Exercise – 7

a.Write a JAVA program that creates threads by extending Thread class.First thread display "Good Morning "every 1 sec, the second thread displays "Hello "every 2 seconds and the third display "Welcome" every 3 seconds,(Repeat the same by implementing Runnable)

AIM:
A JAVA program that creates threads by extending thread class.

PROGRAM DESCRIPTION:

- A thread is a lightweight subprocess, the smallest unit of processing in an application.
- Multithreading allows concurrent execution of multiple threads, which can improve the performance of CPU-bound tasks.
- Java provides built-in support for multithreaded programming using the Thread class and the Runnable interface.
- The program defines three classes (One, Two, and Three) that extend or implement the Thread class or Runnable interface respectively, each overriding the run method to print different messages at varying time intervals.
- In the ThreadEx class's main method, instances of these three classes are created and started as threads.
- The names of these threads are set, and the details of each thread, including the main thread, are printed.

PROGRAM:

```
import java.io.*;
class One extends Thread
{
public void run()
{
for(int i=0;i<5;i++)
{
try{
Thread.sleep(1000); }
catch(InterruptedException e){
System.out.println(e); }
System.out.println("Good Morning");
}
}
}
class Two extends Thread
{
public void run()
{
```

```
for(int i=0;i<5;i++)
{
try{
Thread.sleep(2000); }
catch(InterruptedException e)
{
System.out.println(e);
}
System.out.println("Hello ");
}
}
}
class Three implements Runnable
{
public void run()
{
for(int i=0;i<5;i++)
{
try{
Thread.sleep(3000); }
catch(InterruptedException e){
System.out.println(e); }
System.out.println("Wel come");
}
}
}
class Main
{
public static void main(String[] args)
{
One t1=new One();
Two t2=new Two();
Three tt=new Three();
Thread t3=new Thread(tt);
t1.setName("One");
t2.setName("Two");
t3.setName("Three");
System.out.println(t1);
System.out.println(t2);
System.out.println(t3);
Thread t=Thread.currentThread();
System.out.println(t);
t1.start();t2.start();t3.start();
}
}
```

out put

Thread[One,5,main]
Thread[Two,5,main]
Thread[Three,5,main]
Thread[main,5,main]
Good MorningHello
Good Morning
Wel come
Good Morning
Hello
Good Morning
Good Morning
Hello Wel come
Hello
Wel come
Hello
Wel come
Wel come

b.Write a program illustrating **is Alive** and **join ()**

AIM:
A java program illustarting alive and join

PROGRAM DESCRIPTION:

- isAlive(): A method in Java's Thread class that checks if a thread is currently executing (i.e., alive).
- join(): A method used to pause the current thread's execution until the thread it's called on completes its execution.
- Using join() helps in ensuring a sequence or order in which threads complete, while isAlive() assists in monitoring a thread's state.

In this program:

- two threads (thread1 and thread2) are created and started.
- After starting both threads, the program checks if the threads are alive using the isAlive() method.
- The join() method is used to make the main thread wait for thread1 and thread2 to complete their execution.
- After the join() method, the program checks again if the threads are alive using the isAlive() method.
- Finally, a message is printed indicating the main thread has completed its execution.

PROGRAM:

```
public class ThreadExample {

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

```
        Thread thread1 = new Thread(() -> {
            for (int i = 0; i < 5; i++) {
                System.out.println("Thread 1 is running");
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        Thread thread2 = new Thread(() -> {
            for (int i = 0; i < 5; i++) {
                System.out.println("Thread 2 is running");
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        thread1.start();
        thread2.start();

        System.out.println("Thread 1 is alive: " + thread1.isAlive());
        System.out.println("Thread 2 is alive: " + thread2.isAlive());

        try {
            // The main thread waits for thread1 and thread2 to complete.
            thread1.join();
            thread2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Thread 1 is alive: " + thread1.isAlive());
        System.out.println("Thread 2 is alive: " + thread2.isAlive());

        System.out.println("Main thread is complete.");
    }
}
```

out put

Thread 1 is running

Thread 2 is running
Thread 1 is alive: true
Thread 2 is alive: true
Thread 1 is runningThread 2 is running
Thread 1 is running
Thread 2 is running
Thread 1 is running
Thread 2 is running
Thread 1 is running
Thread 2 is running
Thread 1 is alive: false
Thread 2 is alive: false
Main thread is complete.

c.Write a Program illustrating Daemon Threads.

AIM:

A java program illustrating daemon threads.

PROGRAM DESCRIPTION:

- Daemon threads are background threads in Java that provide services to user threads.
- They automatically terminate when all user threads finish their execution.
- By default, threads are user threads, but you can make them daemon using the setDaemon(true) method.
- Daemon threads can't prevent the JVM from exiting when all user threads complete their execution.
- Examples include the garbage collector thread and various internal JVM threads.

PROGRAM:

```
public class DaemonThreadExample {

    public static void main(String[] args) {

        Thread daemonThread = new Thread(() -> {
            while (true) {
                System.out.println("Daemon thread is running");
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        // Set the thread as a daemon thread
        daemonThread.setDaemon(true);
```

```
        daemonThread.start();

        try {
            Thread.sleep(2000); // Main thread sleeps for 2 seconds
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Main thread is done. Exiting the program.");
        // Daemon thread will be terminated once the main thread completes its execution.
    }
}
```

out put

Daemon thread is running
Daemon thread is running
Daemon thread is running
Daemon thread is running
Main thread is done. Exiting the program.

d.Write a JAVA program Producer Consumer Problem

AIM:
A java program on producer consumer problem

PROGRAM DESCRIPTION:

- We have a shared Queue that both the producer and consumer threads access.
- The producer thread adds integers to the queue, and the consumer thread removes them.
- When the queue is full, the producer waits. When there are items in the queue, the consumer consumes them.
- After each action, the producer or consumer notifies the other thread using the notify() method, which might wake up the other thread if it's waiting.
- Both threads synchronize on the shared queue to ensure safe concurrent access.

PROGRAM:

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

public class ProducerConsumerExample {

    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        int maxSize = 5;

        Thread producer = new Thread(() -> {
            for (int i = 0; i < 10; i++) {
                synchronized (queue) {
```

```
                while (queue.size() == maxSize) {
                    try {
                        queue.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                System.out.println("Producing value: " + i);
                queue.add(i);
                queue.notify();
            }
        }
    });

    Thread consumer = new Thread(() -> {
        for (int i = 0; i < 10; i++) {
            synchronized (queue) {
                while (queue.isEmpty()) {
                    try {
                        queue.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                System.out.println("Consuming value: " + queue.poll());
                queue.notify();
            }
        }
    });

    producer.start();
    consumer.start();
  }
}
```

out put

```
Producing value: 0
Producing value: 1
Producing value: 2
Producing value: 3
Producing value: 4
Consuming value: 0
Consuming value: 1
Consuming value: 2
```

Consuming value: 3
Consuming value: 4
Producing value: 5
Producing value: 6Producing value: 7
Consuming value: 5Consuming value: 6
Consuming value: 7
Producing value: 8
Producing value: 9
Consuming value: 8
Consuming value: 9