Lesson 04 Demo 03

Implementing Abstraction with Interfaces

Objective: To implement abstraction using interfaces

Tools required: Eclipse IDE

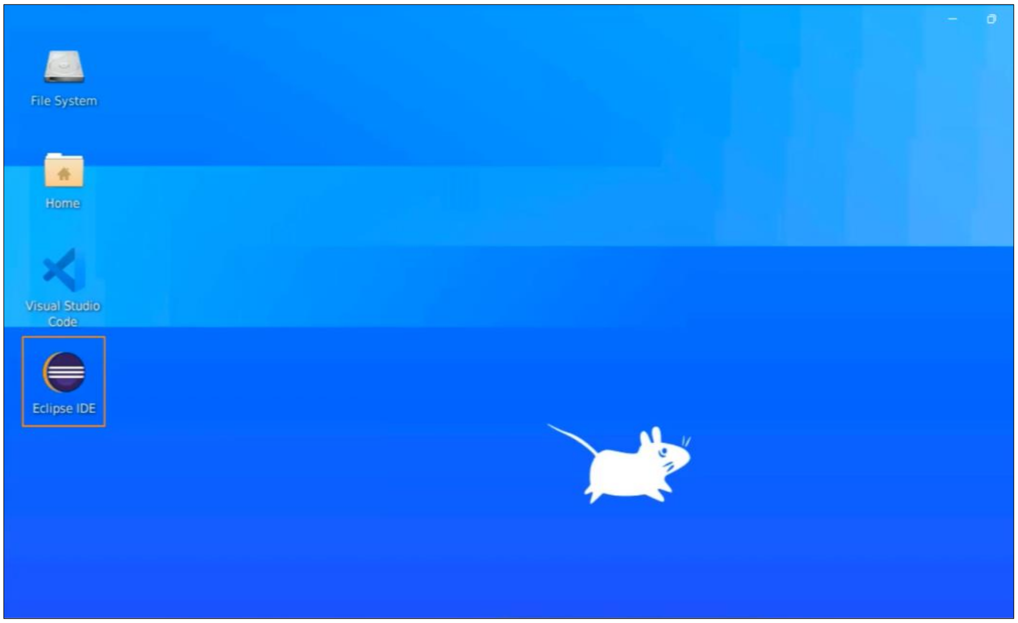
Prerequisites: None

Steps to be followed:

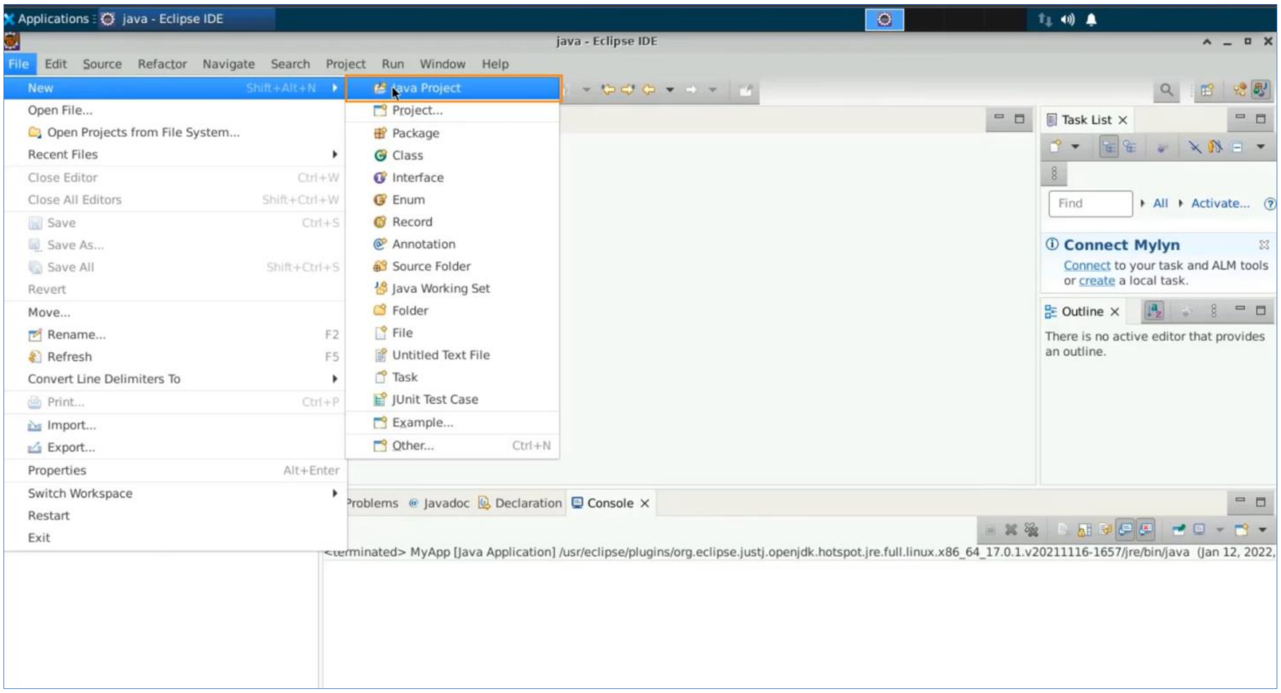
1. Open the IDE and create a new project
2. Consider a class marked as abstract and two methods, each for failure and success
3. Execute the code with example data
4. Make the abstract class as an interface
5. Implement the polymorphic statement and execute the code

Step 1: Open IDE and create a new project

1.1 Open the **Eclipse IDE**

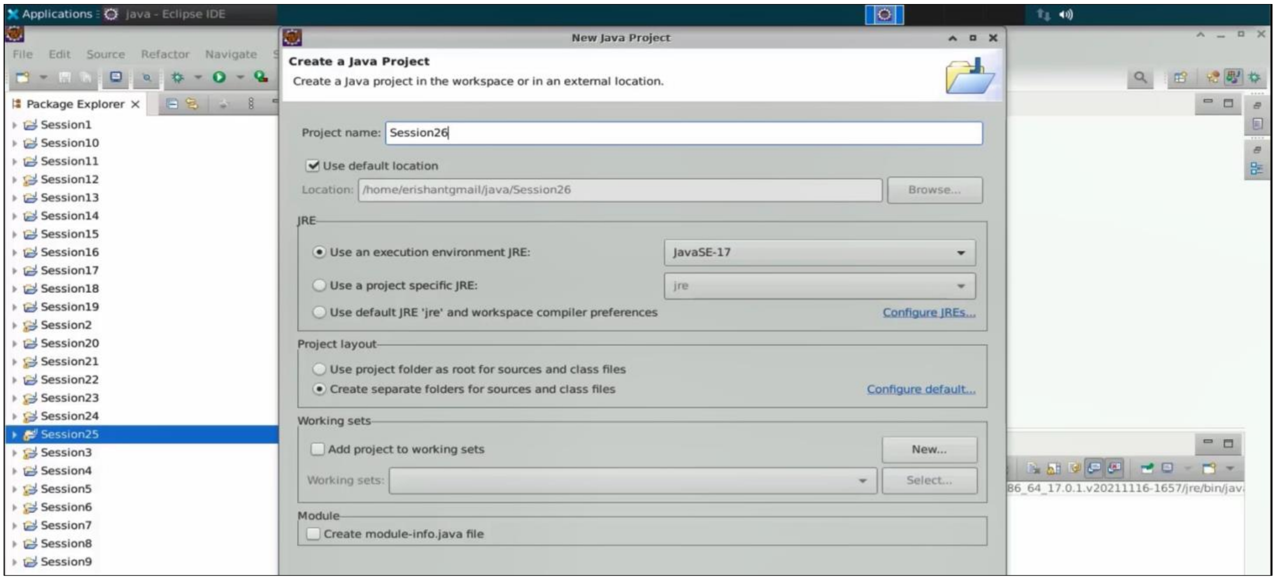


1.2 Select **File**, then **New,** and then **Java project**



```
<terminated> MyApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java (Jan 12, 2022,
```

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



Create a Java Project

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

Project name: Session26

☑ Use default location

Location: /home/erishantgmail/java/Session26 Browse...

JRE

● Use an execution environment JRE: JavaSE-17

○ Use a project specific JRE: jre

○ Use default JRE 'jre' and workspace compiler preferences Configure JREs...

Project layout

○ Use project folder as root for sources and class files

● Create separate folders for sources and class files Configure default...

Working sets

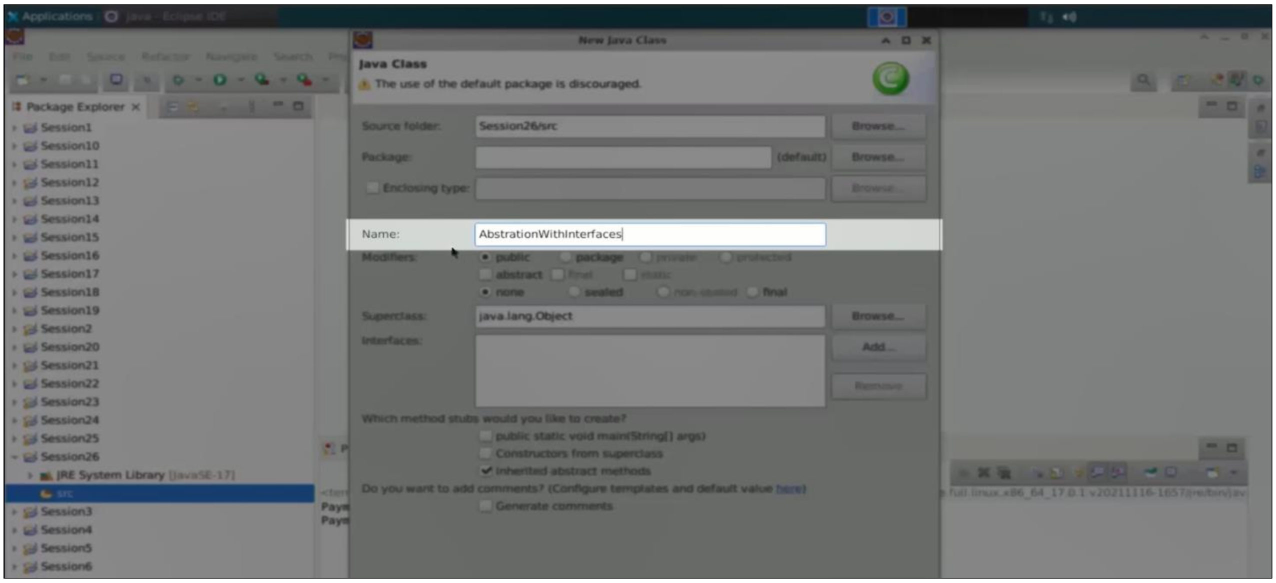
□ Add project to working sets New...

Working sets: Select...

Module

□ Create module-info.java file

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.**



New Java Class

Java Class

The use of the default package is discouraged.

Source folder: Session26/src

Package: (default)

□ Enclosing type:

Name: AbstrationWithInterfaces

Modifiers: ● public ○ package ○ private ○ protected

□ abstract □ final □ static

● none ○ sealed ○ non-sealed ○ final

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

□ public static void main(String[] args)

□ Constructors from superclass

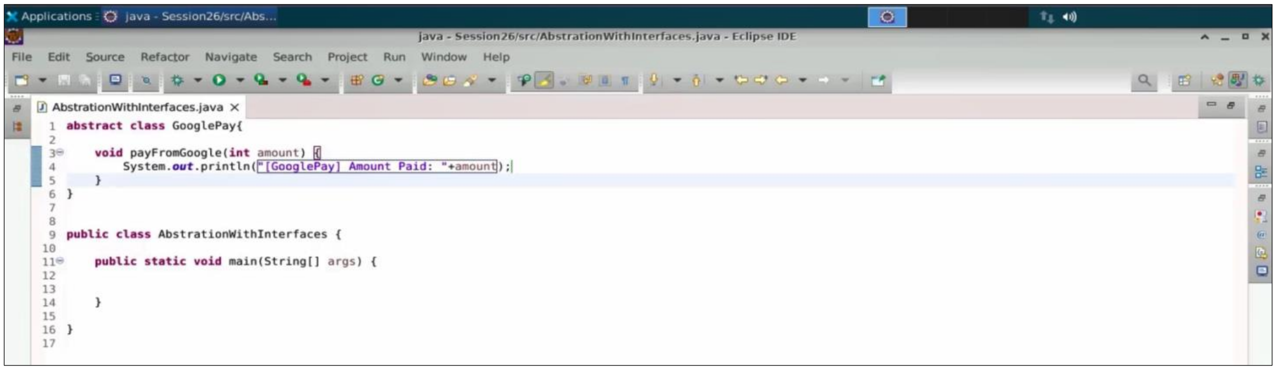
☑ Inherited abstract methods

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

□ Generate comments

Step 2: 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.

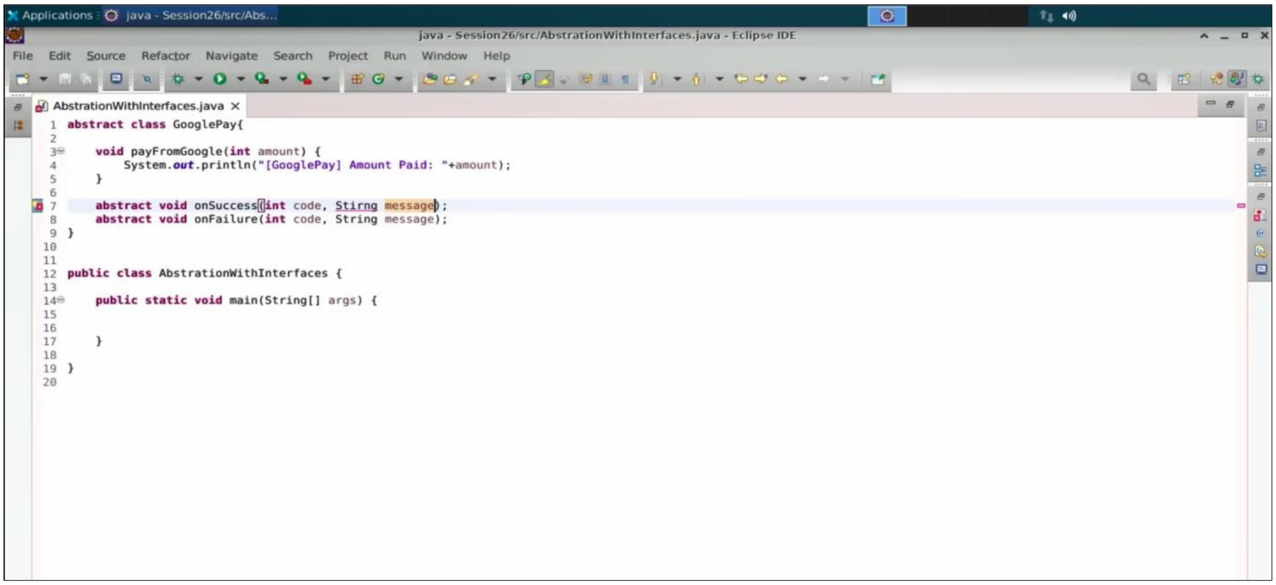


java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE

AbstrationWithInterfaces.java

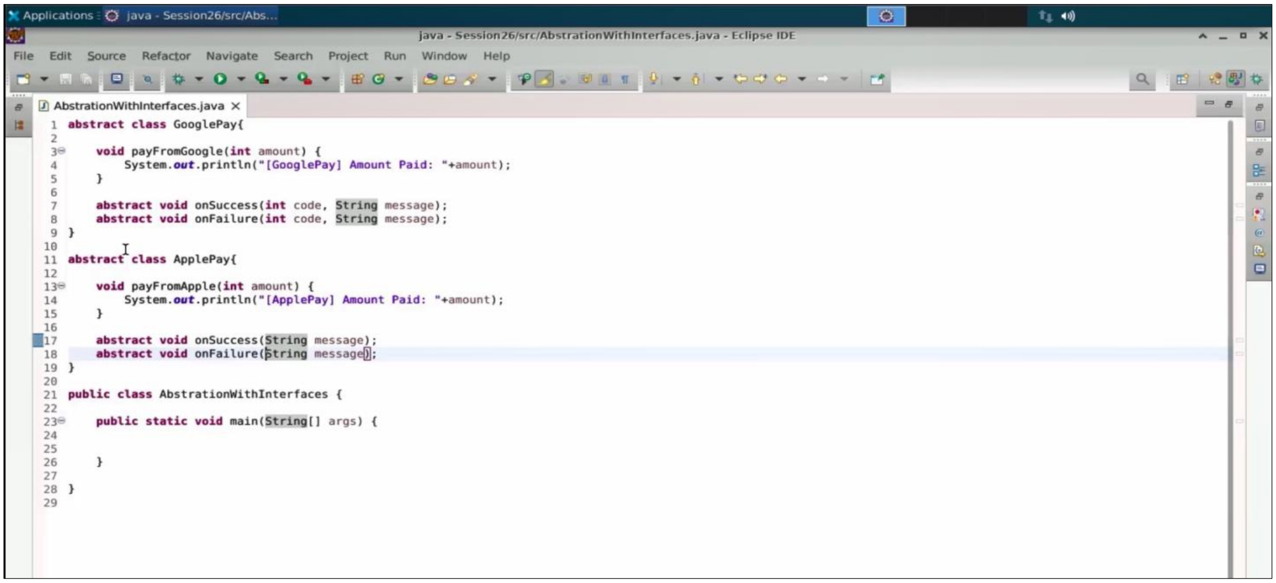
```java
abstract class GooglePay{

    void payFromGoogle(int amount) {
        System.out.println("[GooglePay] Amount Paid: "+amount);
    }
}


public class AbstrationWithInterfaces {

    public static void main(String[] args) {


    }

}

```

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.


Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Help
AbstrationWithInterfaces.java

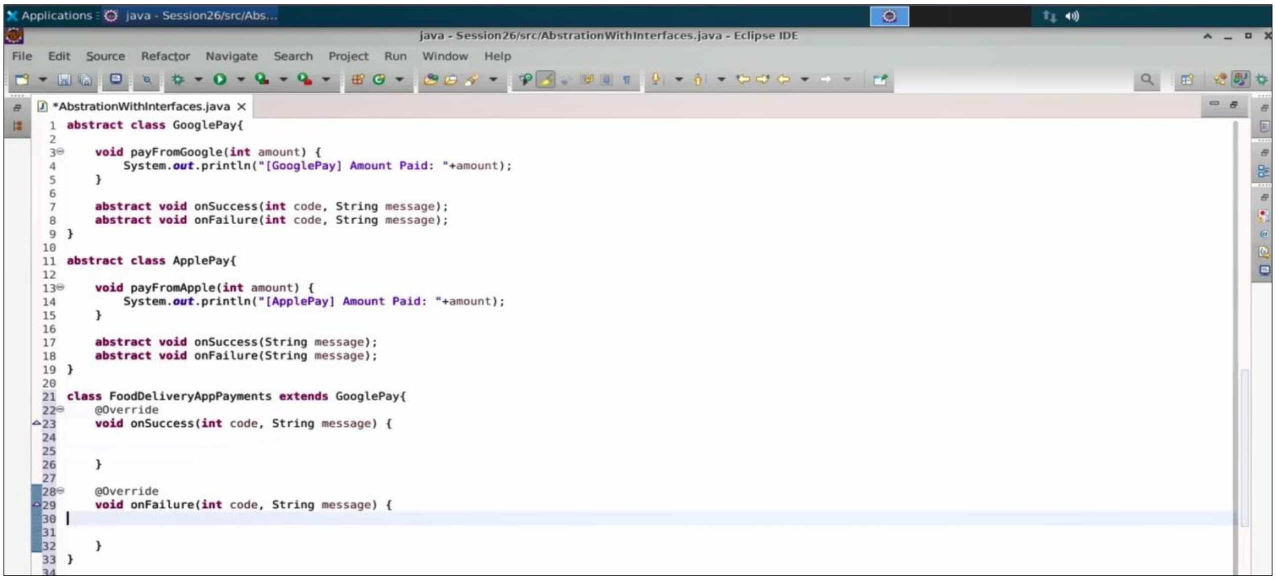
```java
abstract class GooglePay{

    void payFromGoogle(int amount) {
        System.out.println("[GooglePay] Amount Paid: "+amount);
    }

    abstract void onSuccess(int code, Stirng message);
    abstract void onFailure(int code, String message);
}


public class AbstrationWithInterfaces {

    public static void main(String[] args) {


    }

}
```

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.


Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Help
AbstrationWithInterfaces.java

```java
abstract class GooglePay{

    void payFromGoogle(int amount) {
        System.out.println("[GooglePay] Amount Paid: "+amount);
    }

    abstract void onSuccess(int code, String message);
    abstract void onFailure(int code, String message);
}

abstract class ApplePay{

    void payFromApple(int amount) {
        System.out.println("[ApplePay] Amount Paid: "+amount);
    }

    abstract void onSuccess(String message);
    abstract void onFailure(String message);
}

public class AbstrationWithInterfaces {

    public static void main(String[] args) {


    }

}
```

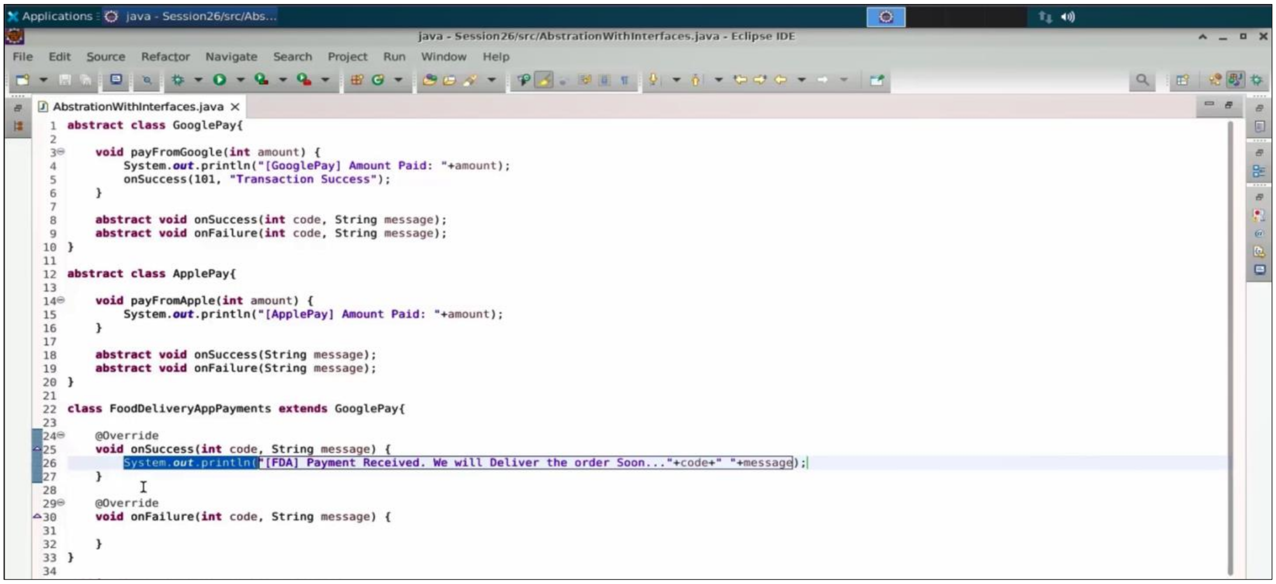
2.4 Create a food delivery app with a payment module called FoodDeliveryAppPayments that extends Google Pay, defining success and failure methods as per the Google method with the code and message.


Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*AbstrationWithInterfaces.java

```java
abstract class GooglePay{

    void payFromGoogle(int amount) {
        System.out.println("[GooglePay] Amount Paid: "+amount);
    }

    abstract void onSuccess(int code, String message);
    abstract void onFailure(int code, String message);
}

abstract class ApplePay{

    void payFromApple(int amount) {
        System.out.println("[ApplePay] Amount Paid: "+amount);
    }

    abstract void onSuccess(String message);
    abstract void onFailure(String message);
}

class FoodDeliveryAppPayments extends GooglePay{
    @Override
    void onSuccess(int code, String message) {


    }

    @Override
    void onFailure(int code, String message) {


    }
}
```

Step 3: Execute the code with example data

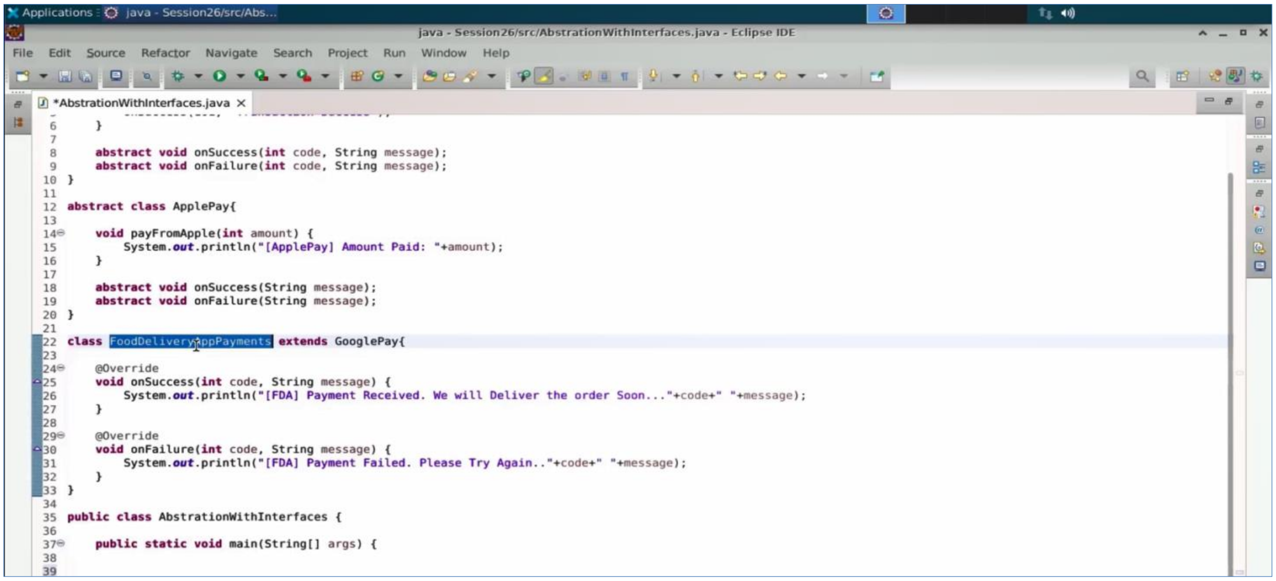
3.1 Once the payment is made, Google will execute the success method, sending code 101 and the message "transaction success." Print "FDA payment received; we will deliver the order soon" in the food delivery app to handle this success case.



Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
AbstrationWithInterfaces.java

```java
abstract class GooglePay{

    void payFromGoogle(int amount) {
        System.out.println("[GooglePay] Amount Paid: "+amount);
        onSuccess(101, "Transaction Success");
    }

    abstract void onSuccess(int code, String message);
    abstract void onFailure(int code, String message);
}

abstract class ApplePay{

    void payFromApple(int amount) {
        System.out.println("[ApplePay] Amount Paid: "+amount);
    }

    abstract void onSuccess(String message);
    abstract void onFailure(String message);
}

class FoodDeliveryAppPayments extends GooglePay{

    @Override
    void onSuccess(int code, String message) {
        System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+" "+message);
    }

    @Override
    void onFailure(int code, String message) {

    }
}
```

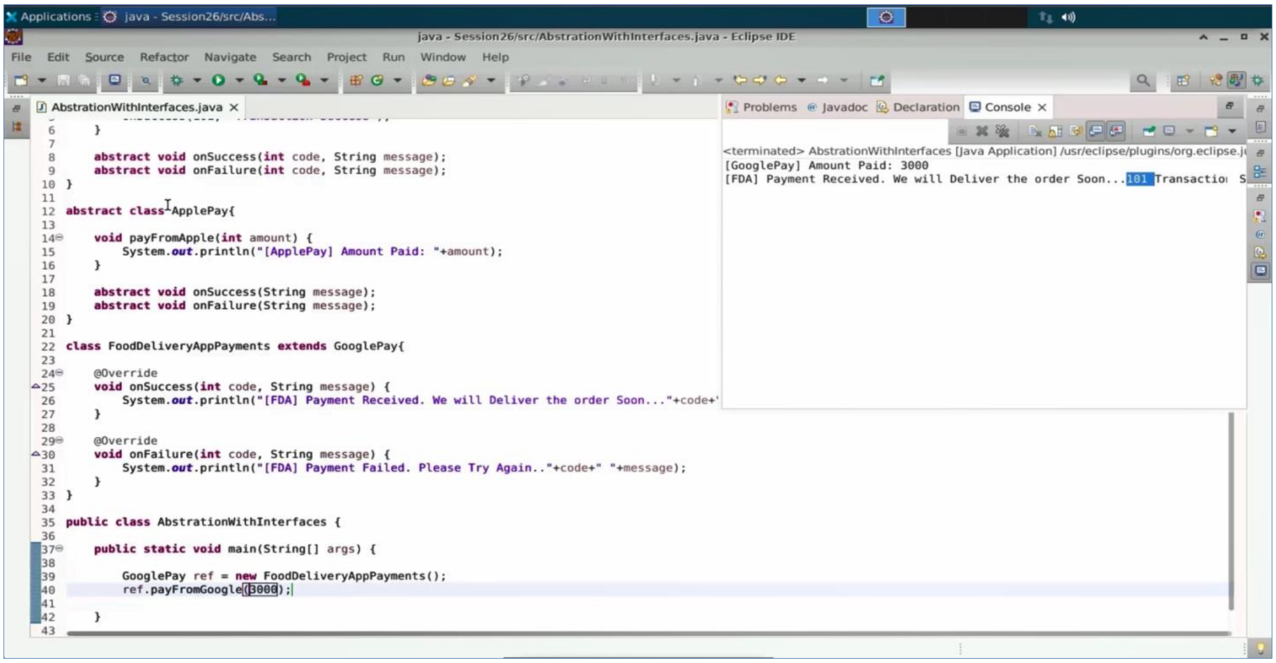
3.2 Same way, if something goes wrong, it can be changed as payment failed over here. And here you can give as, please try again, with the code and the message. Later in the execution phase, you will create the object of your food delivery app payments.



Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*AbstrationWithInterfaces.java

```java
    }

    abstract void onSuccess(int code, String message);
    abstract void onFailure(int code, String message);
}

abstract class ApplePay{

    void payFromApple(int amount) {
        System.out.println("[ApplePay] Amount Paid: "+amount);
    }

    abstract void onSuccess(String message);
    abstract void onFailure(String message);
}

class FoodDeliveryAppPayments extends GooglePay{

    @Override
    void onSuccess(int code, String message) {
        System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+" "+message);
    }

    @Override
    void onFailure(int code, String message) {
        System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
    }
}

public class AbstrationWithInterfaces {

    public static void main(String[] args) {

```

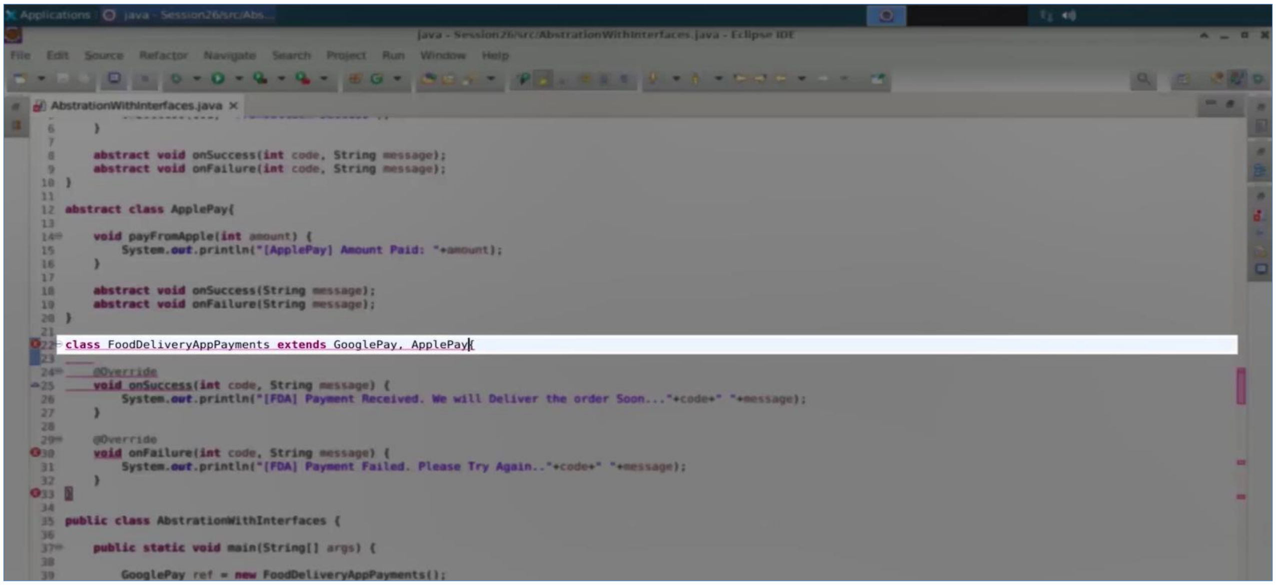
3.3 Let us write a polymorphic statement. Suppose you are using Google Pay as a reference for this food delivery app payment. Then, you can execute this method and write it as pay from Google, with the amount as 3000. Run the code. The Google Pay amount is paid, and you receive a message inside your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


Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
AbstrationWithInterfaces.java
Problems Javadoc Declaration Console

```java
    }

    abstract void onSuccess(int code, String message);
    abstract void onFailure(int code, String message);
}

abstract class ApplePay{

    void payFromApple(int amount) {
        System.out.println("[ApplePay] Amount Paid: "+amount);
    }

    abstract void onSuccess(String message);
    abstract void onFailure(String message);
}

class FoodDeliveryAppPayments extends GooglePay{

    @Override
    void onSuccess(int code, String message) {
        System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+'
    }

    @Override
    void onFailure(int code, String message) {
        System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
    }
}

public class AbstrationWithInterfaces {

    public static void main(String[] args) {

        GooglePay ref = new FoodDeliveryAppPayments();
        ref.payFromGoogle(3000);

    }
```

```
<terminated> AbstrationWithInterfaces [Java Application] /usr/eclipse/plugins/org.eclipse.j
[GooglePay] Amount Paid: 3000
[FDA] Payment Received. We will Deliver the order Soon...101 Transactio  S
```

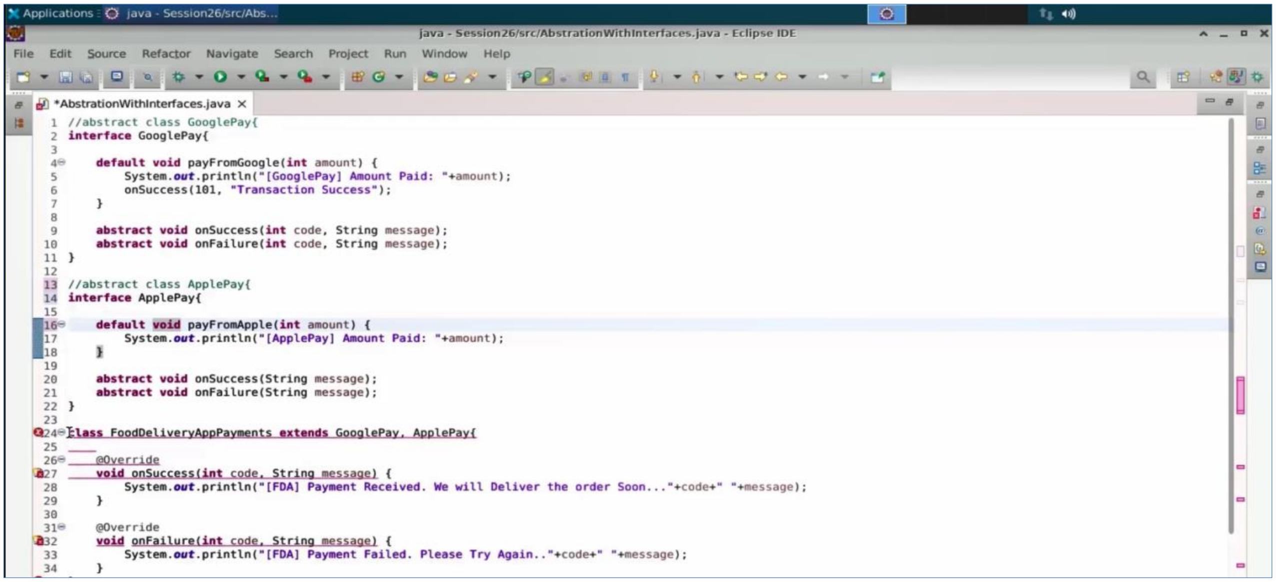
3.4 Sometimes, as a developer, you may wish to integrate Apple Pay as well. This is where abstraction comes into play. If you implement Apple Pay using abstraction, you are not required to use multiple inheritance. Abstraction with interfaces will eliminate this challenge.


java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
AbstrationWithInterfaces.java

```java
    }

    abstract void onSuccess(int code, String message);
    abstract void onFailure(int code, String message);
}

abstract class ApplePay{

    void payFromApple(int amount) {
        System.out.println("[ApplePay] Amount Paid: "+amount);
    }

    abstract void onSuccess(String message);
    abstract void onFailure(String message);
}

class FoodDeliveryAppPayments extends GooglePay, ApplePay{

    @Override
    void onSuccess(int code, String message) {
        System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+" "+message);
    }

    @Override
    void onFailure(int code, String message) {
        System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
    }
}

public class AbstrationWithInterfaces {

    public static void main(String[] args) {

        GooglePay ref = new FoodDeliveryAppPayments();
```

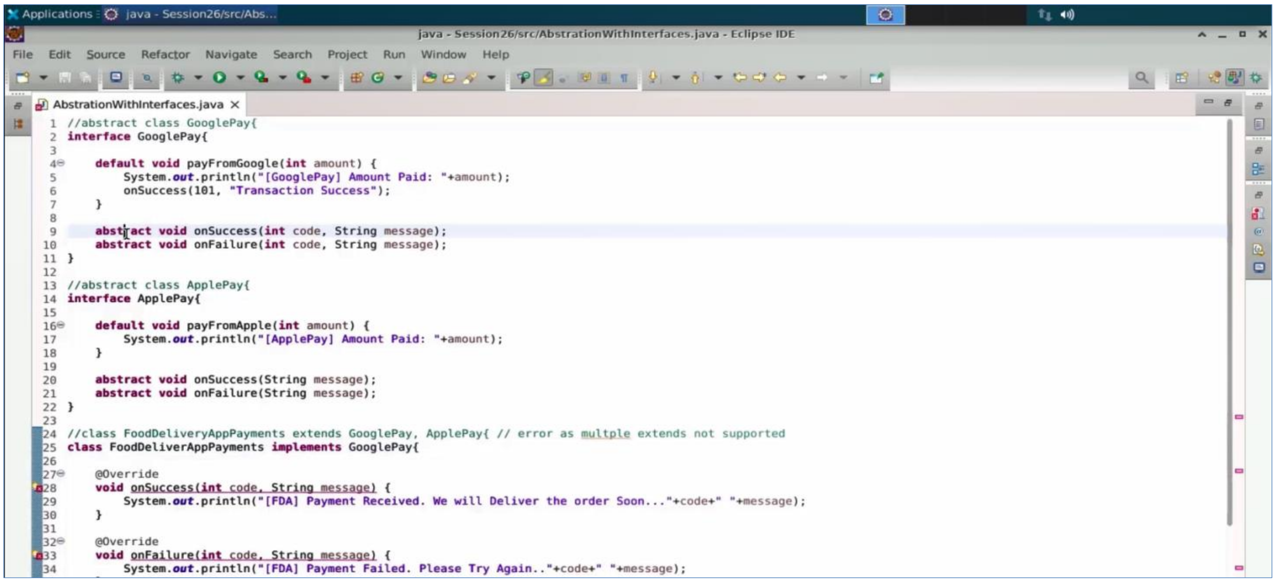
Step 4: Make the abstract class as an interface

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.


Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*AbstrationWithInterfaces.java

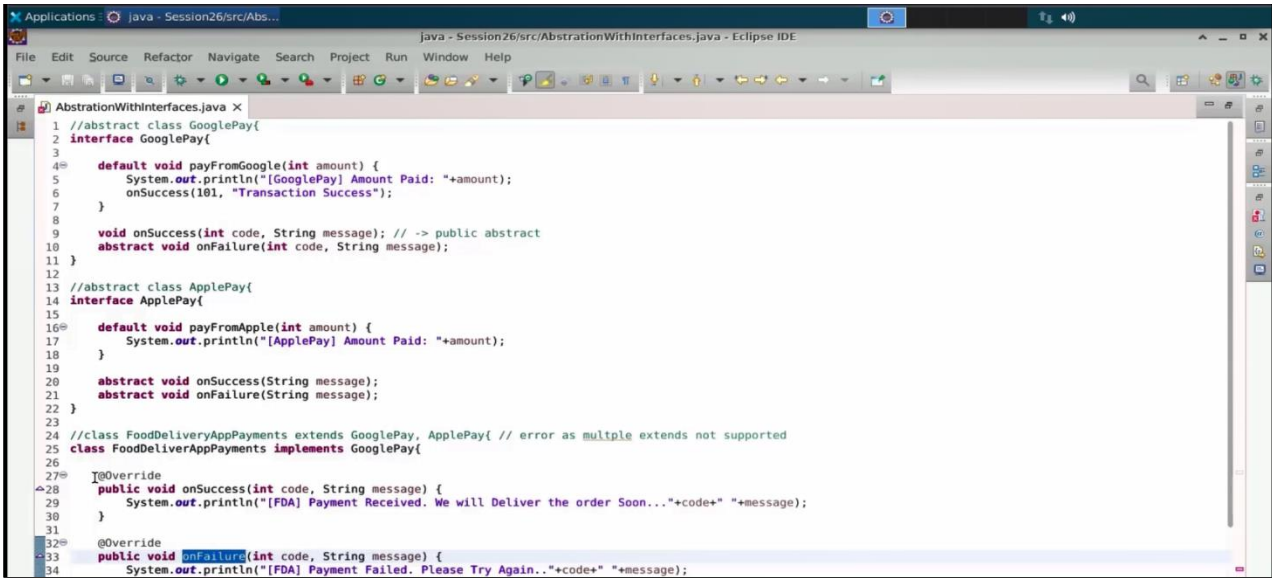
```java
//abstract class GooglePay{
interface GooglePay{

    default void payFromGoogle(int amount) {
        System.out.println("[GooglePay] Amount Paid: "+amount);
        onSuccess(101, "Transaction Success");
    }

    abstract void onSuccess(int code, String message);
    abstract void onFailure(int code, String message);
}

//abstract class ApplePay{
interface ApplePay{

    default void payFromApple(int amount) {
        System.out.println("[ApplePay] Amount Paid: "+amount);
    }

    abstract void onSuccess(String message);
    abstract void onFailure(String message);
}

class FoodDeliveryAppPayments extends GooglePay, ApplePay{

    @Override
    void onSuccess(int code, String message) {
        System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+" "+message);
    }

    @Override
    void onFailure(int code, String message) {
        System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
    }
```

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.


Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Help
AbstrationWithInterfaces.java

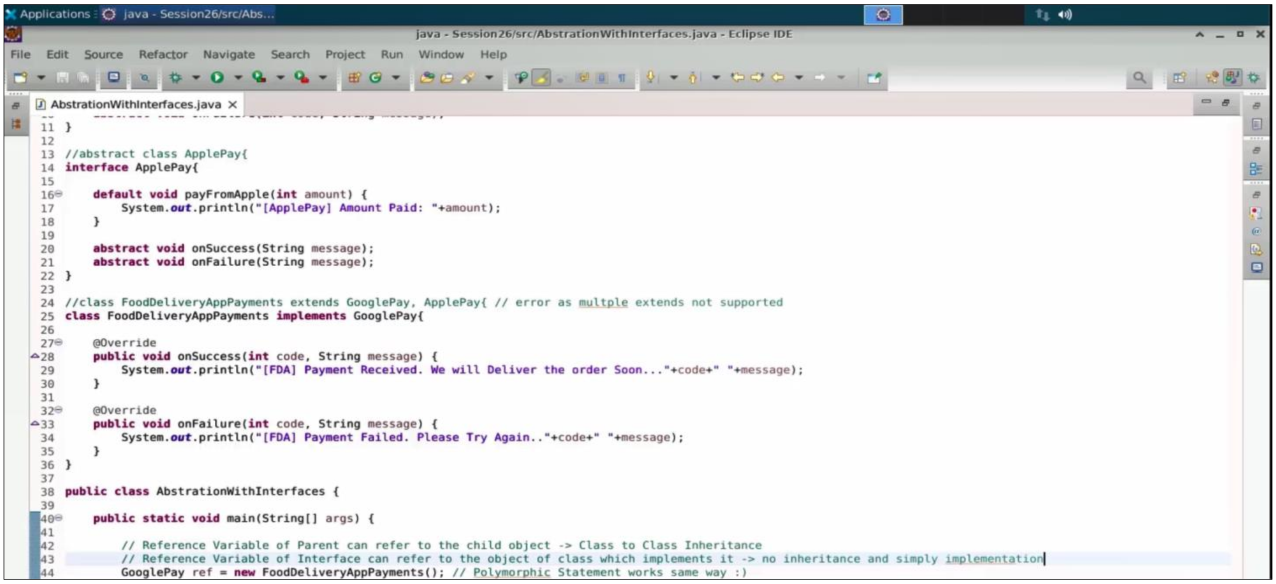
```java
//abstract class GooglePay{
interface GooglePay{

    default void payFromGoogle(int amount) {
        System.out.println("[GooglePay] Amount Paid: "+amount);
        onSuccess(101, "Transaction Success");
    }

    abstract void onSuccess(int code, String message);
    abstract void onFailure(int code, String message);
}

//abstract class ApplePay{
interface ApplePay{

    default void payFromApple(int amount) {
        System.out.println("[ApplePay] Amount Paid: "+amount);
    }

    abstract void onSuccess(String message);
    abstract void onFailure(String message);
}

//class FoodDeliveryAppPayments extends GooglePay, ApplePay{ // error as multple extends not supported
class FoodDeliverAppPayments implements GooglePay{

    @Override
    void onSuccess(int code, String message) {
        System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+" "+message);
    }

    @Override
    void onFailure(int code, String message) {
        System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
```

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.


Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Help
AbstrationWithInterfaces.java

```java
//abstract class GooglePay{
interface GooglePay{

    default void payFromGoogle(int amount) {
        System.out.println("[GooglePay] Amount Paid: "+amount);
        onSuccess(101, "Transaction Success");
    }

    void onSuccess(int code, String message); // -> public abstract
    abstract void onFailure(int code, String message);
}

//abstract class ApplePay{
interface ApplePay{

    default void payFromApple(int amount) {
        System.out.println("[ApplePay] Amount Paid: "+amount);
    }

    abstract void onSuccess(String message);
    abstract void onFailure(String message);
}

//class FoodDeliveryAppPayments extends GooglePay, ApplePay{ // error as multple extends not supported
class FoodDeliverAppPayments implements GooglePay{

    @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);
```

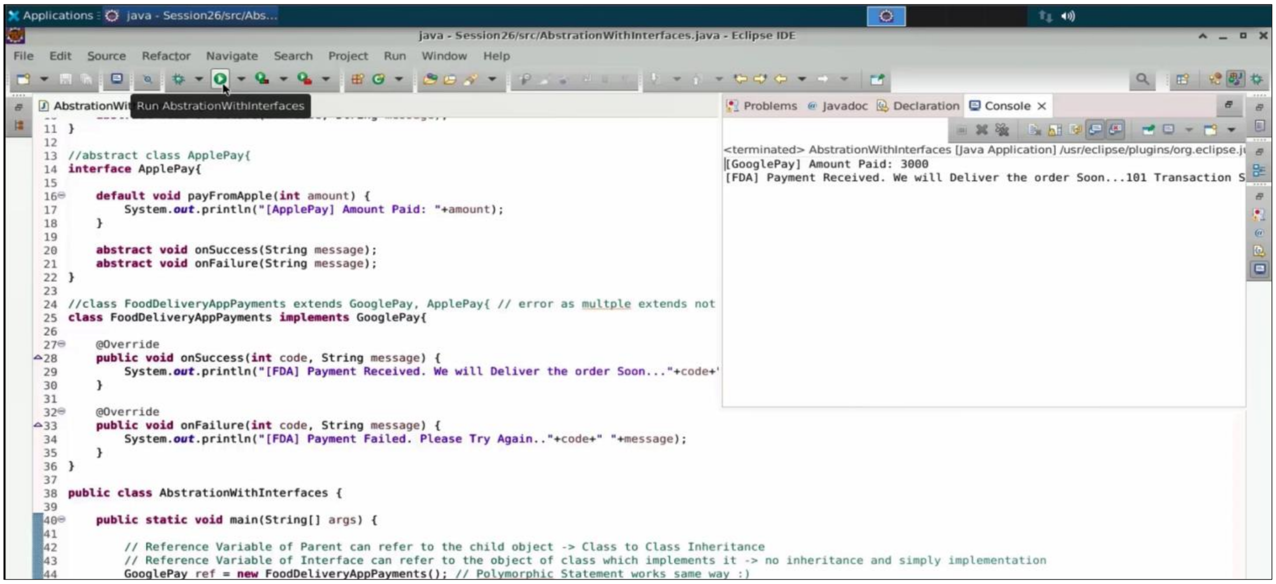
4.4 Here, success and failure are two methods implemented the same way. You can work in the same manner, and the polymorphic statement is not affected. Hence, the polymorphic statement works the same way. It simply means that a reference variable of the parent can refer to the child object. Similarly,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


Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Help
AbstrationWithInterfaces.java

```
}

//abstract class ApplePay{
interface ApplePay{

    default void payFromApple(int amount) {
        System.out.println("[ApplePay] Amount Paid: "+amount);
    }

    abstract void onSuccess(String message);
    abstract void onFailure(String message);
}

//class FoodDeliveryAppPayments extends GooglePay, ApplePay{ // error as multple extends not supported
class FoodDeliveryAppPayments implements GooglePay{

    @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);
    }
}

public class AbstrationWithInterfaces {

    public static void main(String[] args) {

        // Reference Variable of Parent can refer to the child object -> Class to Class Inheritance
        // Reference Variable of Interface can refer to the object of class which implements it -> no inheritance and simply implementation
        GooglePay ref = new FoodDeliveryAppPayments(); // Polymorphic Statement works same way :)
```

Step 5: Implement the polymorphic statement and execute the code

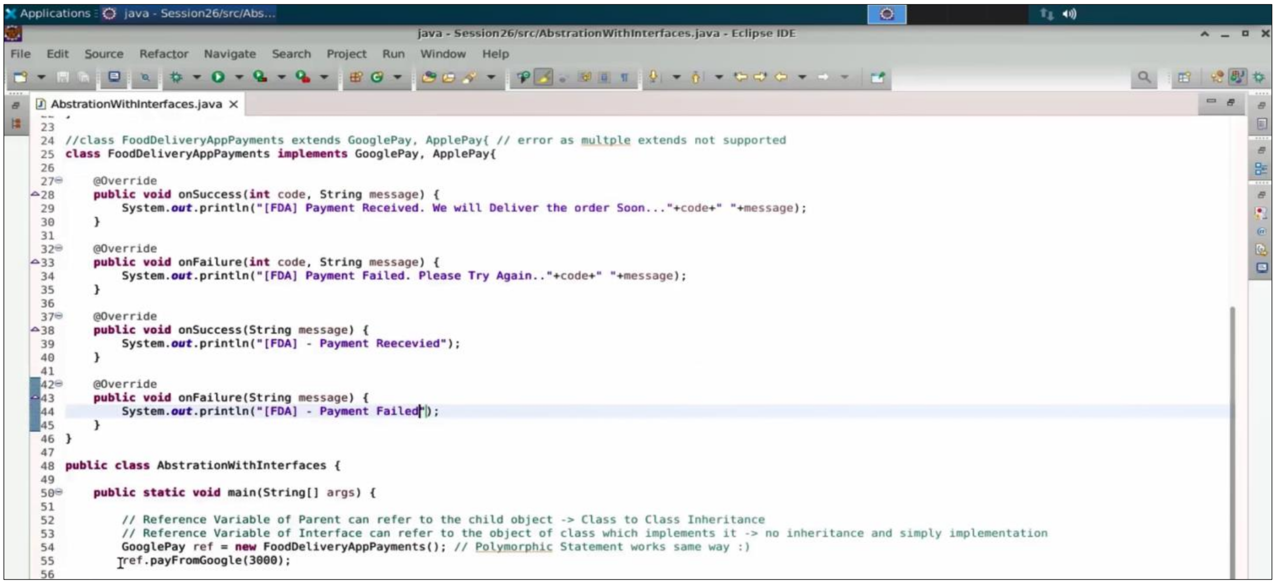
5.1 The main fundamental of a polymorphic statement is that it works for both classes and interfaces, but the fundamental structure is different. When you use "pay from Google" and run the code, the abstraction is implemented in the same way. However, now you are seeing it more crisply with interfaces.


Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
AbstrationWi
Run AbstrationWithInterfaces
Problems Javadoc Declaration Console

```java
}

//abstract class ApplePay{
interface ApplePay{

    default void payFromApple(int amount) {
        System.out.println("[ApplePay] Amount Paid: "+amount);
    }

    abstract void onSuccess(String message);
    abstract void onFailure(String message);
}

//class FoodDeliveryAppPayments extends GooglePay, ApplePay{ // error as multple extends not
class FoodDeliveryAppPayments implements GooglePay{

    @Override
    public void onSuccess(int code, String message) {
        System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+'
    }

    @Override
    public void onFailure(int code, String message) {
        System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
    }
}

public class AbstrationWithInterfaces {

    public static void main(String[] args) {

        // Reference Variable of Parent can refer to the child object -> Class to Class Inheritance
        // Reference Variable of Interface can refer to the object of class which implements it -> no inheritance and simply implementation
        GooglePay ref = new FoodDeliveryAppPayments(); // Polymorphic Statement works same way :)
```

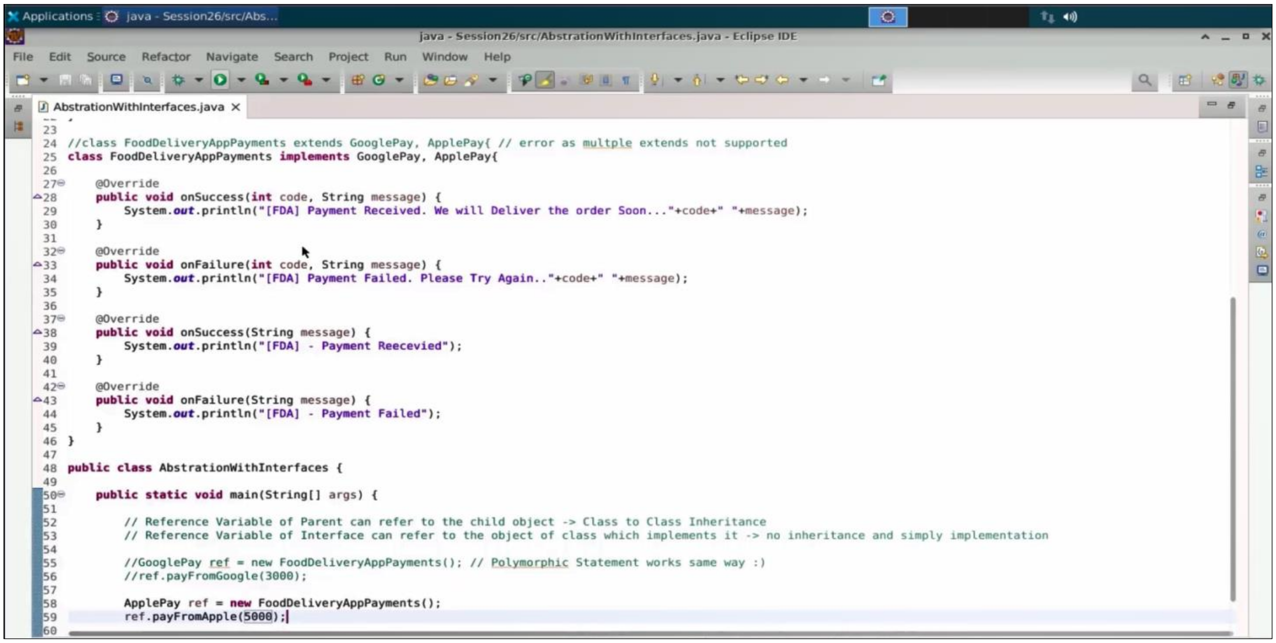
```
<terminated> AbstrationWithInterfaces [Java Application] /usr/eclipse/plugins/org.eclipse.j
[GooglePay] Amount Paid: 3000
[FDA] Payment Received. We will Deliver the order Soon...101 Transaction S
```

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

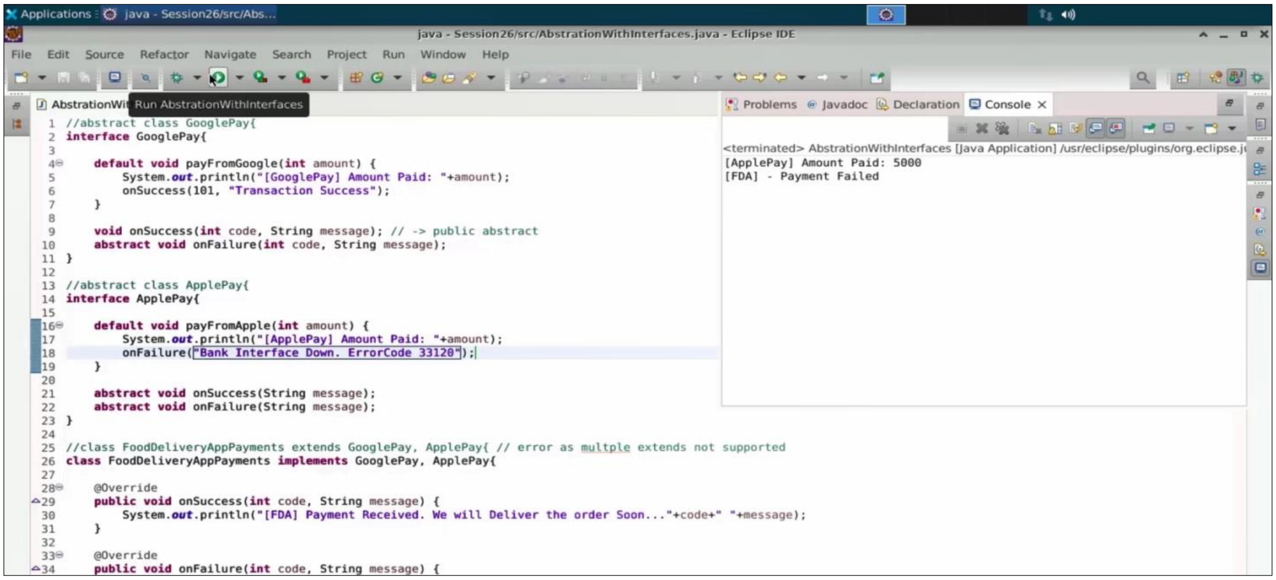
```
//class FoodDeliveryAppPayments extends GooglePay, ApplePay{ // error as multple extends not supported
class FoodDeliveryAppPayments implements GooglePay, ApplePay{

    @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
    public void onSuccess(String message) {
        System.out.println("[FDA] - Payment Reecevied");
    }

    @Override
    public void onFailure(String message) {
        System.out.println("[FDA] - Payment Failed");
    }
}

public class AbstrationWithInterfaces {

    public static void main(String[] args) {

        // Reference Variable of Parent can refer to the child object -> Class to Class Inheritance
        // Reference Variable of Interface can refer to the object of class which implements it -> no inheritance and simply implementation
        GooglePay ref = new FoodDeliveryAppPayments(); // Polymorphic Statement works same way :)
        ref.payFromGoogle(3000);

```

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.


Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
AbstrationWithInterfaces.java

```java

//class FoodDeliveryAppPayments extends GooglePay, ApplePay{ // error as multple extends not supported
class FoodDeliveryAppPayments implements 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
    public void onSuccess(String message) {
        System.out.println("[FDA] - Payment Reecevied");
    }

    @Override
    public void onFailure(String message) {
        System.out.println("[FDA] - Payment Failed");
    }
}

public class AbstrationWithInterfaces {

    public static void main(String[] args) {

        // Reference Variable of Parent can refer to the child object -> Class to Class Inheritance
        // Reference Variable of Interface can refer to the object of class which implements it -> no inheritance and simply implementation

        //GooglePay ref = new FoodDeliveryAppPayments(); // Polymorphic Statement works same way :)
        //ref.payFromGoogle(3000);

        ApplePay ref = new FoodDeliveryAppPayments();
        ref.payFromApple(5000);

```

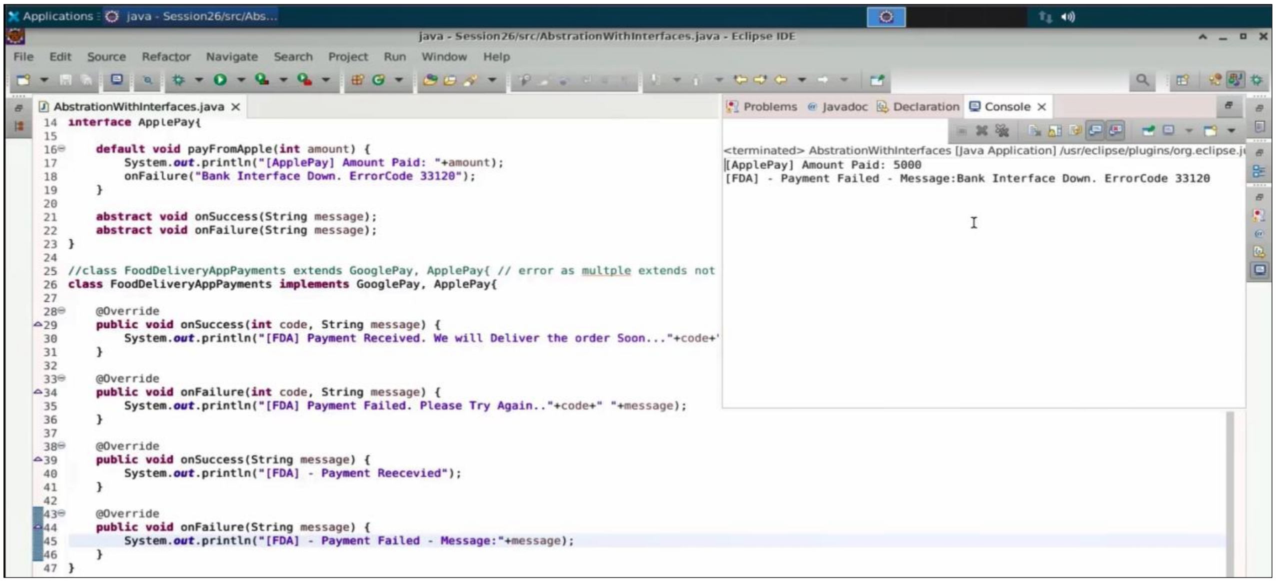
5.4 When you run the code here, it shows "Amount paid: 5000." This comes from Apple Pay, but if you execute the failure method, it shows "Payment failed.


Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
AbstrationWi
Run AbstrationWithInterfaces
Problems
Javadoc
Declaration
Console

```java
//abstract class GooglePay{
interface GooglePay{

    default void payFromGoogle(int amount) {
        System.out.println("[GooglePay] Amount Paid: "+amount);
        onSuccess(101, "Transaction Success");
    }

    void onSuccess(int code, String message); // -> public abstract
    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);
}

//class FoodDeliveryAppPayments extends GooglePay, ApplePay{ // error as multple extends not supported
class FoodDeliveryAppPayments implements 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) {
```

```
<terminated> AbstrationWithInterfaces [Java Application] /usr/eclipse/plugins/org.eclipse.j
[ApplePay] Amount Paid: 5000
[FDA] - Payment Failed
```

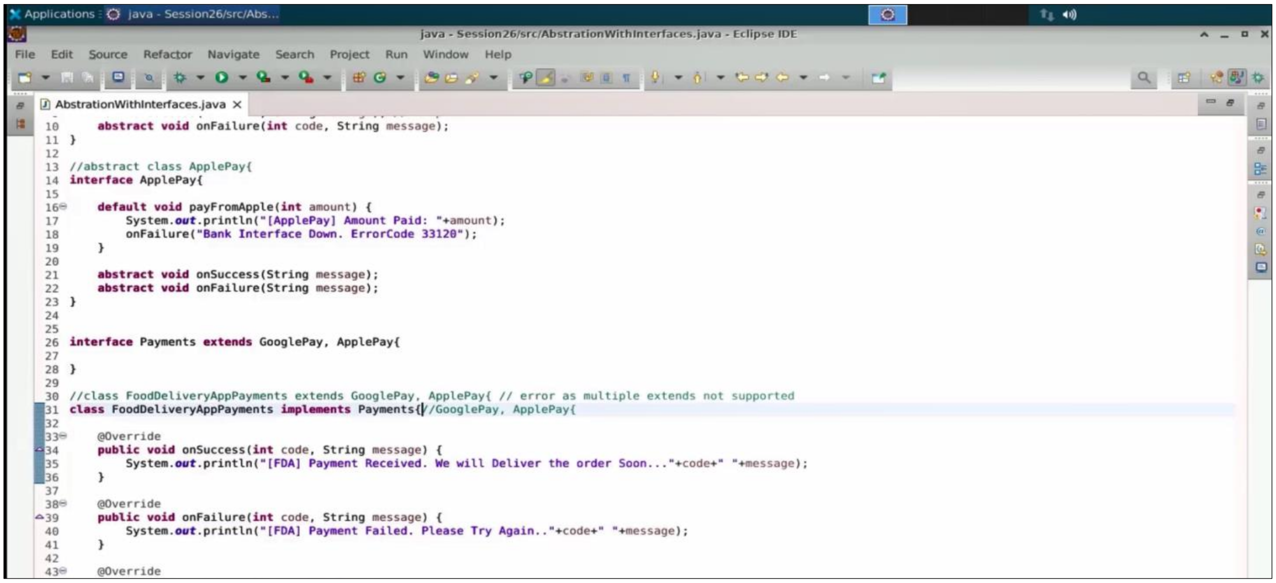
5.5 And with the payment failed, let us come here and display the message. Save it and rerun it. You can see the message as bank interface down, error Code 33120. Thus, you can work with the abstraction, and you can implement it through interfaces, which is the best of the choices rather than working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


Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Help
AbstrationWithInterfaces.java
Problems
Javadoc
Declaration
Console

```java
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);
}

//class FoodDeliveryAppPayments extends GooglePay, ApplePay{ // error as multple extends not
class FoodDeliveryAppPayments implements GooglePay, ApplePay{

    @Override
    public void onSuccess(int code, String message) {
        System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+'
    }

    @Override
    public void onFailure(int code, String message) {
        System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
    }

    @Override
    public void onSuccess(String message) {
        System.out.println("[FDA] - Payment Reecevied");
    }

    @Override
    public void onFailure(String message) {
        System.out.println("[FDA] - Payment Failed - Message:"+message);
    }
}
```

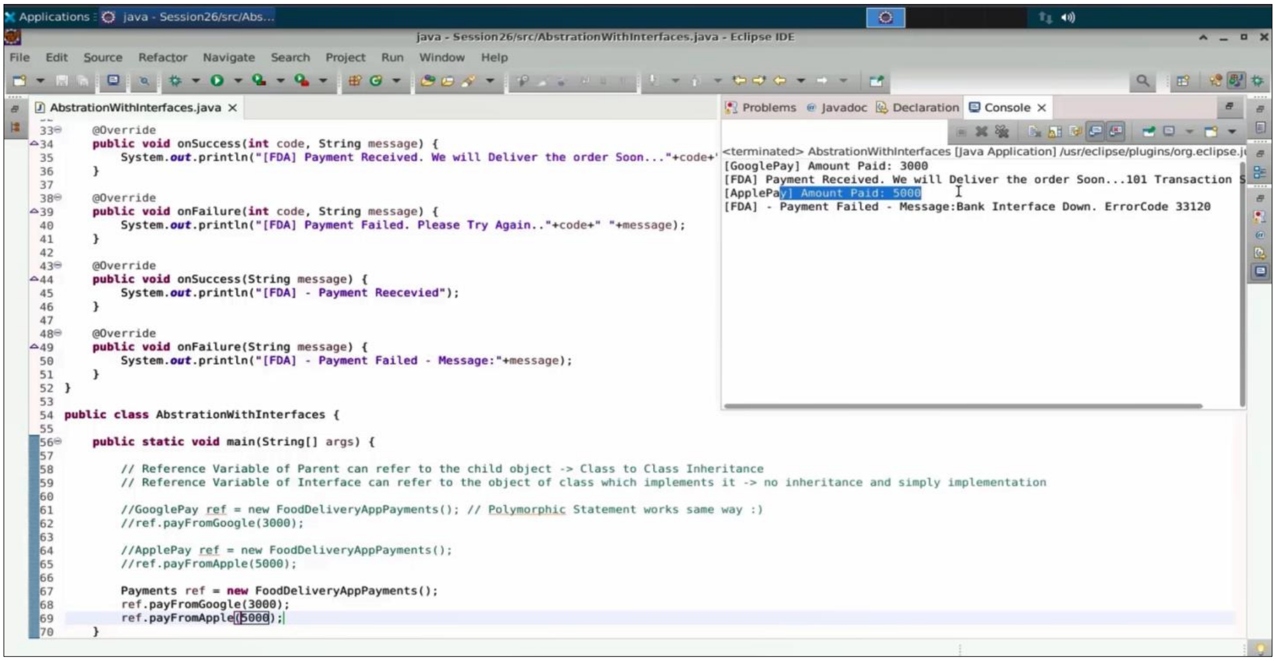
```
<terminated> AbstrationWithInterfaces [Java Application] /usr/eclipse/plugins/org.eclipse.j
[ApplePay] Amount Paid: 5000
[FDA] - Payment Failed - Message:Bank Interface Down. ErrorCode 33120
```

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.


Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Help
AbstrationWithInterfaces.java

```java
    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.


Applications
java - Session26/src/Abs...
java - Session26/src/AbstrationWithInterfaces.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
AbstrationWithInterfaces.java
Problems
Javadoc
Declaration
Console

```java
    @Override
    public void onSuccess(int code, String message) {
        System.out.println("[FDA] Payment Received. We will Deliver the order Soon..."+code+'
    }

    @Override
    public void onFailure(int code, String message) {
        System.out.println("[FDA] Payment Failed. Please Try Again.."+code+" "+message);
    }

    @Override
    public void onSuccess(String message) {
        System.out.println("[FDA] - Payment Reecevied");
    }

    @Override
    public void onFailure(String message) {
        System.out.println("[FDA] - Payment Failed - Message:"+message);
    }
}

public class AbstrationWithInterfaces {

    public static void main(String[] args) {

        // Reference Variable of Parent can refer to the child object -> Class to Class Inheritance
        // Reference Variable of Interface can refer to the object of class which implements it -> no inheritance and simply implementation

        //GooglePay ref = new FoodDeliveryAppPayments(); // Polymorphic Statement works same way :)
        //ref.payFromGoogle(3000);

        //ApplePay ref = new FoodDeliveryAppPayments();
        //ref.payFromApple(5000);

        Payments ref = new FoodDeliveryAppPayments();
        ref.payFromGoogle(3000);
        ref.payFromApple(5000);
    }
```

```
<terminated> AbstrationWithInterfaces [Java Application] /usr/eclipse/plugins/org.eclipse.j
[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.