Write a Java program that creates three threads by extending the Thread class.

- The first thread should display "Good Morning" every 1 second.
- The second thread should display "Hello" every 2 seconds.
- The third thread should display "Welcome" every 3 seconds.

```
class GoodMorningThread extends Thread {
  public void run() {
    try {
      while (true) {
        System.out.println("Good Morning");
        Thread.sleep(1000); // 1 second
      }
    } catch (InterruptedException e) {
      System.out.println("GoodMorning Thread interrupted");
    }
  }
}

class HelloThread extends Thread {
  public void run() {
    try {
      while (true) {
        System.out.println("Hello");
        Thread.sleep(2000); // 2 seconds
      }
    } catch (InterruptedException e) {
      System.out.println("Hello Thread interrupted");
    }
    }
}

class WelcomeThread extends Thread {
    public void run() {
        try {
            while (true) {
                System.out.println("Welcome");
                Thread.sleep(3000); // 3 seconds
            }
        } catch (InterruptedException e) {
            System.out.println("Welcome Thread interrupted");
        }
    }
}

public class MultiThreadDemo {
    public static void main(String[] args) {
        GoodMorningThread t1 = new GoodMorningThread();
        HelloThread t2 = new HelloThread();
        WelcomeThread t3 = new WelcomeThread();

        // start the threads
        t1.start();
        t2.start();
        t3.start();
    }
}
```

Output:

```
Good Morning

Hello

Good Morning

Welcome

Good Morning

Hello

Good Morning

Good Morning

Welcome

Hello

Good Morning
```

2. Write a Java program that creates three threads by implements the runnable interface.

The first thread should display "Good Morning" every 1 second.

The second thread should display "Hello" every 2 seconds.

The third thread should display "Welcome" every 3 seconds.

```
class MessageTask implements Runnable {

  private String message;

  private int interval; // in milliseconds


  // constructor

  public MessageTask(String message, int interval) {

    this.message = message;
```

```
        this.interval = interval;
    }

    @Override
    public void run() {
        try {
            while (true) {
                System.out.println(message);
                Thread.sleep(interval);
            }
        } catch (InterruptedException e) {
            System.out.println(message + " thread stopped.");
        }
    }
}

public class MultiRunnableDemo {
    public static void main(String[] args) {
        // create runnable tasks
        Runnable task1 = new MessageTask("Good Morning", 1000); // every 1 sec
        Runnable task2 = new MessageTask("Hello", 2000);       // every 2 sec
        Runnable task3 = new MessageTask("Welcome", 3000);     // every 3 sec

        // wrap in threads
        Thread t1 = new Thread(task1);
        Thread t2 = new Thread(task2);
        Thread t3 = new Thread(task3);
        // start threads
        t1.start();
        t2.start();
        t3.start();
    }
}
```

Output:

```
Good Morning
Hello
Good Morning
Welcome
Good Morning
Hello
Good Morning
Good Morning
Welcome
Hello
```

3. write a java program that illustrating isAlive() and join()

```
class MyThread extends Thread {
    private String name;

    public MyThread(String name) {
        this.name = name;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(name + " : " + i);
            try {
                Thread.sleep(500); // half second delay
            } catch (InterruptedException e) {
                System.out.println(name + " interrupted.");
            }
        }
        System.out.println(name + " finished.");
    }
}

public class IsAliveJoinDemo {
    public static void main(String[] args) {
        MyThread t1 = new MyThread("Thread-1");
        MyThread t2 = new MyThread("Thread-2");

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

        // Check if threads are alive
        System.out.println("t1 is alive? " + t1.isAlive());
        System.out.println("t2 is alive? " + t2.isAlive());

        try {
            // join: wait for both threads to finish
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted.");
        }

        // After join, threads should be finished
        System.out.println("t1 is alive? " + t1.isAlive());
        System.out.println("t2 is alive? " + t2.isAlive());

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

Output:

```
t1 is alive? true
t2 is alive? true
Thread-1 : 1
Thread-2 : 1
Thread-1 : 2
Thread-2 : 2
Thread-1 : 3
Thread-2 : 3
Thread-1 : 4
Thread-2 : 4
Thread-1 : 5
Thread-2 : 5
Thread-1 finished.
Thread-2 finished.
t1 is alive? false
t2 is alive? false
```

```
Main thread finished.
```

4. write a program that illustrates daemon threads

```
public class DaemonThreadExample {

  public static void main(String[] args) {
    // Create a user thread
    Thread userThread = new Thread(() -> {
      System.out.println("User thread started.");
      try {
        for (int i = 1; i <= 5; i++) {
          System.out.println("User thread working... step " + i);
          Thread.sleep(1000); // simulate work
        }
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      System.out.println("User thread finished.");
    });

    // Create a daemon thread
    Thread daemonThread = new Thread(() -> {
      while (true) {
        System.out.println("Daemon thread running in background...");
        try {
          Thread.sleep(500); // pause between messages
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
```

```
            }
        });

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

        // Start the daemon thread
        daemonThread.start();

        // Start the user thread
        userThread.start();

        // Wait for the user thread to finish
        try {
            userThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Main thread ending. Daemon thread will stop automatically.");
    }
}
```

Output:

```
Daemon thread running in background...
User thread started.
User thread working... step 1
Daemon thread running in background...
User thread working... step 2
```

```
Daemon thread running in background...

Daemon thread running in background...

User thread working... step 3

Daemon thread running in background...

Daemon thread running in background...

User thread working... step 4

Daemon thread running in background...

Daemon thread running in background...

User thread working... step 5

Daemon thread running in background...

Daemon thread running in background...

User thread finished.

Main thread ending. Daemon thread will stop automatically.
```

5. write a java program for producer consumer problem

```
public class ProducerConsumer {

  public static void main(String[] args) {

    SharedBuffer buffer = new SharedBuffer(5); // Buffer size of 5

    Thread producer = new Thread(new Producer(buffer));

    Thread consumer = new Thread(new Consumer(buffer));


    producer.start();

    consumer.start();

  }

}


class SharedBuffer {

  private final int[] buffer;

  private int count = 0; // Number of items in buffer
private int in = 0;    // Next position to produce
private int out = 0;   // Next position to consume

public SharedBuffer(int size) {
    buffer = new int[size];
}

public synchronized void produce(int item) throws InterruptedException {
    while (count == buffer.length) {
        wait(); // Buffer is full
    }
    buffer[in] = item;
    in = (in + 1) % buffer.length;
    count++;
    System.out.println("Produced: " + item);
    notifyAll(); // Notify consumer
}

public synchronized int consume() throws InterruptedException {
    while (count == 0) {
        wait(); // Buffer is empty
    }
    int item = buffer[out];
    out = (out + 1) % buffer.length;
    count--;
    System.out.println("Consumed: " + item);
    notifyAll(); // Notify producer
    return item;
    }
}

class Producer implements Runnable {
    private final SharedBuffer buffer;
    private int item = 0;

    public Producer(SharedBuffer buffer) {
        this.buffer = buffer;
    }

    @Override
    public void run() {
        try {
            while (true) {
                buffer.produce(item++);
                Thread.sleep(500); // Simulate time taken to produce
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}

class Consumer implements Runnable {
    private final SharedBuffer buffer;

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

    @Override
    public void run() {
        try {
            while (true) {
                buffer.consume();
                Thread.sleep(800); // Simulate time taken to consume
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

Output:

```
Produced: 0
Consumed: 0
Produced: 1
Produced: 2
Consumed: 1
Produced: 3
Produced: 4
Consumed: 2
Produced: 5
Produced: 6
Consumed: 3
Produced: 7
```

```
Produced: 8

Consumed: 4

Produced: 9

Produced: 10

Consumed: 5

...
```