PART-VII Multithreading

No.	Aim of the Practical
32.	**AIM :** Write a program to create thread which display “Hello World” message. A. by extending Thread class B. by using Runnable interface.

PROGRAM CODE :

```
// using Thread class
class Prac_32 extends Thread {
public void run() {
System.out.println("Hello World");
System.out.print("\nName : OM BARVALIYA \nID : 23DCS009 ");

}

public static void main(String[] args) {
Prac_32 thread = new Prac_32();
thread.start();

}
}


// using Runnable interface
class Prac_32 implements Runnable {
public void run() {
System.out.println("Hello World");
System.out.print("\nName : OM BARVALIYA \nID : 23DCS009 ");

}

public static void main(String[] args) {
Thread thread = new Thread(new Prac_32());
thread.start();
}
}
```

OUTPUT:

```
PS C:\Users\Om> cd "d:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part7\" ;
Hello World

Name : OM BARVALIYA
ID : 23DCS009
PS D:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part7>
```

OUTPUT: PRACTICAL-32

CONCLUSION:

This code demonstrates two methods of creating threads in Java: by extending the Thread class and by implementing the Runnable interface. The run method prints a "Hello World" message along with the author's name and ID. The first approach is implemented and executed, while the second approach is commented out. Both methods achieve the same functionality but offer different ways to manage thread behavior and inheritance.

33. **AIM :** Write a program which takes N and number of threads as an argument. Program should distribute the task of summation of N numbers amongst number of threads and final result to be displayed on the console.

PROGRAM CODE :

```
class SumThread extends Thread {
int start;
int end;
int[] numbers;
int partialSum;

public SumThread(int start, int end, int[] numbers) {
this.start = start;
this.end = end;
this.numbers = numbers;
}

public void run() {
partialSum = 0;
for (int i = start; i < end; i++) {
partialSum += numbers[i];
}
}

public int getPartialSum() {
return partialSum;
}
```

```
}
public class Prac_33 {
public static void main(String[] args) {
int N = 4;
int n = 2;

if (N < n) {
System.out.println("N should be greater than or equal to n.");
return;
}

int[] numbers = new int[N];
for (int i = 0; i < N; i++) {
numbers[i] = i + 1;
}

SumThread[] threads = new SumThread[n];
int chunkSize = N / n;
int remainder = N % n;

int start = 0;
for (int i = 0; i < n; i++) {
int end = start + chunkSize + (i < remainder ? 1 : 0);
threads[i] = new SumThread(start, end, numbers);
threads[i].start();
start = end;
}

int totalSum = 0;
try {
for (SumThread thread : threads) {
thread.join();
totalSum += thread.getPartialSum();
}
} catch (InterruptedException e) {
e.printStackTrace();
}

System.out.println("The total sum is: " + totalSum);
System.out.print("\nName : OM BARVALIYA \nID : 23DCS009 ");
}
}
```

OUTPUT:

```
Warning: PowerShell detected that you might be using a screen reader and has
un 'Import-Module PSReadLine'.

PS C:\Users\Om> cd "d:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part7\" ; if
The total sum is: 10

Name : OM BARVALIYA
ID : 23DCS009
PS D:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part7>
```

OUTPUT: PRACTICAL-33

CONCLUSION:

This code divides an array into chunks and processes each chunk in parallel using multiple threads to calculate the sum. Each thread computes a partial sum, which is then combined to get the total sum. The join() method ensures that the main thread waits for all threads to complete before calculating the final result. The program concludes by printing the total sum along with the author's name and ID.

34. **AIM :** Write a java program that implements a multi-thread application that has three threads. First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print thevalue of cube of the number.

PROGRAM CODE :

```
class Square extends Thread {
int number;

public void setNumber(int number) {
this.number = number;
}

public void run() {
System.out.println("Square of " + number + " is " + (number * number));
}
}

class Cube extends Thread {
int number;

public void setNumber(int number) {
this.number = number;
}

public void run() {
System.out.println("Cube of " + number + " is " + (number * number * number));
```

```
}
}

public class Prac_34{
public static void main(String[] args) {
Square square = new Square();
Cube cube = new Cube();

for (int i = 1; i <= 10; i++) {
System.out.println("Checking Number: " + i);
if (i % 2 == 0) {
square.setNumber(i);
square.run();
} else {
cube.setNumber(i);
cube.run();
}
try {
Thread.sleep(1000);
} catch (InterruptedException e) {
e.printStackTrace();
}
}
System.out.print("\nName : OM BARVALIYA \nID : 23DCS009 ");
}
}
```

OUTPUT:

```
PS C:\Users\Om> cd "d:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part7\"
Checking Number: 1
Cube of 1 is 1
Checking Number: 2
Square of 2 is 4
Checking Number: 3
Cube of 3 is 27
Checking Number: 4
Square of 4 is 16
Checking Number: 5
Cube of 5 is 125
Checking Number: 6
Square of 6 is 36
Checking Number: 7
Cube of 7 is 343
Checking Number: 8
Square of 8 is 64
Checking Number: 9
Cube of 9 is 729
Checking Number: 10
Square of 10 is 100

Name : OM BARVALIYA
ID : 23DCS009
PS D:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part7>
```

OUTPUT: PRACTICAL-34

CONCLUSION:

This code alternates between calculating the square and cube of numbers from 1 to 10 using two separate classes, Square and Cube. It runs the appropriate calculation based on whether the number is even or odd, with a one-second delay between each calculation. The program demonstrates basic thread usage and control flow, although it directly calls the run() method instead of starting a new thread. It concludes by printing the author's name and ID.

35. **AIM :** Write a program to increment the value of one variable by one and display it after one second using thread using sleep() method.

PROGRAM CODE :

```
class Prac_35 {
public static void main(String[] args) {
IncrementThread incrementThread = new IncrementThread();
incrementThread.start();
}
}

class IncrementThread extends Thread {
private int value = 0;

public void run() {
try {
System.out.println("Value before increment: " + value);
Thread.sleep(1000);
value++;
System.out.println("Value after increment: " + value);
System.out.print("\nName : OM BARVALIYA \nID : 23DCS009 ");
} catch (InterruptedException e) {
System.out.println("Thread interrupted: " + e.getMessage());
}
}
}
```

OUTPUT:

```
PS C:\Users\Om> cd "d:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part7\" ;
Value before increment: 0
Value after increment: 1

Name : OM BARVALIYA
ID : 23DCS009
PS D:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part7>
```

OUTPUT: PRACTICAL-35

CONCLUSION:

This code demonstrates creating a thread by extending the Thread class to increment a variable's value after a one-second delay. The IncrementThread class overrides the run() method to perform the increment operation and print the value before and after the increment. The main method starts the thread, showcasing basic thread usage and synchronization with Thread.sleep(). The program concludes by printing the author's name and ID.

36. **AIM :** Write a program to create three threads ‘FIRST’, ‘SECOND’, ‘THIRD’. Set the priority of the ‘FIRST’ thread to 3, the ‘SECOND’ thread to 5(default) and the ‘THIRD’ thread to 7.

PROGRAM CODE :

```
class FirstThread extends Thread {
public FirstThread() {
super("FIRST");
}

public void run() {
System.out.println("Thread FirstThread with priority " + getPriority() + " is running.");
}
}
class SecondThread extends Thread {
public SecondThread() {
super("SECOND");
}
public void run() {
System.out.println("Thread SecondThread with priority " + getPriority() + " is running.");
}
}
class ThirdThread extends Thread {
public ThirdThread() {
```

```
super("THIRD");
}

public void run() {
System.out.println("Thread ThirdThread with priority " + getPriority() + " is running.");
}
}

public class Prac_36 {
public static void main(String[] args) {
System.out.print("\nName : OM BARVALIYA \nID : 23DCS009\n");
FirstThread first = new FirstThread();
SecondThread second = new SecondThread();
ThirdThread third = new ThirdThread();

first.setPriority(3);
second.setPriority(5);
third.setPriority(7);

first.start();
second.start();
third.start();

}
}
```

OUTPUT:

```
PS C:\Users\Om> cd "d:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part7\" ;

Name : OM BARVALIYA
ID : 23DCS009
Thread ThirdThread with priority 7 is running.
Thread FirstThread with priority 3 is running.
Thread SecondThread with priority 5 is running.
PS D:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part7>
```

OUTPUT: PRACTICAL-36

CONCLUSION:

This code demonstrates creating and running three threads with different priorities. Each thread prints a message indicating its priority when it runs. The Prac_36 class sets the priorities for the threads and starts them. The program concludes by printing the author's name and ID.

37. **AIM :** Write a program to solve producer-consumer problem using thread synchronization.

PROGRAM CODE :

```
class Prac_37 {
public static void main(String[] args) {
System.out.print("\nName : OM BARVALIYA \nID : 23DCS009\n");
Buffer buffer = new Buffer();
Thread producerThread = new Thread(new Producer(buffer));
Thread consumerThread = new Thread(new Consumer(buffer));

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

class Buffer {
private int data;
private boolean isEmpty = true;

public synchronized void produce(int value) throws InterruptedException {
while (!isEmpty) {
wait();
}
data = value;
isEmpty = false;
System.out.println("Produced: " + data);
notify();
}

public synchronized int consume() throws InterruptedException {
while (isEmpty) {
wait();
}
isEmpty = true;
System.out.println("Consumed: " + data);
notify();
return data;
}
}

class Producer implements Runnable {
private Buffer buffer;

public Producer(Buffer buffer) {
this.buffer = buffer;
```

```
}
public void run() {
for (int i = 0; i < 10; i++) {
try {
buffer.produce(i);
Thread.sleep(500);
} catch (InterruptedException e) {
Thread.currentThread().interrupt();
}
}
}
}
class Consumer implements Runnable {
private Buffer buffer;

public Consumer(Buffer buffer) {
this.buffer = buffer;
}

public void run() {
for (int i = 0; i < 10; i++) {
try {
buffer.consume();
Thread.sleep(1000);
} catch (InterruptedException e) {
Thread.currentThread().interrupt();
}}}}
```

OUTPUT:

```
PS C:\Users\Om> cd "d:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part7\"

Name : OM BARVALIYA
ID : 23DCS009
Produced: 0
Consumed: 0
Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Consumed: 3
Produced: 4
Consumed: 4
Produced: 5
Consumed: 5
Produced: 6
Consumed: 6
Produced: 7
Consumed: 7
Produced: 8
Consumed: 8
Produced: 9
Consumed: 9
PS D:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part7>
```

OUTPUT: PRACTICAL-37

CONCLUSION:

This code implements the producer-consumer problem using thread synchronization with a shared buffer. The Buffer class uses wait() and notify() to manage access between the Producer and Consumer threads. The Producer thread adds data to the buffer, while the Consumer thread removes data from it. The program demonstrates effective use of thread synchronization to coordinate the actions of multiple threads.