Lesson 03 Demo 07

Implementing an Abstract Class

Objective: Using abstract classes in Java

Tools required: Eclipse IDE

Prerequisites: None

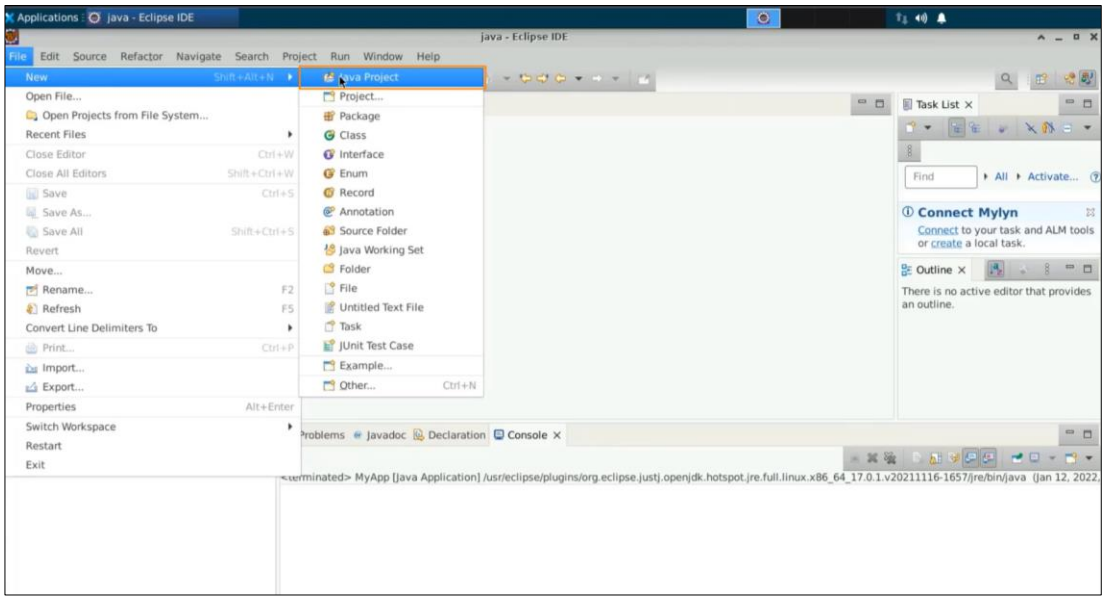
Steps to be followed:

1. Create a class called AbstractDemo, followed by selecting the main method
2. Write a class called cab and create a constructor for cab class
3. Create the object of a regular class
4. Understand the concept of abstract class with examples
5. Execute the code, along with the use of regular methods

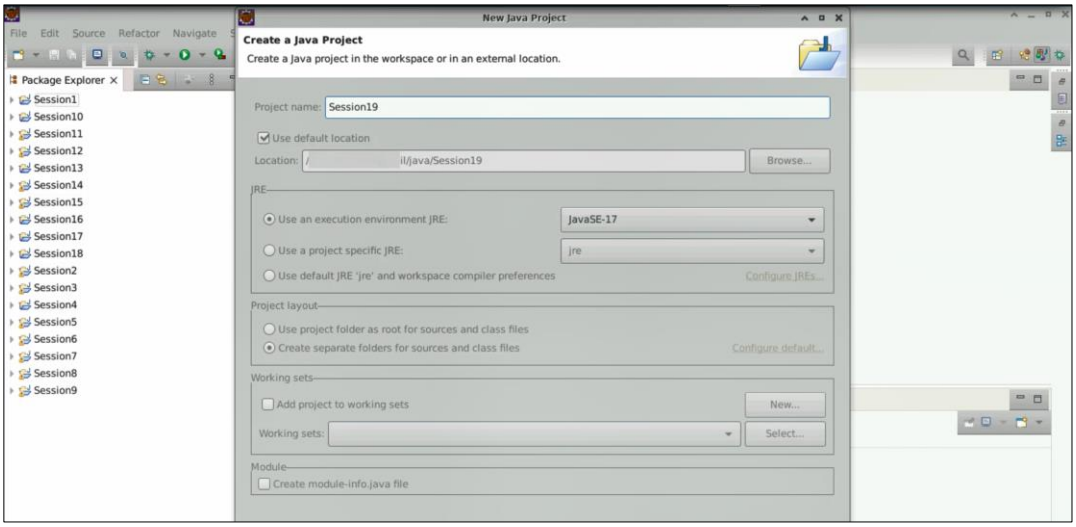
Step 1: Create a class called AbstractDemo, followed by selecting the main method

1.1 Open the **Eclipse IDE**

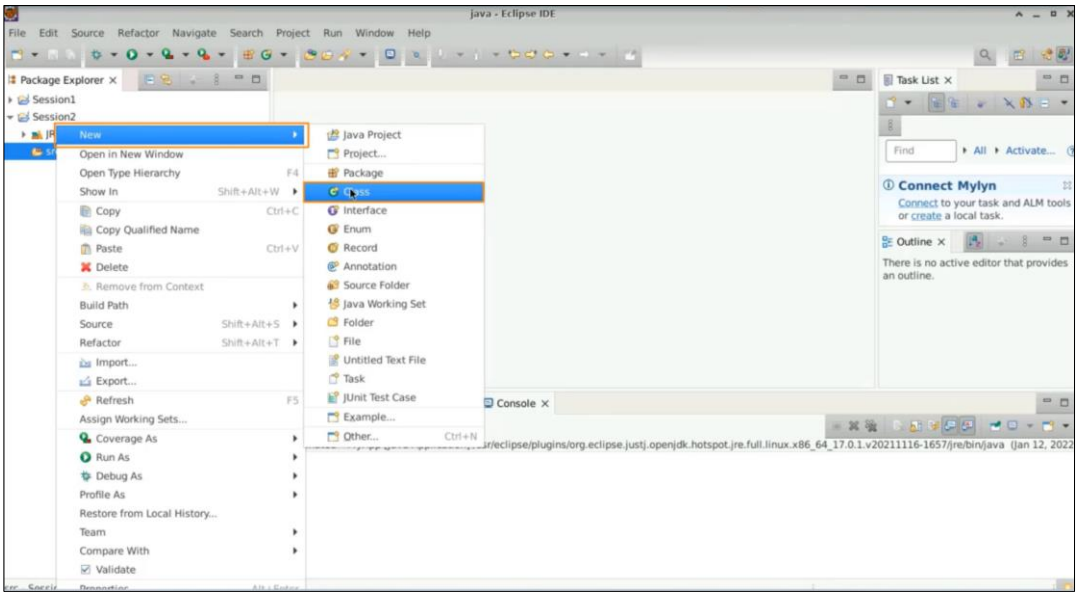
1.2. Select **File**, then **New,** and then **Java project**



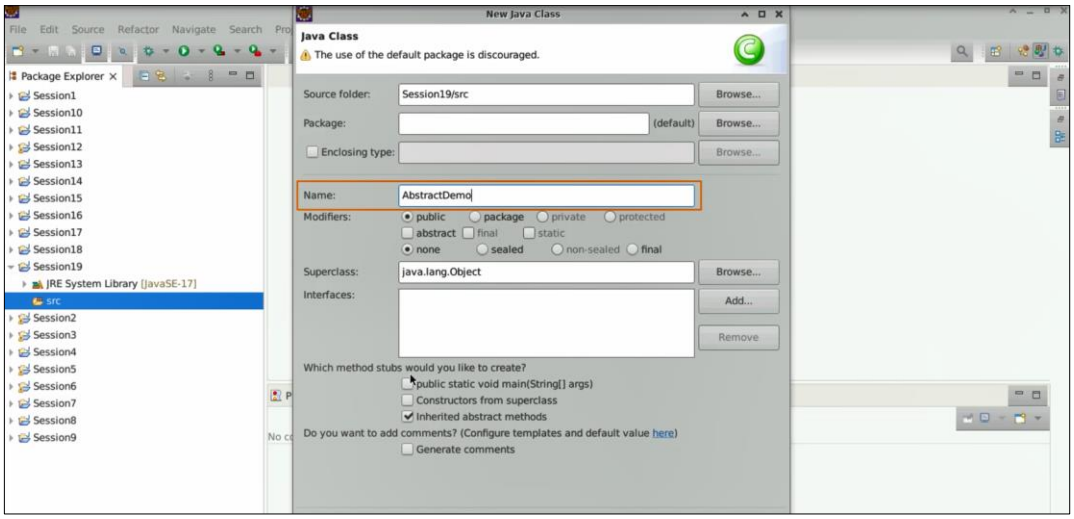
1.3 Name the project **"Session19",** uncheck **"Create a module info dot Java file"**, and press **Finish**



1.4 With a **Session19** on the src, do a right-click and create a **new class**

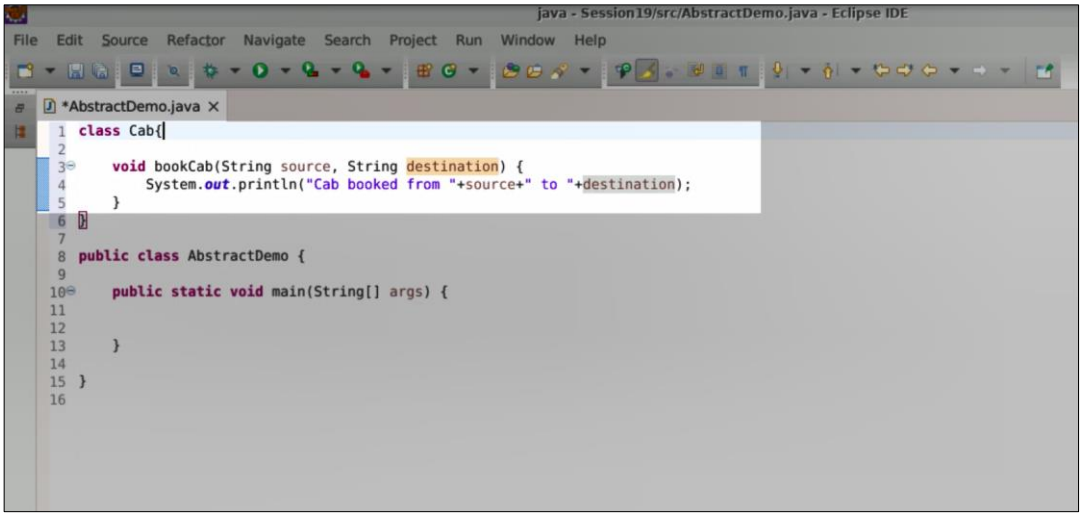


1.5 Name this class as an **AbstractDemo**, then select the **main method,** and then select **finish**

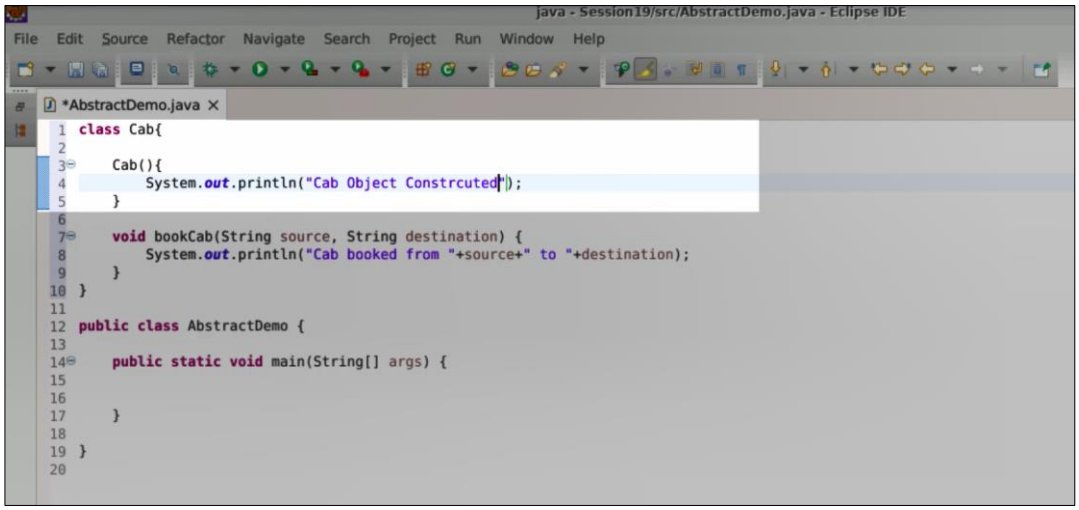


Step 2: Write a class called cab and create a constructor for cab class

2.1 Create a class known as **Cab**. This is a regular class with a method called **bookCab**. Let us specify that it has a source and a destination. Then, write **System.out.println("Cab booked from " + source + " to " + destination);**

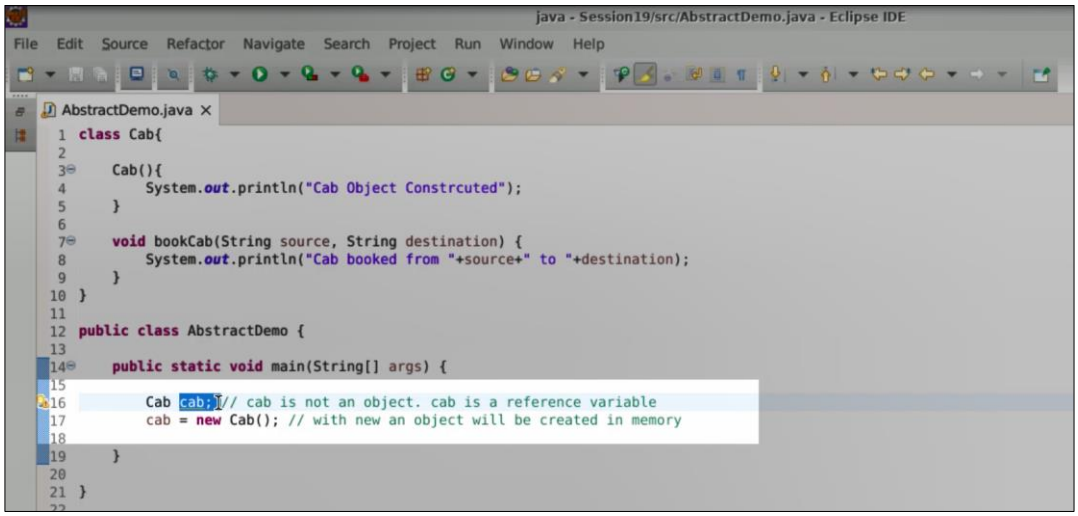


2.2 Create the constructor for the cab class, which is a regular class and you can write as cab object constructed

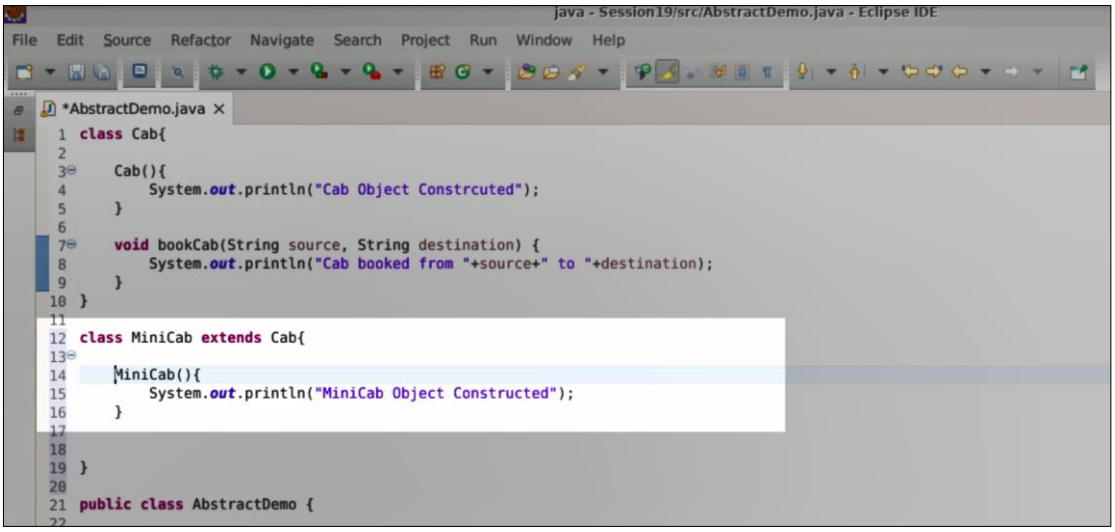


Step 3: Create the object of a regular class

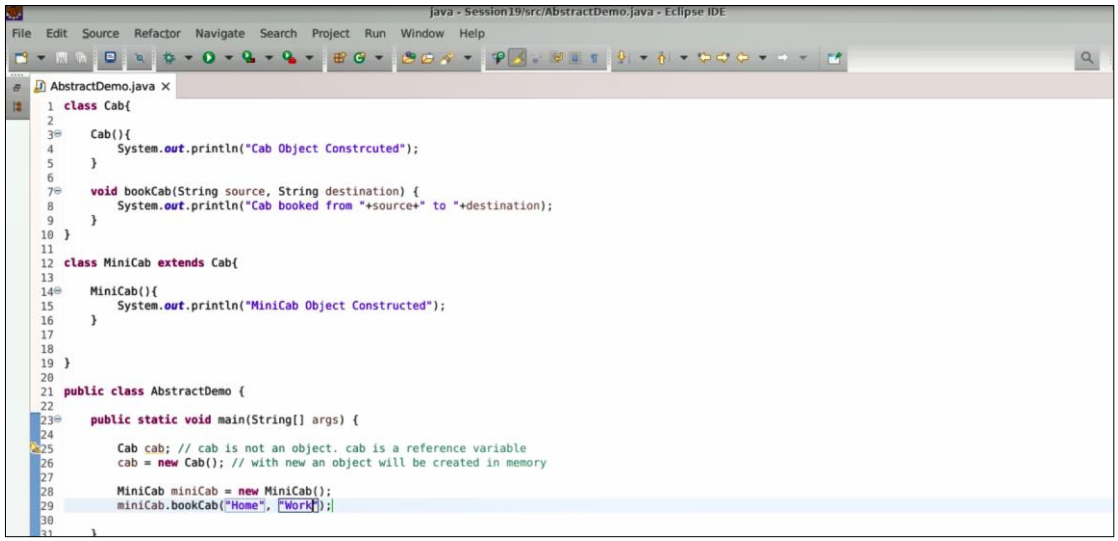
3.1 Create an object of a regular class. Write new **Cab();,**and assign it to a reference variable by writing **cab = new Cab();.** As you know, this is not an object; this is a reference variable. Hence, you can simply state, Cab is not an object. Cab is a reference variable. With new, an object will be created in memory, which is the definition of the new operator.



3.2 If you try to create a class called **MiniCab**, which is the child of Cab, create the constructor and write **'MiniCab Object Constructed'.** You can choose to override the methods from the parent class or skip the override, depending on the requirement

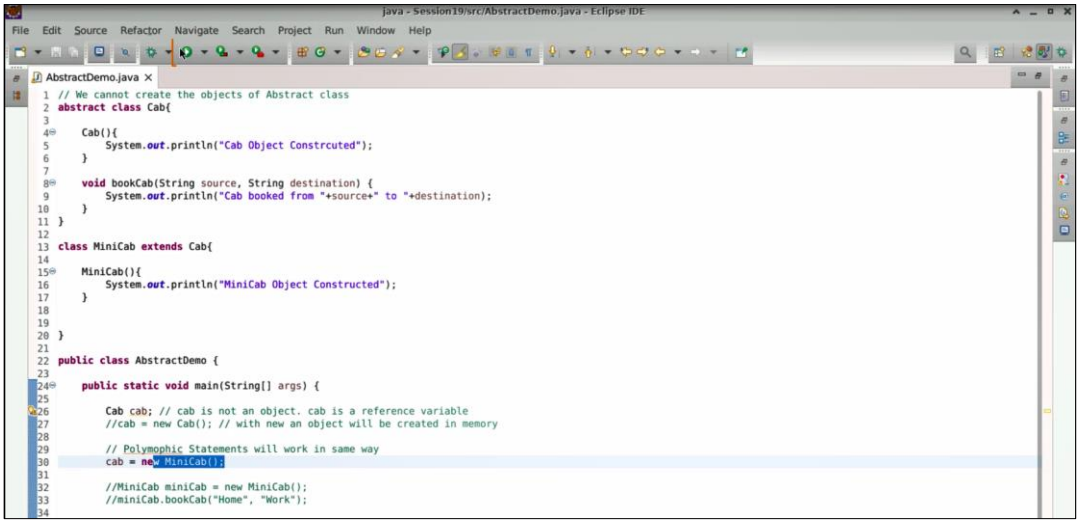


3.3 To create an object of **MiniCab**, you can write **MiniCab miniCab = new MiniCab();**. This creates a direct object and is not a polymorphic statement. With this **MiniCab** object, you can execute the method called **bookCab** from 'home' to 'work'



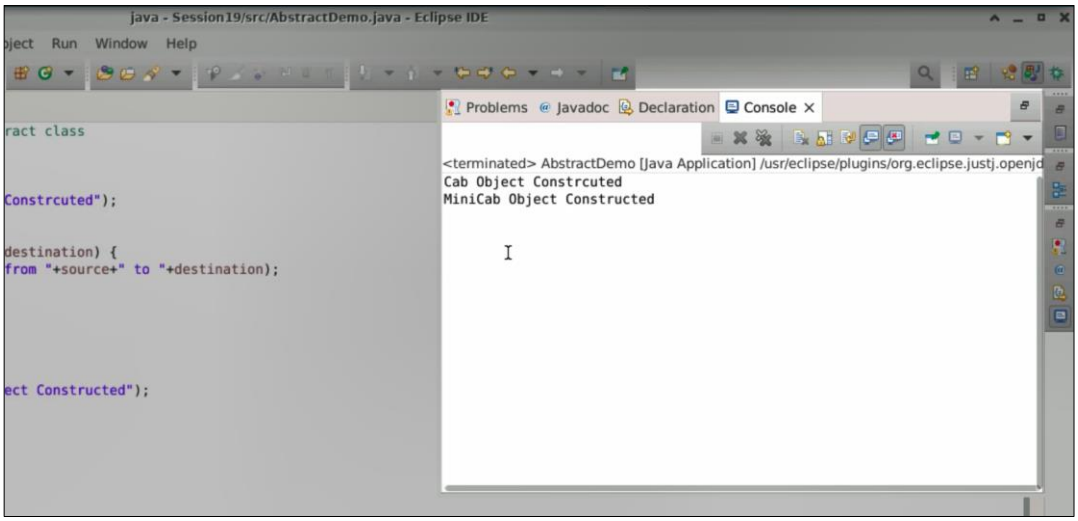
Step 4: Understand the concept of abstract class with examples

4.1 Let us understand what happens when you mark your Cab class as abstract. You will get an error indicating that it 'cannot instantiate the type Cab,' which means you cannot create objects of an abstract class. So, you can comment out the reference variable. This Cab reference can point to the child class. This means you can write **Cab cab = new MiniCab();**. Hence, with the abstract class, polymorphism will work in the same way

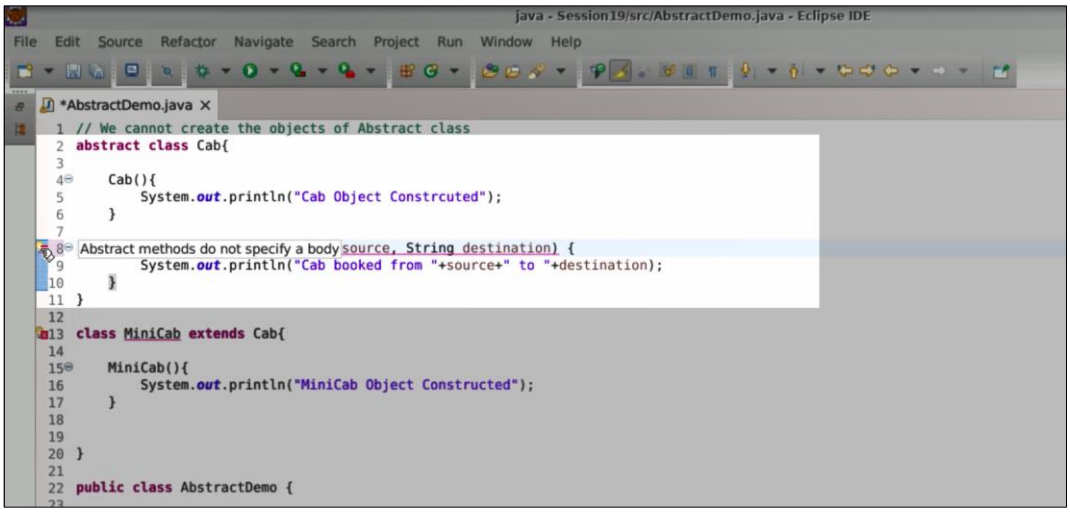


Step 5: Execute the code, along with the use of regular methods

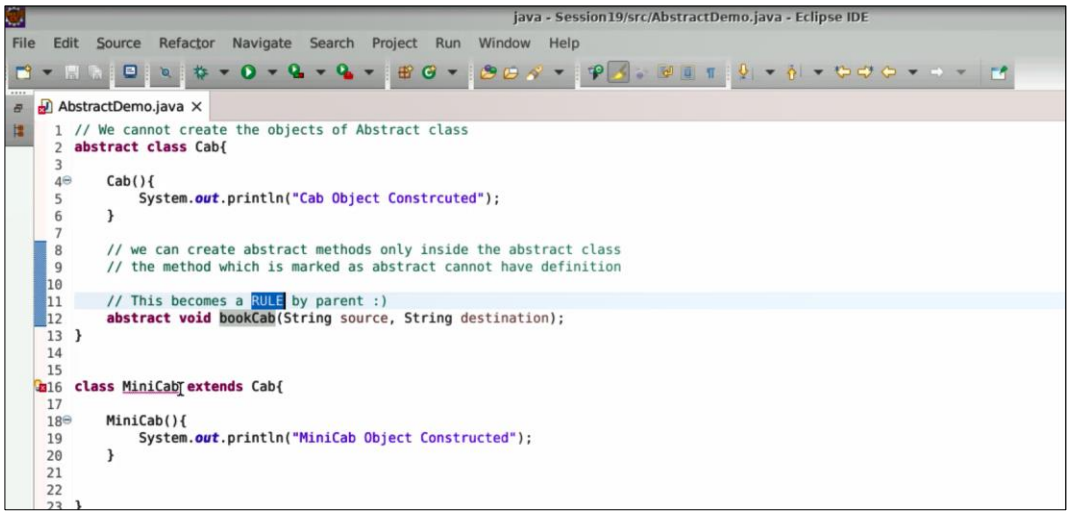
5.1 Run the code. As you can see, the parent constructor is executed first, followed by the child constructor. This means that you cannot create a direct object of the abstract class, but the rule of inheritance is implemented. This means there will be two objects in memory: one from the abstract class Cab and another from the **MiniCab**



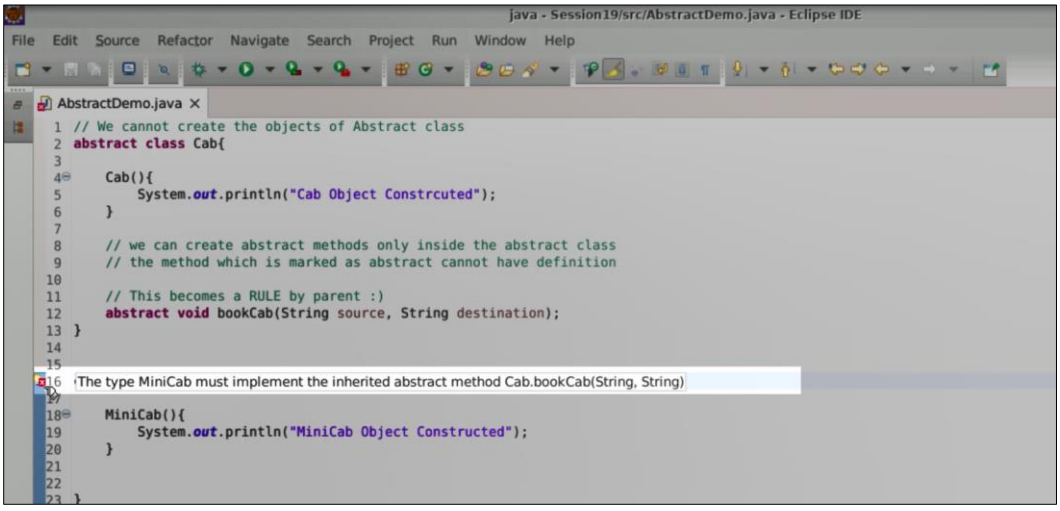
5.2 Why do you need to create an abstract class? The reason is to prevent anyone from creating an object of that class, but this can also be done by marking your constructor as private. The **bookCab** method defined here will give you an error if you mark it as abstract because abstract methods cannot have a body. Therefore, a body for your method, like an abstract method, is not allowed



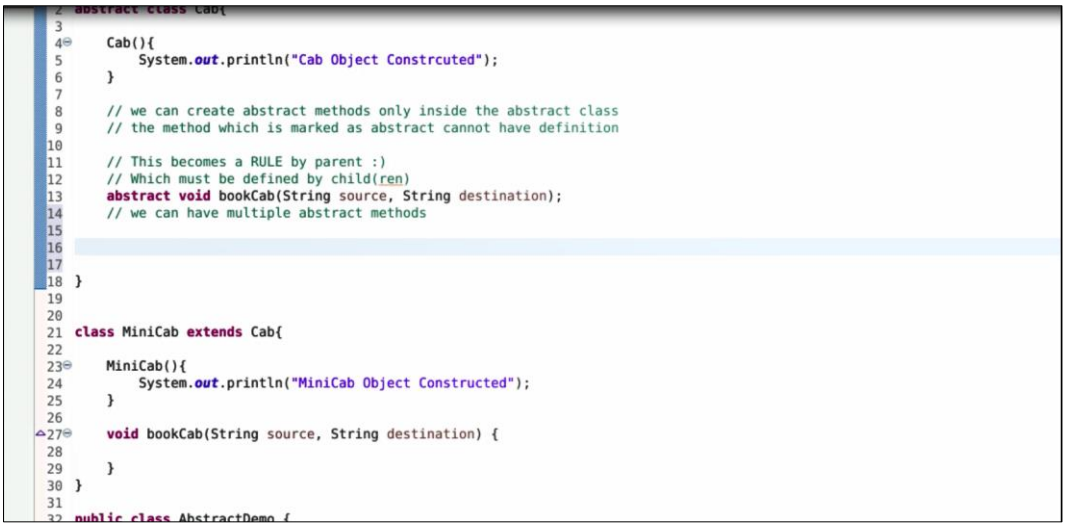
5.3 This abstract keyword is given with the class so that you can create the abstract methods only inside the abstract class. The next thing, which you need to understand is that the method which is marked as abstract cannot have a definition, you cannot define an abstract method. This becomes a rule by the parent



5.4 Consider your **bookCab** method as a rule. You will observe that in the child class, **MiniCab**, you will get an error indicating that it must implement the inherited **bookCab** method. Therefore, the definition of the **bookCab** method must be provided inside the **MiniCab** class

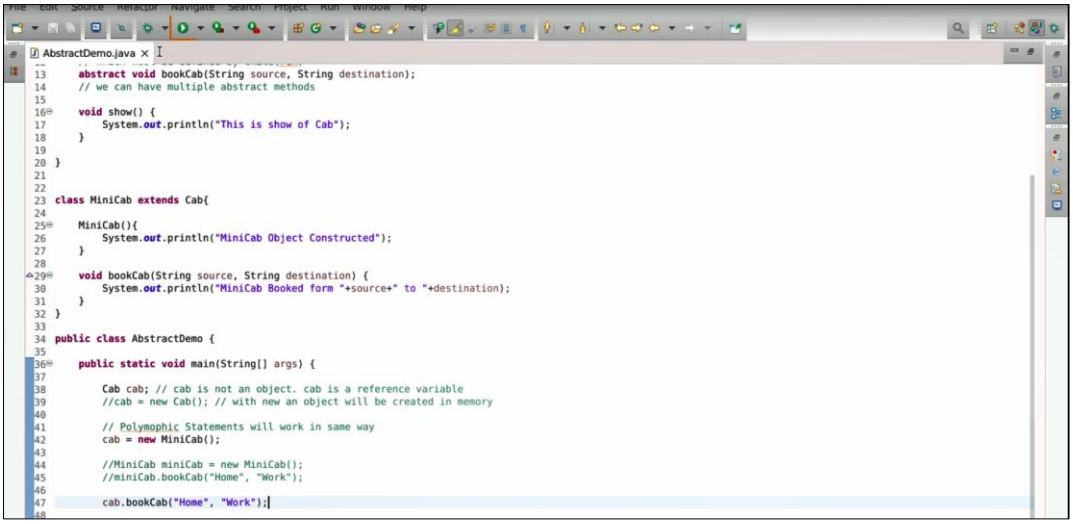


5.5 The parent class has created a rule, and this rule must be defined by its children. Whether there is one child or multiple children, any class that extends Cab must define the **bookCab** method. That is the main rule you need to follow. In addition to the **bookCab** method, you can have multiple abstract methods in the abstract class



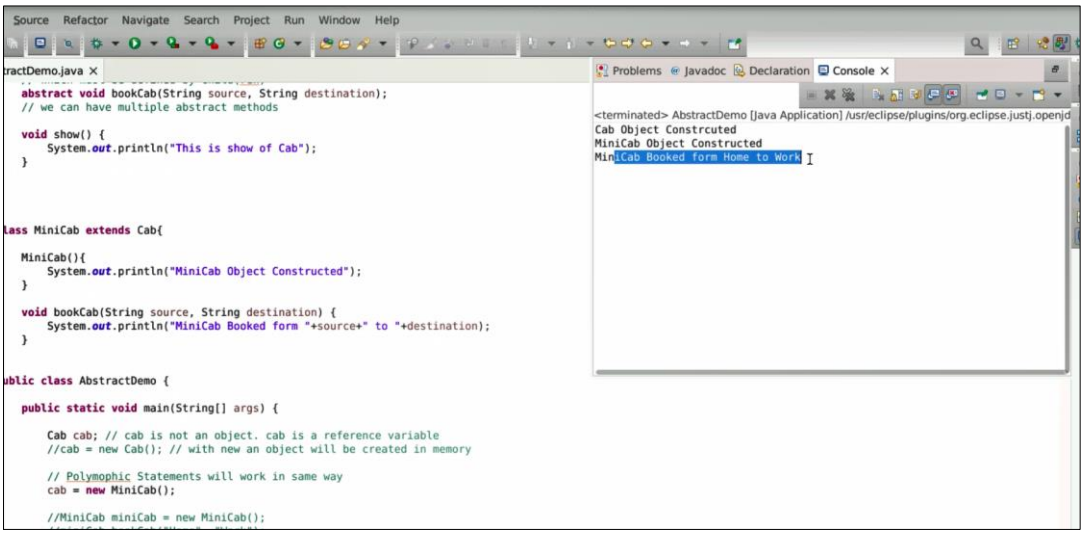
```
abstract class Cab{

    Cab(){
        System.out.println("Cab Object Constrcuted");
    }

    // we can create abstract methods only inside the abstract class
    // the method which is marked as abstract cannot have definition

    // This becomes a RULE by parent :)
    // Which must be defined by child(ren)
    abstract void bookCab(String source, String destination);
    // we can have multiple abstract methods


}


class MiniCab extends Cab{

    MiniCab(){
        System.out.println("MiniCab Object Constructed");
    }

    void bookCab(String source, String destination) {

    }
}

public class AbstractDemo {
```

5.6 At the same time, you can also have regular methods, such as a void show method. For example, write **System.out.println("This is show of Cabs");** Thus, you can have regular methods. Then, write **System.out.println("MiniCab booked from " + source + " to " + destination);** With this cab object, you will be able to execute the method **bookCab** with the source and destination, such as 'home' and 'work'

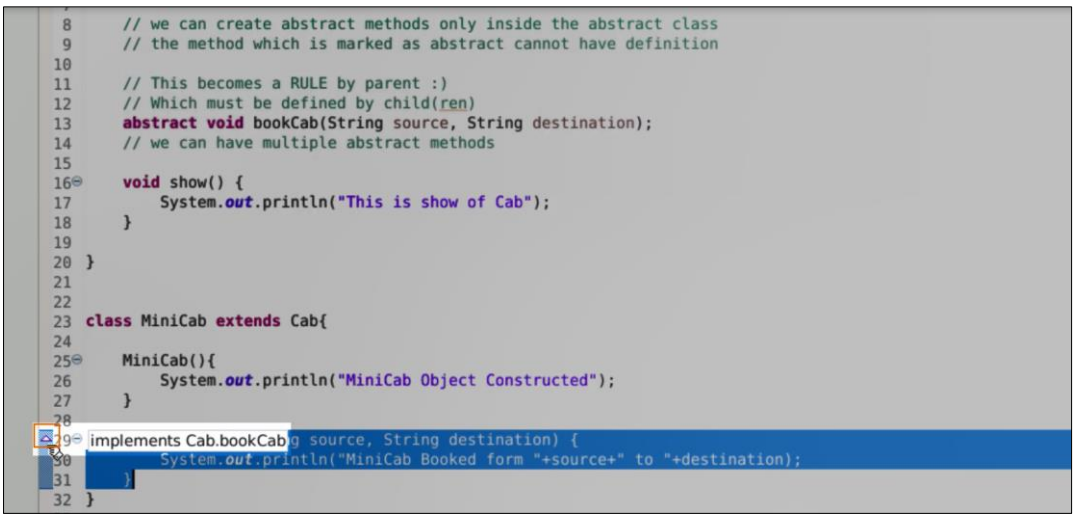


```
AbstractDemo.java
    abstract void bookCab(String source, String destination);
    // we can have multiple abstract methods

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

}

class MiniCab extends Cab{

    MiniCab(){
        System.out.println("MiniCab Object Constructed");
    }

    void bookCab(String source, String destination) {
        System.out.println("MiniCab Booked form "+source+" to "+destination);
    }
}

public class AbstractDemo {

    public static void main(String[] args) {

        Cab cab; // cab is not an object. cab is a reference variable
        //cab = new Cab(); // with new an object will be created in memory

        // Polymophic Statements will work in same way
        cab = new MiniCab();

        //MiniCab miniCab = new MiniCab();
        //miniCab.bookCab("Home", "Work");

        cab.bookCab("Home", "Work");
```

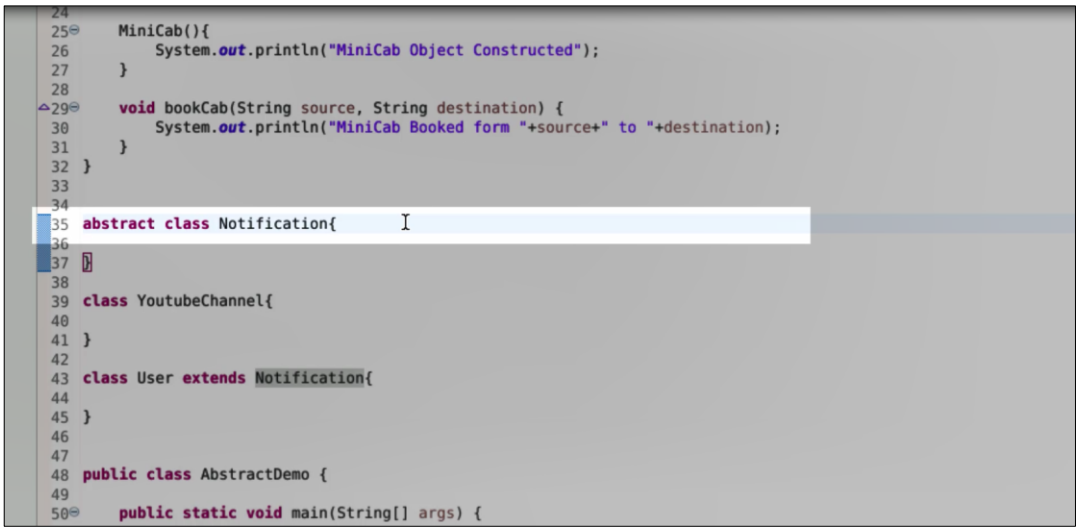
5.7 Run the program. It shows that the **MiniCab** object is constructed, the Cab is constructed, and the **MiniCab** is booked from home to work



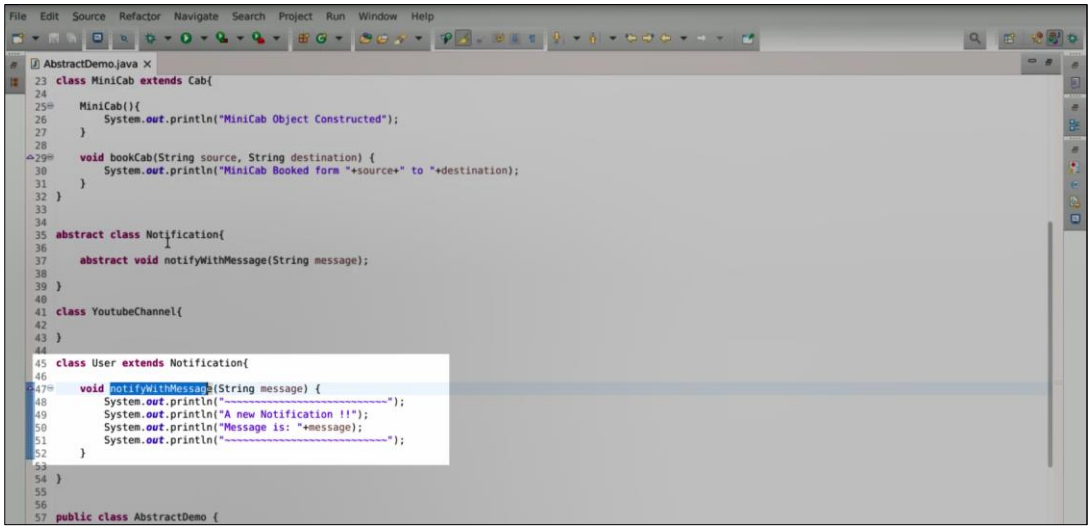
5.8 Whenever you create an abstract class, remember that you have the option to define some methods. When you extend the abstract class, you then work on the method that was abstract by creating its definition. In Eclipse, you will see a white triangle, indicating that it is an implementation of a rule rather than an override



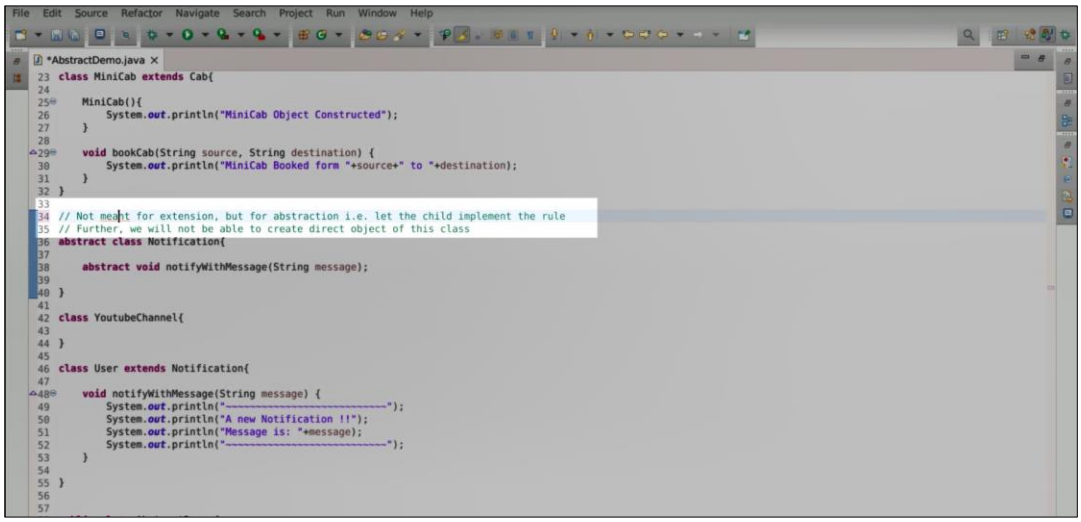
5.9 Let us write that there is a class called **YouTubeChannel**. And there is another class called User. When a user subscribes to a YouTube channel, the user should receive a notification whenever a new video is uploaded on the channel. For the notification, you will create an abstract class called Notification



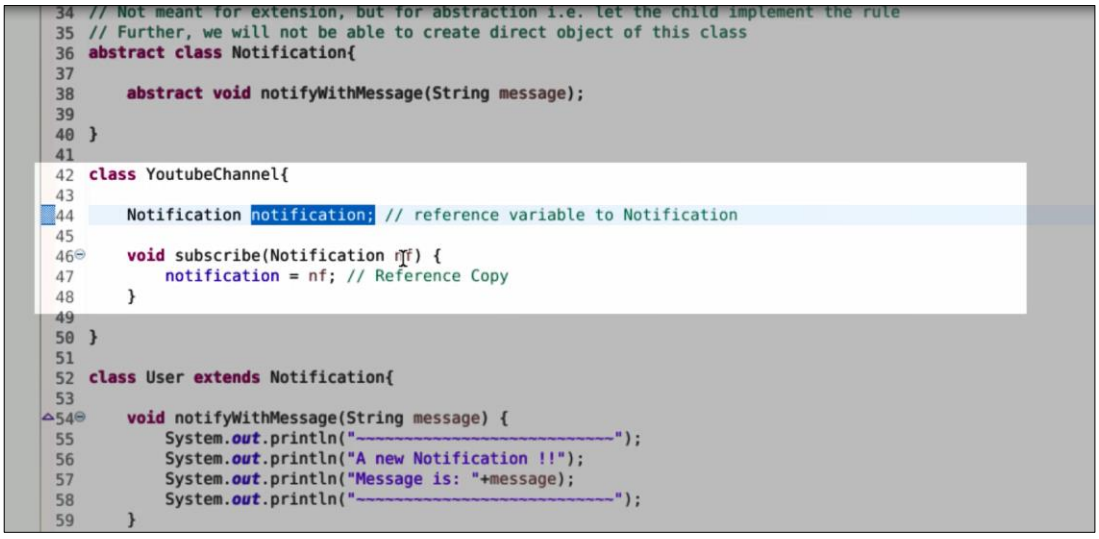
5.10 Let us create this abstract method by writing abstract **void notify(String message);.** However, the User class is getting an error. Therefore, in the User class, you need to define this method called notify. When you define this method and it is executed, you will create a new notification and then print out **"Message is:" + message** to make the presentation a bit more beautiful



5.11 If the User class is implementing the notify method with a message, then it means that you are extending the Notification class. This abstract class is meant for abstraction, allowing the child class to implement the rule. Therefore, we will not be able to create a direct object for this class



5.12 Let us now move further and understand the next part where the YouTube channel user has to subscribe. For this, you can create a reference variable of type Notification, which can be written as **Notification notification;** Create a method called **subscribe** and take this notification as input. You can write it as **void subscribe(Notification nf)**. You can then assign this notification to **nf**, which is a reference copy



5.13 In the main method, let us create a **YouTubeChannel** object. Write **YouTubeChannel channel = new YouTubeChannel();**. For the user, write **User John = new User();**. This creates a new User object named John and a **YouTubeChannel** object. On the **YouTubeChannel** object, the subscription is for John. Here, you need to understand the polymorphic behavior: the reference variable John holds the hash code of the User object

```
59      }
60
61  }
62
63
64  public class AbstractDemo {
65
66      public static void main(String[] args) {
67
68          Cab cab; // cab is not an object. cab is a reference variable
69          //cab = new Cab(); // with new an object will be created in memory
70
71          // Polymophic Statements will work in same way
72          cab = new MiniCab();
73
74          //MiniCab miniCab = new MiniCab();
75          //miniCab.bookCab("Home", "Work");
76
77          cab.bookCab("Home", "Work");
78
79          User john = new User();
80
81          YoutubeChannel channel = new YoutubeChannel();
82          channel.subscribe(john)  User john - AbstractDemo.main(String[])
83
84      }
```

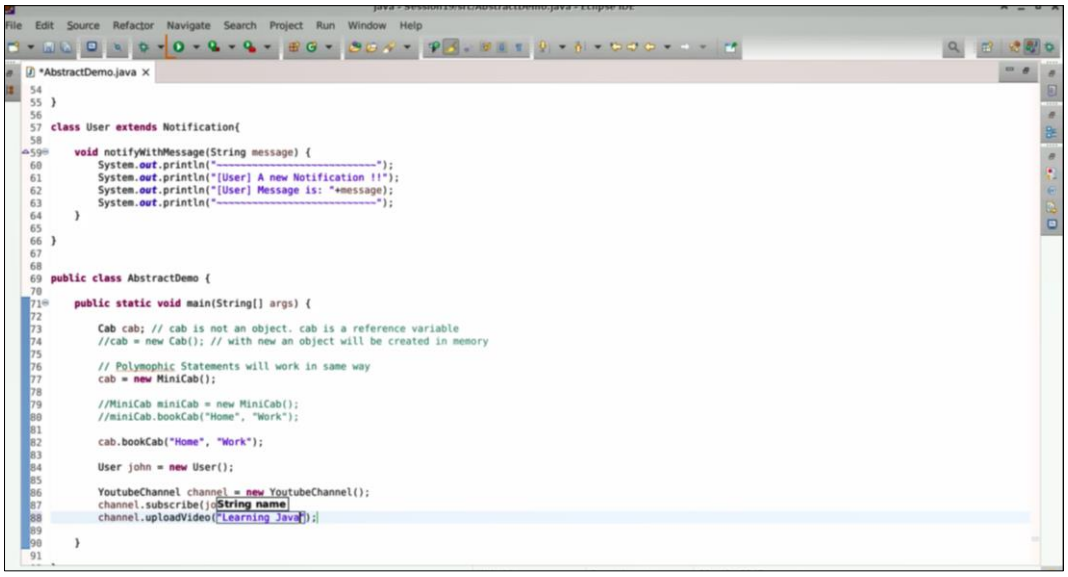
5.14 In the **YouTubeChannel** class, you can create a method called **uploadVideo**. This method will take a video name as input and create a message that says **'A new video uploaded: '** followed by the name of the video. Using the notification reference variable, you will execute the notify method with this message

File Edit Source Refactor Navigate Search Project Run Window Help

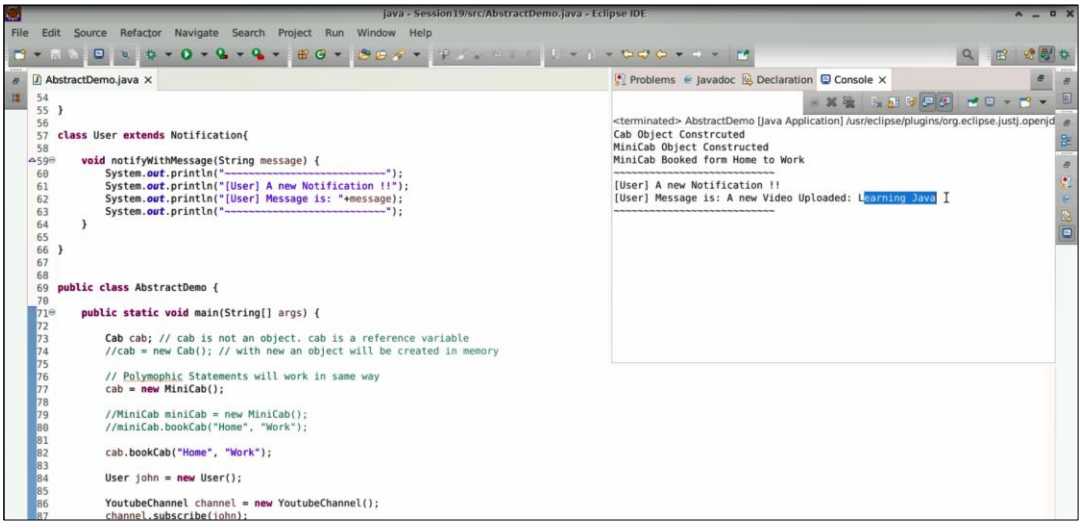
AbstractDemo.java

```
41
42  class YoutubeChannel{
43
44      Notification notification; // reference variable to Notification
45
46      void subscribe(Notification nf) { // Polymorphic Statement: nf has reference to the User Object
47          notification = nf; // Reference Copy -> notification now has reference to john i.e. the User Object
48      }
49
50      void uploadVideo(String name) {
51          String message = "A new Video Uploaded: "+name;
52          notification.notifyWithMessage(message);
53      }
54
55  }
56
57  class User extends Notification{
58
59      void notifyWithMessage(String message) {
60          System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
61          System.out.println("A new Notification !!");
62          System.out.println("Message is: "+message);
63          System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
```

5.15 All this content is generated for the user when a new video gets uploaded to the YouTube channel. If you call **channel.uploadVideo("Learning Java")**, it simulates uploading a video named 'Learning Java' to the channel



5.16 When you run this code, you will notice that you receive a new notification message: **'A new video uploaded named Learning Java'**. What is abstraction? Abstraction helps you hide complexity and provides a clear scenario where you can implement a beautiful concept using runtime polymorphism. What you have implemented here is a callback. This means that the notify method with the message is executed in the User object.



By following the above steps, you have successfully implemented and used an abstract class in Java.