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

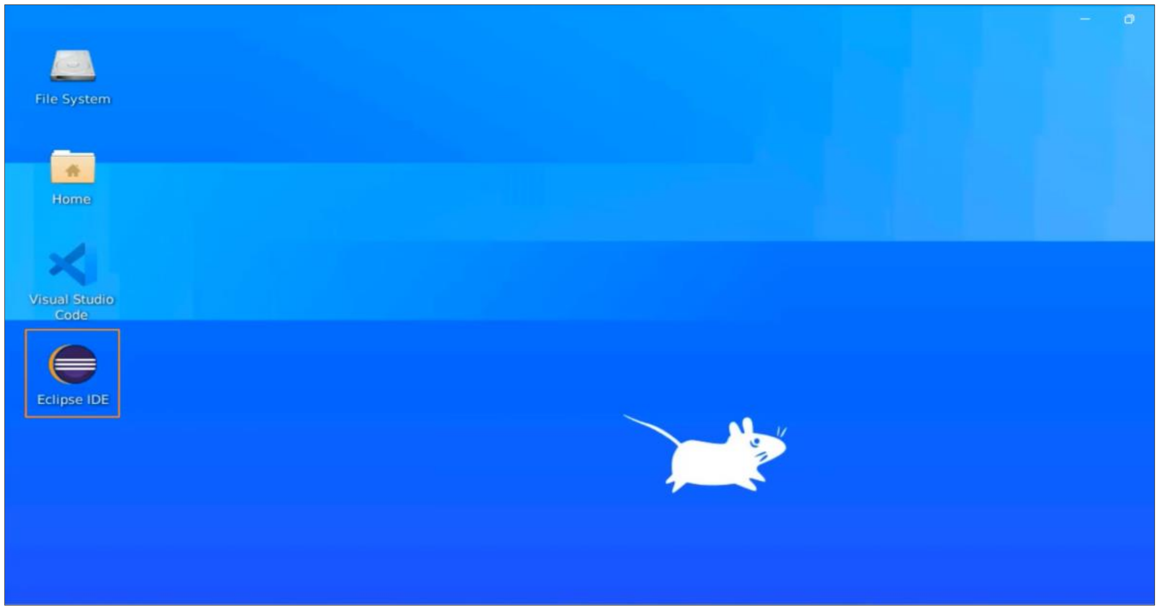
Prerequisites: None

Steps to be followed:

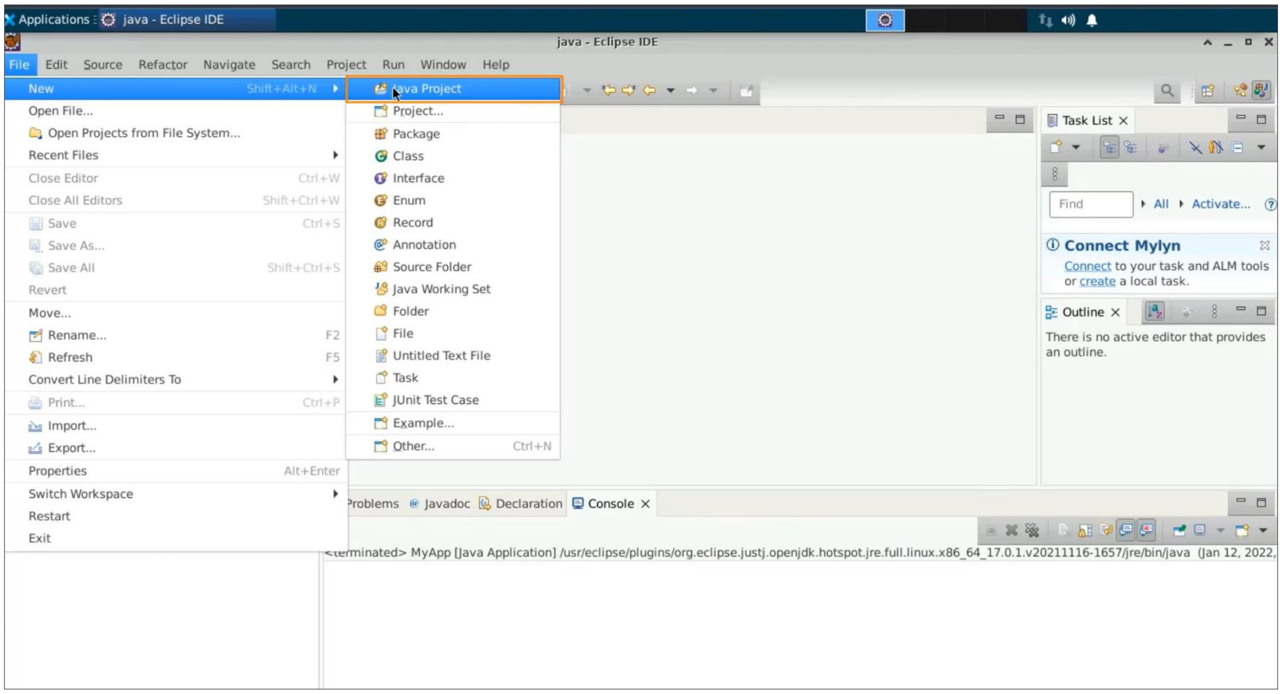
1. Open Eclipse IDE and create a new class
2. Write a for loop to print the document n number of times
3. Write a thread dot sleep, with the try catch
4. Create an object and execute the code
5. Create two threads and work on the same object
6. Execute the code with sample data
7. Implement the concept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

Step 1: Open Eclipse IDE and create a new class

1.1 Open the **Eclipse IDE**

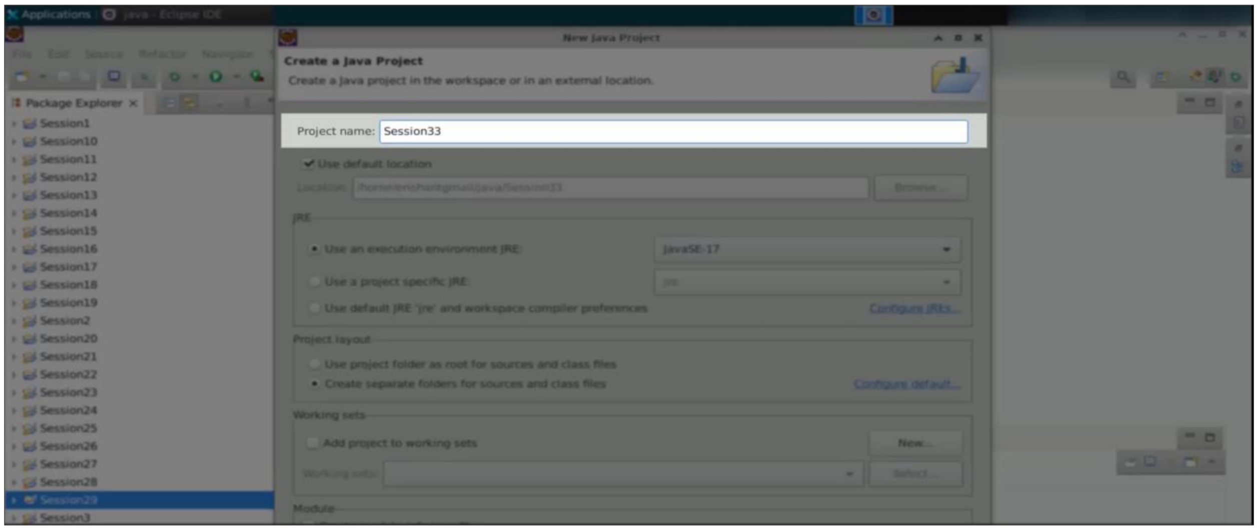


1.2 Select **File**, then **New,** and then **Java project**



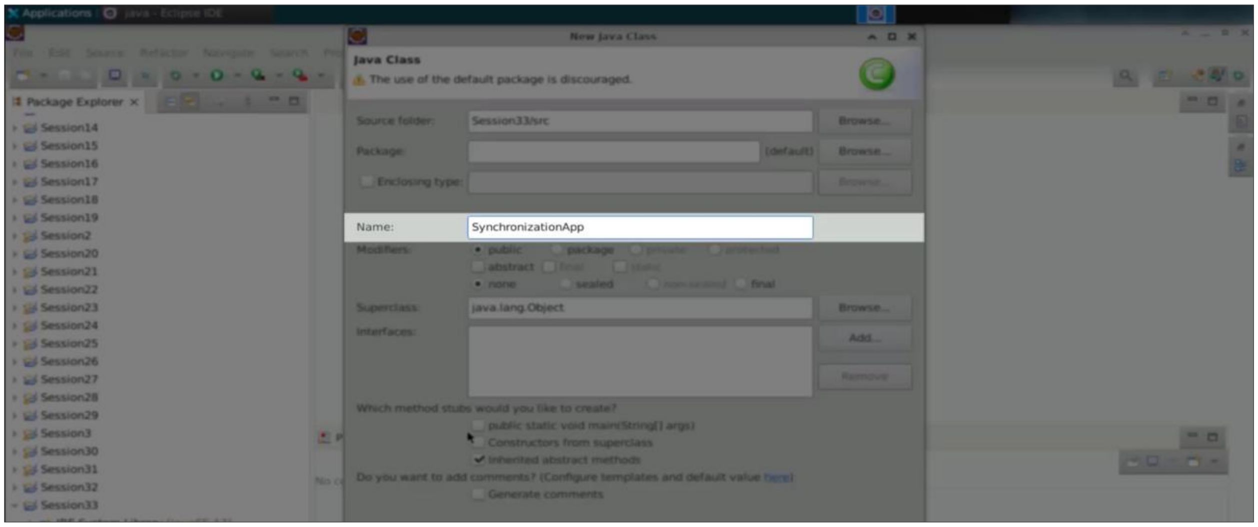
```
<terminated> MyApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java (Jan 12, 2022,
```

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


New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.

Project name: Session33

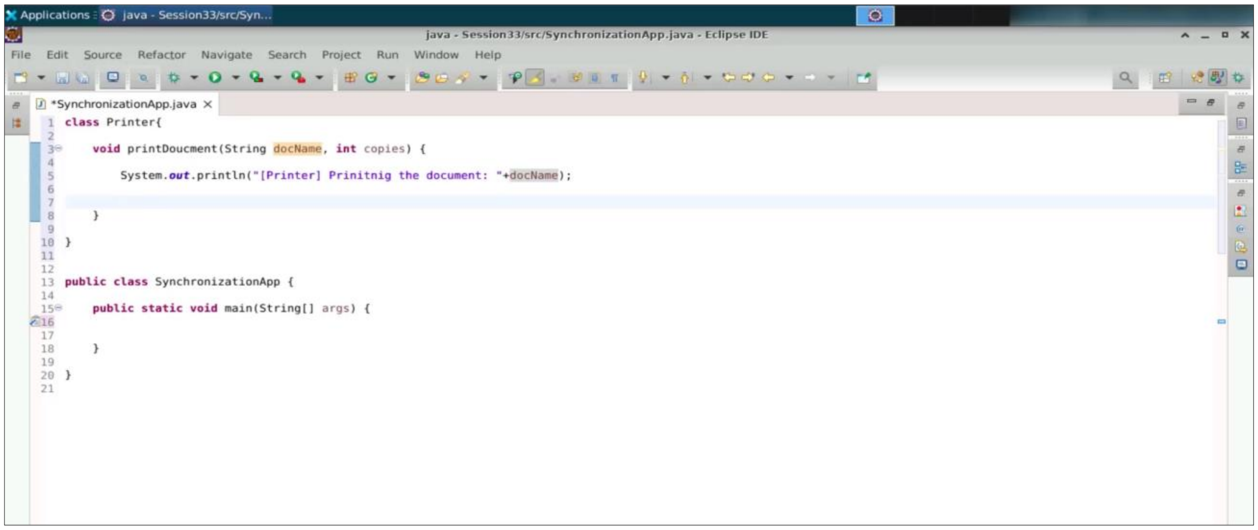
1.4 With a **Session33** on the src, do a right-click and create a **new class**. Name this class as an **SynchronizationApp**, then select the **main method,** and then select **finish.**



Java Class
The use of the default package is discouraged.

Source folder: Session33/src
Package: (default)
Name: SynchronizationApp
Modifiers: [✓ public] package private protected
abstract final static
[✓ none] sealed non-sealed final
Superclass: java.lang.Object
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
✓ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments

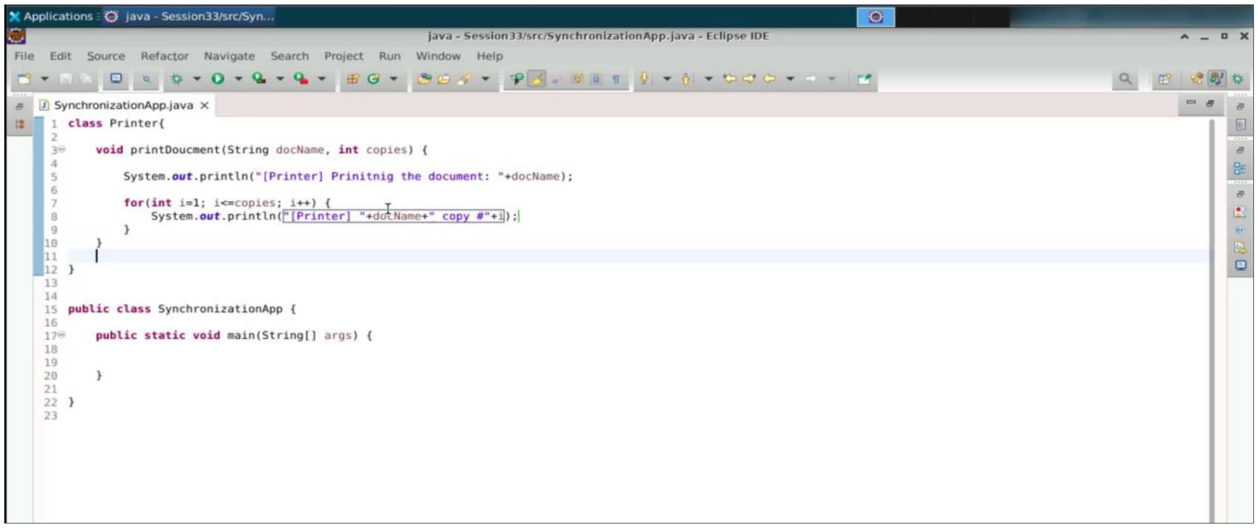
1.5 Create a class `Printer`, which can simulate printing documents on a real printer. For the `Printer`, create a method called `print_document`, which is a print command. This method will take a string as input for the document name and an integer for the number of copies you want for this document.



```
class Printer{

    void printDoucment(String docName, int copies) {

        System.out.println("[Printer] Prinitnig the document: "+docName);



    }

}


public class SynchronizationApp {

    public static void main(String[] args) {


    }

}
```

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

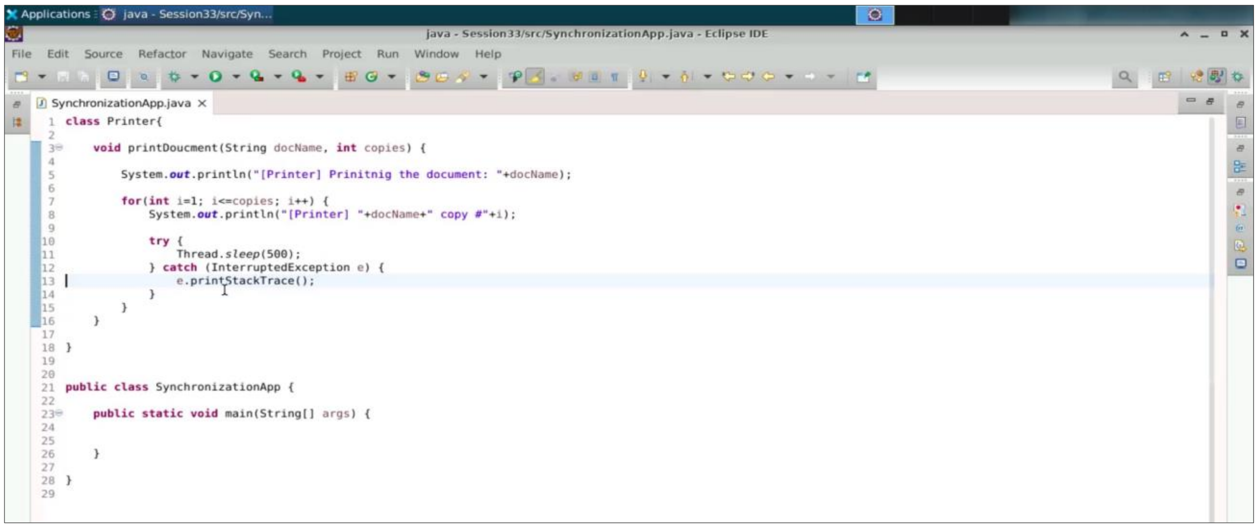
2.1 Next, create a for loop to print the document the specified number of times. Start with i at 1 and run the loop while i is less than or equal to the number of copies, incrementing i with each iteration. Print the document name followed by the copy number, +i. Since this also comes from your printer,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


java - Session33/src/SynchronizationApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java

```
class Printer{

    void printDoucment(String docName, int copies) {

        System.out.println("[Printer] Prinitnig the document: "+docName);

        for(int i=1; i<=copies; i++) {
            System.out.println("[Printer] "+docName+" copy #"+i);
        }
    }

}


public class SynchronizationApp {

    public static void main(String[] args) {


    }

}
```

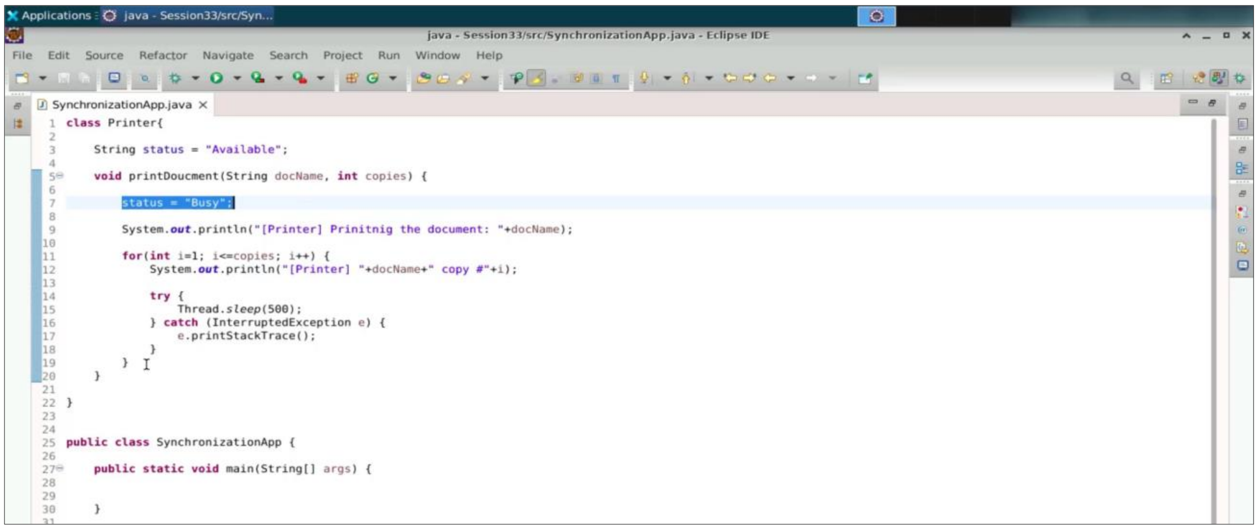
Step 3: Write a thread dot sleep, with the try catch

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.


java - Session33/src/SynchronizationApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java

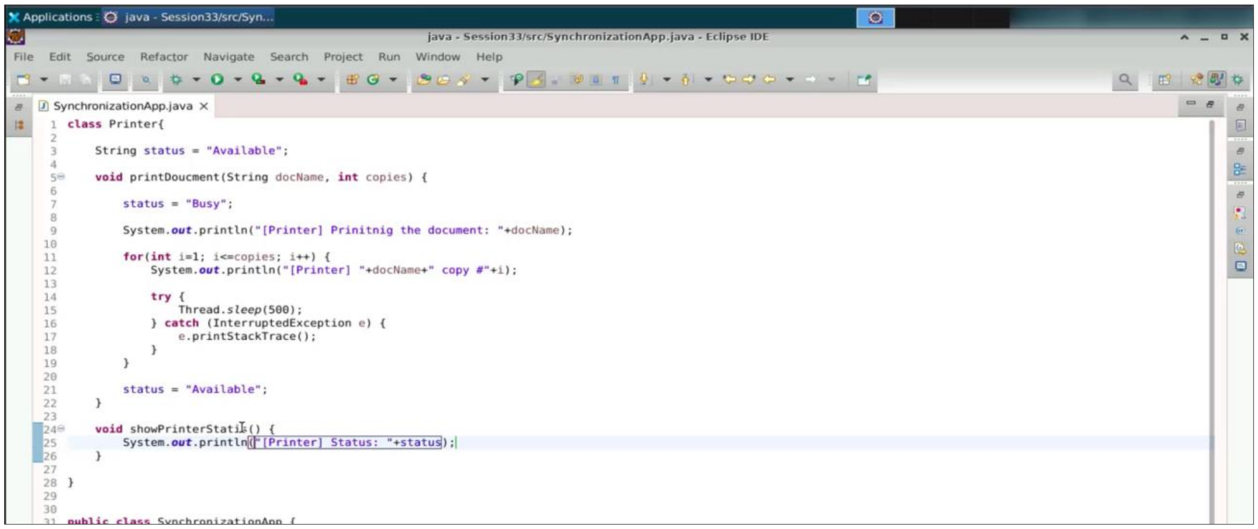
```
class Printer{

    void printDoucment(String docName, int copies) {

        System.out.println("[Printer] Prinitnig the document: "+docName);

        for(int i=1; i<=copies; i++) {
            System.out.println("[Printer] "+docName+" copy #"+i);

            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

}


public class SynchronizationApp {

    public static void main(String[] args) {


    }

}
```

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.


java - Session33/src/SynchronizationApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java

```java
class Printer{

    String status = "Available";

    void printDoucment(String docName, int copies) {

        status = "Busy";

        System.out.println("[Printer] Prinitnig the document: "+docName);

        for(int i=1; i<=copies; i++) {
            System.out.println("[Printer] "+docName+" copy #"+i);

            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}


public class SynchronizationApp {

    public static void main(String[] args) {


    }
```

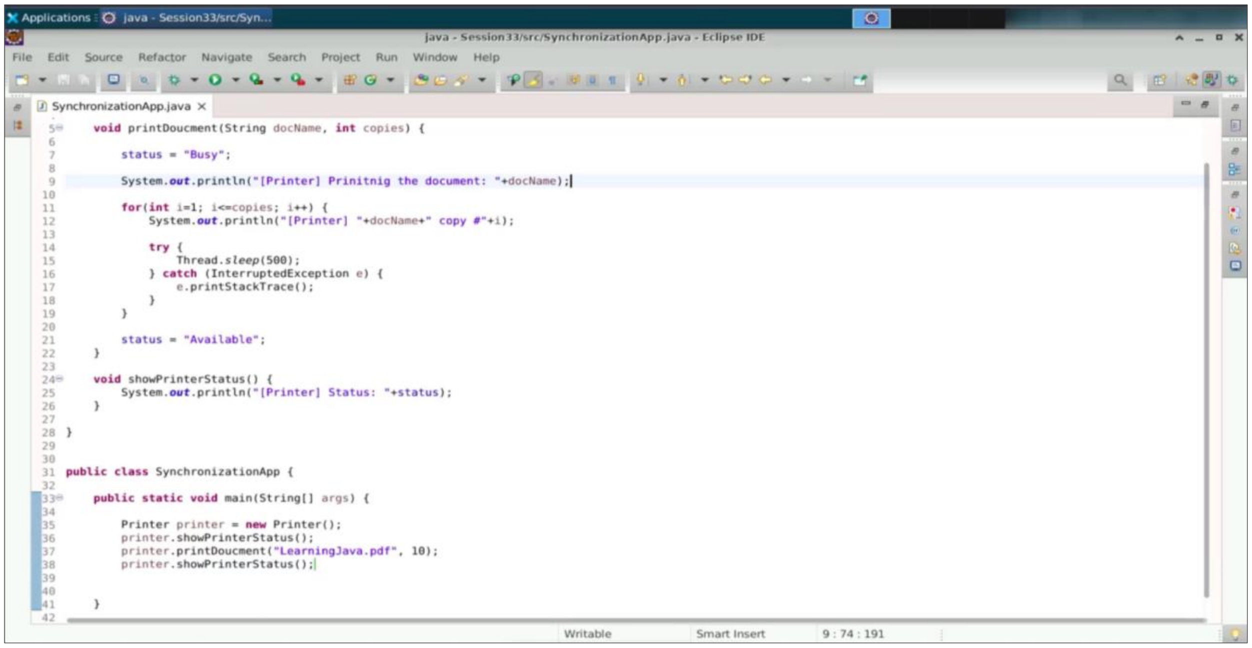
3.3 When the documents are printed, this loop, once it terminates, will make the printer available again. You can add a method called show_printer_status to print out the status of the printer. This method will display the message "Printer status: available". You have simply printed the status for the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


java - Session33/src/SynchronizationApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java

```java
class Printer{

    String status = "Available";

    void printDoucment(String docName, int copies) {

        status = "Busy";

        System.out.println("[Printer] Prinitnig the document: "+docName);

        for(int i=1; i<=copies; i++) {
            System.out.println("[Printer] "+docName+" copy #"+i);

            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        status = "Available";
    }

    void showPrinterStatus() {
        System.out.println("[Printer] Status: "+status);
    }
}


public class SynchronizationApp {
```

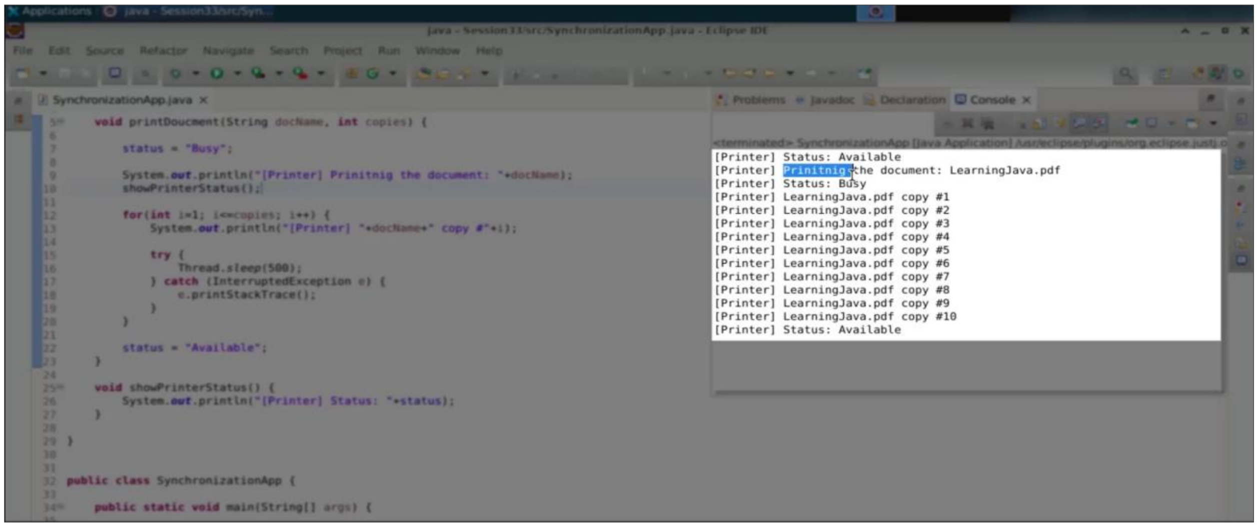
Step 4: Create an object and execute the code

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.


Applications
java - Session33/src/Syn...
java - Session33/src/SynchronizationApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java

```java
    void printDoucment(String docName, int copies) {

        status = "Busy";

        System.out.println("[Printer] Prinitnig the document: "+docName);

        for(int i=1; i<=copies; i++) {
            System.out.println("[Printer] "+docName+" copy #"+i);

            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        status = "Available";
    }

    void showPrinterStatus() {
        System.out.println("[Printer] Status: "+status);
    }

}


public class SynchronizationApp {

    public static void main(String[] args) {

        Printer printer = new Printer();
        printer.showPrinterStatus();
        printer.printDoucment("LearningJava.pdf", 10);
        printer.showPrinterStatus();


    }
```

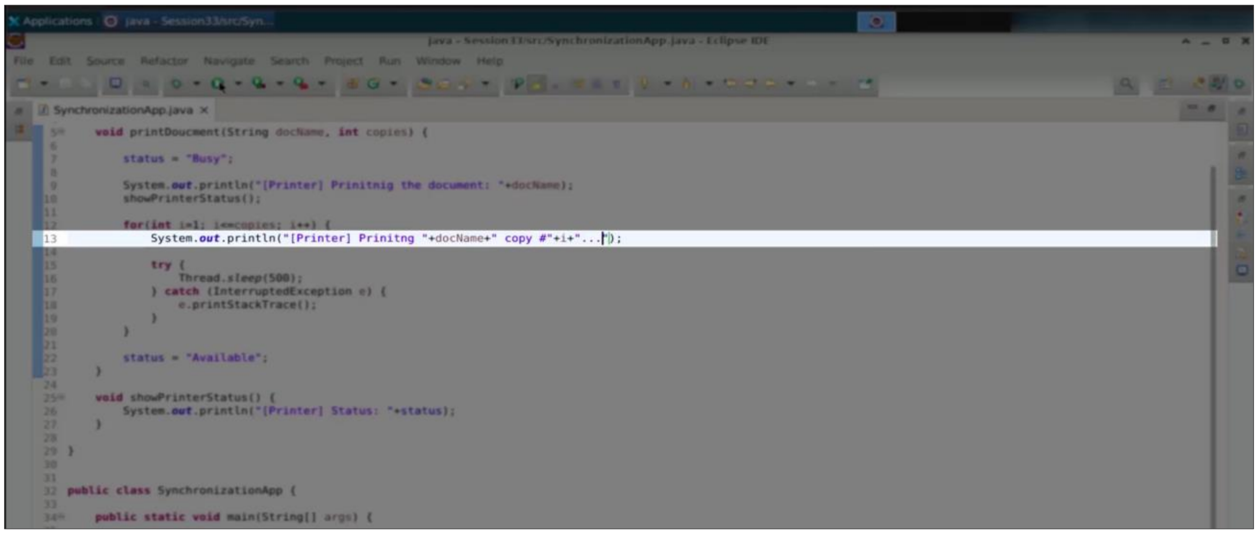
4.2 When you run this code, this is very simple equation, the printer status is available. The printer says, printing the document and here the status goes as busy. Once, you get to see all the copies being printed, then, the status becomes available.



```java
    void printDoucment(String docName, int copies) {

        status = "Busy";

        System.out.println("[Printer] Prinitnig the document: "+docName);
        showPrinterStatus();

        for(int i=1; i<=copies; i++) {
            System.out.println("[Printer] "+docName+" copy #"+i);

            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        status = "Available";
    }

    void showPrinterStatus() {
        System.out.println("[Printer] Status: "+status);
    }

}


public class SynchronizationApp {

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

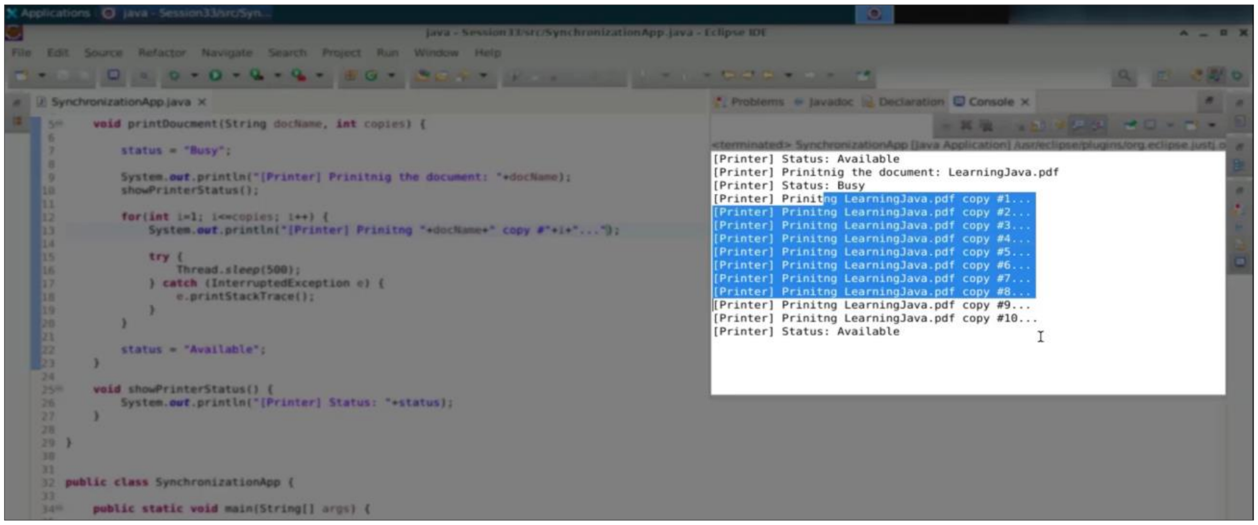
```
<terminated> SynchronizationApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.o
[Printer] Status: Available
[Printer] Prinitnig the document: LearningJava.pdf
[Printer] Status: Busy
[Printer] LearningJava.pdf copy #1
[Printer] LearningJava.pdf copy #2
[Printer] LearningJava.pdf copy #3
[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.



```java
    void printDoucment(String docName, int copies) {

        status = "Busy";

        System.out.println("[Printer] Prinitnig the document: "+docName);
        showPrinterStatus();

        for(int i=1; i<=copies; i++) {
            System.out.println("[Printer] Prinitng "+docName+" copy #"+i+"...");

            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        status = "Available";
    }

    void showPrinterStatus() {
        System.out.println("[Printer] Status: "+status);
    }

}


public class SynchronizationApp {

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

4.4 Thus, this is the printing status as shown.

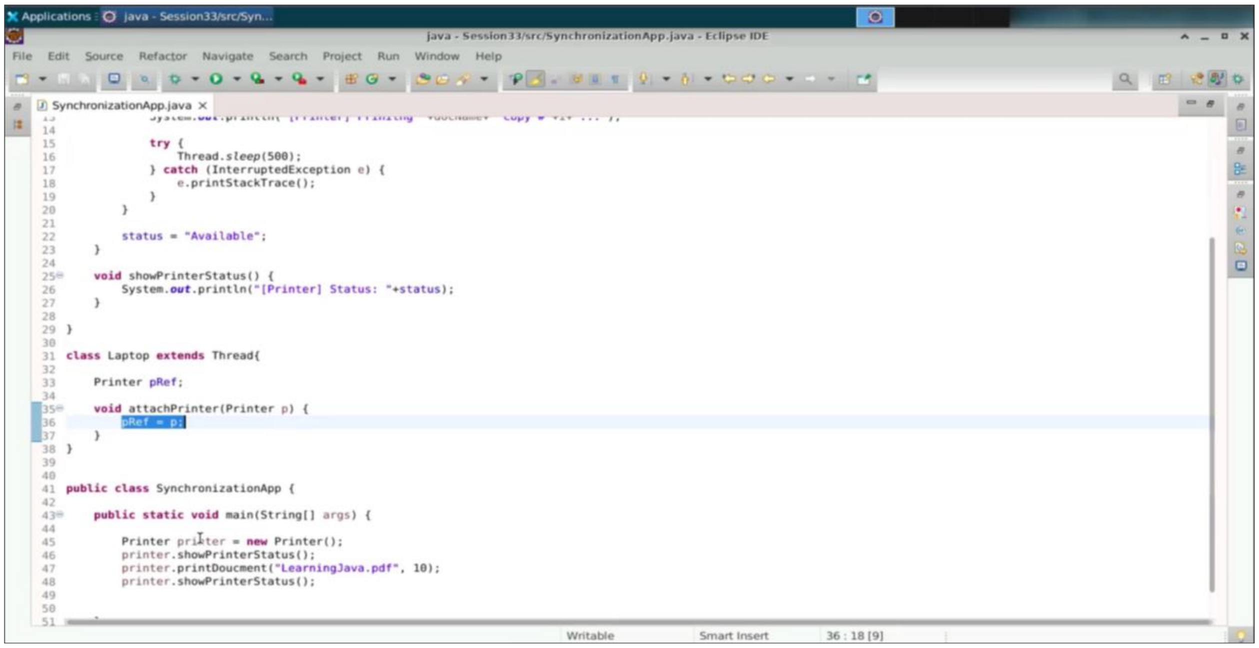


```java
    void printDoucment(String docName, int copies) {

        status = "Busy";

        System.out.println("[Printer] Prinitnig the document: "+docName);
        showPrinterStatus();

        for(int i=1; i<=copies; i++) {
            System.out.println("[Printer] Prinitng "+docName+" copy #"+i+"...");

            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        status = "Available";
    }

    void showPrinterStatus() {
        System.out.println("[Printer] Status: "+status);
    }

}


public class SynchronizationApp {

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

```
<terminated> SynchronizationApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.o
[Printer] Status: Available
[Printer] Prinitnig the document: LearningJava.pdf
[Printer] Status: Busy
[Printer] Prinitng LearningJava.pdf copy #1...
[Printer] Prinitng LearningJava.pdf copy #2...
[Printer] Prinitng LearningJava.pdf copy #3...
[Printer] Prinitng LearningJava.pdf copy #4...
[Printer] Prinitng LearningJava.pdf copy #5...
[Printer] Prinitng LearningJava.pdf copy #6...
[Printer] Prinitng LearningJava.pdf copy #7...
[Printer] Prinitng LearningJava.pdf copy #8...
[Printer] Prinitng LearningJava.pdf copy #9...
[Printer] Prinitng LearningJava.pdf copy #10...
[Printer] Status: Available
```

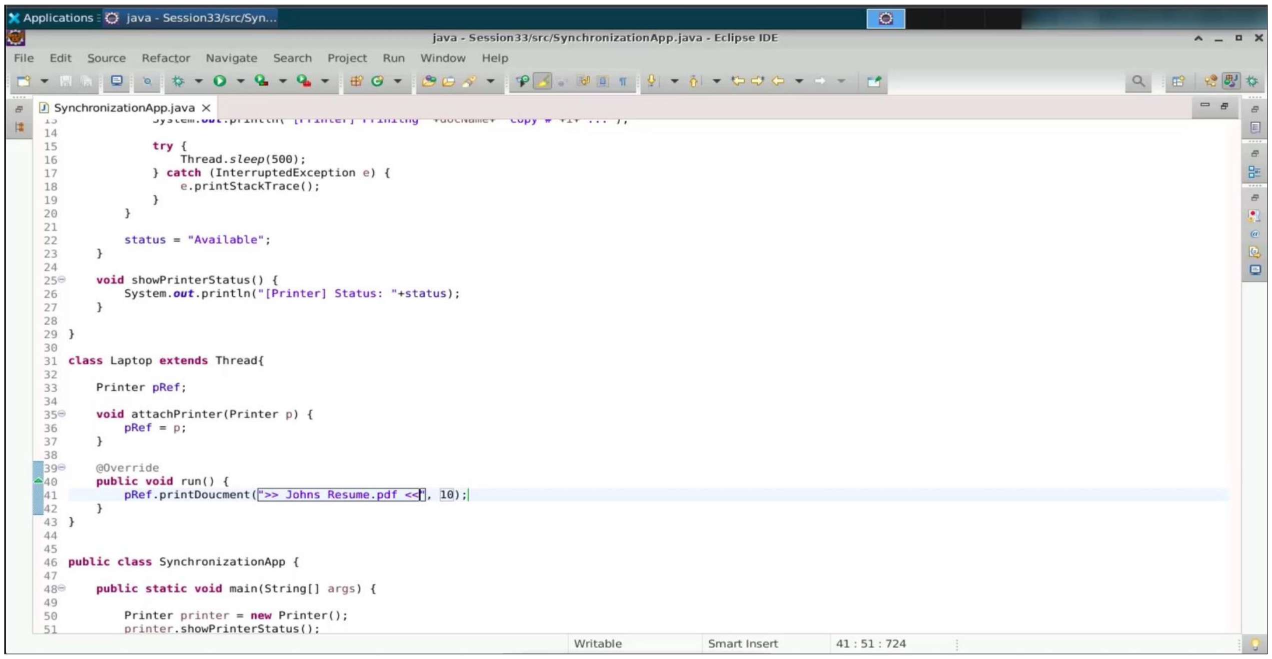
Step 5: Create two threads and work on the same object

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



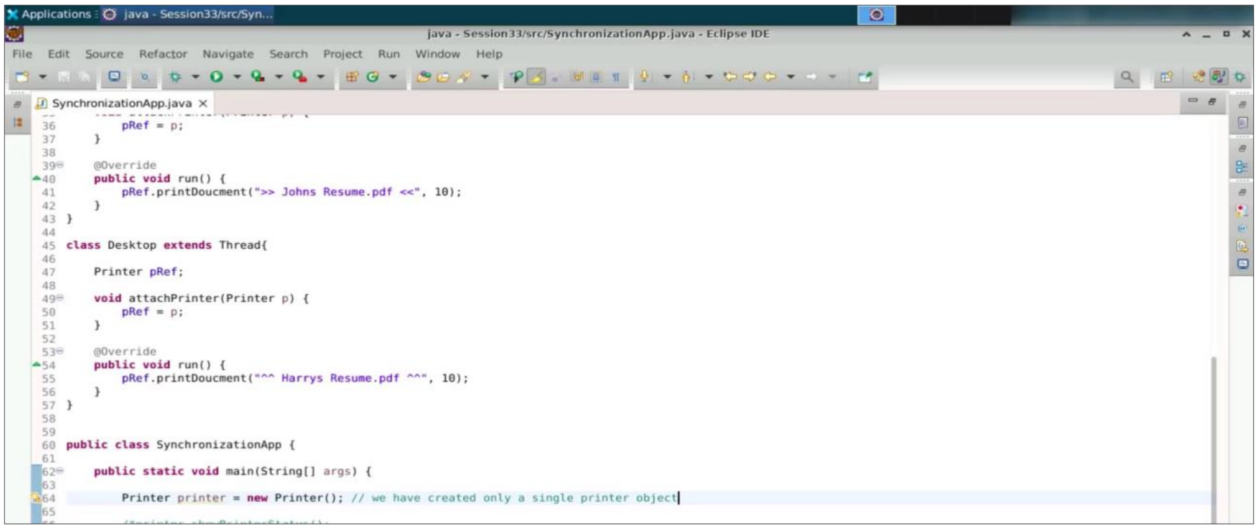
```java
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        status = "Available";
    }

    void showPrinterStatus() {
        System.out.println("[Printer] Status: "+status);
    }

}

class Laptop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }
}


public class SynchronizationApp {

    public static void main(String[] args) {

        Printer printer = new Printer();
        printer.showPrinterStatus();
        printer.printDoucment("LearningJava.pdf", 10);
        printer.showPrinterStatus();
```

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.


java - Session33/src/SynchronizationApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java
Writable
Smart Insert
41 : 51 : 724

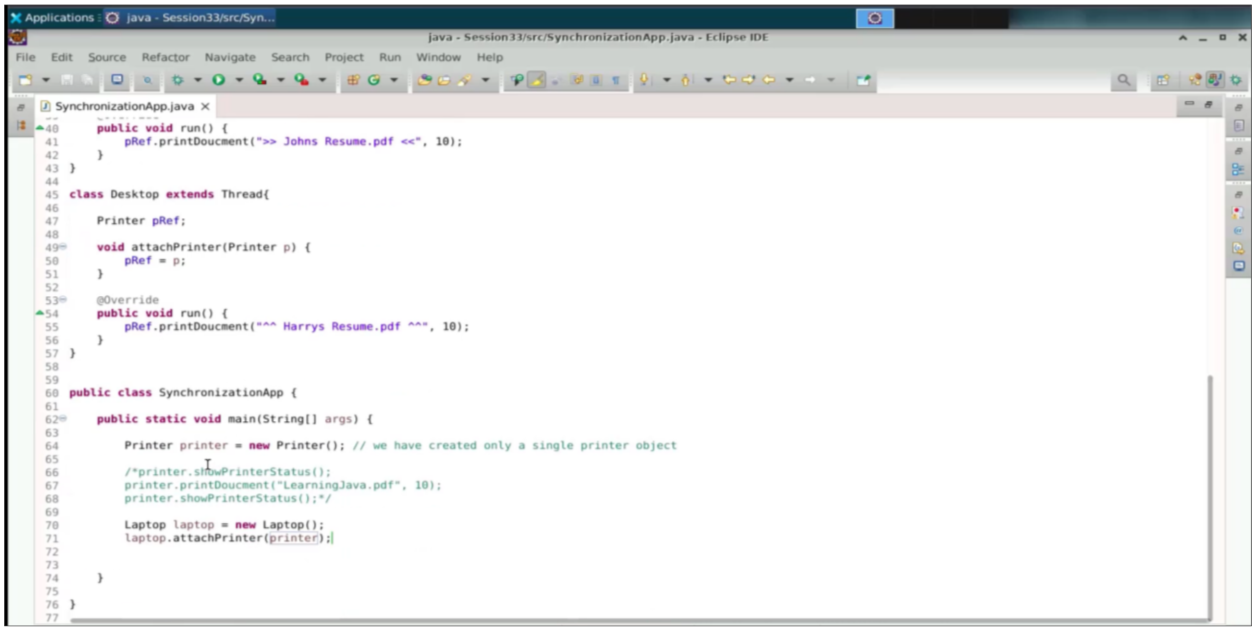
```
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        status = "Available";
    }

    void showPrinterStatus() {
        System.out.println("[Printer] Status: "+status);
    }

}

class Laptop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }

    @Override
    public void run() {
        pRef.printDoucment(">> Johns Resume.pdf <<", 10);
    }
}


public class SynchronizationApp {

    public static void main(String[] args) {

        Printer printer = new Printer();
        printer.showPrinterStatus();
```

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.


java - Session33/src/SynchronizationApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java

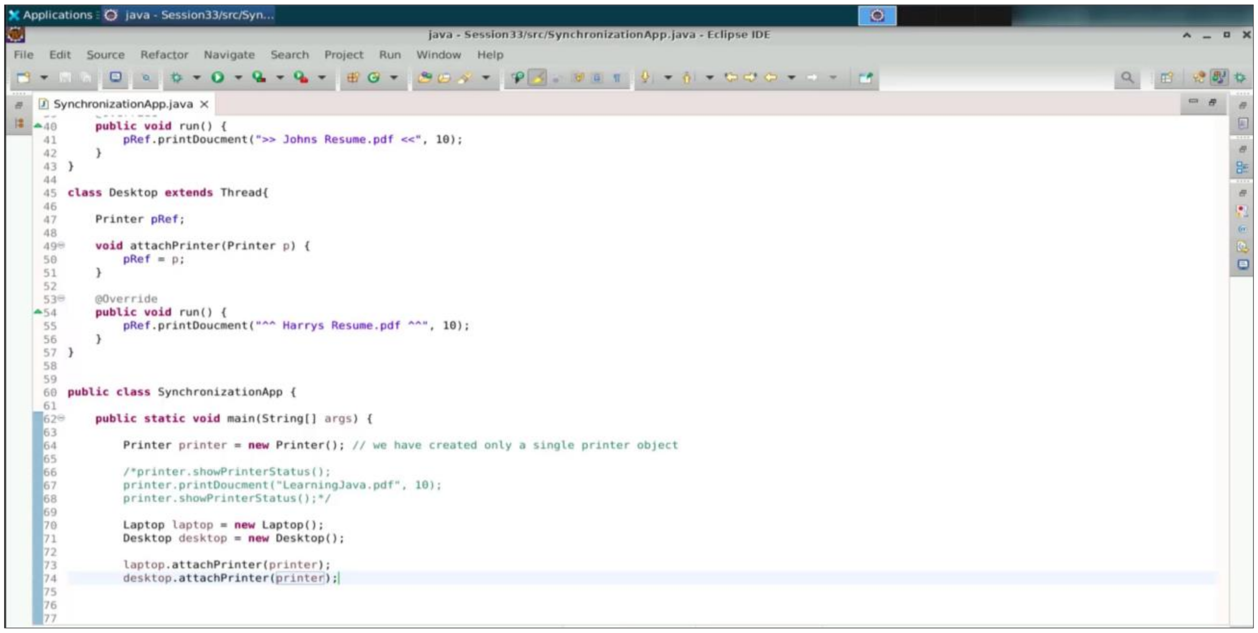
```
        pRef = p;
    }

    @Override
    public void run() {
        pRef.printDoucment(">> Johns Resume.pdf <<", 10);
    }
}

class Desktop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }

    @Override
    public void run() {
        pRef.printDoucment("^^ Harrys Resume.pdf ^^", 10);
    }
}


public class SynchronizationApp {

    public static void main(String[] args) {

        Printer printer = new Printer(); // we have created only a single printer object

```

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.


Applications
java - Session33/src/Syn...
java - Session33/src/SynchronizationApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java

```java
    public void run() {
        pRef.printDoucment(">> Johns Resume.pdf <<", 10);
    }
}

class Desktop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }

    @Override
    public void run() {
        pRef.printDoucment("^^ Harrys Resume.pdf ^^", 10);
    }
}


public class SynchronizationApp {

    public static void main(String[] args) {

        Printer printer = new Printer(); // we have created only a single printer object

        /*printer.showPrinterStatus();
        printer.printDoucment("LearningJava.pdf", 10);
        printer.showPrinterStatus();*/

        Laptop laptop = new Laptop();
        laptop.attachPrinter(printer);


    }

}
```

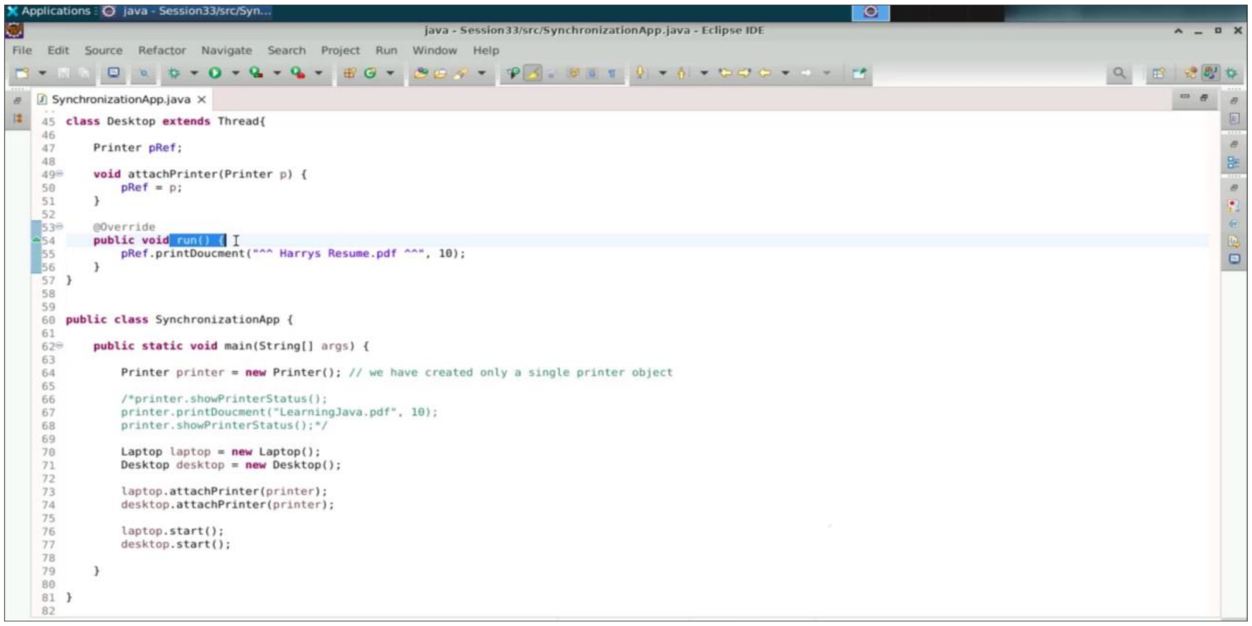
5.5 Next, create the Desktop object by writing Desktop desktop = new Desktop();. Illustrate better usage, create both the Laptop and Desktop objects. Then, attach the same Printer to the Desktop by calling desktop.attach_printer(printer);. This demonstrates that both the Laptop and the Desktop are working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


Applications
java - Session33/src/Syn...
java - Session33/src/SynchronizationApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
SynchronizationApp.java

```java
    public void run() {
        pRef.printDoucment(">> Johns Resume.pdf <<", 10);
    }
}

class Desktop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }

    @Override
    public void run() {
        pRef.printDoucment("^^ Harrys Resume.pdf ^^", 10);
    }
}


public class SynchronizationApp {

    public static void main(String[] args) {

        Printer printer = new Printer(); // we have created only a single printer object

        /*printer.showPrinterStatus();
        printer.printDoucment("LearningJava.pdf", 10);
        printer.showPrinterStatus();*/

        Laptop laptop = new Laptop();
        Desktop desktop = new Desktop();

        laptop.attachPrinter(printer);
        desktop.attachPrinter(printer);
```

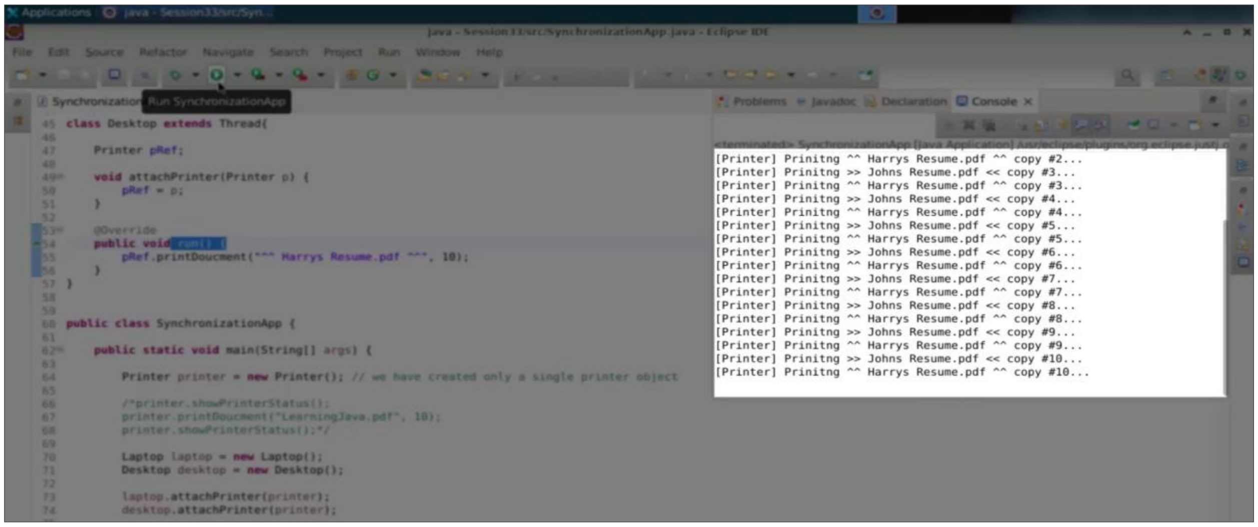
Step 6: Execute the code with sample data

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.



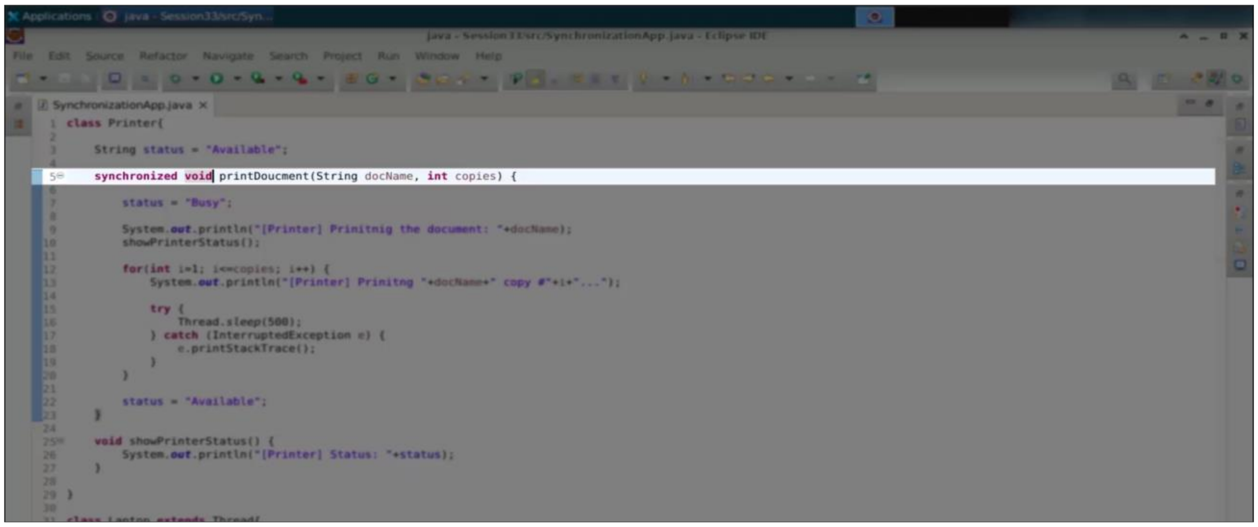
```java
class Desktop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }

    @Override
    public void run() {
        pRef.printDoucment("^^ Harrys Resume.pdf ^^", 10);
    }
}


public class SynchronizationApp {

    public static void main(String[] args) {

        Printer printer = new Printer(); // we have created only a single printer object

        /*printer.showPrinterStatus();
        printer.printDoucment("LearningJava.pdf", 10);
        printer.showPrinterStatus();*/

        Laptop laptop = new Laptop();
        Desktop desktop = new Desktop();

        laptop.attachPrinter(printer);
        desktop.attachPrinter(printer);

        laptop.start();
        desktop.start();

    }
}
```

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.



```
<terminated> SynchronizationApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.o
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #2...
[Printer] Prinitng >> Johns Resume.pdf << copy #3...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #3...
[Printer] Prinitng >> Johns Resume.pdf << copy #4...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #4...
[Printer] Prinitng >> Johns Resume.pdf << copy #5...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #5...
[Printer] Prinitng >> Johns Resume.pdf << copy #6...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #6...
[Printer] Prinitng >> Johns Resume.pdf << copy #7...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #7...
[Printer] Prinitng >> Johns Resume.pdf << copy #8...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #8...
[Printer] Prinitng >> Johns Resume.pdf << copy #9...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #9...
[Printer] Prinitng >> Johns Resume.pdf << copy #10...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #10...
```

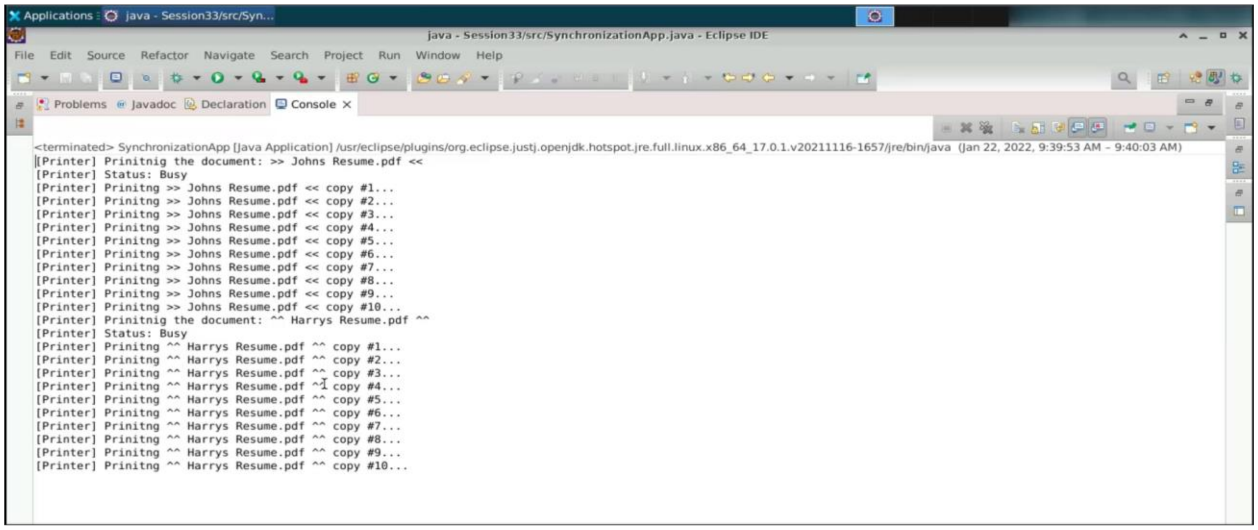
6.3 Let’s come here and first mark the printer to go as synchronized. When the method is marked as synchronized, it particularly means, the other thread cannot execute the same method in the same object till time the first thread has not finished its execution.



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

```java
class Printer{

    String status = "Available";

    synchronized void printDoucment(String docName, int copies) {

        status = "Busy";

        System.out.println("[Printer] Prinitnig the document: "+docName);
        showPrinterStatus();

        for(int i=1; i<=copies; i++) {
            System.out.println("[Printer] Prinitng "+docName+" copy #"+i+"...");

            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        status = "Available";
    }

    void showPrinterStatus() {
        System.out.println("[Printer] Status: "+status);
    }

}

class Laptop extends Thread{
```

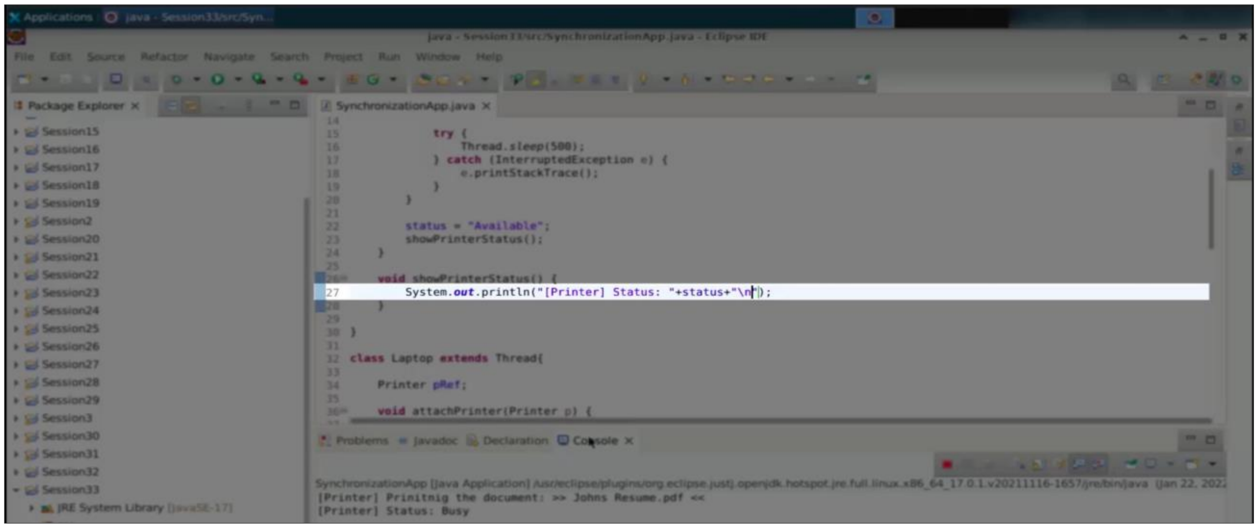
6.4 Let us run the code, to understand this discussion better. As shown only John’s resume is getting printed and thereafter Harry’s resume is getting printed. That is where you understand the fundamental of synchronization.



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

```
<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] Prinitnig the document: >> Johns Resume.pdf <<
[Printer] Status: Busy
[Printer] Prinitng >> Johns Resume.pdf << copy #1...
[Printer] Prinitng >> Johns Resume.pdf << copy #2...
[Printer] Prinitng >> Johns Resume.pdf << copy #3...
[Printer] Prinitng >> Johns Resume.pdf << copy #4...
[Printer] Prinitng >> Johns Resume.pdf << copy #5...
[Printer] Prinitng >> Johns Resume.pdf << copy #6...
[Printer] Prinitng >> Johns Resume.pdf << copy #7...
[Printer] Prinitng >> Johns Resume.pdf << copy #8...
[Printer] Prinitng >> Johns Resume.pdf << copy #9...
[Printer] Prinitng >> Johns Resume.pdf << copy #10...
[Printer] Prinitnig the document: ^^ Harrys Resume.pdf ^^
[Printer] Status: Busy
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #1...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #2...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #3...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #4...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #5...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #6...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #7...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #8...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #9...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #10...
```

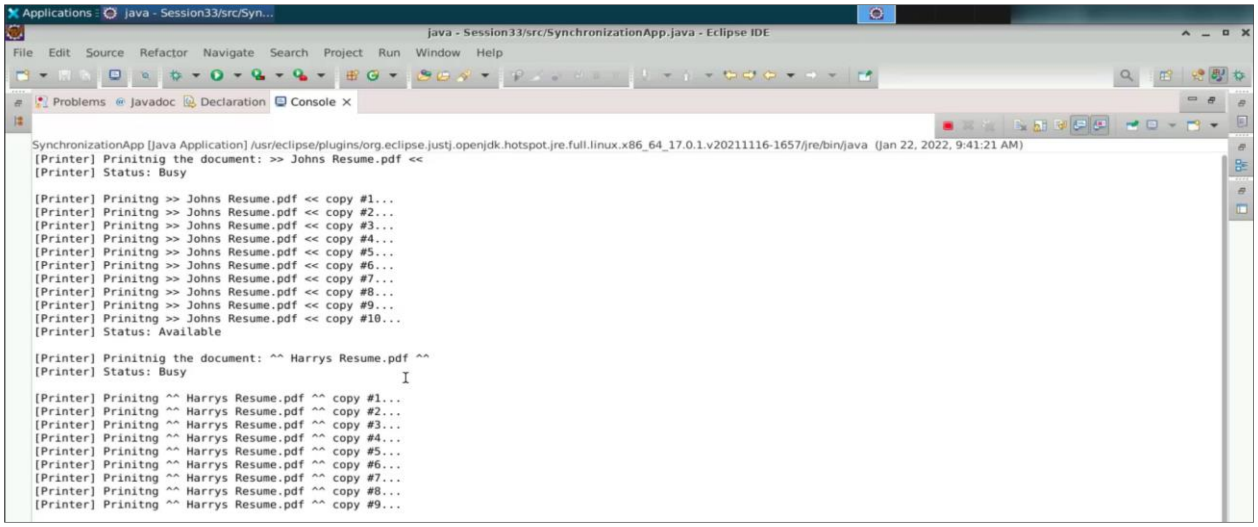
6.5 Let’s give the printer status once again and make it to available. Then, come here and do one empty print line, with a \n.



```
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        status = "Available";
        showPrinterStatus();
    }

    void showPrinterStatus() {
        System.out.println("[Printer] Status: "+status+"\n");
    }

}

class Laptop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
```

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

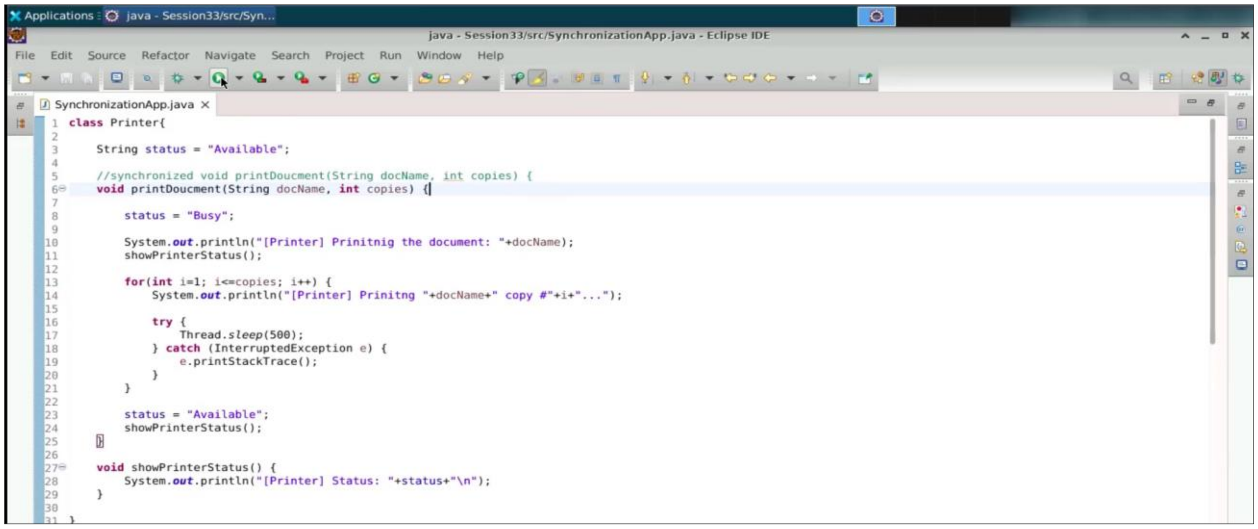
6.6 Rerun the code and check the output. It shows, the printer status as busy and now it is made available, hence printing the document with the Harry’s resume status went busy that is, it started printing again. And now it is available again.



```
SynchronizationApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java (Jan 22, 2022, 9:41:21 AM)
[Printer] Prinitnig the document: >> Johns Resume.pdf <<
[Printer] Status: Busy

[Printer] Prinitng >> Johns Resume.pdf << copy #1...
[Printer] Prinitng >> Johns Resume.pdf << copy #2...
[Printer] Prinitng >> Johns Resume.pdf << copy #3...
[Printer] Prinitng >> Johns Resume.pdf << copy #4...
[Printer] Prinitng >> Johns Resume.pdf << copy #5...
[Printer] Prinitng >> Johns Resume.pdf << copy #6...
[Printer] Prinitng >> Johns Resume.pdf << copy #7...
[Printer] Prinitng >> Johns Resume.pdf << copy #8...
[Printer] Prinitng >> Johns Resume.pdf << copy #9...
[Printer] Prinitng >> Johns Resume.pdf << copy #10...
[Printer] Status: Available

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

[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #1...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #2...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #3...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #4...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #5...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #6...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #7...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #8...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #9...
```

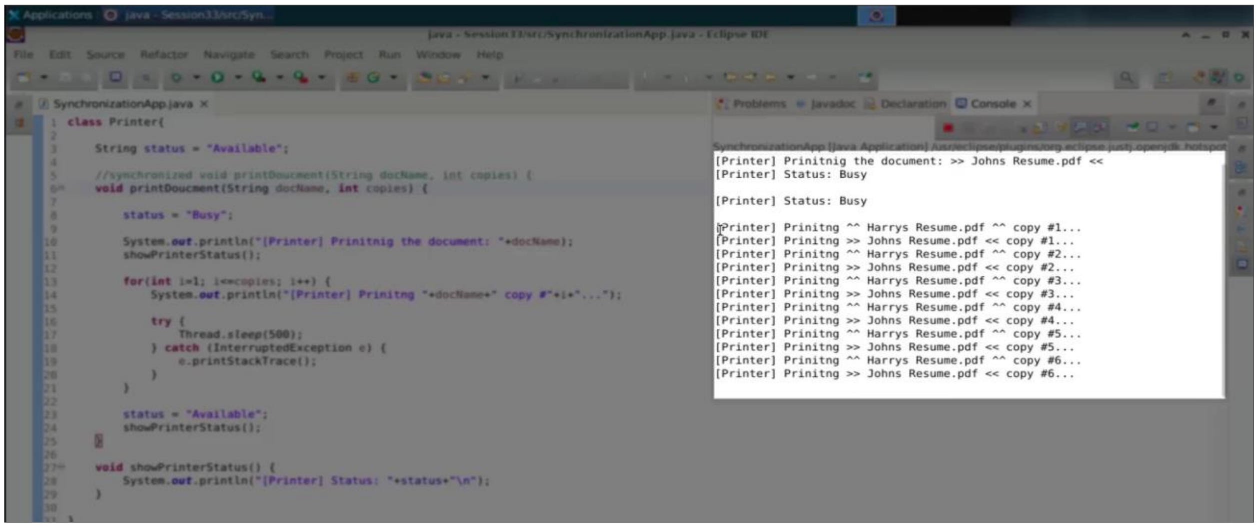
Step 7: Implement the concept of synchronization

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



```java
class Printer{

    String status = "Available";

    //synchronized void printDoucment(String docName, int copies) {
    void printDoucment(String docName, int copies) {

        status = "Busy";

        System.out.println("[Printer] Prinitnig the document: "+docName);
        showPrinterStatus();

        for(int i=1; i<=copies; i++) {
            System.out.println("[Printer] Prinitng "+docName+" copy #"+i+"...");

            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        status = "Available";
        showPrinterStatus();
    }

    void showPrinterStatus() {
        System.out.println("[Printer] Status: "+status+"\n");
    }

}
```

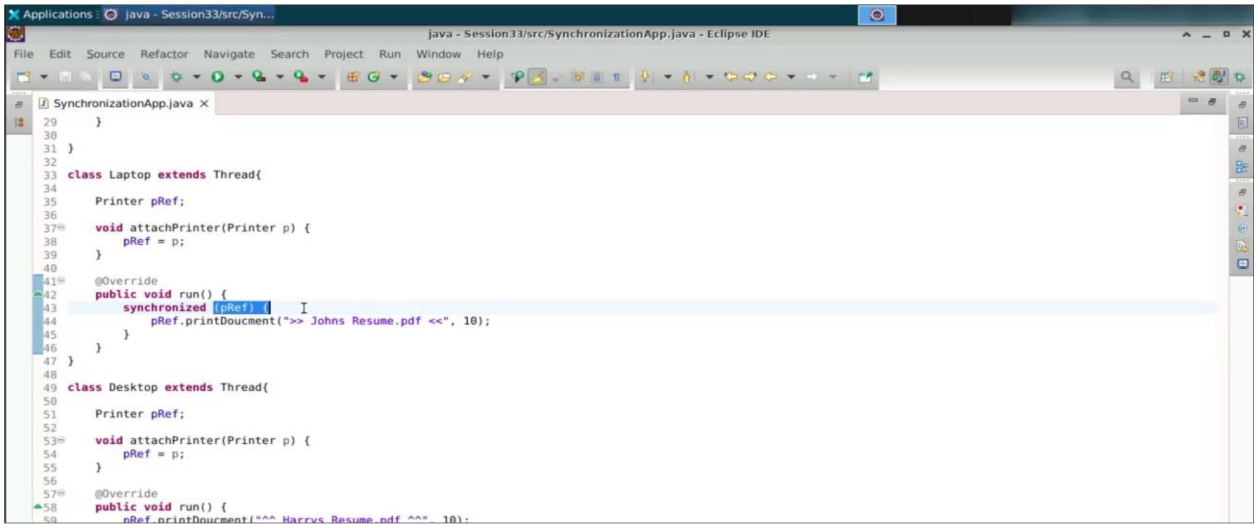
7.2 The other thing is let your method be working in the same way as a regular method. This means that both the threads will now access it in parallel. Thus, the printer is again messed up.



```java
class Printer{

    String status = "Available";

    //synchronized void printDoucment(String docName, int copies) {
    void printDoucment(String docName, int copies) {

        status = "Busy";

        System.out.println("[Printer] Prinitnig the document: "+docName);
        showPrinterStatus();

        for(int i=1; i<=copies; i++) {
            System.out.println("[Printer] Prinitng "+docName+" copy #"+i+"...");

            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        status = "Available";
        showPrinterStatus();
    }

    void showPrinterStatus() {
        System.out.println("[Printer] Status: "+status+"\n");
    }
```

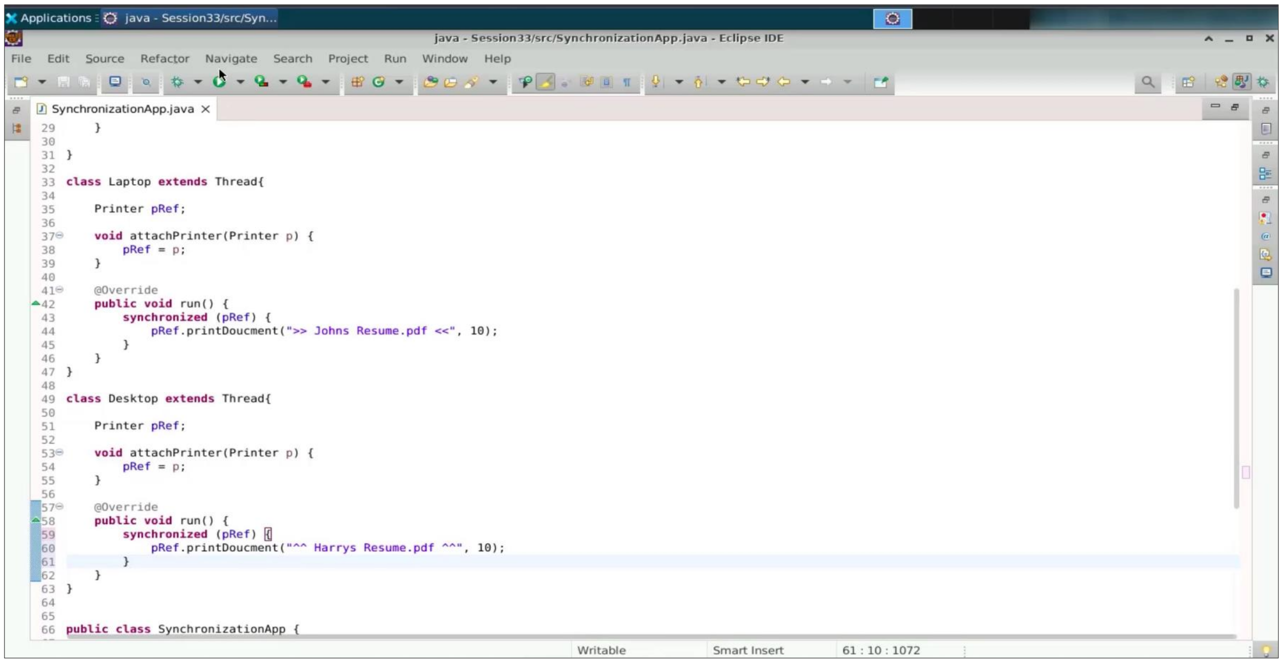
```
SynchronizationApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot
[Printer] Prinitnig the document: >> Johns Resume.pdf <<
[Printer] Status: Busy

[Printer] Status: Busy

[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #1...
[Printer] Prinitng >> Johns Resume.pdf << copy #1...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #2...
[Printer] Prinitng >> Johns Resume.pdf << copy #2...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #3...
[Printer] Prinitng >> Johns Resume.pdf << copy #3...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #4...
[Printer] Prinitng >> Johns Resume.pdf << copy #4...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #5...
[Printer] Prinitng >> Johns Resume.pdf << copy #5...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #6...
[Printer] Prinitng >> 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.


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

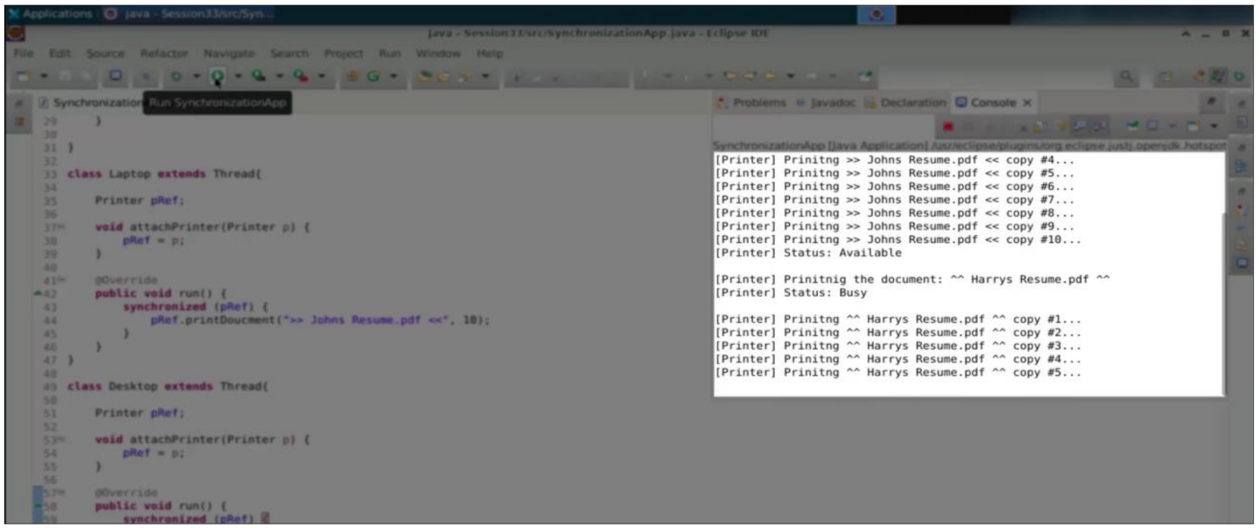
```
    }

}

class Laptop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }

    @Override
    public void run() {
        synchronized (pRef) {
            pRef.printDoucment(">> Johns Resume.pdf <<", 10);
        }
    }
}

class Desktop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }

    @Override
    public void run() {
        pRef.printDoucment("^^ Harrys Resume.pdf ^^", 10);
```

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.


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

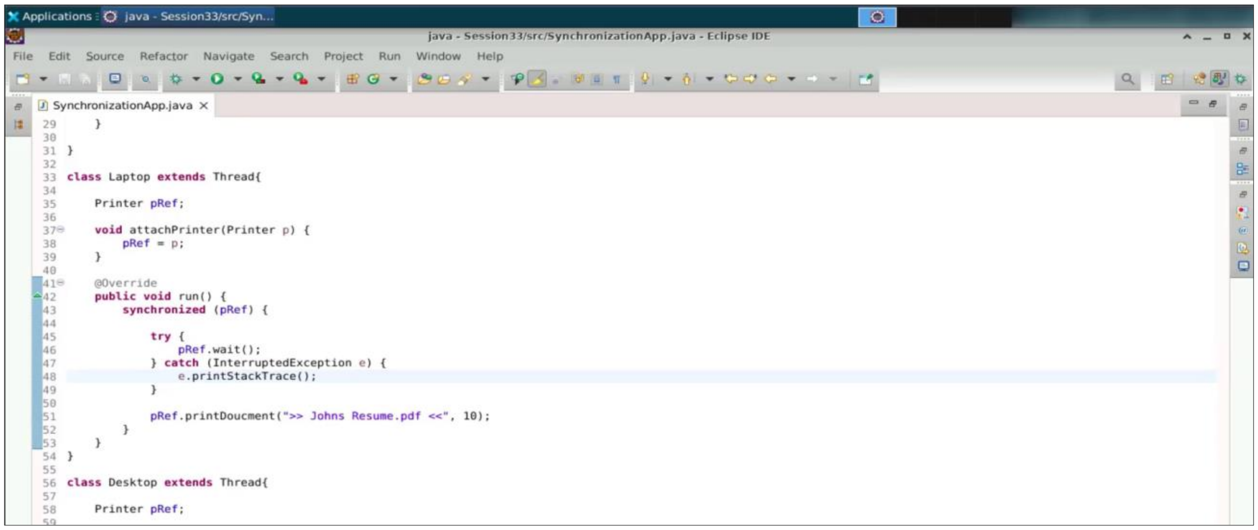
```
    }

}

class Laptop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }

    @Override
    public void run() {
        synchronized (pRef) {
            pRef.printDoucment(">> Johns Resume.pdf <<", 10);
        }
    }
}

class Desktop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }

    @Override
    public void run() {
        synchronized (pRef) {
            pRef.printDoucment("^^ Harrys Resume.pdf ^^", 10);
        }
    }
}


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.



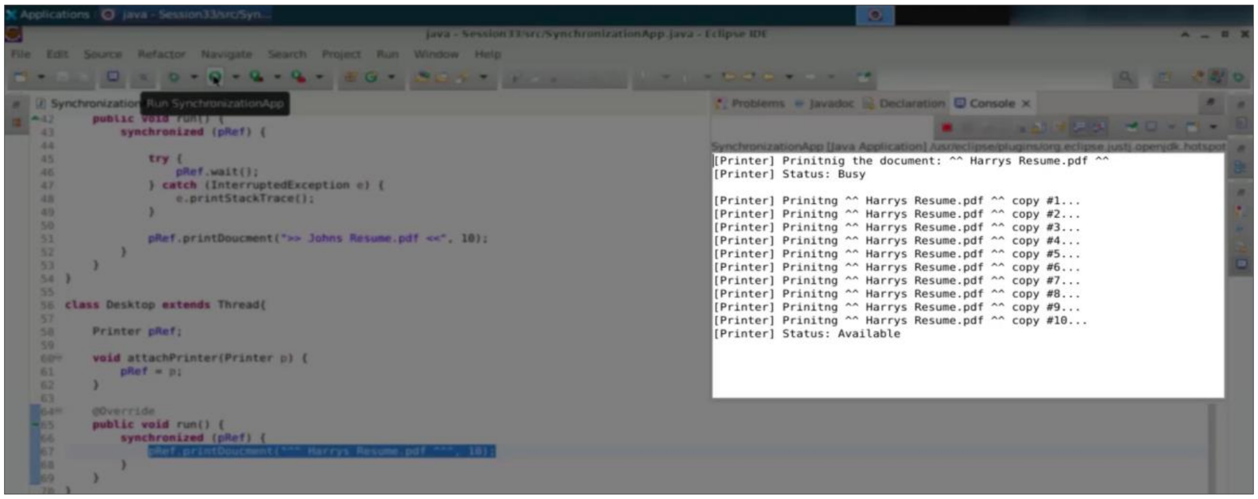
```java
    }

}

class Laptop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }

    @Override
    public void run() {
        synchronized (pRef) {
            pRef.printDoucment(">> Johns Resume.pdf <<", 10);
        }
    }
}

class Desktop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }

    @Override
    public void run() {
        synchronized (pRef) {
```

```
SynchronizationApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot
[Printer] Prinitng >> Johns Resume.pdf << copy #4...
[Printer] Prinitng >> Johns Resume.pdf << copy #5...
[Printer] Prinitng >> Johns Resume.pdf << copy #6...
[Printer] Prinitng >> Johns Resume.pdf << copy #7...
[Printer] Prinitng >> Johns Resume.pdf << copy #8...
[Printer] Prinitng >> Johns Resume.pdf << copy #9...
[Printer] Prinitng >> Johns Resume.pdf << copy #10...
[Printer] Status: Available

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

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



```java
    }

}

class Laptop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }

    @Override
    public void run() {
        synchronized (pRef) {

            try {
                pRef.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            pRef.printDoucment(">> Johns Resume.pdf <<", 10);
        }
    }
}

class Desktop extends Thread{

    Printer pRef;
```

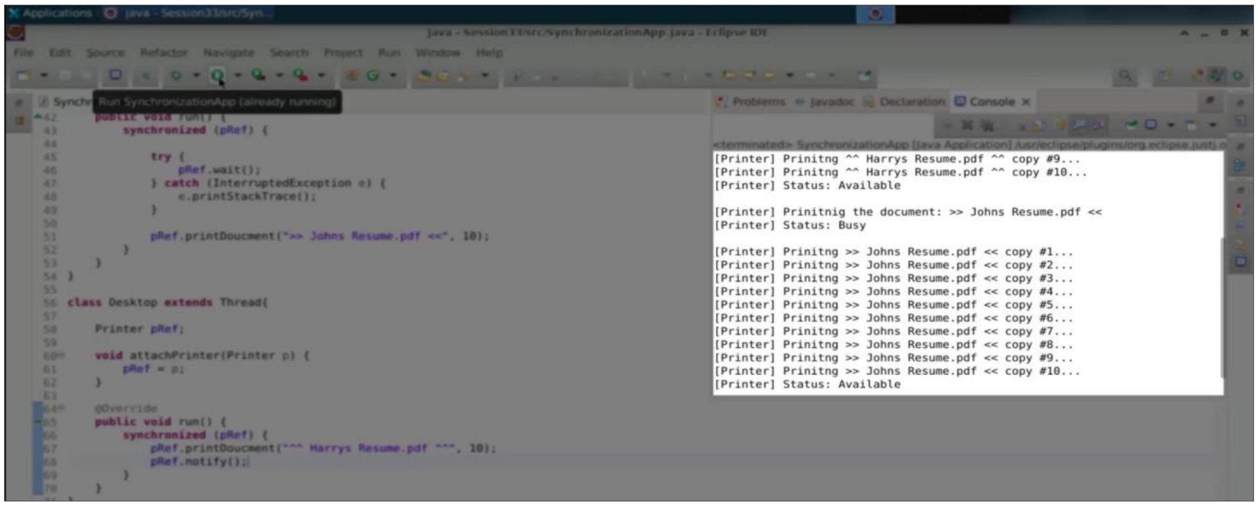
7.6 Now the output here shows that Harry's resume is getting printed before the John's resume and you do not see any output coming from the John's resume. This is a situation where you have made your laptop thread to wait infinitely. When the laptop thread is waiting in an infinite way, it means you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



```
        public void run() {
            synchronized (pRef) {

                try {
                    pRef.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

                pRef.printDoucment(">> Johns Resume.pdf <<", 10);
            }
        }
}

class Desktop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }

    @Override
    public void run() {
        synchronized (pRef) {
            pRef.printDoucment("^^ Harrys Resume.pdf ^^", 10);
        }
    }
```

```
SynchronizationApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot
[Printer] Prinitnig the document: ^^ Harrys Resume.pdf ^^
[Printer] Status: Busy

[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #1...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #2...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #3...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #4...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #5...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #6...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #7...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #8...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #9...
[Printer] Prinitng ^^ 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.



```
        public void run() {
            synchronized (pRef) {

                try {
                    pRef.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

                pRef.printDoucment(">> Johns Resume.pdf <<", 10);
            }
        }
}

class Desktop extends Thread{

    Printer pRef;

    void attachPrinter(Printer p) {
        pRef = p;
    }

    @Override
    public void run() {
        synchronized (pRef) {
            pRef.printDoucment("^^ Harrys Resume.pdf ^^", 10);
            pRef.notify();
        }
    }
```

```
<terminated> SynchronizationApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.o
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #9...
[Printer] Prinitng ^^ Harrys Resume.pdf ^^ copy #10...
[Printer] Status: Available

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

[Printer] Prinitng >> Johns Resume.pdf << copy #1...
[Printer] Prinitng >> Johns Resume.pdf << copy #2...
[Printer] Prinitng >> Johns Resume.pdf << copy #3...
[Printer] Prinitng >> Johns Resume.pdf << copy #4...
[Printer] Prinitng >> Johns Resume.pdf << copy #5...
[Printer] Prinitng >> Johns Resume.pdf << copy #6...
[Printer] Prinitng >> Johns Resume.pdf << copy #7...
[Printer] Prinitng >> Johns Resume.pdf << copy #8...
[Printer] Prinitng >> Johns Resume.pdf << copy #9...
[Printer] Prinitng >> Johns Resume.pdf << copy #10...
[Printer] Status: Available
```

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