Lesson 04 Demo 09

Implementing Multithreading

Objective: To implement multithreading in Java

Tools required: Eclipse IDE

Prerequisites: None

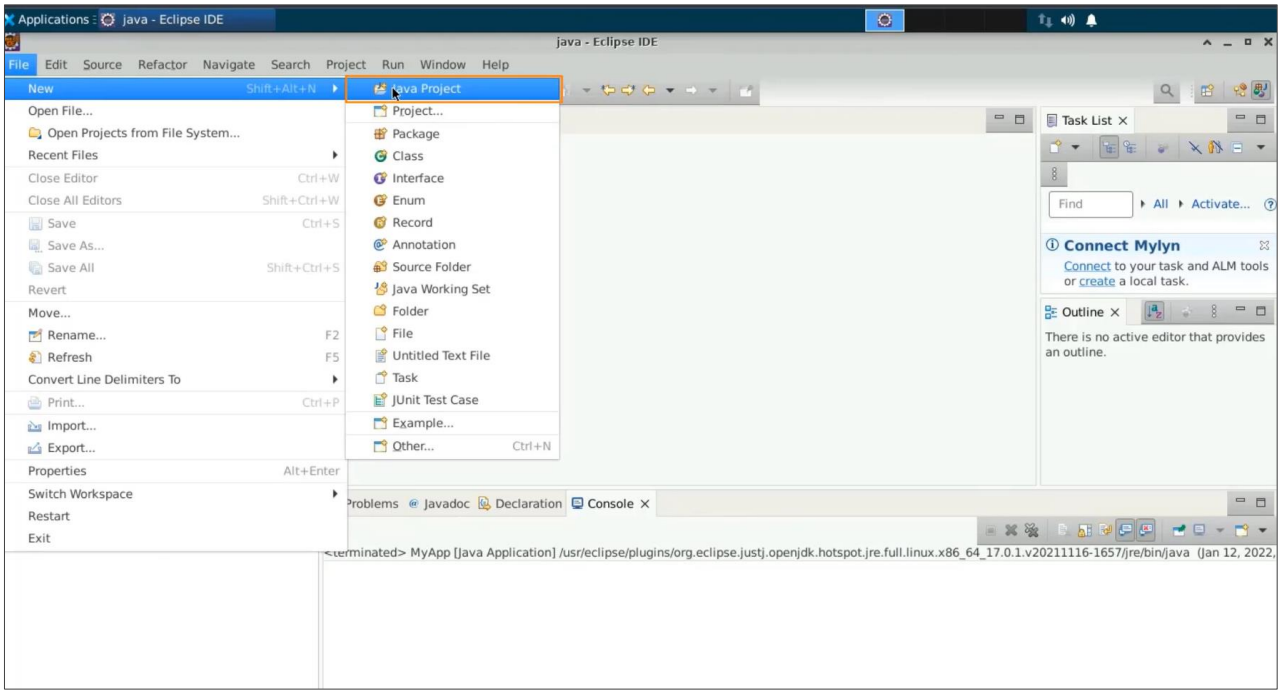
Steps to be followed:

1. Open Eclipse IDE and create a new class
2. Write a basic for loop and execute the code
3. Use the paste operation, write a print task
4. Implement a use case, with sample data and execute the code
5. Implement multithreading and override a method
6. Write a polymorphic statement
7. Print out the names of threads
8. Implement the concept of priority and Max Priority and execute the code
9. Access the state
10. Mark the thread as a daemon method

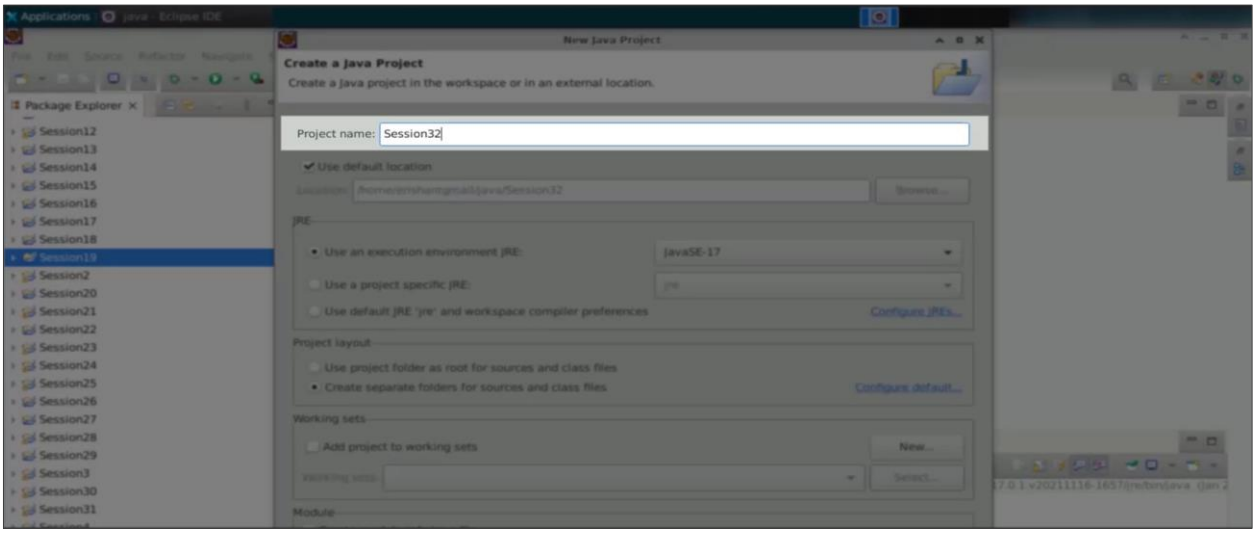
Step 1: Open Eclipse IDE and create a new class

1.1 Open the **Eclipse IDE**

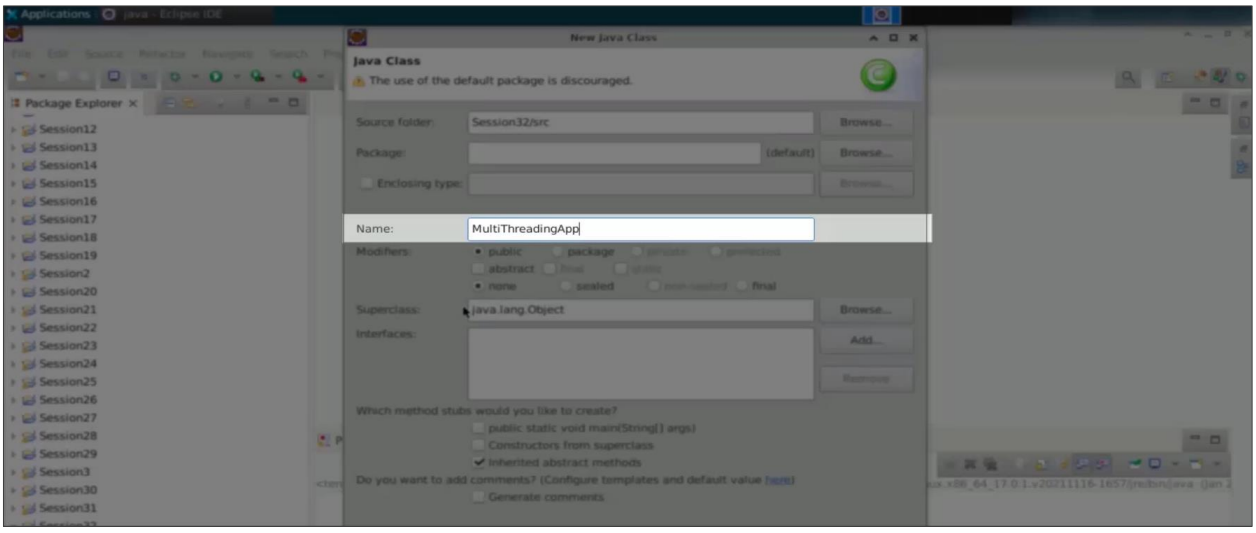
1.2 Select **File**, then **New,** and then **Java project**



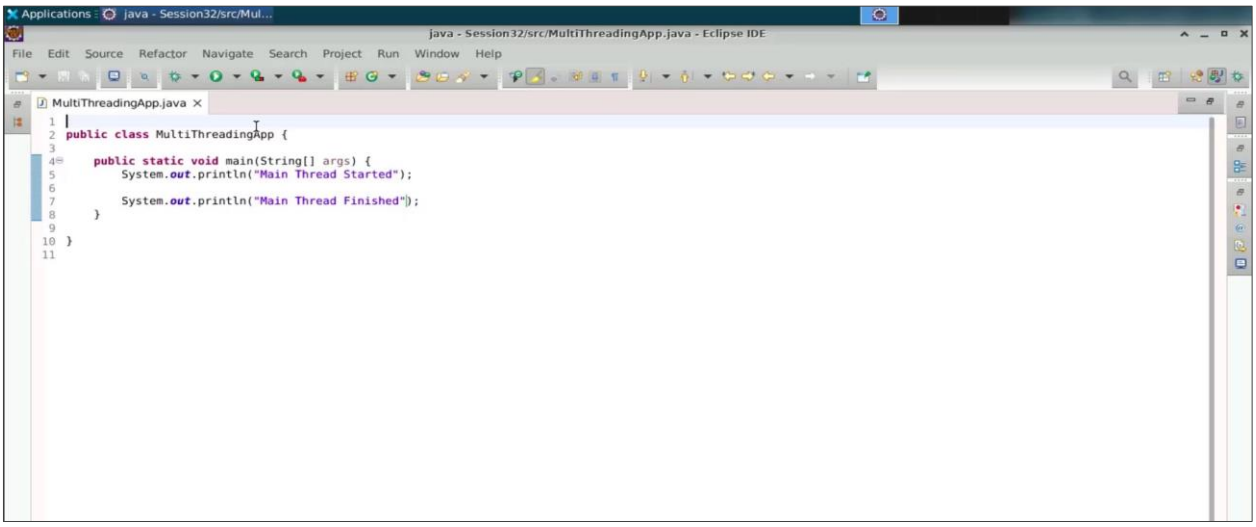
1.3 Name the project **“Session32”,** uncheck **“Create a module info.Java file”**, and press **Finish**



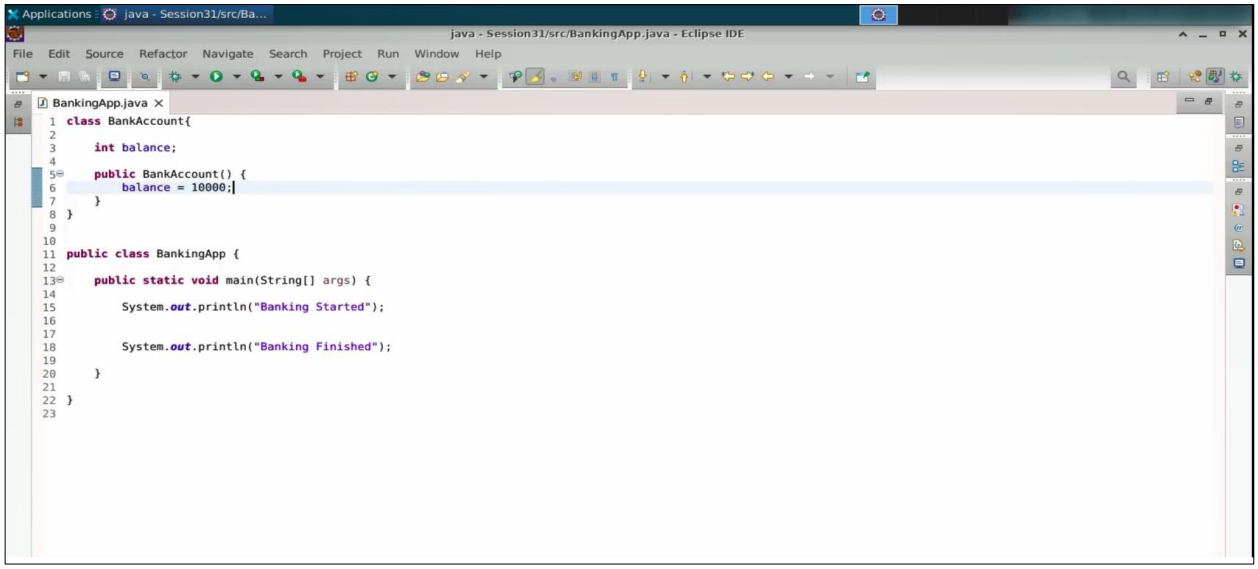
1.4 With Session32 in the src folder, right-click and create a new class. Name this class `MultiThreadingApp`, select the main method option, and then click finish.



1.5 Now, the main method itself is a thread. So, add `System.out.println("Main thread started");` at the beginning and `System.out.println("Main thread finished");` at the end.

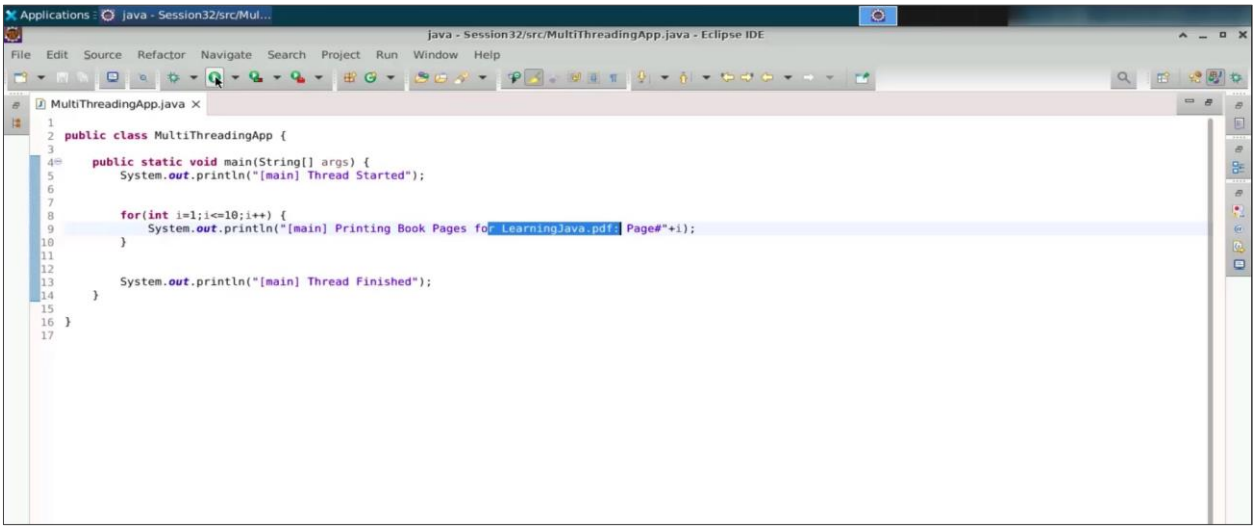


1.6 Create a class called `BankAccount`. In this class, define a balance variable. When you create an object of `BankAccount`, set the initial balance to 10,000.

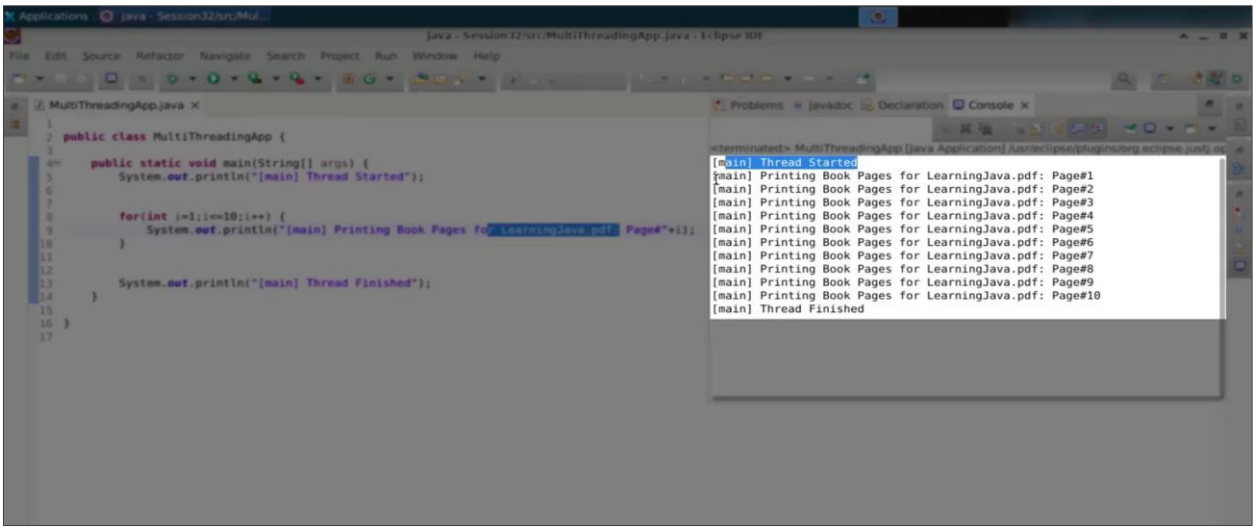


Step 2: Write a basic for loop and execute the code

2.1 Come here and assign a task to the main method by writing a basic for-loop. It starts with `i` set to 1 and runs while `i` is less than or equal to 10, incrementing `i` each time. Let us say you are printing book pages, and the message is "Page number: " followed by the value of `i`. You are printing the first 10 pages of a book called "learning java.pdf," so write "Book pages for learning java.pdf".

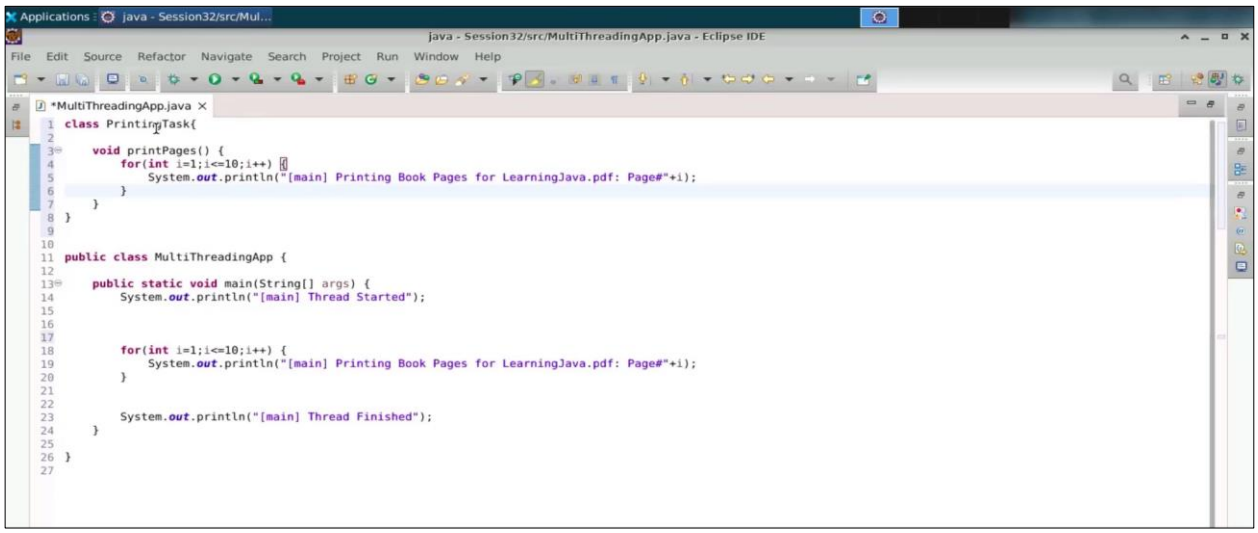


2.2 When you run your program, you can see that the main started, which was your main task. That is, printing 10 pages from 1 to 10.

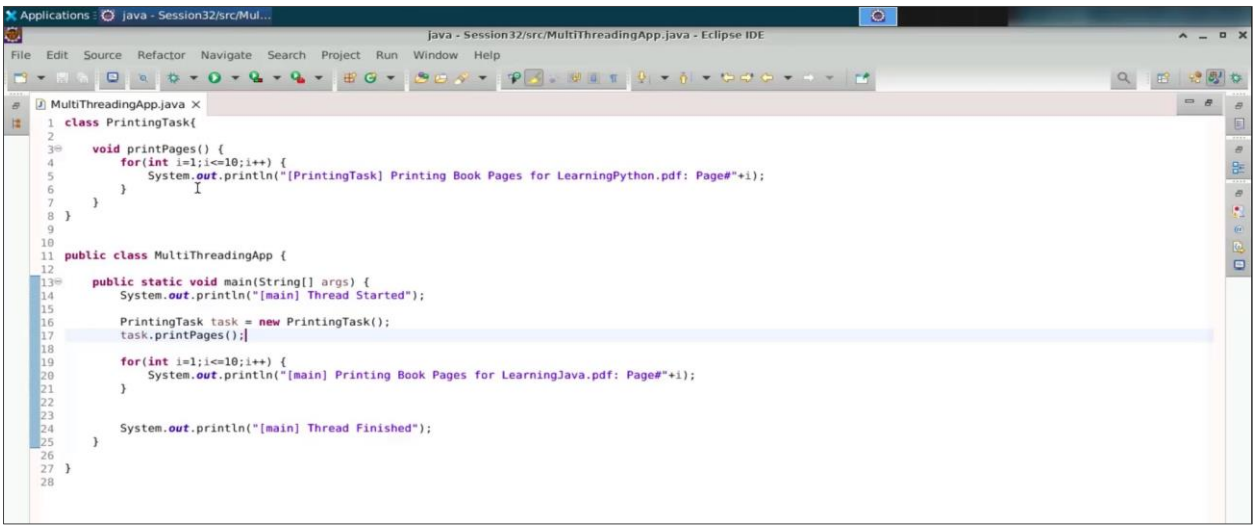


Step 3: Use the paste operation, write a printing task

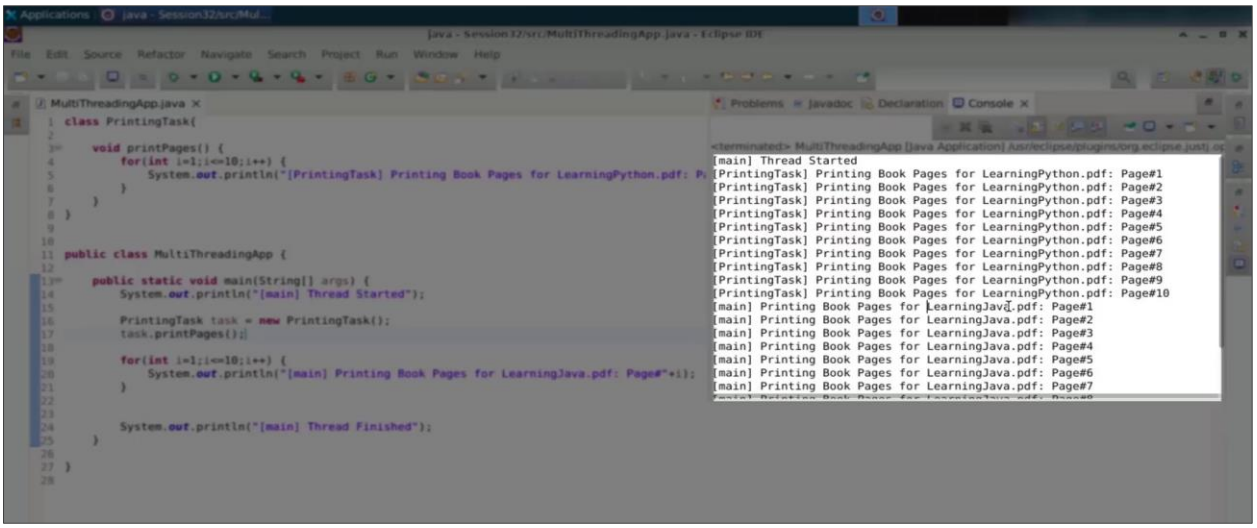
3.1 Consider that you are having one more Class, and this is referred to as printing task, and here you can have a method called print pages. So, you will do the same job that the main is doing. Come here and do a paste operation, this is through this class called printing task.



3.2 Here, you will write a PrintingTask class. In this class, write a method to print "Book pages for learning python.pdf" for 10 pages. In the main method, before the for loop, create an object of this PrintingTask class. Write PrintingTask task = new PrintingTask(); and then call task.printPages();

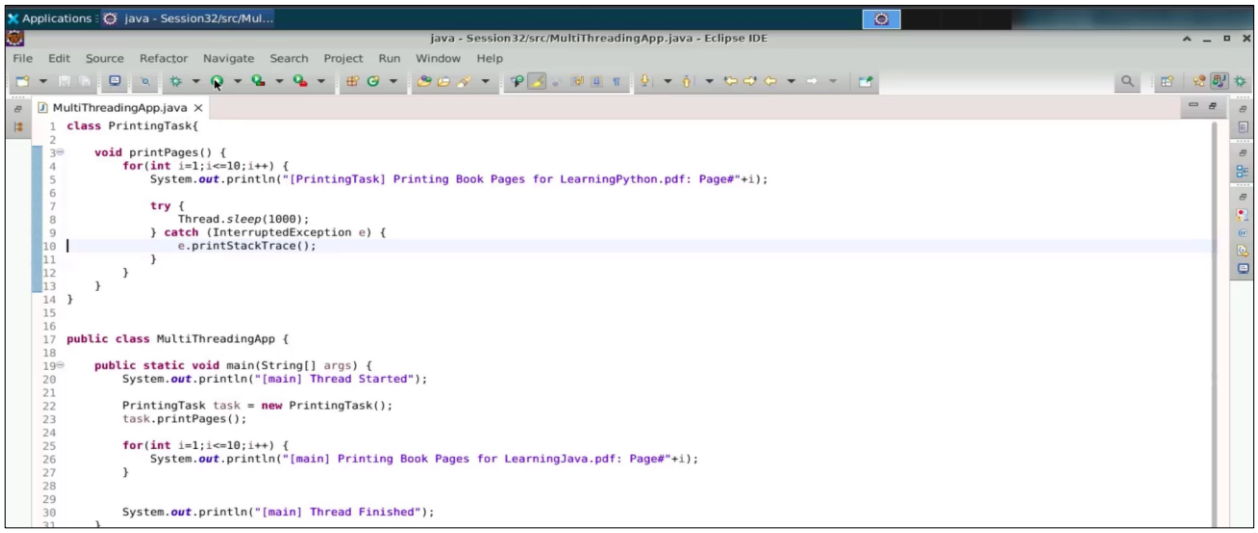


3.3 When you run the code here, you can observe that the application started, that is, the main thread started. The Main is printing the task for learning Python.pdf, and not for learning Java.pdf. So, it starts only when the task for learning Python.pdf is finished.

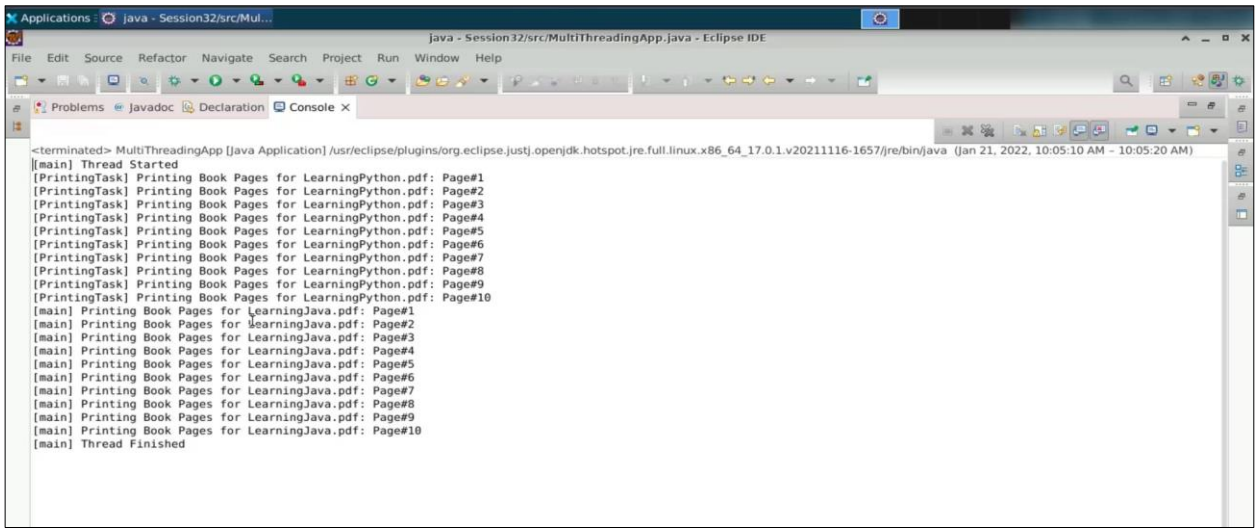


Step 4: Implement a use case, with sample data and execute the code

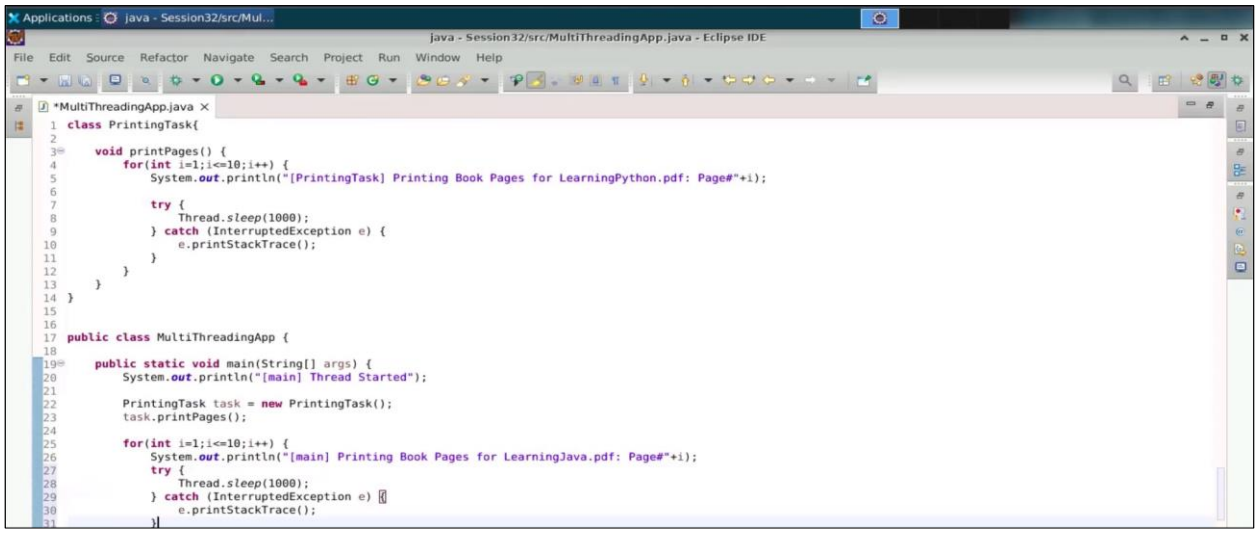
4.1 Let us take a use case where printing one page takes one second. You can introduce a delay using the Thread class, which has a built-in static method called sleep. Pass the duration as 1000 milliseconds, which means one second. Since sleep throws an InterruptedException, a checked exception, you need to surround this code with a try-catch block.



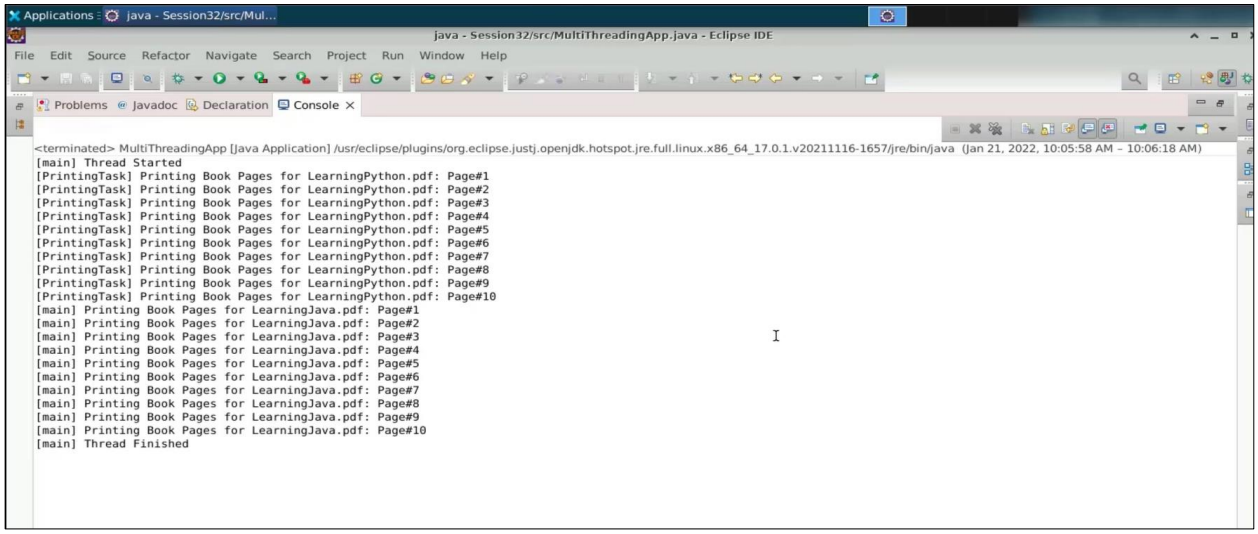
4.2 When you run the program, what you can observe is that a single page is getting printed after one second. So, till the time this printing task does not get finished, the below task in the main will not start, hence this learning Java dot PDF will not start till the time you do not finish off this printing task.



4.3 This means multithreading is important when you know that a certain task is getting blocked. Now, how can you work on this? You can also introduce this sleep inside the for-loop. In the main, you'll introduce the same delay.

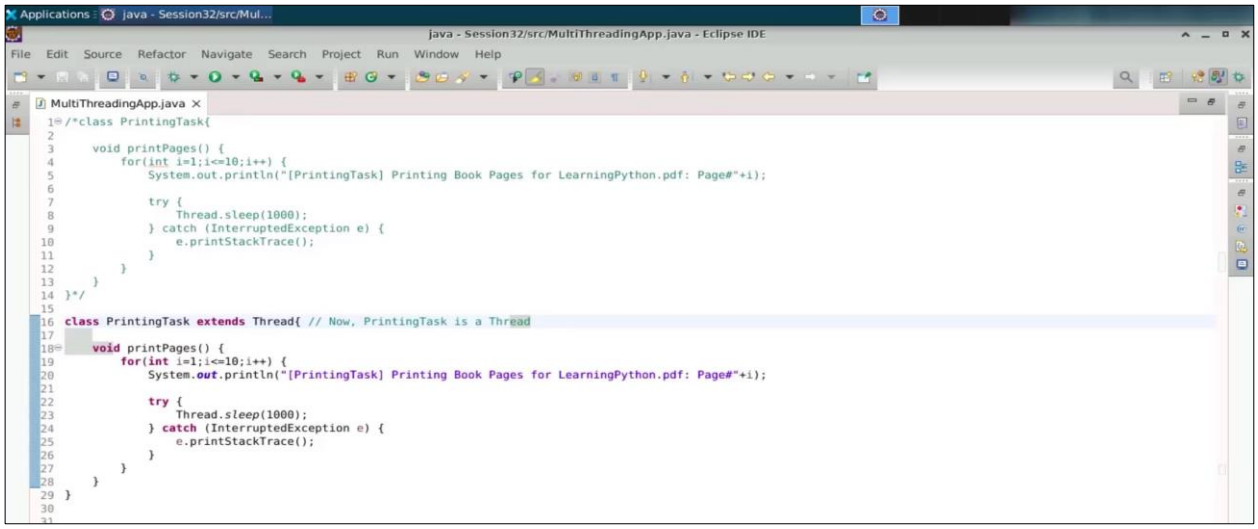


4.4 Let's run this code. You can see, first, all the learning Python dot PDF would be printed, and then it would start for the Java dot pdf, the execution happens in a sequence one after the other, and learning Java dot pdf is a blocker.

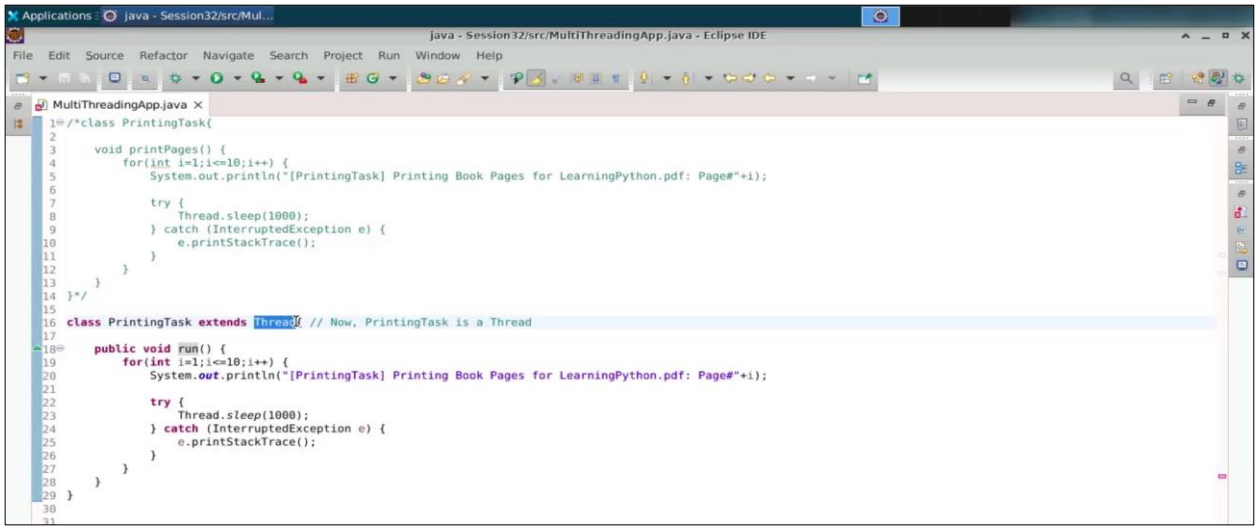


Step 5: Implement multithreading and override a method

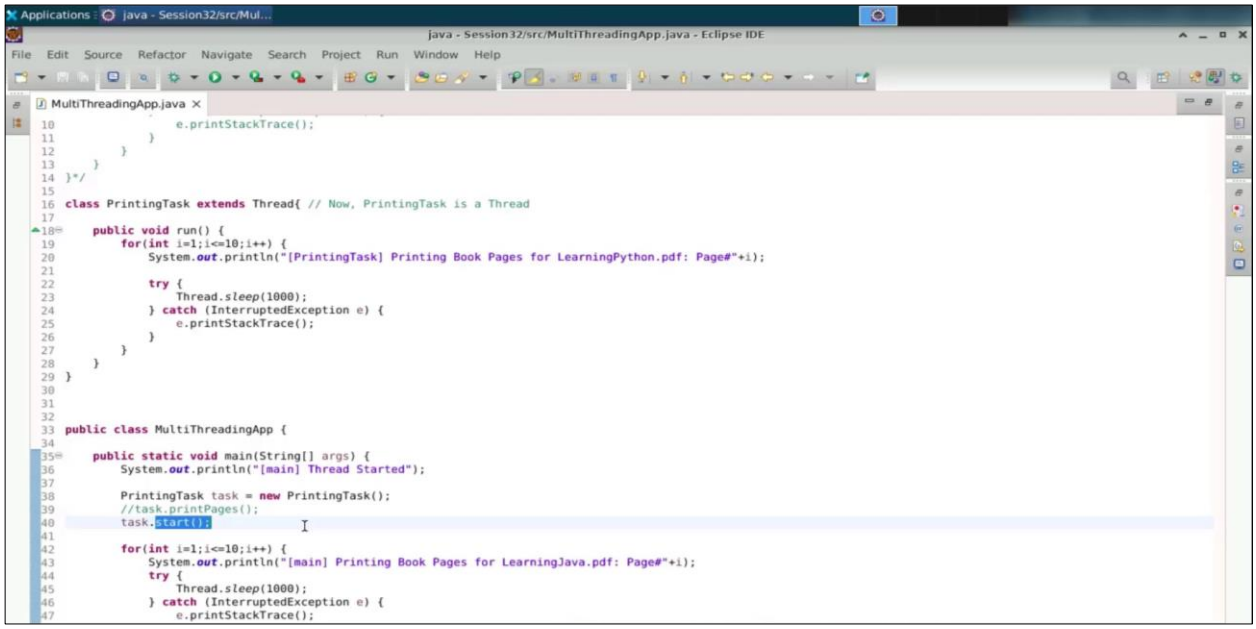
5.1 Let us implement multithreading and see how you can make both things run in parallel. The fundamental is basic, rather than you create a regular class, this was your previous code that is the regular class. You will commend this and write Printing task will extend the thread class, hence this becomes a thread. Now printing task is a thread and the parent becomes the thread.



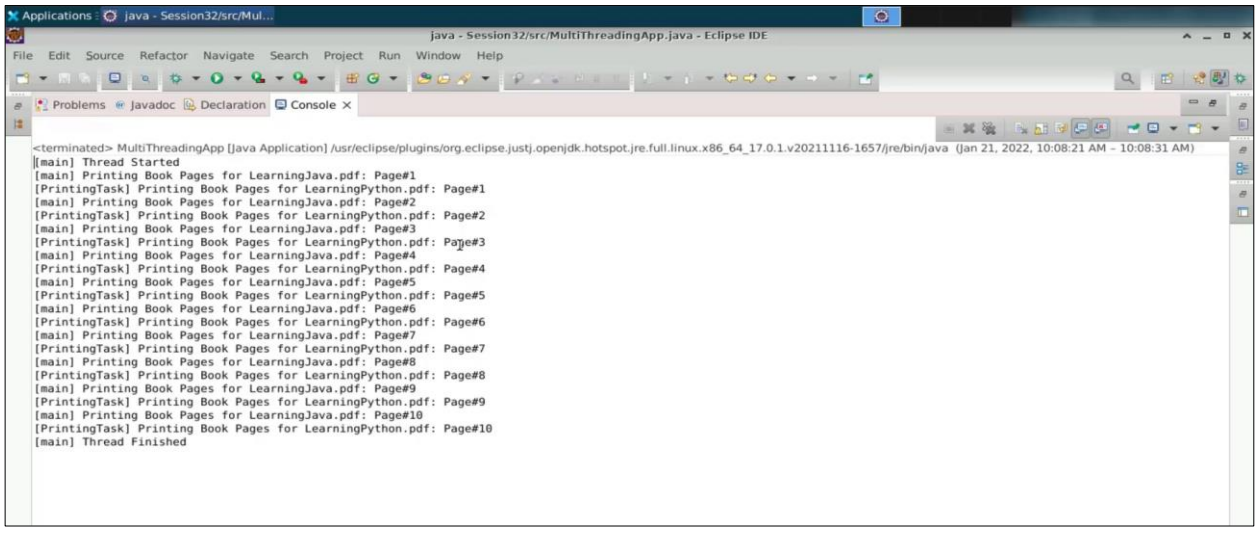
5.2 You can just eliminate this method called print pages, and override a method called run. And When you override the run method, this is a method that is available in the parent class thread. Run method is a method in which you need to write the task to be performed or your code to be executed with the thread.



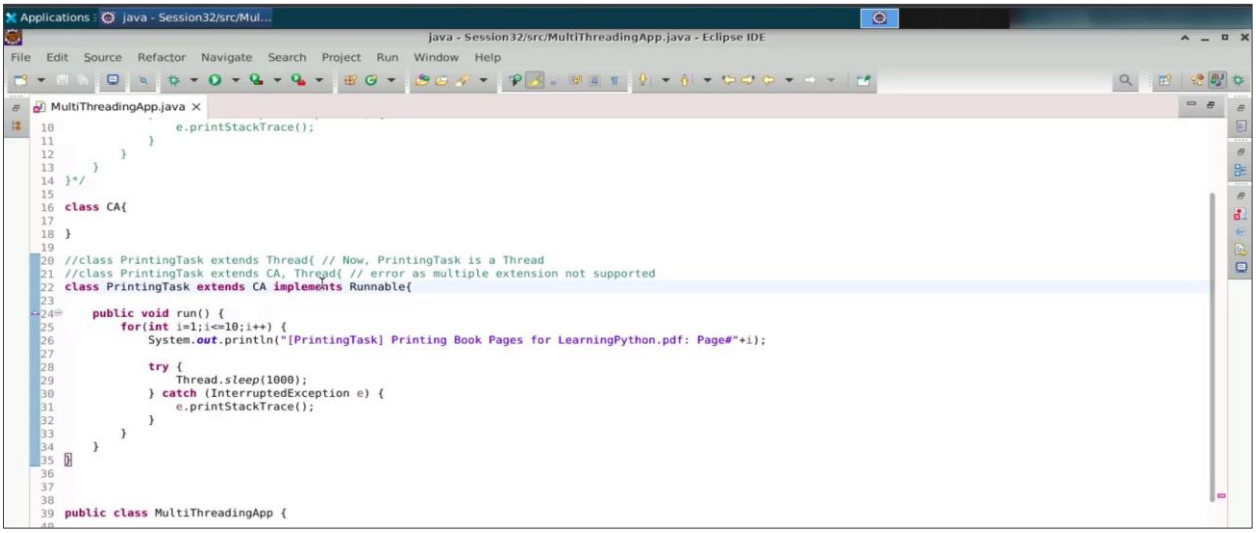
5.3 Create the object of printing task. However, now you do not have the method called print pages, you have the method called run. So, you need to execute a method called start. That is, the start method on the thread will execute internally the run method.



5.4 When you create a thread, you need to execute your codes within the run method. This is mandatory to override, and the run method will be executed internally by the start method. This is one of the protocols you need to follow to create a thread in your program.

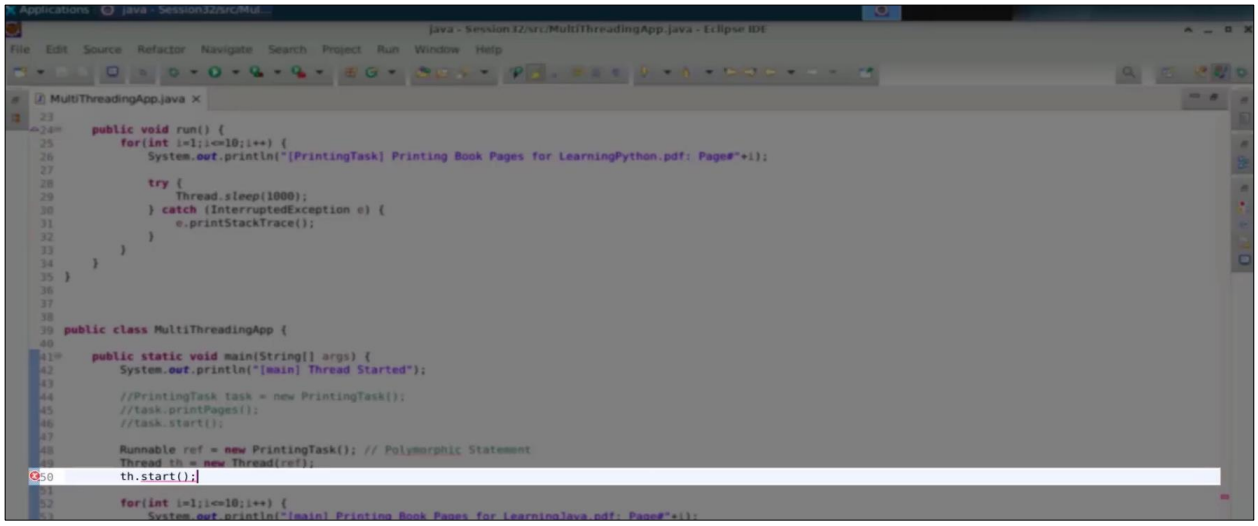


5.5 If you want to create a thread, there is one more way to do it. Let us assume that you have class called CA, and the printing task extends CA. When you write comma thread, it will give an error. Next, write class printing task, let it extend CA and along with the extension, it will implement runnable. That is, you will implement the runnable interface rather than doing an extends on the thread.

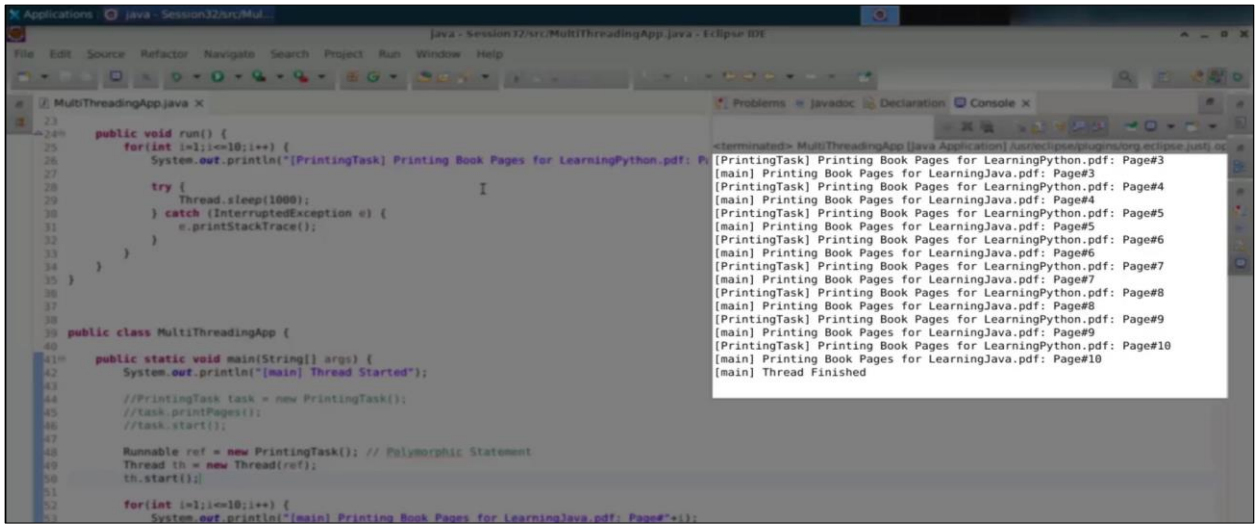


Step 6: Write a polymorphic statement

6.1 Now, you will not create a direct object of printing task and you will not use start. What you will do is, you will write a polymorphic statement that is runnable create a reference of runnable and then create the object of printing task, and polymorphic statement. Next, create the object of thread, write thread th is a new thread and pass the target runnable inside your thread constructor.

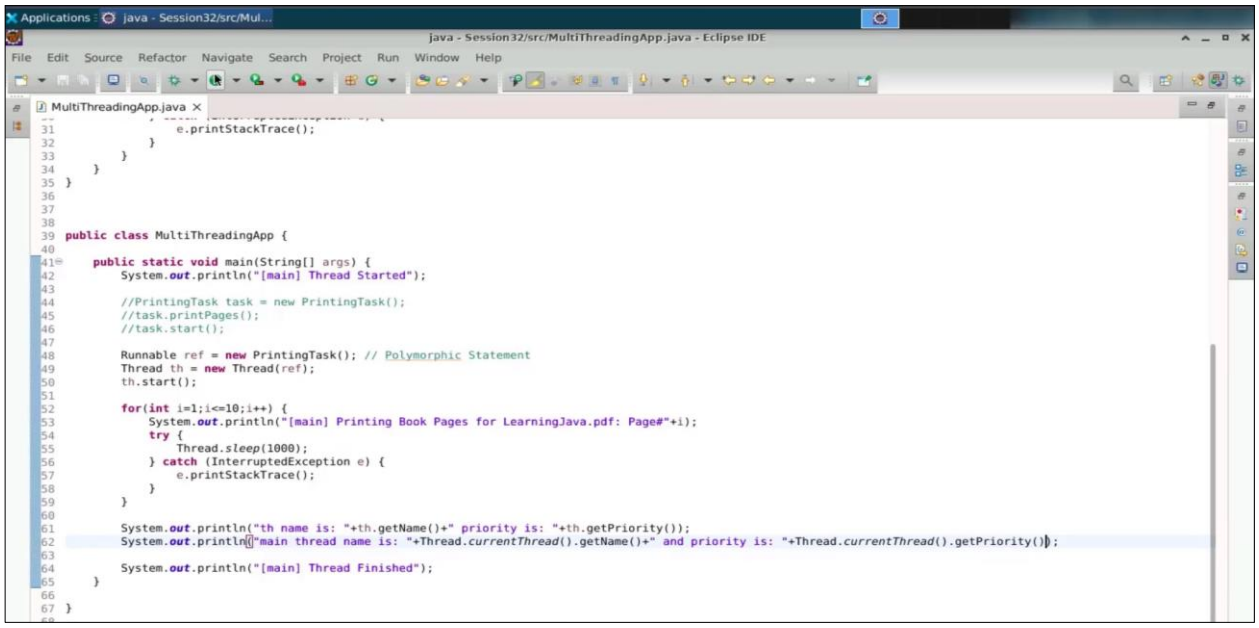


6.2 If you run the code, you can find that you have achieved the same output again. Now you have two threads in your program, one is the main thread, another one is the child thread called printing task, if you wish, you can either create the thread by extending the thread, or you can implement runnable; both the ways you can work with multithreading.

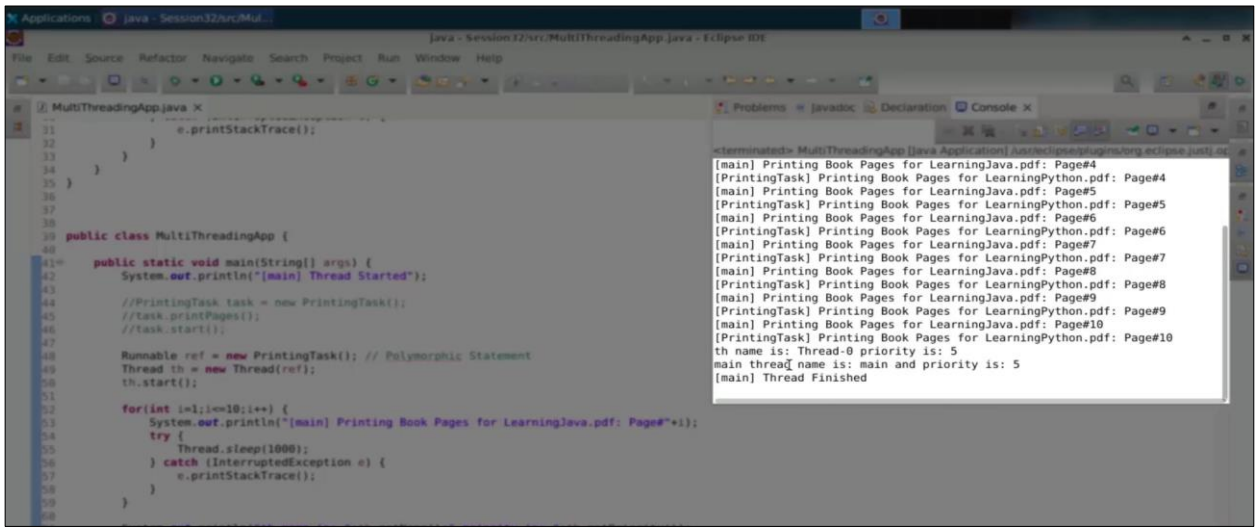


Step 7: Print out the names of threads

7.1 Let's print out the names of your threads. Write th name is plus th.getName(), so your threads can have a name. Then, write main thread name is plus Thread.currentThread().getName(). For the main thread, you don't have a direct reference like you do for other threads. You can also print their priority by writing priority is plus th.getPriority(). Similarly, print the main thread's priority by writing priority is plus Thread.currentThread().getPriority().

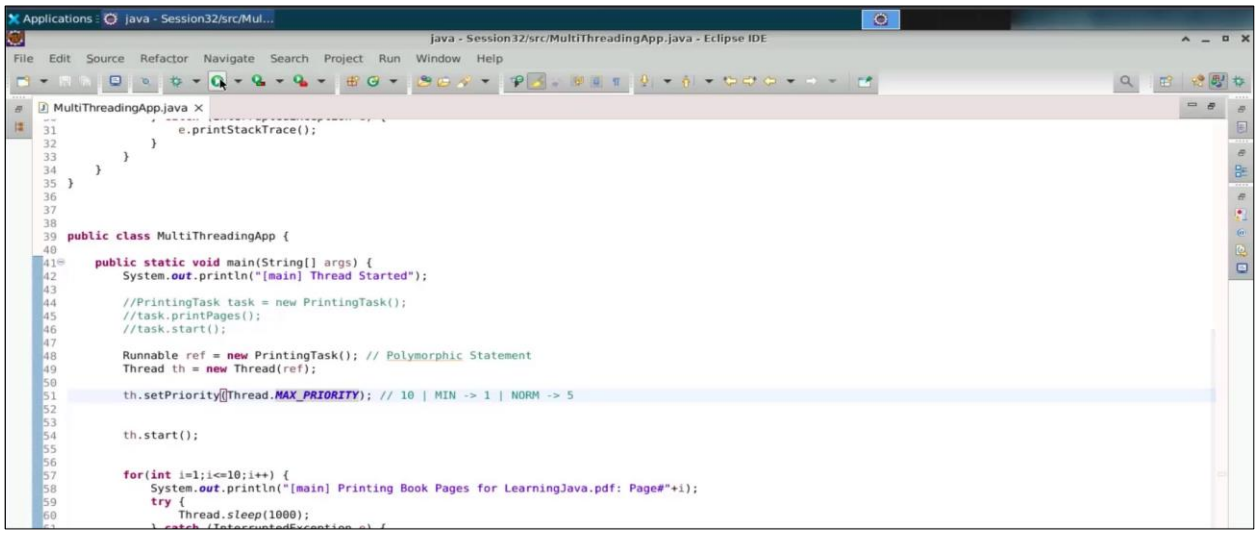


7.2 Run the code and see what the output is. You will have your thread execution finished and now you can see, by default, the name of the thread is given as Thread 0, and the priority is 5, the same way for the main, the name is main only and the priority is 5.

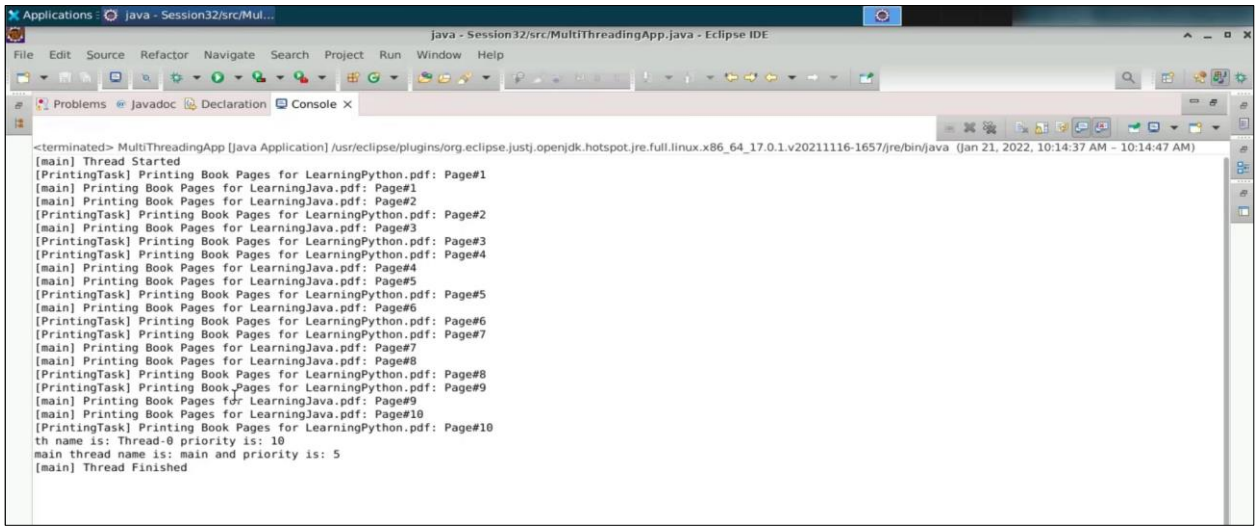


Step 8: Implement the concept of priority and Max Priority and execut the code

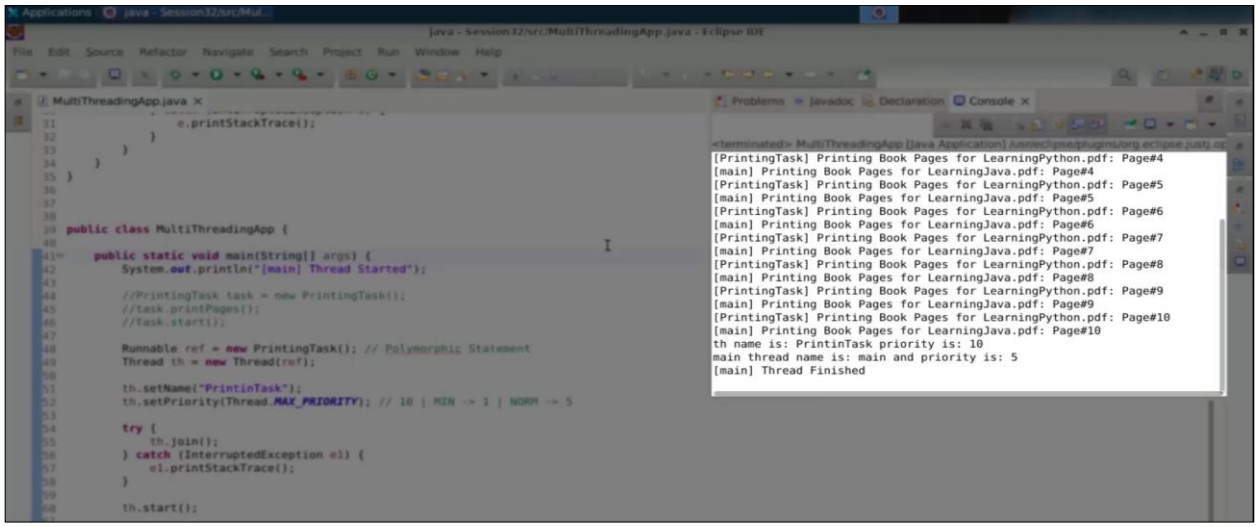
8.1 What is meant by priority? Priority is one of the indicators which tells the JVM to execute the threads, accordingly, whenever there are low memory conditions. If you want, you can set the priorities. Before your thread starts, here you can write thread dot set the priority, there are three priorities which you can set, one is the priority called Max Priority.



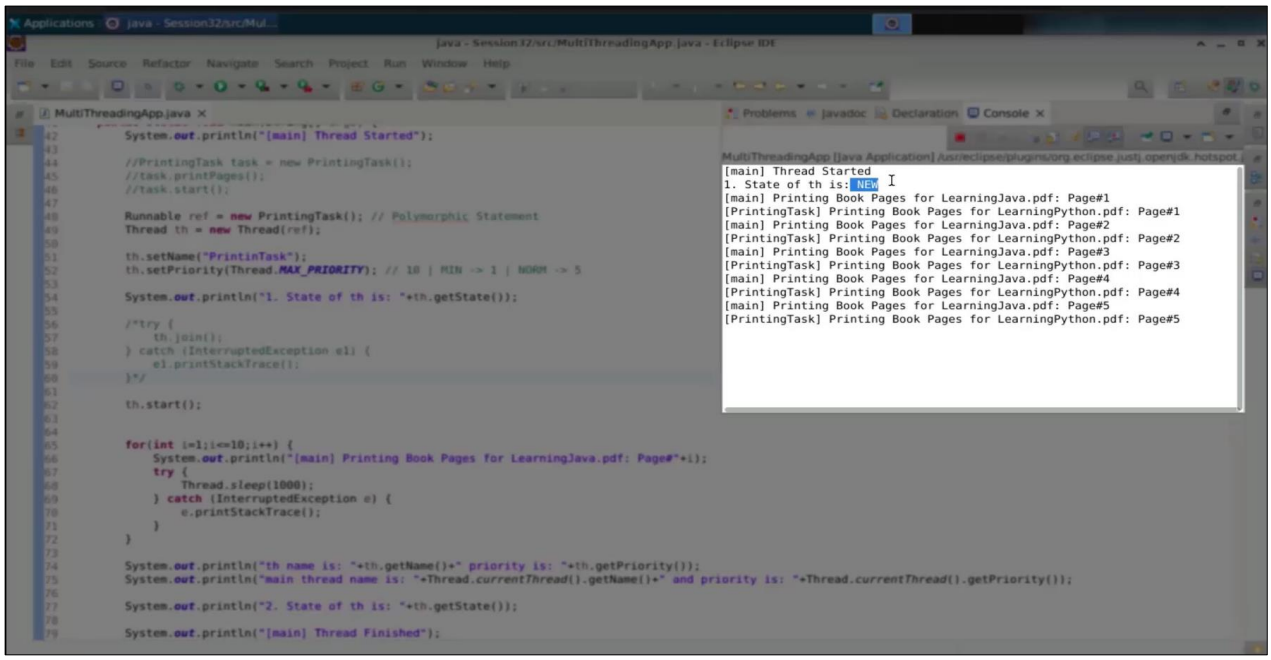
8.2 However, priority does not mean that this entire thread will be executed before any other thread. When you use printing task as a priority, you can see it is just prioritizing the output, and here the value goes as 10.



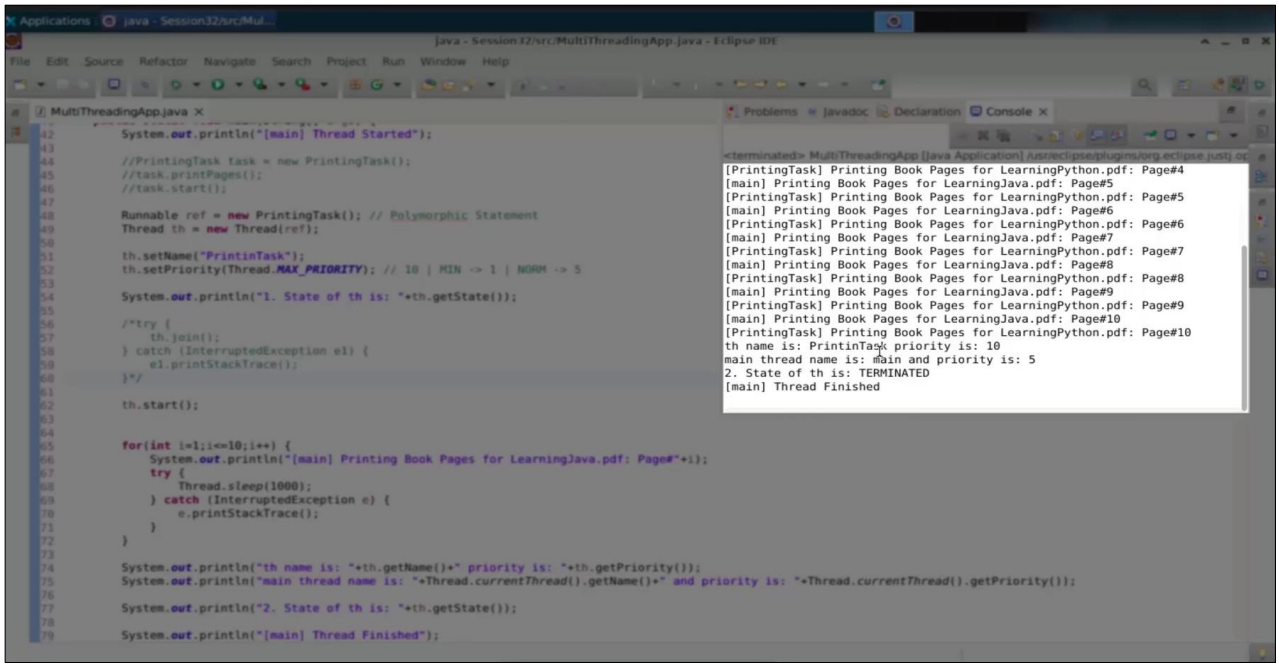
8.3 When you run the program, now you can see the priority. When you use join, you should make sure that you need to prioritize your thread completely. And with this multithreading, you have a few more methods, when you put a method known as the state of the thread.



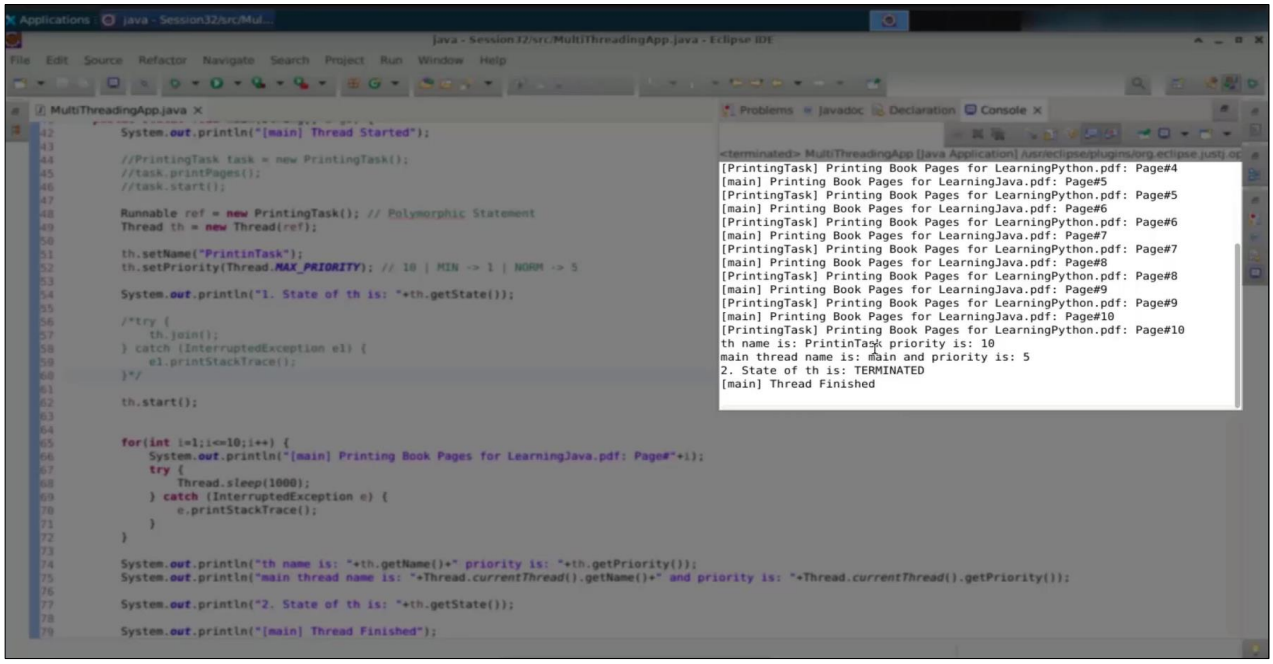
8.4 Print “State of TH is “ followed by th.getState(). This is the second print statement. Write System.out.println(“State of TH is “ + th.getState());. Comment out the join method and run the code. The first state of the thread will be shown as NEW.



8.5 Let's see what happens with the last statement, and the last state is shown as terminated.

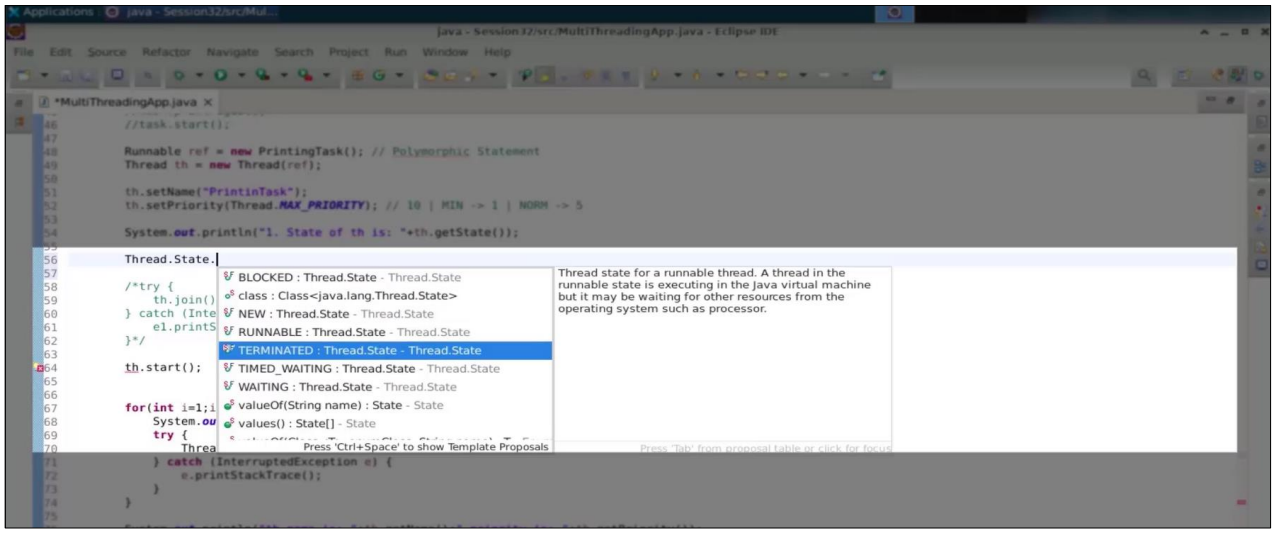


8.6 Let's see what happens with the last statement, and the last state is shown as terminated.

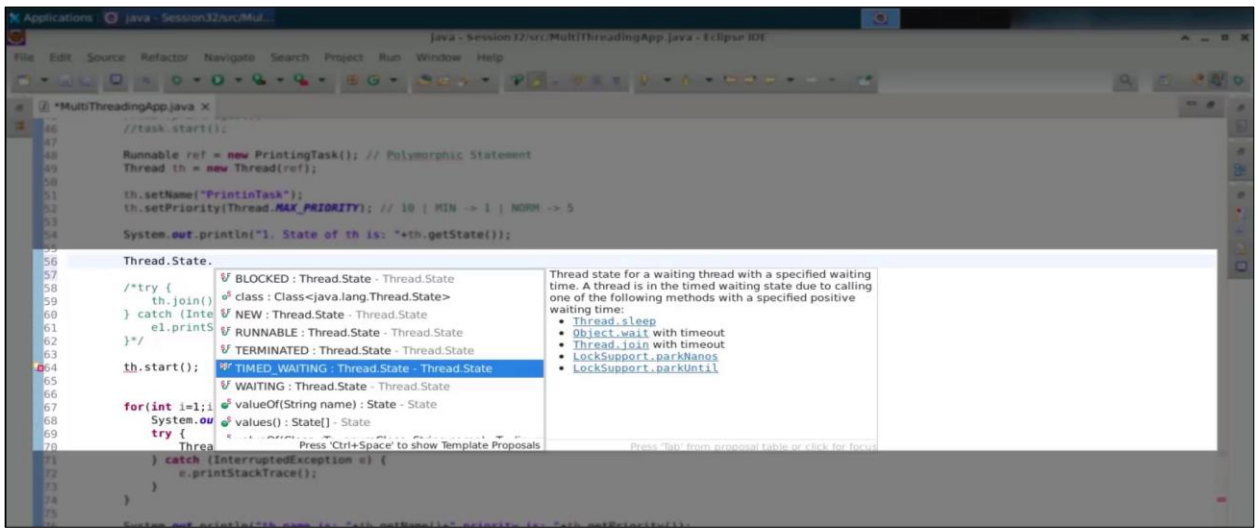


Step 9: Access the state

9.1 Let us also see how you can access the state, write thread dot state dot, and these are the states on a thread. New is a state when your thread must start, runnable is the state when your thread has started its execution, terminated is a state when your thread has finished its execution, these are three major states.

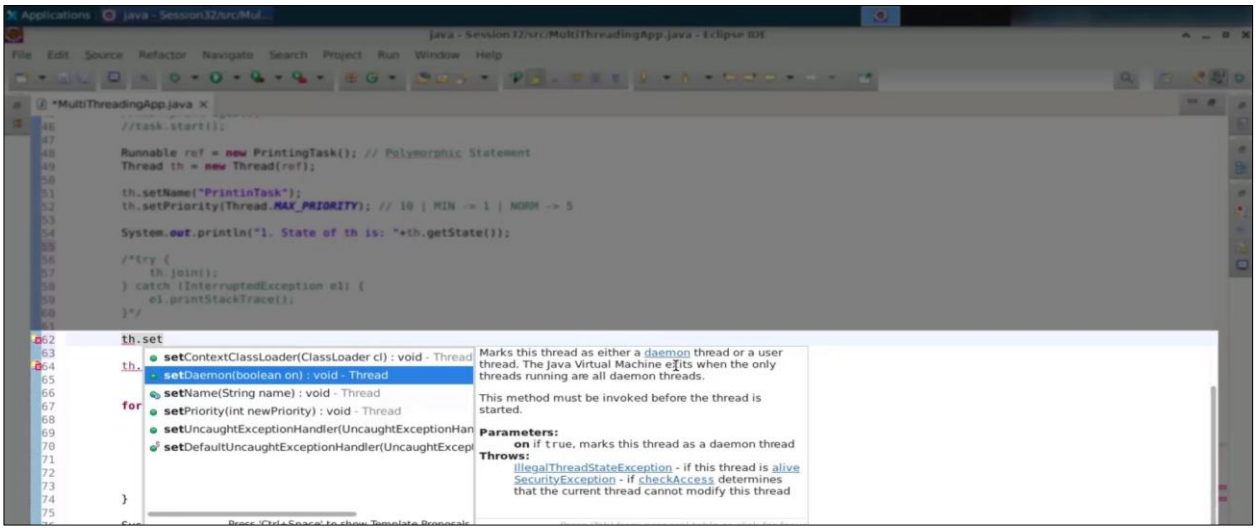


9.2 Then you have a state known as timed waiting. When you use the `Thread.sleep` method for a time interval, the state goes to time waiting. Another state, waiting, is achieved when you execute the `wait` method on the thread. There are various methods available for managing thread states.



Step 10: Mark the thread as a daemon method

10.1 Lastly, you can mark your thread as a daemon thread. When you do this, the Java Virtual Machine will exit if only daemon threads are running. You can mark a thread as a daemon by setting its daemon status to true. Additionally, you can explore more methods in multithreading for your Java application.



By using the following steps, you have successfully implemented multithreading in Java, enhancing your application's performance and responsiveness through concurrent execution of tasks.