**MEHER SHRISHTI NIGAM**

**20BRS1193**

**JAVA LAB 8**

Q1) Assume that a course CSE1007 is offered in two slots D1 and D2 and each slot has only one class number. Write a simple java program using threads to track the student registration of each class and print the total number of students registered for D1 slot and D2 slot. Implement the solution by extending the thread class initially followed by implementation using runnable interface as well.

CODE:

public class Registration\_Thread\_20BRS1193 extends Thread{

    String reg\_no;

    String slot;

    static int D1\_Students = 0;

    static int D2\_Students = 0;

    public void run(String *reg\_no*, int *slot\_preference*){

        this.reg\_no = *reg\_no*;

        if(*slot\_preference* == 1)

            this.Slot\_D1();

        else if(*slot\_preference* == 2)

            this.Slot\_D2();

        this.print();

    }

    public void Slot\_D1(){

        this.slot = "D1";

        System.out.println("Slot chosen as D1");

        D1\_Students++;

    }

    public void Slot\_D2(){

        this.slot = "D2";

        System.out.println("Slot chosen as D2");

        D2\_Students++;

    }

    public void print(){

        System.out.println("Reg No:" + this.reg\_no);

        System.out.println("Slot:" + this.slot);

        System.out.println("Number of students D1 slot:" + D1\_Students);

        System.out.println("Number of students D2 slot:" + D2\_Students);

        System.out.println();

    }

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

        Registration\_Thread\_20BRS1193 stu1 = new Registration\_Thread\_20BRS1193();

        Registration\_Thread\_20BRS1193 stu2 = new Registration\_Thread\_20BRS1193();

        Registration\_Thread\_20BRS1193 stu3 = new Registration\_Thread\_20BRS1193();

        stu2.run("101",1);

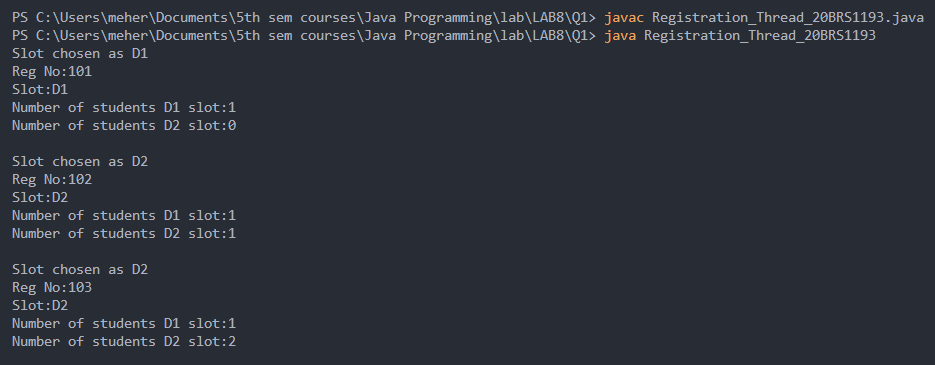
        stu1.run("102",2);

        stu3.run("103",2);

    }

}

OUTPUT:



CODE:

import java.util.Scanner;

public class Registration\_Runnable\_20BRS1193 implements Runnable {

    String reg\_no;

    String slot;

    static int D1\_Students = 0;

    static int D2\_Students = 0;

    public void run(){

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter register no: ");

        this.reg\_no = sc.nextLine();

        System.out.println("Enter slot preference: ");

        int slot\_preference = sc.nextInt();

        if(slot\_preference == 1)

            this.Slot\_D1();

        else if(slot\_preference == 2)

            this.Slot\_D2();

        this.print();

    }

    public void Slot\_D1(){

        this.slot = "D1";

        System.out.println("Slot chosen as D1");

        D1\_Students++;

    }

    public void Slot\_D2(){

        this.slot = "D2";

        System.out.println("Slot chosen as D2");

        D2\_Students++;

    }

    public void print(){

        System.out.println("Reg No:" + this.reg\_no);

        System.out.println("Slot:" + this.slot);

        System.out.println("Number of students D1 slot:" + D1\_Students);

        System.out.println("Number of students D2 slot:" + D2\_Students);

        System.out.println();

    }

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

        Registration\_Runnable\_20BRS1193 stu1 = new Registration\_Runnable\_20BRS1193();

        stu1.run();

        Registration\_Runnable\_20BRS1193 stu2 = new Registration\_Runnable\_20BRS1193();

        stu2.run();

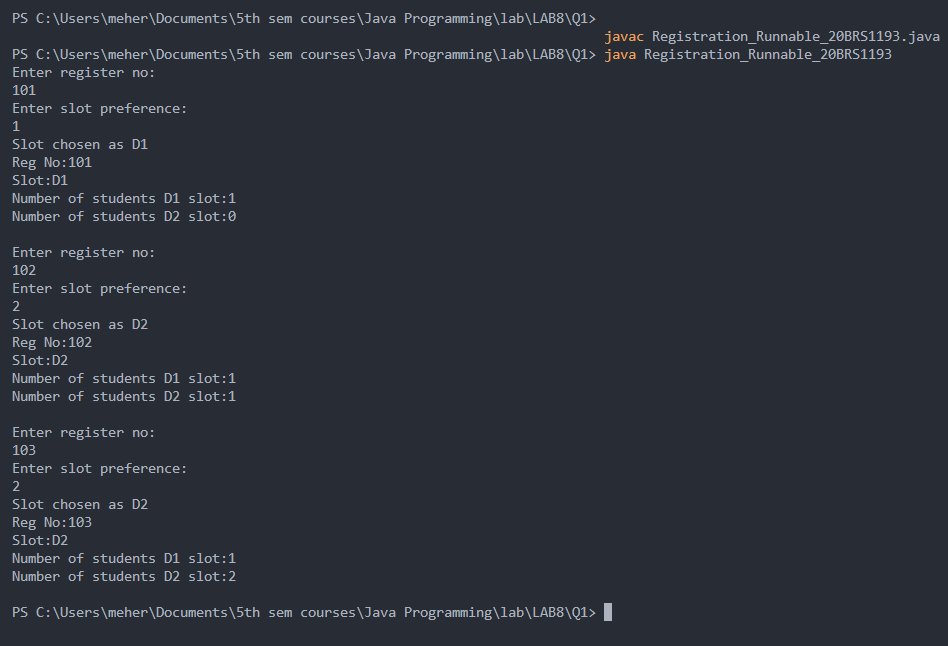
        Registration\_Runnable\_20BRS1193 stu3 = new Registration\_Runnable\_20BRS1193();

        stu3.run();

    }

}

OUTPUT:



Q2) Write a Java program to print three multiplication tables (2, 3 and 5) using threads. (i.e assign each thread for a table).

CODE:

*class* Q2\_20BRS1193 {

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

*Table\_20BRS1193* ob = new Table\_20BRS1193();

        thread1\_20BRS1193 t1 = new thread1\_20BRS1193(ob);

        thread2\_20BRS1193 t2 = new thread2\_20BRS1193(ob);

        thread3\_20BRS1193 t3 = new thread3\_20BRS1193(ob);

        t1.start();

        t2.start();

        t3.start();

    }

}

*class* Table\_20BRS1193 {

*synchronized* *void* table(*int* *n*) {

        for (*int* i = 1; i <= 10; i++) {

            System.out.println(n + " \* " + i + " = " + n \* i);

        }

    }

}

*class* thread1\_20BRS1193 *extends* *Thread* {

*Table\_20BRS1193* m;

    thread1\_20BRS1193(*Table\_20BRS1193* *m*) {

*this*.m = m;

    }

*public* *void* run() {

        m.table(2);

    }

}

*class* thread2\_20BRS1193 *extends* *Thread* {

*Table\_20BRS1193* m;

    thread2\_20BRS1193(*Table\_20BRS1193* *m*) {

*this*.m = m;

    }

*public* *void* run() {

        m.table(3);

    }

}

*class* thread3\_20BRS1193 *extends* *Thread* {

*Table\_20BRS1193* m;

    thread3\_20BRS1193(*Table\_20BRS1193* *m*) {

*this*.m = m;

    }

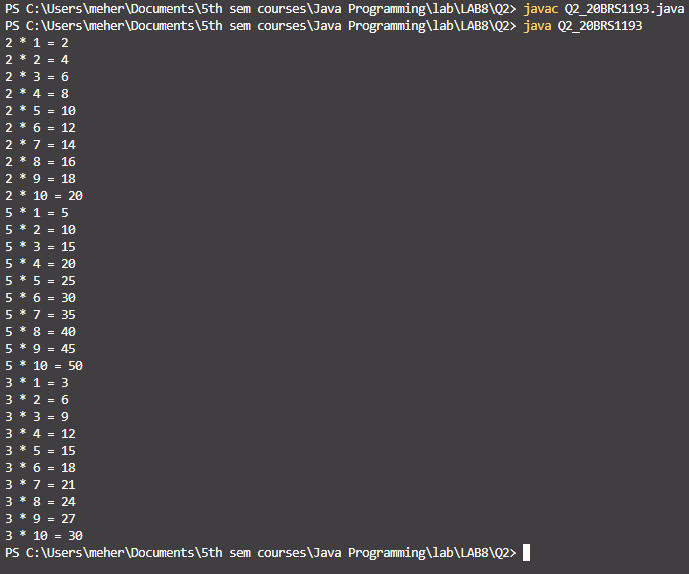
*public* *void* run() {

        m.table(5);

    }

}

OUTPUT:



Q3) Consider the case of a producer consumer problem. Demonstrate the behaviour when there is no Inter Thread Communication (ITC) and subsequently show how ITC promotes effective communication between two threads.

Below code was provided as a part of module 3 material –

NO ITC ->

CODE:

class ConsumeThread\_20BRS1193 extends Thread {

    Produce\_Consume\_20BRS1193 obj;

    ConsumeThread\_20BRS1193(Produce\_Consume\_20BRS1193 *obj*){

        this.obj = obj;

    }

    public void run() {

        for(int k = 0; k <= 5; k++)

            obj.Consume();

    }

}

class NoCommunication\_20BRS1193 {

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

        Produce\_Consume\_20BRS1193 obj = new Produce\_Consume\_20BRS1193();

        ProduceThread\_20BRS1193 P = new ProduceThread\_20BRS1193(obj);

        P.start();

        ConsumeThread\_20BRS1193 C = new ConsumeThread\_20BRS1193(obj);

        C.start();

    }

}

class Produce\_Consume\_20BRS1193{

    int i;

    synchronized void Produce(int *i*) {

        this.i = *i*; System.out.println("Data Delivered: " +*i*);

    }

    synchronized int Consume() {

        System.out.println("Data Received: " + i);

        return i;

    }

}

class ProduceThread\_20BRS1193 extends Thread{

    Produce\_Consume\_20BRS1193 obj;

    ProduceThread\_20BRS1193(Produce\_Consume\_20BRS1193 *obj*){

        this.obj = *obj*;

    }

    public void run() {

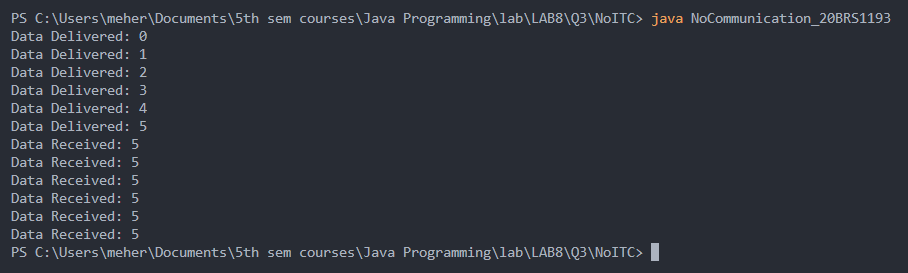
        for(int k = 0; k <= 5; k++)

        obj.Produce(k);

    }

}

OUTPUT:



We observe that all the data received are 5’s, i.e., the last delivered data.

USING ITC ->

CODE:

class Produce\_Consume\_20BRS1193{

    int i;

    boolean flag = false;

    synchronized void Produce(int *i*) {

        if(flag)

            try { *// Wait till a notification is received from Thread2.*

                wait();

            }

            catch(InterruptedException *ie*) {

                System.out.println(ie);

            }

        this.i = *i*;

        System.out.println("Data Delivered: " +*i*);

        flag = true; *// When data production is over.*

        notify(); *// Notification to Thread2, when data Produced.*

    }

    synchronized int Consume() {

    if(!flag)

        try { *// Wait till a notification is received from Thread1.*

            wait();

        }

        catch(InterruptedException *ie*){

            System.out.println(ie);

        }

        System.out.println("Data Received: " + i);

        flag = false; *// When data is received.*

        notify(); *//Notification to Thread1, when data Consumed.*

        return i;

    }

}

class ConsumeThread\_20BRS1193 extends Thread {

    Produce\_Consume\_20BRS1193 obj;

    ConsumeThread\_20BRS1193(Produce\_Consume\_20BRS1193 *obj*){ this.obj = *obj*; }

    public void run() { for(int k = 0; k <= 5; k++) obj.Consume(); }

}

class ProduceThread\_20BRS1193 extends Thread {

    Produce\_Consume\_20BRS1193 obj;

    ProduceThread\_20BRS1193(Produce\_Consume\_20BRS1193 *obj*){ this.obj = *obj*; }

    public void run() { for(int k = 0; k <= 5; k++) obj.Produce(k); }

    }

class Communication\_20BRS1193 {

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

    Produce\_Consume\_20BRS1193 obj = new Produce\_Consume\_20BRS1193();

    ProduceThread\_20BRS1193 P = new ProduceThread\_20BRS1193(obj);

    P.start();

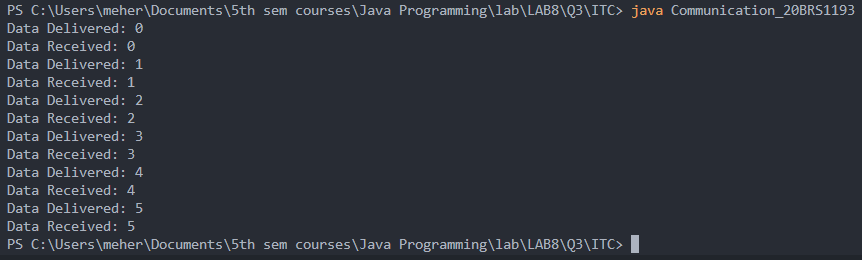
    ConsumeThread\_20BRS1193 C = new ConsumeThread\_20BRS1193(obj);

    C.start();

    }

}

OUTPUT:



Q4) As part of a technical week celebration three events namely workshop, poster presentation and hackathon are organized. There are three registration desks put up for these events. The registration cost for workshop, poster presentation and hackathon is Rs.100, Rs.200 and Rs.300 respectively. Using threads track the registration count and registration cost of each event.

CODE:

public class TechnicalWeek\_20BRS1193 {

    final static int WORKSHOP\_COST = 100;

    final static int POSTER\_COST = 200;

    final static int HACKATHON\_COST = 300;

    static int workshopCount = 0;

    static int posterCount = 0;

    static int hackathonCount = 0;

    int regNo;

    int totalCost;

    public void run(Boolean *workshop*, Boolean *poster*, Boolean *hackathon*){

        if(workshop)

            this.Workshop();

        if(poster)

            this.Poster();

        if(hackathon)

            this.Hackathon();

        this.print();

    }

    void Workshop(){

        this.totalCost += WORKSHOP\_COST;

        workshopCount++;

    }

    void Poster(){

        this.totalCost += POSTER\_COST;

        posterCount++;

    }

    void Hackathon(){

        this.totalCost += HACKATHON\_COST;

        hackathonCount++;

    }

    public void print(){

        System.out.println("Total Cost to be payed by current student:" + this.totalCost);

        System.out.println("Number of students for workshop:" + workshopCount);

        System.out.println("Number of students for poster presentation:" + posterCount);

        System.out.println("Number of students for hackathon:" + hackathonCount);

        int totalFunds = (workshopCount \* WORKSHOP\_COST) + (posterCount \* POSTER\_COST) + (hackathonCount \* HACKATHON\_COST);

        System.out.println("Total funds:" + totalFunds);

        System.out.println();

    }

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

        TechnicalWeek\_20BRS1193 stu1 = new TechnicalWeek\_20BRS1193();

        stu1.run(true, true, false);

        TechnicalWeek\_20BRS1193 stu2 = new TechnicalWeek\_20BRS1193();

        stu2.run(false, true, true);

        TechnicalWeek\_20BRS1193 stu3 = new TechnicalWeek\_20BRS1193();

        stu3.run(true, false, false);

        TechnicalWeek\_20BRS1193 stu4 = new TechnicalWeek\_20BRS1193();

        stu4.run(false, false, false);

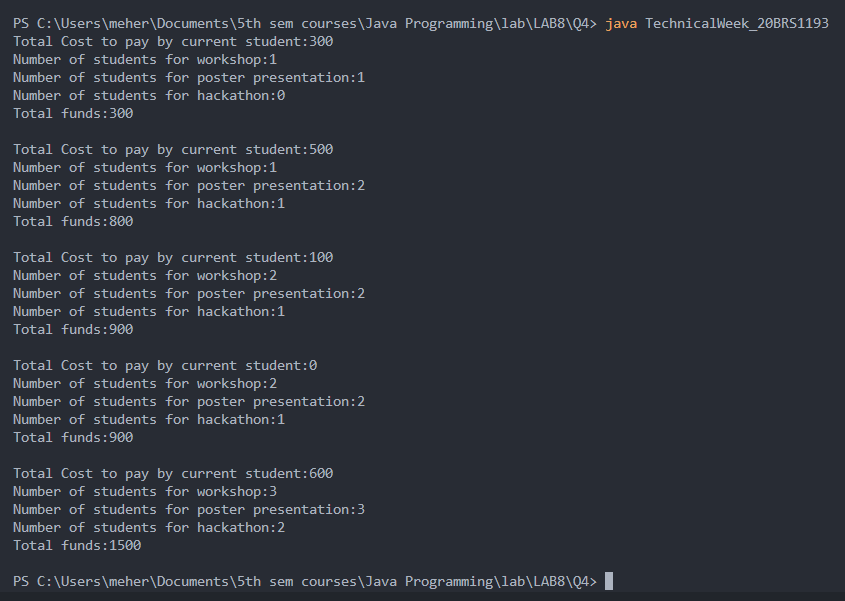
        TechnicalWeek\_20BRS1193 stu5 = new TechnicalWeek\_20BRS1193();

        stu5.run(true, true, true);

    }

}

OUTPUT:



Q5) Assume that a courier delivery office has to deliver 10 parcels to two customers. Write a java program using threads to deliver the odd parcels to customer1 and even parcels to customer2.

CODE:

public class DeliveryOffice\_20BRS1193 {

    int parcelNumber = 1;

    static int N;

    public void customer1()

    {

        synchronized (this)

        {

            while (parcelNumber < N) {

                while (parcelNumber % 2 == 0) {

                    try {

                        wait();

                    }

                    catch (

                        InterruptedException *e*) {

                        e.printStackTrace();

                    }

                }

                System.out.println("Parcel Number: " + parcelNumber + " is received by customer 1.");

                parcelNumber++;

                notify();

            }

        }

    }

    public void customer2()

    {

        synchronized (this)

        {

            while (parcelNumber < N) {

                while (parcelNumber % 2 == 1) {

                    try {

                        wait();

                    }

                    catch (

                        InterruptedException *e*) {

                        e.printStackTrace();

                    }

                }

                System.out.println("Parcel Number: " + parcelNumber + " is received by customer 2.");

                parcelNumber++;

                notify();

            }

        }

    }

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

    {

        N = 10;

        DeliveryOffice\_20BRS1193 parcels = new DeliveryOffice\_20BRS1193();

        Thread t1 = new Thread(new Runnable() {

            public void run()

            {

                parcels.customer2();

            }

        });

        Thread t2 = new Thread(new Runnable() {

            public void run()

            {

                parcels.customer1();

            }

        });

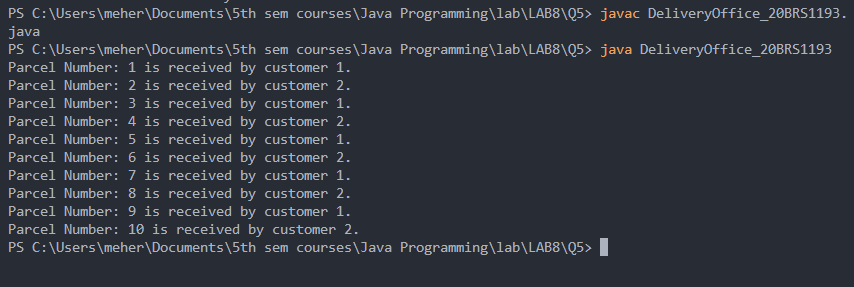
        t1.start();

        t2.start();

    }

}

OUTPUT:



Q6) Assume that a restaurant has received 20 orders. The first 5 orders have to be catered to table1, orders 6-10 for table 2, orders 11-15 for table 3 and orders 16-20 for table 4. Implement a solution for this using Java threads.

CODE:

*class* Communication\_20BRS1193 {

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

*Produce\_Consume\_20BRS1193* obj = new Produce\_Consume\_20BRS1193();

*ProduceThread\_20BRS1193* P = new ProduceThread\_20BRS1193(obj);

        P.start();

*ConsumeThread\_Table1\_20BRS1193* C1 = new ConsumeThread\_Table1\_20BRS1193(obj);

        C1.start();

*ConsumeThread\_Table2\_20BRS1193* C2 = new ConsumeThread\_Table2\_20BRS1193(obj);

        C2.start();

*ConsumeThread\_Table3\_20BRS1193* C3 = new ConsumeThread\_Table3\_20BRS1193(obj);

        C3.start();

*ConsumeThread\_Table4\_20BRS1193* C4 = new ConsumeThread\_Table4\_20BRS1193(obj);

        C4.start();

    }

}

*class* ConsumeThread\_Table1\_20BRS1193 *extends* *Thread* {

*Produce\_Consume\_20BRS1193* obj;

    ConsumeThread\_Table1\_20BRS1193(*Produce\_Consume\_20BRS1193* *obj*){ *this*.obj = obj; }

*public* *void* run()

    {

        for(*int* k = 1; k <= 5; k++)

            obj.Consume(k, 1);

    }

}

*class* ConsumeThread\_Table2\_20BRS1193 *extends* *Thread* {

*Produce\_Consume\_20BRS1193* obj;

    ConsumeThread\_Table2\_20BRS1193(*Produce\_Consume\_20BRS1193* *obj*){ *this*.obj = obj; }

*public* *void* run() { for(*int* k = 6; k <= 10; k++) obj.Consume(k, 2); }

}

*class* ConsumeThread\_Table3\_20BRS1193 *extends* *Thread* {

*Produce\_Consume\_20BRS1193* obj;

    ConsumeThread\_Table3\_20BRS1193(*Produce\_Consume\_20BRS1193* *obj*){ *this*.obj = obj; }

*public* *void* run() { for(*int* k = 11; k <= 16; k++) obj.Consume(k, 3); }

}

*class* ConsumeThread\_Table4\_20BRS1193 *extends* *Thread* {

*Produce\_Consume\_20BRS1193* obj;

    ConsumeThread\_Table4\_20BRS1193(*Produce\_Consume\_20BRS1193* *obj*){ *this*.obj = obj; }

*public* *void* run() { for(*int* k = 15; k <= 20; k++) obj.Consume(k, 4); }

}

*class* Produce\_Consume\_20BRS1193{

*int* i;

*boolean* flag = false;

*synchronized* *void* Produce(*int* *i*) {

        if(flag)

            try { *// Wait till a notification is received from Thread2.*

                wait();

            }

            catch(*InterruptedException* *ie*) {

                System.out.println(ie);

            }

*this*.i = i;

        System.out.println("Data Delivered: " + i);

        flag = true; *// When data production is over.*

        notify(); *// Notification to Thread2, when data Produced.*

    }

*synchronized* *int* Consume(*int* *x*, *int* *tableNo*) {

    if(!flag)

        try { *// Wait till a notification is received from Thread1.*

            wait();

        }

        catch(*InterruptedException* *ie*){

            System.out.println(ie);

        }

        System.out.println("Data  "+ x + " Received by table no.: " + tableNo);

        flag = false; *// When data is received.*

        notify(); *//Notification to Thread1, when data Consumed.*

        return i;

    }

}

*class* ProduceThread\_20BRS1193 *extends* *Thread* {

*Produce\_Consume\_20BRS1193* obj;

    ProduceThread\_20BRS1193(*Produce\_Consume\_20BRS1193* *obj*){ *this*.obj = obj; }

*public* *void* run() { for(*int* k = 1; k <= 20; k++) obj.Produce(k); }

    }

OUTPUT:

