EXPERIMENT NO : 3

Name: Fale Manish Dinkar
Class: TE
Div: A
Roll No: T211036
Batch: A2

Problem Statement: Write a program to solve Classical Problems of Synchronization using Mutex and Semaphore.
1.Bounded-buffer (or Producer-Consumer) Problem:
2.Dining-Philosophers Problem:
3.Reader and Writers Problem:

1. Bounded_Buffer

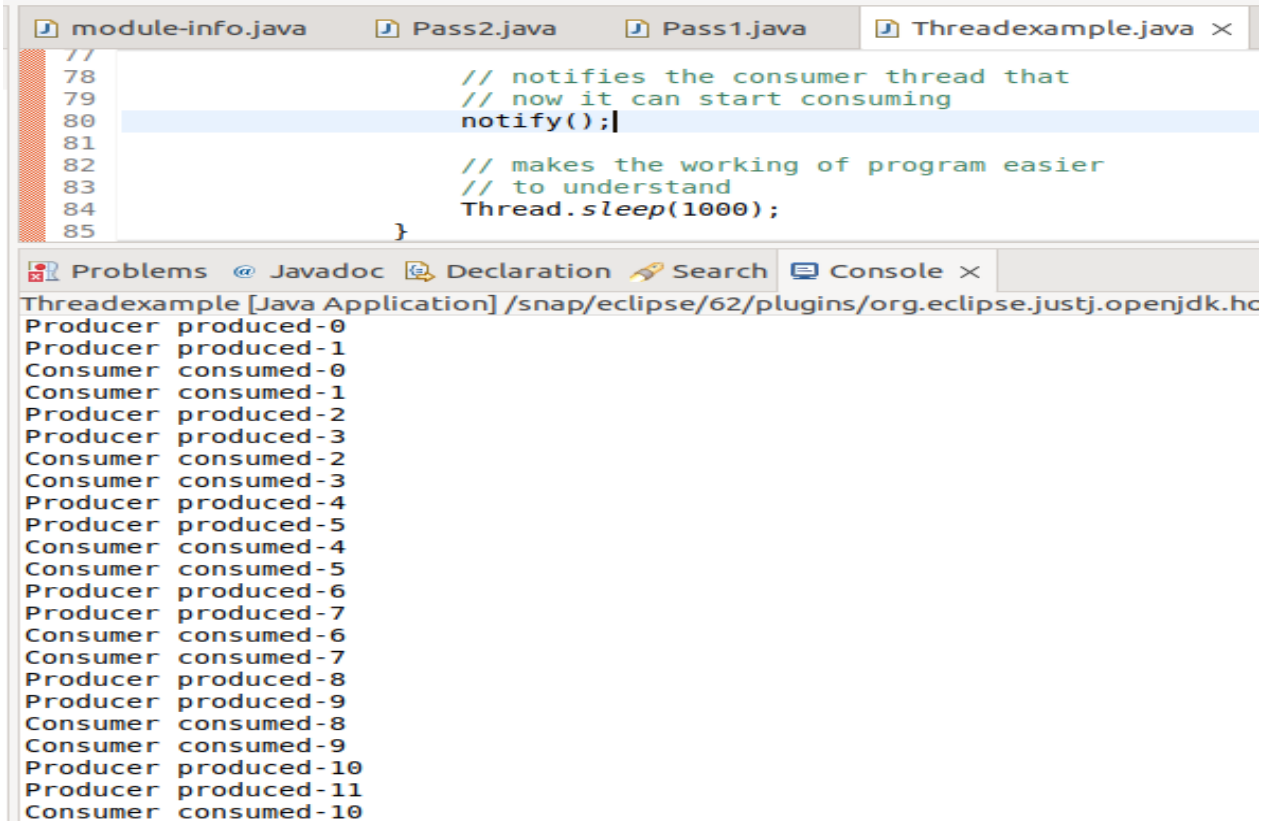
// Java program to implement solution of producer consumer problem.

```
package Threadexample;
import java.util.LinkedList;

public class Threadexample {
	public static void main(String[] args)
		throws InterruptedException
	{
		// Object of a class that has both produce()
		// and consume() methods
		final PC pc = new PC();

		// Create producer thread
		Thread t1 = new Thread(new Runnable() {
			@Override
			public void run()
			{
				try {
					pc.produce();
				}
				catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		});

		// Create consumer thread
		Thread t2 = new Thread(new Runnable() {
			@Override
			public void run()
			{
				try {
					pc.consume();
				}
				catch (InterruptedException e) {
					e.printStackTrace();
```

```
                        }
                }
        });

        // Start both threads
        t1.start();
        t2.start();

        // t1 finishes before t2
        t1.join();
        t2.join();
}

// This class has a list, producer (adds items to list
// and consumer (removes items).
public static class PC {

        // Create a list shared by producer and consumer
        // Size of list is 2.
        LinkedList<Integer> list = new LinkedList<>();
        int capacity = 2;

        // Function called by producer thread
        public void produce() throws InterruptedException
        {
                int value = 0;
                while (true) {
                        synchronized (this)
                        {
                                // producer thread waits while list
                                // is full
                                while (list.size() == capacity)
                                        wait();

                                System.out.println("Producer produced-"
                                                                + value);

                                // to insert the jobs in the list
                                list.add(value++);

                                // notifies the consumer thread that
                                // now it can start consuming
                                notify();

                                // makes the working of program easier
                                // to understand
                                Thread.sleep(1000);
                        }
                }
        }

        // Function called by consumer thread
        public void consume() throws InterruptedException
        {
                while (true) {
                        synchronized (this)
                        {
                                // consumer thread waits while list
                                // is empty
                                while (list.size() == 0)
                                        wait();
```

```
                                        // to retrieve the first job in the list
                                        int val = list.removeFirst();

                                        System.out.println("Consumer consumed-"
                                                                    + val);

                                        // Wake up producer thread
                                        notify();

                                        // and sleep
                                        Thread.sleep(1000);
                                }
                        }
                }
        }
}
```

Output:



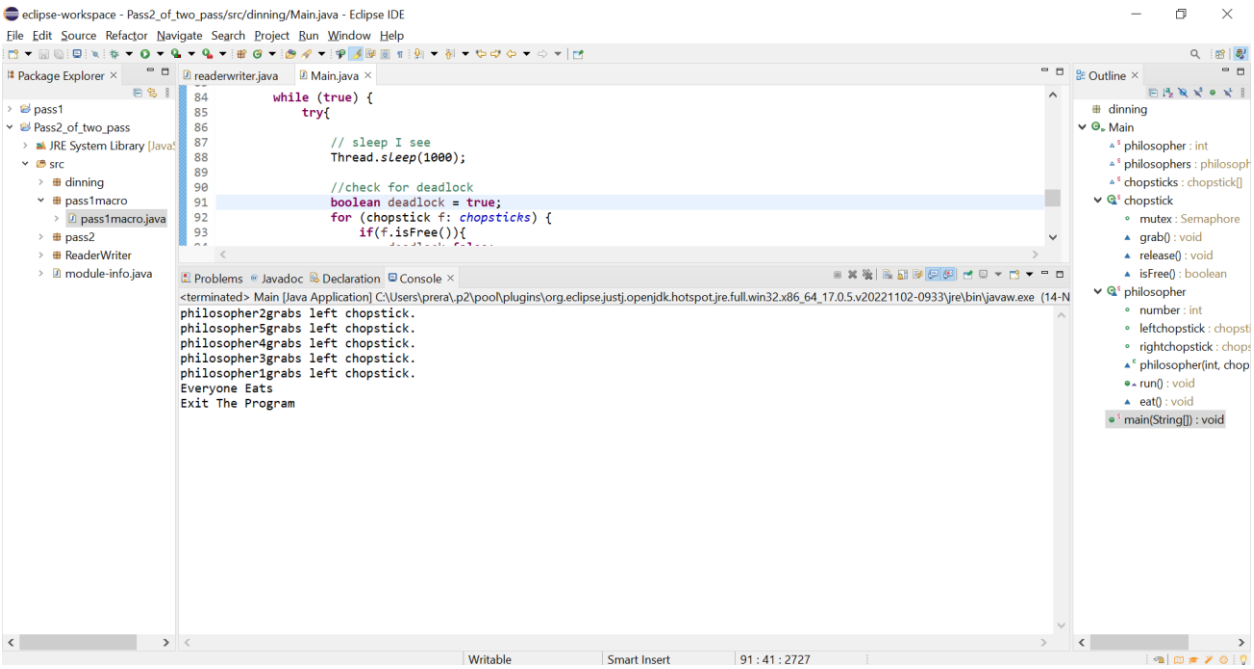
2. Dinning-Philosophers

```
    package dinning;

import java.util.concurrent.Semaphore;
import java.util.concurrent.ThreadLocalRandom;

public class Main {
    static int philosopher = 5;
    static philosopher philosophers[] = new philosopher[philosopher];
    static chopstick chopsticks[] = new chopstick[philosopher];

    static class chopstick {

        public Semaphore mutex = new Semaphore(1);

        void grab(){
            try {
                mutex.acquire();
            }
            catch (Exception e) {
                e.printStackTrace(System.out);
            }
        }


                void release() {
                    mutex.release();
                }
                boolean isFree(){
                    return mutex.availablePermits()>0;
                }
    }

    static class philosopher extends Thread {

            public int number;
        public chopstick leftchopstick;
        public chopstick rightchopstick;

        philosopher(int num, chopstick left, chopstick right) {
            number = num;
            leftchopstick = left;
            rightchopstick = right;
        }

        public void run(){

            while (true) {
                leftchopstick.grab();
```

```
            System.out.println("philosopher" + (number+1) + "grabs left chopstick.");
            rightchopstick.grab();
            System.out.println("philosopher" + (number+1) + "grabs right chopstick.");
            eat();
            leftchopstick.release();
            System.out.println("philosopher" + (number+1) + "releases left chopstick.");
            rightchopstick.release();
            System.out.println("philosopher" + (number+1)+" releases right chopstick.");
        }
    }


    void eat(){
        try {
            int sleepTime = ThreadLocalRandom.current().nextInt(0, 1000);
            System.out.println("philosopher" + (number+1)+"eats for"+ sleepTime);
            Thread.sleep(sleepTime);
            }
        catch (Exception e) {
            e.printStackTrace(System.out);
            }
        }
}

public static void main(String argv[]) {

    for (int i=0;i<philosopher;i++) {
        chopsticks[i]= new chopstick();
    }

    for (int i=0;i<philosopher;i++){
        philosophers[i] = new philosopher(i, chopsticks[i],chopsticks[(i+1)%philosopher]);
        philosophers[i].start();
    }

    while (true) {
        try{

            // sleep I see
            Thread.sleep(1000);

            //check for deadlock
            boolean deadlock = true;
            for (chopstick f: chopsticks) {
                if(f.isFree()){
                    deadlock=false;
                    break;
                }
            }
            if (deadlock) {
                Thread.sleep(1000);
                System.out.println("Everyone Eats");
```

```
                break;
            }
        }

        catch (Exception e) {
            e.printStackTrace(System.out);
        }
    }

System.out.println("Exit The Program");
    System.exit(0);
    }
}
```

OUTPUT:-



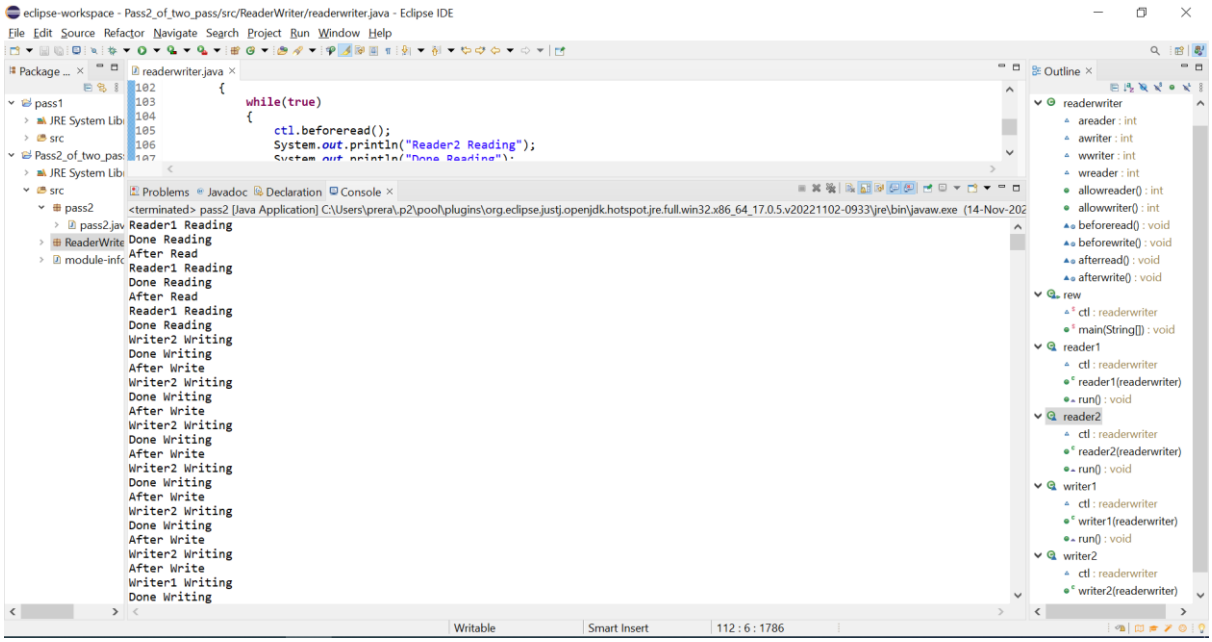
3. Reader-Writer

```
package ReaderWriter;

public class readerwriter {

	int areader=0, awriter=0, wwriter=0, wreader=0;

	public int allowreader(){
		int res=0;
		if(wreader==0&&awriter==0){
			res=1;
		}
		return res;
	}

	public int allowwriter(){
		int res=0;
		if(areader==0&&awriter==0){
			res=1;
		}
		return res;
	}

	synchronized void beforeread(){
		wreader++;
		while(allowreader()!=0){
			try
			{
				wait();
			}
			catch(InterruptedException e)
			{
			}
			wreader--;
			areader++;
		}
	}

	synchronized void beforewrite(){
		wwriter++;
		while(allowwriter()!=0){
			try
			{
				wait();
			}
			catch(InterruptedException e)
			{
			}
			wwriter--;
			awriter++;
		}
```

```
}

synchronized void afterread(){
	areader--;
	notifyAll();
}

synchronized void afterwrite(){
	awriter--;
	notifyAll();
}

}


class rew{
	static readerwriter ctl;

	public static void main(String[] args){
		ctl=new readerwriter();
		new reader1(ctl).start();
		new reader2(ctl).start();
		new writer1(ctl).start();
		new writer2(ctl).start();
	}
}

class reader1 extends Thread{
	readerwriter ctl;
	public reader1(readerwriter c){
		ctl=c;
	}
	public void run(){
		while(true)
		{
			ctl.beforeread();
			System.out.println("Reader1 Reading");
			System.out.println("Done Reading");
			ctl.afterread();
			System.out.println("After Read");
		}
	}
}

class reader2 extends Thread{
	readerwriter ctl;
	public reader2(readerwriter c)
	{
		ctl=c;
	}
	public void run()
	{
```

```
                while(true)
                {
                        ctl.beforeread();
                        System.out.println("Reader2 Reading");
                        System.out.println("Done Reading");
                        ctl.afterread();
                        System.out.println("After Read");
                }
        }
    }

    class writer1 extends Thread{
        readerwriter ctl;
        public writer1(readerwriter c){
                ctl=c;
        }
        public void run(){
                while(true)
                {
                        ctl.beforewrite();
                        System.out.println("Writer1 Writing");
                        System.out.println("Done Writing");
                        ctl.afterread();
                        System.out.println("After Write");
                }
        }
    }

    class writer2 extends Thread{
        readerwriter ctl;
        public writer2(readerwriter c){
                ctl=c;
        }
        public void run(){
                while(true)
                {
                        ctl.beforewrite();
                        System.out.println("Writer2 Writing");
                        System.out.println("Done Writing");
                        ctl.afterread();
                        System.out.println("After Write");
                }
        }

}
```

OUTPUT :-


eclipse-workspace - Pass2_of_two_pass/src/ReaderWriter/readerwriter.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
readerwriter.java
while(true)
{
ctl.beforeread();
System.out.println("Reader2 Reading");
Problems Javadoc Declaration Console
<terminated> pass2 [Java Application] C:\Users\prera\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe (14-Nov-202
Reader1 Reading
Done Reading
After Read
Reader1 Reading
Done Reading
After Read
Reader1 Reading
Done Reading
Writer2 Writing
Done Writing
After Write
Writer2 Writing
Done Writing
After Write
Writer2 Writing
Done Writing
After Write
Writer2 Writing
Done Writing
After Write
Writer2 Writing
Done Writing
After Write
Writer2 Writing
After Write
Writer1 Writing
Done Writing
Writable
Smart Insert
112 : 6 : 1786