

Lesson 04 Demo 10

Executing the Synchronization of Threads in Java

Objective: To demonstrate the concept of synchronization of threads in Java

Tools required: Eclipse IDE

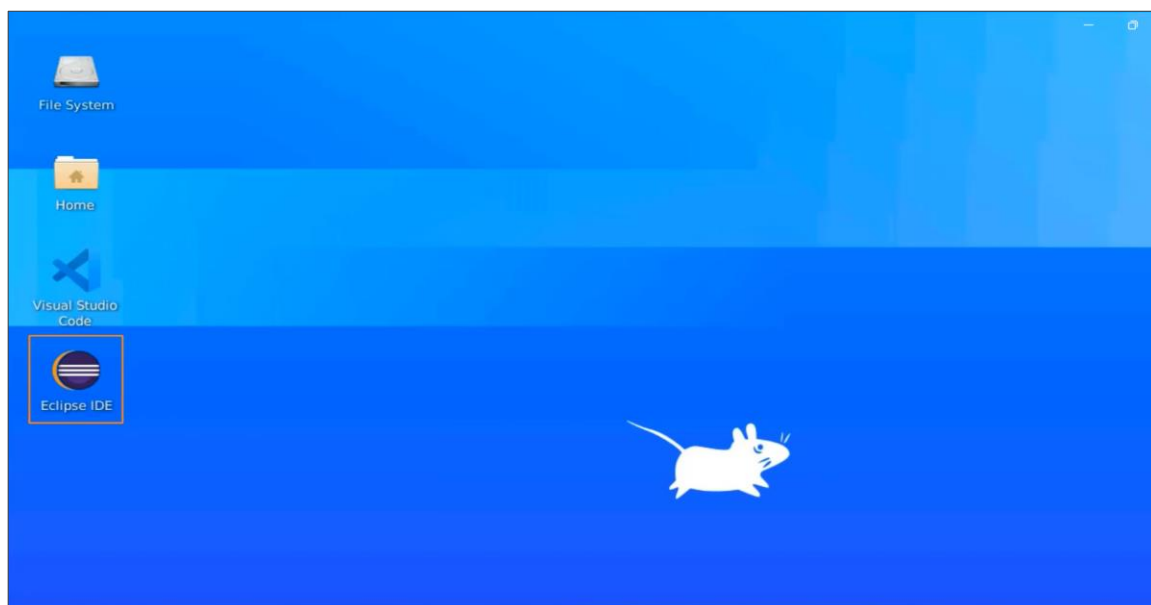
Prerequisites: None

Steps to be followed:

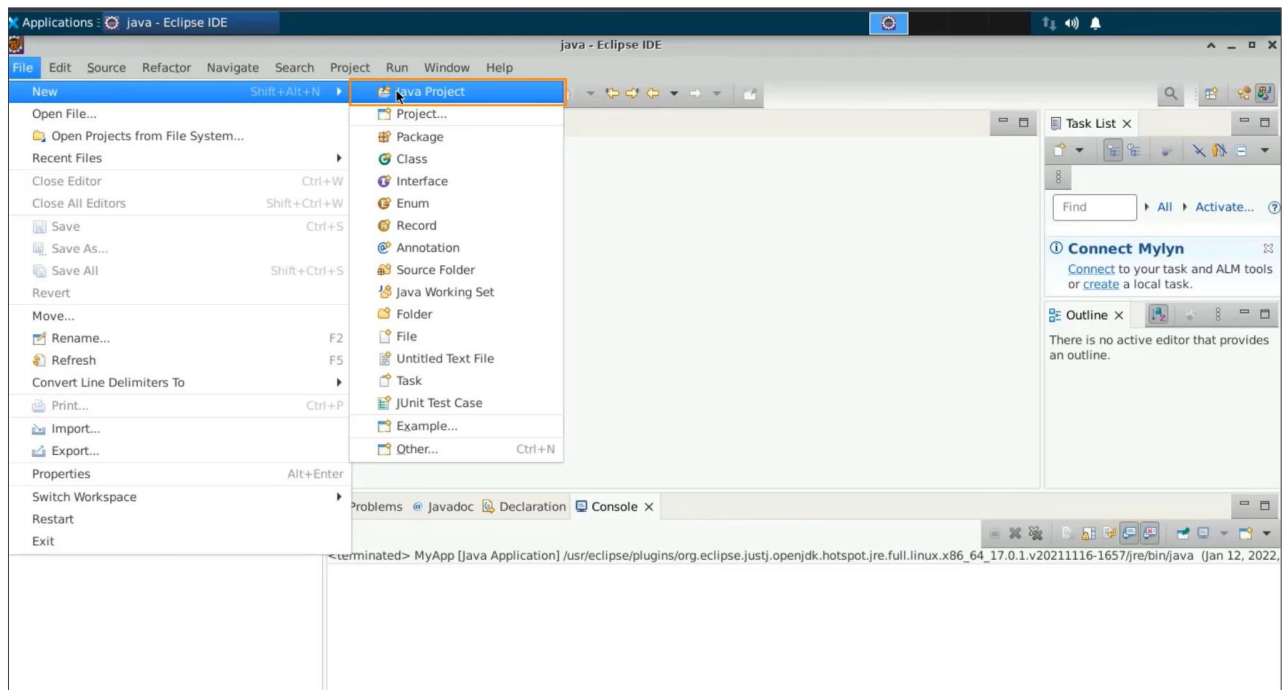
1. Open Eclipse IDE and create a new class
2. Write a for loop to print the document n number of times
3. Write a thread dot sleep, with the try catch
4. Create an object and execute the code
5. Create two threads and work on the same object
6. Execute the code with sample data
7. Implement the concept of synchronization

Step 1: Open Eclipse IDE and create a new class

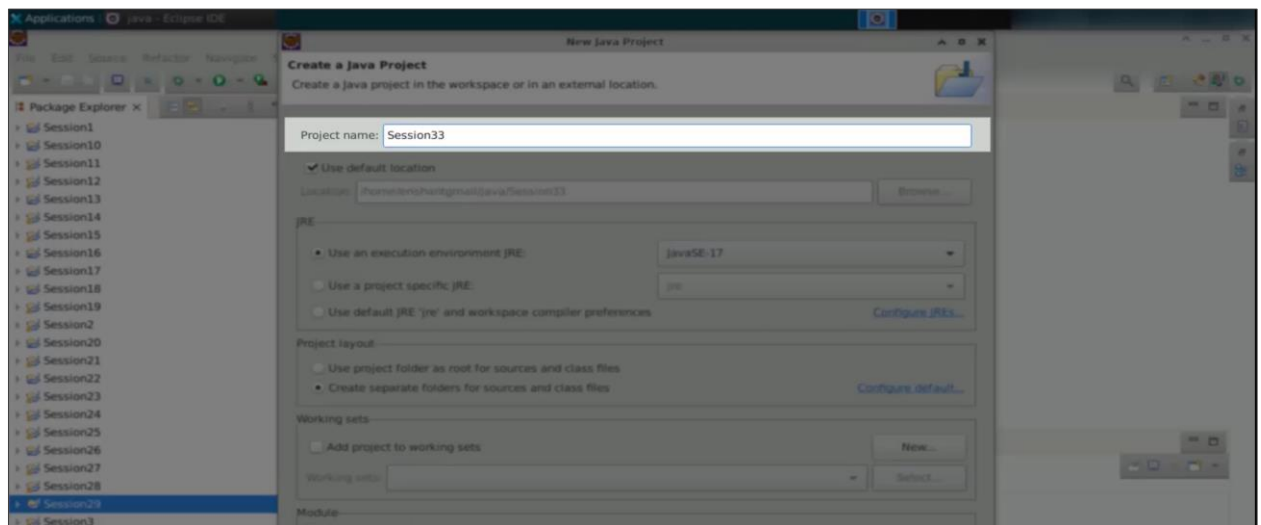
1.1 Open the Eclipse IDE



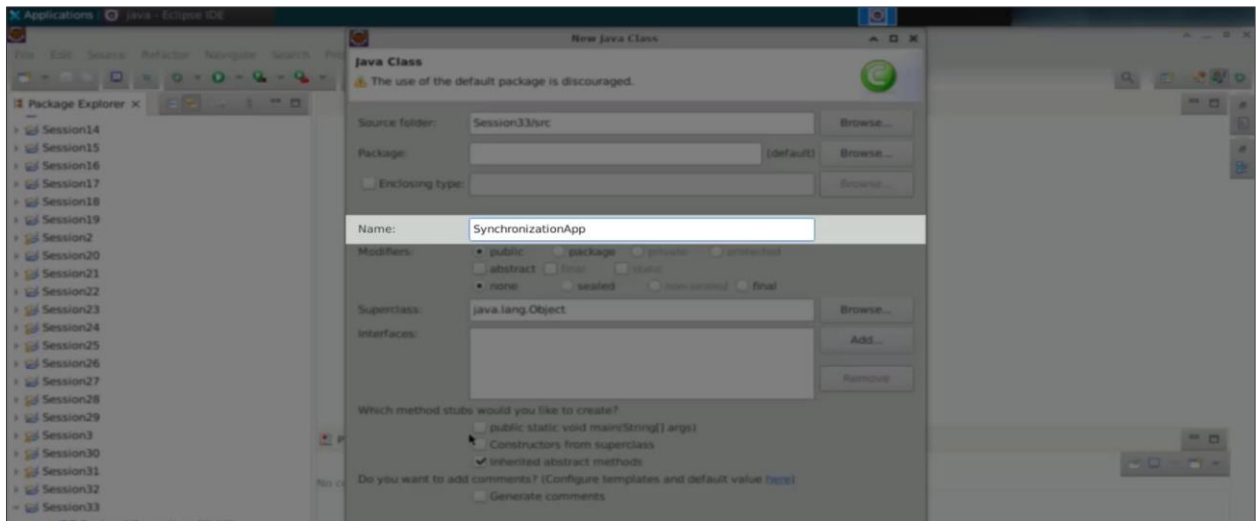
1.2 Select **File**, then **New**, and then **Java project**



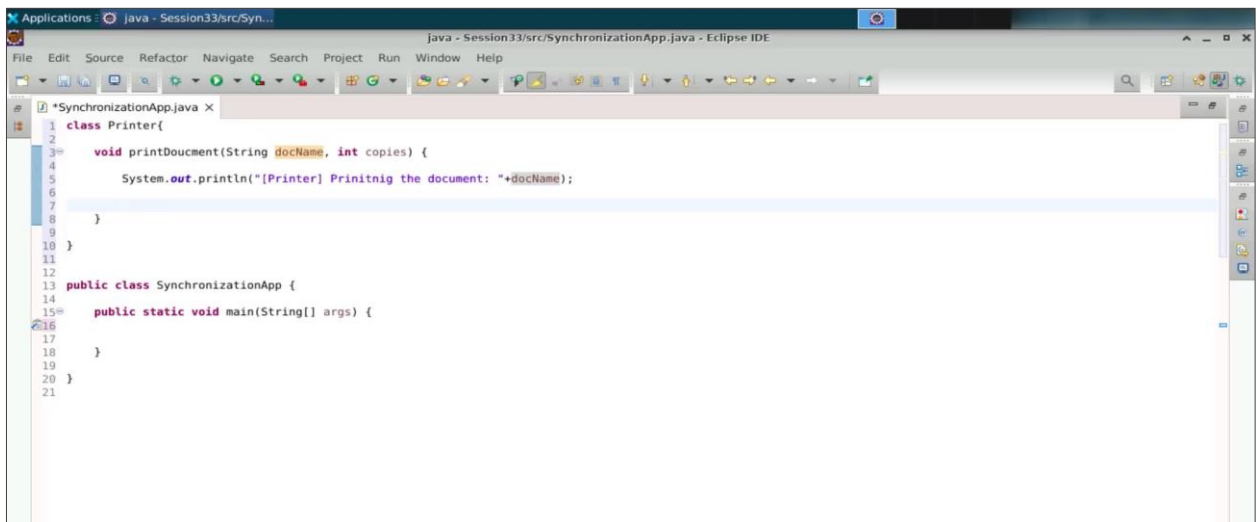
1.3 Name the project **“Session33”**, uncheck **“Create a module info.Java file”**, and press **Finish**



- 1.4 With a **Session33** on the src, do a right-click and create a **new class**. Name this class as an **SynchronizationApp**, then select the **main method**, and then select **finish**.

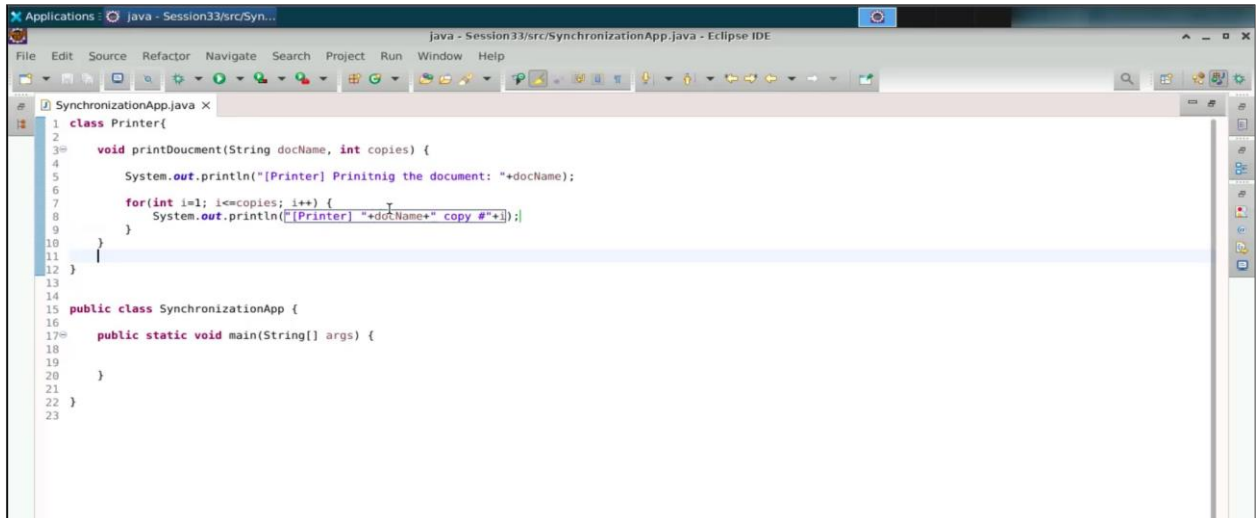


- 1.5 Create a class 'Printer', which can simulate printing documents on a real printer. For the 'Printer', create a method called 'print_document', which is a print command. This method will take a string as input for the document name and an integer for the number of copies you want for this document.



Step 2: Write a for loop to print the document n number of times

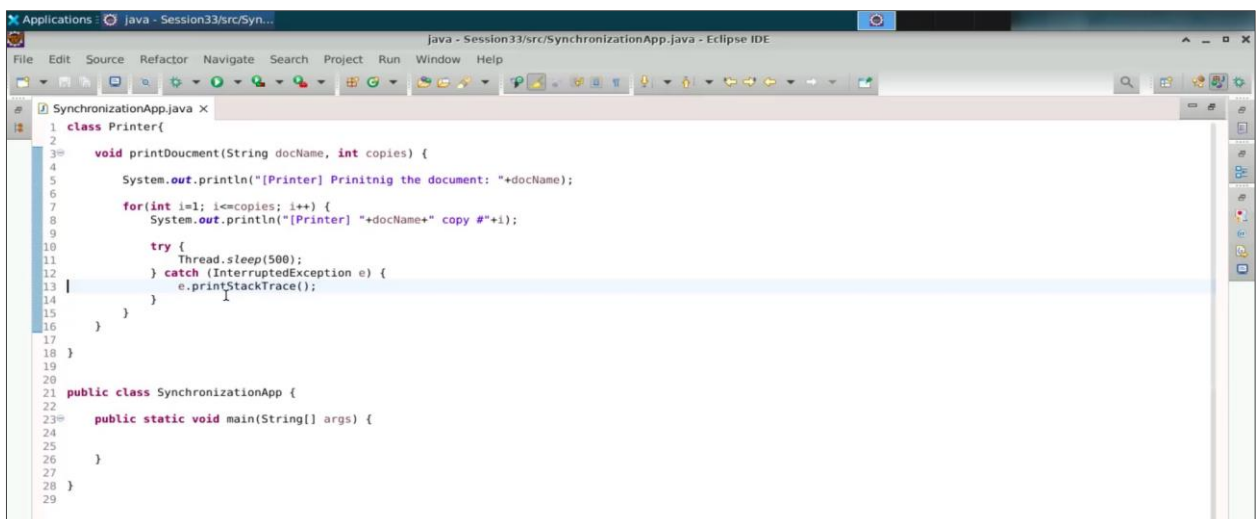
- 2.1 Next, create a for loop to print the document the specified number of times. Start with i at 1 and run the loop while i is less than or equal to the number of copies, incrementing i with each iteration. Print the document name followed by the copy number, +i. Since this also comes from your printer, include this information as well. With the Printer in action, you can now print the document with the specified copy number.



```
1 class Printer {
2
3     void printDocument(String docName, int copies) {
4         System.out.println("[Printer] Printnig the document: "+docName);
5
6         for(int i=1; i<=copies; i++) {
7             System.out.println("[Printer] "+docName+" copy #"+i);
8         }
9     }
10 }
11
12
13
14
15 public class SynchronizationApp {
16
17     public static void main(String[] args) {
18
19     }
20
21 }
22
23 }
```

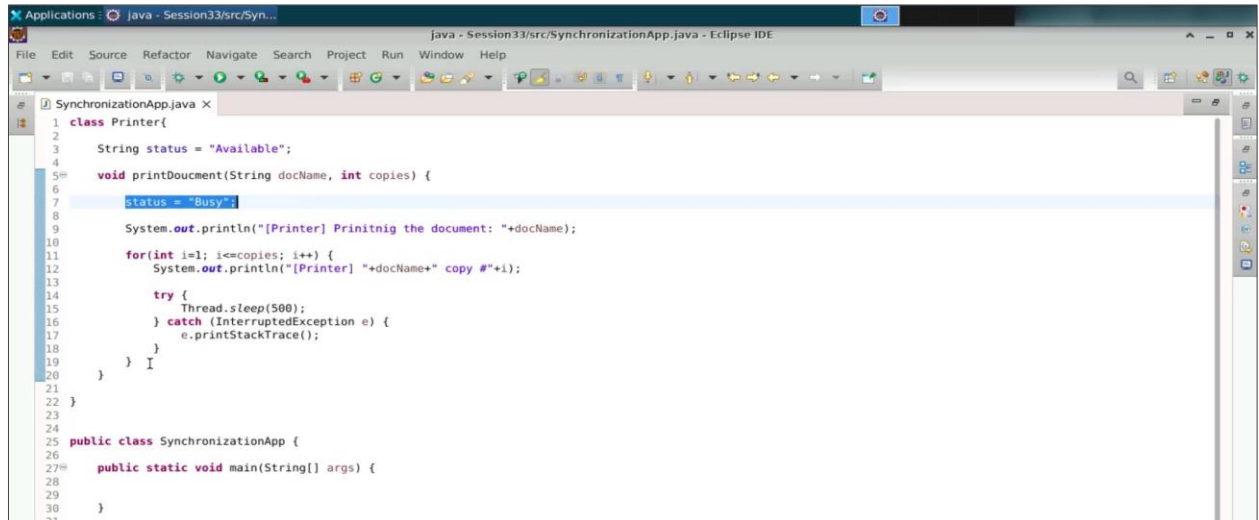
Step 3: Write a thread dot sleep, with the try catch

- 3.1 If you want, you can introduce a sleep delay by writing Thread.sleep(500). This will pause the thread for 500 milliseconds. Since the sleep method throws a checked exception, it is mandatory to surround this code with a try-catch block.



```
1 class Printer {
2
3     void printDocument(String docName, int copies) {
4         System.out.println("[Printer] Printnig the document: "+docName);
5
6         for(int i=1; i<=copies; i++) {
7             System.out.println("[Printer] "+docName+" copy #"+i);
8
9             try {
10                 Thread.sleep(500);
11             } catch (InterruptedException e) {
12                 e.printStackTrace();
13             }
14         }
15     }
16 }
17
18
19
20
21 public class SynchronizationApp {
22
23     public static void main(String[] args) {
24
25     }
26
27 }
28
29 }
```

- 3.2 You can take one attribute for the printer that is like the status, which as of now is available. And when you execute this command over here called print document, then you can give the status is as busy. Hence, you get the printer as not available.

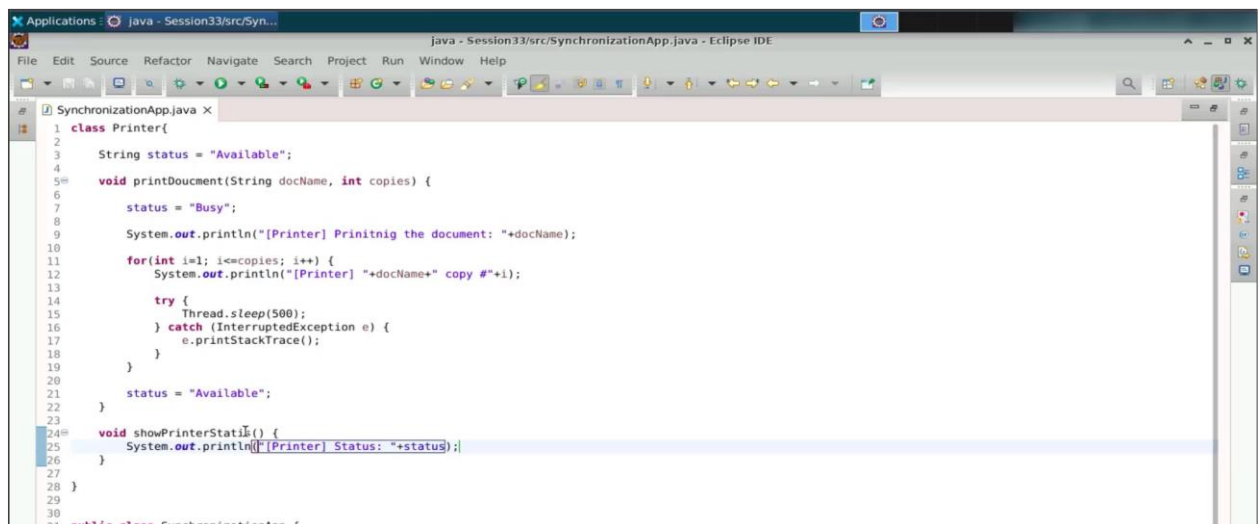


```

1  class Printer{
2
3      String status = "Available";
4
5      void printDocument(String docName, int copies) {
6
7          status = "Busy";
8
9          System.out.println("[Printer] Printnig the document: "+docName);
10
11          for(int i=1; i<=copies; i++) {
12              System.out.println("[Printer] "+docName+" copy #"+i);
13
14              try {
15                  Thread.sleep(500);
16              } catch (InterruptedException e) {
17                  e.printStackTrace();
18              }
19          }
20      }
21  }
22
23
24
25  public class SynchronizationApp {
26
27      public static void main(String[] args) {
28
29      }
30
31  }

```

- 3.3 When the documents are printed, this loop, once it terminates, will make the printer available again. You can add a method called show_printer_status to print out the status of the printer. This method will display the message "Printer status: available". You have simply printed the status for the printer in this method.



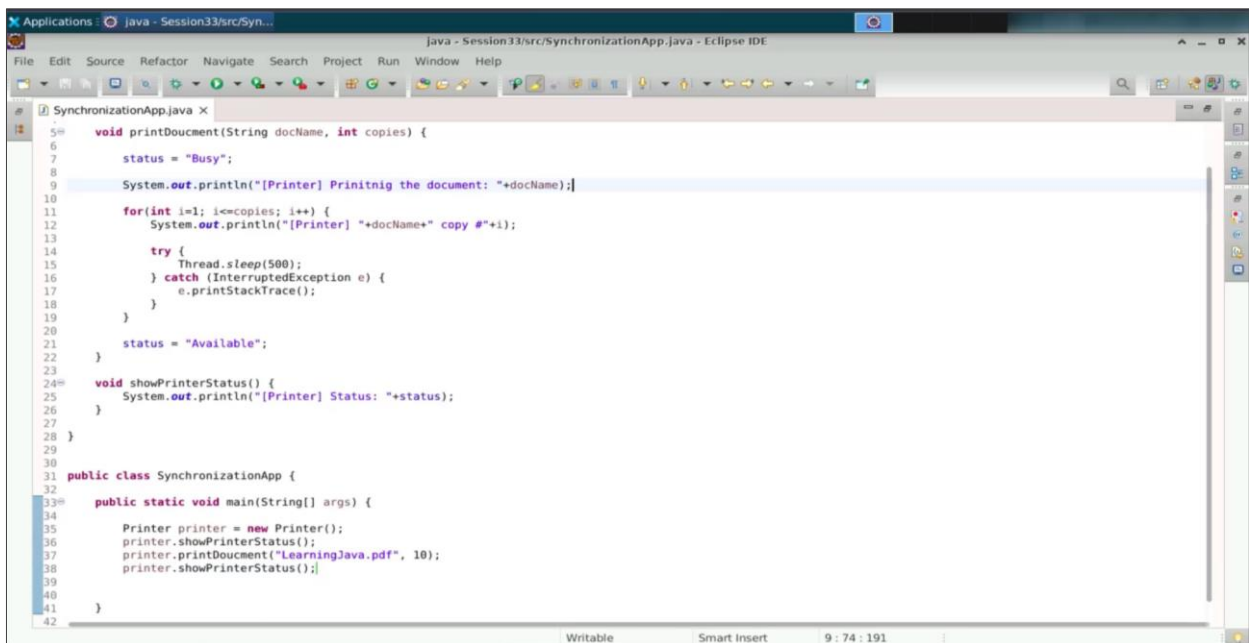
```

1  class Printer{
2
3      String status = "Available";
4
5      void printDocument(String docName, int copies) {
6
7          status = "Busy";
8
9          System.out.println("[Printer] Printnig the document: "+docName);
10
11          for(int i=1; i<=copies; i++) {
12              System.out.println("[Printer] "+docName+" copy #"+i);
13
14              try {
15                  Thread.sleep(500);
16              } catch (InterruptedException e) {
17                  e.printStackTrace();
18              }
19          }
20
21          status = "Available";
22      }
23
24      void showPrinterStatus() {
25          System.out.println("[Printer] Status: "+status);
26      }
27
28  }
29
30
31  public class SynchronizationApp {

```

Step 4: Create an object and execute the code

- 4.1 Let's create an object of Printer. Write it as `Printer printer = new Printer();`. If you want to print documents, it is a very simple command: call the `print_document` method on this printer object. The `print_document` method will take a document name, such as `"learning_java.pdf"`, and the number of copies, let us say 10 copies. Before this, let us read the status of the printer by calling `printer.show_printer_status()`. If you want to check the status of the printer at any point, you can simply execute the `show_printer_status` method.



```
1  void printDocument(String docName, int copies) {
2      status = "Busy";
3      System.out.println("[Printer] Printing the document: "+docName);
4      for(int i=1; i<=copies; i++) {
5          System.out.println("[Printer] "+docName+" copy #"+i);
6          try {
7              Thread.sleep(500);
8          } catch (InterruptedException e) {
9              e.printStackTrace();
10         }
11     }
12     status = "Available";
13 }
14
15 void showPrinterStatus() {
16     System.out.println("[Printer] Status: "+status);
17 }
18
19 public class SynchronizationApp {
20     public static void main(String[] args) {
21         Printer printer = new Printer();
22         printer.showPrinterStatus();
23         printer.printDocument("LearningJava.pdf", 10);
24         printer.showPrinterStatus();
25     }
26 }
```

4.2 When you run this code, this is very simple equation, the printer status is available. The printer says, printing the document and here the status goes as busy. Once, you get to see all the copies being printed, then, the status becomes available.

The screenshot shows the Eclipse IDE with the file `SynchronizationApp.java` open. The code defines a `printDocument` method that sets the status to "Busy", prints the document name, and then prints 10 copies of the document. A `showPrinterStatus` method is also defined. The console output shows the status changing from "Available" to "Busy" and then back to "Available" after printing 10 copies of "LearningJava.pdf".

```

5: void printDocument(String docName, int copies) {
6:     status = "Busy";
7:     System.out.println("[Printer] Printing the document: "+docName);
8:     showPrinterStatus();
9:     for(int i=1; i<=copies; i++) {
10:        System.out.println("[Printer] "+docName+" copy #"+i);
11:        try {
12:            Thread.sleep(500);
13:        } catch (InterruptedException e) {
14:            e.printStackTrace();
15:        }
16:        status = "Available";
17:    }
18: }
19: void showPrinterStatus() {
20:     System.out.println("[Printer] Status: "+status);
21: }
22: public class SynchronizationApp {
23:     public static void main(String[] args) {

```

```

[Printer] Status: Available
[Printer] Printing the document: LearningJava.pdf
[Printer] Status: Busy
[Printer] LearningJava.pdf copy #1
[Printer] LearningJava.pdf copy #2
[Printer] LearningJava.pdf copy #3
[Printer] LearningJava.pdf copy #4
[Printer] LearningJava.pdf copy #5
[Printer] LearningJava.pdf copy #6
[Printer] LearningJava.pdf copy #7
[Printer] LearningJava.pdf copy #8
[Printer] LearningJava.pdf copy #9
[Printer] LearningJava.pdf copy #10
[Printer] Status: Available

```

4.3 With this, over here, give as printing, copy #+i, just to make it a little different for the part.

The screenshot shows the Eclipse IDE with the file `SynchronizationApp.java` open. The code is similar to the previous one, but the line for printing the document name has been modified to include the copy number: `System.out.println("[Printer] Printing "+docName+" copy #"+i+"...");`.

```

5: void printDocument(String docName, int copies) {
6:     status = "Busy";
7:     System.out.println("[Printer] Printing the document: "+docName);
8:     showPrinterStatus();
9:     for(int i=1; i<=copies; i++) {
10:        System.out.println("[Printer] Printing "+docName+" copy #"+i+"...");
11:        try {
12:            Thread.sleep(500);
13:        } catch (InterruptedException e) {
14:            e.printStackTrace();
15:        }
16:        status = "Available";
17:    }
18: }
19: void showPrinterStatus() {
20:     System.out.println("[Printer] Status: "+status);
21: }
22: public class SynchronizationApp {
23:     public static void main(String[] args) {

```

4.4 Thus, this is the printing status as shown.

```

1  void printDocument(String docName, int copies) {
2      status = "Busy";
3      System.out.println("[Printer] Printing the document: "+docName);
4      showPrinterStatus();
5      for(int i=1; i<=copies; i++) {
6          System.out.println("[Printer] Printing "+docName+" copy #"+i+"...");
7          try {
8              Thread.sleep(500);
9          } catch (InterruptedException e) {
10             e.printStackTrace();
11          }
12      }
13      status = "Available";
14  }
15
16  void showPrinterStatus() {
17      System.out.println("[Printer] Status: "+status);
18  }
19
20  public class SynchronizationApp {
21
22      public static void main(String[] args) {

```

```

[Printer] Status: Available
[Printer] Printing the document: LearningJava.pdf
[Printer] Status: Busy
[Printer] Printing LearningJava.pdf copy #1...
[Printer] Printing LearningJava.pdf copy #2...
[Printer] Printing LearningJava.pdf copy #3...
[Printer] Printing LearningJava.pdf copy #4...
[Printer] Printing LearningJava.pdf copy #5...
[Printer] Printing LearningJava.pdf copy #6...
[Printer] Printing LearningJava.pdf copy #7...
[Printer] Printing LearningJava.pdf copy #8...
[Printer] Printing LearningJava.pdf copy #9...
[Printer] Printing LearningJava.pdf copy #10...
[Printer] Status: Available

```

Step 5: Create two threads and work on the same object

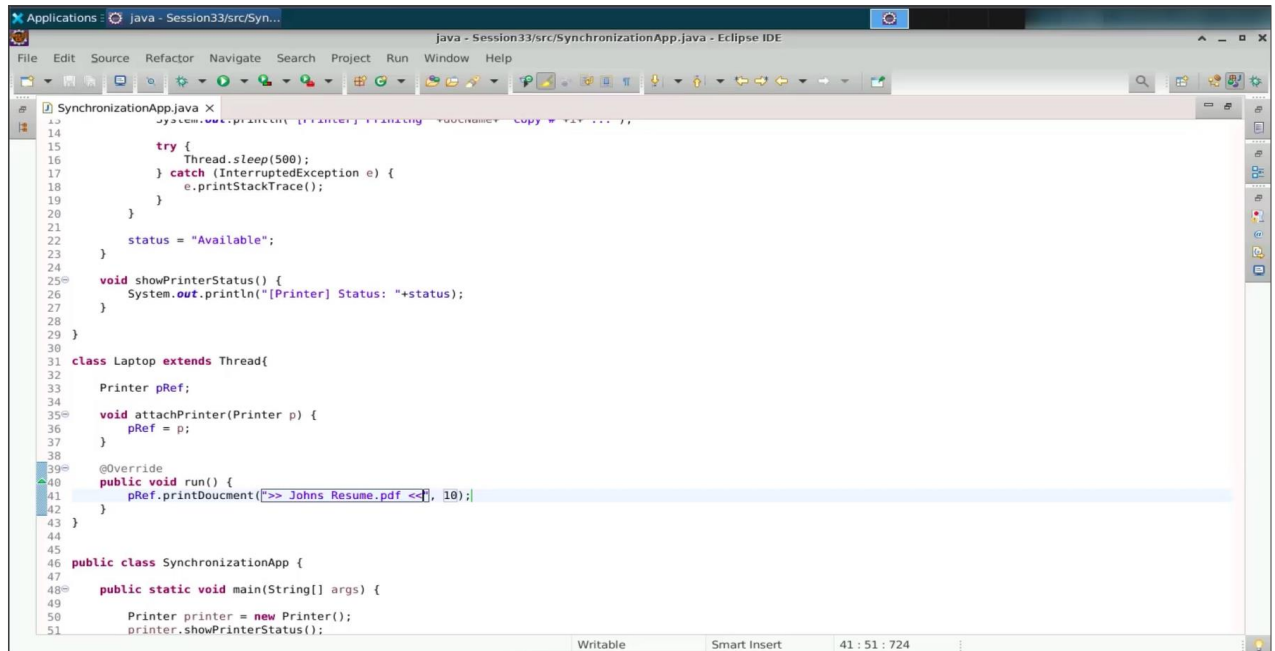
5.1 To create two threads that work on the same Printer object, you can create a class called Laptop that extends Thread. This class will have a reference variable pRef to the Printer object. You will create a method called attachPrinter that takes a printer object as an input and assigns it to the reference variable pRef.

```

14      try {
15          Thread.sleep(500);
16      } catch (InterruptedException e) {
17          e.printStackTrace();
18      }
19  }
20  }
21  status = "Available";
22  }
23  }
24
25  void showPrinterStatus() {
26      System.out.println("[Printer] Status: "+status);
27  }
28  }
29  }
30
31  class Laptop extends Thread{
32      Printer pRef;
33
34      void attachPrinter(Printer p) {
35          pRef = p;
36      }
37  }
38
39
40
41  public class SynchronizationApp {
42
43      public static void main(String[] args) {
44          Printer printer = new Printer();
45          printer.showPrinterStatus();
46          printer.printDocument("LearningJava.pdf", 10);
47          printer.showPrinterStatus();
48      }
49  }
50
51

```

- 5.2 To override the run method in the laptop class, you need to ensure that it calls the print_document method from the Printer object. In this method, you will print a document named "John's Resume.pdf" and specify 10 copies.

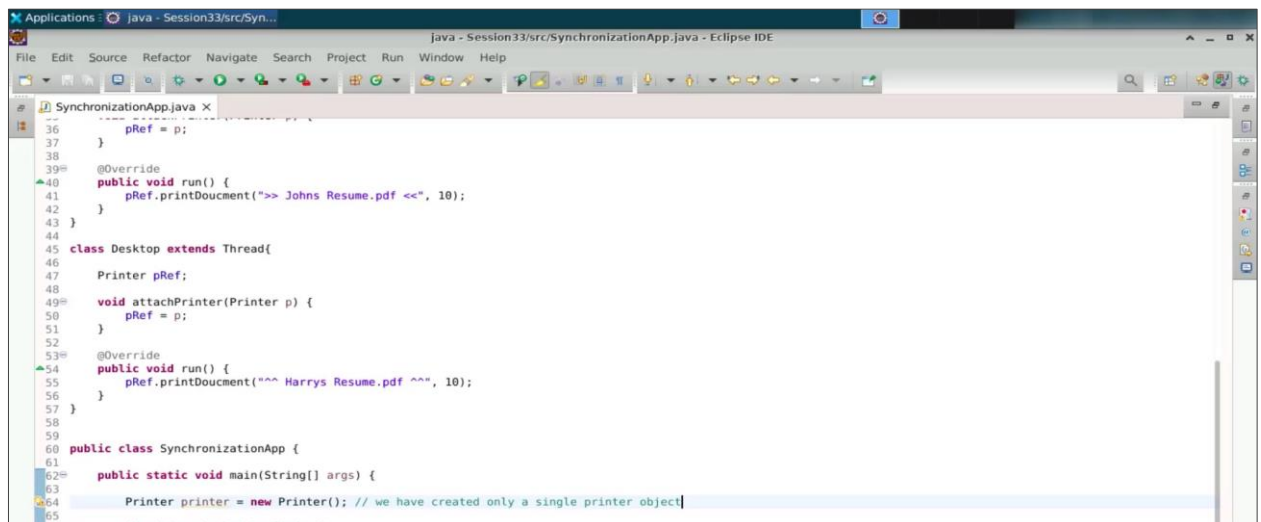


```

14      try {
15          Thread.sleep(500);
16      } catch (InterruptedException e) {
17          e.printStackTrace();
18      }
19  }
20  }
21  status = "Available";
22  }
23  }
24  }
25  void showPrinterStatus() {
26      System.out.println("[Printer] Status: "+status);
27  }
28  }
29  }
30  }
31  class Laptop extends Thread{
32      Printer pRef;
33      void attachPrinter(Printer p) {
34          pRef = p;
35      }
36  }
37  }
38  }
39  @Override
40  public void run() {
41      pRef.printDocument(">> Johns Resume.pdf <<", 10);
42  }
43  }
44  }
45  }
46  public class SynchronizationApp {
47      public static void main(String[] args) {
48          Printer printer = new Printer();
49          printer.showPrinterStatus();
50      }
51  }

```

- 5.3 To achieve this, we will follow the same structure for the Desktop class as we did for the laptop class. The Desktop class will also extend Thread and override the run method to print "Harry's Resume.pdf" 10 times. Both the Laptop and Desktop classes will use the same Printer object, ensuring they share this resource.

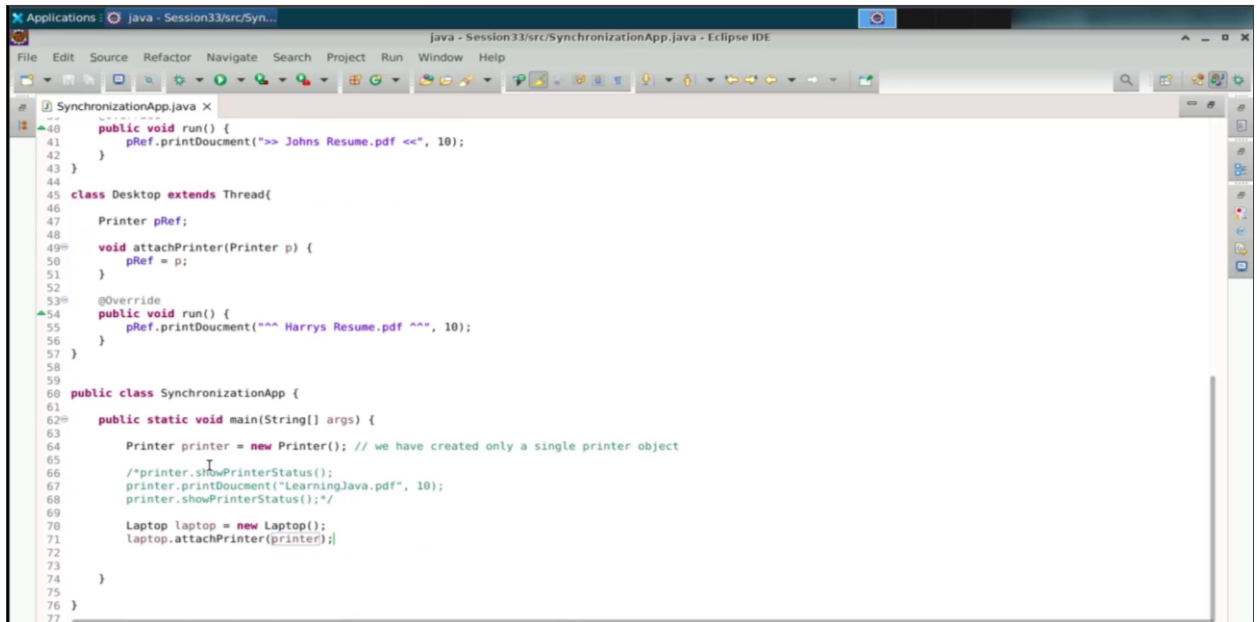


```

36      pRef = p;
37  }
38  }
39  @Override
40  public void run() {
41      pRef.printDocument(">> Johns Resume.pdf <<", 10);
42  }
43  }
44  }
45  }
46  class Desktop extends Thread{
47      Printer pRef;
48      void attachPrinter(Printer p) {
49          pRef = p;
50      }
51  }
52  }
53  @Override
54  public void run() {
55      pRef.printDocument("^ Harrys Resume.pdf ^", 10);
56  }
57  }
58  }
59  }
60  public class SynchronizationApp {
61      public static void main(String[] args) {
62          Printer printer = new Printer(); // we have created only a single printer object
63      }
64  }
65  }

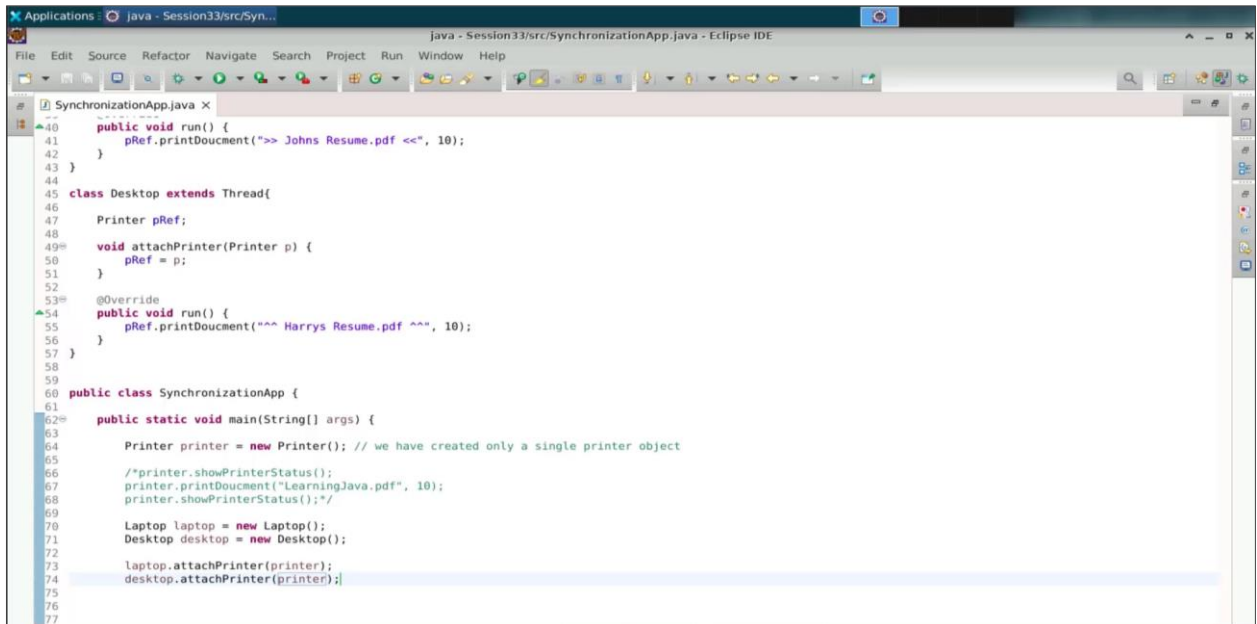
```

- 5.4 To create the laptop object, write `Laptop laptop = new Laptop();`. Next, attach the `Printer` object to the laptop by calling `laptop.attach_printer(printer);`. This action copies the reference variable `printer` into the `pRef` of the laptop, ensuring that both `pRef` and `printer` point to the same `Printer` object. This way, the laptop can utilize the `Printer` for its print tasks.



```
1  SynchronizationApp.java
2
3  public void run() {
4      pRef.printDocument(">> Johns Resume.pdf <<", 10);
5  }
6
7  }
8
9  class Desktop extends Thread{
10     Printer pRef;
11
12     void attachPrinter(Printer p) {
13         pRef = p;
14     }
15
16     @Override
17     public void run() {
18         pRef.printDocument("^ ^ Harrys Resume.pdf ^ ^", 10);
19     }
20 }
21
22 public class SynchronizationApp {
23     public static void main(String[] args) {
24         Printer printer = new Printer(); // we have created only a single printer object
25
26         /*printer.showPrinterStatus();
27         printer.printDocument("LearningJava.pdf", 10);
28         printer.showPrinterStatus();*/
29
30         Laptop laptop = new Laptop();
31         laptop.attachPrinter(printer);
32     }
33 }
34 }
```

- 5.5 Next, create the Desktop object by writing `Desktop desktop = new Desktop();`. Illustrate better usage, create both the Laptop and Desktop objects. Then, attach the same Printer to the Desktop by calling `desktop.attach_printer(printer);`. This demonstrates that both the Laptop and the Desktop are working with the same Printer



```
1  SynchronizationApp.java x
2
3  java - Session33/src/SynchronizationApp.java - Eclipse IDE
4
5  File Edit Source Refactor Navigate Search Project Run Window Help
6
7  # SynchronizationApp.java x
8
9  40 public void run() {
10     41     pRef.printDocument(">> Johns Resume.pdf <<", 10);
11     42 }
12     43 }
13     44
14     45 class Desktop extends Thread{
15     46
16     47     Printer pRef;
17     48
18     49 void attachPrinter(Printer p) {
19     50     pRef = p;
20     51 }
21     52
22     53 @Override
23     54 public void run() {
24     55     pRef.printDocument("^ ^ Harrys Resume.pdf ^ ^", 10);
25     56 }
26     57 }
27     58
28     59 public class SynchronizationApp {
29     60
30     61     public static void main(String[] args) {
31     62
32     63         Printer printer = new Printer(); // we have created only a single printer object
33     64
34     65         /*printer.showPrinterStatus();
35     66         printer.printDocument("LearningJava.pdf", 10);
36     67         printer.showPrinterStatus();*/
37     68
38     69         Laptop laptop = new Laptop();
39     70         Desktop desktop = new Desktop();
40     71
41     72         laptop.attachPrinter(printer);
42     73         desktop.attachPrinter(printer);
43     74
44     75
45     76
46     77
```

Step 6: Execute the code with sample data

6.1 Consider this real-world scenario: you have a desktop and a laptop at home, both sharing a single printer. You attach both your desktop and laptop to the same printer. On the laptop, you want to print John's resume, and on the desktop, you want to print Harry's resume, each in 10 copies. You then start both threads by writing `laptop.start()` and `desktop.start()`. This setup allows both the laptop and desktop to print their respective documents concurrently using the same printer.

```

1  Applications  java - Session33/src/Syn...
2  java - Session33/src/SynchronizationApp.java - Eclipse IDE
3  File Edit Source Refactor Navigate Search Project Run Window Help
4
5  SynchronizationApp.java x
6
7  45 class Desktop extends Thread{
8
9  46     Printer pRef;
10
11  47     void attachPrinter(Printer p) {
12
13  48         pRef = p;
14
15  49     }
16
17  50
18  51
19  52
20  53 @Override
21  54 public void run() {
22  55     pRef.printDocument("^^ Harrys Resume.pdf ^^", 10);
23  56 }
24
25  57
26  58
27  59 public class SynchronizationApp {
28
29  60     public static void main(String[] args) {
30
31  61         Printer printer = new Printer(); // we have created only a single printer object
32
33  62
34  63         /*printer.showPrinterStatus();
35  64         printer.printDocument("LearningJava.pdf", 10);
36  65         printer.showPrinterStatus();*/
37
38  66
39  67         Laptop laptop = new Laptop();
40  68         Desktop desktop = new Desktop();
41
42  69
43  70         laptop.attachPrinter(printer);
44  71         desktop.attachPrinter(printer);
45
46  72
47  73         laptop.start();
48  74         desktop.start();
49
50  75
51  76
52  77
53  78
54  79
55  80
56  81 }
57  82

```

6.2 When you run the code, it shows, that John and Harry's resume is printing together. Sometimes, it prints a copy of John's resume and sometimes it prints the copy of Harry's resume.

```

1  Applications  java - Session33/src/Syn...
2  java - Session33/src/SynchronizationApp.java - Eclipse IDE
3  File Edit Source Refactor Navigate Search Project Run Window Help
4
5  Synchronization Run SynchronizationApp
6
7  45 class Desktop extends Thread{
8
9  46     Printer pRef;
10
11  47     void attachPrinter(Printer p) {
12
13  48         pRef = p;
14
15  49     }
16
17  50
18  51
19  52
20  53 @Override
21  54 public void run() {
22  55     pRef.printDocument("^^ Harrys Resume.pdf ^^", 10);
23  56 }
24
25  57
26  58
27  59 public class SynchronizationApp {
28
29  60     public static void main(String[] args) {
30
31  61         Printer printer = new Printer(); // we have created only a single printer object
32
33  62
34  63         /*printer.showPrinterStatus();
35  64         printer.printDocument("LearningJava.pdf", 10);
36  65         printer.showPrinterStatus();*/
37
38  66
39  67         Laptop laptop = new Laptop();
40  68         Desktop desktop = new Desktop();
41
42  69
43  70         laptop.attachPrinter(printer);
44  71         desktop.attachPrinter(printer);
45
46  72
47  73
48  74
49  75
50  76
51  77
52  78
53  79
54  80
55  81
56  82

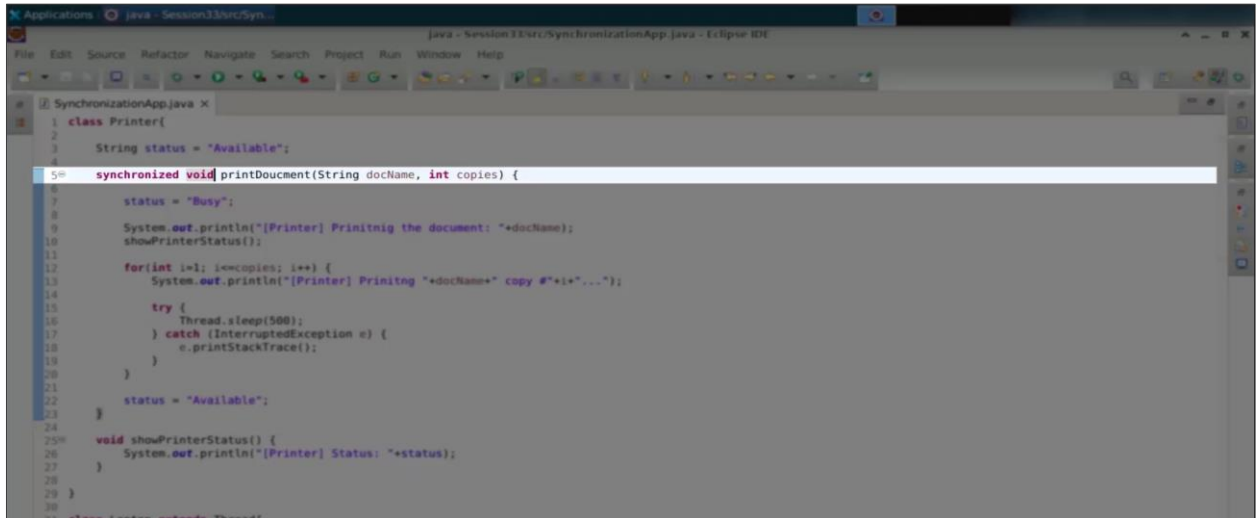
```

```

[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #2...
[Printer] Printing ^^ Johns Resume.pdf ^^ copy #3...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #3...
[Printer] Printing ^^ Johns Resume.pdf ^^ copy #4...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #4...
[Printer] Printing ^^ Johns Resume.pdf ^^ copy #5...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #5...
[Printer] Printing ^^ Johns Resume.pdf ^^ copy #6...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #6...
[Printer] Printing ^^ Johns Resume.pdf ^^ copy #7...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #7...
[Printer] Printing ^^ Johns Resume.pdf ^^ copy #8...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #8...
[Printer] Printing ^^ Johns Resume.pdf ^^ copy #9...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #9...
[Printer] Printing ^^ Johns Resume.pdf ^^ copy #10...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #10...

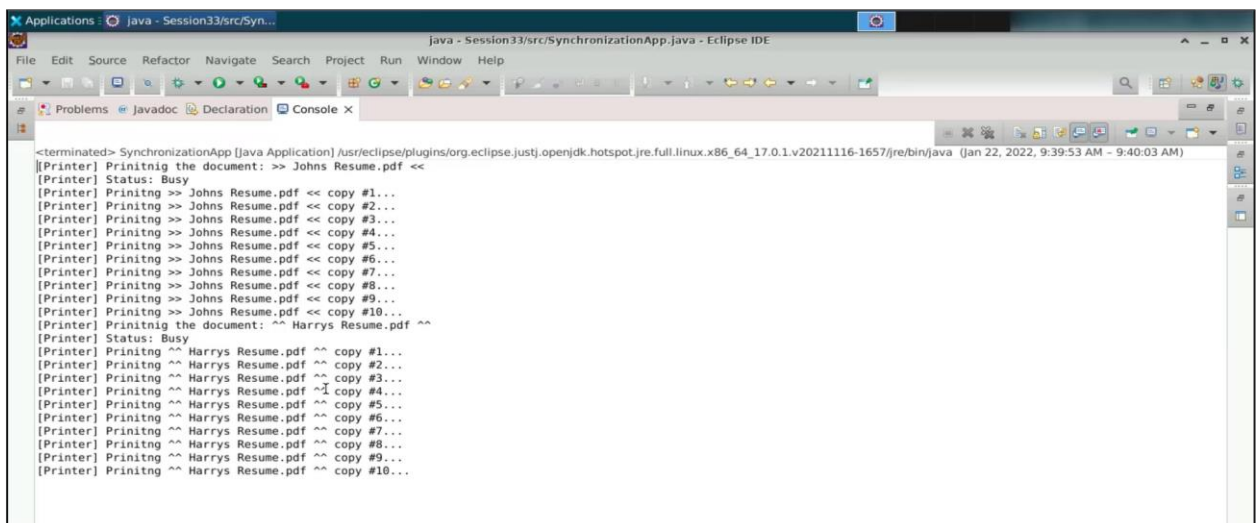
```

6.3 Let's come here and first mark the printer to go as synchronized. When the method is marked as synchronized, it particularly means, the other thread cannot execute the same method in the same object till time the first thread has not finished its execution.



```
1 class Printer{
2
3     String status = "Available";
4
5     synchronized void printDocument(String docName, int copies) {
6
7         status = "Busy";
8
9         System.out.println("[Printer] Printing the document: "+docName);
10        showPrinterStatus();
11
12        for(int i=1; i<=copies; i++){
13            System.out.println("[Printer] Printing "+docName+" copy #"+i+"...");
14
15            try {
16                Thread.sleep(500);
17            } catch (InterruptedException e) {
18                e.printStackTrace();
19            }
20        }
21        status = "Available";
22    }
23
24    void showPrinterStatus() {
25        System.out.println("[Printer] Status: "+status);
26    }
27
28 }
29
30 // class Printer extends Thread
```

6.4 Let us run the code, to understand this discussion better. As shown only John's resume is getting printed and thereafter Harry's resume is getting printed. That is where you understand the fundamental of synchronization.



```
<terminated> SynchronizationApp [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, 9:39:53 AM - 9:40:03 AM)
[Printer] Printing the document: >> Johns Resume.pdf <<
[Printer] Status: Busy
[Printer] Printing >> Johns Resume.pdf << copy #1...
[Printer] Printing >> Johns Resume.pdf << copy #2...
[Printer] Printing >> Johns Resume.pdf << copy #3...
[Printer] Printing >> Johns Resume.pdf << copy #4...
[Printer] Printing >> Johns Resume.pdf << copy #5...
[Printer] Printing >> Johns Resume.pdf << copy #6...
[Printer] Printing >> Johns Resume.pdf << copy #7...
[Printer] Printing >> Johns Resume.pdf << copy #8...
[Printer] Printing >> Johns Resume.pdf << copy #9...
[Printer] Printing >> Johns Resume.pdf << copy #10...
[Printer] Printing the document: ^^ Harrys Resume.pdf ^^
[Printer] Status: Busy
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #1...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #2...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #3...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #4...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #5...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #6...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #7...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #8...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #9...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #10...
```

6.5 Let's give the printer status once again and make it to available. Then, come here and do one empty print line, with a `\n`.

```

14
15     try {
16         Thread.sleep(500);
17     } catch (InterruptedException e) {
18         e.printStackTrace();
19     }
20 }
21
22 status = "Available";
23 showPrinterStatus();
24 }
25
26 void showPrinterStatus() {
27     System.out.println("[Printer] Status: "+status+"\n");
28 }
29
30 }
31
32 class Laptop extends Thread{
33     Printer pRef;
34
35     void attachPrinter(Printer p) {
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

SynchronizationApp [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, 9:41:21 AM)

[Printer] Printnig the document: >> Johns Resume.pdf <<
[Printer] Status: Busy

6.6 Rerun the code and check the output. It shows, the printer status as busy and now it is made available, hence printing the document with the Harry's resume status went busy that is, it started printing again. And now it is available again.

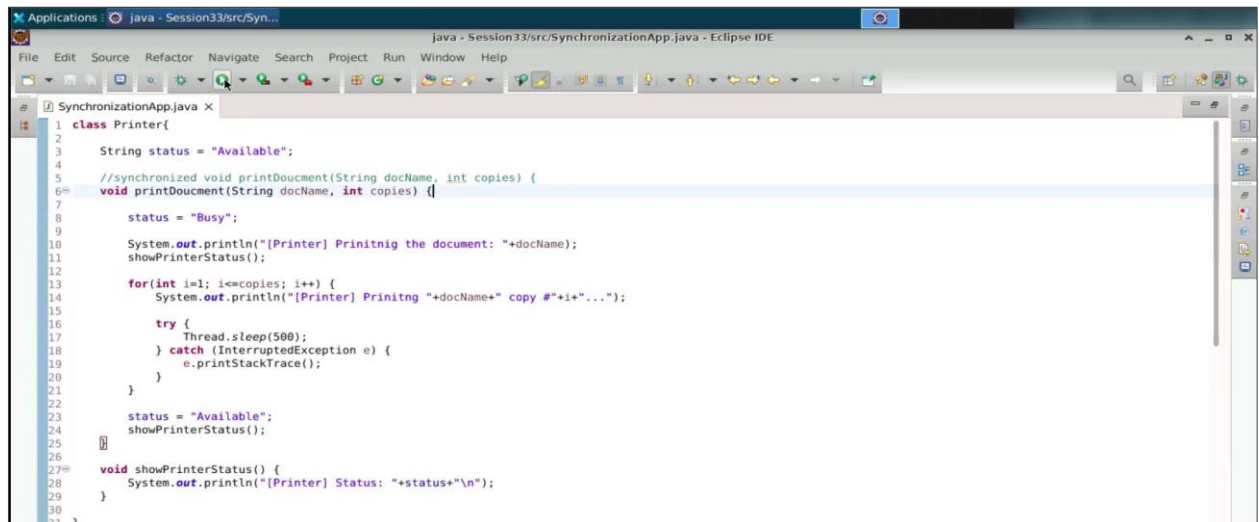
```

SynchronizationApp [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, 9:41:21 AM)
[Printer] Printnig the document: >> Johns Resume.pdf <<
[Printer] Status: Busy
[Printer] Printntng >> Johns Resume.pdf << copy #1...
[Printer] Printntng >> Johns Resume.pdf << copy #2...
[Printer] Printntng >> Johns Resume.pdf << copy #3...
[Printer] Printntng >> Johns Resume.pdf << copy #4...
[Printer] Printntng >> Johns Resume.pdf << copy #5...
[Printer] Printntng >> Johns Resume.pdf << copy #6...
[Printer] Printntng >> Johns Resume.pdf << copy #7...
[Printer] Printntng >> Johns Resume.pdf << copy #8...
[Printer] Printntng >> Johns Resume.pdf << copy #9...
[Printer] Printntng >> Johns Resume.pdf << copy #10...
[Printer] Status: Available
[Printer] Printntng the document: ^^ Harrys Resume.pdf ^^
[Printer] Status: Busy
[Printer] Printntng ^^ Harrys Resume.pdf ^^ copy #1...
[Printer] Printntng ^^ Harrys Resume.pdf ^^ copy #2...
[Printer] Printntng ^^ Harrys Resume.pdf ^^ copy #3...
[Printer] Printntng ^^ Harrys Resume.pdf ^^ copy #4...
[Printer] Printntng ^^ Harrys Resume.pdf ^^ copy #5...
[Printer] Printntng ^^ Harrys Resume.pdf ^^ copy #6...
[Printer] Printntng ^^ Harrys Resume.pdf ^^ copy #7...
[Printer] Printntng ^^ Harrys Resume.pdf ^^ copy #8...
[Printer] Printntng ^^ Harrys Resume.pdf ^^ copy #9...

```

Step 7: Implement the concept of synchronization

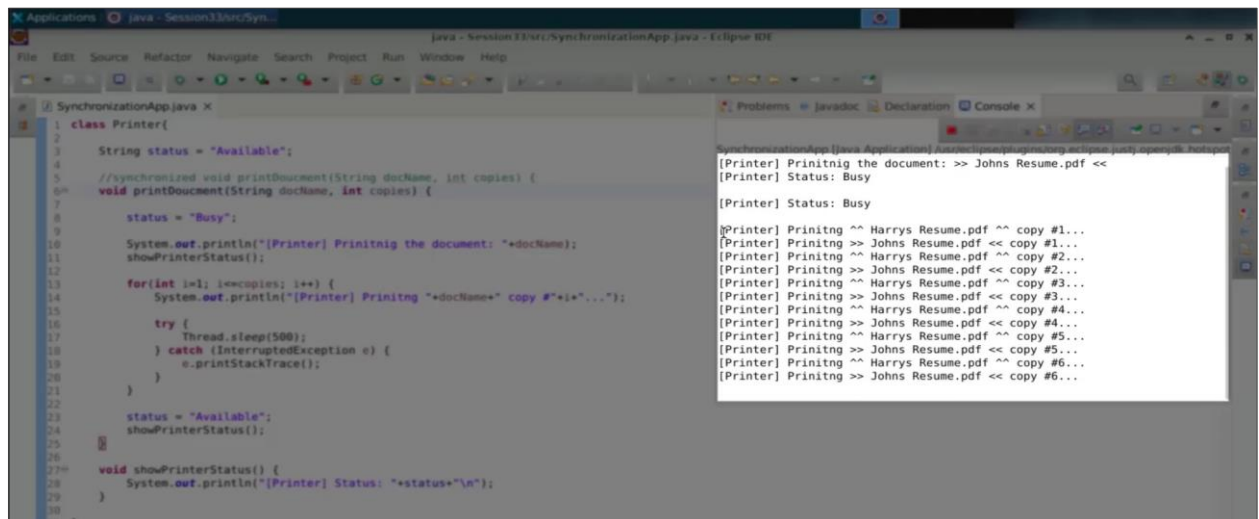
7.1 Synchronize as a keyword can be used on the methods, which is the first thing when you want to implement synchronization.



```

1 class Printer{
2
3     String status = "Available";
4
5     //synchronized void printDocument(String docName, int copies) {
6     void printDocument(String docName, int copies) {
7
8         status = "Busy";
9
10        System.out.println("[Printer] Printinig the document: "+docName);
11        showPrinterStatus();
12
13        for(int i=1; i<=copies; i++) {
14            System.out.println("[Printer] Printinig "+docName+" copy #"+i+"...");
15
16            try {
17                Thread.sleep(500);
18            } catch (InterruptedException e) {
19                e.printStackTrace();
20            }
21        }
22
23        status = "Available";
24        showPrinterStatus();
25    }
26
27    void showPrinterStatus() {
28        System.out.println("[Printer] Status: "+status+"\n");
29    }
30
31 }
  
```

7.2 The other thing is let your method be working in the same way as a regular method. This means that both the threads will now access it in parallel. Thus, the printer is again messed up.



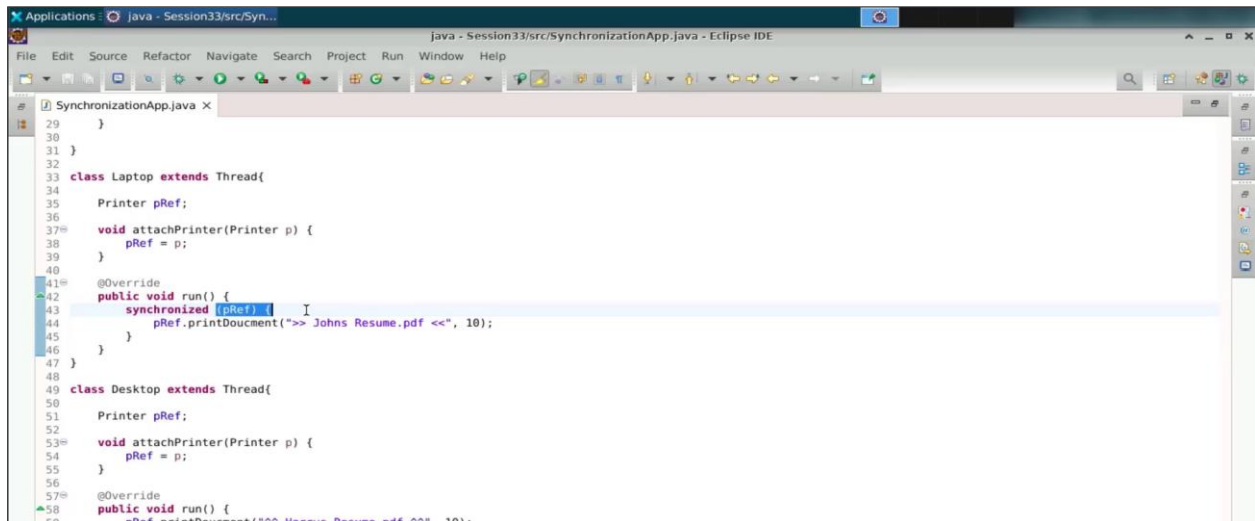
```

1 class Printer{
2
3     String status = "Available";
4
5     //synchronized void printDocument(String docName, int copies) {
6     void printDocument(String docName, int copies) {
7
8         status = "Busy";
9
10        System.out.println("[Printer] Printinig the document: "+docName);
11        showPrinterStatus();
12
13        for(int i=1; i<=copies; i++) {
14            System.out.println("[Printer] Printinig "+docName+" copy #"+i+"...");
15
16            try {
17                Thread.sleep(500);
18            } catch (InterruptedException e) {
19                e.printStackTrace();
20            }
21        }
22
23        status = "Available";
24        showPrinterStatus();
25    }
26
27    void showPrinterStatus() {
28        System.out.println("[Printer] Status: "+status+"\n");
29    }
30
31 }
  
```

```

[Printer] Printinig the document: >> Johns Resume.pdf <<
[Printer] Status: Busy
[Printer] Status: Busy
[Printer] Printinig ^ Harrys Resume.pdf ^ copy #1...
[Printer] Printinig >> Johns Resume.pdf << copy #1...
[Printer] Printinig ^ Harrys Resume.pdf ^ copy #2...
[Printer] Printinig >> Johns Resume.pdf << copy #2...
[Printer] Printinig ^ Harrys Resume.pdf ^ copy #3...
[Printer] Printinig >> Johns Resume.pdf << copy #3...
[Printer] Printinig ^ Harrys Resume.pdf ^ copy #4...
[Printer] Printinig >> Johns Resume.pdf << copy #4...
[Printer] Printinig ^ Harrys Resume.pdf ^ copy #5...
[Printer] Printinig >> Johns Resume.pdf << copy #5...
[Printer] Printinig ^ Harrys Resume.pdf ^ copy #6...
[Printer] Printinig >> Johns Resume.pdf << copy #6...
  
```

7.3 With synchronization to work in one more way, in the run method, you can synchronize the object itself. You can use the synchronized block and for the mutex on which you want to acquire the lock is the P ref. In this way, none of the other thread can access this object pointed by the P ref, which is like the printer object is now non accessible till time this synchronized block is exited.

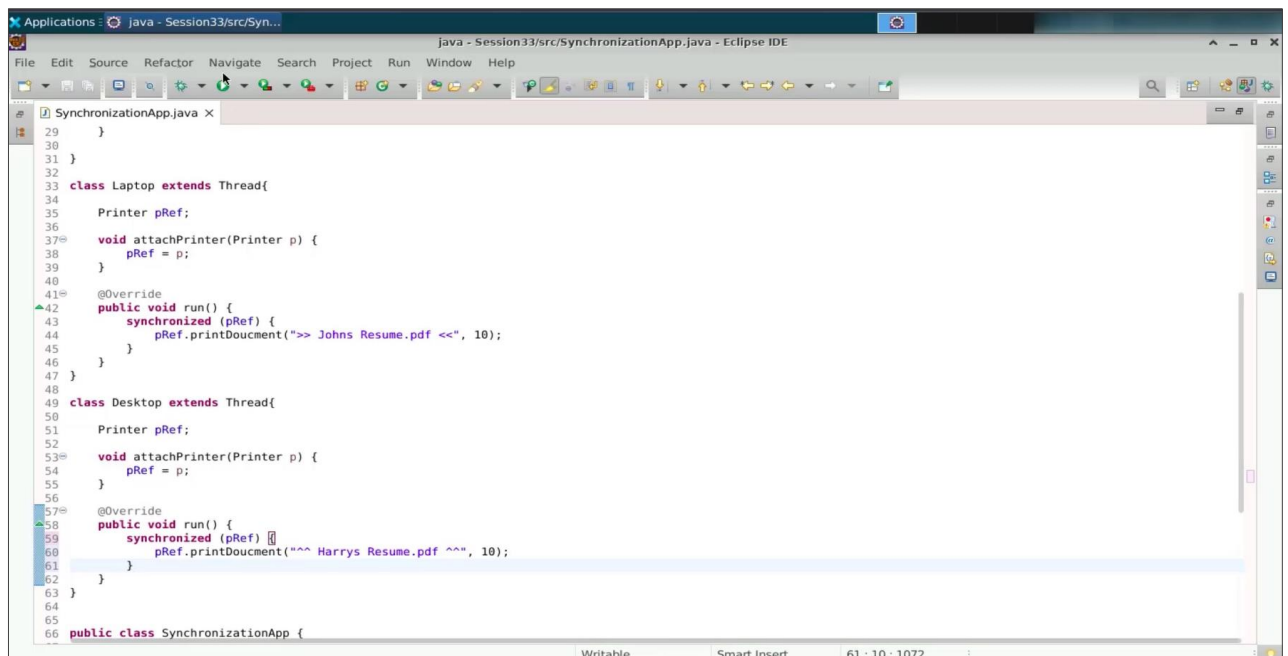


```

29 }
30 }
31 }
32
33 class Laptop extends Thread{
34     Printer pRef;
35
36     void attachPrinter(Printer p) {
37         pRef = p;
38     }
39 }
40
41 @Override
42 public void run() {
43     synchronized (pRef) {
44         pRef.printDocument(">> Johns Resume.pdf <<", 10);
45     }
46 }
47 }
48
49 class Desktop extends Thread{
50     Printer pRef;
51
52     void attachPrinter(Printer p) {
53         pRef = p;
54     }
55 }
56
57 @Override
58 public void run() {
59     pRef.printDocument("^Harrys Resume.pdf ^", 10);
60 }
61 }
62
63 }
64
65 public class SynchronizationApp {
66

```

7.4 Rather than blocking your methods separately as synchronized, you can even work on synchronized block. Let us come here and use the same synchronized block in the desktop as well. This is another way how you can implement synchronization.



```

29 }
30 }
31 }
32
33 class Laptop extends Thread{
34     Printer pRef;
35
36     void attachPrinter(Printer p) {
37         pRef = p;
38     }
39 }
40
41 @Override
42 public void run() {
43     synchronized (pRef) {
44         pRef.printDocument(">> Johns Resume.pdf <<", 10);
45     }
46 }
47 }
48
49 class Desktop extends Thread{
50     Printer pRef;
51
52     void attachPrinter(Printer p) {
53         pRef = p;
54     }
55 }
56
57 @Override
58 public void run() {
59     synchronized (this) {
60         pRef.printDocument("^Harrys Resume.pdf ^", 10);
61     }
62 }
63 }
64
65 public class SynchronizationApp {
66

```

7.5 If you run the code, you can see the output which is exactly the same which shows that John's resume is getting printed before the Harry's resume and this is what is expected as the synchronized output.

```

29 }
30 }
31 }
32 }
33 class Laptop extends Thread{
34     Printer pRef;
35     void attachPrinter(Printer p) {
36         pRef = p;
37     }
38     @Override
39     public void run() {
40         synchronized (pRef) {
41             pRef.printDocument(">> Johns Resume.pdf <<", 10);
42         }
43     }
44 }
45 class Desktop extends Thread{
46     Printer pRef;
47     void attachPrinter(Printer p) {
48         pRef = p;
49     }
50     @Override
51     public void run() {
52         synchronized (pRef) {
53             pRef.printDocument(">> Harrys Resume.pdf <<", 5);
54         }
55     }
56 }
57 }
58 }
59 }

```

```

[Printer] Printing >> Johns Resume.pdf << copy #4...
[Printer] Printing >> Johns Resume.pdf << copy #5...
[Printer] Printing >> Johns Resume.pdf << copy #6...
[Printer] Printing >> Johns Resume.pdf << copy #7...
[Printer] Printing >> Johns Resume.pdf << copy #8...
[Printer] Printing >> Johns Resume.pdf << copy #9...
[Printer] Printing >> Johns Resume.pdf << copy #10...
[Printer] Status: Available

[Printer] Printnig the document: ^^ Harrys Resume.pdf ^^
[Printer] Status: Busy

[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #1...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #2...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #3...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #4...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #5...

```

```

29 }
30 }
31 }
32 }
33 class Laptop extends Thread{
34     Printer pRef;
35     void attachPrinter(Printer p) {
36         pRef = p;
37     }
38     @Override
39     public void run() {
40         synchronized (pRef) {
41             try {
42                 pRef.wait();
43             } catch (InterruptedException e) {
44                 e.printStackTrace();
45             }
46             pRef.printDocument(">> Johns Resume.pdf <<", 10);
47         }
48     }
49 }
50 class Desktop extends Thread{
51     Printer pRef;
52     void attachPrinter(Printer p) {
53         pRef = p;
54     }
55     @Override
56     public void run() {
57         synchronized (pRef) {
58             pRef.printDocument(">> Harrys Resume.pdf <<", 5);
59         }
60     }
61 }
62 }
63 }
64 }

```

- 7.6 Now the output here shows that Harry's resume is getting printed before the John's resume and you do not see any output coming from the John's resume. This is a situation where you have made your laptop thread to wait infinitely. When the laptop thread is waiting in an infinite way, it means you have blocked the thread.

```

42 public void run() {
43     synchronized (pRef) {
44         try {
45             pRef.wait();
46         } catch (InterruptedException e) {
47             e.printStackTrace();
48         }
49     }
50     pRef.printDocument(">> Johns Resume.pdf <<", 10);
51 }
52 }
53 }
54 }
55 }
56 class Desktop extends Thread{
57     Printer pRef;
58     void attachPrinter(Printer p) {
59         pRef = p;
60     }
61     @Override
62     public void run() {
63         synchronized (pRef) {
64             pRef.printDocument("^^ Harrys Resume.pdf ^^", 10);
65         }
66     }
67 }
68 }
69 }
70 }

```

```

[Printer] Printinig the document: ^^ Harrys Resume.pdf ^^
[Printer] Status: Busy
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #1...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #2...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #3...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #4...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #5...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #6...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #7...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #8...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #9...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #10...
[Printer] Status: Available

```

- 7.7 Ideal way is when you have executed your desktop thread, then later you mention as **ref.notify()**. Now, the output shows, that initially the Harry's resume will be printed, then the notification goes to John thread, which is like the laptop thread, hence, it resumes and finishes its printing. One of the ways how you can implement a concept called Wait and notify with the synchronization when you are synchronizing the threads, the keyword synchronize with the methods will make them synchronized. Or you can take the lock entirely on the object rather than a printer using the synchronized block.

```

42 public void run() {
43     synchronized (pRef) {
44         try {
45             pRef.wait();
46         } catch (InterruptedException e) {
47             e.printStackTrace();
48         }
49     }
50     pRef.printDocument(">> Johns Resume.pdf <<", 10);
51     pRef.notify();
52 }
53 }
54 }
55 }
56 class Desktop extends Thread{
57     Printer pRef;
58     void attachPrinter(Printer p) {
59         pRef = p;
60     }
61     @Override
62     public void run() {
63         synchronized (pRef) {
64             pRef.printDocument("^^ Harrys Resume.pdf ^^", 10);
65             pRef.notify();
66         }
67     }
68 }
69 }
70 }

```

```

<terminated> SynchronizationApp [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.ui/opersdk.hotspot
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #9...
[Printer] Printing ^^ Harrys Resume.pdf ^^ copy #10...
[Printer] Status: Available
[Printer] Printinig the document: >> Johns Resume.pdf <<
[Printer] Status: Busy
[Printer] Printing >> Johns Resume.pdf << copy #1...
[Printer] Printing >> Johns Resume.pdf << copy #2...
[Printer] Printing >> Johns Resume.pdf << copy #3...
[Printer] Printing >> Johns Resume.pdf << copy #4...
[Printer] Printing >> Johns Resume.pdf << copy #5...
[Printer] Printing >> Johns Resume.pdf << copy #6...
[Printer] Printing >> Johns Resume.pdf << copy #7...
[Printer] Printing >> Johns Resume.pdf << copy #8...
[Printer] Printing >> Johns Resume.pdf << copy #9...
[Printer] Printing >> Johns Resume.pdf << copy #10...
[Printer] Status: Available

```

By following these steps, you have successfully demonstrated the concept of synchronization of threads in Java.