Lesson 04 Demo 11

Implementing Callable and Future

Objective: To demonstrate the usage of callable interface and futures

Tools required: Eclipse IDE

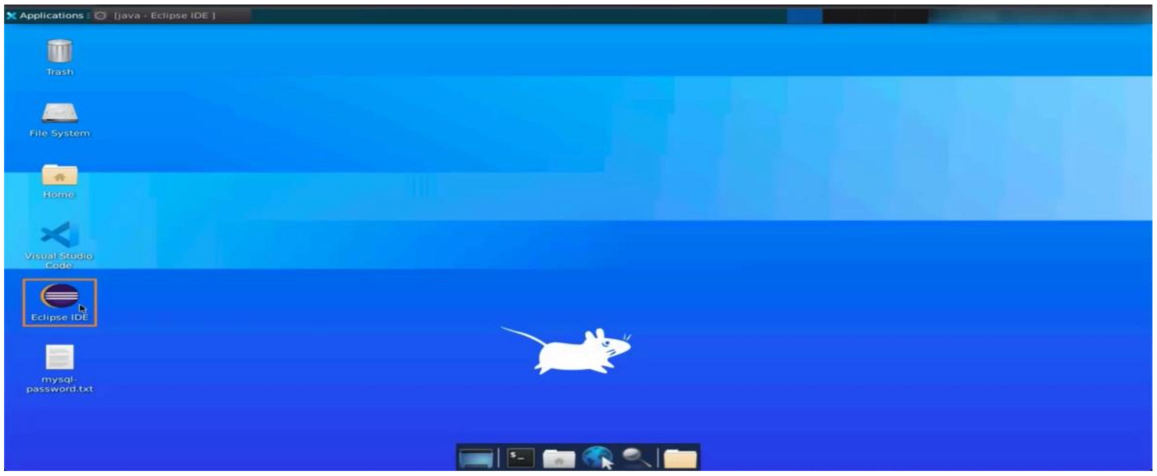
Prerequisites: None

Steps to be followed:

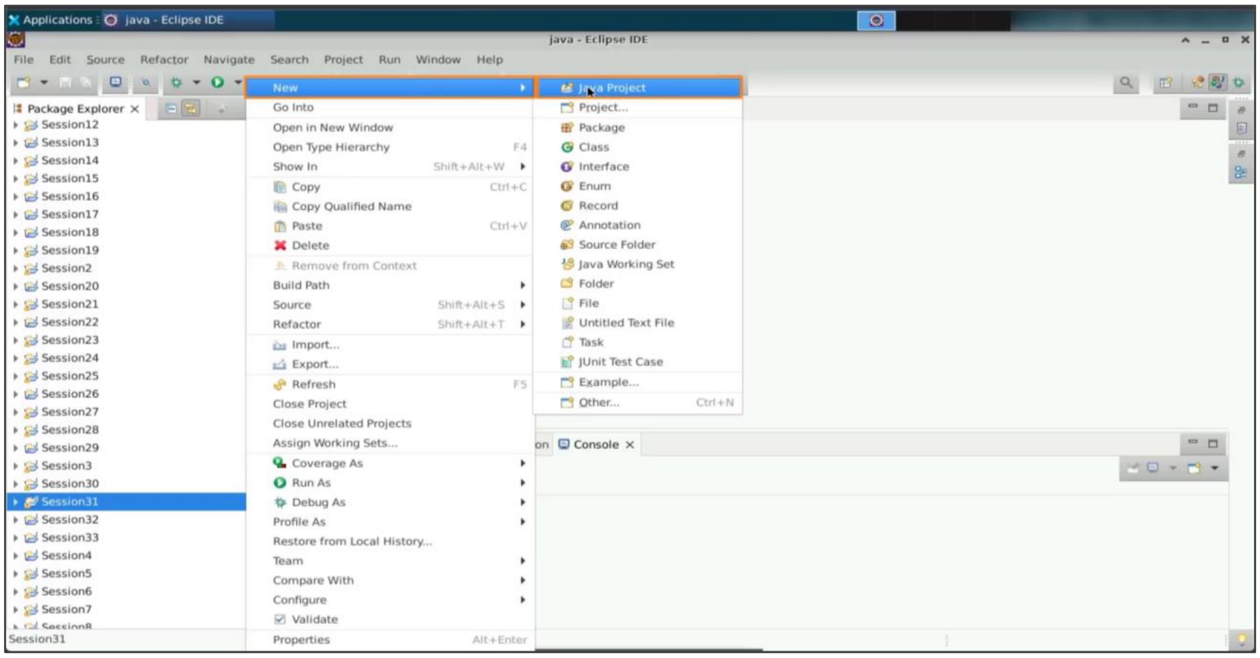
1. Implement a thread using Callable and Future Interfaces along with the suitable scenarios

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

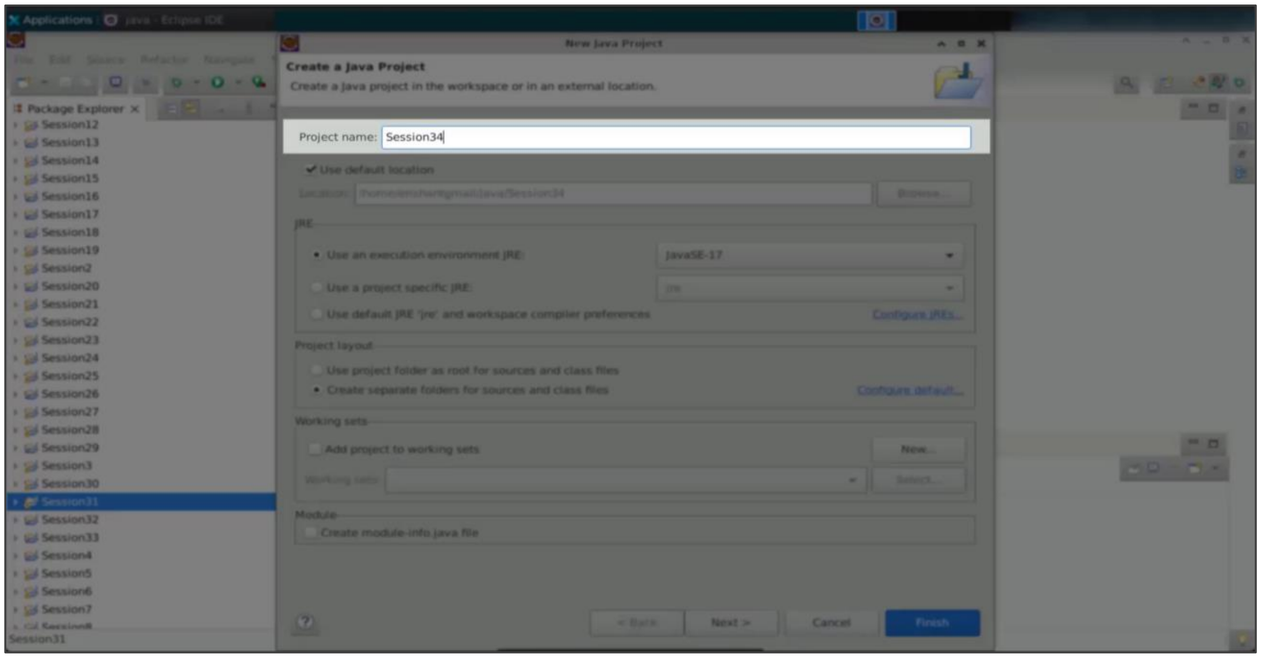
1.1 If you wish to create a thread, you can either extend the `Thread` class or implement the `Runnable` interface. When you start the thread, it runs asynchronously, and you will not be notified of its termination. However, if you need to capture results from a thread, Java provides the `Callable` interface, 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.**



Project name: Session34

☑ Use default location

Location: /home/.../java/Session34

● Use an execution environment JRE: JavaSE-17

○ Use a project specific JRE:

○ Use default JRE 'jre' and workspace compiler preferences

○ Use project folder as root for sources and class files

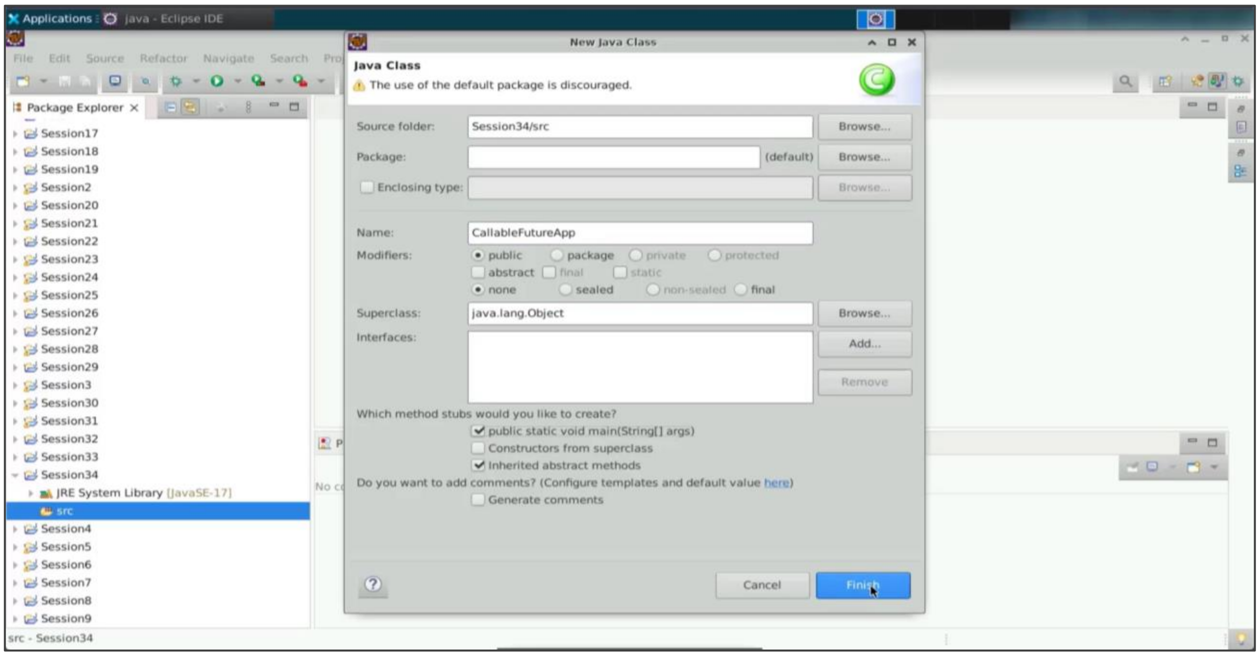
● Create separate folders for sources and class files

☐ Add project to working sets

Working sets:

☐ Create module-info.java file

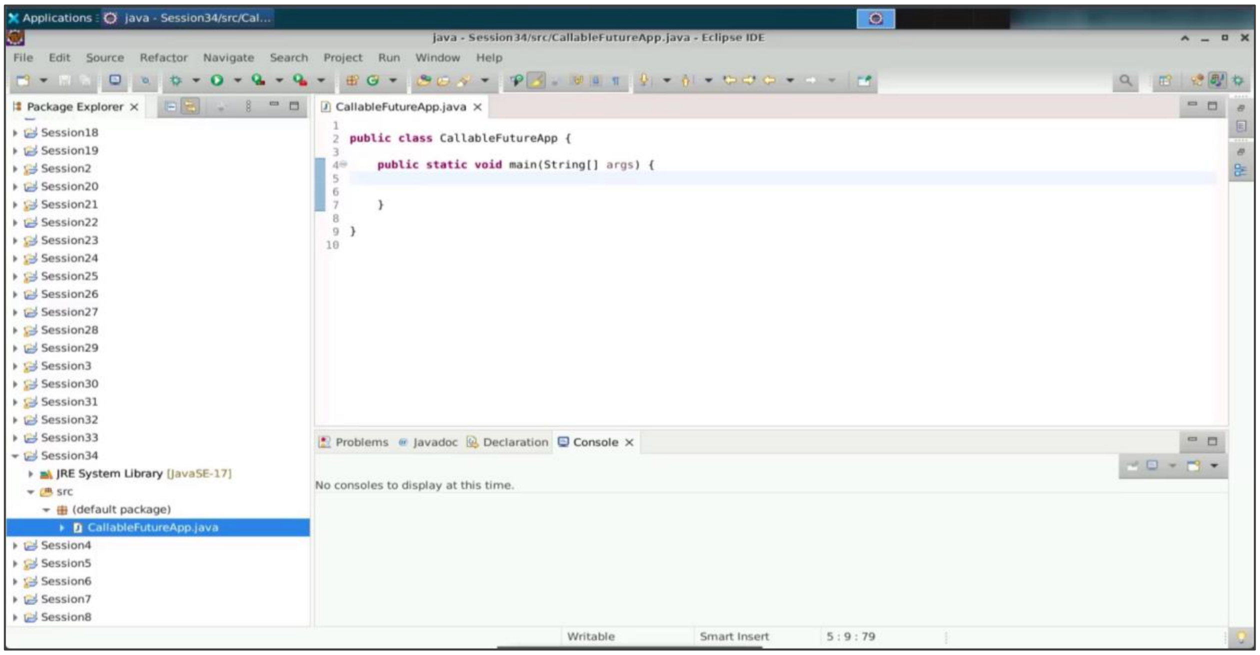
1.4 In the source, do a right click and create a new class and name the class as **CallableFutureApp** along with the main method. Click **Finish.**



Java Class
The use of the default package is discouraged.

Source folder: Session34/src
Package: (default)
Enclosing type: ☐
Name: CallableFutureApp
Modifiers: [✓ public] package private protected
☐ abstract ☐ final ☐ static
[✓ none] sealed non-sealed final
Superclass: java.lang.Object
Interfaces:
Which method stubs would you like to create?
☑ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
☐ Generate comments

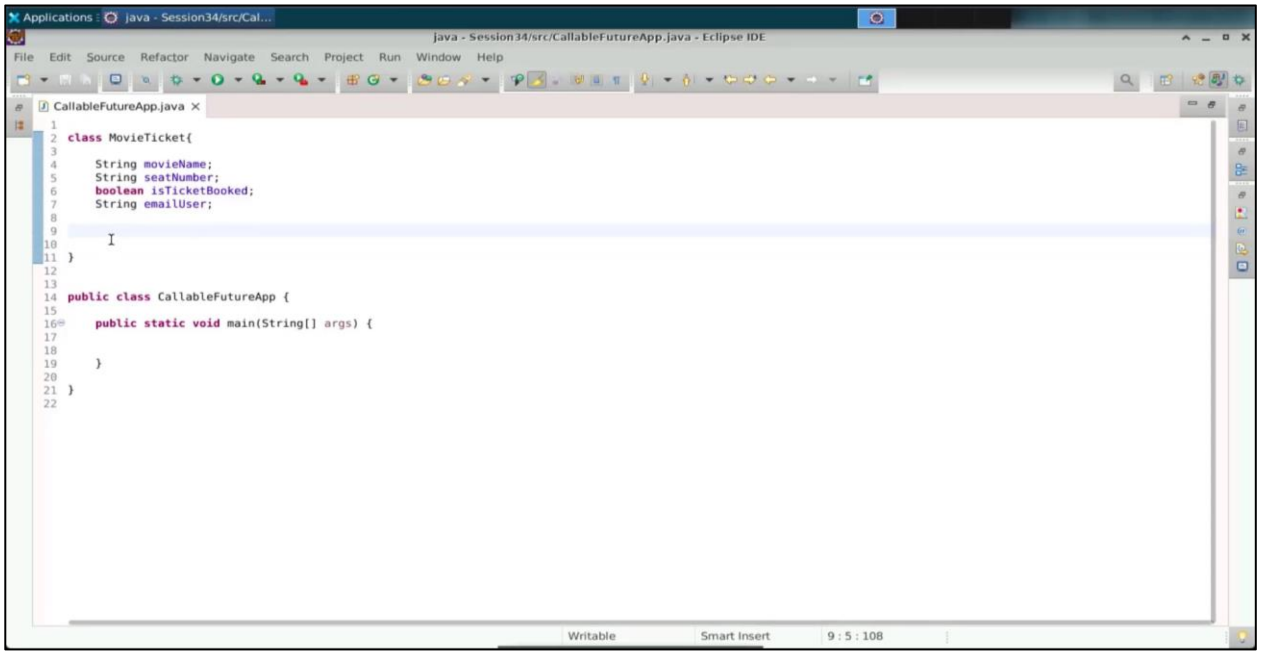
Note: For writing this demonstration, you will learn about the executor service in Java, whose key role is to execute the tasks on the threads asynchronously. When you will create your threads with callable, then the executable service will submit the task with the help of which you can get some results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



```
public class CallableFutureApp {

    public static void main(String[] args) {


    }

}
```

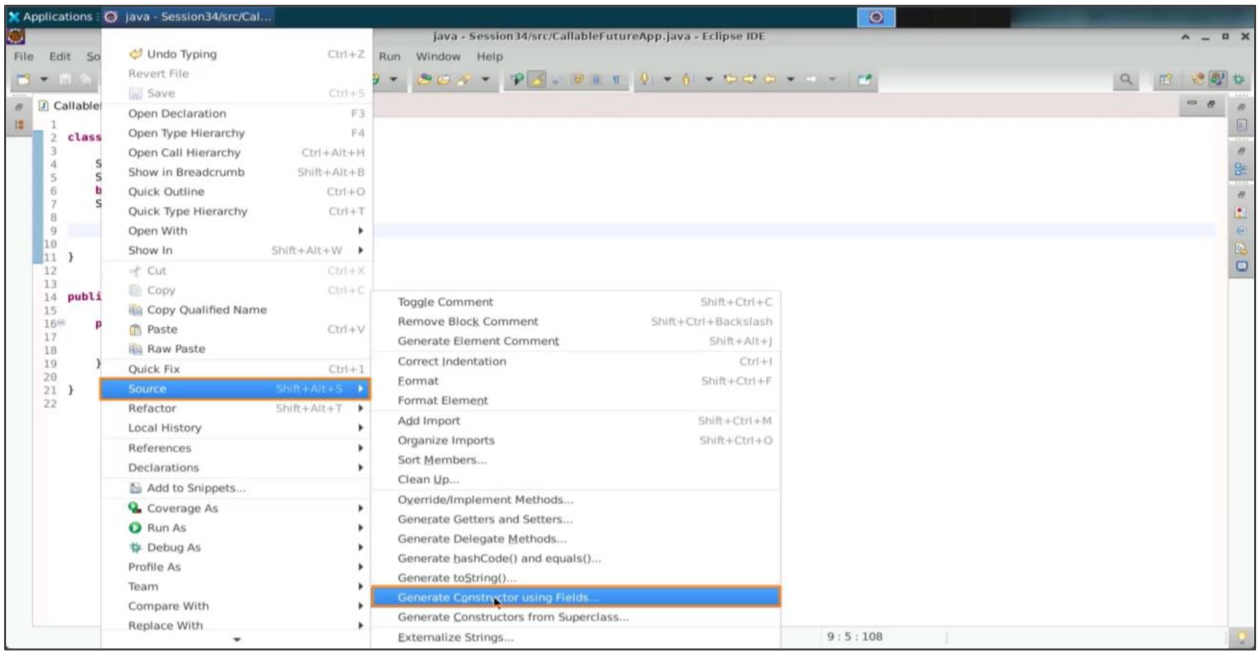
```
No consoles to display at this time.
```

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.

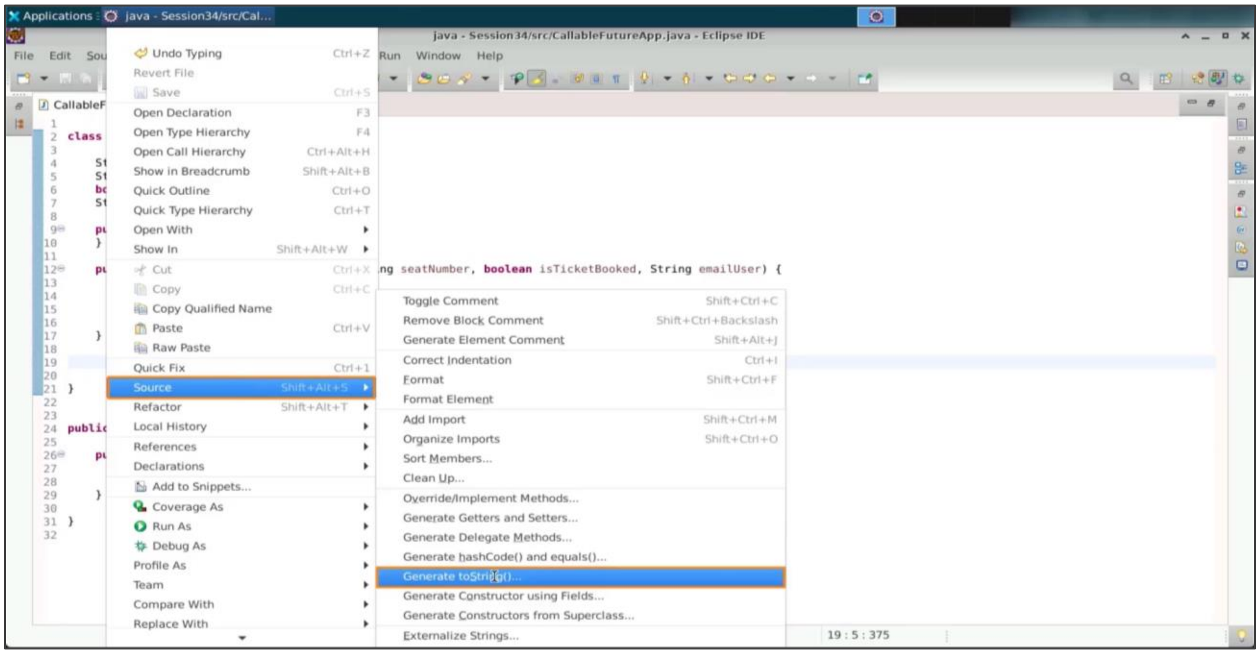

java - Session34/src/CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
CallableFutureApp.java
Writable Smart Insert 9 : 5 : 108

```
class MovieTicket{

    String movieName;
    String seatNumber;
    boolean isTicketBooked;
    String emailUser;


}


public class CallableFutureApp {

    public static void main(String[] args) {


    }

}
```

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

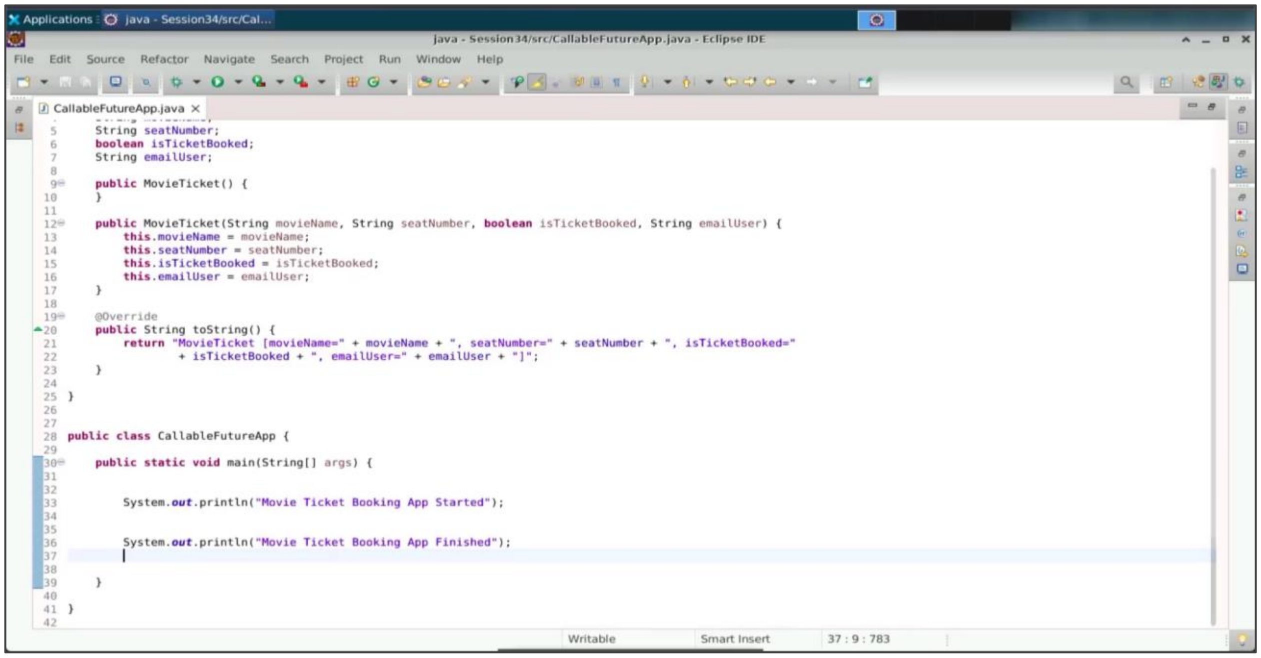


1.7 Let's remove this super part and write a default constructor, which will be used to create a regular movie ticket object. To see the details in the ticket, you can generate a **toString** method through which you will know what the data inside this movie ticket is.



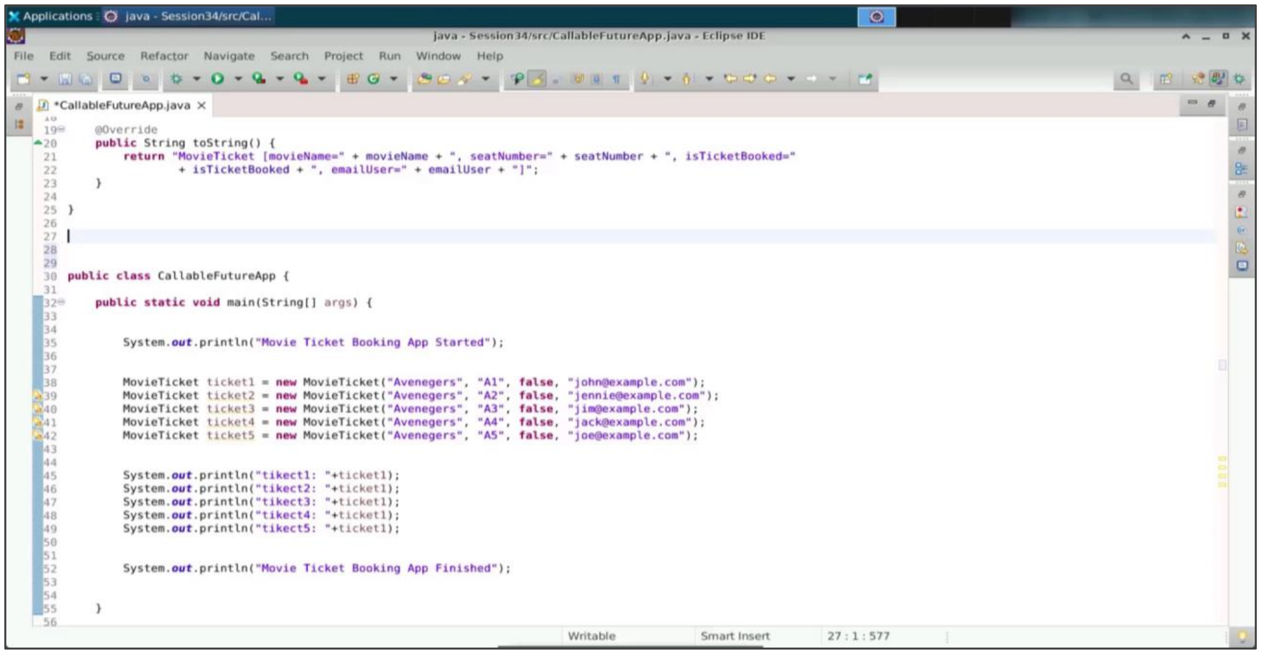
```
ng seatNumber, boolean isTicketBooked, String emailUser) {
```

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.


Applications
java - Session34/src/Cal...
java - Session34/src/CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
CallableFutureApp.java
Writable
Smart Insert
37 : 9 : 783

```java
    String seatNumber;
    boolean isTicketBooked;
    String emailUser;

    public MovieTicket() {
    }

    public MovieTicket(String movieName, String seatNumber, boolean isTicketBooked, String emailUser) {
        this.movieName = movieName;
        this.seatNumber = seatNumber;
        this.isTicketBooked = isTicketBooked;
        this.emailUser = emailUser;
    }

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

}


public class CallableFutureApp {

    public static void main(String[] args) {


        System.out.println("Movie Ticket Booking App Started");


        System.out.println("Movie Ticket Booking App Finished");


    }

}
```

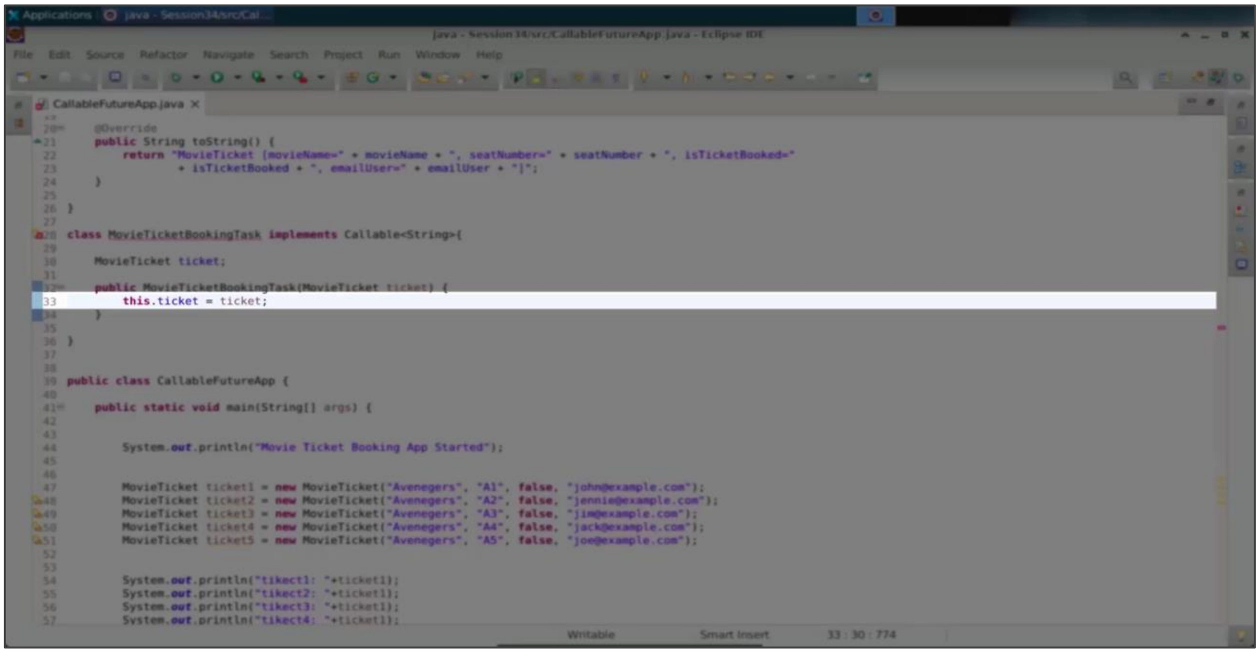
1.9 Create five movie tickets for "Avengers" with seat numbers A1 to A5 and set "Is the Ticket Booked" to false. Users: John, Jenni, Jim, Jack, and Joe (e.g., john@example.com). Print the tickets to see details. Use `Callable` for the ticket booking thread to return a response when the thread terminates.



Applications
java - Session34/src/Cal...
java - Session34/src/CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*CallableFutureApp.java
Writable
Smart Insert
27 : 1 : 577

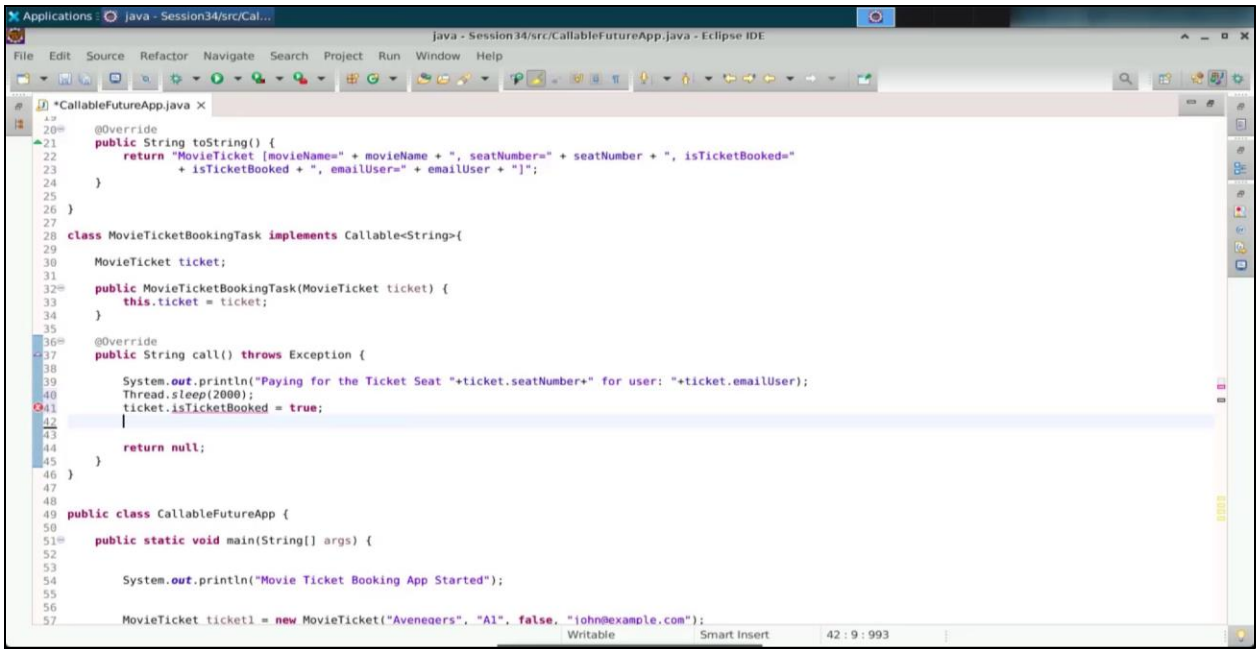
```java
    @Override
    public String toString() {
        return "MovieTicket [movieName=" + movieName + ", seatNumber=" + seatNumber + ", isTicketBooked="
                + isTicketBooked + ", emailUser=" + emailUser + "]";
    }

}



public class CallableFutureApp {

    public static void main(String[] args) {


        System.out.println("Movie Ticket Booking App Started");


        MovieTicket ticket1 = new MovieTicket("Avenegers", "A1", false, "john@example.com");
        MovieTicket ticket2 = new MovieTicket("Avenegers", "A2", false, "jennie@example.com");
        MovieTicket ticket3 = new MovieTicket("Avenegers", "A3", false, "jim@example.com");
        MovieTicket ticket4 = new MovieTicket("Avenegers", "A4", false, "jack@example.com");
        MovieTicket ticket5 = new MovieTicket("Avenegers", "A5", false, "joe@example.com");


        System.out.println("tikect1: "+ticket1);
        System.out.println("tikect2: "+ticket1);
        System.out.println("tikect3: "+ticket1);
        System.out.println("tikect4: "+ticket1);
        System.out.println("tikect5: "+ticket1);


        System.out.println("Movie Ticket Booking App Finished");


    }

```

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.


Applications
java - Session34/src/Cal...
java - Session34/src/CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
CallableFutureApp.java
Writable
Smart Insert
33 : 30 : 774

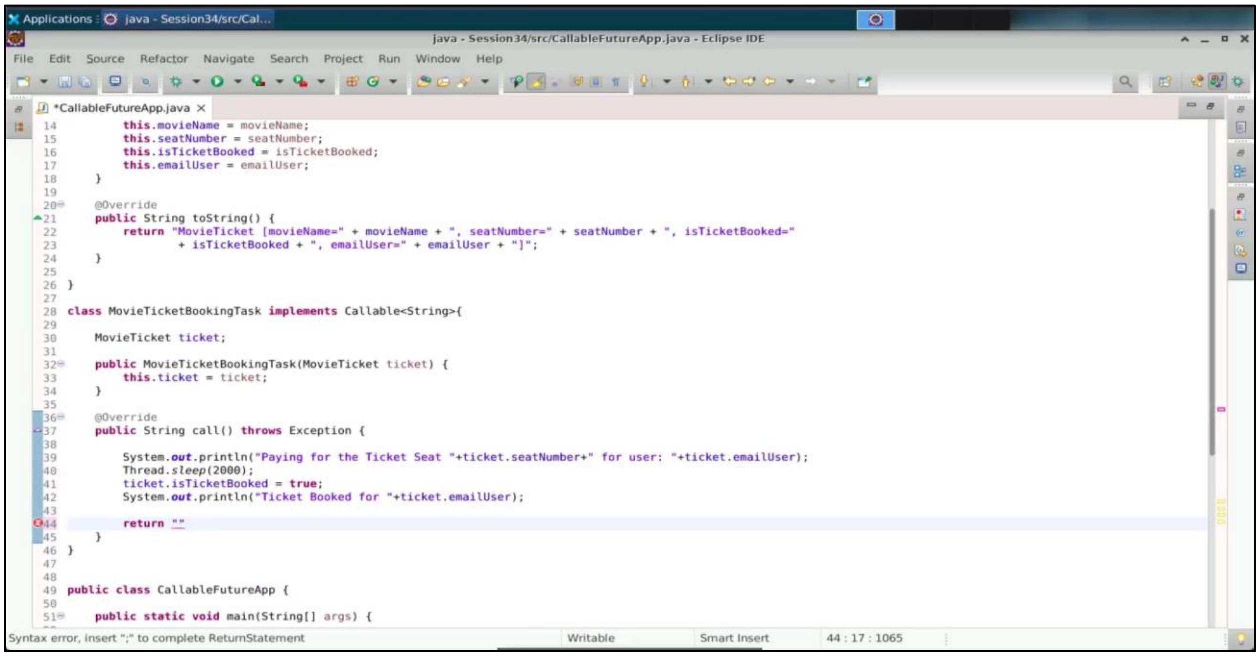
```java
    @Override
    public String toString() {
        return "MovieTicket [movieName=" + movieName + ", seatNumber=" + seatNumber + ", isTicketBooked="
                + isTicketBooked + ", emailUser=" + emailUser + "]";
    }

}

class MovieTicketBookingTask implements Callable<String>{

    MovieTicket ticket;

    public MovieTicketBookingTask(MovieTicket ticket) {
        this.ticket = ticket;
    }

}


public class CallableFutureApp {

    public static void main(String[] args) {


        System.out.println("Movie Ticket Booking App Started");


        MovieTicket ticket1 = new MovieTicket("Avenegers", "A1", false, "john@example.com");
        MovieTicket ticket2 = new MovieTicket("Avenegers", "A2", false, "jennie@example.com");
        MovieTicket ticket3 = new MovieTicket("Avenegers", "A3", false, "jim@example.com");
        MovieTicket ticket4 = new MovieTicket("Avenegers", "A4", false, "jack@example.com");
        MovieTicket ticket5 = new MovieTicket("Avenegers", "A5", false, "joe@example.com");


        System.out.println("tikect1: "+ticket1);
        System.out.println("tikect2: "+ticket1);
        System.out.println("tikect3: "+ticket1);
        System.out.println("tikect4: "+ticket1);
```

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.



```
    @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("Aveneqers", "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.



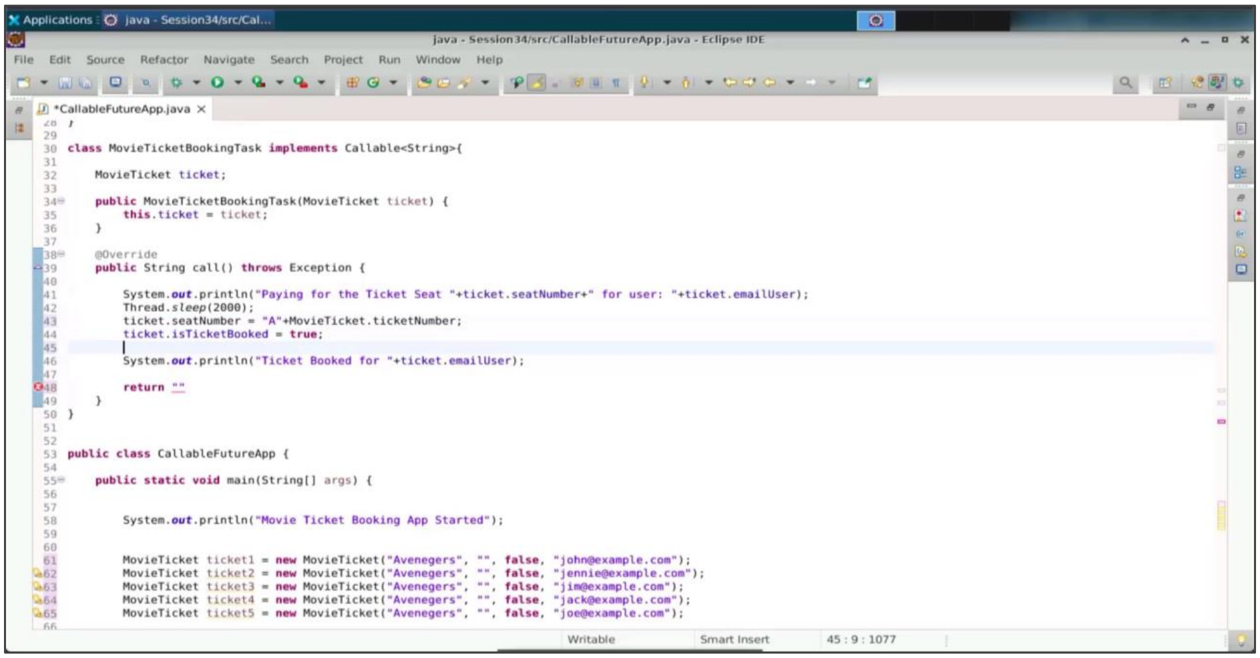
java - Session34/src/CallableFutureApp.java - Eclipse IDE

*CallableFutureApp.java

```java
        this.movieName = movieName;
        this.seatNumber = seatNumber;
        this.isTicketBooked = isTicketBooked;
        this.emailUser = emailUser;
    }

    @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;
        System.out.println("Ticket Booked for "+ticket.emailUser);

        return ""
    }
}


public class CallableFutureApp {

    public static void main(String[] args) {
```

```
Syntax error, insert ";" to complete ReturnStatement    Writable    Smart Insert    44 : 17 : 1065
```

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.



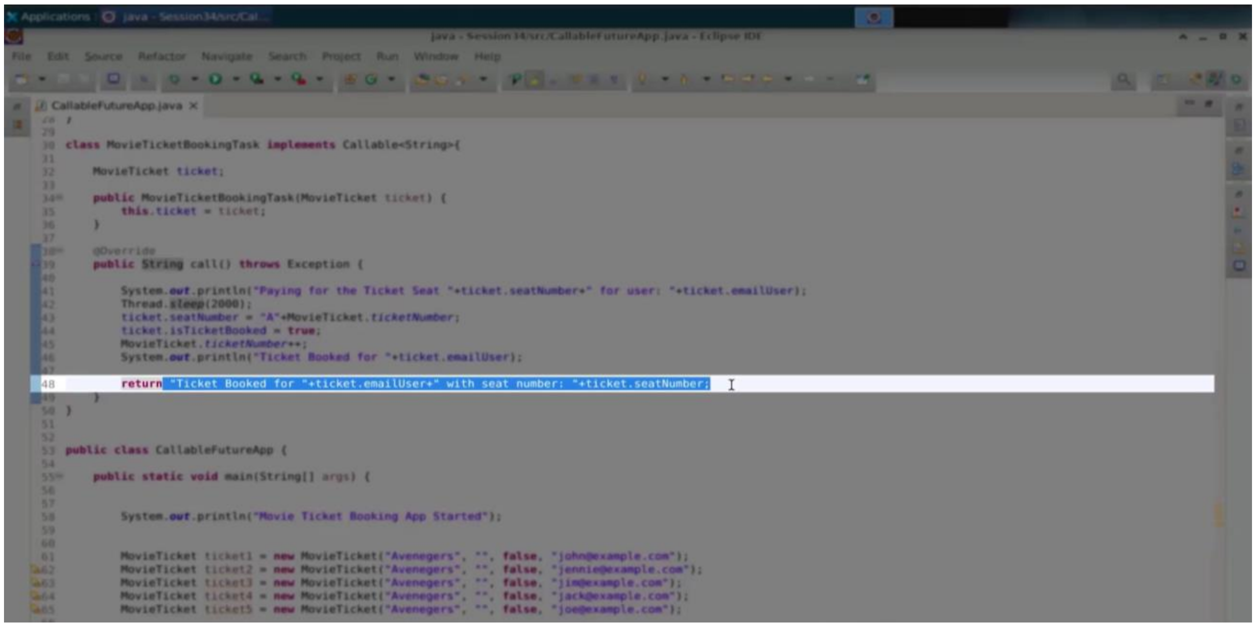
java - Session34/src/CallableFutureApp.java - Eclipse IDE

*CallableFutureApp.java

```java
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.seatNumber = "A"+MovieTicket.ticketNumber;
        ticket.isTicketBooked = true;

        System.out.println("Ticket Booked for "+ticket.emailUser);

        return ""
    }
}


public class CallableFutureApp {

    public static void main(String[] args) {


        System.out.println("Movie Ticket Booking App Started");


        MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
        MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
        MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
        MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
        MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");
```

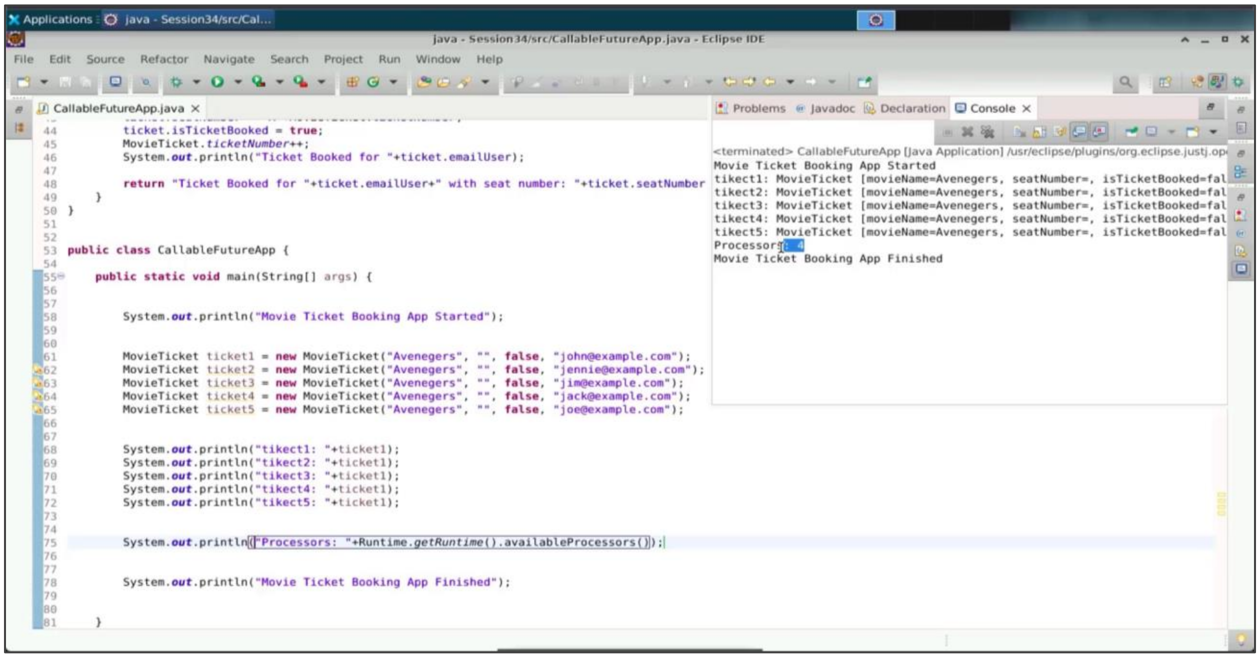
```
Writable    Smart Insert    45 : 9 : 1077
```

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.



```
}

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.seatNumber = "A"+MovieTicket.ticketNumber;
        ticket.isTicketBooked = true;
        MovieTicket.ticketNumber++;
        System.out.println("Ticket Booked for "+ticket.emailUser);

        return "Ticket Booked for "+ticket.emailUser+" with seat number: "+ticket.seatNumber;
    }
}


public class CallableFutureApp {

    public static void main(String[] args) {


        System.out.println("Movie Ticket Booking App Started");


        MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
        MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
        MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
        MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
        MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");
```

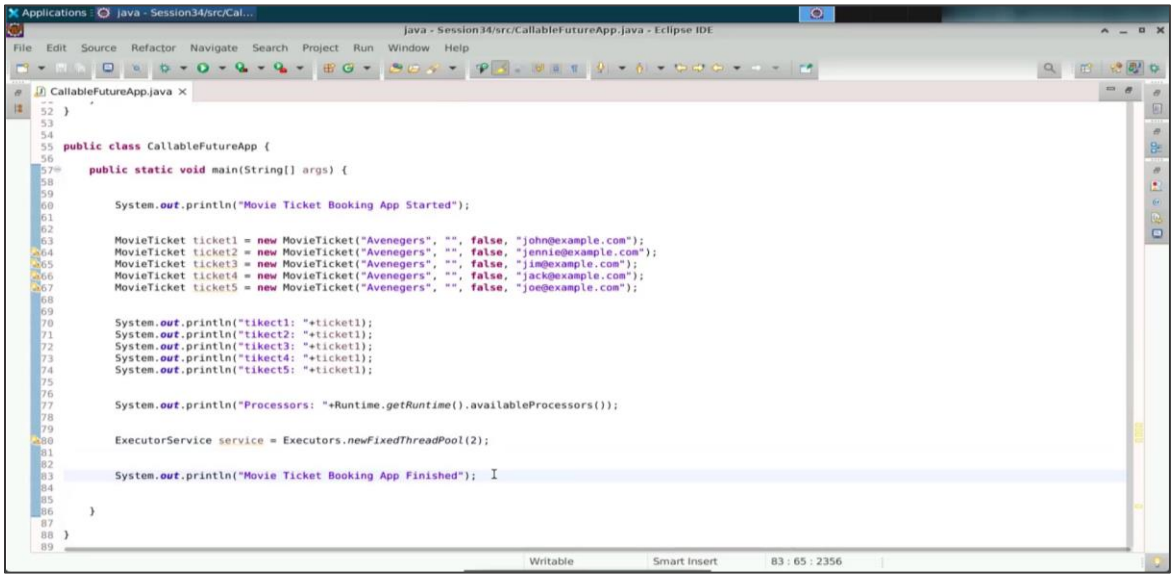
1.15 Let's navigate to the main method to check the movie ticket booking execution. Print the available processors using the runtime; there are four, indicating a quad-core system. Create a thread pool using two cores to run threads. To reduce CPU load, use Java's executor service, which runs threads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


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

```java
        ticket.isTicketBooked = true;
        MovieTicket.ticketNumber++;
        System.out.println("Ticket Booked for "+ticket.emailUser);

        return "Ticket Booked for "+ticket.emailUser+" with seat number: "+ticket.seatNumber
    }
}


public class CallableFutureApp {

    public static void main(String[] args) {


        System.out.println("Movie Ticket Booking App Started");


        MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
        MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
        MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
        MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
        MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");


        System.out.println("tikect1: "+ticket1);
        System.out.println("tikect2: "+ticket1);
        System.out.println("tikect3: "+ticket1);
        System.out.println("tikect4: "+ticket1);
        System.out.println("tikect5: "+ticket1);


        System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());


        System.out.println("Movie Ticket Booking App Finished");


    }
```

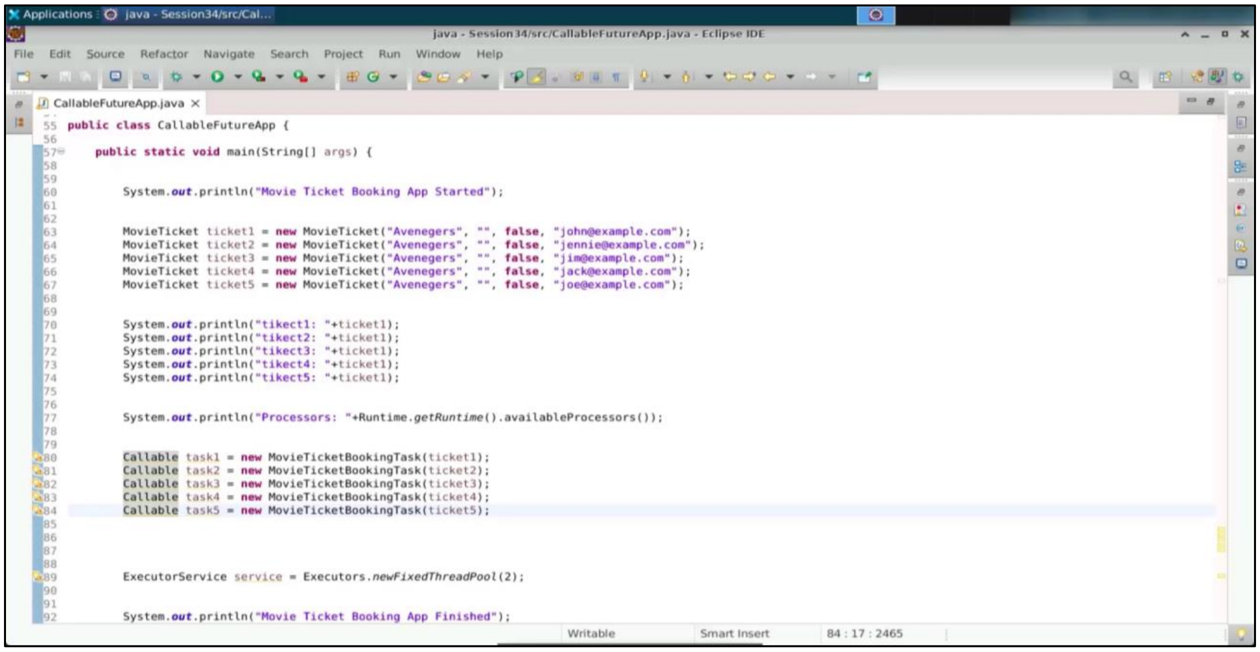
```
<terminated> CallableFutureApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.op
Movie Ticket Booking App Started
tikect1: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
tikect2: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
tikect3: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
tikect4: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
tikect5: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=fal
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.


Applications
java - Session34/src/Cal...
java - Session34/src/CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
CallableFutureApp.java
Writable
Smart Insert
83 : 65 : 2356

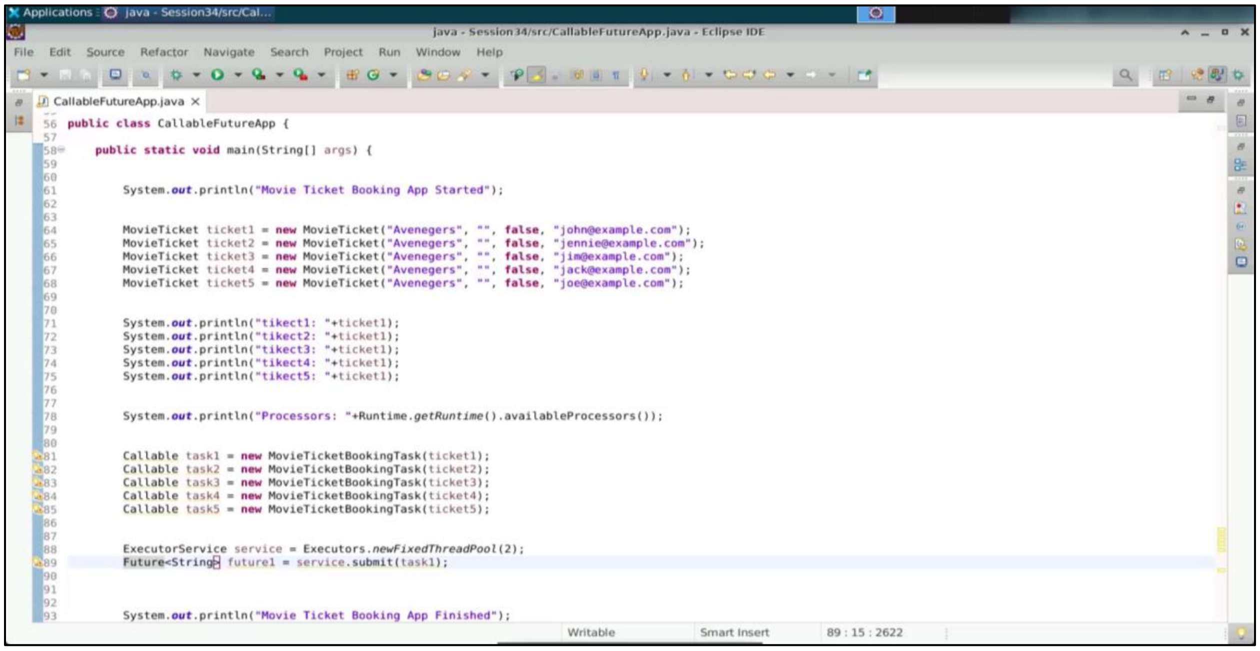
```java
}


public class CallableFutureApp {

    public static void main(String[] args) {


        System.out.println("Movie Ticket Booking App Started");


        MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
        MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
        MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
        MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
        MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");


        System.out.println("tikect1: "+ticket1);
        System.out.println("tikect2: "+ticket1);
        System.out.println("tikect3: "+ticket1);
        System.out.println("tikect4: "+ticket1);
        System.out.println("tikect5: "+ticket1);


        System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());


        ExecutorService service = Executors.newFixedThreadPool(2);


        System.out.println("Movie Ticket Booking App Finished");


    }

}
```

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.


Applications : java - Session34/src/Cal...
java - Session34/src/CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
CallableFutureApp.java
Writable
Smart Insert
84 : 17 : 2465

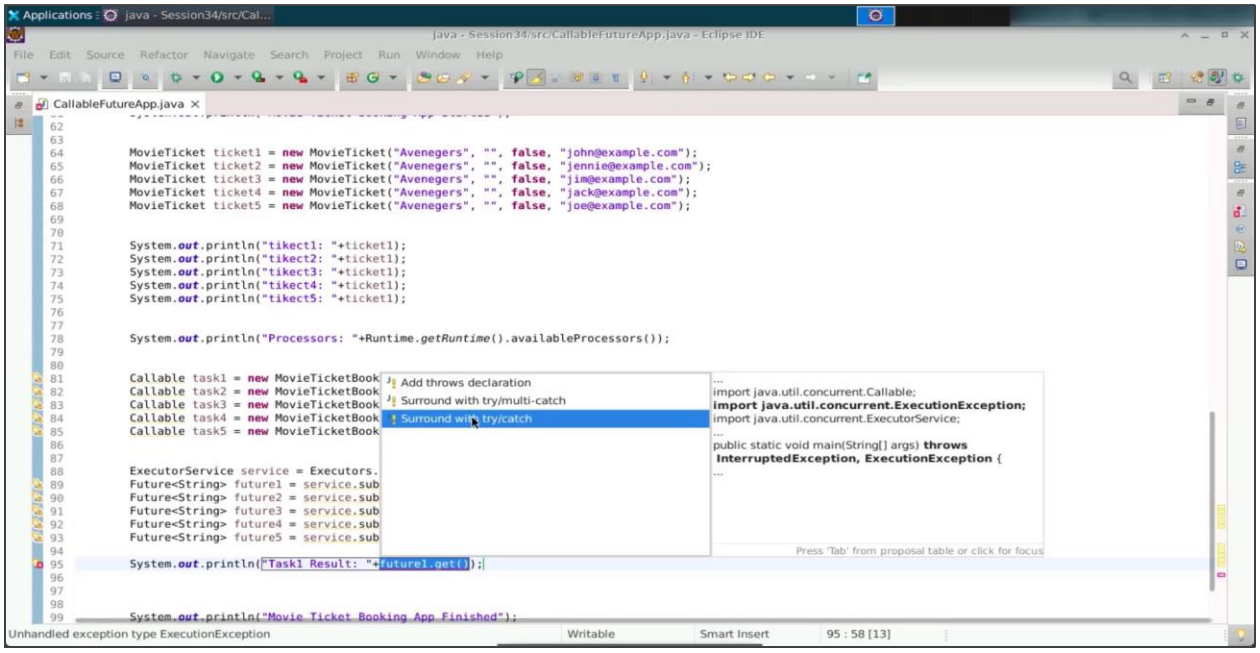
```java
public class CallableFutureApp {

    public static void main(String[] args) {


        System.out.println("Movie Ticket Booking App Started");


        MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
        MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
        MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
        MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
        MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");


        System.out.println("tikect1: "+ticket1);
        System.out.println("tikect2: "+ticket1);
        System.out.println("tikect3: "+ticket1);
        System.out.println("tikect4: "+ticket1);
        System.out.println("tikect5: "+ticket1);


        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);


        System.out.println("Movie Ticket Booking App Finished");
```

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.



```
public class CallableFutureApp {

    public static void main(String[] args) {


        System.out.println("Movie Ticket Booking App Started");


        MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
        MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
        MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
        MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
        MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");


        System.out.println("tikect1: "+ticket1);
        System.out.println("tikect2: "+ticket1);
        System.out.println("tikect3: "+ticket1);
        System.out.println("tikect4: "+ticket1);
        System.out.println("tikect5: "+ticket1);


        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.

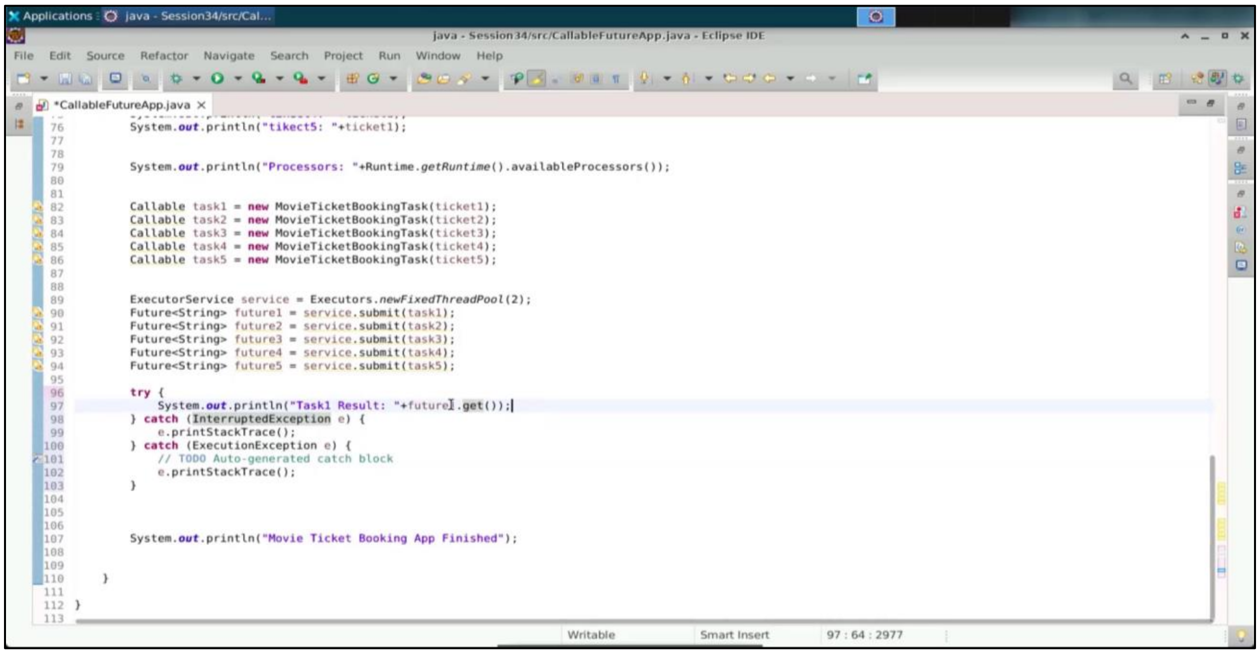


```
        MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
        MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
        MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
        MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
        MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");


        System.out.println("tikect1: "+ticket1);
        System.out.println("tikect2: "+ticket1);
        System.out.println("tikect3: "+ticket1);
        System.out.println("tikect4: "+ticket1);
        System.out.println("tikect5: "+ticket1);


        System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());


        Callable task1 = new MovieTicketBook
        Callable task2 = new MovieTicketBook
        Callable task3 = new MovieTicketBook
        Callable task4 = new MovieTicketBook
        Callable task5 = new MovieTicketBook


        ExecutorService service = Executors.
        Future<String> future1 = service.sub
        Future<String> future2 = service.sub
        Future<String> future3 = service.sub
        Future<String> future4 = service.sub
        Future<String> future5 = service.sub

        System.out.println("Task1 Result: "+future1.get());


        System.out.println("Movie Ticket Booking App Finished");
```

```
...
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
...
public static void main(String[] args) throws
 InterruptedException, ExecutionException {
...
```

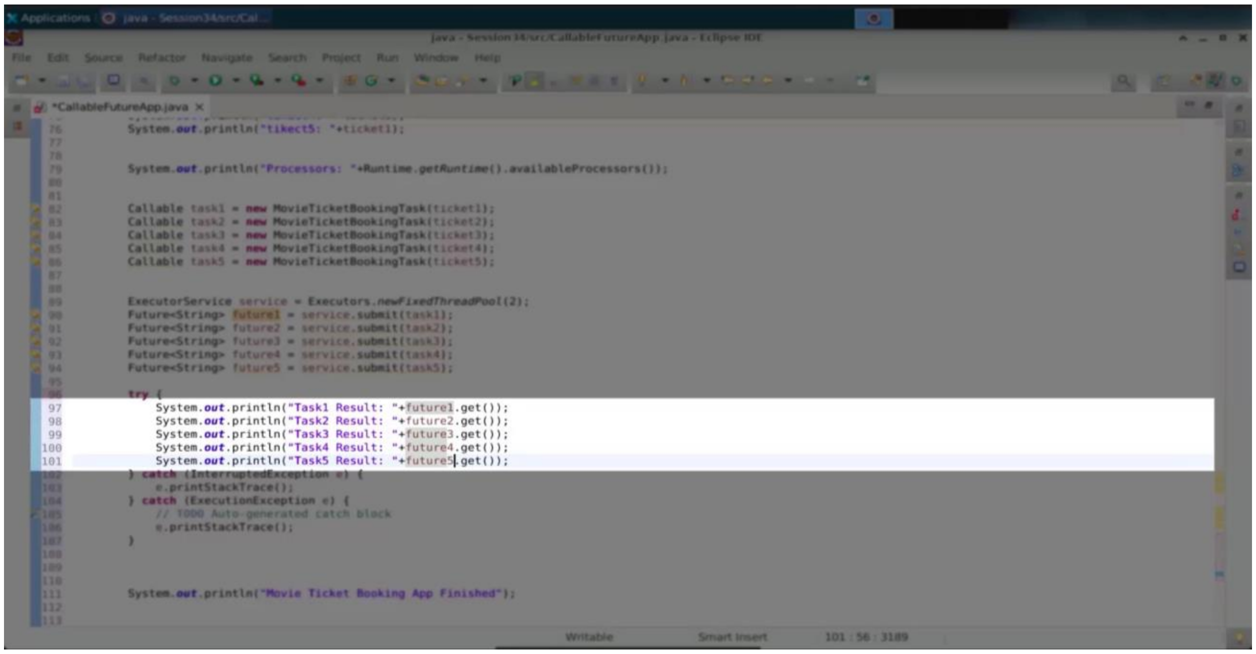
Unhandled exception type ExecutionException

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



```
        System.out.println("tikect5: "+ticket1);


        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);
        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());
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }



        System.out.println("Movie Ticket Booking App Finished");


    }

}
```

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.



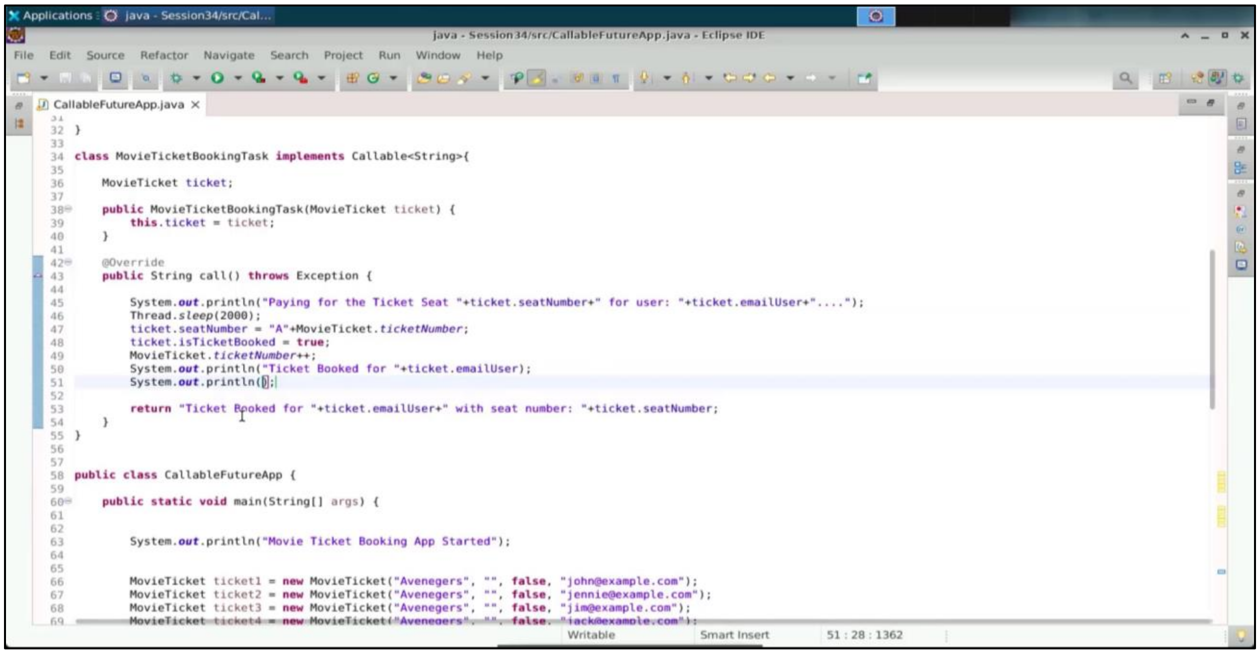
java - Session34/src/CallableFutureApp.java - Eclipse IDE

*CallableFutureApp.java

```java
        System.out.println("tikect5: "+ticket1);


        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);
        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");
```

Writable   Smart Insert   101 : 56 : 3189

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.



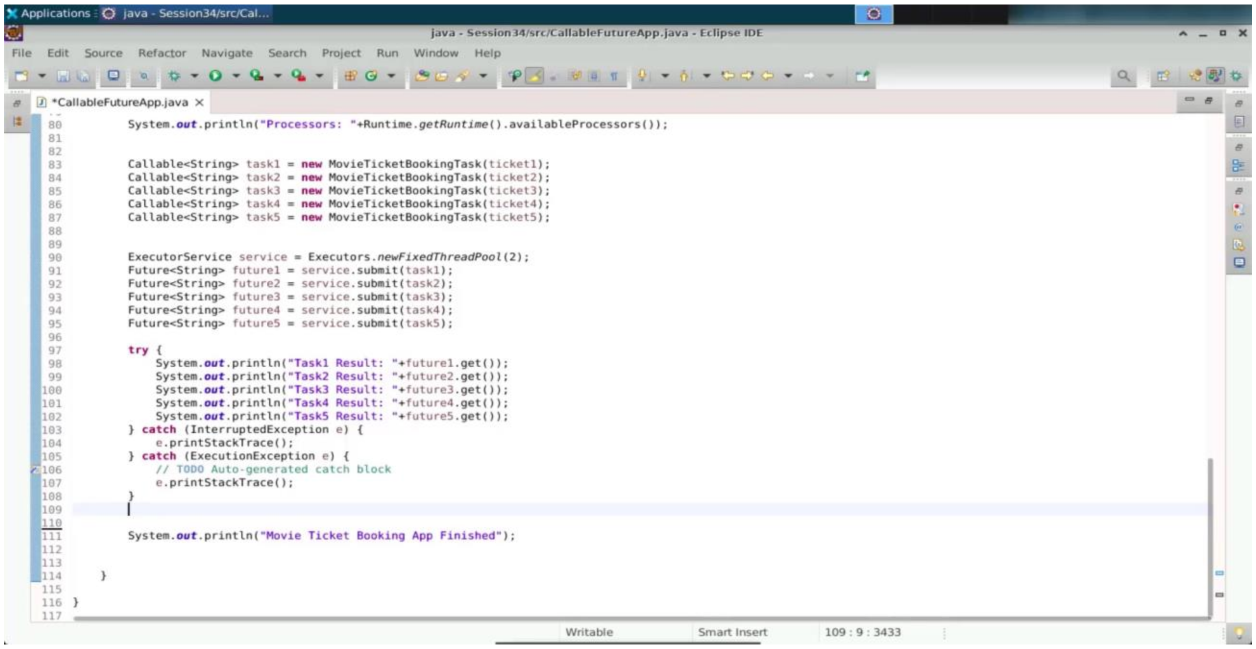
java - Session34/src/CallableFutureApp.java - Eclipse IDE

CallableFutureApp.java

```java
}

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.seatNumber = "A"+MovieTicket.ticketNumber;
        ticket.isTicketBooked = true;
        MovieTicket.ticketNumber++;
        System.out.println("Ticket Booked for "+ticket.emailUser);
        System.out.println();

        return "Ticket Booked for "+ticket.emailUser+" with seat number: "+ticket.seatNumber;
    }
}


public class CallableFutureApp {

    public static void main(String[] args) {


        System.out.println("Movie Ticket Booking App Started");


        MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
        MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
        MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
        MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
```

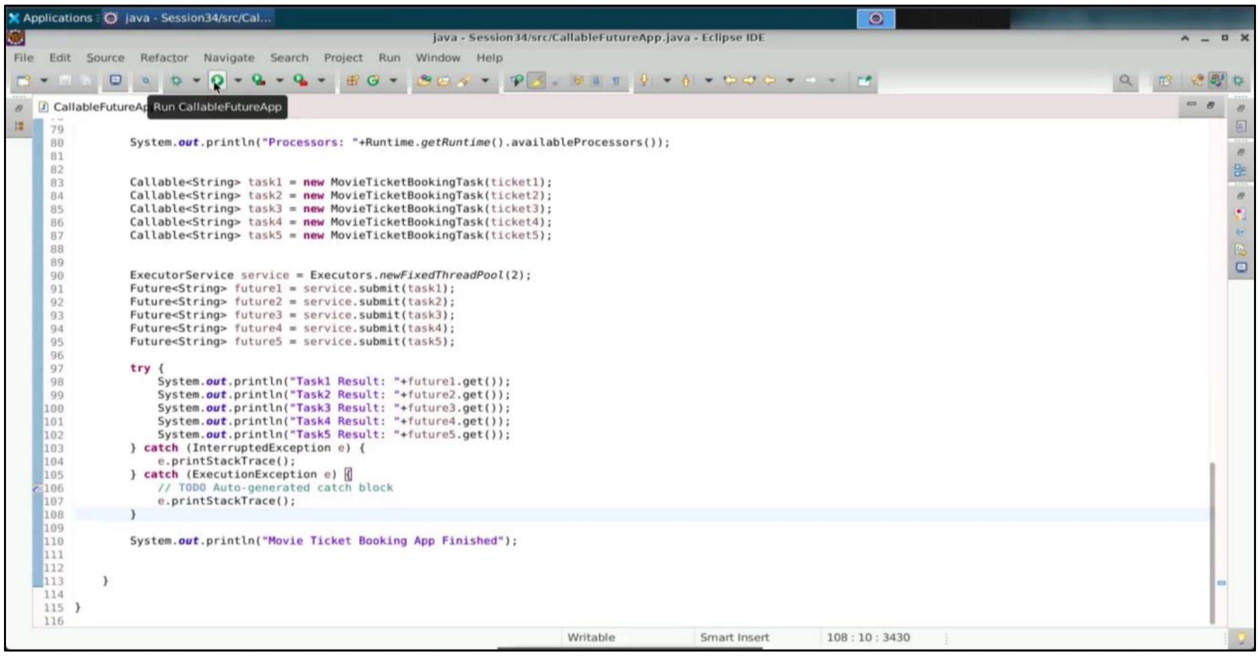
Writable   Smart Insert   51 : 28 : 1362

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.


java - Session34/src/CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*CallableFutureApp.java
Writable
Smart Insert
109 : 9 : 3433

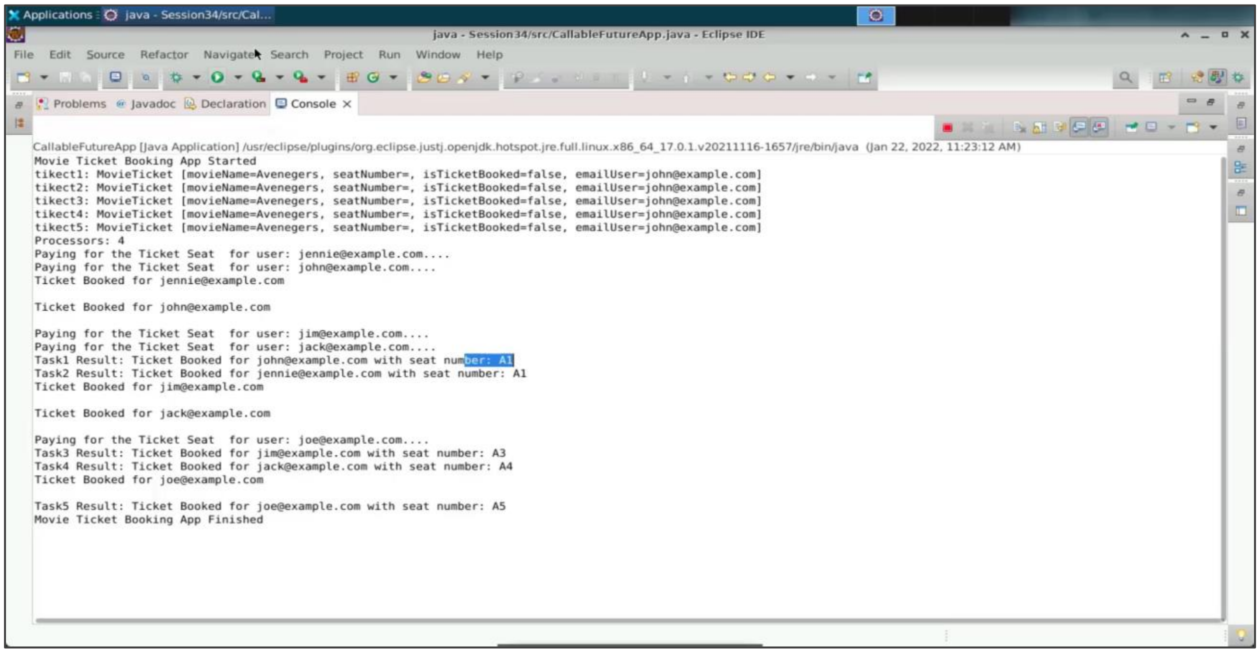
```
        System.out.println("Processors: "+Runtime.getRuntime().availableProcessors());


        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");


    }

}
```

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


java - Session34/src/CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
CallableFutureAp
Run CallableFutureApp
Writable
Smart Insert
108 : 10 : 3430

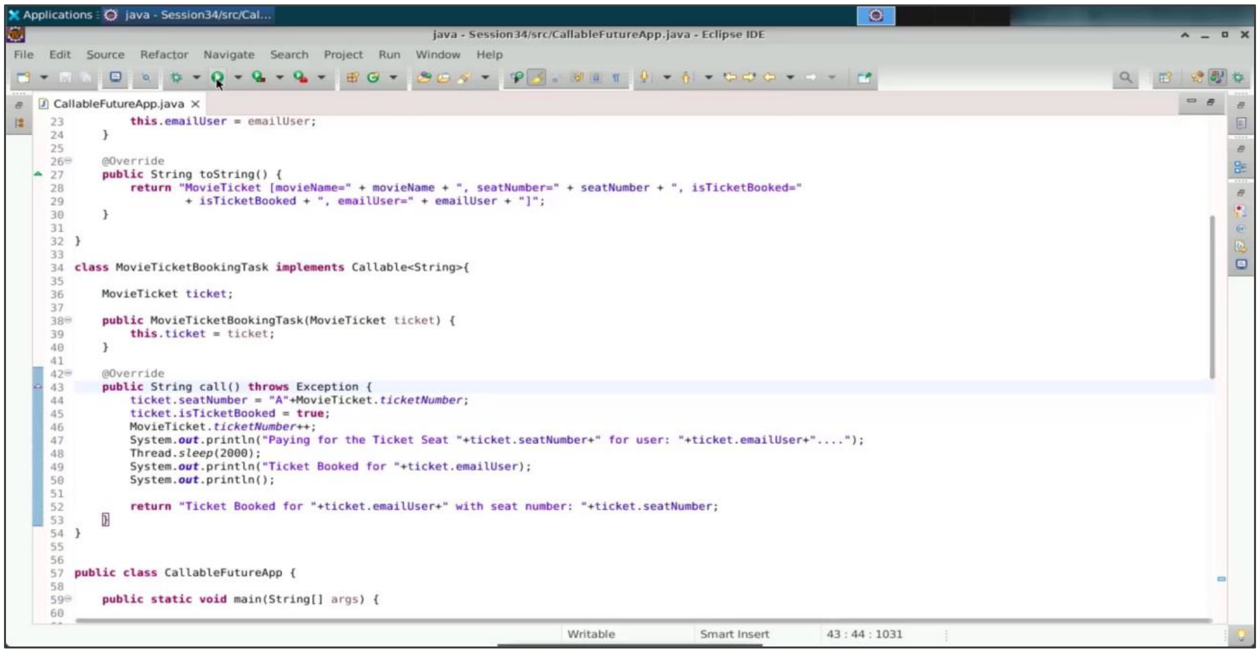
```

        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");


    }

}
```

1.25 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/Cal...
java - Session34/src/CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems
Javadoc
Declaration
Console

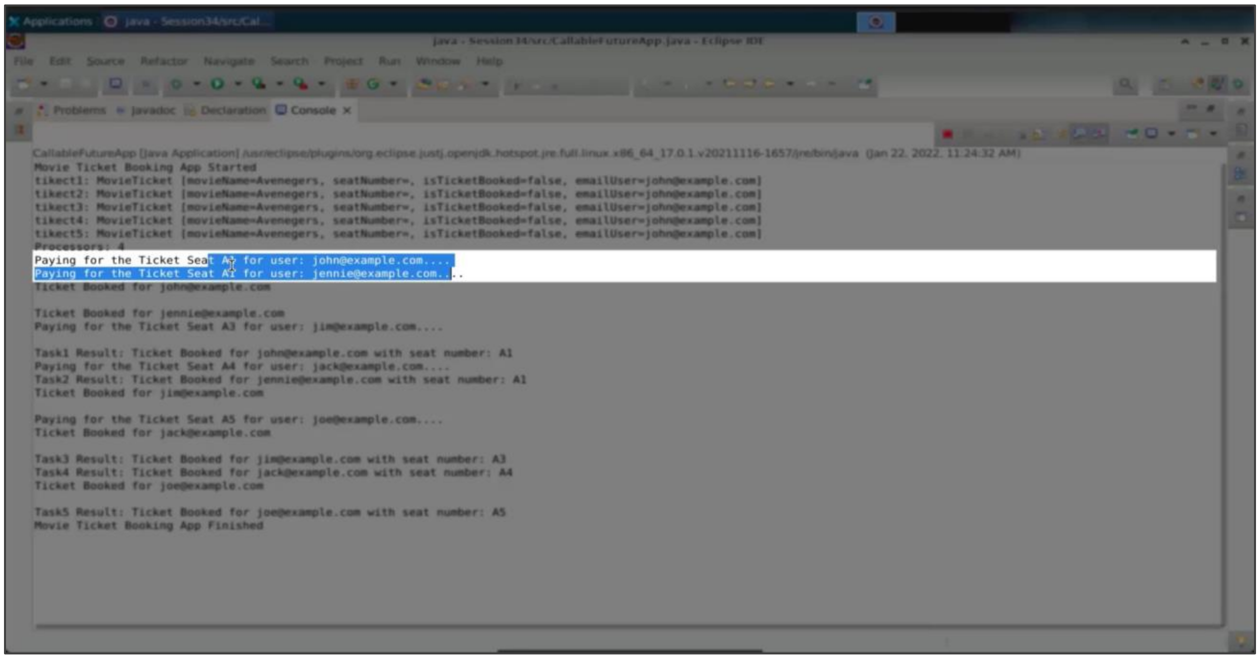
```
CallableFutureApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java (Jan 22, 2022, 11:23:12 AM)
Movie Ticket Booking App Started
tikect1: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
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  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.


Applications
java - Session34/src/Cal...
java - Session34/src/CallableFutureApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
CallableFutureApp.java
Writable
Smart Insert
43 : 44 : 1031

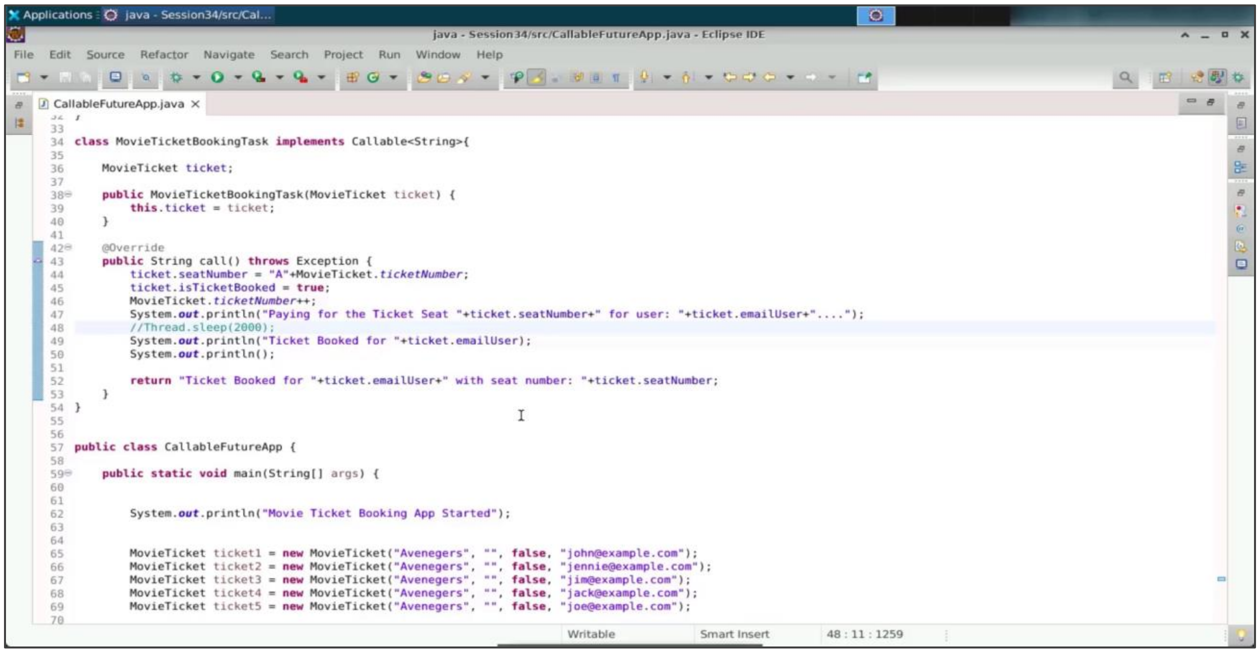
```
        this.emailUser = emailUser;
    }

    @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 {
        ticket.seatNumber = "A"+MovieTicket.ticketNumber;
        ticket.isTicketBooked = true;
        MovieTicket.ticketNumber++;
        System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser+"....");
        Thread.sleep(2000);
        System.out.println("Ticket Booked for "+ticket.emailUser);
        System.out.println();

        return "Ticket Booked for "+ticket.emailUser+" with seat number: "+ticket.seatNumber;
    }
}


public class CallableFutureApp {

    public static void main(String[] args) {

```

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.



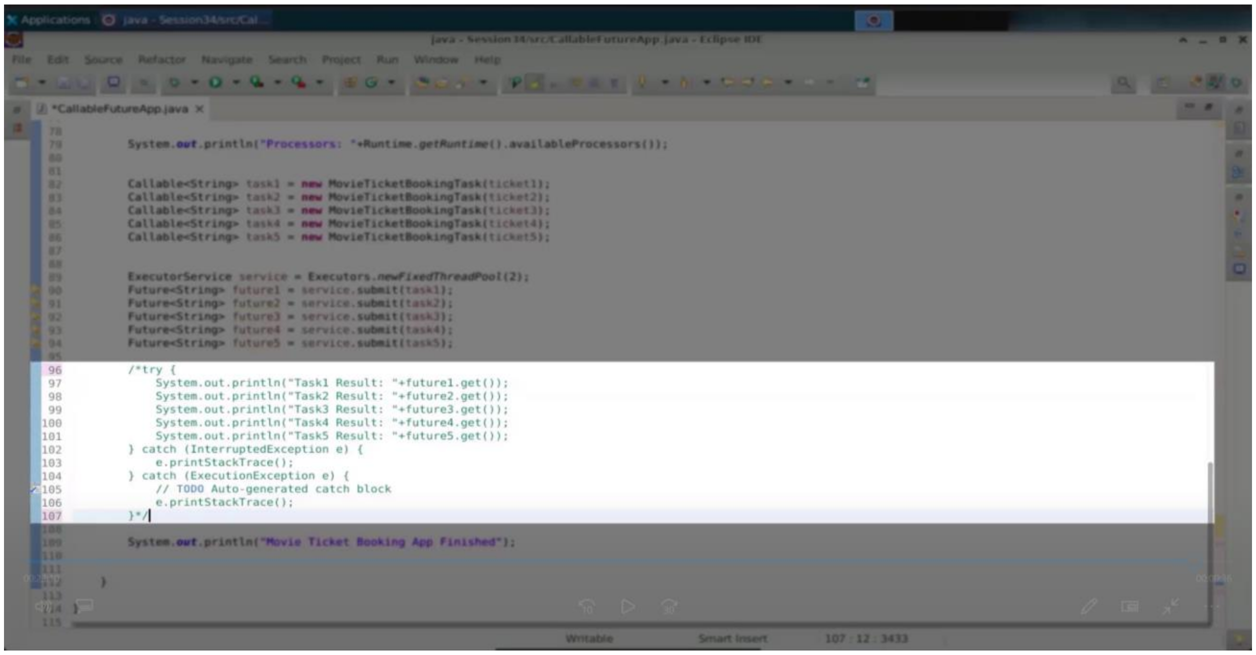
```
CallableFutureApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java (Jan 22, 2022, 11:24:32 AM)
Movie Ticket Booking App Started
tikect1: MovieTicket [movieName=Avenegers, seatNumber=, isTicketBooked=false, emailUser=john@example.com]
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 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.



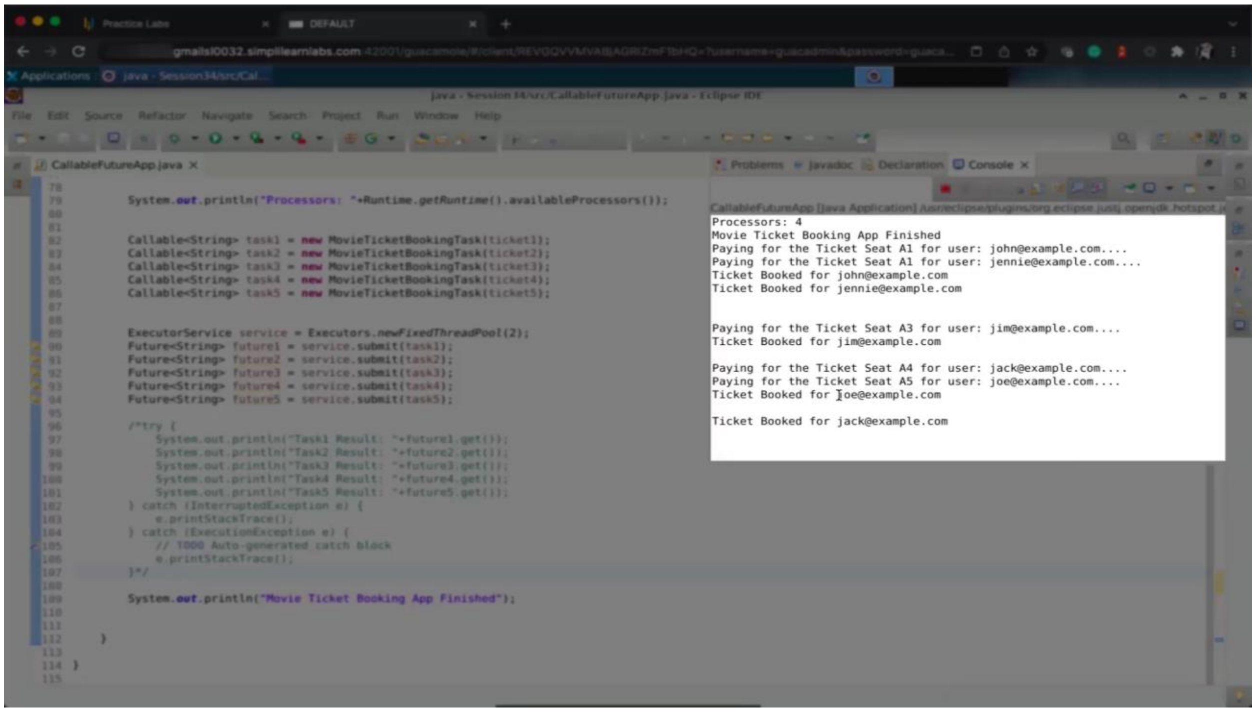
```
class MovieTicketBookingTask implements Callable<String>{

    MovieTicket ticket;

    public MovieTicketBookingTask(MovieTicket ticket) {
        this.ticket = ticket;
    }

    @Override
    public String call() throws Exception {
        ticket.seatNumber = "A"+MovieTicket.ticketNumber;
        ticket.isTicketBooked = true;
        MovieTicket.ticketNumber++;
        System.out.println("Paying for the Ticket Seat "+ticket.seatNumber+" for user: "+ticket.emailUser+"....");
        //Thread.sleep(2000);
        System.out.println("Ticket Booked for "+ticket.emailUser);
        System.out.println();

        return "Ticket Booked for "+ticket.emailUser+" with seat number: "+ticket.seatNumber;
    }
}


public class CallableFutureApp {

    public static void main(String[] args) {


        System.out.println("Movie Ticket Booking App Started");


        MovieTicket ticket1 = new MovieTicket("Avenegers", "", false, "john@example.com");
        MovieTicket ticket2 = new MovieTicket("Avenegers", "", false, "jennie@example.com");
        MovieTicket ticket3 = new MovieTicket("Avenegers", "", false, "jim@example.com");
        MovieTicket ticket4 = new MovieTicket("Avenegers", "", false, "jack@example.com");
        MovieTicket ticket5 = new MovieTicket("Avenegers", "", false, "joe@example.com");
```

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



```
        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.



```java
        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");


    }

}
```

```
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.