**Java Assignment No.7**

/\*

    Ques : 1.Can we call the run() method instead of start() ans : Yes you can

    Name : Sandesh Shivaji Shinde

    PRN : 23620006

 \*/

public class Ques\_1 implements *Runnable* {

    public *void* run() {

        System.out.println("Thread is running...");

    }

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

*Ques\_1* myThread = new Ques\_1();

        // Call run() directly instead of start()

        myThread.run();

    }

}

****

2. Explain the use of word Synchronized

Ans -

When a method or a block of code is marked as synchronized, only one thread can execute that code at any given time. Other threads must wait until the executing thread releases the lock on the object that the synchronized block is synchronized on.

/\*

    Ques : Write a program to display thread information.

    Name : Sandesh Shivaji Shinde

    PRN : 23620006

 \*/

public class ThreadInfo {

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

        // Get a reference to the current thread

*Thread* mainThread = Thread.currentThread();

        // Display information about the main thread

        System.out.println("Main Thread:");

        System.out.println("Thread name: " + mainThread.getName());

        System.out.println("Thread ID: " + mainThread.getId());

        System.out.println("Thread priority: " + mainThread.getPriority());

        System.out.println("Thread state: " + mainThread.getState());

        System.out.println("Thread is daemon: " + mainThread.isDaemon());

        System.out.println();

        // Display information about all active threads

        System.out.println("Currently active threads:");

*ThreadGroup* currentThreadGroup = Thread.currentThread().getThreadGroup();

*Thread*[] activeThreads = new *Thread*[currentThreadGroup.activeCount()];

        currentThreadGroup.enumerate(activeThreads);

        for (*Thread* thread : activeThreads) {

            System.out.println("Thread name: " + thread.getName());

            System.out.println("Thread ID: " + thread.getId());

            System.out.println("Thread priority: " + thread.getPriority());

            System.out.println("Thread state: " + thread.getState());

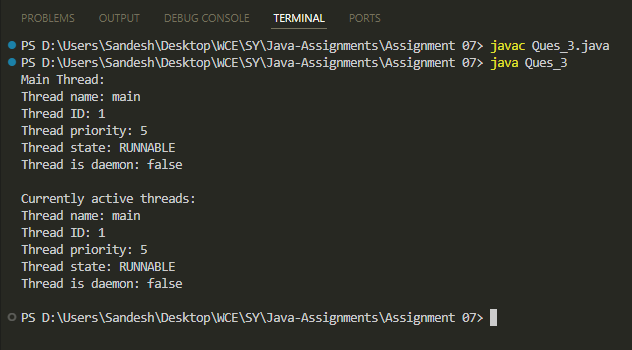
            System.out.println("Thread is daemon: " + thread.isDaemon());

            System.out.println();

        }

    }

}

****

/\*

    Ques : Create a thread using Thread class.

    Name : Sandesh Shivaji Shinde

    PRN : 23620006

 \*/

public class Ques\_4\_1 extends *Thread* {

    public *void* run() {

        System.out.println("Thread using Thread class is running.");

    }

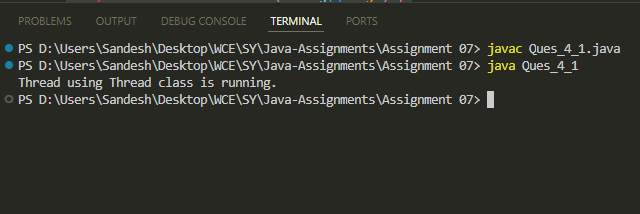
    public static *void* main(*String*[] *args*) {

*Ques\_4\_1* thread = new Ques\_4\_1();

        thread.start(); // Start the thread

    }

}

****

/\*

    Ques : Create a thread using  Runnable class.

    Name : Sandesh Shivaji Shinde

    PRN : 23620006

 \*/

public class Ques\_4\_2 implements *Runnable* {

    public *void* run() {

        System.out.println("Thread using Runnable interface is running.");

    }

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

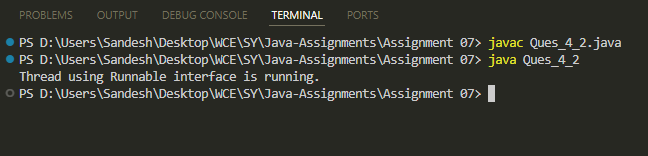
*Ques\_4\_2* runnable = new Ques\_4\_2();

*Thread* thread = new Thread(runnable); // Create a new thread with the Runnable object

        thread.start(); // Start the thread

    }

}

****

/\*

    Ques : Write a program for thread communication and synchronization.

    Name : Sandesh Shivaji Shinde

    PRN : 23620006

 \*/

public class Ques\_5 {

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

        final *SharedResource* sharedResource = new SharedResource();

        // Creating two threads

*Thread* producerThread = new Thread(new Producer(sharedResource));

*Thread* consumerThread = new Thread(new Consumer(sharedResource));

        // Start both threads

        producerThread.start();

        consumerThread.start();

    }

}

class SharedResource {

    private *int* data;

    private *boolean* produced;

    // Method for producing data

    public synchronized *void* produce(*int* *newData*) {

        // If data is already produced, wait for it to be consumed

        while (produced) {

            try {

                wait(); // Wait for the consumer to consume the data

            } catch (*InterruptedException* *e*) {

                e.printStackTrace();

            }

        }

        // Produce new data

        data = newData;

        System.out.println("Produced: " + data);

        produced = true;

        // Notify the consumer that data is available

        notify();

    }

    // Method for consuming data

    public synchronized *void* consume() {

        // If data is not produced yet, wait for it to be produced

        while (!produced) {

            try {

                wait(); // Wait for the producer to produce data

            } catch (*InterruptedException* *e*) {

                e.printStackTrace();

            }

        }

        // Consume the data

        System.out.println("Consumed: " + data);

        produced = false;

        // Notify the producer that data has been consumed

        notify();

    }

}

// Producer class

class Producer implements *Runnable* {

    private final *SharedResource* sharedResource;

    public Producer(*SharedResource* *sharedResource*) {

        this.sharedResource = sharedResource;

    }

    public *void* run() {

        for (*int* i = 0; i < 5; i++) {

            sharedResource.produce(i);

            try {

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

            } catch (*InterruptedException* *e*) {

                e.printStackTrace();

            }

        }

    }

}

// Consumer class

class Consumer implements *Runnable* {

    private final *SharedResource* sharedResource;

    public Consumer(*SharedResource* *sharedResource*) {

        this.sharedResource = sharedResource;

    }

    public *void* run() {

        for (*int* i = 0; i < 5; i++) {

            sharedResource.consume();

            try {

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

            } catch (*InterruptedException* *e*) {

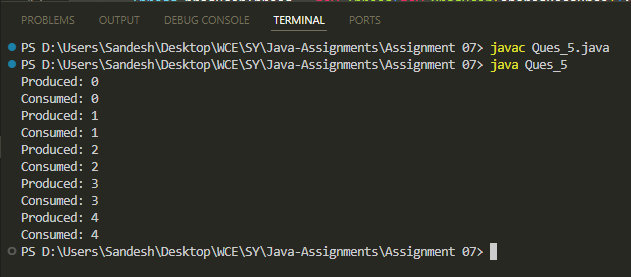
                e.printStackTrace();

            }

        }

    }

}

****