

Lesson 04 Demo 11

Implementing Callable and Future

Objective: To demonstrate the usage of callable interface and futures

Tools required: Eclipse IDE

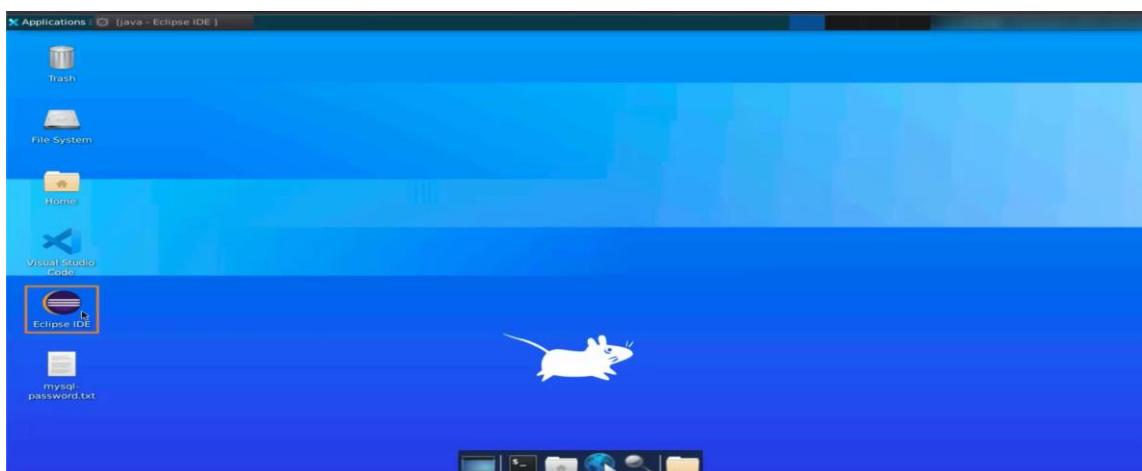
Prerequisites: None

Steps to be followed:

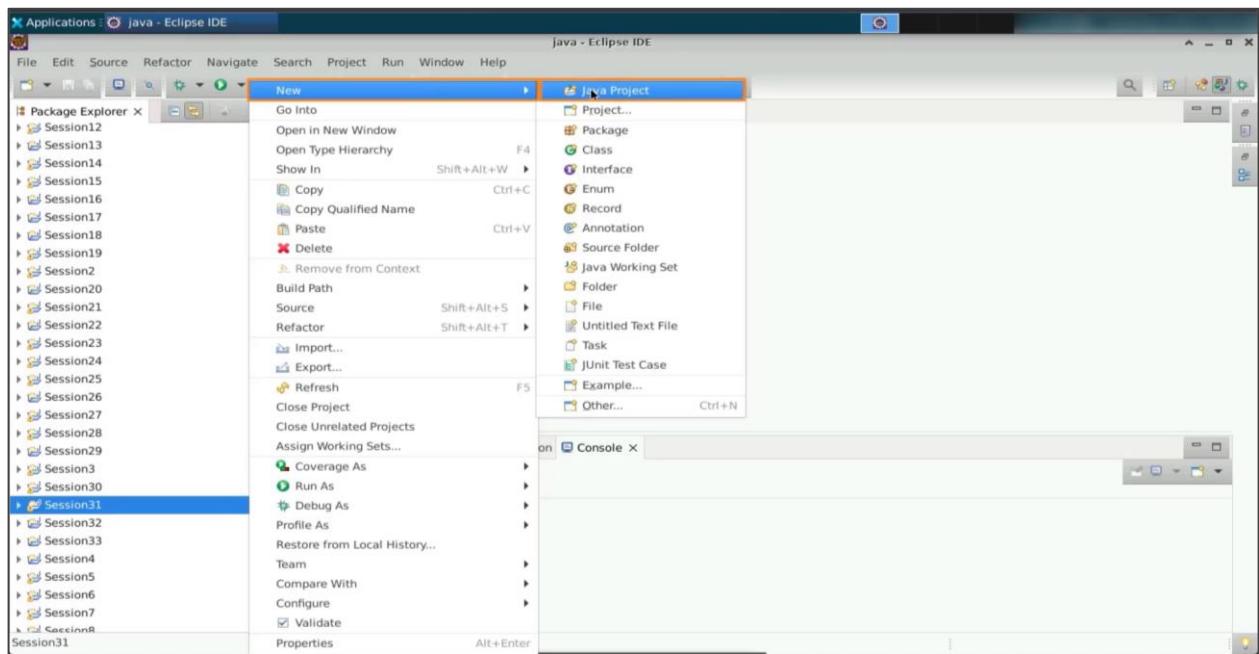
1. Implement a thread using Callable and Future Interfaces along with the suitable scenarios

Step 1: Implement a thread using Callable and Future Interfaces along with suitable scenarios

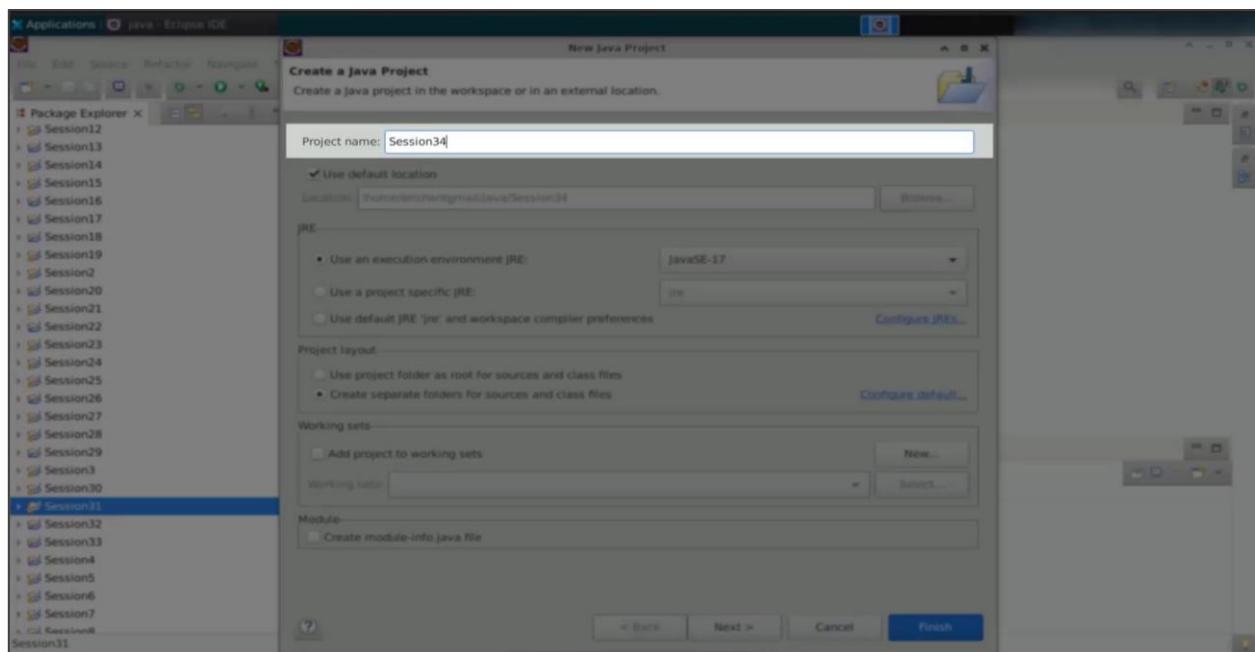
1.1 If you wish to create a thread, you can either extend the `Thread` class or implement the `Runnable` interface. When you start the thread, it runs asynchronously, and you will not be notified of its termination. However, if you need to capture results from a thread, Java provides the `Callable` interface, which allows your thread to run asynchronously and return a result in the future. Let us get started by opening our Eclipse IDE.



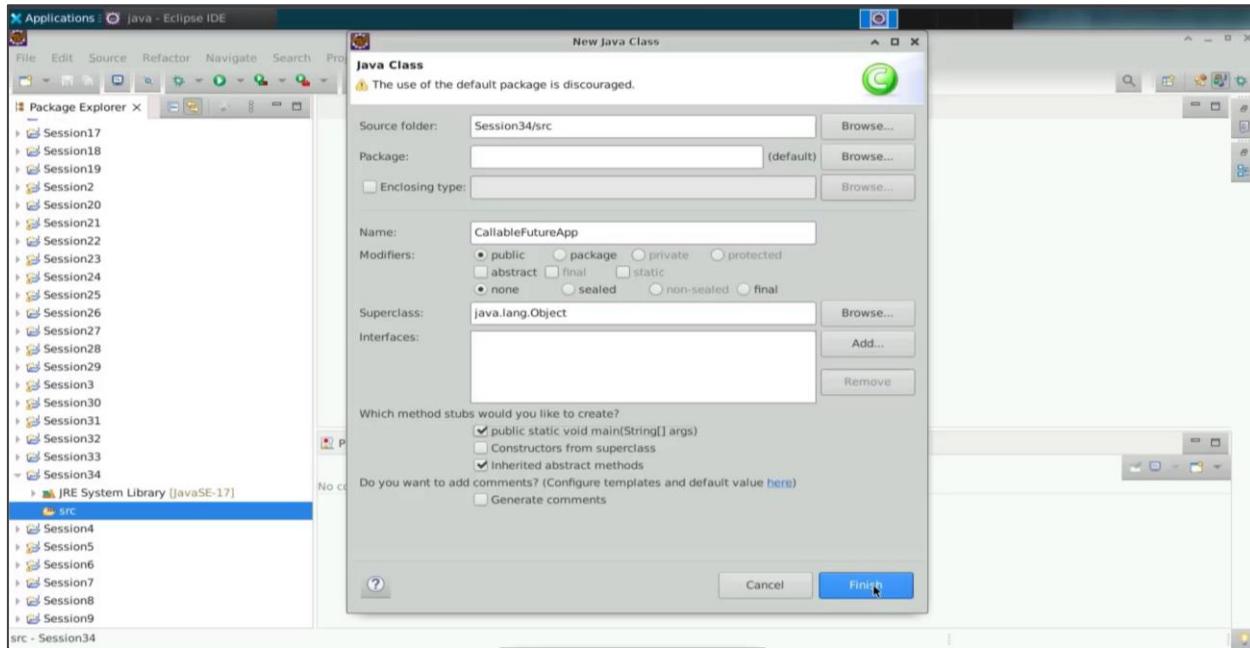
1.2 In the package Explorer, create a new Java project by selecting **New > Java Project**.



1.3 Name the project **Session34**, uncheck **Create a module info.java file**, and press **Finish**.



- 1.4 In the source, do a right click and create a new class and name the class as **CallableFutureApp** along with the main method. Click **Finish**.



Note: For writing this demonstration, you will learn about the executor service in Java, whose key role is to execute the tasks on the threads asynchronously. When you will create your threads with callable, then the executable service will submit the task with the help of which you can get some results in the future. Let us now represent a use case for movie tickets.

```

1 Applications : Java - Session34/src/Cal...
2 File Edit Source Refactor Navigate Search Project Run Window Help
3 Package Explorer X CallableFutureApp.java X
4 Session18 Session19 Session2 Session20 Session21 Session22 Session23 Session24 Session25 Session26 Session27 Session28 Session29 Session30 Session31 Session32 Session33 Session34 JRE System Library [javaSE-17] src (default package) CallableFutureApp.java
5 Session4 Session5 Session6 Session7 Session8
6 CallableFutureApp.java
7
8 public class CallableFutureApp {
9
10    public static void main(String[] args) {
11        System.out.println("Hello, Future!");
12    }
13}
14
15 public class CallableFutureApp {
16    public static void main(String[] args) {
17        System.out.println("Hello, Future!");
18    }
19}
20
21
22

```

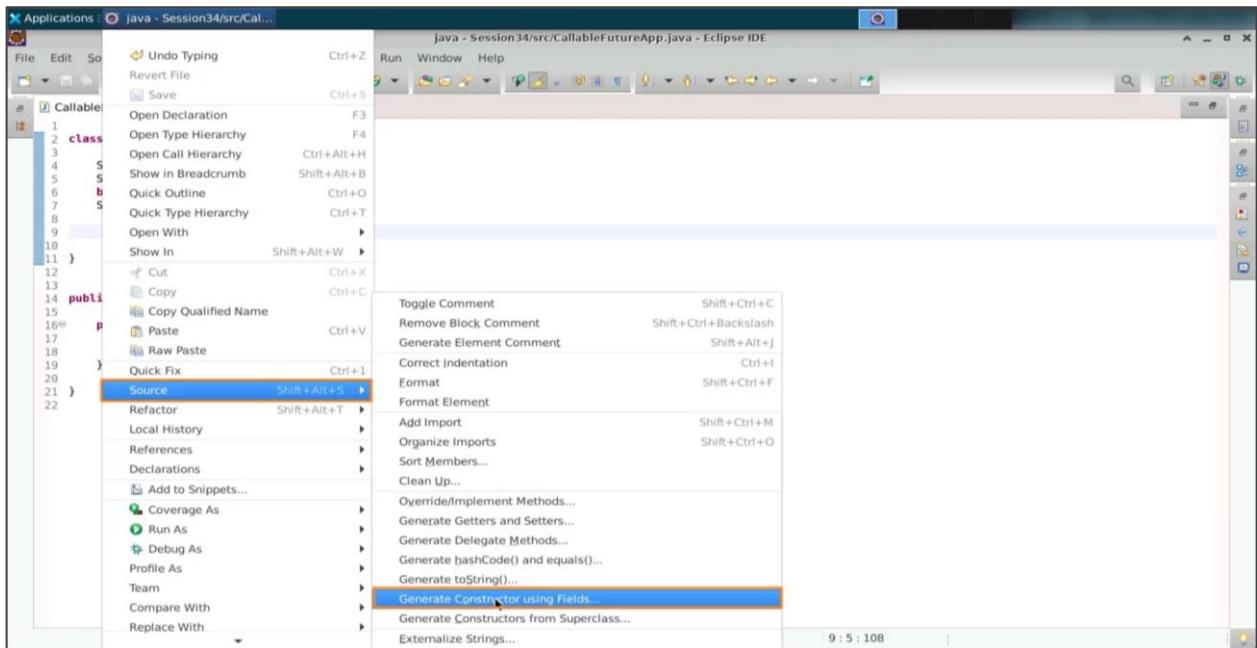
- 1.5 Create a class called Movie Ticket. And for movie tickets, let us give a movie name. Then you can give the seat number. Next, is the ticket booked, or is a ticket available? Let us keep a status known as Is the ticket booked? Now, let us record who booked it, and the email for the user.

```

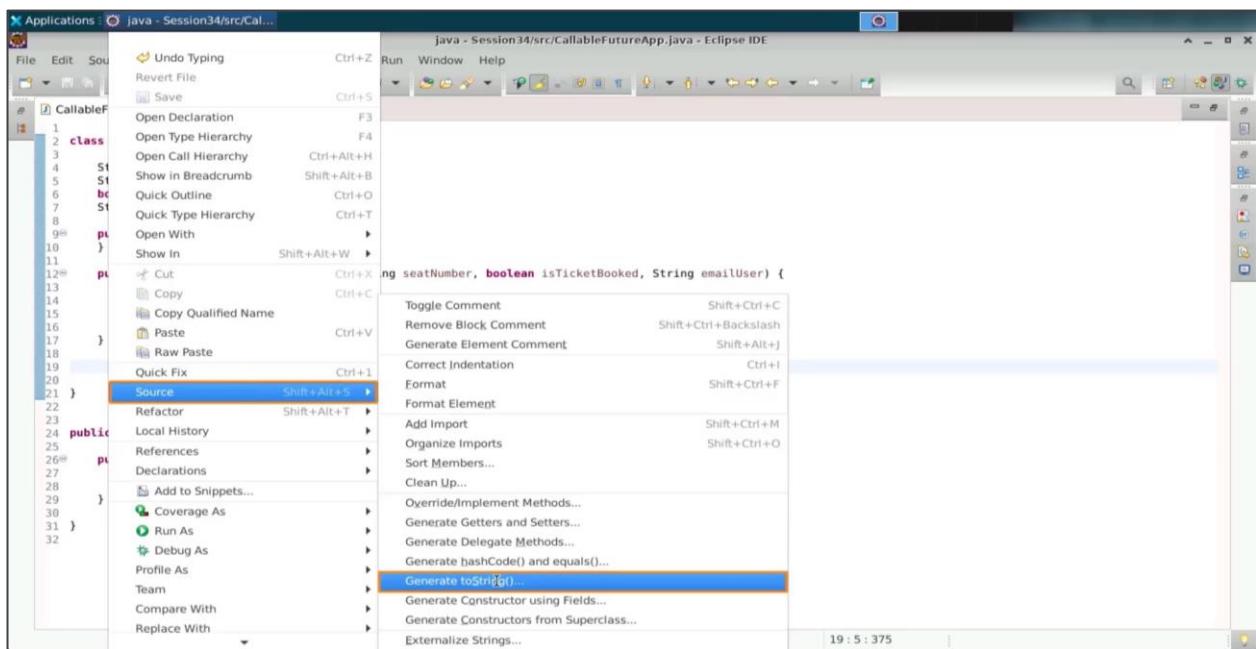
1 Applications : Java - Session34/src/Cal...
2 File Edit Source Refactor Navigate Search Project Run Window Help
3 Package Explorer X CallableFutureApp.java X
4 CallableFutureApp.java
5
6 class MovieTicket {
7     String movieName;
8     String seatNumber;
9     boolean isTicketBooked;
10    String emailUser;
11
12    I
13}
14
15 public class CallableFutureApp {
16    public static void main(String[] args) {
17        System.out.println("Hello, Future!");
18    }
19}
20
21
22

```

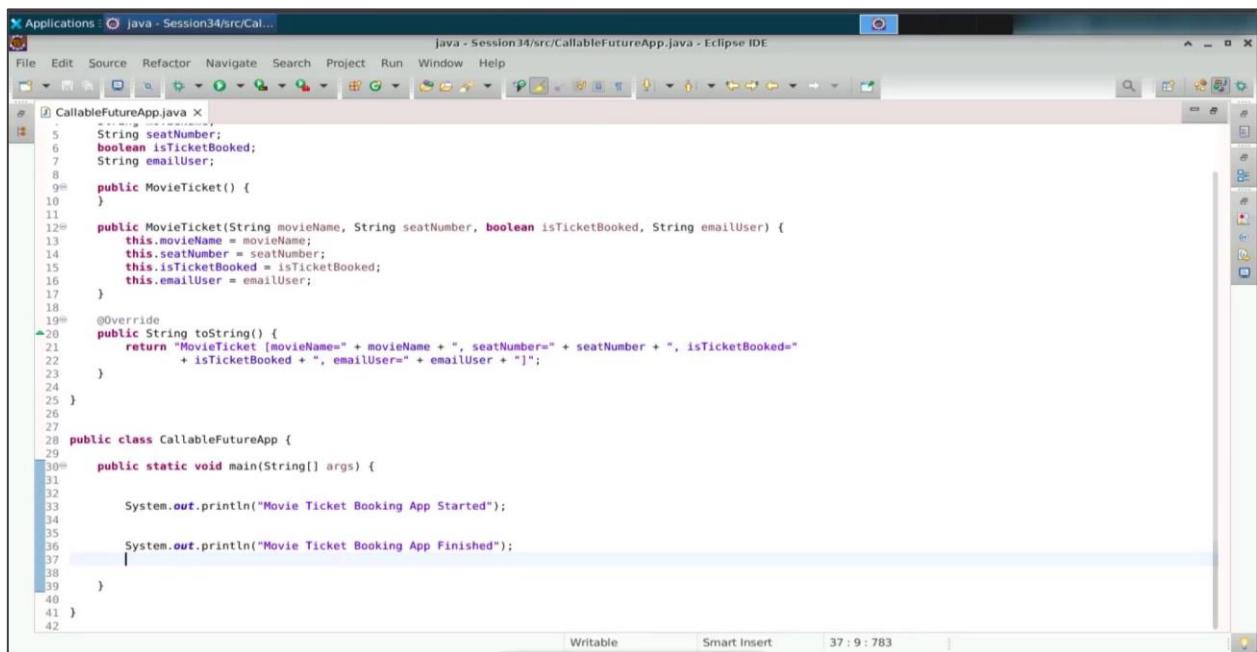
- 1.6 Now, right-click in the code console window and select **Source > Generate Constructor using Fields**. Then, click Finish.



- 1.7 Let's remove this super part and write a default constructor, which will be used to create a regular movie ticket object. To see the details in the ticket, you can generate a **toString** method through which you will know what the data inside this movie ticket is.



1.8 In the main method, consider there is a movie booking app, where you are supposed to book the movie tickets. Let us give a print statement as **Movie Ticket Booking App Started**. The last statement will be entered as **Movie Ticket Booking App Finished**. This is with the application logs coming in.

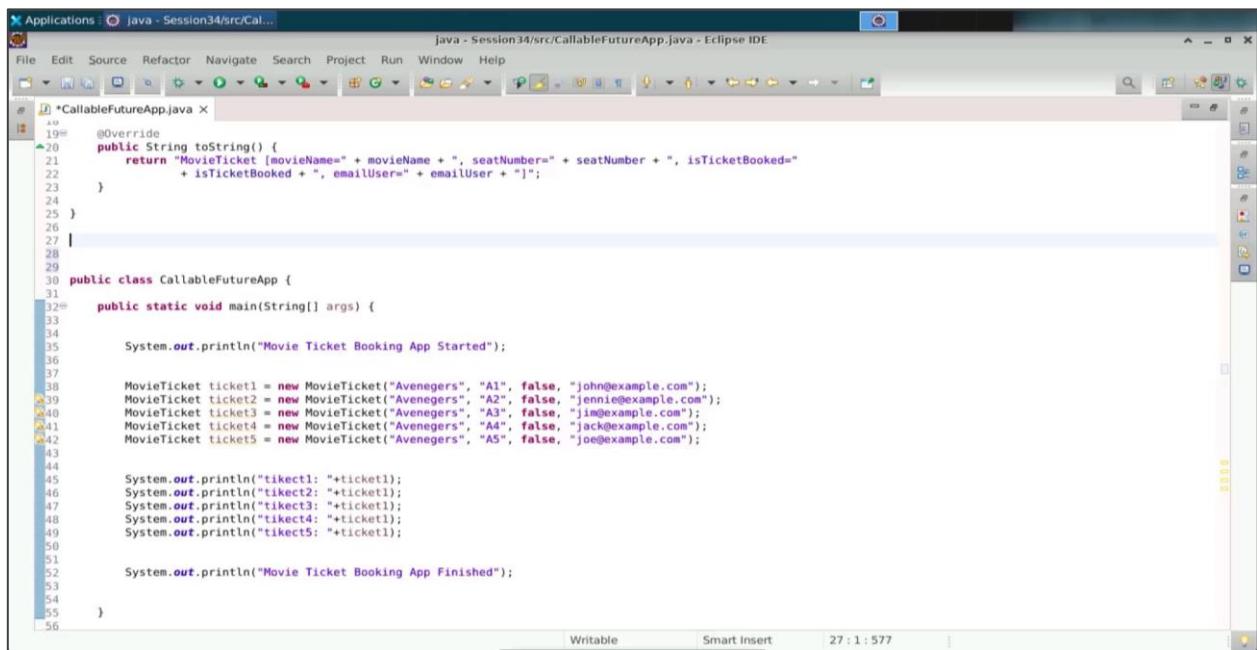


The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Applications - java - Session34/src/CallableFutureApp.java - Eclipse IDE
- File Menu:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Standard toolbar icons for file operations, search, and project management.
- Code Editor:** The main window displays the Java code for `CallableFutureApp.java`. The code defines a `MovieTicket` class with a constructor and a `toString` method, and a `CallableFutureApp` class with a `main` method. The `main` method contains two `System.out.println` statements: one for "Movie Ticket Booking App Started" and one for "Movie Ticket Booking App Finished".
- Status Bar:** Shows "Writable", "Smart Insert", and "37 : 9 : 783".

```
5  String seatNumber;
6  boolean isTicketBooked;
7  String emailUser;
8
9  public MovieTicket() {
10 }
11
12  public MovieTicket(String movieName, String seatNumber, boolean isTicketBooked, String emailUser) {
13      this.movieName = movieName;
14      this.seatNumber = seatNumber;
15      this.isTicketBooked = isTicketBooked;
16      this.emailUser = emailUser;
17  }
18
19  @Override
20  public String toString() {
21      return "MovieTicket [movieName=" + movieName + ", seatNumber=" + seatNumber + ", isTicketBooked=" +
22          + isTicketBooked + ", emailUser=" + emailUser + "]";
23  }
24
25 }
26
27
28 public class CallableFutureApp {
29
30     public static void main(String[] args) {
31
32         System.out.println("Movie Ticket Booking App Started");
33
34
35         System.out.println("Movie Ticket Booking App Finished");
36
37     }
38
39 }
40
41 }
42 }
```

1.9 Create five movie tickets for "Avengers" with seat numbers A1 to A5 and set "Is the Ticket Booked" to false. Users: John, Jenni, Jim, Jack, and Joe (e.g., john@example.com). Print the tickets to see details. Use `Callable` for the ticket booking thread to return a response when the thread terminates.



The screenshot shows the Eclipse IDE interface with a Java file named `CallableFutureApp.java` open in the editor. The code defines a `MovieTicket` class and a `CallableFutureApp` application. The `MovieTicket` class has a constructor taking movie name, seat number, user email, and a boolean for booking status. The `CallableFutureApp` class prints out five instances of `MovieTicket` with seat numbers A1 to A5 and booking status set to false for users John, Jenni, Jim, Jack, and Joe respectively.

```
File Edit Source Refactor Navigate Search Project Run Window Help
# CallableFutureApp.java
19  *Override
20  public String toString() {
21      return "MovieTicket [movieName=" + movieName + ", seatNumber=" + seatNumber + ", isTicketBooked=" +
22             + isTicketBooked + ", emailUser=" + emailUser + "]";
23  }
24
25
26
27
28
29
30 public class CallableFutureApp {
31
32     public static void main(String[] args) {
33
34         System.out.println("Movie Ticket Booking App Started");
35
36
37         MovieTicket ticket1 = new MovieTicket("Avenegers", "A1", false, "john@example.com");
38         MovieTicket ticket2 = new MovieTicket("Avenegers", "A2", false, "jennie@example.com");
39         MovieTicket ticket3 = new MovieTicket("Avenegers", "A3", false, "jim@example.com");
40         MovieTicket ticket4 = new MovieTicket("Avenegers", "A4", false, "jack@example.com");
41         MovieTicket ticket5 = new MovieTicket("Avenegers", "A5", false, "joe@example.com");
42
43
44
45         System.out.println("ticket1: "+ticket1);
46         System.out.println("ticket2: "+ticket1);
47         System.out.println("ticket3: "+ticket1);
48         System.out.println("ticket4: "+ticket1);
49         System.out.println("ticket5: "+ticket1);
50
51
52         System.out.println("Movie Ticket Booking App Finished");
53
54
55     }
56 }
```

- 1.10 Create a class `MovieTicketBookingTask` that implements `Callable` from `java.util.concurrent`, which returns a value. Define the thread's return type as `String`. Use a reference variable for the ticket and create a constructor that takes a `MovieTicket` as input. Assign the ticket to the reference with `this.ticket = ticket`. Pass the ticket to be booked as a reference.

The screenshot shows the Eclipse IDE interface with the title bar "java - Session 34/src/CallableFutureApp.java - Eclipse IDE". The left pane displays the Java code for `CallableFutureApp.java`, which includes a `MovieTicket` class, a `MovieTicketBookingTask` task class, and a `CallableFutureApp` main class. The right pane shows the Java code for `CallableFutureApp.java`.

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

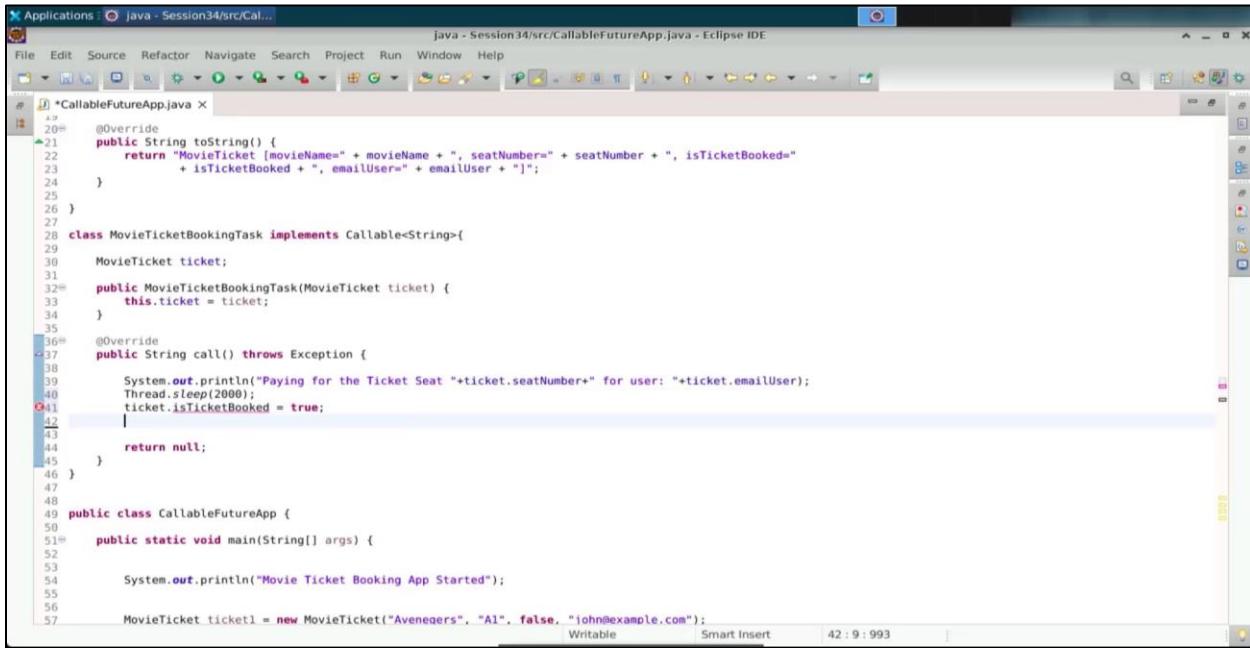
a CallableFutureApp.java X
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57

public String toString() {
    return "MovieTicket [movieName=" + movieName + ", seatNumber=" + seatNumber + ", isTicketBooked=" +
           + isTicketBooked + ", emailUser=" + emailUser + "]";
}

class MovieTicketBookingTask implements Callable<String>{
    MovieTicket ticket;
    public MovieTicketBookingTask(MovieTicket ticket) {
        this.ticket = ticket;
    }
}

public class CallableFutureApp {
    public static void main(String[] args) {
        System.out.println("Movie Ticket Booking App Started");
        MovieTicket ticket1 = new MovieTicket("Avengers", "A1", false, "john@example.com");
        MovieTicket ticket2 = new MovieTicket("Avengers", "A2", false, "jennie@example.com");
        MovieTicket ticket3 = new MovieTicket("Avengers", "A3", false, "jim@example.com");
        MovieTicket ticket4 = new MovieTicket("Avengers", "A4", false, "jack@example.com");
        MovieTicket ticket5 = new MovieTicket("Avengers", "A5", false, "joe@example.com");
        System.out.println("ticket1: "+ticket1);
        System.out.println("ticket2: "+ticket2);
        System.out.println("ticket3: "+ticket3);
        System.out.println("ticket4: "+ticket4);
    }
}
```

1.11 Override the `call` method, which will throw an exception if anything goes wrong. The `call` method's return type is `String`. The user will make the payment, so let us write "paying for the ticket seat" and include the ticket seat number and the ticket's email. Since this operation can take some time, introduce a dummy thread sleep of 2000 milliseconds, which can also throw an exception. Assume the payment transaction takes 2 seconds, then set `ticket.isTicketBooked` to true.



The screenshot shows the Eclipse IDE interface with the file `CallableFutureApp.java` open. The code implements a `Callable` interface to handle movie ticket booking. It includes a `toString` method, a `MovieTicketBookingTask` inner class, and a `CallableFutureApp` main class. The `call` method performs a sleep operation and then sets the ticket's `isTicketBooked` flag to `true`.

```
File Edit Source Refactor Navigate Search Project Run Window Help
File Edit Source Refactor Navigate Search Project Run Window Help
@Override
public String toString() {
    return "MovieTicket [movieName=" + movieName + ", seatNumber=" + seatNumber + ", isTicketBooked="
        + isTicketBooked + ", emailUser=" + emailUser + "]";
}
}

class MovieTicketBookingTask implements Callable<String>{
MovieTicket ticket;
public MovieTicketBookingTask(MovieTicket ticket) {
    this.ticket = ticket;
}
@Override
public String call() throws Exception {
    System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser);
    Thread.sleep(2000);
    ticket.isTicketBooked = true;
    |
    return null;
}
}

public class CallableFutureApp {
public static void main(String[] args) {
    System.out.println("Movie Ticket Booking App Started");
    MovieTicket ticket1 = new MovieTicket("Avengers", "A1", false, "john@example.com");
}
}
```

- 1.12 Next, you can give as print, ticket booked for plus ticket.the email user. The email of the user is the identifier for the ticket. If you wish, you can even allocate the seat numbers here. Consider, this is by default nothing, and no seat number is allocated.

```

    14     this.movieName = movieName;
    15     this.seatNumber = seatNumber;
    16     this.isTicketBooked = isTicketBooked;
    17     this.emailUser = emailUser;
    18   }
    19
    20  @Override
    21  public String toString() {
    22     return "MovieTicket [movieName=" + movieName + ", seatNumber=" + seatNumber + ", isTicketBooked=" +
    23           + isTicketBooked + ", emailUser=" + emailUser + "]";
    24   }
    25
    26 }
    27
    28 class MovieTicketBookingTask implements Callable<String>{
    29
    30   MovieTicket ticket;
    31
    32   public MovieTicketBookingTask(MovieTicket ticket) {
    33     this.ticket = ticket;
    34   }
    35
    36   @Override
    37   public String call() throws Exception {
    38
    39     System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser);
    40     Thread.sleep(2000);
    41     ticket.isTicketBooked = true;
    42     System.out.println("Ticket Booked for "+ticket.emailUser);
    43
    44     return "";
    45   }
    46 }
    47
    48
    49 public class CallableFutureApp {
    50
    51   public static void main(String[] args) {
    52
    53
    54
    55
    56
    57
    58     System.out.println("Movie Ticket Booking App Started");
    59
    60
    61     MovieTicket ticket1 = new MovieTicket("Avengers", "", false, "john@example.com");
    62     MovieTicket ticket2 = new MovieTicket("Avengers", "", false, "jennie@example.com");
    63     MovieTicket ticket3 = new MovieTicket("Avengers", "", false, "jim@example.com");
    64     MovieTicket ticket4 = new MovieTicket("Avengers", "", false, "jack@example.com");
    65     MovieTicket ticket5 = new MovieTicket("Avengers", "", false, "joe@example.com");
    66
  
```

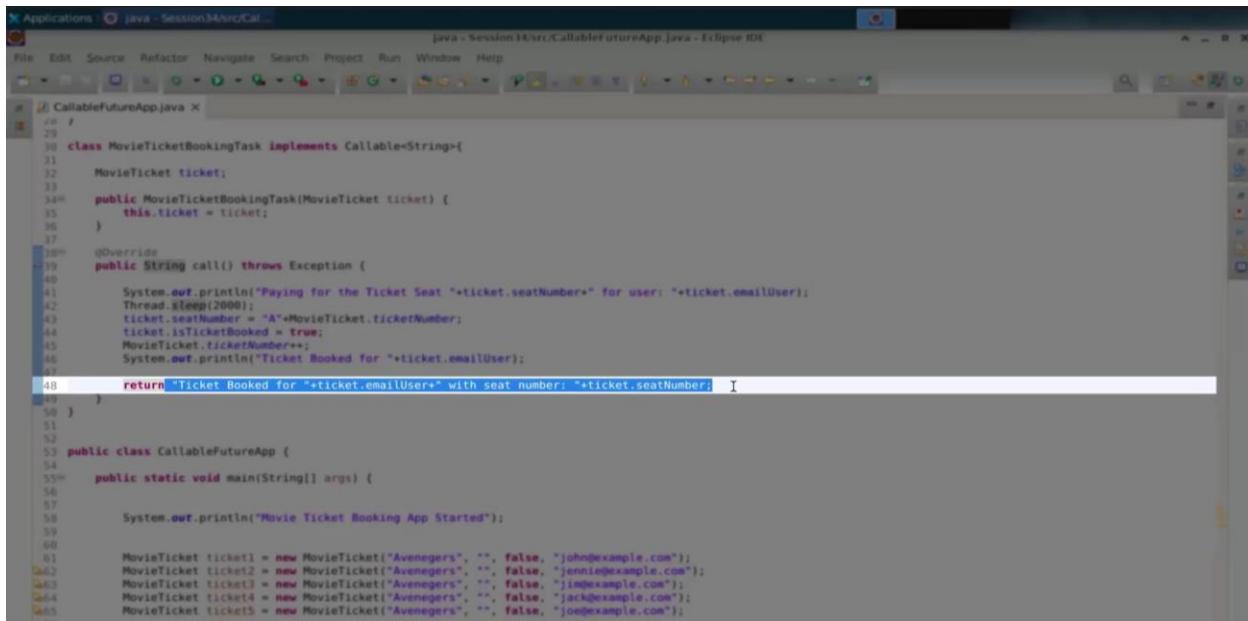
Syntax error, insert ";" to complete ReturnStatement

- 1.13 You have one of the static variables called ticket number, which begins with one integer type. This ticket serial number starts from 1 and before you book the ticket, you will write as ticket.seatnumber goes as the A row + from the movie ticket.ticketNumber.

```

    14     this.movieName = movieName;
    15     this.seatNumber = seatNumber;
    16     this.isTicketBooked = isTicketBooked;
    17     this.emailUser = emailUser;
    18   }
    19
    20  @Override
    21  public String toString() {
    22     return "MovieTicket [movieName=" + movieName + ", seatNumber=" + seatNumber + ", isTicketBooked=" +
    23           + isTicketBooked + ", emailUser=" + emailUser + "]";
    24   }
    25
    26 }
    27
    28 class MovieTicketBookingTask implements Callable<String>{
    29
    30   MovieTicket ticket;
    31
    32   public MovieTicketBookingTask(MovieTicket ticket) {
    33     this.ticket = ticket;
    34   }
    35
    36   @Override
    37   public String call() throws Exception {
    38
    39     System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser);
    40     Thread.sleep(2000);
    41     ticket.seatNumber = "A"+MovieTicket.ticketNumber;
    42     ticket.isTicketBooked = true;
    43
    44     System.out.println("Ticket Booked for "+ticket.emailUser);
    45
    46     return "";
    47   }
    48 }
    49
    50
    51
    52
    53 public class CallableFutureApp {
    54
    55   public static void main(String[] args) {
    56
    57
    58     System.out.println("Movie Ticket Booking App Started");
    59
    60
    61     MovieTicket ticket1 = new MovieTicket("Avengers", "", false, "john@example.com");
    62     MovieTicket ticket2 = new MovieTicket("Avengers", "", false, "jennie@example.com");
    63     MovieTicket ticket3 = new MovieTicket("Avengers", "", false, "jim@example.com");
    64     MovieTicket ticket4 = new MovieTicket("Avengers", "", false, "jack@example.com");
    65     MovieTicket ticket5 = new MovieTicket("Avengers", "", false, "joe@example.com");
    66
  
```

- 1.14 Increment `movieTicket.ticketNumber` by 1 to allocate a seat number. Return the ticket booked for the user's email with the seat number. This completes the call implementation. In the `run` method of `Runnable`, which returns void, you cannot return data. However, with the `Callable` interface, you can create threads that return data of the expected type.



The screenshot shows the Eclipse IDE interface with the file 'CallableFutureApp.java' open in the editor. The code implements a Callable interface to handle movie ticket bookings.

```
1  // Applications  Java - Session 34/src/CallableFutureApp.java - Eclipse IDE
2
3  File Edit Source Refactor Navigate Search Project Run Window Help
4
5  CallableFutureApp.java X
6
7  class MovieTicketBookingTask implements Callable<String>{
8
9     MovieTicket ticket;
10
11    public MovieTicketBookingTask(MovieTicket ticket) {
12        this.ticket = ticket;
13    }
14
15    @Override
16    public String call() throws Exception {
17
18        System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser);
19        Thread.sleep(2000);
20        ticket.seatNumber = "A"+MovieTicket.ticketNumber;
21        ticket.isTicketBooked = true;
22        MovieTicket.ticketNumber++;
23        System.out.println("Ticket Booked for "+ticket.emailUser);
24
25        return "Ticket Booked for "+ticket.emailUser+ " with seat number: "+ticket.seatNumber; I
26    }
27
28 }
29
30
31
32
33 public class CallableFutureApp {
34
35     public static void main(String[] args) {
36
37         System.out.println("Movie Ticket Booking App Started");
38
39         MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
40         MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
41         MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
42         MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
43         MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");
44
45     }
46 }
```

- 1.15 Let's navigate to the main method to check the movie ticket booking execution. Print the available processors using the runtime; there are four, indicating a quad-core system. Create a thread pool using two cores to run threads. To reduce CPU load, use Java's executor service, which runs threads in a pool. The creation and usage of the executor service will be covered next.

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

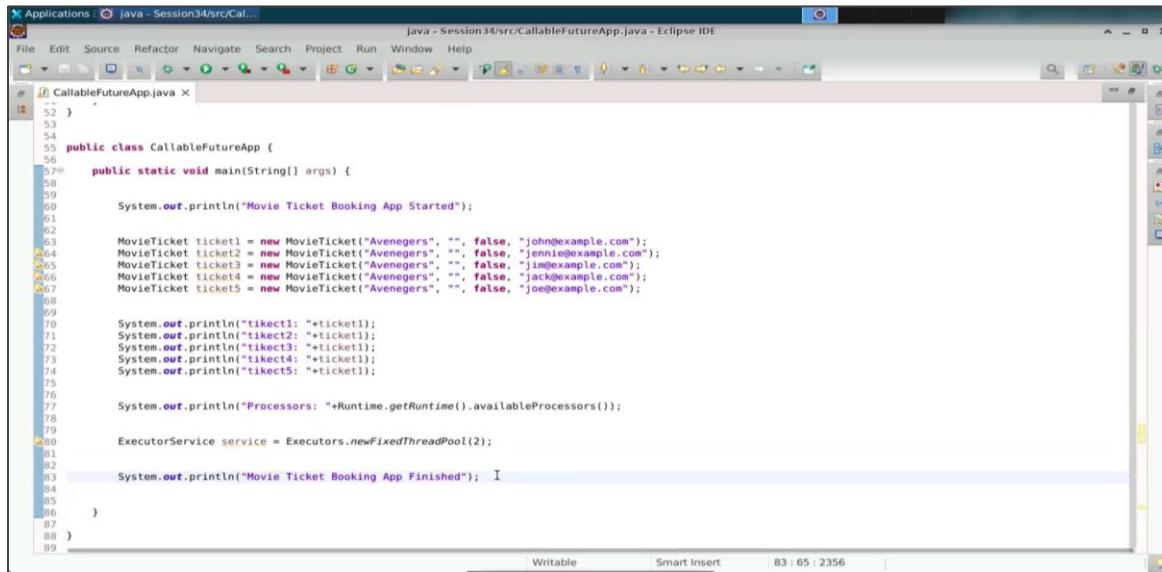
- Title Bar:** Applications > java - Session34/src/CallableFutureApp...
- File Menu:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Toolbar:** Standard Eclipse toolbar icons.
- Left Panel (Editor):** Shows the source code for `CallableFutureApp.java`. The code creates five movie tickets and prints them out. It also prints the number of processors available and the final message.

```
44     ticket.isTicketBooked = true;
45     MovieTicket.ticketNumber++;
46     System.out.println("Ticket Booked for "+ticket.emailUser);
47
48     return "Ticket Booked for "+ticket.emailUser+" with seat number: "+ticket.seatNumber
49   }
50 }
51
52 public class CallableFutureApp {
53
54   public static void main(String[] args) {
55
56
57     System.out.println("Movie Ticket Booking App Started");
58
59
60     MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
61     MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "annie@example.com");
62     MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
63     MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
64     MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");
65
66
67     System.out.println("tikect1: "+ticket1);
68     System.out.println("tikect2: "+ticket1);
69     System.out.println("tikect3: "+ticket1);
70     System.out.println("tikect4: "+ticket1);
71     System.out.println("tikect5: "+ticket1);
72
73
74     System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
75
76
77     System.out.println("Movie Ticket Booking App Finished");
78
79
80   }
81 }
```

- Right Panel (Console):** Displays the standard output of the application. It shows the movie ticket details, the processor count, and the final message.

```
<terminated> CallableFutureApp [java Application] /usr/eclipse/plugins/org.eclipse.just.jop
Movie Ticket Booking App Started
tikect1: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
tikect2: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
tikect3: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
tikect4: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
tikect5: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
Processor: 4
Movie Ticket Booking App Finished
```

1.16 First, select `ExecutorService` from the `java.util.concurrent` package and create it using the `Executors` class. The `Executors` class provides factory methods for creating thread pools, such as `newFixedThreadPool`, which reuses a fixed number of threads. For example, a thread pool with two threads can be created. This helps manage tasks efficiently by queuing tasks until threads are free to execute them, ensuring idle threads pick up new tasks.

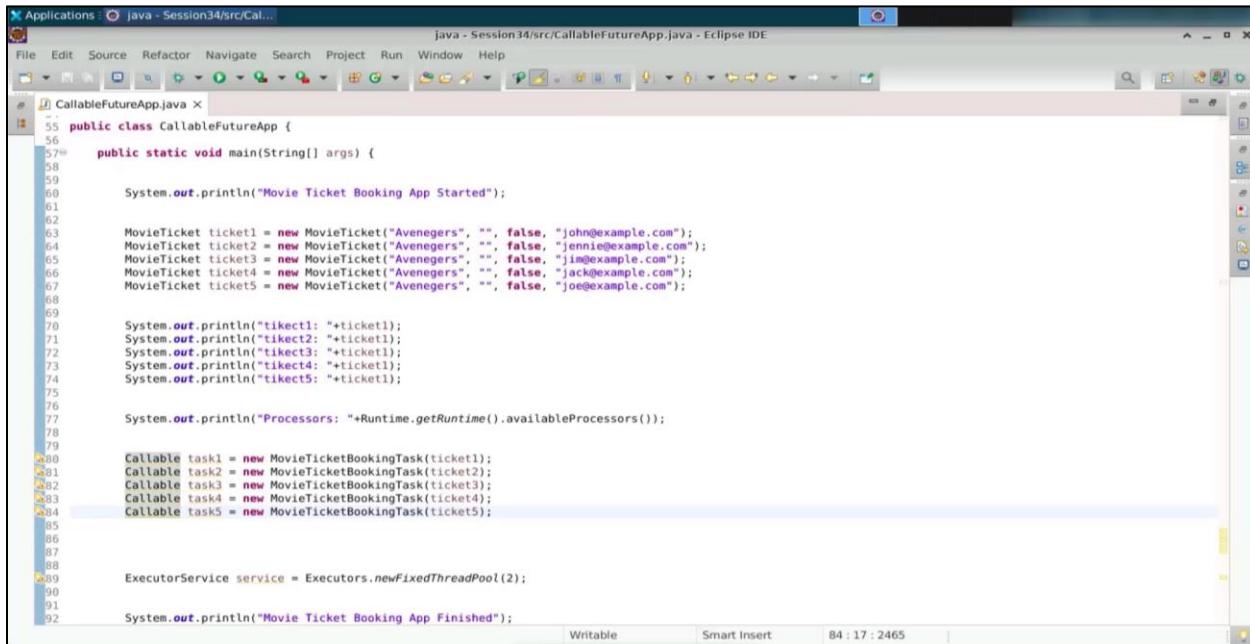


The screenshot shows the Eclipse IDE interface with a Java file named "CallableFutureApp.java" open. The code creates five movie tickets and prints their details. It then creates an ExecutorService with 2 threads and prints the available processors. Finally, it prints a message indicating the app is finished.

```
java - Session34/src/Call... CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Project Run Window Help
 CallableFutureApp.java ×
52 }
53
54 public class CallableFutureApp {
55
56     public static void main(String[] args) {
57
58         System.out.println("Movie Ticket Booking App Started");
59
60         MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
61         MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
62         MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "im@example.com");
63         MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
64         MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");
65
66         System.out.println("ticket1: "+ticket1);
67         System.out.println("ticket2: "+ticket1);
68         System.out.println("ticket3: "+ticket1);
69         System.out.println("ticket4: "+ticket1);
70         System.out.println("ticket5: "+ticket1);
71
72         System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
73
74         ExecutorService service = Executors.newFixedThreadPool(2);
75
76         System.out.println("Movie Ticket Booking App Finished");
77
78     }
79
80 }
```

1.17 To submit tasks to the executor service, start by creating objects for movie ticket booking tasks.

For example, task one will book ticket one, and similarly, create five tasks for five different tickets. Use a polymorphic statement with the Callable interface to create these tasks. First, create these five tasks as Callable objects, each responsible for booking tickets from ticket 1 to ticket 5.



The screenshot shows the Eclipse IDE interface with the file 'CallableFutureApp.java' open. The code implements a main method that prints a welcome message, creates five MovieTicket objects, and then creates five Callable objects (task1 through task5) using a polymorphic statement. It then prints the number of processors available and submits the tasks to a fixed thread pool of size 2. Finally, it prints a completion message. The code is as follows:

```
java - Session 34/src/CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
CallableFutureApp.java X
55 public class CallableFutureApp {
56     public static void main(String[] args) {
57         System.out.println("Movie Ticket Booking App Started");
58
59         MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
60         MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
61         MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
62         MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
63         MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");
64
65         System.out.println("ticket1: "+ticket1);
66         System.out.println("ticket2: "+ticket2);
67         System.out.println("ticket3: "+ticket3);
68         System.out.println("ticket4: "+ticket4);
69         System.out.println("ticket5: "+ticket5);
70
71         System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
72
73         Callable task1 = new MovieTicketBookingTask(ticket1);
74         Callable task2 = new MovieTicketBookingTask(ticket2);
75         Callable task3 = new MovieTicketBookingTask(ticket3);
76         Callable task4 = new MovieTicketBookingTask(ticket4);
77         Callable task5 = new MovieTicketBookingTask(ticket5);
78
79
80         ExecutorService service = Executors.newFixedThreadPool(2);
81
82         System.out.println("Movie Ticket Booking App Finished");
83
84
85
86
87
88
89
90
91
92
```

- 1.18 In the executor service created below, use `service.submit` to create a pool of tasks. This method submits a callable and returns a `Future` object, which is of type `String` here. `Future`, imported from `java.util`, represents the result of your asynchronous computation, checking if it is complete and acting as a blocking operation. Thus, `service.submit(task1)` blocks further execution until the computation is complete, putting other statements on hold.

```

Applications: Java - Session34/src/Call...
File Edit Source Refactor Navigate Search Project Run Window Help
CallableFutureApp.java X
public class CallableFutureApp {
    public static void main(String[] args) {
        System.out.println("Movie Ticket Booking App Started");
        MovieTicket ticket1 = new MovieTicket("Avengers", "", false, "john@example.com");
        MovieTicket ticket2 = new MovieTicket("Avengers", "", false, "jennie@example.com");
        MovieTicket ticket3 = new MovieTicket("Avengers", "", false, "jim@example.com");
        MovieTicket ticket4 = new MovieTicket("Avengers", "", false, "jack@example.com");
        MovieTicket ticket5 = new MovieTicket("Avengers", "", false, "joe@example.com");
        System.out.println("ticket1: "+ticket1);
        System.out.println("ticket2: "+ticket2);
        System.out.println("ticket3: "+ticket3);
        System.out.println("ticket4: "+ticket4);
        System.out.println("ticket5: "+ticket5);
        System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
        Callable task1 = new MovieTicketBookingTask(ticket1);
        Callable task2 = new MovieTicketBookingTask(ticket2);
        Callable task3 = new MovieTicketBookingTask(ticket3);
        Callable task4 = new MovieTicketBookingTask(ticket4);
        Callable task5 = new MovieTicketBookingTask(ticket5);
        ExecutorService service = Executors.newFixedThreadPool(2);
        Future<String> future1 = service.submit(task1);
        System.out.println("Movie Ticket Booking App Finished");
    }
}

```

- 1.19 Next, submit five different tasks to the executor service. Submit these tasks one by one, then return future2, future3, future4, and future5 by submitting tasks 2, 3, 4, and 5, respectively. This way, you will obtain these five different futures once you submit the tasks. Retrieve the data, use `task1.result`, which corresponds to `future1.get()`. When using `future1.get()`, you will see an error indicating that a surrounding try-catch block is required. Add the necessary try-catch block to the code.

The screenshot shows the Eclipse IDE interface with the title bar "Applications : Java - Session34/src/CallableFutureApp.java - Eclipse IDE". The main window displays Java code for creating five movie tickets and printing them to System.out. A code completion tooltip is open at line 95, showing suggestions for handling exceptions. The tooltip includes imports for Callable, ExecutionException, and ExecutorService, along with a try/catch block template.

```

--snip--
62
63
64 MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
65 MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
66 MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jin@example.com");
67 MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
68 MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");
69
70
71 System.out.println("ticket1: "+ticket1);
72 System.out.println("ticket2: "+ticket1);
73 System.out.println("ticket3: "+ticket1);
74 System.out.println("ticket4: "+ticket1);
75 System.out.println("ticket5: "+ticket1);
76
77 System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
78
79
80 Callable task1 = new MovieTicketBook();
81 Callable task2 = new MovieTicketBook();
82 Callable task3 = new MovieTicketBook();
83 Callable task4 = new MovieTicketBook();
84 Callable task5 = new MovieTicketBook();
85
86
87 ExecutorService service = Executors.
88 Future<String> future1 = service.submit(
89 Future<String> future2 = service.submit(
90 Future<String> future3 = service.submit(
91 Future<String> future4 = service.submit(
92 Future<String> future5 = service.submit(
93
94 System.out.println("Task1 Result: "+future1.get());
95
96
97
98 System.out.println("Movie Ticket Booking App Finished");

```

1.20 There can be any exception when your executor service is running or interrupted exception for your thread sleep.

The screenshot shows the Eclipse IDE interface with the title bar "Applications : Java - Session34/src/CallableFutureApp.java - Eclipse IDE". The main window displays Java code for creating five movie ticket booking tasks and submitting them to an executor service. A try/catch block is present to handle any exceptions that may occur during the execution of the tasks. The code prints the result of the first task and handles any InterruptedException or ExecutionException that might be thrown.

```

--snip--
76 System.out.println("ticket5: "+ticket1);
77
78 System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
79
80 Callable task1 = new MovieTicketBookingTask(ticket1);
81 Callable task2 = new MovieTicketBookingTask(ticket2);
82 Callable task3 = new MovieTicketBookingTask(ticket3);
83 Callable task4 = new MovieTicketBookingTask(ticket4);
84 Callable task5 = new MovieTicketBookingTask(ticket5);
85
86
87 ExecutorService service = Executors.newFixedThreadPool(2);
88 Future<String> future1 = service.submit(task1);
89 Future<String> future2 = service.submit(task2);
90 Future<String> future3 = service.submit(task3);
91 Future<String> future4 = service.submit(task4);
92 Future<String> future5 = service.submit(task5);
93
94
95 try {
96     System.out.println("Task1 Result: "+future1.get());
97 } catch (InterruptedException e) {
98     e.printStackTrace();
99 } catch (ExecutionException e) {
100     // TODO Auto-generated catch block
101     e.printStackTrace();
102 }
103
104
105
106
107 System.out.println("Movie Ticket Booking App Finished");
108
109
110
111
112 }
113

```

1.21 Let us now finish this by adding the other task results along with the corresponding futures. Whenever you give as **future.get()**, you will be able to get the result in the form of string.

The screenshot shows the Eclipse IDE interface with the title bar "Java - Session 34/src/CallableFutureApp.java - Eclipse IDE". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has icons for New, Open, Save, Cut, Copy, Paste, Find, etc. The code editor displays Java code for a CallableFutureApp. The code uses ExecutorService to submit five callable tasks (MovieTicketBookingTask) and prints their results using Future.get(). It also handles InterruptedException and ExecutionException. The code ends with a final println statement.

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

# CallableFutureApp.java X
76 System.out.println("Ticket5: "+ticket5);
77
78 System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
79
80
81 Callable task1 = new MovieTicketBookingTask(ticket1);
82 Callable task2 = new MovieTicketBookingTask(ticket2);
83 Callable task3 = new MovieTicketBookingTask(ticket3);
84 Callable task4 = new MovieTicketBookingTask(ticket4);
85 Callable task5 = new MovieTicketBookingTask(ticket5);
86
87
88 ExecutorService service = Executors.newFixedThreadPool(2);
89 Future<String> future1 = service.submit(task1);
90 Future<String> future2 = service.submit(task2);
91 Future<String> future3 = service.submit(task3);
92 Future<String> future4 = service.submit(task4);
93 Future<String> future5 = service.submit(task5);
94
95 try {
96     System.out.println("Task1 Result: "+future1.get());
97     System.out.println("Task2 Result: "+future2.get());
98     System.out.println("Task3 Result: "+future3.get());
99     System.out.println("Task4 Result: "+future4.get());
100    System.out.println("Task5 Result: "+future5.get());
101 } catch (InterruptedException e) {
102     e.printStackTrace();
103 } catch (ExecutionException e) {
104     // TODO Auto-generated catch block
105     e.printStackTrace();
106 }
107
108
109
110 System.out.println("Movie Ticket Booking App Finished");
111
112
```

1.22 Do remember that for every task being submitted, there is a two second of delay when you are paying for your ticket. Once the ticket is booked, internally here you can do an empty print line. This is being returned with the seat number. Also, you will be able to fetch the string.

The screenshot shows the Eclipse IDE interface with the title bar "Applications > java - Session34/src/CallableFUTUREApp.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 project structure with "CallableFutureApp.java" selected. The main editor area contains the following Java code:

```
31
32 }
33
34 class MovieTicketBookingTask implements Callable<String>{
35
36     MovieTicket ticket;
37
38     public MovieTicketBookingTask(MovieTicket ticket) {
39         this.ticket = ticket;
40     }
41
42     @Override
43     public String call() throws Exception {
44
45         System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser+"....");
46         Thread.sleep(2000);
47         ticket.seatNumber = "A"+MovieTicket.ticketNumber;
48         ticket.isTicketBooked = true;
49         MovieTicket.ticketNumber++;
50         System.out.println("Ticket Booked for "+ticket.emailUser);
51         System.out.println();
52
53         return "Ticket Booked for "+ticket.emailUser+" with seat number: "+ticket.seatNumber;
54     }
55 }
56
57
58 public class CallableFutureApp {
59
60     public static void main(String[] args) {
61
62
63         System.out.println("Movie Ticket Booking App Started");
64
65
66         MovieTicket ticket1 = new MovieTicket("Avengers", "", false, "john@example.com");
67         MovieTicket ticket2 = new MovieTicket("Avengers", "", false, "jennie@example.com");
68         MovieTicket ticket3 = new MovieTicket("Avengers", "", false, "jim@example.com");
69         MovieTicket ticket4 = new MovieTicket("Avengers", "", false, "jack@example.com");

```

1.23 Here is how you can complete the entire process. So, what is next? Thus, the code is completed and next is where you will run the program and see the outputs coming in for the five different movie ticket booking sequentially.

```

Applications : java - Session34/src/Call...
File Edit Source Refactor Navigate Search Project Run Window Help
* CallableFutureApp.java
80     System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
81
82
83     Callable<String> task1 = new MovieTicketBookingTask(ticket1);
84     Callable<String> task2 = new MovieTicketBookingTask(ticket2);
85     Callable<String> task3 = new MovieTicketBookingTask(ticket3);
86     Callable<String> task4 = new MovieTicketBookingTask(ticket4);
87     Callable<String> task5 = new MovieTicketBookingTask(ticket5);
88
89
90     ExecutorService service = Executors.newFixedThreadPool(2);
91     Future<String> future1 = service.submit(task1);
92     Future<String> future2 = service.submit(task2);
93     Future<String> future3 = service.submit(task3);
94     Future<String> future4 = service.submit(task4);
95     Future<String> future5 = service.submit(task5);
96
97     try {
98         System.out.println("Task1 Result: "+future1.get());
99         System.out.println("Task2 Result: "+future2.get());
100        System.out.println("Task3 Result: "+future3.get());
101        System.out.println("Task4 Result: "+future4.get());
102        System.out.println("Task5 Result: "+future5.get());
103    } catch (InterruptedException e) {
104        e.printStackTrace();
105    } catch (ExecutionException e) {
106        // TODO Auto-generated catch block
107        e.printStackTrace();
108    }
109
110
111    System.out.println("Movie Ticket Booking App Finished");
112
113
114 }
115
116 }

```

1.24 Now, run the application by clicking on the green play button.

```

Applications : java - Session34/src/Call...
File Edit Source Refactor Navigate Search Project Run Window Help
* CallableFutureApp Run CallableFutureApp
79     System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
80
81
82     Callable<String> task1 = new MovieTicketBookingTask(ticket1);
83     Callable<String> task2 = new MovieTicketBookingTask(ticket2);
84     Callable<String> task3 = new MovieTicketBookingTask(ticket3);
85     Callable<String> task4 = new MovieTicketBookingTask(ticket4);
86     Callable<String> task5 = new MovieTicketBookingTask(ticket5);
87
88
89     ExecutorService service = Executors.newFixedThreadPool(2);
90     Future<String> future1 = service.submit(task1);
91     Future<String> future2 = service.submit(task2);
92     Future<String> future3 = service.submit(task3);
93     Future<String> future4 = service.submit(task4);
94     Future<String> future5 = service.submit(task5);
95
96     try {
97         System.out.println("Task1 Result: "+future1.get());
98         System.out.println("Task2 Result: "+future2.get());
99         System.out.println("Task3 Result: "+future3.get());
100        System.out.println("Task4 Result: "+future4.get());
101        System.out.println("Task5 Result: "+future5.get());
102    } catch (InterruptedException e) {
103        e.printStackTrace();
104    } catch (ExecutionException e) {
105        // TODO Auto-generated catch block
106        e.printStackTrace();
107    }
108
109
110    System.out.println("Movie Ticket Booking App Finished");
111
112
113
114 }
115
116 }

```

- 1.25 As you can see, each task is being executed sequentially. There are five movie ticket objects, initially without seat numbers and without a booking status. Additionally, four processors manage the tasks, and the seat numbers A1, A2, A3, A4, and A5 are assigned. Currently, as tasks are being submitted, seat number allocation should be done before the thread sleeps.

```

Applications : Java - Session34/src/Call...
Java - Session34/src/CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console X
CallableFutureApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot/jre.full/linux.x86_64_17.0.1.v20211116-1657/re/bin/java (Jan 22, 2022, 11:23:12 AM)
Movie Ticket Booking App Started
ticket1: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
ticket2: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
ticket3: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
ticket4: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
ticket5: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
Processors: 4
Paying for the Ticket Seat for user: jennie@example.com...
Paying for the Ticket Seat for user: john@example.com...
Ticket Booked for jennie@example.com

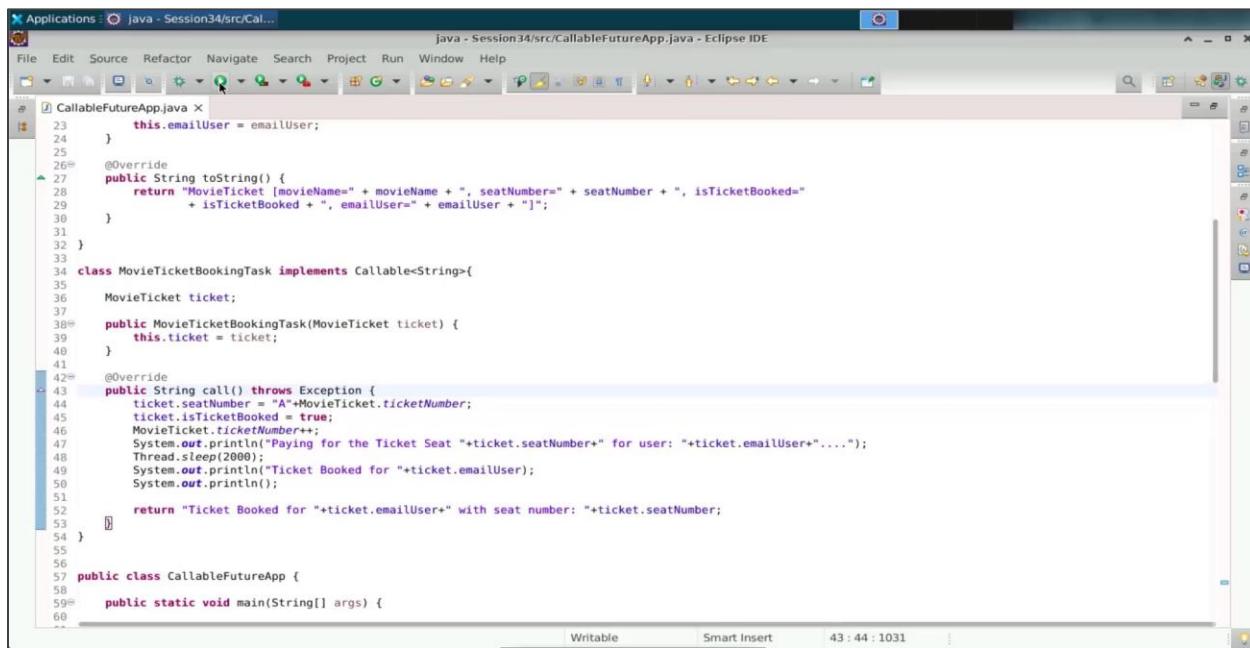
Ticket Booked for john@example.com
Paying for the Ticket Seat for user: jim@example.com...
Paying for the Ticket Seat for user: jack@example.com...
Task1 Result: Ticket Booked for john@example.com with seat number: A1
Task2 Result: Ticket Booked for jennie@example.com with seat number: A1
Ticket Booked for jim@example.com

Ticket Booked for jack@example.com
Paying for the Ticket Seat for user: joe@example.com...
Task3 Result: Ticket Booked for jim@example.com with seat number: A3
Task4 Result: Ticket Booked for jack@example.com with seat number: A4
Ticket Booked for joe@example.com

Task5 Result: Ticket Booked for joe@example.com with seat number: A5
Movie Ticket Booking App Finished

```

1.26 Navigate back to **CallableFuture.java** file and modify the code by allocating the seat, incrementing the ticket and then you can write the part of payment. You can attach the payment process later, not first, such that a better logic is being implemented. Now, re-run the code.



The screenshot shows the Eclipse IDE interface with the title bar "Applications : Java - Session34/src/Call..." and "java - Session34/src/CallableFutureApp.java - Eclipse IDE". The main window displays the Java code for CallableFutureApp.java:

```
Applications : Java - Session34/src/Call...
File Edit Source Refactor Navigate Search Project Run Window Help
 CallableFutureApp.java X
 23     this.emailUser = emailUser;
 24   }
 25
 26@ 27   public String toString() {
 28     return "MovieTicket [movieName=" + movieName + ", seatNumber=" + seatNumber + ", isTicketBooked="
 29           + isTicketBooked + ", emailUser=" + emailUser + "]";
 30   }
 31
 32 }
 33
 34 class MovieTicketBookingTask implements Callable<String>{
 35
 36   MovieTicket ticket;
 37
 38@ 39   public MovieTicketBookingTask(MovieTicket ticket) {
 40     this.ticket = ticket;
 41   }
 42
 43   @Override
 44   public String call() throws Exception {
 45     ticket.seatNumber = "A"+MovieTicket.ticketNumber;
 46     ticket.isTicketBooked = true;
 47     MovieTicket.ticketNumber++;
 48     System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser+"....");
 49     Thread.sleep(2000);
 50     System.out.println("Ticket Booked for "+ticket.emailUser);
 51     System.out.println();
 52
 53     return "Ticket Booked for "+ticket.emailUser+ " with seat number: "+ticket.seatNumber;
 54   }
 55
 56
 57 public class CallableFutureApp {
 58
 59   public static void main(String[] args) {
 60

```

The code implements a `CallableFutureApp` class with a `main` method and a `MovieTicketBookingTask` inner class that implements `Callable<String>`. The `call` method of the inner class allocates a seat, increments the ticket number, and prints payment confirmation messages. The `toString` method of the `MovieTicket` class is also shown.

1.27 As shown in the output, different tasks are being submitted and producing results. One challenge is that John and Jenny are both contending for the same A1 seat due to the introduction of `thread.sleep()`. Without `thread.sleep()`, the results are better. Therefore, without any delay, the code runs perfectly, and the A1, A2, A3, A4, and A5 tickets are correctly allocated to various users.

The screenshot shows the Eclipse IDE interface with the Java perspective selected. The title bar reads "Applications java - Session 34/src/CallableFutureApp.java - Eclipse IDE". The left sidebar shows the package structure: "src" and "CallableFutureApp [Java Application]". The main editor area contains the code for `CallableFutureApp.java`, which defines a `CallableFuture` task for booking movie tickets. The console tab at the bottom displays the execution output:

```
Movie Ticket Booking App Started
[...]
tikect1: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
tikect2: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
tikect3: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
tikect4: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
tikect5: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
Processors: 4
Paying for the Ticket Seat A7 for user: john@example.com...
Paying for the Ticket Seat A2 for user: jennie@example.com...
Ticket Booked for john@example.com

Ticket Booked for jennie@example.com
Paying for the Ticket Seat A3 for user: jim@example.com...
Task1 Result: Ticket Booked for john@example.com with seat number: A1
Paying for the Ticket Seat A4 for user: jack@example.com...
Task2 Result: Ticket Booked for jennie@example.com with seat number: A1
Ticket Booked for jim@example.com

Paying for the Ticket Seat A5 for user: joe@example.com...
Ticket Booked for jack@example.com

Task3 Result: Ticket Booked for jim@example.com with seat number: A3
Task4 Result: Ticket Booked for jack@example.com with seat number: A4
Ticket Booked for joe@example.com

Task5 Result: Ticket Booked for joe@example.com with seat number: A5
Movie Ticket Booking App Finished
```

1.28 Since, you were trying to introduce a sleep for a payment a scenario, whenever a thread was sleeping then have another thread being executed. Hence, comment the code `thread.sleep()`. If you are implementing callable, you are going to work with futures. Hence submitting your tasks to the executor service, you will be able to get something in future.

The screenshot shows the Eclipse IDE interface with the title bar "Applications : Java - Session34/src/Cal..." and "File - Java - Session34/src/CallableFutureApp.java - Eclipse IDE". The main window displays the Java code for `CallableFutureApp.java`. The code defines a `MovieTicketBookingTask` class that implements the `Callable<String>` interface. It contains a constructor that takes a `MovieTicket` object and initializes it. The `call()` method performs some processing, including printing the booking details to `System.out`, sleeping for 2 seconds, and then printing the ticket booked message. The `CallableFutureApp` class has a static `main` method that prints a welcome message and creates five `MovieTicket` objects with specific details.

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

Callables Future Tasks CallableFutureApp.java ×
33
34 class MovieTicketBookingTask implements Callable<String>{
35
36     MovieTicket ticket;
37
38     public MovieTicketBookingTask(MovieTicket ticket) {
39         this.ticket = ticket;
40     }
41
42     @Override
43     public String call() throws Exception {
44         ticket.seatNumber = "A"+MovieTicket.ticketNumber;
45         ticket.isTicketBooked = true;
46         MovieTicket.ticketNumber++;
47         System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser+"....");
48         //Thread.sleep(2000);
49         System.out.println("Ticket Booked for "+ticket.emailUser);
50         System.out.println();
51
52         return "Ticket Booked for "+ticket.emailUser+" with seat number: "+ticket.seatNumber;
53     }
54
55
56
57 public class CallableFutureApp {
58
59     public static void main(String[] args) {
60
61         System.out.println("Movie Ticket Booking App Started");
62
63
64         MovieTicket ticket1 = new MovieTicket("Avengers", "", false, "john@example.com");
65         MovieTicket ticket2 = new MovieTicket("Avengers", "", false, "jennie@example.com");
66         MovieTicket ticket3 = new MovieTicket("Avengers", "", false, "jim@example.com");
67         MovieTicket ticket4 = new MovieTicket("Avengers", "", false, "jack@example.com");
68         MovieTicket ticket5 = new MovieTicket("Avengers", "", false, "joe@example.com");
69
70     }
71 }
```

1.29 Let us try commenting out the code with get method. Now, without this code snippet, run the code.

The screenshot shows the Eclipse IDE interface with the title bar "Applications > java - Session34/src/Call...". 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 code editor displays "CallableFutureApp.java" with the following content:

```
System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());  
Callable<String> task1 = new MovieTicketBookingTask(ticket1);  
Callable<String> task2 = new MovieTicketBookingTask(ticket2);  
Callable<String> task3 = new MovieTicketBookingTask(ticket3);  
Callable<String> task4 = new MovieTicketBookingTask(ticket4);  
Callable<String> task5 = new MovieTicketBookingTask(ticket5);  
ExecutorService service = Executors.newFixedThreadPool(2);  
Future<String> future1 = service.submit(task1);  
Future<String> future2 = service.submit(task2);  
Future<String> future3 = service.submit(task3);  
Future<String> future4 = service.submit(task4);  
Future<String> future5 = service.submit(task5);  
/*try {  
    System.out.println("Task1 Result: "+future1.get());  
    System.out.println("Task2 Result: "+future2.get());  
    System.out.println("Task3 Result: "+future3.get());  
    System.out.println("Task4 Result: "+future4.get());  
    System.out.println("Task5 Result: "+future5.get());  
} catch (InterruptedException e) {  
    e.printStackTrace();  
} catch (ExecutionException e) {  
    // TODO Auto-generated catch block  
    e.printStackTrace();  
}  
*/  
System.out.println("Movie Ticket Booking App Finished");
```

1.30 You can view the output that the tasks are getting executed. But the get method will return you the result. Thus, when your thread terminates and if you want to get the result back, you need to execute the get method in the future. This comes to the end of the discussion on how to implement future and callable in Java. Thank you.

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

- Title Bar:** Practice Labs - DEFAULT
- Toolbar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Left Panel:** Shows the file `CallableFutureApp.java`.
- Right Panel:** Shows the `Console` tab with the following output:

```

processors: 4
Movie Ticket Booking App Finished
Paying for the Ticket Seat A1 for user: john@example.com...
Paying for the Ticket Seat A1 for user: jennie@example.com...
Ticket Booked for john@example.com
Ticket Booked for jennie@example.com

Paying for the Ticket Seat A3 for user: jim@example.com...
Ticket Booked for jim@example.com

Paying for the Ticket Seat A4 for user: jack@example.com...
Paying for the Ticket Seat A5 for user: joe@example.com...
Ticket Booked for jack@example.com
Ticket Booked for joe@example.com

```

By following these steps, you have successfully demonstrated the usage of the `Callable` interface and futures in Java.