St. Francis Institute of Technology
Borivli (West), Mumbai-400103
Department of Information Technology

Experiment – 14

1. Aim: To implement synchronization in concurrent programming using Java programming language

2. Objective: After performing the experiment, the students will be able to implement

- Create a thread and Perform simple thread operations
- Synchronize the threads

3. Lab objective mapped: To understand alternative paradigm through concurrent programming fundamentals and design, develop applications based on concurrent programming (PSO2) (PO2)

4. Prerequisite: Basics of Java programming- classes, objects, functions, data abstraction

5. Requirements: The following are the requirements – Java (JDK8) Compiler

6. Pre-Experiment Theory:

Concurrency generally refers to events or circumstances that are happening or existing at the same time. In programming terms, concurrent programming is a technique in which

- Two or more processes start
- Run in an interleaved fashion through switching and
- Complete in an overlapping time period by managing access to shared resources

Process

- Process means any program is in execution.
- Process control block contains information about processes for example Process priority, process id, process state, CPU, register, etc.
- A process can create other processes which are known as **Child Processes**.
- Process takes more time to terminate and it is isolated means it does not share memory with any other process.
- Process is called heavy-weight process

Thread

- One or more **threads** run in the context of the **process**.
- A **thread** is the basic unit to which the operating system allocates processor time.
- A **thread** can execute any part of the **process** code, including parts currently being executed by another **thread**
- **Thread is called light weight process**

Synchronization

- Multithreaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results.
- So it needs to be made sure by some synchronization method that only one thread can access the resource at a given point of time.
- Java provides a way of creating threads and synchronizing their task by using synchronized blocks.
- Synchronized blocks in Java are marked with the synchronized keyword.

- A synchronized block in Java is synchronized on some object.
- All synchronized blocks synchronized on the same object can only have one thread executing inside them at a time.
- Synchronized method is used to lock an object for any shared resource.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

7. Laboratory Exercise

A. Steps to be implemented

 1. **Compilation using JDK 8 using Turbo C**

- Write the code in notepad and save as a .java file.
- Run command prompt and set the path (Eg. Set path= 'C:\Users\m09mu\Desktop\Javacodes')
- Compile the code using the command 'javac name_of_file.java'
- Correct compile time errors (if any) and rerun the code
- After successful compilation, run the code using the command 'java name_of_file'

 Using Online IDE for C/C++/Java

- Log on to www.onlinegdb.com,/ www.jdoodle.com
- Select the programming language for coding
- Create new project
- Save the project using CTRL+S
- Run the program using F9
- For debugging use F8, along with step-into function of onlinegdb

B. Program Code

1. WAP in Java to implement thread synchronization

```
class List{
    synchronized public void display(int[] arr){ //synchronized method
        for (int i = 0; i < arr.length; i++) {
            System.out.println(arr[i]);
            try {
                Thread .sleep( 100 );
            }
            catch ( Exception e){ System .out.println(e);}
        }
    }
}
//Class myThread 1 for printing array 0,1,2,3,4,5
class myThread1 extends Thread{
    List l;
    private int[] myNum = {0,1,2,3,4,5};
    myThread1(List l){
        this.l = l;
    }
```

```
    public void  run(){
        l.display(myNum);
    }
}
//Class myThread 2 for printing array 10,11,12,13,14,15
class myThread2 extends Thread{
    List l;
    private int[] myNum = {10,11,12,13,14,15};
    myThread2(List l){
        this.l = l;
    }
    public void  run(){
        l.display(myNum);
    }
}
//Driver code
public class Exp14_1{
    public static void main(String[] args) {
        List l = new List();//only one object
        myThread1 t1 = new myThread1(l);
        myThread2 t2 = new myThread2(l);
        t1.start();
        t2.start();
    }
}
```

2. WAP in Java to implement the concept of deadlock

```
//Shared class between two threads
class Friend {
    private final String name;
    //Constructor for class friend
    public Friend(String name) {
        this.name = name;
    }
    //method to get name
    public String getName() {
        return this.name;
    }
    // first synchronized method
```

```
    public synchronized void greet(Friend myFriend) {
        System.out.format(this.name + ": " + myFriend.getName() + " says
Hello to me!\n");
        myFriend.greetBack(this);
    }
    // second synchronized method
    public synchronized void greetBack(Friend myFriend) {
        System.out.format(this.name +": " + myFriend.getName() + " says
Hii to me!\n");
    }
}
//myThread class extending Thread class
class myThread extends Thread{
    private Friend friend1;
    private Friend friend2;
    // constructor to initialize fields
    myThread(Friend friend1, Friend friend2){
        this.friend1 = friend1;
        this.friend2 = friend2;
    }
    public void run(){
        friend1.greet(friend2);
    }
}
// Driver class
public class Exp14_2 {
    public static void main(String[] args) {
        //resource 1
        final Friend vincent = new Friend("Vincent");
        //resource 2
        final Friend charles = new Friend("Charles");
        //First thread
        myThread t1 = new myThread(vincent,charles);
        //Second thread
        myThread t2 = new myThread(charles, vincent);
        t1.start();
        t2.start();
    }
}
```

8. Post Experimental Exercise

A. Questions:

1. What are various methods to create thread.

There are two ways to create a thread:

- By implementing the Runnable interface.
- By extending the Thread class.

- **Extending Thread class**
  This is another way to create a thread by a new class that extends Thread class and create an instance of that class. The extending class must override run() method which is the entry point of the new thread.

```
class myThread extends Thread{
    //Defining run method
    public void run(){
        System.out.println("Hello World");
    }
}
public class Thread1 {
    //Driver code
    public static void main(String[] args) {
        //Instantiating object of HelloThread class
        myThread t1 = new myThread();
        t1.start();
    }
}
```

We must override the run() and then use the start() method to run the thread.
Also, when you create a MyThread class object, Thread class constructor will also be invoked, as it is the super class, hence MyThread class object acts as Thread class object.
Output:
Hello World

- **Implementing the Runnable Interface**
  The easiest way to create a thread is to create a class that implements the runnable interface. After implementing a runnable interface, the class needs to implement the run() method.
  Run Method Syntax:
  public void run()
  1) It introduces a concurrent thread into your program. This thread will end when run() method terminates.
  2) You must specify the code that your thread will execute inside the run() method.
  3) run() method can call other methods, can use other classes and declare variables just like any other normal method.

```java
//class myThread Implements runnable interface
class myThread implements Runnable{
    //Defining run method
    public void run(){
        System.out.println("Hello World");
    }
}
public class Thread2 {
    //Driver code
    public static void main(String[] args) {
        //Instantiating object of myThread class
        myThread m1 = new myThread();
        Thread t1 = new Thread(m1);
        t1.start();
    }
}
```

Output:
Hello World

To invoke the run()method, start() method is used. On calling start(), a new stack is provided to the thread and run() method is invoked to introduce the new thread into the program.

2. Explain the concept of deadlock in java.

Deadlock can occur in a situation when a thread is waiting for an object lock that is acquired by another thread and the second thread is waiting for an object lock that is acquired by the first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock. A Java multithreaded program may suffer from the deadlock condition because the synchronized keyword causes the executing thread to block while waiting for the lock, or monitor, associated with the specified object. Although it is not completely possible to avoid deadlock condition, but we can follow certain measures or pointers to avoid them:

- Avoid Nested Locks – You must avoid giving locks to multiple threads, this is the main reason for a deadlock condition. It normally happens when you give locks to multiple threads.
- Avoid Unnecessary Locks – The locks should be given to the important threads. Giving locks to the unnecessary threads that cause the deadlock condition.
- Using Thread Join – A deadlock usually happens when one thread is waiting for the other to finish. In this case, we can use Thread.join with a maximum time that a thread will take.

B. Results/Observations/Program output:

Present the program input/output results if any and comment on the same.

Program 1:-

```
D:\College\PCPF\ConcurrentProgramming\Experiments\Exp14>javac Exp14_1.java

D:\College\PCPF\ConcurrentProgramming\Experiments\Exp14>java Exp14_1
0
1
2
3
4
5
10
11
12
13
14
15

D:\College\PCPF\ConcurrentProgramming\Experiments\Exp14>
```

From this output we can infer that Synchronized methods enable a simple strategy for preventing thread interference and memory consistency errors.

Program 2:-

```
D:\College\PCPF\ConcurrentProgramming\Experiments\Exp14>javac Exp14_2.java

D:\College\PCPF\ConcurrentProgramming\Experiments\Exp14>java Exp14_2
Charles: Vincent says Hello to me!
Vincent: Charles says Hello to me!
```

From this output we can understand that when the program runs, it's extremely likely that both threads will block when they attempt to invoke greetBack method. Neither block will ever end, because each thread is waiting for the other to exit the greet method.

C. Conclusion:

In this experiment we have learned how synchronization is implemented in Java multithreaded programming. To achieve this we have written Java programs that implement the concept of synchronization of threads. Also we have understood what a Deadlock is and how to avoid it.

To perform this experiment we have used Java 15 and Visual Studio Code as a text editor.

From this experiment we infer that it is not possible for two invocations of synchronized methods on the same object to interleave. When one thread is executing a synchronized method for an object, all other threads that invoke synchronized methods for the same object block (suspend execution) until the first thread is done with the object. This will prevent a deadlock condition.

D. References:

[1] Michael L Scott, “Programming Language Pragmatics”, Third edition, Elsevier publication
[2] Doug Lea, “Concurrent Programming in Java: Design Principles and Pattern