

MIT WORLD PEACE UNIVERSITY

Object Oriented Programming with Java and C++
Second Year B. Tech, Semester 1

MULTITHREADING USING THREAD CLASS AND
RUNNABLE INTERFACE IN JAVA

PRACTICAL REPORT
ASSIGNMENT 7

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

November 21, 2022

Contents

1 Aim and Objectives	1
2 Problem Statements	1
2.1 Problem 1 in Java	1
2.2 Problem 2 in Java	1
3 Theory	1
3.1 Multithreading in Java	1
3.2 Life Cycle of a Thread in Java	1
3.3 Ways of Creating a Thread	2
3.4 Performing Multiple tasks by multiple threads	2
3.5 Thread Scheduler	3
3.6 Joining a Thread in Java	3
4 Platform	4
5 Input	4
6 Output	4
7 Code	5
7.1 Java Implementation of Problem 1	5
7.1.1 Java Output	7
7.2 Java Implementation of Problem 2	8
7.2.1 Java Output	9
8 Conclusion	9
9 FAQs	10

1 Aim and Objectives

Aim

Implementing Solutions on Multithreading using Thread Class and Runnable Interface

Objectives

1. To understand Multithreading in Java
2. To learn two different ways to create threads in Java

2 Problem Statements

2.1 Problem 1 in Java

Write a program to create a multithreaded calculator that does addition, subtraction, multiplication, and division using separate threads. Additionally also handle ' / by zero ' exception by the division method.

2.2 Problem 2 in Java

Print even and odd numbers in increasing order using two threads in Java

3 Theory

3.1 Multithreading in Java

In Java, Multithreading refers to a process of executing two or more threads simultaneously for maximum utilization of the CPU. A thread in Java is a lightweight process requiring fewer resources to create and share the process resources. Multithreading and Multiprocessing are used for multitasking in Java, but we prefer multithreading over multiprocessing.

3.2 Life Cycle of a Thread in Java

There are five states a thread has to go through in its life cycle. This life cycle is controlled by JVM (Java Virtual Machine). These states are:

1. New : In this state, a new thread begins its life cycle. This is also called a born thread. The thread is in the new state if you create an instance of Thread class but before the invocation of the start() method.
2. Runnable : A thread becomes runnable after a newly born thread is started. In this state, a thread would be executing its task.
3. Running : When the thread scheduler selects the thread then, that thread would be in a running state.
4. Non-Runnable (Blocked) : The thread is still alive in this state, but currently, it is not eligible to run.
5. Terminated : A thread that is in a terminated state does not consume any cycle of the CPU.

3.3 Ways of Creating a Thread

There are multiple ways of creating Threads in Java

1. By Extending the Thread Class

```
1 class Multi extends Thread{
2     public void run(){
3         System.out.println("thread is running...");
4     }
5     public static void main(String args[]){
6         Multi t1=new Multi();
7         t1.start();
8     }
9 }
10
```

2. By Implementing the Runnable Interface

```
1 class Multi3 implements Runnable{
2     public void run(){
3         System.out.println("thread is running...");
4     }
5
6     public static void main(String args[]){
7         Multi3 m1=new Multi3();
8         Thread t1 =new Thread(m1);    // Using the constructor Thread(Runnable r)
9         t1.start();
10    }
11 }
12
```

3. Using the Thread Class

```
1 public class MyThread1
2 {
3     // Main method
4     public static void main(String argvs[])
5     {
6         // creating an object of the Thread class using the constructor Thread(
        String name)
7         Thread t= new Thread("My first thread");
8
9         // the start() method moves the thread to the active state
10        t.start();
11        // getting the thread name by invoking the getName() method
12        String str = t.getName();
13        System.out.println(str);
14    }
15 }
```

3.4 Performing Multiple tasks by multiple threads

You could just create multiple classes that perform their own tasks, each a child of the Thread class. So you could then invoke instances of those classes, and run them as multiple threads.

```
1 class MultithreadEx3 extends Thread
2 {
```

```
3     public void run()
4     {
5         System.out.println("Start task one");
6     }
7 }
8 class MultithreadEx4 extends Thread
9 {
10    public void run()
11    {
12        System.out.println("Start task two");
13    }
14 }
15 class Run
16 {
17     public static void main(String args[])
18     {
19         MultithreadEx3 th1 = new MultithreadEx3();
20         MultithreadEx4 th2 = new MultithreadEx4();
21         th1.start();
22         th2.start();
23     }
24 }
```

3.5 Thread Scheduler

A component of Java that decides which thread to run or execute and which thread to wait is called a thread scheduler in Java. In Java, a thread is only chosen by a thread scheduler if it is in the runnable state.

However, if there is more than one thread in the runnable state, it is up to the thread scheduler to pick one of the threads and ignore the other ones. There are some criteria that decide which thread will execute first. There are two factors for scheduling a thread i.e. Priority and Time of arrival.

- *Priority*: Priority of each thread lies between 1 to 10. If a thread has a higher priority, it means that thread has got a better chance of getting picked up by the thread scheduler.
- *Time of Arrival*: Suppose two threads of the same priority enter the runnable state, then priority cannot be the factor to pick a thread from these two threads. In such a case, arrival time of thread is considered by the thread scheduler. A thread that arrived first gets the preference over the other threads.

3.6 Joining a Thread in Java

java.lang.Thread class provides the join() method which allows one thread to wait until another thread completes its execution.

If t is a Thread object whose thread is currently executing, then t.join() will make sure that t is terminated before the next instruction is executed by the program. If there are multiple threads calling the join() methods that means overloading on join allows the programmer to specify a waiting period.

However, as with sleep, join is dependent on the OS for timing, so you should not assume that join will wait exactly as long as you specify. There are three overloaded join functions.

```
1 public class JoinExample1 extends Thread
2 {
3     public void run()
4     {
5         for(int i=1; i<=4; i++)
6         {
7             try
8             {
9                 Thread.sleep(500);
10            }catch(Exception e){System.out.println(e);}
11            System.out.println(i);
12        }
13    }
14    public static void main(String args[])
15    {
16        JoinExample1 thread1 = new JoinExample1();
17        JoinExample1 thread2 = new JoinExample1();
18        JoinExample1 thread3 = new JoinExample1();
19        thread1.start();
20        try
21        {
22            thread1.join();
23        }catch(Exception e){System.out.println(e);}
24        thread2.start();
25        thread3.start();
26    }
27 }
```

4 Platform

Operating System: Arch Linux x86-64

IDEs or Text Editors Used: Visual Studio Code

Compilers : g++ and gcc on linux for C++, and javac, with JDK 18.0.2 for Java

5 Input

For Problem 1

1. 2 numbers
2. Choice about what to do with those numbers

For Problem 2

1. The Maximum limit up to which the user wants to see the odd and even numbers printed

6 Output

For Problem 1

1. Menu about what to do with numbers
2. Output of the calculation done with those numbers

For Problem 2

1. Even numebers and Odd numbers in Ascending order upto the specified limit.

7 Code

7.1 Java Implementation of Problem 1

```
1 // Krishnaraj Thadesar
2 // Batch A1, PA20
3 // OOPJC Assignment 7.1
4 // Write a program to create a multithreaded calculator that does addition,
5 // subtraction,
6 // multiplication, and division using separate threads.
7 // Additionally also handle '/ by zero' exception by the division method.
8
9 import java.lang.Thread;
10 import java.util.Scanner;
11
12 class Calculator extends Thread implements Runnable {
13     public int a, b, what_to_do = 0;
14
15     Calculator(int a, int b, int choice, String name) {
16         this.a = a;
17         this.b = b;
18         this.what_to_do = choice;
19         this.setName(name);
20     }
21
22     @Override
23     public synchronized void start() {
24         System.out.println("Starting the Thread");
25         System.out.println("The Name of this Thread is: " + getName());
26         super.start();
27     }
28
29     @Override
30     public void run() {
31         switch (what_to_do) {
32             case 1:
33                 System.out.println(a + b);
34                 break;
35             case 2:
36                 System.out.println(a - b);
37                 break;
38             case 3:
39                 System.out.println(a * b);
40                 break;
41             case 4:
42                 try {
43                     System.out.println(a / b);
44                 } catch (ArithmeticException e) {
45                     System.out.println("You cant Divide by Zero!");
46                 }
47                 break;
48             default:
```

```
49         break;
50     }
51 }
52 }
53
54 public class assignment_7_problem_1 {
55     public static Calculator add, sub, mul, div;
56     public static Scanner input = new Scanner(System.in);
57
58     public static void main(String[] args) {
59         int choice = 0;
60         int a, b;
61         System.out.println("Welcome To Thread Calculator of Assignment 7");
62         while (choice != 5) {
63             System.out.println("What would you like to do? ");
64             System.out.println(
65                 "1. Addition of 2 Numbers\n2. Subtraction of 2 Numbers\n3.
Multiplication of 2 Numbers\n4. Division of 2 Numbers\n\n");
66             choice = input.nextInt();
67             if (choice == 5) {
68                 break;
69             }
70             System.out.println("Enter the 2 Numbers");
71             a = input.nextInt();
72             b = input.nextInt();
73             switch (choice) {
74                 case 1:
75                     System.out.println("You have chosen Addition!");
76                     add = new Calculator(a, b, choice, "Adder");
77                     try {
78                         add.start();
79                         add.join();
80                     } catch (Exception e) {
81                         System.out.println("Got some problem with making the
thread!");
82                         System.out.println(e);
83                     }
84                     break;
85                 case 2:
86                     System.out.println("You have chosen Subtraction!");
87                     sub = new Calculator(a, b, choice, "Subtractor");
88                     try {
89                         sub.start();
90                         sub.join();
91                     } catch (Exception e) {
92                         System.out.println("Got some problem with making the
thread!");
93                         System.out.println(e);
94                     }
95                     break;
96                 case 3:
97                     System.out.println("You have chosen Multiplication!");
98                     mul = new Calculator(a, b, choice, "Multiplier");
99                     try {
100                         mul.start();
101                         mul.join();
102                     } catch (Exception e) {
103                         System.out.println("Got some problem with making the
thread!");
```



```
104         System.out.println(e);
105     }
106     break;
107     case 4:
108         System.out.println("You have chosen Division!");
109         div = new Calculator(a, b, choice, "Divider");
110         try {
111             div.start();
112             div.join();
113         } catch (Exception e) {
114             System.out.println("Got some problem with making the
thread!");
115         }
116         System.out.println(e);
117     }
118     break;
119     case 5:
120         System.out.println("You have chosed to Exit!");
121     default:
122         break;
123     }
124 }
125 System.exit(0);
126 }
127 }
```

Listing 1: Probelm 1.java

7.1.1 Java Output

```
1 Welcome To Thread Calculator of Assignment 7
2 What would you like to do?
3 1. Addition of 2 Numbers
4 2. Subtraction of 2 Numbers
5 3. Multiplication of 2 Numbers
6 4. Division of 2 Numbers
7
8
9 1
10 Enter the 2 Numbers
11 2
12 2
13 You have chosen Addition!
14 Starting the Thread
15 The Name of this Thread is: Adder
16 4
17 What would you like to do?
18 1. Addition of 2 Numbers
19 2. Subtraction of 2 Numbers
20 3. Multiplication of 2 Numbers
21 4. Division of 2 Numbers
22
23
24 4
25 Enter the 2 Numbers
26 5
27 0
28 You have chosen Division!
29 Starting the Thread
```

```
30 The Name of this Thread is: Divider
31 You cant Divide by Zero!
32 What would you like to do?
33 1. Addition of 2 Numbers
34 2. Subtraction of 2 Numbers
35 3. Multiplication of 2 Numbers
36 4. Division of 2 Numbers
37
38
39 5
```

Listing 2: Output for Problem 1 - calculations

7.2 Java Implementation of Problem 2

```
1 // Krishnaraj Thadesar
2 // Batch A1, PA20
3 // OOPJC Assignment 7.2
4 // Print even and odd numbers in increasing order using two threads in Java
5
6 import java.security.ProtectionDomain;
7 import java.util.Scanner;
8
9 import javax.swing.InputMap;
10
11 class printEven extends Thread implements Runnable {
12     int limit;
13
14     printEven(int limit) {
15         this.limit = limit;
16     }
17
18     @Override
19     public synchronized void start() {
20         super.start();
21         System.out.println("Printing Even Numbers");
22     }
23
24     @Override
25     public void run() {
26         for (int i = 0; i < limit; i++) {
27             if (i % 2 == 0) {
28                 System.out.println(i);
29             }
30         }
31     }
32 }
33
34 class printOdd extends Thread implements Runnable {
35     int limit;
36
37     printOdd(int limit) {
38         this.limit = limit;
39     }
40
41     @Override
42     public synchronized void start() {
43         super.start();
44         System.out.println("Printing Odd Numbers");
```

```
45     }
46
47     @Override
48     public void run() {
49         for (int i = 0; i < limit; i++) {
50             if (i % 2 != 0) {
51                 System.out.println(i);
52             }
53         }
54     }
55 }
56
57 public class assignment_7_problem_2 {
58     static printEven pe;
59     static printOdd po;
60     static Scanner input = new Scanner(System.in);
61
62     public static void main(String[] args) {
63         int limit = 0;
64         System.out.println("Enter To what limit Even or Odd numbers you want to
65         See");
66         limit = input.nextInt();
67         pe = new printEven(limit);
68         po = new printOdd(limit);
69         try {
70             pe.start();
71             pe.join();
72             po.start();
73             po.join();
74         } catch (Exception e) {
75             System.out.println(e);
76         }
77 }
```

Listing 3: Multithreading Even Odd

7.2.1 Java Output

```
1 Enter To what limit Even or Odd numbers you want to See
2 10
3 Printing Even Numbers
4 0
5 2
6 4
7 6
8 8
9 Printing Odd Numbers
10 1
11 3
12 5
13 7
14 9
```

Listing 4: Output for ProblemHillStation 2

8 Conclusion

Thus, learnt the use of thread class in java and performed multithreading operations.

9 FAQs

1. *Why do we use collection framework?*
2. *Which is best collection framework in Java?*
3. *What is difference between array and collection?*
4. *What is HashMap in Java?*