

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

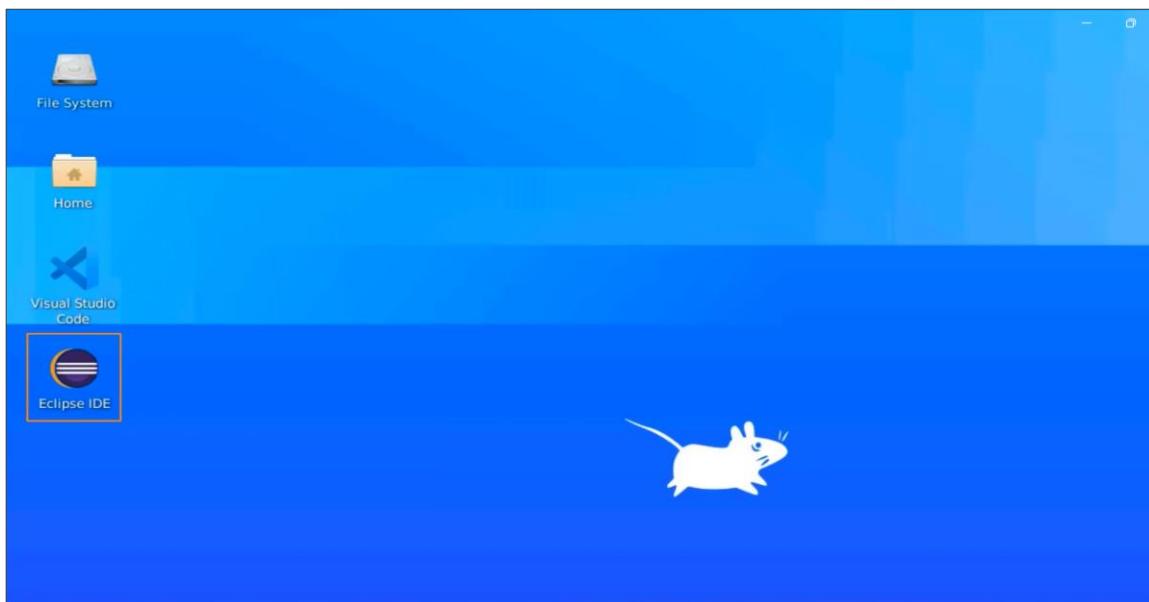
Prerequisites: None

Steps to be followed:

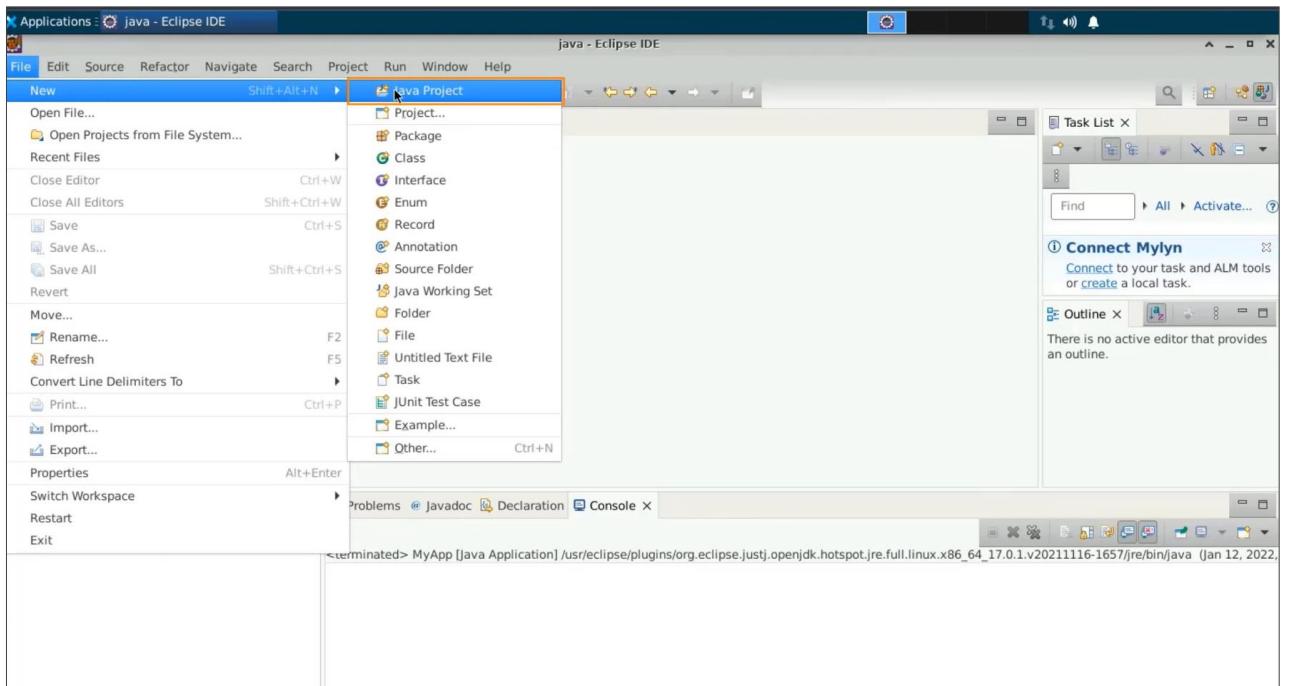
1. Open Eclipse IDE and create a new class
2. Write a for loop to print the document n number of times
3. Write a thread dot sleep, with the try catch
4. Create an object and execute the code
5. Create two threads and work on the same object
6. Execute the code with sample data
7. Implement the concept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

Step 1: Open Eclipse IDE and create a new class

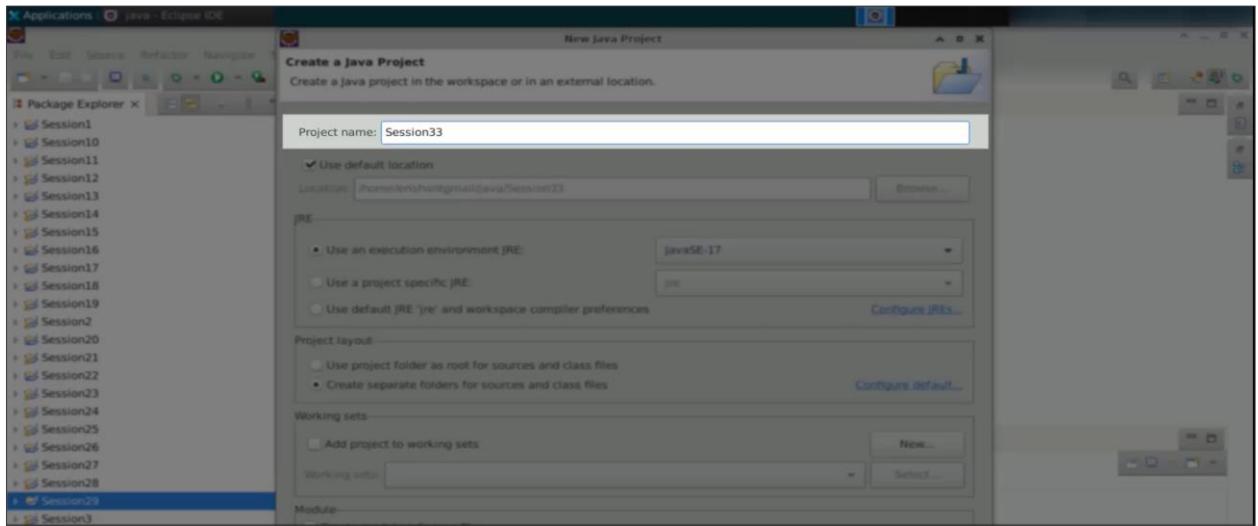
1.1 Open the Eclipse IDE



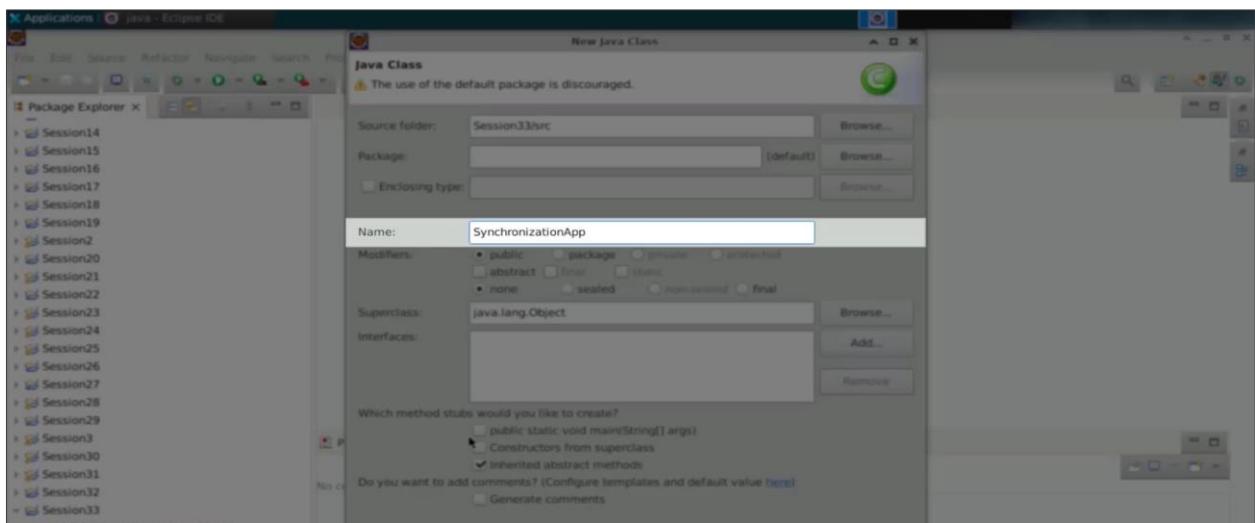
1.2 Select File, then New, and then Java project



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.

```

 1  *SynchronizationApp.java *
 2  class Printer {
 3      void printDocument(String docName, int copies) {
 4          System.out.println("[Printer] Printinig the document: "+docName);
 5      }
 6  }
 7
 8  public class SynchronizationApp {
 9      public static void main(String[] args) {
10
11
12
13
14
15
16
17
18
19
20
21

```

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.

```

Applications : Java - Session33/src/Syn...
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java X
1 class Printer{
2
3     void printDocument(String docName, int copies) {
4
5         System.out.println("[Printer] Prinitng the document: "+docName);
6
7         for(int i=1; i<=copies; i++) {
8             System.out.println("[Printer] "+docName+" copy #"+i);
9         }
10    }
11 }
12
13
14 public class SynchronizationApp {
15
16     public static void main(String[] args) {
17
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.

```

Applications : Java - Session33/src/Syn...
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java X
1 class Printer{
2
3     void printDocument(String docName, int copies) {
4
5         System.out.println("[Printer] Prinitng the document: "+docName);
6
7         for(int i=1; i<=copies; i++) {
8             System.out.println("[Printer] "+docName+" copy #"+i);
9
10            try {
11                Thread.sleep(500);
12            } catch (InterruptedException e) {
13                e.printStackTrace();
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.

```

Applications : java - Session33/src/Syn...
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java
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] Prinitng 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.

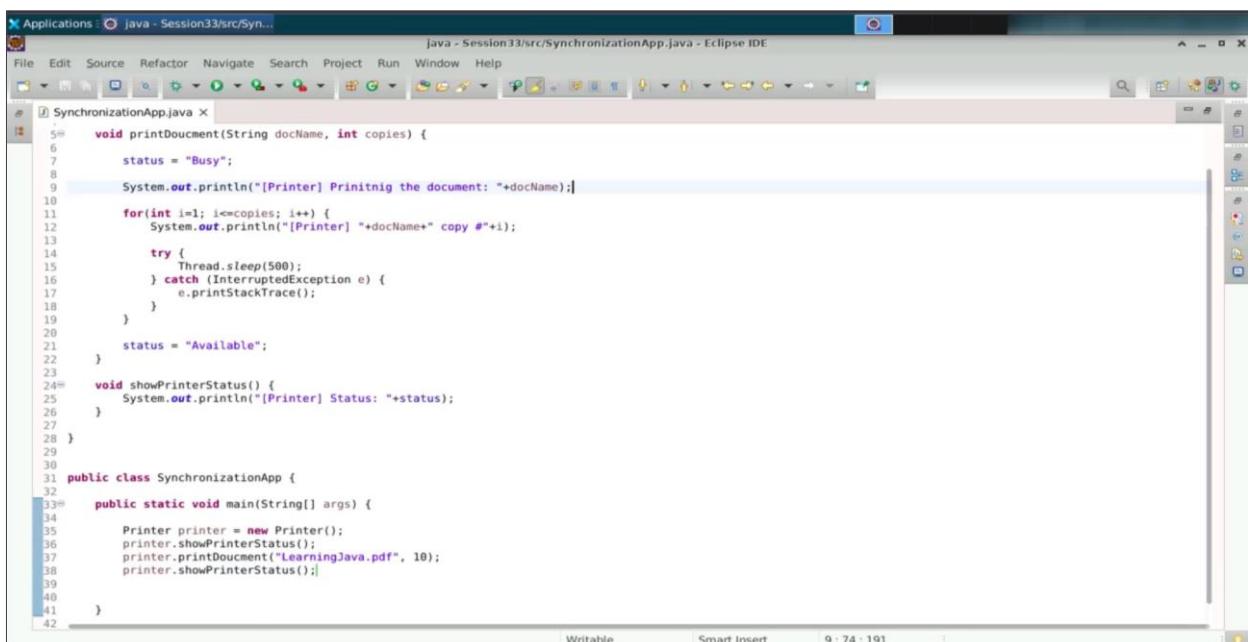
```

Applications : java - Session33/src/Syn...
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java
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] Prinitng 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.



The screenshot shows the Eclipse IDE interface with a Java file named SynchronizationApp.java open. The code implements a Printer class and a SynchronizationApp class. The Printer class has methods for printing documents and showing printer status. The SynchronizationApp class creates a Printer object, prints a document named 'LearningJava.pdf' 10 times, and then checks the printer status. The code uses System.out.println statements for logging.

```
Applications : java - Session33/src/Syn...
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java
5 void printDocument(String docName, int copies) {
6     status = "Busy";
7     System.out.println("[Printer] Prinitng the document: "+docName);
8
9     for(int i=1; i<copies; i++) {
10         System.out.println("[Printer] "+docName+" copy #"+i);
11
12         try {
13             Thread.sleep(500);
14         } catch (InterruptedException e) {
15             e.printStackTrace();
16         }
17     }
18     status = "Available";
19 }
20
21 void showPrinterStatus() {
22     System.out.println("[Printer] Status: "+status);
23 }
24
25 public class SynchronizationApp {
26
27     public static void main(String[] args) {
28
29         Printer printer = new Printer();
30         printer.showPrinterStatus();
31         printer.printDocument("LearningJava.pdf", 10);
32         printer.showPrinterStatus();
33
34     }
35 }
36
37
38
39
40
41
42
```

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.

```

Applications java - Session3/src/Syn...
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java X
1 void printDocument(String docName, int copies) {
2     status = "Busy";
3     System.out.println("[Printer] Printing the document: "+docName);
4     showPrinterStatus();
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     status = "Available";
17 }
18
19 void showPrinterStatus() {
20     System.out.println("[Printer] Status: "+status);
21 }
22
23
24 public class SynchronizationApp {
25     public static void main(String[] args) {
26
27
28     }
29 }
30
31
32 public class SynchronizationApp {
33
34     public static void main(String[] args) {

```

<terminated> SynchronizationApp [Java Application] /usr/eclipse/plugins/org.eclipse.justiz/0.1.0.20180320-1000-SNAPSHOT/bin/Session3/src/SynchronizationApp.java

[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.

```

Applications java - Session3/src/Syn...
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java X
1 void printDocument(String docName, int copies) {
2     status = "Busy";
3     System.out.println("[Printer] Printing the document: "+docName);
4     showPrinterStatus();
5
6     for(int i=1; i<copies; i++) {
7         System.out.println("[Printer] Printing "+docName+" copy #"+i+"..."); // Cursor is here
8
9         try {
10             Thread.sleep(500);
11         } catch (InterruptedException e) {
12             e.printStackTrace();
13         }
14     }
15
16     status = "Available";
17 }
18
19
20
21
22
23
24
25 void showPrinterStatus() {
26     System.out.println("[Printer] Status: "+status);
27 }
28
29
30
31
32 public class SynchronizationApp {
33
34     public static void main(String[] args) {

```

4.4 Thus, this is the printing status as shown.

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

- Title Bar:** Applications - java - Session33/src/SynchronizationApp.java - Eclipse IDE
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Toolbar:** Standard Eclipse toolbar icons.
- Left Panel:** Shows the project structure with a file named SynchronizationApp.java selected.
- Code Editor:** Displays the Java code for the SynchronizationApp class. The code includes methods for printing documents and showing printer status.
- Output View:** Shows the execution logs from the console. It starts with the printer being available, then prints a document named LearningJava.pdf, and then prints it 10 times, each time indicating the copy number (1 through 10). Finally, it returns to an available state.

```

50     void printDocument(String docName, int copies) {
51         status = "Busy";
52         System.out.println("[Printer] Printing the document: "+docName);
53         showPrinterStatus();
54         for(int i=1; i<=copies; i++) {
55             System.out.println("[Printer] Printing "+docName+" copy #"+i+"...");
56             try {
57                 Thread.sleep(500);
58             } catch (InterruptedException e) {
59                 e.printStackTrace();
60             }
61         }
62         status = "Available";
63     }
64     void showPrinterStatus() {
65         System.out.println("[Printer] Status: "+status);
66     }
67 }
68 public class SynchronizationApp {
69     public static void main(String[] args) {
70
71     }
72 }

```

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 attach printer that takes a printer object as an input and assigns it to the reference variable pRef.

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

- Title Bar:** Applications - java - Session33/src/SynchronizationApp.java - Eclipse IDE
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Toolbar:** Standard Eclipse toolbar icons.
- Left Panel:** Shows the project structure with a file named SynchronizationApp.java selected.
- Code Editor:** Displays the Java code for the SynchronizationApp class. It includes a new class called Laptop that extends Thread. The attachPrinter method sets the pRef variable to the passed Printer object.
- Output View:** Not visible in this screenshot, but the code structure is shown.

```

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 Laptop extends Thread{
31     Printer pRef;
32     void attachPrinter(Printer p) {
33         pRef = p;
34     }
35 }
36
37 }
38 }
39 }
40 public class SynchronizationApp {
41     public static void main(String[] args) {
42
43         Printer printer = new Printer();
44         printer.showPrinterStatus();
45         printer.printDocument("LearningJava.pdf", 10);
46         printer.showPrinterStatus();
47
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.

```

Applications : java - Session33/src/Syn...
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java X
1 package com.simplilearn.threads;
2
3 import java.awt.print.PrinterException;
4 import java.io.IOException;
5 import java.util.concurrent.TimeUnit;
6
7 public class Printer {
8     String status;
9
10    void sleep() {
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
23 }
24
25 class Laptop extends Thread{
26
27     Printer pRef;
28
29     void attachPrinter(Printer p) {
30         pRef = p;
31     }
32
33     @Override
34     public void run() {
35         pRef.printDocument(">> John's Resume.pdf <<", 10);
36     }
37 }
38
39 public class SynchronizationApp {
40
41     public static void main(String[] args) {
42         Printer printer = new Printer();
43         printer.showPrinterStatus();
44     }
45 }
46
47 public class SynchronizationApp {
48
49     public static void main(String[] args) {
50         Printer printer = new Printer();
51         printer.showPrinterStatus();
52     }
53 }
54
55 public class SynchronizationApp {
56
57     public static void main(String[] args) {
58         Printer printer = new Printer();
59         printer.showPrinterStatus();
60     }
61 }
62
63 public class SynchronizationApp {
64
65     public static void main(String[] args) {
66         Printer printer = new Printer(); // we have created only a single printer object
67     }
68 }

```

- 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.

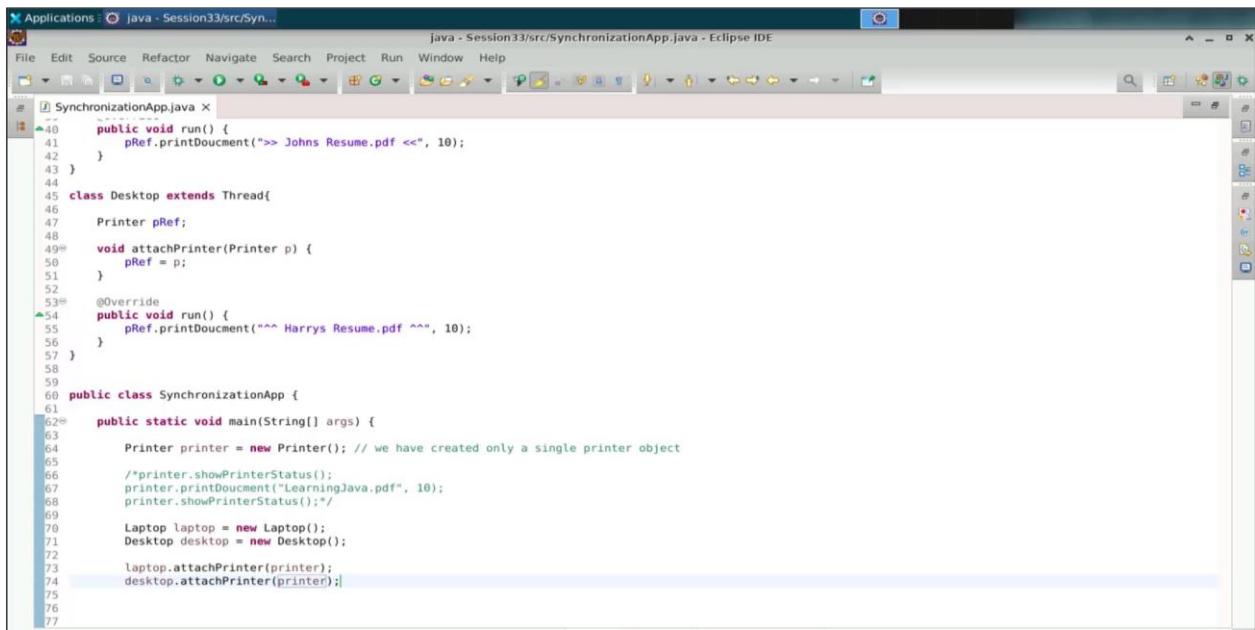
```

Applications : java - Session33/src/Syn...
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java X
1 package com.simplilearn.threads;
2
3 import java.awt.print.PrinterException;
4 import java.io.IOException;
5
6 public class Printer {
7     String pRef;
8
9     void pRef() {
10        pRef = p;
11    }
12
13    @Override
14    public void run() {
15        pRef.printDocument(">> John's Resume.pdf <<", 10);
16    }
17 }
18
19 class Laptop extends Thread{
20
21     Printer pRef;
22
23     void attachPrinter(Printer p) {
24         pRef = p;
25     }
26
27     @Override
28     public void run() {
29         pRef.printDocument(">> Harry's Resume.pdf <<", 10);
30     }
31 }
32
33 class Desktop extends Thread{
34
35     Printer pRef;
36
37     void attachPrinter(Printer p) {
38         pRef = p;
39     }
40
41     @Override
42     public void run() {
43         pRef.printDocument(">> Harry's Resume.pdf <<", 10);
44     }
45 }
46
47 public class SynchronizationApp {
48
49     public static void main(String[] args) {
50         Printer printer = new Printer(); // we have created only a single printer object
51     }
52 }
53
54 public class SynchronizationApp {
55
56     public static void main(String[] args) {
57         Printer printer = new Printer();
58         printer.showPrinterStatus();
59     }
60 }
61
62 public class SynchronizationApp {
63
64     public static void main(String[] args) {
65         Printer printer = new Printer();
66         printer.showPrinterStatus();
67     }
68 }

```

- 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.

- 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



The screenshot shows the Eclipse IDE interface with the title bar "Applications : java - Session33/src/SynchronizationApp.java - Eclipse IDE". The main window displays the Java code for SynchronizationApp.java. The code defines a class SynchronizationApp with a main method. It creates a single Printer object and prints two documents ("Johns Resume.pdf" and "Harrys Resume.pdf") using it. It then creates a Desktop object and attaches the same Printer to it, demonstrating that both objects share the same printer reference.

```
# SynchronizationApp.java X
40     public void run() {
41         pRef.printDocument(">> Johns Resume.pdf <<", 10);
42     }
43 }
44
45 class Desktop extends Thread{
46
47     Printer pRef;
48
49     void attachPrinter(Printer p) {
50         pRef = p;
51     }
52
53     @Override
54     public void run() {
55         pRef.printDocument("^^ Harrys Resume.pdf ^^", 10);
56     }
57 }
58
59
60 public class SynchronizationApp {
61
62     public static void main(String[] args) {
63
64         Printer printer = new Printer(); // we have created only a single printer object
65
66         /*printer.showPrinterStatus();
67         printer.printDocument("LearningJava.pdf", 10);
68         printer.showPrinterStatus();*/
69
70         Laptop laptop = new Laptop();
71         Desktop desktop = new Desktop();
72
73         laptop.attachPrinter(printer);
74         desktop.attachPrinter(printer);
```

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.

The screenshot shows the Eclipse IDE interface with the title bar "java - Session33/src/Syn..." and the window title "java - Session33/src/SynchronizationApp.java - Eclipse IDE". The code editor displays the following Java code:

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

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

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.

The screenshot shows the Eclipse IDE interface. The left pane displays a Java file named `SynchronizationApp.java` with code related to printing resumes. The right pane shows a terminal window with multiple instances of the `Printitng` class printing resume files.

```
45 class Desktop extends Thread{  
46     Printer pRef;  
47     void attachPrinter(Printer p) {  
48         pRef = p;  
49     }  
50     @Override  
51     public void run() {  
52         pRef.printDocument(new File("Harrys.Resume.pdf"), 10);  
53     }  
54 }  
55  
56 public class SynchronizationApp {  
57     public static void main(String[] args) {  
58         Printer printer = new Printer(); // we have created only a single printer object  
59         /*printer.showPrinterStatus();  
60         printer.printDocument("LearningJava.pdf", 10);  
61         printer.showPrinterStatus();*/  
62         Laptop laptop = new Laptop();  
63         Desktop desktop = new Desktop();  
64         laptop.attachPrinter(printer);  
65         desktop.attachPrinter(printer);  
66     }  
67 }
```

```
[Printer] Printitng ^~ Harrys.Resume.pdf ^~ copy #2...  
[Printer] Printitng >> Johns.Resume.pdf << copy #3...  
[Printer] Printitng ^~ Harrys.Resume.pdf ^~ copy #3...  
[Printer] Printitng >> Johns.Resume.pdf << copy #4...  
[Printer] Printitng ^~ Harrys.Resume.pdf ^~ copy #4...  
[Printer] Printitng >> Johns.Resume.pdf << copy #5...  
[Printer] Printitng ^~ Harrys.Resume.pdf ^~ copy #5...  
[Printer] Printitng >> Johns.Resume.pdf << copy #6...  
[Printer] Printitng ^~ Harrys.Resume.pdf ^~ copy #6...  
[Printer] Printitng >> Johns.Resume.pdf << copy #7...  
[Printer] Printitng ^~ Harrys.Resume.pdf ^~ copy #7...  
[Printer] Printitng >> Johns.Resume.pdf << copy #8...  
[Printer] Printitng ^~ Harrys.Resume.pdf ^~ copy #8...  
[Printer] Printitng >> Johns.Resume.pdf << copy #9...  
[Printer] Printitng ^~ Harrys.Resume.pdf ^~ copy #9...  
[Printer] Printitng >> Johns.Resume.pdf << copy #10...  
[Printer] Printitng ^~ 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.

The screenshot shows the Eclipse IDE interface with the title bar "Applications > Java - Session3/src/SynchronizationApp.java - Eclipse IDE". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has various icons for file operations. The code editor displays a Java file named "SynchronizationApp.java" containing the following code:

```
# SynchronizationApp.java X
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
22        status = "Available";
23    }
24
25    void showPrinterStatus() {
26        System.out.println("[Printer] Status: "+status);
27    }
28
29 }
30
31 class Listener extends Thread{
32 }
```

- 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.



The screenshot shows the Eclipse IDE interface with the title bar "java - Session33/src/Synch..." and the menu bar "File Edit Source Refactor Navigate Search Project Run Window Help". The toolbar includes icons for New, Open, Save, Cut, Copy, Paste, Find, and others. Below the toolbar, there are tabs for "Problems", "Javadoc", "Declaration", and "Console X". The "Console" tab is active, displaying the following text:

```
<terminated> SynchronizationApp [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, 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.

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

- Title Bar:** Applications > java - Session13/src/Syn...
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Toolbars:** Standard toolbar with icons for New, Open, Save, Cut, Copy, Paste, Find, etc.
- Left Sidebar:** Package Explorer view showing multiple session files (Session15, Session16, Session17, Session18, Session19, Session2, Session20, Session21, Session22, Session23, Session24, Session25, Session26, Session27, Session28, Session29, Session3, Session30, Session31, Session32, Session33) and a JRE System Library entry.
- Central View:** Editor showing the `SynchronizationApp.java` file content. The code defines a `Laptop` class that extends `Thread`. It contains a `status` variable and a `showPrinterStatus()` method that prints the printer status to the console. A `void attachPrinter(Printer p)` method is also present.
- Bottom Status Bar:** Shows the current project is `SynchronizationApp` (Java Application), the path is `/usr/eclipse/plugins/org.eclipse.jdt.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java`, and the date is `(Jan 22, 2022)`.

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.

```
[Applications] Java - Session33/src/Syn...          java - Session33/src/SynchronizationApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console X
SynchronizationApp [Java Application] /usr/eclipse/plugins/org.eclipse.jst.jdt.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/re/bin/java (Jan 22, 2022, 9:41:21 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...

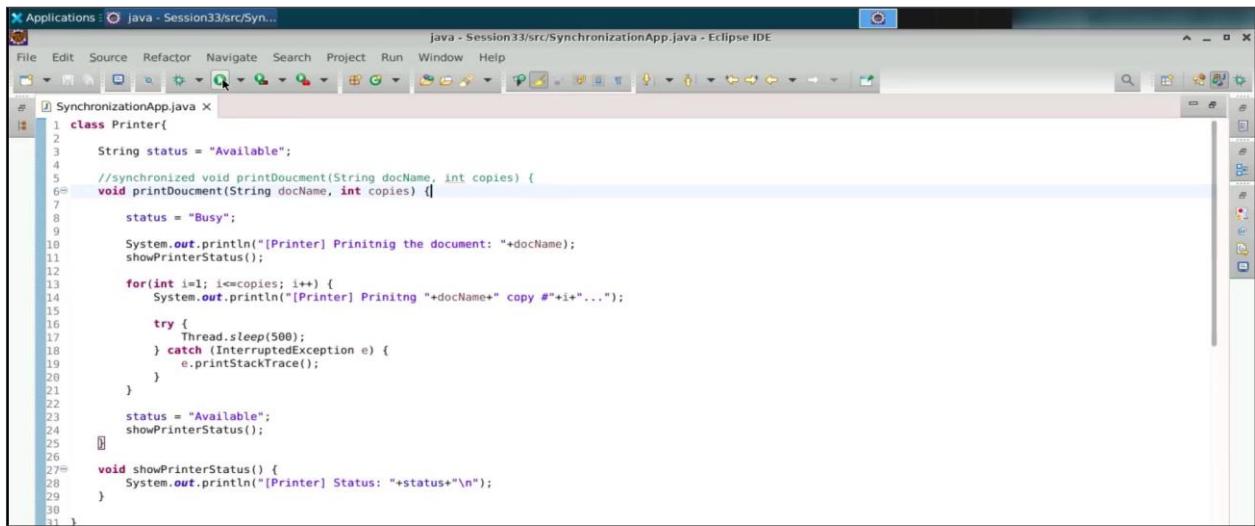
[Printer] Status: Available

[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...
```

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.

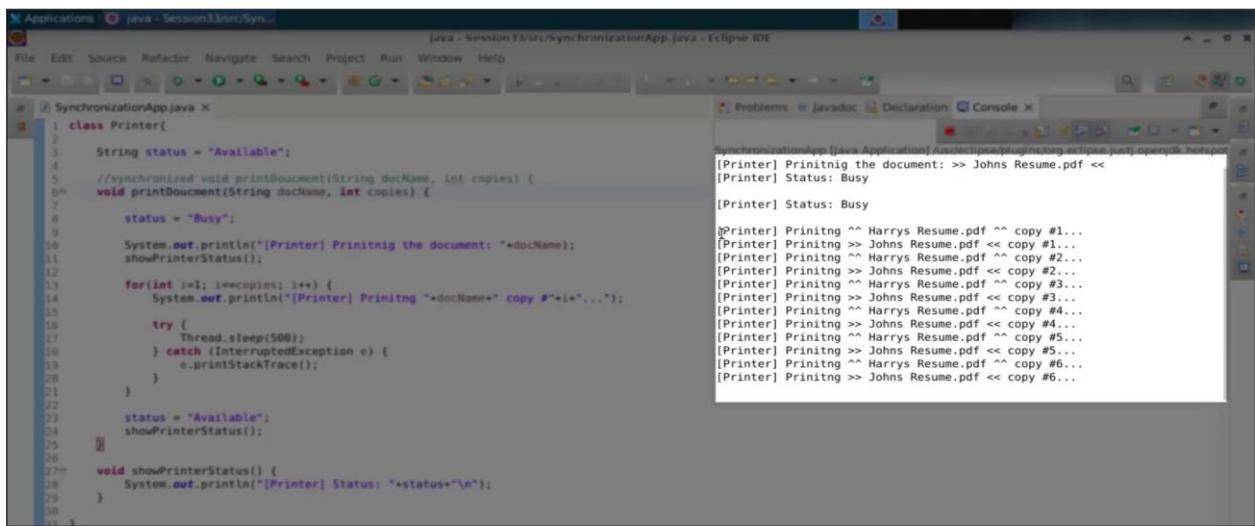


```

Applications : java - Session33/src/Syn...
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java X
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] Printning the document: "+docName);
11        showPrinterStatus();
12
13        for(int i=1; i<=copies; i++) {
14            System.out.println("[Printer] Printing "+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    void showPrinterStatus() {
27        System.out.println("[Printer] Status: "+status+"\n");
28    }
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.



```

Applications : java - Session33/src/Syn...
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java X
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] Printning the document: "+docName);
11        showPrinterStatus();
12
13        for(int i=1; i<=copies; i++) {
14            System.out.println("[Printer] Printing "+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    void showPrinterStatus() {
27        System.out.println("[Printer] Status: "+status+"\n");
28    }
29
30
31

```

SynchronizationApp [Java Application] /src/SynchronizationApp.java line 6: public static void main(String[] args) {
[Printer] Printning the document: >> Johns Resume.pdf <<
[Printer] Status: Busy
[Printer] Status: Busy
[Printer] Printing ^ Harrys Resume.pdf ^ copy #1...
[Printer] Printing >> Harrys Resume.pdf << copy #1...
[Printer] Printing ^ Harrys Resume.pdf ^ copy #2...
[Printer] Printing >> Harrys Resume.pdf << copy #2...
[Printer] Printing ^ Harrys Resume.pdf ^ copy #3...
[Printer] Printing >> Harrys Resume.pdf << copy #3...
[Printer] Printing ^ Harrys Resume.pdf ^ copy #4...
[Printer] Printing >> Harrys Resume.pdf << copy #4...
[Printer] Printing ^ Harrys Resume.pdf ^ copy #5...
[Printer] Printing >> Harrys Resume.pdf << copy #5...
[Printer] Printing ^ Harrys Resume.pdf ^ copy #6...
[Printer] Printing >> Harrys Resume.pdf << copy #6...
[Printer] Printing ^ Johns Resume.pdf ^ copy #7...
[Printer] Printing >> Johns Resume.pdf << copy #7...
[Printer] Printing ^ Johns Resume.pdf ^ copy #8...
[Printer] Printing >> Johns Resume.pdf << copy #8...
[Printer] Printing ^ Johns Resume.pdf ^ copy #9...
[Printer] Printing >> Johns Resume.pdf << copy #9...
[Printer] Printing ^ Johns Resume.pdf ^ copy #10...
[Printer] Printing >> Johns Resume.pdf << copy #10...

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
35     Printer pRef;
36
37     void attachPrinter(Printer p) {
38         pRef = p;
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
51     Printer pRef;
52
53     void attachPrinter(Printer p) {
54         pRef = p;
55     }
56
57     @Override
58     public void run() {
59         synchronized (pRef) {
60             pRef.printDocument("^^ Harrys Resume.pdf ^^", 10);
61         }
62     }
63 }
64
65 public class SynchronizationApp {

```

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
35     Printer pRef;
36
37     void attachPrinter(Printer p) {
38         pRef = p;
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
51     Printer pRef;
52
53     void attachPrinter(Printer p) {
54         pRef = p;
55     }
56
57     @Override
58     public void run() {
59         synchronized (pRef) {
60             pRef.printDocument("^^ Harrys Resume.pdf ^^", 10);
61         }
62     }
63 }
64
65 public class SynchronizationApp {

```

- 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.

The screenshot shows the Eclipse IDE interface with the Java editor open. The code defines two threads, Laptop and Desktop, which print documents. The Desktop thread prints 'Johns Resume.pdf' and the Laptop thread prints 'Harrys Resume.pdf'. Both threads use a synchronized block to access the printer reference, ensuring they don't print at the same time. The console window shows the actual output of the program, where the Desktop prints first, followed by the Laptop.

```

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

SynchronizationApp [Java Application] /Session33/src/SynchronizationApp.java - Eclipse IDE

```

[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] Prinitng 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...

```

This screenshot shows the same Java code as above, but with a try-catch block added to the Desktop thread's run method. This block catches an InterruptedException and prints a stack trace. The rest of the code remains the same, with the synchronized block on the printer reference and the two threads printing their respective documents.

```

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

- 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.

The screenshot shows the Eclipse IDE interface with the Java - Session 3/src/SynchronizationApp.java file open. The code implements two threads: a printer thread and a desktop thread. The printer thread prints Harry's resume 10 times, while the desktop thread prints John's resume 10 times. The printer thread uses a synchronized block on a reference to a Printer object, causing the desktop thread to wait indefinitely. The console output shows the printer thread printing Harry's resume 10 times, but the desktop thread's output is missing.

```

public void run() {
    synchronized (pRef) {
        try {
            pRef.wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        pRef.printDocument(">> Johns Resume.pdf <<", 10);
    }
}

class Desktop extends Thread {
    Printer pRef;
    void attachPrinter(Printer p) {
        pRef = p;
    }
    @Override
    public void run() {
        synchronized (pRef) {
            pRef.printDocument(">> Harrys Resume.pdf <<", 10);
        }
    }
}

```

SynchronizationApp [Java Application] /src/plugins/org.eclipse.justui.openedit.hotspot
[Printer] Printning the document: ^~ Harrys Resume.pdf ^~
[Printer] Status: Busy
[Printer] Printning ^~ Harrys Resume.pdf ^~ copy #1...
[Printer] Printning ^~ Harrys Resume.pdf ^~ copy #2...
[Printer] Printning ^~ Harrys Resume.pdf ^~ copy #3...
[Printer] Printning ^~ Harrys Resume.pdf ^~ copy #4...
[Printer] Printning ^~ Harrys Resume.pdf ^~ copy #5...
[Printer] Printning ^~ Harrys Resume.pdf ^~ copy #6...
[Printer] Printning ^~ Harrys Resume.pdf ^~ copy #7...
[Printer] Printning ^~ Harrys Resume.pdf ^~ copy #8...
[Printer] Printning ^~ Harrys Resume.pdf ^~ copy #9...
[Printer] Printning ^~ 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.

The screenshot shows the Eclipse IDE interface with the same Java code as the previous screenshot, but with a modification: the desktop thread now calls pRef.notify() after printing Harry's resume. This allows the printer thread to wake up and print John's resume. The console output shows both sets of prints occurring sequentially.

```

public void run() {
    synchronized (pRef) {
        try {
            pRef.wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        pRef.printDocument(">> Johns Resume.pdf <<", 10);
    }
}

class Desktop extends Thread {
    Printer pRef;
    void attachPrinter(Printer p) {
        pRef = p;
    }
    @Override
    public void run() {
        synchronized (pRef) {
            pRef.printDocument(">> Harrys Resume.pdf <<", 10);
            pRef.notify();
        }
    }
}

```

<terminated> SynchronizationApp [Java Application] /src/plugins/org.eclipse.justui.openedit.hotspot
[Printer] Printning ^~ Harrys Resume.pdf ^~ copy #9...
[Printer] Printning ^~ Harrys Resume.pdf ^~ copy #10...
[Printer] Status: Available
[Printer] Printning the document: >> Johns Resume.pdf <<
[Printer] Status: Busy
[Printer] Printning >> Johns Resume.pdf << copy #1...
[Printer] Printning >> Johns Resume.pdf << copy #2...
[Printer] Printning >> Johns Resume.pdf << copy #3...
[Printer] Printning >> Johns Resume.pdf << copy #4...
[Printer] Printning >> Johns Resume.pdf << copy #5...
[Printer] Printning >> Johns Resume.pdf << copy #6...
[Printer] Printning >> Johns Resume.pdf << copy #7...
[Printer] Printning >> Johns Resume.pdf << copy #8...
[Printer] Printning >> Johns Resume.pdf << copy #9...
[Printer] Printning >> Johns Resume.pdf << copy #10...
[Printer] Status: Available

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