# **Multithreading in java**

1. Write a Java program that starts two threads and each thread will print the numbers from 1 - 4. While performing this task each thread will be at sleep for two seconds

Sample output:

1

1

2

2

3

3

4

4

2. Write a Java program that starts three threads and each will take 1 second to print out each number from 1 - 5. However, the second and third thread must wait until the first thread finishes. Use sleep and join.

Sample output:

Thread[Thread-0,5,main]

1

2

3

4

5

Thread[Thread-1,5,main]

Thread[Thread-2,5,main]

1

1

2

2

3

3

4

4

5

5

3. Repeat the work done in question 2, however, this time give name to each thread.

Sample output:

Thread[My First Thread,5,main]

1

2

3

4

5

Thread[My Third Thread,5,main]

Thread[My Second Thread,5,main]

1

1

2

2

3

3

4

4

5

5

4. Write a program that assigns priority to three threads (MIN\_PRIORITY, NORM\_PRIORITY, MAX\_PRIORITY) and starts all thread. Each will take 2 seconds to print out the thread name, it’s priority and a line separator. This task will happen 3 times.

Sample Output:

running thread name is:Thread-2

running thread priority is:5

running thread name is:Thread-0

running thread name is:Thread-1

running thread priority is:10

=======================================

running thread priority is:1

=======================================

=======================================

running thread name is:Thread-1

running thread priority is:10

=======================================

running thread name is:Thread-0

running thread name is:Thread-2

running thread priority is:5

=======================================

running thread priority is:1

=======================================

running thread name is:Thread-1

running thread priority is:10

=======================================

running thread name is:Thread-2

running thread priority is:5

=======================================

running thread name is:Thread-0

running thread priority is:1

=======================================

5. Create a class called Account that represents a bank account. This account starts with a balance of $50 and can be used only for withdrawals. The withdrawal will be accepted even if there isn't enough money in the account to cover it. The account simply reduces the balance by the amount you want to withdraw.

Imagine a couple, Ranjeet and Reema, who both have access to the account and want to make withdrawals. But they don't want the account to ever be overdrawn. Create a class AccountTesting that will start two threads and both thread trying to withdraw money from same account object in the loop. Withdrawal is two steps process :

1. Check the balance.

2. If there's enough in the account (withdraw 10), make the withdrawal. Wait 100 before withdraw

Sample Output:

Reema is going to withdraw

Reema completes the withdrawal

Ranjeet is going to withdraw

Ranjeet completes the withdrawal

Ranjeet is going to withdraw

Ranjeet completes the withdrawal

Reema is going to withdraw

Reema completes the withdrawal

Reema is going to withdraw

Reema completes the withdrawal

Not enough in account for Reema to withdraw 0

Not enough in account for Reema to withdraw 0

Not enough in account for Ranjeet to withdraw 0

Not enough in account for Ranjeet to withdraw 0

Not enough in account for Ranjeet to withdraw 0

6. Create a bean class called Message on which threads will work and call wait, notify and notifyAll methods. This class contains a private string property fields called message. Provide a constructor that takes one parameter to initialize the member field message. Also provide getter and setters to the private message field.

Create a Waiter class that implements the Runnable interface. This class has three parts:

1. A private field of type Message cass called msg
2. A constructor that takes a Message as a parameter to initialize the private member field msg
3. Overrides the run method. Inside the run method, first store the name of the current thread name in a variable name. Then pass the member msg variable to a synchronized block Ex: synchronized(msg) { //...code here}. Inside the block, print out the following: “[name] waiting to get notified at time: [current time in milliseconds]”. Then invoke the wait method of the msg member variable. Finally print the following messages: “[name] waiter thread got notified at time: [current time in milliseconds]”, “[name] processed: [message from the Message class]”

Create a Notifier class that will process on Message object and invoke notify method to wake up threads waiting for Message object.

This class has three parts:

1. A private field of type Message cass called msg
2. A constructor that takes a Message as a parameter to initialize the private member field msg
3. Overrides the run method. Inside the run method, first store the name of the current thread name in a variable name and print out: “[name] started”. Get the Thread to sleep for 1000 milliseconds. Then pass the msg to a synchronized block. Inside the block, use the setter to set the message to “[name Notifier word done” and invoke the notify or notifyAll() of the msg variable.

Create a WatNotifyTest class that will create multiple threads of Waiter and Notifier and start them. In a main method follow the following instructions:

1. Create a Message object and initialize it to “process it”.
2. Create a Waiter object waiter1 and pass it the message object
3. Create a thread that is initialize to waiter1 and is named waiter1 and start the thread
4. Create a second Waiter object waiter2 and pass it the same message object
5. Create a thread that is initialize to waiter1 and is named waiter1 and start the thread
6. Create a Notifirer object notifier and pass it the message object
7. Create a thread that is initialize to notifier and is named notifier and start the thread
8. Print out “All the threads are started”

Sample out 1:

waiter waiting to get notified at time:1520282886138

waiter1 waiting to get notified at time:1520282886139

All the threads are started

notifier started

waiter waiter thread got notified at time:1520282887139

waiter processed: notifier Notifier work done

Sample out 2:

All the threads are started

waiter waiting to get notified at time:1520283209517

waiter1 waiting to get notified at time:1520283209518

notifier started

waiter1 waiter thread got notified at time:1520283210519

waiter1 processed: notifier Notifier work done

waiter waiter thread got notified at time:1520283210519

waiter processed: notifier Notifier work done