Assignment 2:

Write a program which implements threads with locks by using synchronized keyword.

Source Code:

```
// Scenerio is : Multiple Threads(Employees) Working on Same Printer Objects.
// Result:It makes a problem because of Ambigious Printer Object, thus it makes
class Printer{
    // Synchronized method: Makes lock on object of printer
    // 1) --> synchronized void printDocuments(String author, int noOfCopies){
    void printDocuments(String author, int noOfCopies){
        for(int i = 0; i < noOfCopies; i++){
            System.out.println("Printing Copy no: " + (i + 1) + " of Author: " + author);
        }
    }
}

class Emp1 extends Thread{
    Printer mRef;

    Emp1(Printer p){
        mRef = p;
    }

    public void run(){
        synchronized(mRef){ // 2--> Synchronized block: Makes lock on object of printer
        mRef.printDocuments("Akshit", 10);
        }
    }

}

class Emp2 extends Thread{
    Printer mRef;

    Emp2(Printer p){
        mRef = p;
    }

    public void run(){
        synchronized(mRef){ // 2--> Synchronized block: Makes lock on object of printer
        mRef.printDocuments("Amisha Sahu", 10);
        }
    }
}

public class SynchronizedPrinter{
    public static void main(String[] args) {
       System.out.println(" ----- App Started ---- ");

       // Calling Printer in the background to print copies:
       Printer printer = new Printer();
       Emp1 obj = new Emp1(printer);
       obj.start();

// Solution 1: Using join() method before starting Emp2 so it will wait for Emp1 to
finish:
       // Drawback: we cannot put join on each method if there are many Emp's to finish.

       /*
       try{
       obj.join(); // Waiting for Emp1 to finish
       }
       catch(Exception e){
              System.out.println(e);
       }*/
       Emp2 obj2 = new Emp2(printer); // Emp2 using the same printer.
       obj2.start();

       // Solution 2: Using synchronized keyword on Printer class method:
       // By this: it acquires an intrisic lock on the the Printer object.
       // So no other thread can acquire the lock on the same object.
       // This is called as a "Synchronized Method".

       System.out.println(" ---- App Finished ---- ");
    }
}
```

Output:

```
----- App Started ----

---- App Finished ----
Printing Copy no: 1 of Author: Akshit
Printing Copy no: 2 of Author: Akshit
Printing Copy no: 3 of Author: Akshit
Printing Copy no: 4 of Author: Akshit
Printing Copy no: 5 of Author: Akshit
Printing Copy no: 6 of Author: Akshit
Printing Copy no: 7 of Author: Akshit
Printing Copy no: 8 of Author: Akshit
Printing Copy no: 9 of Author: Akshit
Printing Copy no: 10 of Author: Akshit
Printing Copy no: 1 of Author: Amisha Sahu
Printing Copy no: 2 of Author: Amisha Sahu
Printing Copy no: 3 of Author: Amisha Sahu
Printing Copy no: 4 of Author: Amisha Sahu
Printing Copy no: 5 of Author: Amisha Sahu
Printing Copy no: 6 of Author: Amisha Sahu
Printing Copy no: 7 of Author: Amisha Sahu
Printing Copy no: 8 of Author: Amisha Sahu
Printing Copy no: 9 of Author: Amisha Sahu
Printing Copy no: 10 of Author: Amisha Sahu
```

Write a program which elaborates the concept of producer consumer problem using wait() ,notify() & all required functionalities in it.

```java
import java.util.LinkedList;
import java.util.Queue;
import java.util.Random;

public class ProducerConsumerTest {

    public static void main(String[] args) throws InterruptedException {

        final Queue sharedQ = new LinkedList<Integer>();

        Thread consumerThread = new Thread(new Consumer(sharedQ, 2), "CONSUMER");
        Thread producerThread = new Thread(new Producer(sharedQ, 2), "PRODUCER");

        producerThread.start();
        consumerThread.start();

    }

}

class Producer implements Runnable {
    private final Queue sharedQ;
    private int maxSize;

    public Producer(Queue sharedQ, int maxSize) {
        this.sharedQ = sharedQ;
        this.maxSize = maxSize;
    }

    @Override
    public void run() {

        while (true) {
            synchronized (sharedQ) {
                while (sharedQ.size() == maxSize) {
                    try {
                        System.out.println("Queue is full");
                        sharedQ.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }

                }
                Random random = new Random();
                int number = random.nextInt(100);
                System.out.println("Producing value " + number);
                sharedQ.add(number);
                sharedQ.notify();
            }

        }
    }
}

class Consumer implements Runnable {
    private final Queue sharedQ;
    private int maxSize;

    public Consumer(Queue sharedQ, int maxSize) {
        this.sharedQ = sharedQ;
        this.maxSize = maxSize;
    }

    @Override
    public void run() {
        while (true) {
            synchronized (sharedQ) {
                while (sharedQ.isEmpty()) {
                    try {
                        System.out.println("Que is Empty");
                        sharedQ.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }

                int number = (int) sharedQ.poll();
                System.out.println("removing Element " + number);
                sharedQ.notify();

            }
        }
    }

}
```

Output:

```
Producing value 75
Producing value 65
removing Element 75
removing Element 65
Que is Empty
Producing value 68
Producing value 2
removing Element 68
removing Element 2
Que is Empty
Producing value 40
Producing value 86
Queue is full
```

```
removing Element 40
removing Element 86
Que is Empty
Producing value 84
Producing value 43
removing Element 84
removing Element 43
Que is Empty
Producing value 36
Producing value 39
removing Element 36
removing Element 39
Que is Empty
Producing value 57
Producing value 81
Queue is full
removing Element 57
removing Element 81
Producing value 45
Producing value 5
Queue is full
removing Element 45
removing Element 5
Que is Empty
```

Write a program with data structure ,use atomic methods like get(),incrementAndGet(),decrementAndGet(),compareAndSet(),etc ,also use all other functionalities to make the program more responsive.

Main.java

```java
import java.util.List;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.TimeUnit;

import package6.Employe;
import package6.Employegen;
import package6.Thread;

class Main {
    public static void main(String args[]) {
        Employegen gen = new Employegen();
        List<Employe> employes = gen.generate(10);
        Thread thread = new Thread(employes, 0, employes.size(), 0.20);
        for (int i = 0; i < employes.size(); i++) {
            Employe employ = employes.get(i);
            System.out.printf("Employe %s: %f \n", employ.getName(), employ.getSalary());
        }
        System.out
```

```
                .println("----------------------------------------------------------------
--------------------------");
        System.out.println("To Increase the salary of Employes");
        System.out
                .println("----------------------------------------------------------------
--------------------------");
        ForkJoinPool pool = new ForkJoinPool();

        pool.execute(thread);
        do {
            System.out.printf("**************************************\n");
            System.out.printf("Main: Pralleism:%d\n", pool.getCommonPoolParallelism());
        } while (!thread.isDone());
        pool.shutdown();

        if (thread.isCompletedNormally()) {
            System.out.println("Main: The process has completed normally. \n");
        }
        for (int i = 0; i < employes.size(); i++) {
            Employe employ = employes.get(i);
            System.out.printf("Employe %s: %f \n", employ.getName(), employ.getSalary());
        }
    }
}
```

Employe.java

```
public class Employe {
    private int empid;
    private double empsalary;
    private String empname;

    public String getName() {
        return empname;
    }

    public void setName(String name) {
        this.empname = name;
    }

    public double getSalary() {
        return empsalary;
    }

    public void setSalary(double salary) {
        this.empsalary = salary;
    }

    public int getId() {
        return empid;
    }
```

```java
    public void setId(int id) {
        this.empid = id;
    }
}
```

Employegen.java

```java
import java.util.concurrent.atomic.AtomicInteger;
import java.util.*;

public class Employegen {
    public List<Employe> generate(int size) {
        List<Employe> emp = new ArrayList<Employe>();
        AtomicInteger val = new AtomicInteger(0);
        AtomicInteger val1 = new AtomicInteger(20000);
        for (int i = 0; i < size; i++) {
            Employe employe = new Employe();
            employe.setName("emp" + (i + 1));
            employe.setId(val.incrementAndGet());
            employe.setSalary(val1.decrementAndGet());
            emp.add(employe);
        }
        return emp;
    }
}
```

Thread.java

```java
import java.util.*;
import java.util.concurrent.RecursiveAction;

public class Thread extends RecursiveAction {
    private List<Employe> employes;
    private int first;
    private int last;
    private double increment;

    public Thread(List<Employe> Employes, int first, int last, double increment) {
        this.employes = Employes;
        this.first = first;
        this.last = last;
        this.increment = increment;
    }

    protected void compute() {
        if (last - first < 10) {
            updateSalary();
        } else {
            int middle = (first + last) / 2;
            System.out.printf("Task pending tasks: %s\n", getQueuedTaskCount());
            Thread t1 = new Thread(employes, first, middle + 1, increment);
            Thread t2 = new Thread(employes, middle + 1, last, increment);
            invokeAll(t1, t2);

        }
    }

    private void updateSalary() {
        for (int i = first; i < last; i++) {
            Employe employe = employes.get(i);
            employe.setSalary((employe.getSalary()) * 2);
        }
    }
}
```

Output:

```
Employe emp1: 19999.000000
Employe emp2: 19998.000000
Employe emp3: 19997.000000
Employe emp4: 19996.000000
Employe emp5: 19995.000000
Employe emp6: 19994.000000
Employe emp7: 19993.000000
Employe emp8: 19992.000000
Employe emp9: 19991.000000
Employe emp10: 19990.000000
------------------------------------------------------------------------------------------
To Increase the salary of Employes
------------------------------------------------------------------------------------------
**************************************
Main: Pralleism:7
**************************************
Main: Pralleism:7
**************************************
Task pending tasks: 0
Main: Pralleism:7
**************************************
Main: Pralleism:7
**************************************
Main: Pralleism:7
Main: The process has completed normally.

Employe emp1: 39998.000000
Employe emp2: 39996.000000
Employe emp3: 39994.000000
Employe emp4: 39992.000000
Employe emp5: 39990.000000
Employe emp6: 39988.000000
Employe emp7: 39986.000000
Employe emp8: 39984.000000
Employe emp9: 39982.000000
Employe emp10: 39980.000000
```