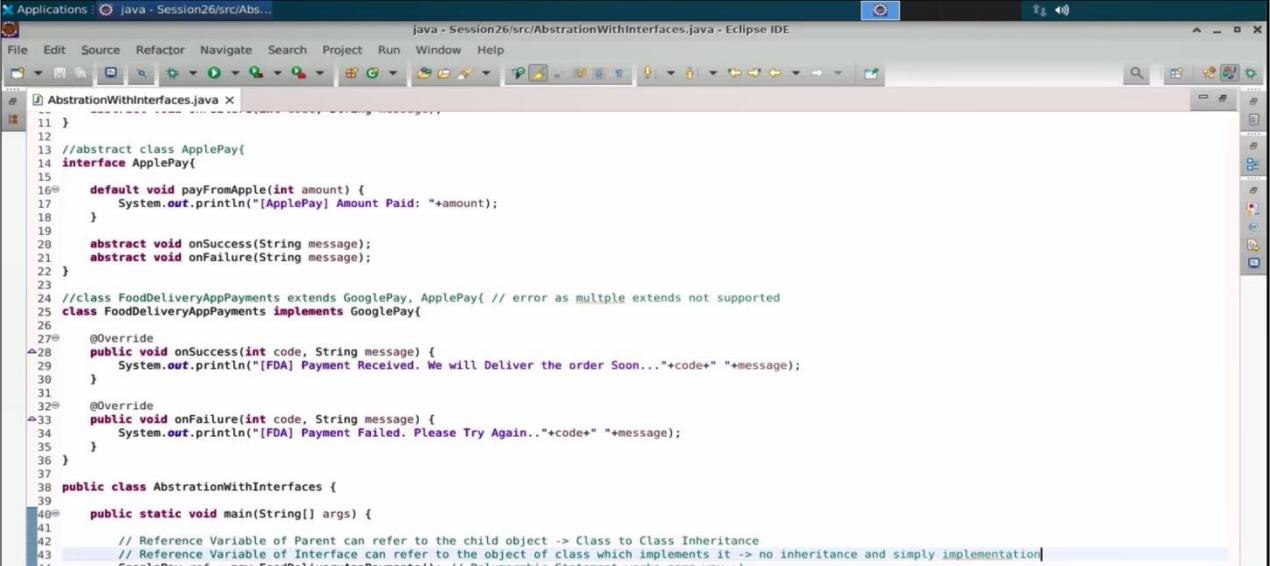
4.3 By default, methods in an interface are public and abstract. You need to use the public keyword in front of the method to ensure that the access level remains the same in both the interface and the implementing class.

```

1 //abstract class GooglePay{
2 interface GooglePay{
3
4     default void payFromGoogle(int amount) {
5         System.out.println("[GooglePay] Amount Paid: "+amount);
6         onSuccess(101, "Transaction Success");
7     }
8
9     public abstract void onSuccess(int code, String message); // -> public abstract
10    abstract void onFailure(int code, String message);
11 }
12
13 //abstract class ApplePay{
14 interface ApplePay{
15
16     default void payFromApple(int amount) {
17         System.out.println("[ApplePay] Amount Paid: "+amount);
18     }
19
20     abstract void onSuccess(String message);
21     abstract void onFailure(String message);
22 }
23
24 //class FoodDeliveryAppPayments extends GooglePay, ApplePay{ // error as multiple extends not supported
25 class FoodDeliveryAppPayments implements GooglePay{
26
27     @Override
28     public void onSuccess(int code, String message) {
29         System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+" "+message);
30     }
31
32     @Override
33     public void onFailure(int code, String message) {
34         System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
35 }

```

4.4 Here, success and failure are two methods implemented the same way. You can work in the same manner, and the polymorphic statement is not affected. Hence, the polymorphic statement works the same way. It simply means that a reference variable of the parent can refer to the child object. Similarly, a reference variable of an interface can refer to the object of a class that implements it. This is not inheritance, but simply implementation.

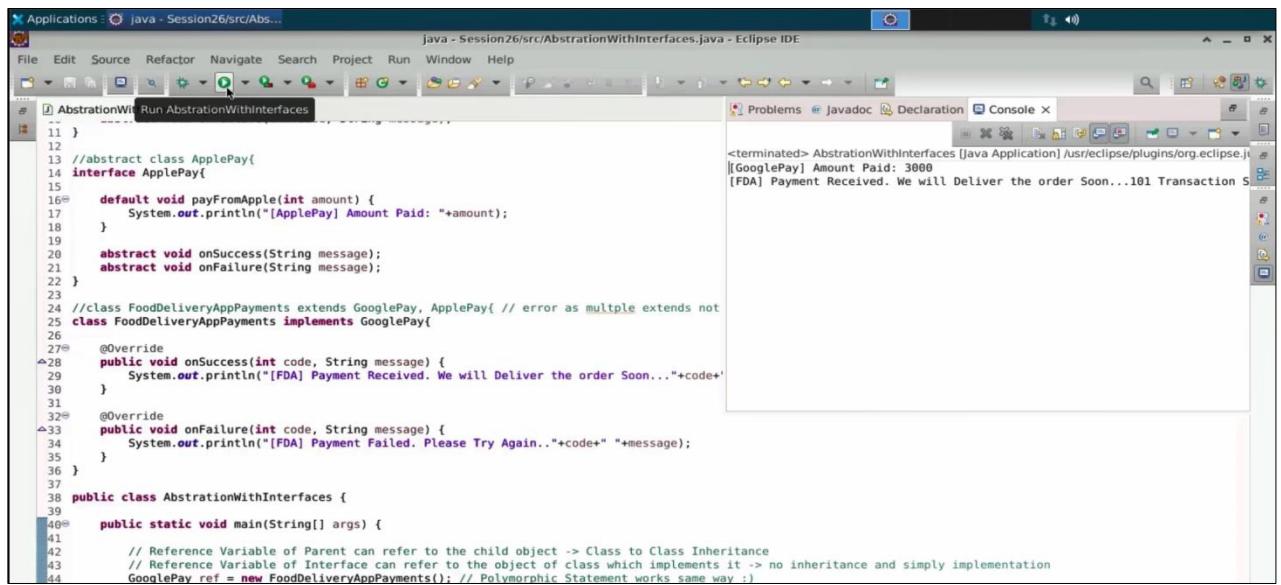


The screenshot shows the Eclipse IDE interface with the file 'AbstractionWithInterfaces.java' open. The code defines an abstract class 'ApplePay' with a default method 'payFromApple' and two abstract methods 'onSuccess' and 'onFailure'. It then defines a concrete class 'FoodDeliveryAppPayments' that implements 'ApplePay'. The 'FoodDeliveryAppPayments' class overrides the abstract methods 'onSuccess' and 'onFailure' with its own implementations. Finally, the main method creates an instance of 'FoodDeliveryAppPayments' and prints a message about polymorphism.

```
11 }
12 //abstract class ApplePay{
13 interface ApplePay{
14
15     default void payFromApple(int amount) {
16         System.out.println("[ApplePay] Amount Paid: "+amount);
17     }
18 }
19
20 abstract void onSuccess(String message);
21 abstract void onFailure(String message);
22 }
23
24 //class FoodDeliveryAppPayments extends GooglePay, ApplePay{ // error as multiple extends not supported
25 class FoodDeliveryAppPayments implements GooglePay{
26
27     @Override
28     public void onSuccess(int code, String message) {
29         System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+" "+message);
30     }
31
32     @Override
33     public void onFailure(int code, String message) {
34         System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
35     }
36 }
37
38 public class AbstractionWithInterfaces {
39
40     public static void main(String[] args) {
41
42         // Reference Variable of Parent can refer to the child object -> Class to Class Inheritance
43         // Reference Variable of Interface can refer to the object of class which implements it -> no inheritance and simply implementation|
44         GooglePay ref = new FoodDeliveryAppPayments(); // Polymorphic Statement works same way :)
```

Step 5: Implement the polymorphic statement and execute the code

5.1 The main fundamental of a polymorphic statement is that it works for both classes and interfaces, but the fundamental structure is different. When you use "pay from Google" and run the code, the abstraction is implemented in the same way. However, now you are seeing it more crisply with interfaces.

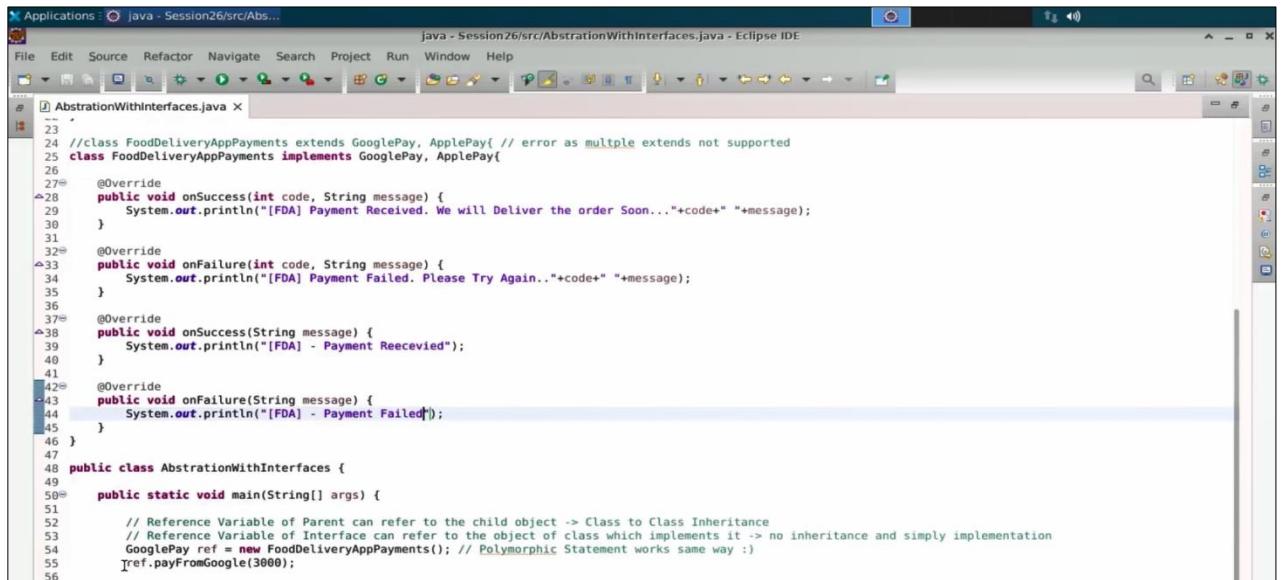


The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Applications - java - Session26/src/Abs...
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Toolbar:** Standard Eclipse toolbar icons.
- Left Panel:** Shows the project structure with a file named "AbstractionWithInterfaces".
- Code Editor:** Displays the Java code for "AbstractionWithInterfaces.java". The code defines an abstract class "ApplePay" with a default method "payFromApple" and two abstract methods "onSuccess" and "onFailure". It also defines a class "FoodDeliveryAppPayments" that implements "ApplePay". The "main" method creates an instance of "FoodDeliveryAppPayments" and calls "payFromApple".
- Console Output:** Shows the execution results:


```
<terminated> AbstractionWithInterfaces [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.core/compiler.jar
[GooglePay] Amount Paid: 3000
[FDA] Payment Received. We will Deliver the order Soon...101 Transaction S...
```

5.2 Then, here you can write, "Apple Pay." With Apple Pay coming into action, you need to define the methods from Apple Pay. You have the success method from Apple Pay and the failure method from Apple Pay. Let's write System.out.println in the success method to display "Payment received from the food delivery app." Similarly, in the failure method, write "Payment failed from the food delivery app.



```
  Applications : java - Session26/src/Abs...
java - Session26/src/AbstractionWithInterfaces.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
AbstractionWithInterfaces.java X
23
24 //class FoodDeliveryAppPayments extends GooglePay, ApplePay{ // error as multiple extends not supported
25 class FoodDeliveryAppPayments implements GooglePay, ApplePay{
26
27@    @Override
28    public void onSuccess(int code, String message) {
29        System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+" "+message);
30    }
31
32@    @Override
33    public void onFailure(int code, String message) {
34        System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
35    }
36
37@    @Override
38    public void onSuccess(String message) {
39        System.out.println("[FDA] - Payment Received");
40    }
41
42@    @Override
43    public void onFailure(String message) {
44        System.out.println("[FDA] - Payment Failed");
45    }
46 }
47
48 public class AbstractionWithInterfaces {
49
50@    public static void main(String[] args) {
51
52        // Reference Variable of Parent can refer to the child object -> Class to Class Inheritance
53        // Reference Variable of Interface can refer to the object of class which implements it -> no inheritance and simply implementation
54        GooglePay ref = new FoodDeliveryAppPayments(); // Polymorphic Statement works same way :
55        ref.payFromGoogle(3000);
56    }
57}
```

- 5.3 The next step is, instead of paying with Google Pay, you can create an Apple Pay instance. You can write it as "Apple Pay reference = new FoodDeliveryAppPayment();" and then, with the reference, you can execute the method called payFromApple with an amount of 5000. When making a payment with Apple Pay, if it fails, write "Bank interface down, error code 33120.

The screenshot shows the Eclipse IDE interface with the title bar "java - Session26/src/Abs...". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has various icons for file operations like Open, Save, Cut, Copy, Paste, Find, etc. The left sidebar shows the package structure with "AbstractionWithInterfaces.java X". The main editor area contains the following Java code:

```
23
24 // class FoodDeliveryAppPayments extends GooglePay, ApplePay{ // error as multiple extends not supported
25 class FoodDeliveryAppPayments implements GooglePay, ApplePay{
26
27@    @Override
28    public void onSuccess(int code, String message) {
29        System.out.println("[FDA] Payment Received. We will Deliver the order Soon..." +code+ " " +message);
30    }
31
32@    @Override
33    public void onFailure(int code, String message) {
34        System.out.println("[FDA] Payment Failed. Please Try Again.." +code+ " " +message);
35    }
36
37@    @Override
38    public void onSuccess(String message) {
39        System.out.println("[FDA] - Payment Recieved");
40    }
41
42@    @Override
43    public void onFailure(String message) {
44        System.out.println("[FDA] - Payment Failed");
45    }
46 }
47
48 public class AbstractionWithInterfaces {
49
50@    public static void main(String[] args) {
51
52        // Reference Variable of Parent can refer to the child object -> Class to Class Inheritance
53        // Reference Variable of Interface can refer to the object of class which implements it -> no inheritance and simply implementation
54
55        //GooglePay ref = new FoodDeliveryAppPayments(); // Polymorphic Statement works same way :)
56        //ref.payFromGoogle(3000);
57
58        ApplePay ref = new FoodDeliveryAppPayments();
59        ref.payFromApple(5000);
60    }
61 }
```

- 5.4 When you run the code here, it shows "Amount paid: 5000." This comes from Apple Pay, but if you execute the failure method, it shows "Payment failed."

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

- Title Bar:** Applications > java - Session26/src/AbstractionWithInterfaces.java
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and project management.
- Left Sidebar:** Shows the package structure and the current file being edited: AbstractionWithInterfaces.java.
- Code Editor:** Displays the Java code for `AbstractionWithInterfaces.java`. The code defines two abstract classes, `GooglePay` and `ApplePay`, each with a `payFrom` method and an `onSuccess` method. It also defines a concrete class `FoodDeliveryAppPayments` that implements both interfaces.
- Console View:** Shows the output of the application's execution. The output window title is "terminated: AbstractionWithInterfaces [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.core". The output text is: "[ApplePay] Amount Paid: 5000 [FDA] - Payment Failed".

- 5.5 And with the payment failed, let us come here and display the message. Save it and rerun it. You can see the message as bank interface down, error Code 33120. Thus, you can work with the abstraction, and you can implement it through interfaces, which is the best of the choices rather than working with the abstract classes.

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

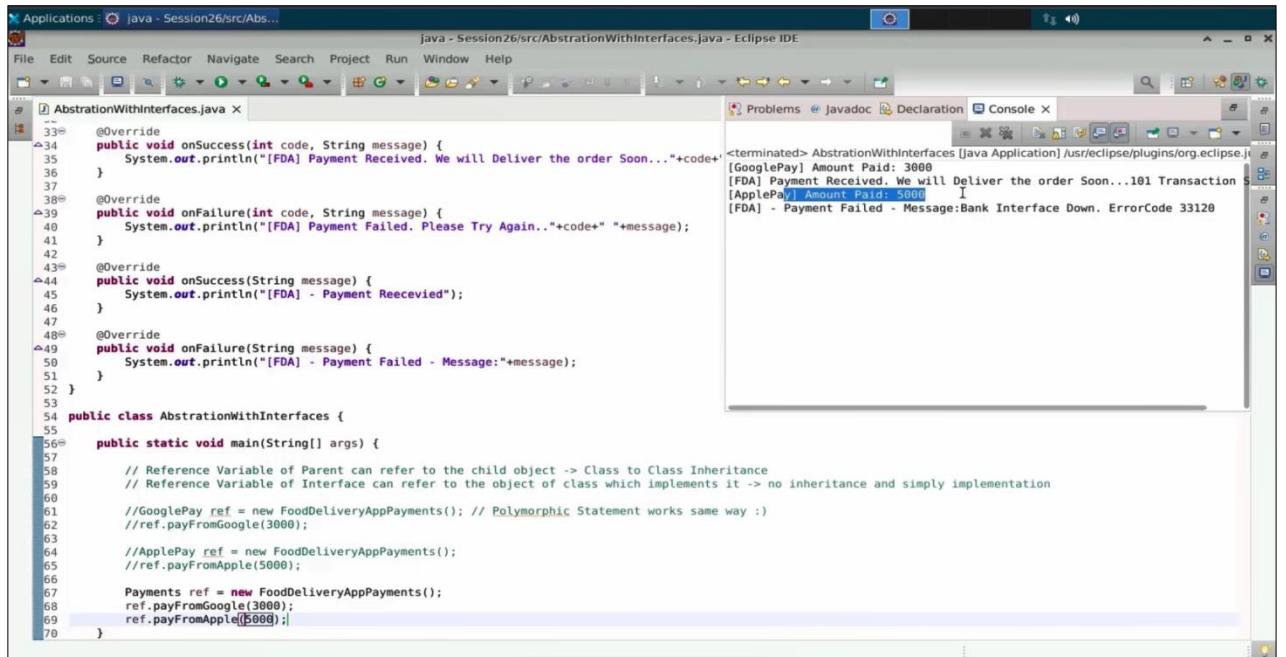
- Title Bar:** Applications > java - Session26/src/Abs... and java - Session26/src/AbstractionWithInterfaces.java - Eclipse IDE.
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbars:** Standard toolbar with icons for file operations, search, and run.
- Left Panel:** Shows the project structure with "AbstractionWithInterfaces.java X".
- Code Editor:** Displays the Java code for the `ApplePay` interface and the `FoodDeliveryAppPayments` class implementation. The code includes methods like `payFromApple`, `onSuccess`, `onFailure`, and `onSuccess` and `onFailure` for the `FDA` class.
- Output View:** Shows the terminal output with the message: `<terminated> AbstractionWithInterfaces [java Application] /usr/eclipse/plugins/org.eclipse.jdt.core/compiler.jar`, followed by the printed messages from the code.

- 5.6 Polymorphic statements work in both ways. At the same time, with interfaces, you can create an interface called Payments that extends both Google Pay and Apple Pay. Then, implement only the Payments interface. This is another way it works.

The screenshot shows the Eclipse IDE interface with the title bar "Applications : java - Session26/src/Abs...". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has various icons for file operations like Open, Save, Cut, Copy, Paste, Find, and Run. The left margin shows the package structure with "src" and "AbstractionWithInterfaces.java". The main editor area displays the following Java code:

```
abstract void onFailure(int code, String message);
}
//abstract class ApplePay{
interface ApplePay{
    default void payFromApple(int amount) {
        System.out.println("ApplePay] Amount Paid: "+amount);
        onFailure("Bank Interface Down. ErrorCode 33120");
    }
    abstract void onSuccess(String message);
    abstract void onFailure(String message);
}
interface Payments extends GooglePay, ApplePay{
}
//class FoodDeliveryAppPayments extends GooglePay, ApplePay{ // error as multiple extends not supported
class FoodDeliveryAppPayments implements Payments{"/GooglePay, ApplePay{
    @Override
    public void onSuccess(int code, String message) {
        System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+" "+message);
    }
    @Override
    public void onFailure(int code, String message) {
        System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
    }
    @Override
```

5.7 Next, you can create the reference variable of these payments, which can be your FoodDeliveryAppPayments. With this reference, you can pay using Google Pay an amount of 3000. With the same reference variable, you can also pay using Apple Pay. You can integrate both in this manner. Thus, Google Pay would work, and Apple Pay would work. This is how you can combine interfaces and create a single interface through multiple inheritance.



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

- Title Bar:** Applications > java - Session26/src/Abs...
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Toolbar:** Standard Eclipse toolbar icons.
- Left Panel:** Shows the package structure and the current file open: AbstractionWithInterfaces.java.
- Code Editor:** Displays the Java code for the `AbstractionWithInterfaces` class. The code defines four methods: `onSuccess(int code, String message)`, `onFailure(int code, String message)`, `onSuccess(String message)`, and `onFailure(String message)`. It also includes a main method that creates a `FoodDeliveryAppPayments` object and calls its `payFromGoogle` and `payFromApple` methods.
- Right Panel:** Shows the `Console` tab with the following output:


```
<terminated> AbstractionWithInterfaces [java Application] /usr/eclipse/plugins/org.eclipse.jdt.core/compiler.jar
[GooglePay] Amount Paid: 3000
[FDA] Payment Received. We will Deliver the order Soon...101 Transaction S
[ApplePay] Amount Paid: 5000
[FDA] - Payment Failed - Message:Bank Interface Down. ErrorCode 33120
```

By using the above steps, you have successfully implemented abstraction using interfaces, enabling a modular, flexible, and maintainable code structure that supports multiple functionalities.