APP ASSIGNMENT – WEEK 7

NAME – ADYA SINGH
REG NO – RA2211003010181

Q1.

```
import java.util.Random;

class NumberGenerator implements Runnable {

    @Override

    public void run() {

        Random random = new Random();

        while (true) {

            int number = random.nextInt(100); // Generate a random integer
between 0 and 99

            System.out.println("Generated number: " + number);

            if (number % 2 == 0) {

                SquareThread squareThread = new SquareThread(number);

                Thread thread = new Thread(squareThread);

                thread.start();

            } else {

                CubeThread cubeThread = new CubeThread(number);

                Thread thread = new Thread(cubeThread);

                thread.start();

            }

            try {

                Thread.sleep(1000); // Sleep for 1 second
```

```
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
class SquareThread implements Runnable {
    private int number
    public SquareThread(int number) {
        this.number = number;
    }
    @Override
    public void run() {
        int square = number * number;
        System.out.println("Square of " + number + " is " + square);
    }
}
class CubeThread implements Runnable {
    private int number;
    public CubeThread(int number) {
        this.number = number;
    }
    @Override
    public void run() {
        int cube = number * number * number;
        System.out.println("Cube of " + number + " is " + cube);
```

```
    }
}
public class Main {
    public static void main(String[] args) {
        NumberGenerator numberGenerator = new NumberGenerator();
        Thread generatorThread = new Thread(numberGenerator);
        generatorThread.start();
    }
}
```

Output –

```
input
Generated number: 80
Square of 80 is 6400
Generated number: 32
Square of 32 is 1024
Generated number: 67
Cube of 67 is 300763
Generated number: 4
Square of 4 is 16
Generated number: 57
Cube of 57 is 185193
Generated number: 24
Square of 24 is 576
Generated number: 66
Square of 66 is 4356
Generated number: 70
Square of 70 is 4900
Generated number: 79
Cube of 79 is 493039
Generated number: 76
Square of 76 is 5776
Generated number: 9
Cube of 9 is 729
```

Q2.

```
import java.util.LinkedList;
import java.util.Queue;
class ProducerConsumer {
    private Queue<Integer> buffer = new LinkedList<>();
    private int capacity = 1; // Capacity of the buffer
    private int item = 0;
    public void produce() throws InterruptedException {
        synchronized (this) {
            while (buffer.size() == capacity) {
                wait(); // Wait if the buffer is full
            }
            System.out.println("Producer produced: " + item);
            buffer.add(item++);
            notify(); // Notify the consumer that an item is produced
        }
    }
    public void consume() throws InterruptedException {
        synchronized (this) {
            while (buffer.isEmpty()) {
                wait(); // Wait if the buffer is empty
            } }
            int consumedItem = buffer.poll();
            System.out.println("Consumer consumed: " + consumedItem);
```

```
            notify(); // Notify the producer that an item is consumed
        }    }
}
public class ProducerConsumerDemo {
    public static void main(String[] args) {
        ProducerConsumer pc = new ProducerConsumer();
        Thread producerThread = new Thread(() -> {
            try {
                while (true) {
                    pc.produce();
                    Thread.sleep(1000); // Simulate some time taken to produce an
item
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
        Thread consumerThread = new Thread(() -> {
            try {
                while (true) {
                    pc.consume();
                    Thread.sleep(1000); // Simulate some time taken to consume an
item
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
```

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

Output –

```
Producer produced: 0
Consumer consumed: 0
Producer produced: 1
Consumer consumed: 1
Producer produced: 2
Consumer consumed: 2
Producer produced: 3
Consumer consumed: 3
Producer produced: 4
Consumer consumed: 4
...
```

Q3.

```
public class ThreadNameChangeDemo {
    public static void main(String[] args) {
        Thread myThread = new Thread(() -> {
            try {
                // Sleep for 5 seconds
                Thread.sleep(5000);
                System.out.println("Thread is awake!");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
        // Start the thread
        myThread.start();
        // Change the name of the thread
        myThread.setName("MyCustomThreadName");
        // Get and print the thread's name
        String threadName = myThread.getName();
        System.out.println("Thread name is: " + threadName);
        // Wait for the thread to finish (optional)
        try {
            myThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
```

```
    }
  }
```

Output –

```
Thread name is: MyCustomThreadName
Thread is awake!
```

Q4.

```
public class ThreadNameChangeDemo {
  public static void main(String[] args) {
    Thread myThread = new Thread(() -> {
      for (int i = 5; i >= 1; i--) {
        try {
          Thread.sleep(6000); // Sleep for 6 seconds
          System.out.println("Thread is awake! Name: " +
Thread.currentThread().getName());
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
        // Change the name of the thread
        Thread.currentThread().setName("CustomThread" + i);
```

```
        }
    });
    // Start the thread
    myThread.start();
    // Wait for the thread to finish (optional)
    try {
        myThread.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
  }
}
```

Output –

```
Thread is awake! Name: CustomThread5
Thread is awake! Name: CustomThread4
Thread is awake! Name: CustomThread3
Thread is awake! Name: CustomThread2
Thread is awake! Name: CustomThread1
```

Q5.

```
public class MultiThreadDemo {
  public static void main(String[] args) {
    // Create a user-defined thread
    Thread userThread = new Thread(() -> {
```

```
            System.out.println("User Thread started.");
            try {
                Thread.sleep(1000); // Sleep for 1 second
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("User Thread finished.");
        });
        // Start the user thread
        userThread.start();

        // Main thread sleeps for 1 second
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Main Thread started.");
        System.out.println("Main Thread finished.");
    }
}
```

Output -

```
User Thread started.
Main Thread started.
User Thread finished.
Main Thread finished.
```

Q6.

```
class Printer {
    private int currentJob = 1;
    public synchronized void print(int jobNumber) throws InterruptedException {
        while (jobNumber != currentJob) {
            wait(); // Wait if it's not this job's turn
        }
        System.out.println("Printing Job " + jobNumber);
        currentJob++;
        notify(); // Notify the next waiting job
    }
}
class PrintJob implements Runnable {
    private Printer printer;
    private int jobNumber;
    public PrintJob(Printer printer, int jobNumber) {
        this.printer = printer;
        this.jobNumber = jobNumber;
    }
    @Override
    public void run() {
        try {
            while (jobNumber <= 10) {
                printer.print(jobNumber);
                jobNumber += 2; // Increment by 2 to alternate between two threads
```

```
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
public class PrinterSynchronizationDemo {
    public static void main(String[] args) {
        Printer printer = new Printer();
        Thread thread1 = new Thread(new PrintJob(printer, 1));
        Thread thread2 = new Thread(new PrintJob(printer, 2));
        thread1.start();
        thread2.start();
    }
}
```

Output –

```
Printing Job 1
Printing Job 2
Printing Job 3
Printing Job 4
Printing Job 5
Printing Job 6
Printing Job 7
Printing Job 8
Printing Job 9
Printing Job 10
```

Q7.

```
public class CharacterCountDemo {
    public static void main(String[] args) {
        String k = "Hello123World456";

        // Create ThreadA to count digits
        Thread threadA = new Thread(() -> {
            int dc = 0;
            for (char c : k.toCharArray()) {
                if (Character.isDigit(c)) {
                    dc++;
                }
            }
            System.out.println("ThreadA:" + dc);
        }};

        // Create ThreadB to count alphabetic characters
        Thread threadB = new Thread(() -> {
            int cc = 0;
            for (char c : k.toCharArray()) {
                if (Character.isLetter(c)) {
                    cc++;
                }
            }
```

```
            System.out.println("ThreadB:" + cc);
        });
        // Start both threads
        threadA.start();
        threadB.start();
    }
}
```

Output –

```
ThreadA:6
ThreadB:10
```

Q8.

```
import java.util.Scanner;
class UserThreadPriority extends Thread {
   String k;
   char c;
   public UserThreadPriority(String name) {
      super(name);
   }
   public void run() {
      System.out.println(getName() + " is running.");
      System.out.println("k: " + k);
      System.out.println("c: " + c);
    }
}
public class ThreadPriorityDemo {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      // Create two thread objects with names "ThreadA" and "ThreadB"
      UserThreadPriority threadobj1 = new UserThreadPriority("ThreadA");
      UserThreadPriority threadobj2 = new UserThreadPriority("ThreadB");

      // Get a string from the user and assign it to threadobj1.k
      System.out.print("Enter a string for ThreadA (k): ");
      threadobj1.k = scanner.nextLine();

      // Get a character from the user and assign it to threadobj1.c
```

```
        System.out.print("Enter a character for ThreadA (c): ");
        threadobj1.c = scanner.nextLine().charAt(0);

        // Get a string from the user and assign it to threadobj2.k
        System.out.print("Enter a string for ThreadB (k): ");
        threadobj2.k = scanner.nextLine();

        // Get a character from the user and assign it to threadobj2.c
        System.out.print("Enter a character for ThreadB (c): ");
        threadobj2.c = scanner.nextLine().charAt(0);
        scanner.close();
        // Start both threads
        threadobj1.start();
        threadobj2.start();
    }
}
```

Output –

```
Enter a string for ThreadA (k): Hello
Enter a character for ThreadA (c): X
Enter a string for ThreadB (k): World
Enter a character for ThreadB (c): Y
ThreadA is running.
k: Hello
c: X
ThreadB is running.
k: World
c: Y
```

Q9.

```
public class ThreadChainDemo {
    public static void main(String[] args) {
        Thread firstThread = new Thread(() -> {
            System.out.println("Thread 1 started.");
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Thread 1 finished.");
        });

        Thread secondThread = new Thread(() -> {
            System.out.println("Thread 2 started.");
            try {
                Thread.sleep(20);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Thread 2 finished.");
        });

        Thread thirdThread = new Thread(() -> {
            System.out.println("Thread 3 started.");
            try {
```

```
        Thread.sleep(50);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("Thread 3 finished.");
});

Thread fourthThread = new Thread(() -> {
    System.out.println("Thread 4 started.");
    try {
        Thread.sleep(70);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("Thread 4 finished.");
});

Thread fifthThread = new Thread(() -> {
    System.out.println("Thread 5 started.");
    try {
        Thread.sleep(100);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("Thread 5 finished.");
});
```

```
        // Start the threads in sequence
        firstThread.start();
        secondThread.start();
        thirdThread.start();
        fourthThread.start();
        fifthThread.start();
    }
}
```

Output –

```
Thread 1 started.
Thread 1 finished.
Thread 2 started.
Thread 2 finished.
Thread 3 started.
Thread 3 finished.
Thread 4 started.
Thread 4 finished.
Thread 5 started.
Thread 5 finished.
```

Q10.

```
public class ThreadPriorityDemo {
    public static void main(String[] args) {
        // Create five threads with different priorities
        Thread thread1 = new Thread(new MyRunnable(), "Thread 1");
        Thread thread2 = new Thread(new MyRunnable(), "Thread 2");
        Thread thread3 = new Thread(new MyRunnable(), "Thread 3");
        Thread thread4 = new Thread(new MyRunnable(), "Thread 4");
        Thread thread5 = new Thread(new MyRunnable(), "Thread 5");

        // Set thread priorities
        thread1.setPriority(Thread.MIN_PRIORITY); // Priority 1
        thread2.setPriority(3); // Priority 3
        thread3.setPriority(Thread.NORM_PRIORITY); // Priority 5
        thread4.setPriority(7); // Priority 7
        thread5.setPriority(Thread.MAX_PRIORITY); // Priority 10

        // Start the threads
        thread1.start();
        thread2.start();
        thread3.start();
        thread4.start();
        thread5.start();
    }
}
```

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println(Thread.currentThread().getName() + " is running with priority " + Thread.currentThread().getPriority());
    }
}
```

Output –

```
Thread 1 is running with priority 1
Thread 2 is running with priority 3
Thread 3 is running with priority 5
Thread 4 is running with priority 7
Thread 5 is running with priority 10
```