

Lesson 04 Demo 11

Implementing Callable and Future

Objective: To demonstrate the usage of callable interface and futures

Tools required: Eclipse IDE

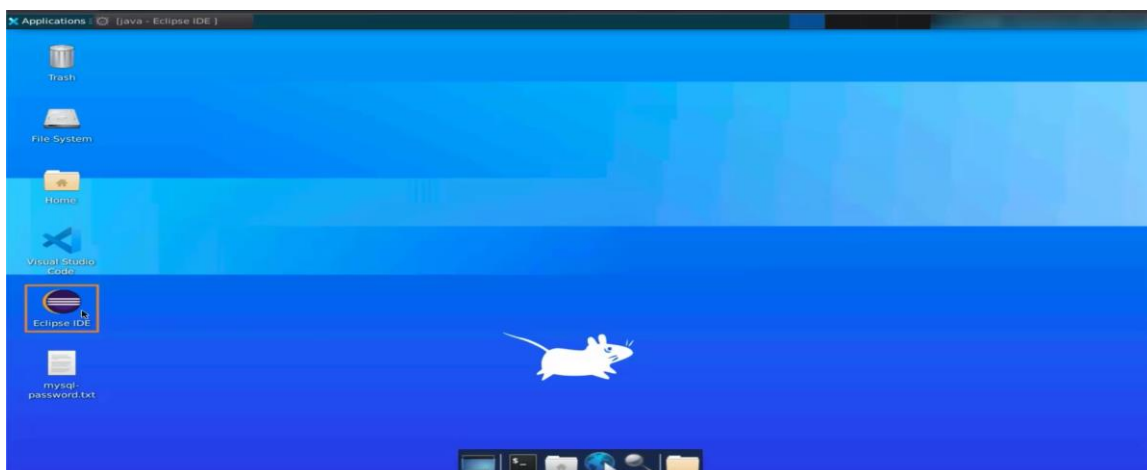
Prerequisites: None

Steps to be followed:

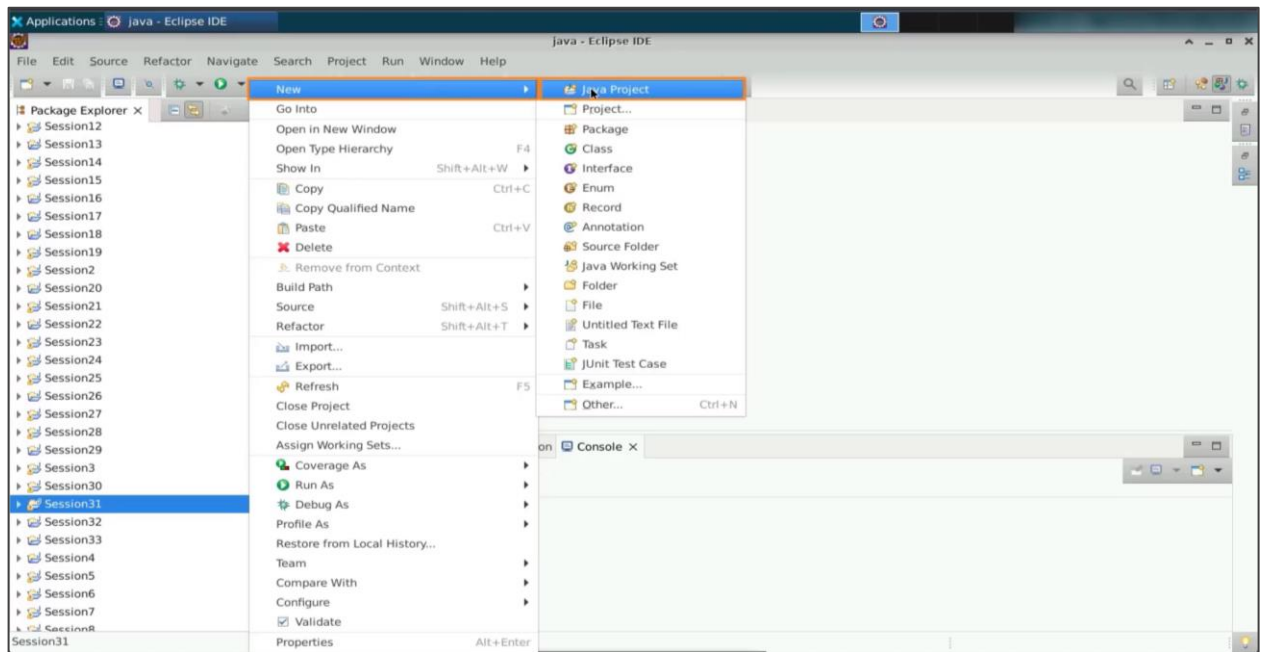
1. Implement a thread using Callable and Future Interfaces along with the suitable scenarios

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

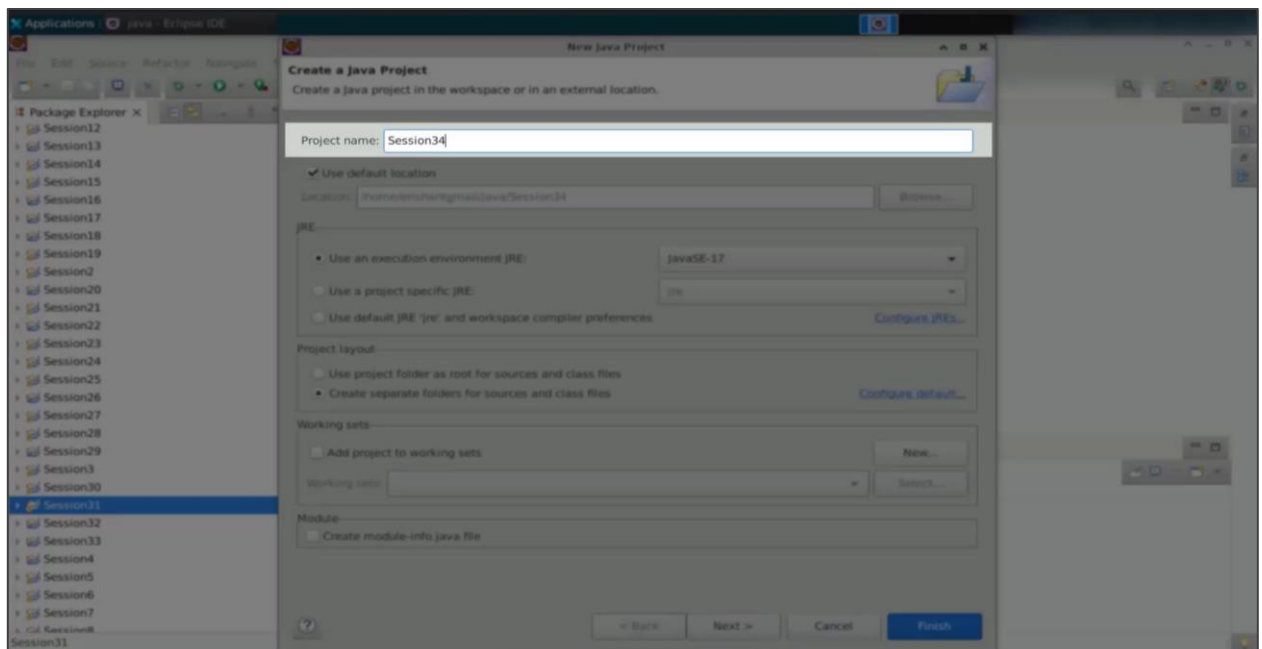
- 1.1 If you wish to create a thread, you can either extend the `Thread` class or implement the `Runnable` interface. When you start the thread, it runs asynchronously, and you will not be notified of its termination. However, if you need to capture results from a thread, Java provides the `Callable`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



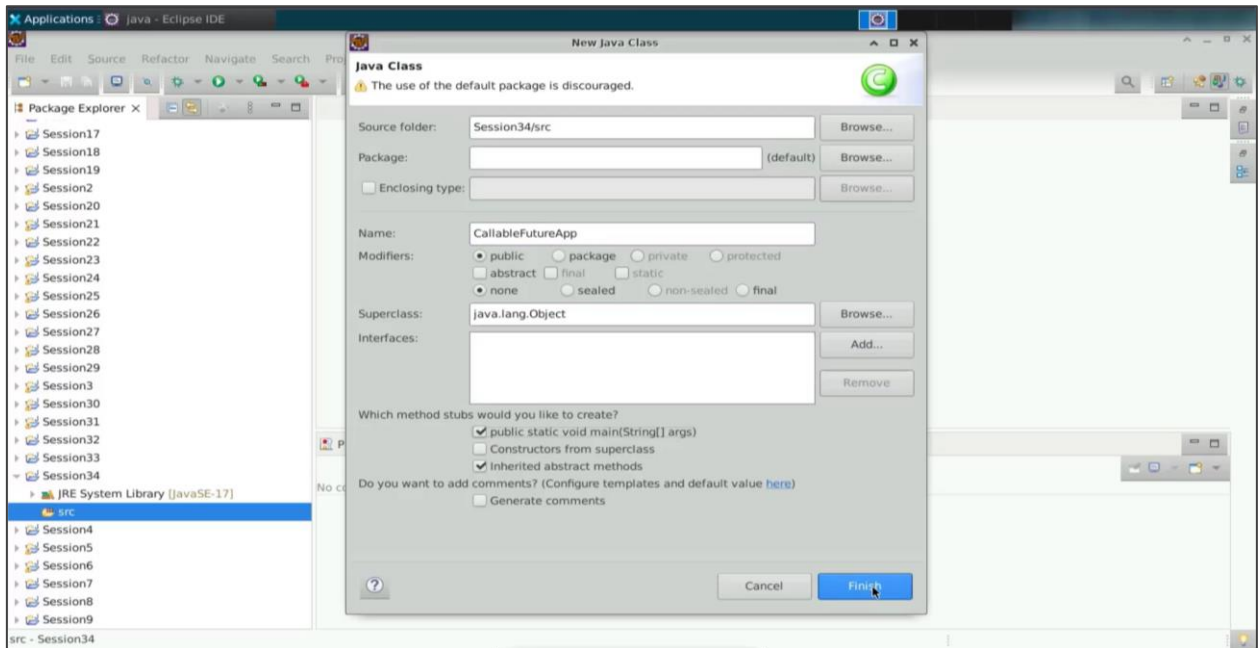
1.2 In the package Explorer, create a new Java project by selecting **New > Java Project**.



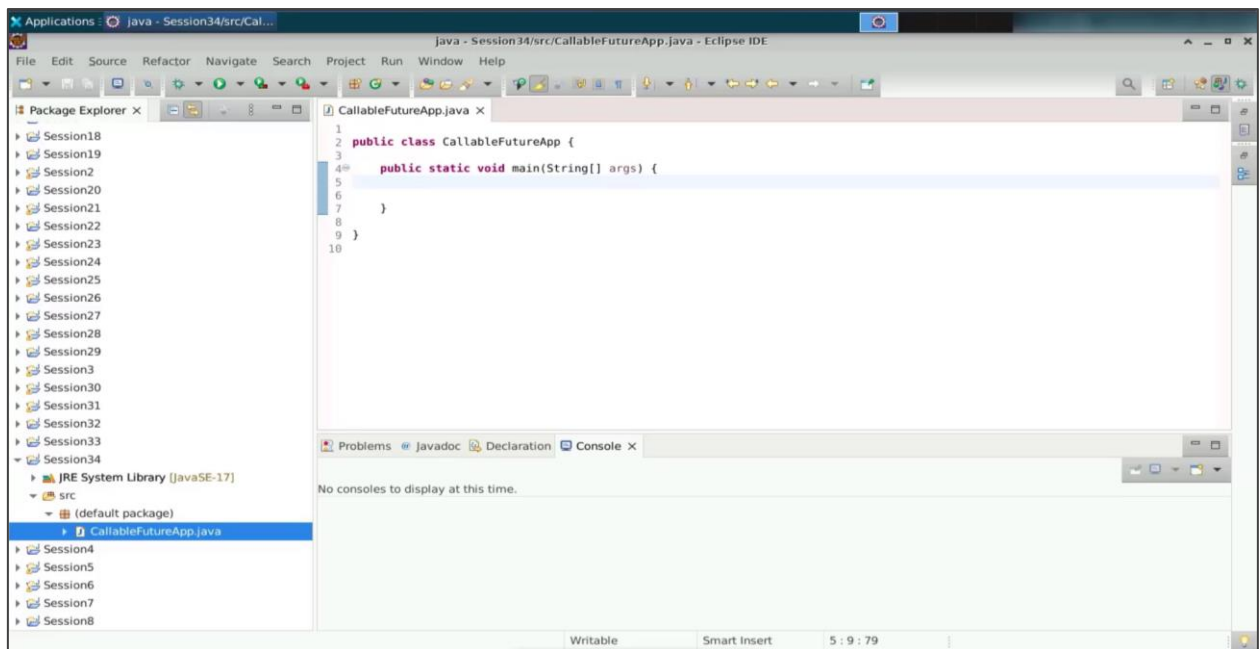
1.3 Name the project **Session34**, uncheck **Create a module info.java file**, and press **Finish**.



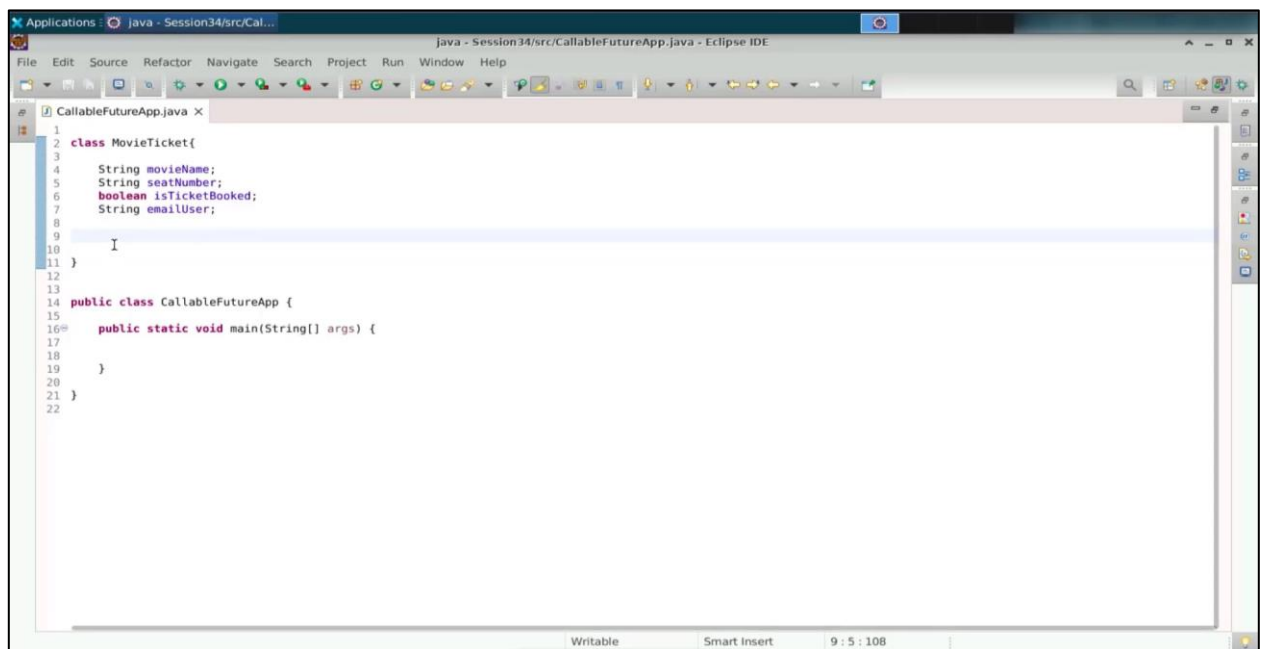
- 1.4 In the source, do a right click and create a new class and name the class as **CallableFutureApp** along with the main method. Click **Finish**.



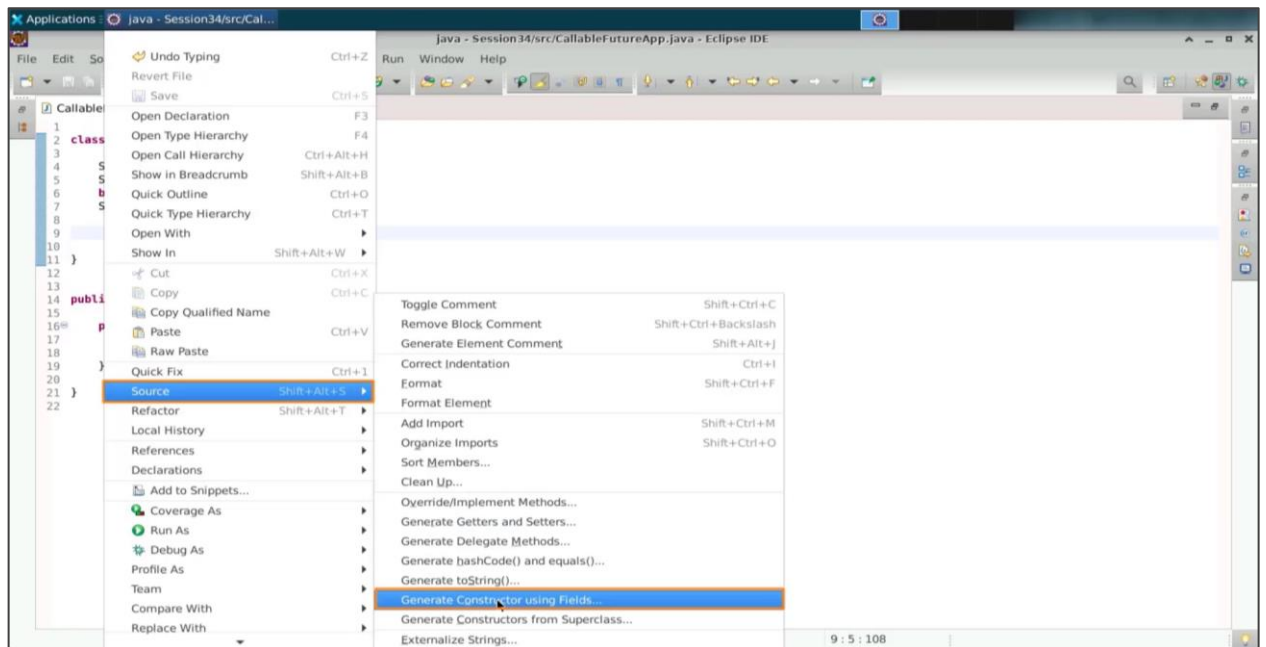
Note: For writing this demonstration, you will learn about the executor service in Java, whose key role is to execute the tasks on the threads asynchronously. When you will create your threads with callable, then the executable service will submit the task with the help of which you can get some results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



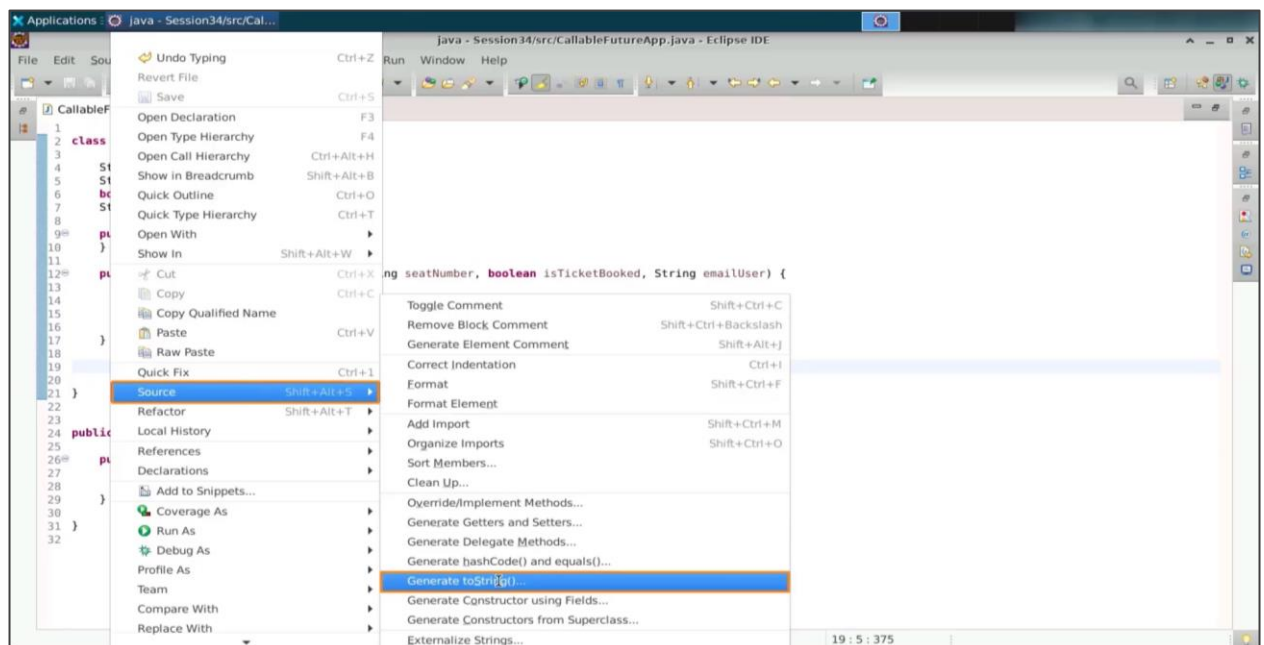
- 1.5 Create a class called Movie Ticket. And for movie tickets, let us give a movie name. Then you can give the seat number. Next, is the ticket booked, or is a ticket available? Let us keep a status known as Is the ticket booked? Now, let us record who booked it, and the email for the user.



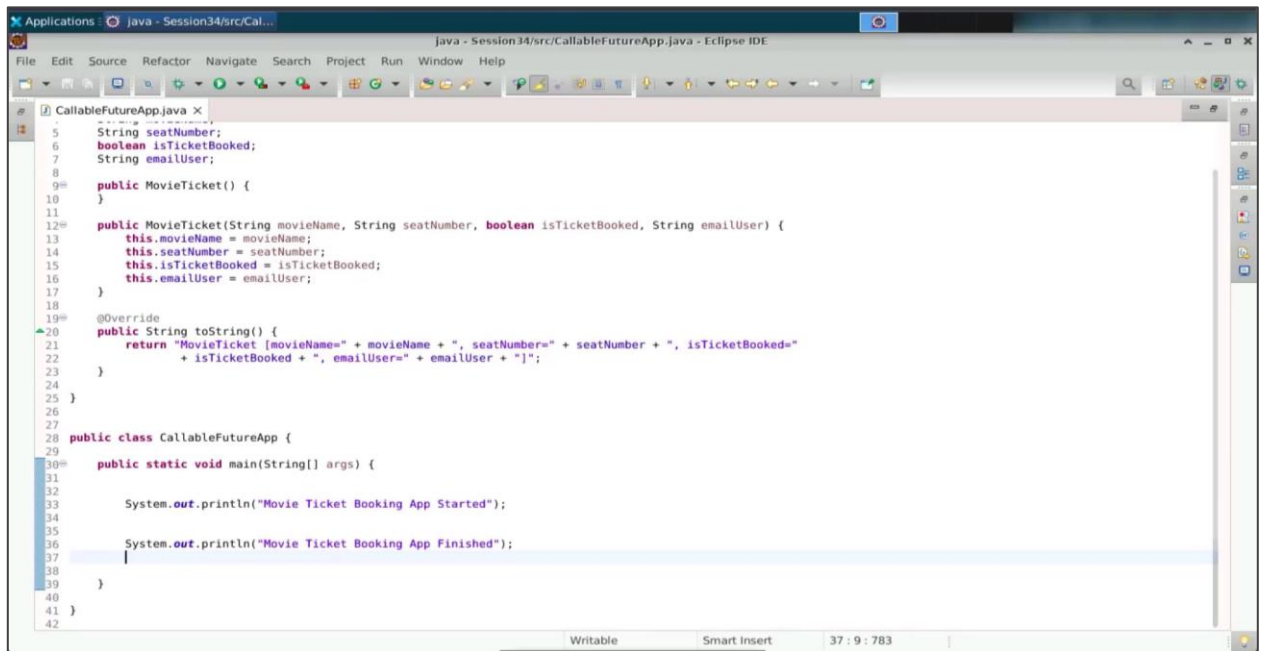
1.6 Now, right-click in the code console window and select **Source > Generate Constructor using Fields**. Then, click Finish.



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.

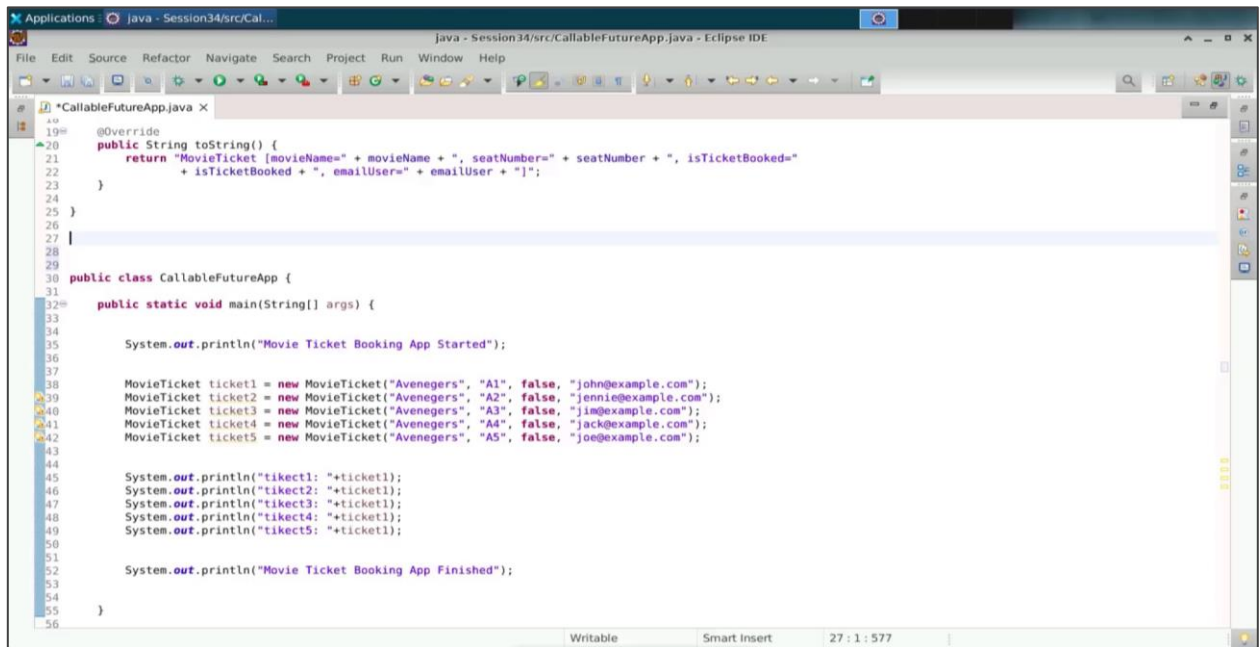


1.8 In the main method, consider there is a movie booking app, where you are supposed to book the movie tickets. Let us give a print statement as **Movie Ticket Booking App Started**. The last statement will be entered as **Movie Ticket Booking App Finished**. This is with the application logs coming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



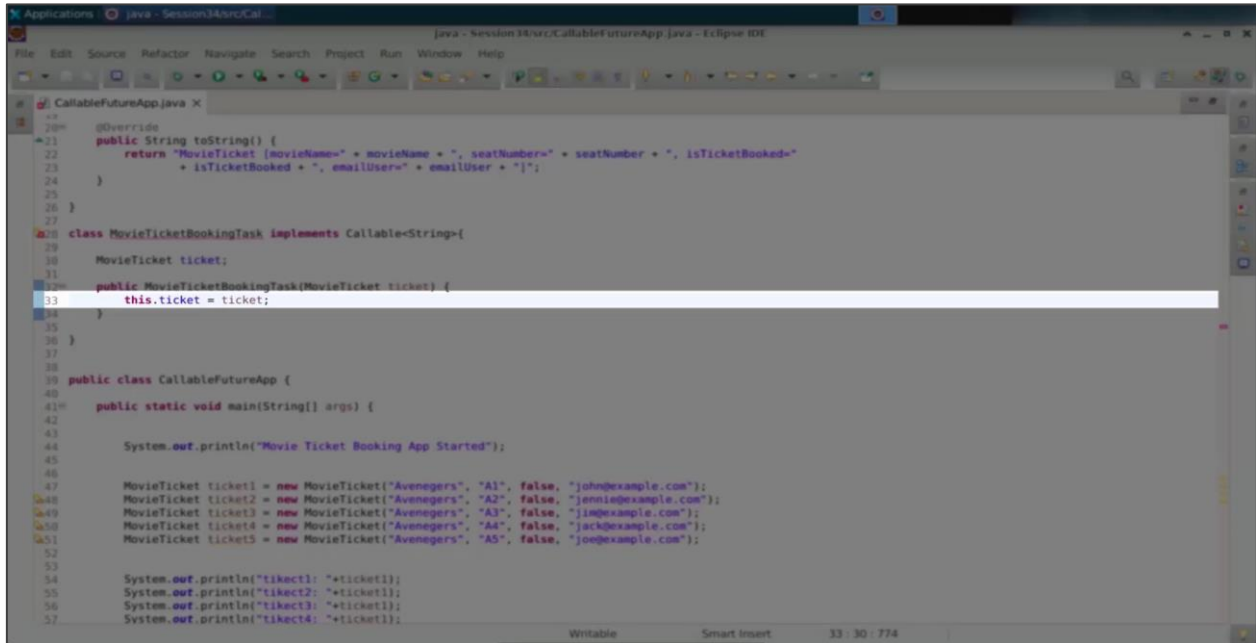
```
1  package org.example;
2
3  import java.util.concurrent.Callable;
4  import java.util.concurrent.Future;
5
6  String seatNumber;
7  boolean isTicketBooked;
8  String emailUser;
9
10 public MovieTicket() {
11 }
12
13 public MovieTicket(String movieName, String seatNumber, boolean isTicketBooked, String emailUser) {
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 + "]\n";
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.



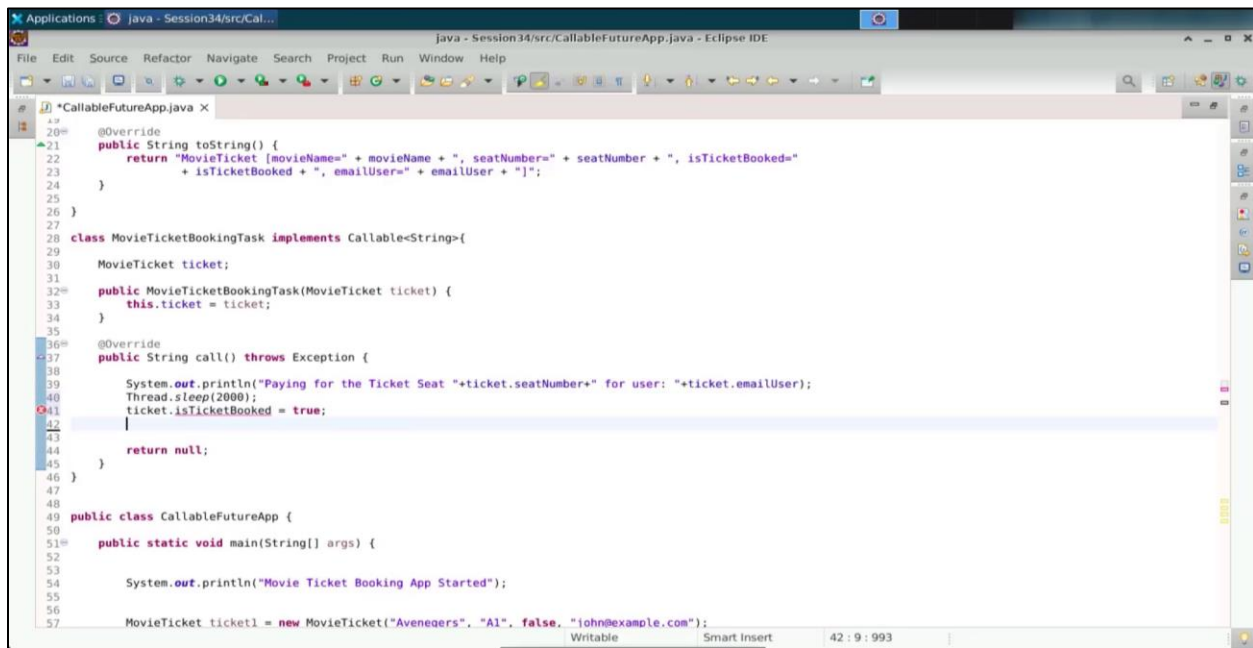
```
19: @Override
20: public String toString() {
21:     return "MovieTicket [movieName=" + movieName + ", seatNumber=" + seatNumber + ", isTicketBooked="
22:         + isTicketBooked + ", emailUser=" + emailUser + "]\n";
23: }
24: }
25:
26:
27:
28:
29:
30: public class CallableFutureApp {
31:
32:  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



```
1  Applications  java - Session 14/src/Cal...
2  java - Session 14/src/CallableFutureApp.java - Eclipse IDE
3  File  Edit  Source  Refactor  Navigate  Search  Project  Run  Window  Help
4
5  CallableFutureApp.java x
6  20  @Override
7  21  public String toString() {
8  22      return "MovieTicket {movieName=" + movieName + ", seatNumber=" + seatNumber + ", isTicketBooked="
9  23          + isTicketBooked + ", emailUser=" + emailUser + "}";
10 24  }
11 25
12 26  }
13 27
14 28  class MovieTicketBookingTask implements Callable<String>{
15 29
16 30      MovieTicket ticket;
17 31
18 32      public MovieTicketBookingTask(MovieTicket ticket) {
19 33          this.ticket = ticket;
20 34      }
21 35
22 36  }
23 37
24 38
25 39  public class CallableFutureApp {
26 40
27 41      public static void main(String[] args) {
28 42
29 43          System.out.println("Movie Ticket Booking App Started");
30 44
31 45
32 46          MovieTicket ticket1 = new MovieTicket("Avenegers", "A1", false, "john@example.com");
33 47
34 48          MovieTicket ticket2 = new MovieTicket("Avenegers", "A2", false, "jennie@example.com");
35 49
36 50          MovieTicket ticket3 = new MovieTicket("Avenegers", "A3", false, "jim@example.com");
37 51
38 52          MovieTicket ticket4 = new MovieTicket("Avenegers", "A4", false, "jack@example.com");
39 53
40 54          MovieTicket ticket5 = new MovieTicket("Avenegers", "A5", false, "joe@example.com");
41 55
42 56          System.out.println("ticket1: "+ticket1);
43 57          System.out.println("ticket2: "+ticket2);
44 58          System.out.println("ticket3: "+ticket3);
45 59          System.out.println("ticket4: "+ticket4);
46 60
47 61      }
48 62  }
49
50  Writable  Smart Insert  33 : 30 : 774
```


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.



```
Applications - java - Session34/src/Cal...
java - Session34/src/CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

CallableFutureApp.java x
20 @Override
21 public String toString() {
22     return "MovieTicket [movieName=" + movieName + ", seatNumber=" + seatNumber + ", isTicketBooked="
23         + isTicketBooked + ", emailUser=" + emailUser + "]\n";
24 }
25
26 }
27
28 class MovieTicketBookingTask implements Callable<String>{
29     MovieTicket ticket;
30
31     public MovieTicketBookingTask(MovieTicket ticket) {
32         this.ticket = ticket;
33     }
34
35
36 @Override
37 public String call() throws Exception {
38     System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser);
39     Thread.sleep(2000);
40     ticket.isTicketBooked = true;
41
42     return null;
43 }
44
45 }
46
47
48 public class CallableFutureApp {
49     public static void main(String[] args) {
50
51
52         System.out.println("Movie Ticket Booking App Started");
53
54         MovieTicket ticket1 = new MovieTicket("Avenegers", "A1", false, "john@example.com");
55
56
57 }
```

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  class MovieTicketBookingTask implements Callable<String>{
28      MovieTicket ticket;
29      public MovieTicketBookingTask(MovieTicket ticket) {
30          this.ticket = ticket;
31      }
32      @Override
33      public String call() throws Exception {
34          System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser);
35          Thread.sleep(2000);
36          ticket.isTicketBooked = true;
37          System.out.println("Ticket Booked for "+ticket.emailUser);
38          return ""
39      }
40  }
41  }
42  public class CallableFutureApp {
43      public static void main(String[] args) {

```

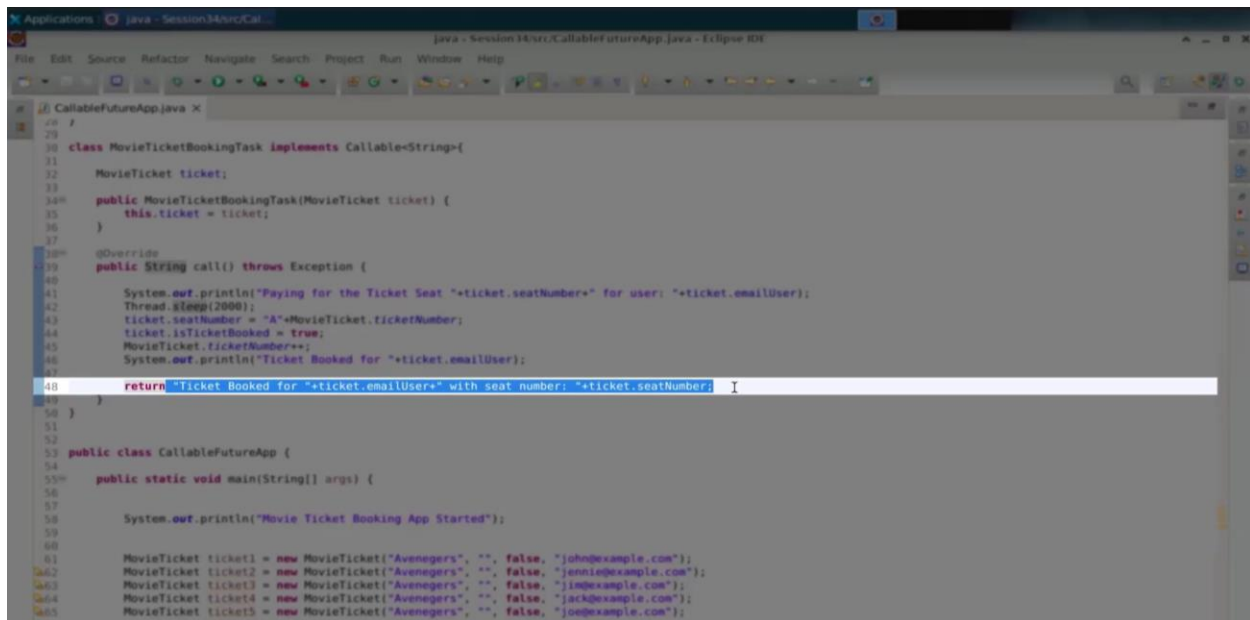
1.13 You have one of the static variables called ticket number, which begins with one integer type. This ticket serial number starts from 1 and before you book the ticket, you will write as ticket.seatnumber goes as the A row + from the movie ticket.ticketNumber.

```

40  }
41  }
42  class MovieTicketBookingTask implements Callable<String>{
43      MovieTicket ticket;
44      public MovieTicketBookingTask(MovieTicket ticket) {
45          this.ticket = ticket;
46      }
47      @Override
48      public String call() throws Exception {
49          System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser);
50          Thread.sleep(2000);
51          ticket.seatNumber = "A"+MovieTicket.ticketNumber;
52          ticket.isTicketBooked = true;
53          System.out.println("Ticket Booked for "+ticket.emailUser);
54          return ""
55      }
56  }
57  }
58  public class CallableFutureApp {
59      public static void main(String[] args) {
60          System.out.println("Movie Ticket Booking App Started");
61          MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
62          MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
63          MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
64          MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
65          MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");

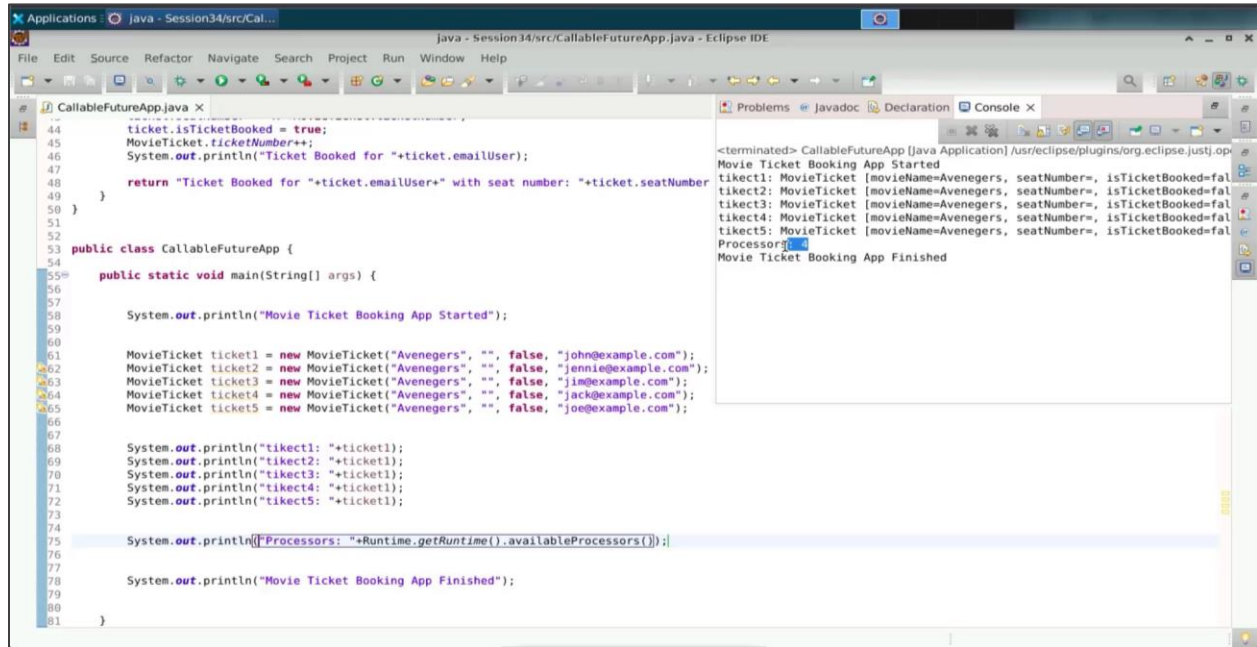
```

- 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.



```
1  Applications  java - Session14/src/Ca...
2  java - Session14/src/Ca...
3  File Edit Source Refactor Navigate Search Project Run Window Help
4
5  CallableFutureApp.java x
6  28
7  29
8  30 class MovieTicketBookingTask implements Callable<String>{
9  31
10 32     MovieTicket ticket;
11
12 33
13 34 public MovieTicketBookingTask(MovieTicket ticket) {
14 35     this.ticket = ticket;
15 36 }
16
17 37
18 38 @Override
19 39 public String call() throws Exception {
20 40
21 41     System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser);
22 42     Thread.sleep(2000);
23 43     ticket.seatNumber = "A"+MovieTicket.ticketNumber;
24 44     ticket.isTicketBooked = true;
25 45     MovieTicket.ticketNumber++;
26 46     System.out.println("Ticket Booked for "+ticket.emailUser);
27 47
28 48     return "Ticket Booked for "+ticket.emailUser+" with seat number: "+ticket.seatNumber;
29 49 }
30
31 50 }
32
33 51
34 52
35 53 public class CallableFutureApp {
36 54
37 55 public static void main(String[] args) {
38 56
39 57     System.out.println("Movie Ticket Booking App Started");
40 58
41 59
42 60     MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
43 61     MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
44 62     MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "im@example.com");
45 63     MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
46 64     MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");
47 65 }
```

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.



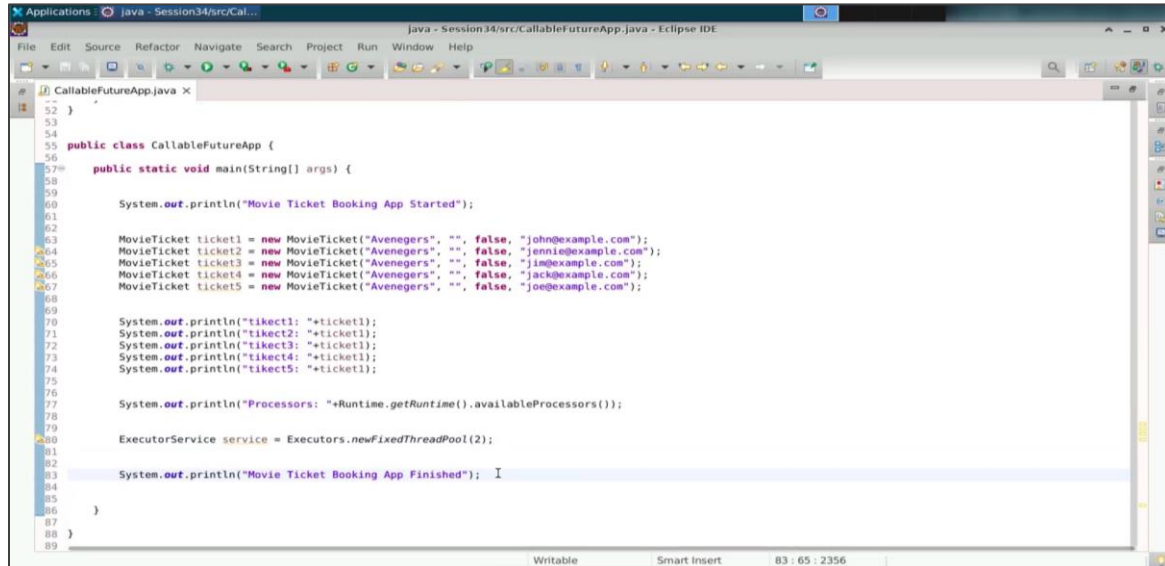
The screenshot shows the Eclipse IDE with the file `CallableFutureApp.java` open. The code defines a `CallableFutureApp` class with a `main` method. The `main` method prints the number of available processors using `Runtime.getRuntime().availableProcessors()`. The console output shows the application started, printed the number of processors (4), and finished.

```
java - Session34/src/Cal...
java - Session34/src/CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

CallableFutureApp.java X
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         System.out.println("Movie Ticket Booking App Started");
57
58         MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
59         MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
60         MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
61         MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
62         MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");
63
64         System.out.println("ticket1: "+ticket1);
65         System.out.println("ticket2: "+ticket2);
66         System.out.println("ticket3: "+ticket3);
67         System.out.println("ticket4: "+ticket4);
68         System.out.println("ticket5: "+ticket5);
69
70         System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
71
72         System.out.println("Movie Ticket Booking App Finished");
73
74     }
75 }

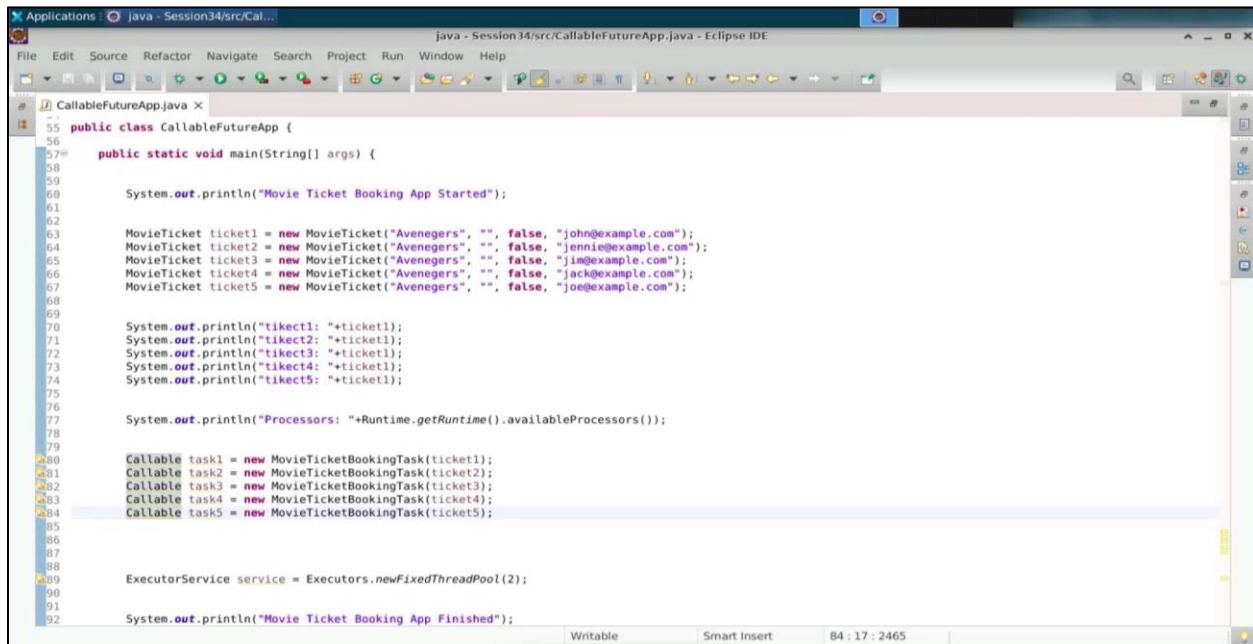
Problems Javadoc Declaration Console X
<terminated> CallableFutureApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj...
Movie Ticket Booking App Started
ticket1: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
ticket2: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
ticket3: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
ticket4: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
ticket5: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
Processors: 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.



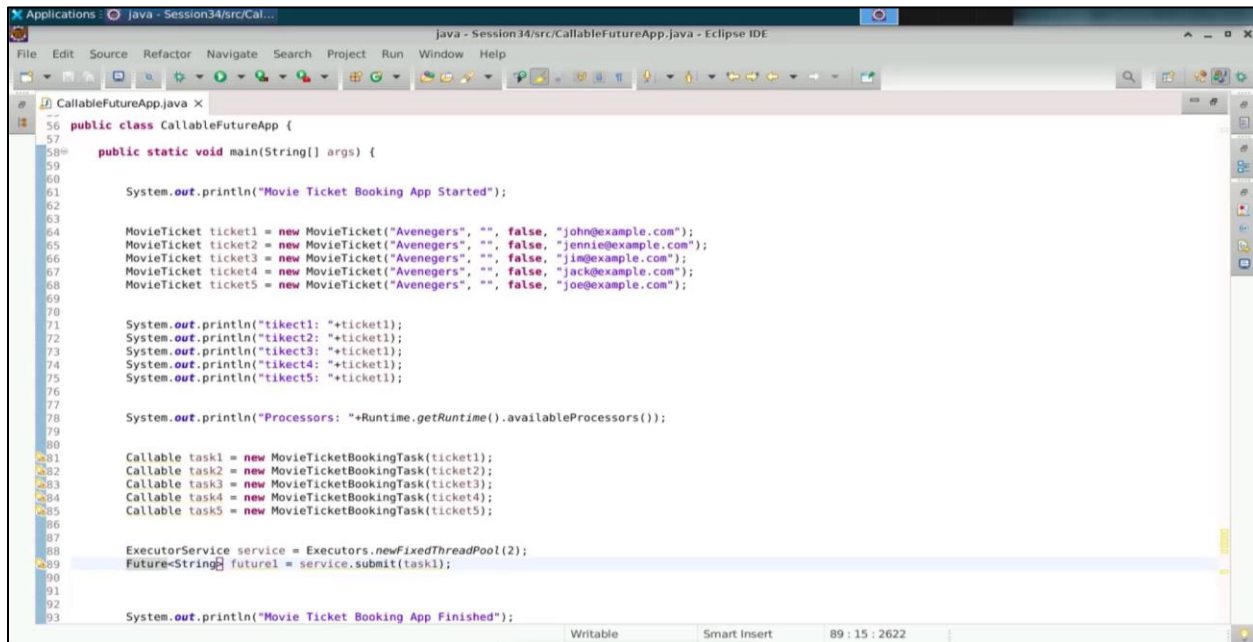
```
52 }
53
54 public class CallableFutureApp {
55     public static void main(String[] args) {
56
57         System.out.println("Movie Ticket Booking App Started");
58
59         MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
60         MovieTicket ticket2 = new MovieTicket("Avenegers", "", 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         ExecutorService service = Executors.newFixedThreadPool(2);
74
75         System.out.println("Movie Ticket Booking App Finished");
76     }
77 }
78
79 }
```

- 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.



```
55 public class CallableFutureApp {
56
57     public static void main(String[] args) {
58
59         System.out.println("Movie Ticket Booking App Started");
60
61         MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
62         MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
63         MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
64         MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
65         MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");
66
67         System.out.println("ticket1: "+ticket1);
68         System.out.println("ticket2: "+ticket2);
69         System.out.println("ticket3: "+ticket3);
70         System.out.println("ticket4: "+ticket4);
71         System.out.println("ticket5: "+ticket5);
72
73         System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
74
75         Callable task1 = new MovieTicketBookingTask(ticket1);
76         Callable task2 = new MovieTicketBookingTask(ticket2);
77         Callable task3 = new MovieTicketBookingTask(ticket3);
78         Callable task4 = new MovieTicketBookingTask(ticket4);
79         Callable task5 = new MovieTicketBookingTask(ticket5);
80
81         ExecutorService service = Executors.newFixedThreadPool(2);
82
83         System.out.println("Movie Ticket Booking App Finished");
84     }
85 }
```

- 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.



```
1  public class CallableFutureApp {
2
3      public static void main(String[] args) {
4
5          System.out.println("Movie Ticket Booking App Started");
6
7          MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
8          MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
9          MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jin@example.com");
10         MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
11         MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");
12
13         System.out.println("ticket1: " + ticket1);
14         System.out.println("ticket2: " + ticket2);
15         System.out.println("ticket3: " + ticket3);
16         System.out.println("ticket4: " + ticket4);
17         System.out.println("ticket5: " + ticket5);
18
19         System.out.println("Processors: " + Runtime.getRuntime().availableProcessors());
20
21         Callable task1 = new MovieTicketBookingTask(ticket1);
22         Callable task2 = new MovieTicketBookingTask(ticket2);
23         Callable task3 = new MovieTicketBookingTask(ticket3);
24         Callable task4 = new MovieTicketBookingTask(ticket4);
25         Callable task5 = new MovieTicketBookingTask(ticket5);
26
27         ExecutorService service = Executors.newFixedThreadPool(2);
28         Future<String> future1 = service.submit(task1);
29
30         System.out.println("Movie Ticket Booking App Finished");
31     }
32 }
```

- 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.


```

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, "jim@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: "+ticket2);
73 System.out.println("ticket3: "+ticket3);
74 System.out.println("ticket4: "+ticket4);
75 System.out.println("ticket5: "+ticket5);
76
77
78 System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
79
80
81 Callable task1 = new MovieTicketBook
82 Callable task2 = new MovieTicketBook
83 Callable task3 = new MovieTicketBook
84 Callable task4 = new MovieTicketBook
85 Callable task5 = new MovieTicketBook
86
87
88 ExecutorService service = Executors.
89 Future<String> future1 = service.sub
90 Future<String> future2 = service.sub
91 Future<String> future3 = service.sub
92 Future<String> future4 = service.sub
93 Future<String> future5 = service.sub
94
95 System.out.println("Task1 Result: "+future1.get());
96
97
98
99 System.out.println("Movie Ticket Booking App Finished");

```

Unhandled exception type ExecutionException Writable Smart Insert 95 : 58 [13]

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

```

76
77
78 System.out.println("ticket5: "+ticket1);
79
80
81 System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
82
83
84 Callable task1 = new MovieTicketBookingTask(ticket1);
85 Callable task2 = new MovieTicketBookingTask(ticket2);
86 Callable task3 = new MovieTicketBookingTask(ticket3);
87 Callable task4 = new MovieTicketBookingTask(ticket4);
88 Callable task5 = new MovieTicketBookingTask(ticket5);
89
90
91 ExecutorService service = Executors.newFixedThreadPool(2);
92 Future<String> future1 = service.submit(task1);
93 Future<String> future2 = service.submit(task2);
94 Future<String> future3 = service.submit(task3);
95 Future<String> future4 = service.submit(task4);
96 Future<String> future5 = service.submit(task5);
97
98
99 try {
100     System.out.println("Task1 Result: "+future1.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 System.out.println("Movie Ticket Booking App Finished");
110
111
112 }
113

```

Writable Smart Insert 97 : 64 : 2977

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.

```

16 System.out.println("ticket5: "+ticket1);
17
18 System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
19
20 Callable task1 = new MovieTicketBookingTask(ticket1);
21 Callable task2 = new MovieTicketBookingTask(ticket2);
22 Callable task3 = new MovieTicketBookingTask(ticket3);
23 Callable task4 = new MovieTicketBookingTask(ticket4);
24 Callable task5 = new MovieTicketBookingTask(ticket5);
25
26 ExecutorService service = Executors.newFixedThreadPool(2);
27 Future<String> future1 = service.submit(task1);
28 Future<String> future2 = service.submit(task2);
29 Future<String> future3 = service.submit(task3);
30 Future<String> future4 = service.submit(task4);
31 Future<String> future5 = service.submit(task5);
32
33 try {
34     System.out.println("Task1 Result: "+future1.get());
35     System.out.println("Task2 Result: "+future2.get());
36     System.out.println("Task3 Result: "+future3.get());
37     System.out.println("Task4 Result: "+future4.get());
38     System.out.println("Task5 Result: "+future5.get());
39 } catch (InterruptedException e) {
40     e.printStackTrace();
41 } catch (ExecutionException e) {
42     // 1000 Auto-generated catch block
43     e.printStackTrace();
44 }
45
46 System.out.println("Movie Ticket Booking App Finished");
47
48

```

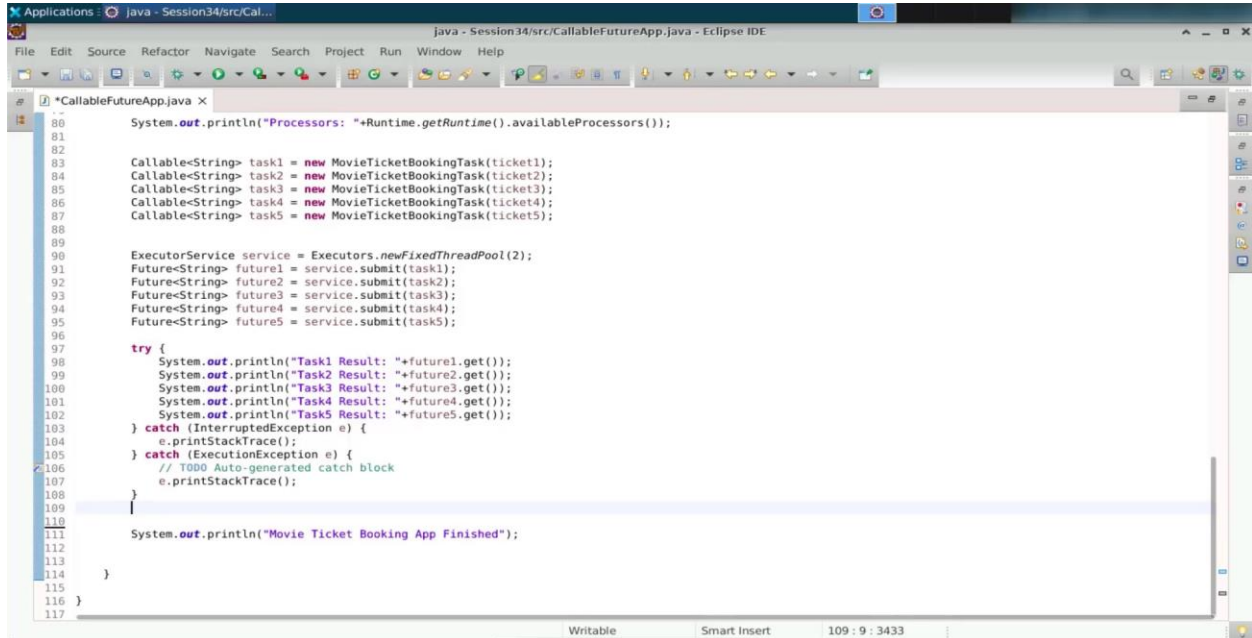
1.22 Do remember that for every task being submitted, there is a two second of delay when you are paying for your ticket. Once the ticket is booked, internally here you can do an empty print line. This is being returned with the seat number. Also, you will be able to fetch the string.

```

24 }
25
26 class MovieTicketBookingTask implements Callable<String>{
27     MovieTicket ticket;
28
29     public MovieTicketBookingTask(MovieTicket ticket) {
30         this.ticket = ticket;
31     }
32
33     @Override
34     public String call() throws Exception {
35         System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser+"...");
36         Thread.sleep(2000);
37         ticket.seatNumber = "A"+MovieTicket.ticketNumber;
38         ticket.isTicketBooked = true;
39         MovieTicket.ticketNumber++;
40         System.out.println("Ticket Booked for "+ticket.emailUser);
41         System.out.println();
42         return "Ticket Boked for "+ticket.emailUser+" with seat number: "+ticket.seatNumber;
43     }
44 }
45
46 public class CallableFutureApp {
47     public static void main(String[] args) {
48
49         System.out.println("Movie Ticket Booking App Started");
50
51         MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
52         MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
53         MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
54         MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
55
56

```

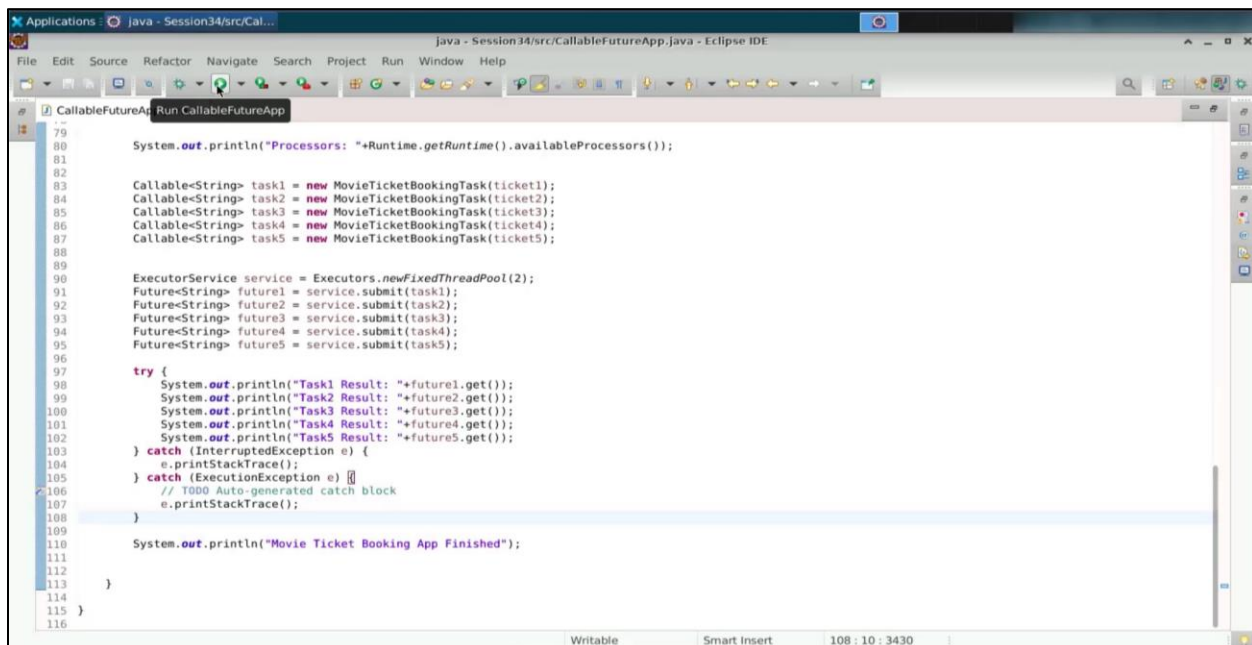
1.23 Here is how you can complete the entire process. So, what is next? Thus, the code is completed and next is where you will run the program and see the outputs coming in for the five different movie ticket booking sequentially.



The screenshot shows the Eclipse IDE with the file `CallableFutureApp.java` open. The code is as follows:

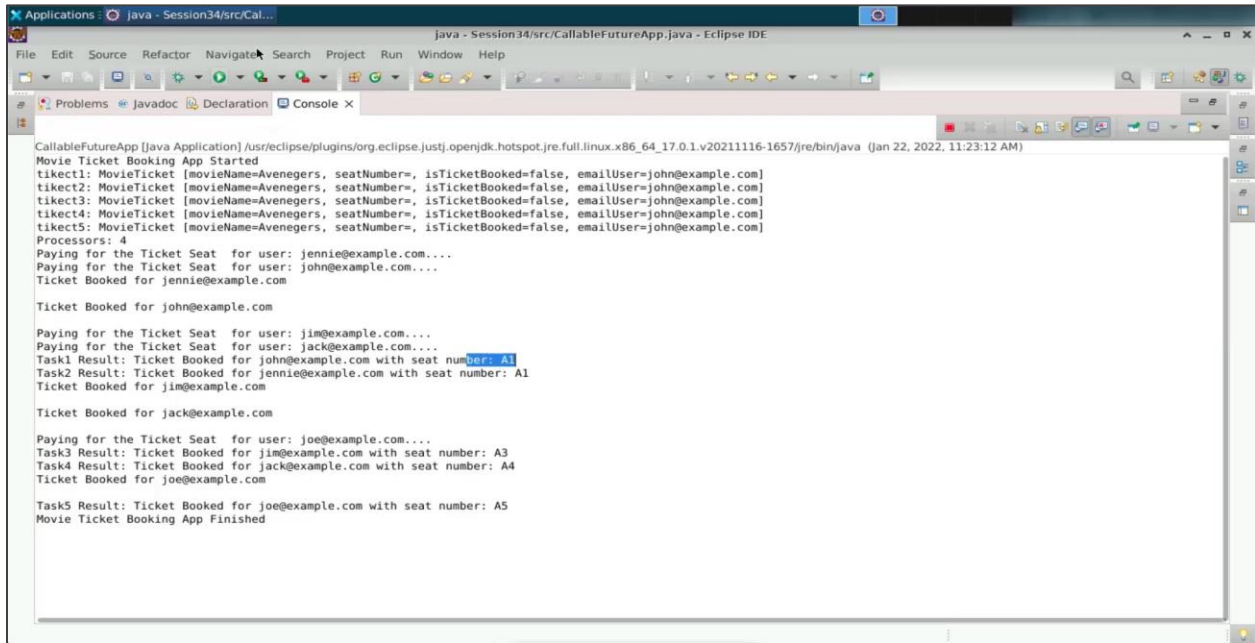
```
80 System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
81
82
83 Callable<String> task1 = new MovieTicketBookingTask(ticket1);
84 Callable<String> task2 = new MovieTicketBookingTask(ticket2);
85 Callable<String> task3 = new MovieTicketBookingTask(ticket3);
86 Callable<String> 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.



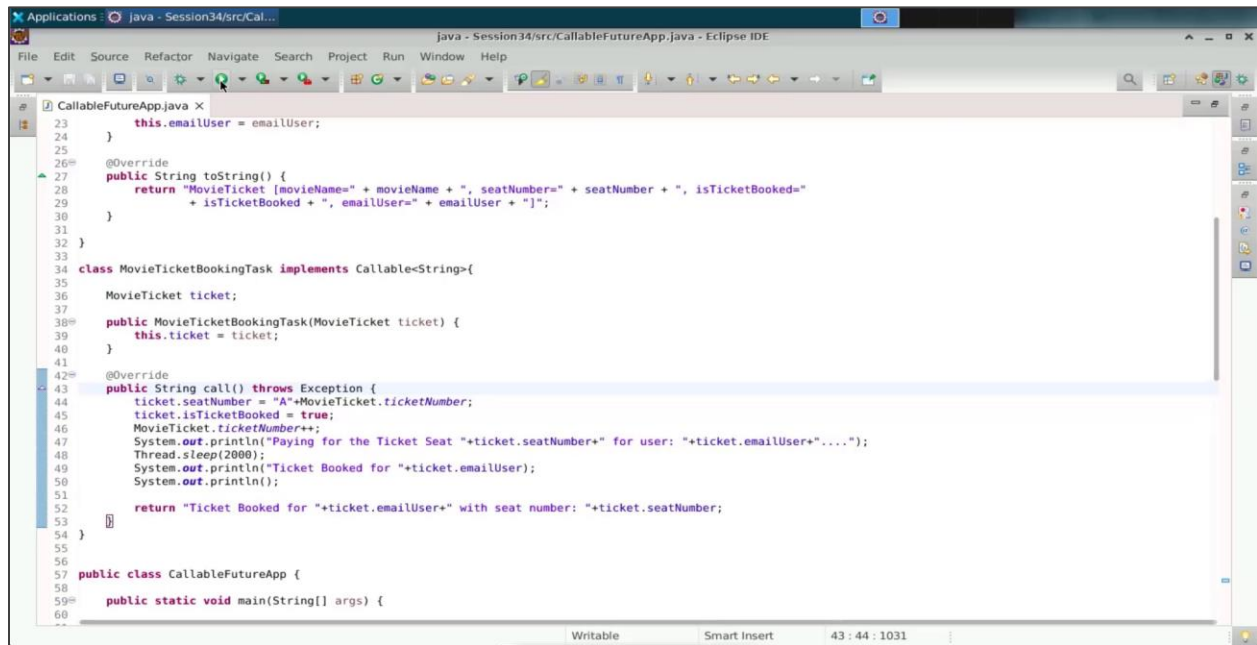
The screenshot shows the Eclipse IDE with the file `CallableFutureApp.java` open. The code is the same as in the previous screenshot. The green play button (Run) is highlighted in the toolbar. The status bar at the bottom shows "Writable", "Smart Insert", and "108 : 10 : 3430".

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.



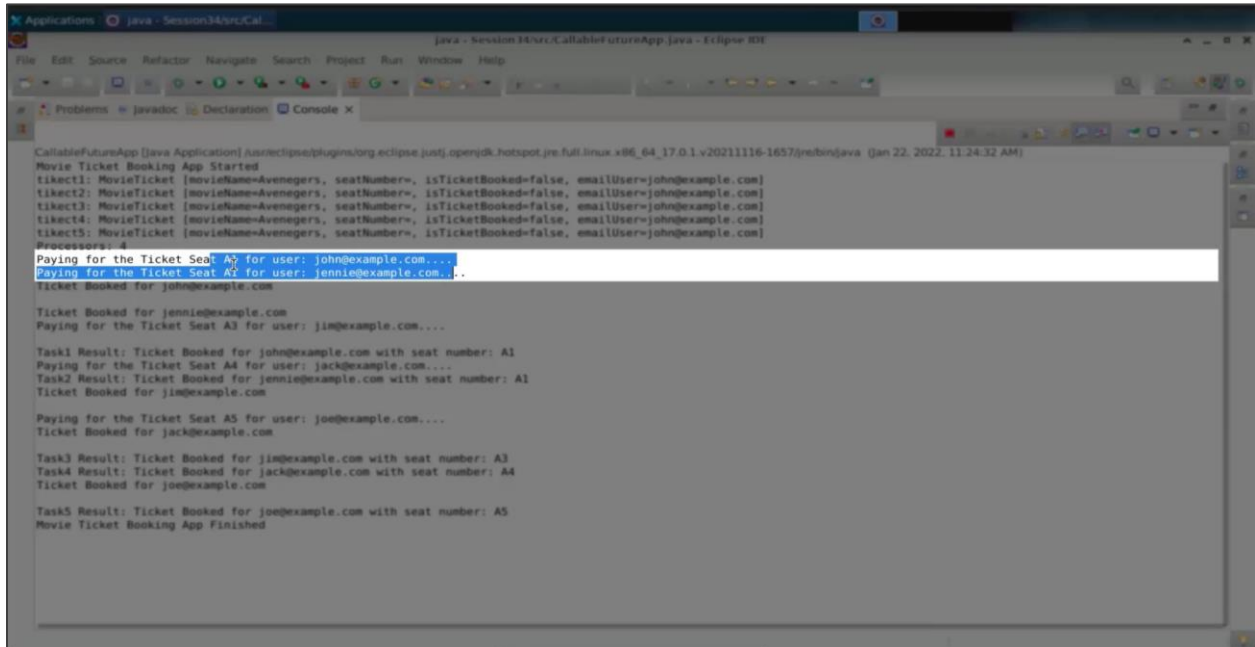
```
CallableFutureApp [Java Application] /usr/eclipse/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java (Jan 22, 2022, 11:23:12 AM)
Movie Ticket Booking App Started
ticket1: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
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.



```
1  CallableFutureApp.java
2
3  23      this.emailUser = emailUser;
4  24  }
5  25
6  26  @Override
7  27  public String toString() {
8  28      return "MovieTicket [movieName=" + movieName + ", seatNumber=" + seatNumber + ", isTicketBooked="
9  29      + isTicketBooked + ", emailUser=" + emailUser + "]";
10 30  }
11 31
12 32  }
13 33
14 34  class MovieTicketBookingTask implements Callable<String>{
15 35
16 36      MovieTicket ticket;
17 37
18 38      public MovieTicketBookingTask(MovieTicket ticket) {
19 39          this.ticket = ticket;
20 40      }
21 41
22 42  @Override
23 43  public String call() throws Exception {
24 44      ticket.seatNumber = "A"+MovieTicket.ticketNumber;
25 45      ticket.isTicketBooked = true;
26 46      MovieTicket.ticketNumber++;
27 47      System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser+"....");
28 48      Thread.sleep(2000);
29 49      System.out.println("Ticket Booked for "+ticket.emailUser);
30 50      System.out.println();
31 51
32 52      return "Ticket Booked for "+ticket.emailUser+" with seat number: "+ticket.seatNumber;
33 53  }
34 54  }
35 55
36 56  public class CallableFutureApp {
37 57
38 58      public static void main(String[] args) {
39 59
40 60
41 61
42 62
43 63
44 64
45 65
46 66
47 67
48 68
49 69
50 70
51 71
52 72
53 73
54 74
55 75
56 76
57 77
58 78
59 79
60 80
61 81
62 82
63 83
64 84
65 85
66 86
67 87
68 88
69 89
70 90
71 91
72 92
73 93
74 94
75 95
76 96
77 97
78 98
79 99
80 100
81 101
82 102
83 103
84 104
85 105
86 106
87 107
88 108
89 109
90 110
91 111
92 112
93 113
94 114
95 115
96 116
97 117
98 118
99 119
100 120
101 121
102 122
103 123
104 124
105 125
106 126
107 127
108 128
109 129
110 130
111 131
112 132
113 133
114 134
115 135
116 136
117 137
118 138
119 139
120 140
121 141
122 142
123 143
124 144
125 145
126 146
127 147
128 148
129 149
130 150
131 151
132 152
133 153
134 154
135 155
136 156
137 157
138 158
139 159
140 160
141 161
142 162
143 163
144 164
145 165
146 166
147 167
148 168
149 169
150 170
151 171
152 172
153 173
154 174
155 175
156 176
157 177
158 178
159 179
160 180
161 181
162 182
163 183
164 184
165 185
166 186
167 187
168 188
169 189
170 190
171 191
172 192
173 193
174 194
175 195
176 196
177 197
178 198
179 199
180 200
181 201
182 202
183 203
184 204
185 205
186 206
187 207
188 208
189 209
190 210
191 211
192 212
193 213
194 214
195 215
196 216
197 217
198 218
199 219
200 220
201 221
202 222
203 223
204 224
205 225
206 226
207 227
208 228
209 229
210 230
211 231
212 232
213 233
214 234
215 235
216 236
217 237
218 238
219 239
220 240
221 241
222 242
223 243
224 244
225 245
226 246
227 247
228 248
229 249
230 250
231 251
232 252
233 253
234 254
235 255
236 256
237 257
238 258
239 259
240 260
241 261
242 262
243 263
244 264
245 265
246 266
247 267
248 268
249 269
250 270
251 271
252 272
253 273
254 274
255 275
256 276
257 277
258 278
259 279
260 280
261 281
262 282
263 283
264 284
265 285
266 286
267 287
268 288
269 289
270 290
271 291
272 292
273 293
274 294
275 295
276 296
277 297
278 298
279 299
280 300
281 301
282 302
283 303
284 304
285 305
286 306
287 307
288 308
289 309
290 310
291 311
292 312
293 313
294 314
295 315
296 316
297 317
298 318
299 319
300 320
301 321
302 322
303 323
304 324
305 325
306 326
307 327
308 328
309 329
310 330
311 331
312 332
313 333
314 334
315 335
316 336
317 337
318 338
319 339
320 340
321 341
322 342
323 343
324 344
325 345
326 346
327 347
328 348
329 349
330 350
331 351
332 352
333 353
334 354
335 355
336 356
337 357
338 358
339 359
340 360
341 361
342 362
343 363
344 364
345 365
346 366
347 367
348 368
349 369
350 370
351 371
352 372
353 373
354 374
355 375
356 376
357 377
358 378
359 379
360 380
361 381
362 382
363 383
364 384
365 385
366 386
367 387
368 388
369 389
370 390
371 391
372 392
373 393
374 394
375 395
376 396
377 397
378 398
379 399
380 400
381 401
382 402
383 403
384 404
385 405
386 406
387 407
388 408
389 409
390 410
391 411
392 412
393 413
394 414
395 415
396 416
397 417
398 418
399 419
400 420
401 421
402 422
403 423
404 424
405 425
406 426
407 427
408 428
409 429
410 430
411 431
412 432
413 433
414 434
415 435
416 436
417 437
418 438
419 439
420 440
421 441
422 442
423 443
424 444
425 445
426 446
427 447
428 448
429 449
430 450
431 451
432 452
433 453
434 454
435 455
436 456
437 457
438 458
439 459
440 460
441 461
442 462
443 463
444 464
445 465
446 466
447 467
448 468
449 469
450 470
451 471
452 472
453 473
454 474
455 475
456 476
457 477
458 478
459 479
460 480
461 481
462 482
463 483
464 484
465 485
466 486
467 487
468 488
469 489
470 490
471 491
472 492
473 493
474 494
475 495
476 496
477 497
478 498
479 499
480 500
481 501
482 502
483 503
484 504
485 505
486 506
487 507
488 508
489 509
490 510
491 511
492 512
493 513
494 514
495 515
496 516
497 517
498 518
499 519
500 520
501 521
502 522
503 523
504 524
505 525
506 526
507 527
508 528
509 529
510 530
511 531
512 532
513 533
514 534
515 535
516 536
517 537
518 538
519 539
520 540
521 541
522 542
523 543
524 544
525 545
526 546
527 547
528 548
529 549
530 550
531 551
532 552
533 553
534 554
535 555
536 556
537 557
538 558
539 559
540 560
541 561
542 562
543 563
544 564
545 565
546 566
547 567
548 568
549 569
550 570
551 571
552 572
553 573
554 574
555 575
556 576
557 577
558 578
559 579
560 580
561 581
562 582
563 583
564 584
565 585
566 586
567 587
568 588
569 589
570 590
571 591
572 592
573 593
574 594
575 595
576 596
577 597
578 598
579 599
580 600
581 601
582 602
583 603
584 604
585 605
586 606
587 607
588 608
589 609
590 610
591 611
592 612
593 613
594 614
595 615
596 616
597 617
598 618
599 619
600 620
601 621
602 622
603 623
604 624
605 625
606 626
607 627
608 628
609 629
610 630
611 631
612 632
613 633
614 634
615 635
616 636
617 637
618 638
619 639
620 640
621 641
622 642
623 643
624 644
625 645
626 646
627 647
628 648
629 649
630 650
631 651
632 652
633 653
634 654
635 655
636 656
637 657
638 658
639 659
640 660
641 661
642 662
643 663
644 664
645 665
646 666
647 667
648 668
649 669
650 670
651 671
652 672
653 673
654 674
655 675
656 676
657 677
658 678
659 679
660 680
661 681
662 682
663 683
664 684
665 685
666 686
667 687
668 688
669 689
670 690
671 691
672 692
673 693
674 694
675 695
676 696
677 697
678 698
679 699
680 700
681 701
682 702
683 703
684 704
685 705
686 706
687 707
688 708
689 709
690 710
691 711
692 712
693 713
694 714
695 715
696 716
697 717
698 718
699 719
700 720
701 721
702 722
703 723
704 724
705 725
706 726
707 727
708 728
709 729
710 730
711 731
712 732
713 733
714 734
715 735
716 736
717 737
718 738
719 739
720 740
721 741
722 742
723 743
724 744
725 745
726 746
727 747
728 748
729 749
730 750
731 751
732 752
733 753
734 754
735 755
736 756
737 757
738 758
739 759
740 760
741 761
742 762
743 763
744 764
745 765
746 766
747 767
748 768
749 769
750 770
751 771
752 772
753 773
754 774
755 775
756 776
757 777
758 778
759 779
760 780
761 781
762 782
763 783
764 784
765 785
766 786
767 787
768 788
769 789
770 790
771 791
772 792
773 793
774 794
775 795
776 796
777 797
778 798
779 799
780 800
781 801
782 802
783 803
784 804
785 805
786 806
787 807
788 808
789 809
790 810
791 811
792 812
793 813
794 814
795 815
796 816
797 817
798 818
799 819
800 820
801 821
802 822
803 823
804 824
805 825
806 826
807 827
808 828
809 829
810 830
811 831
812 832
813 833
814 834
815 835
816 836
817 837
818 838
819 839
820 840
821 841
822 842
823 843
824 844
825 845
826 846
827 847
828 848
829 849
830 850
831 851
832 852
833 853
834 854
835 855
836 856
837 857
838 858
839 859
840 860
841 861
842 862
843 863
844 864
845 865
846 866
847 867
848 868
849 869
850 870
851 871
852 872
853 873
854 874
855 875
856 876
857 877
858 878
859 879
860 880
861 881
862 882
863 883
864 884
865 885
866 886
867 887
868 888
869 889
870 890
871 891
872 892
873 893
874 894
875 895
876 896
877 897
878 898
879 899
880 900
881 901
882 902
883 903
884 904
885 905
886 906
887 907
888 908
889 909
890 910
891 911
892 912
893 913
894 914
895 915
896 916
897 917
898 918
899 919
900 920
901 921
902 922
903 923
904 924
905 925
906 926
907 927
908 928
909 929
910 930
911 931
912 932
913 933
914 934
915 935
916 936
917 937
918 938
919 939
920 940
921 941
922 942
923 943
924 944
925 945
926 946
927 947
928 948
929 949
930 950
931 951
932 952
933 953
934 954
935 955
936 956
937 957
938 958
939 959
940 960
941 961
942 962
943 963
944 964
945 965
946 966
947 967
948 968
949 969
950 970
951 971
952 972
953 973
954 974
955 975
956 976
957 977
958 978
959 979
960 980
961 981
962 982
963 983
964 984
965 985
966 986
967 987
968 988
969 989
970 990
971 991
972 992
973 993
974 994
975 995
976 996
977 997
978 998
979 999
1000 1000
```

- 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.



```
CallabieFutureApp [Java Application] java:Session34/src/CallabieFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console x

CallabieFutureApp [Java Application] java:Session34/src/CallabieFutureApp.java - Eclipse IDE (Jan 22, 2022, 11:24:32 AM)
Movie Ticket Booking App Started
ticket1: MovieTicket [movieName=Avengers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
ticket2: MovieTicket [movieName=Avengers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
ticket3: MovieTicket [movieName=Avengers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
ticket4: MovieTicket [movieName=Avengers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
ticket5: MovieTicket [movieName=Avengers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
Processors: 4
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....
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.

```

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         return "Ticket Booked for "+ticket.emailUser+" with seat number: "+ticket.seatNumber;
52     }
53 }
54
55
56 public class CallableFutureApp {
57
58     public static void main(String[] args) {
59
60
61         System.out.println("Movie Ticket Booking App Started");
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, "jim@example.com");
67         MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
68         MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");
69
70

```

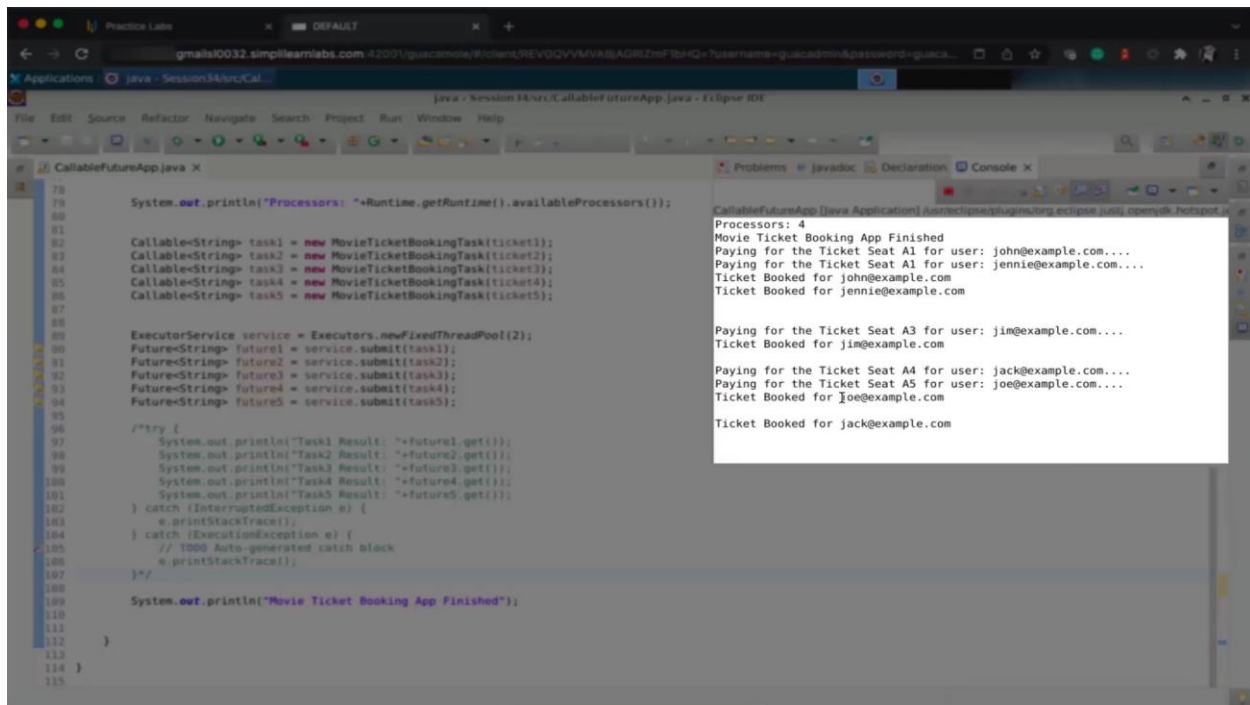
1.29 Let us try commenting out the code with get method. Now, without this code snippet, run the code.

```

78
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     System.out.println("Movie Ticket Booking App Finished");
110
111
112 }
113
114
115

```


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.



```

178
179 System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());
180
181 Callable<String> task1 = new MovieTicketBookingTask(ticket1);
182 Callable<String> task2 = new MovieTicketBookingTask(ticket2);
183 Callable<String> task3 = new MovieTicketBookingTask(ticket3);
184 Callable<String> task4 = new MovieTicketBookingTask(ticket4);
185 Callable<String> task5 = new MovieTicketBookingTask(ticket5);
186
187
188 ExecutorService service = Executors.newFixedThreadPool(2);
189 Future<String> future1 = service.submit(task1);
190 Future<String> future2 = service.submit(task2);
191 Future<String> future3 = service.submit(task3);
192 Future<String> future4 = service.submit(task4);
193 Future<String> future5 = service.submit(task5);
194
195
196 /**try {
197     System.out.println("Task1 Result: "+future1.get());
198     System.out.println("Task2 Result: "+future2.get());
199     System.out.println("Task3 Result: "+future3.get());
200     System.out.println("Task4 Result: "+future4.get());
201     System.out.println("Task5 Result: "+future5.get());
202 } catch (InterruptedException e) {
203     e.printStackTrace();
204 } catch (ExecutionException e) {
205     // TODO Auto-generated catch block
206     e.printStackTrace();
207 }*/
208
209 System.out.println("Movie Ticket Booking App Finished");
210
211
212 }
213
214 }
215

```

```

CallableFutureApp [Java Application] Sourcecodepro>git>git push origin master
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 joe@example.com

Ticket Booked for jack@example.com

```

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