

Institution of Technology

School of Computing

Department of Information Technology and computer science

Course Name: Introduction to Distributed System

Instructor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ [Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_](mailto:yimer@wku.edu.et)

Lab 4: Java Threads

Objectives: This Lab is a demonstration on how to implement Threads. We will be using the Java Programming

Language and it’s threading classes for implementing threads and different concepts related with threads.

Aim: Thread programming in Java. Implement the following programs.

1. Write a program that executes two threads. Create the threads by implementing Runnable interface.

• One thread display “Hello!” every 1000 milliseconds 10 times, and

• Another thread display “Hey” every 2000 milliseconds 10 times.

2. Implement Multithreaded Echo Client-Server program using Socket.

3. Implement producer consumer example.

Requirements:

We are going to use Java Threading and Networking classes and packages. So, we need a computer that has JDK and some decent java IDE.

Background:

A thread is the flow of execution, from beginning to end, of a task in a program. A task is a program unit that is executed independently of other parts of the program. A thread provides the mechanism for running a task. With Java, you can launch multiple threads from a program concurrently. These threads can be executed simultaneously in multiprocessor systems. Multithreading can make your program more responsive and interactive, as well as enhance performance.

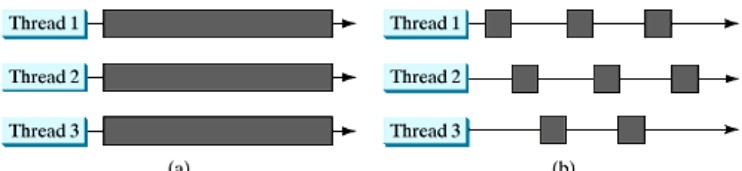


Figure 4.1: (a) Multiple threads are running on multiple CPUs. (b) Multiple threads share a single CPU.

When your program executes as an application, the Java interpreter starts a thread for the main method. When your program executes as an applet, the Web browser starts a thread to run the applet. You can create additional threads to run concurrent tasks in the program. In Java, each task is an instance of the Runnable interface, also called a runnable object. A thread is essentially an object that facilitates the execution of a task.

4-1

Defining and Starting a Thread

Tasks are objects. To create tasks, you have to first declare a class for tasks. An application that creates an instance of

Thread must provide the code that will run in that thread. There are two ways to do this:

By implementing the runnable interface:

One method of making a task to be a thread is to make the task class implement the Runnable interface. The Runnable interface is rather simple. The Runnable interface defines a single method, run, meant to contain the code executed in the thread. You need to implement this method to tell the system how your thread is going to run. The Runnable object is passed to the Thread constructor. A template for developing a task class is shown below.

public class HelloThread implements Runnable {

public void run() { System.out.println(" Hello!");

}

public static void main(String args[]) { Thread t = new Thread(new HelloThread()); t.start();

}

}

Once you have declared a TaskClass, you can create a task using its constructor. For example,

TaskClass task = new TaskClass(...);

A task must be executed in a thread. The Thread class contains the constructors for creating threads and many useful methods for controlling threads. To create a thread for a task, use

Thread thread = new Thread(task);

You can then invoke the start() method to tell the JVM that the thread is ready to run, as follows:

thread.start();

The JVM will execute the task by invoking the task’s run() method.

By extending the Thread class

The second method to create a new thread is to extend Thread Superclass and create an instance of that class. The newly created (extended) class must override the run() method, which is the entry point for the new thread. It must also call start( ) to begin execution of the new thread. Same as Runnable Interface. The Thread class itself implements Runnable, though its run method does nothing. An application can subclass Thread, providing its own implementation of run, as in the next example:

public class HeyThread extends Thread {

public void run() { System.out.println(" Hey!");

}

public static void main(String args[]) { HeyThread t = new HeyThread(); t.start();

}

}

Notice that both methods invoke Thread.start() in order to start the new thread.

The Thread Class

The Thread class contains the constructors for creating threads and the methods for controlling threads. Since the Thread class implements Runnable, you could declare a class that extends Thread and implements the run method. The following diagram shows Constructors and methods defined in Thread class.

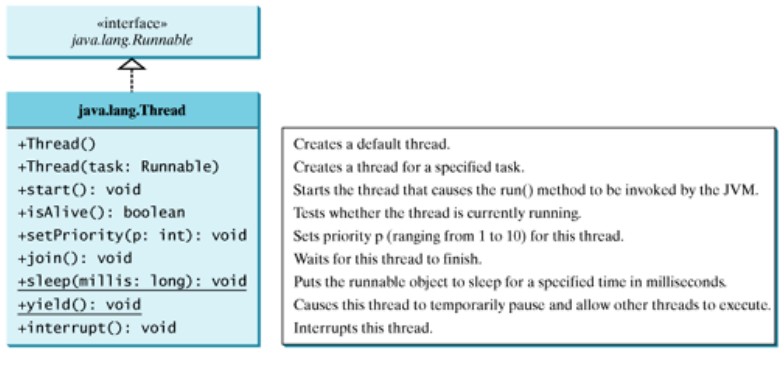


Figure 4.2: Constructors and Methods of Thread Class.

Yield: You can use the yield() method to temporarily release time for other threads. Consider the run method of

HelloThread:

public void run() {

for (int i = 1; i <= 10; i++) { System.out.print(" Hello!"); Thread.yield();

}

}

Every time a Hello is printed, the Hello thread is yielded. So each “Hello” is followed by some “Hey” if both threads are running concurrently.

Sleep: The sleep(long mills) method puts the thread to sleep for the specified time in milliseconds to allow other threads to execute. There is a possibility that the sleep period can be terminated by interrupts, therefore the function calling the sleep function should be declared to throw InterruptedException or the sleep function should be enclosed inside a try-catch block. Consider the run method of HelloThread.

public void run() {

try {

for (int i = 1; i <= 10; i++) { System.out.print(" Hello! "); if (i >= 5) ; Thread.sleep(1000);

}

} catch (InterruptedException ex) {

}

}

When i is greater than 5 (i>= 5), the HelloThread’s thread is put to sleep for 1 second.

Join: You can use the join() method to force one thread to wait for another thread to finish.

public void run() {

HeyThread ht = new HeyThread();

ht.start();

try {

for (int i = 1; i <= 10; i++) { System.out.print(" Hello! "); if (i == 5) ;

ht.join();

}

} catch (InterruptedException ex) {

}

}

A new thread ht is created. It prints “Hey” 10 times. The “Hello” from 5 to 10 are printed after thread ht is finished.

Thread Priority: Java assigns every thread a priority. By default, a thread inherits the priority of the thread that spawned it. You can increase or decrease the priority of any thread by using the setPriority() method, and You can get the thread’s priority by using the getPriority() method. Priorities are numbers ranging from 1 to 10. The Thread class has the int constants MIN-PRIORITY, NORM-PRIORITY, and MAX-PRIORITY, representing 1, 5, and 10, respectively. The priority of the main thread is Thread.NORM-PRIORITY.

The JVM always picks the currently runnable thread with the highest priority. If several runnable threads have equally high priorities, the CPU is allocated to all of them in round-robin fashion. A lower-priority thread can run only when no higher-priority threads are running.

ht.setPriority(Thread.MAX-PRIORITY);

Practical 1: Write a program that executes two threads. Create the threads by implementing Runnable interface.

a One thread display ”Hello!” every 1 second 10 times, and b Another thread display ”Hey” every 2 seconds 10 times.

Step 1: Implement the HelloThread class

public class HelloThread implements Runnable {

public void run() {

try {

for (int i = 1; i <= 10; i++) { System.out.print(" Hello! "); Thread.sleep(1000);

}

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

}

Step 2: Implement the HeyThread class

public class HeyThread extends Thread {

public void run() {

try {

for (int i = 1; i <= 10; i++) { System.out.print(" Hey! "); Thread.sleep(2000);

}

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

}

Step 3: Implementing a Test class which creates and runs both threads.

public class HeyThread extends Thread {

public void run() {

try {

for (int i = 1; i <= 10; i++) { System.out.print(" Hey! "); Thread.sleep(2000);

}

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

}

Practical 2: Implement Multithreaded Echo Client-Server program using Socket.

In order to make the Echo Client-Server program, we just need to modify the server class to handle each client request as a separate thread. The main server simply listens a connection, accept it and pass it to the server thread to handle the communication.

Step 1: Modifying the main server.

import java.net.\*; import java.io.\*; class ChatServer {

public static void main(String args[]) {

try {

ServerSocket server = new ServerSocket(8000); System.out.println("Waiting for client to connect.."); while (true) {

new ServerThread(server.accept()).start();

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

Step 2: Creating the threaded server that will handle the communication between the client and the server.

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

/\*\*

\*

\* [@author Fuad](mailto:@author)

\*/

class ServerThread extends Thread {

private Socket socket = null;

public ServerThread(Socket socket) {

this.socket = socket;

}

public void run() {

try {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream())); String receive, send;

do {

receive = in.readLine(); System.out.println("Client Says: " + receive); if (receive.equals("STOP")) {

break;

}

System.out.print("Server Says : "); send = br.readLine(); out.println(send);

} while (true); br.close(); in.close(); out.close(); socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

Step 3: Create chat client

import java.net.\*;

import java.io.\*;

class ChatClient {

public static void main(String args[]) {

try {

Socket connection = new Socket("Localhost", 8000);

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

PrintWriter out = new PrintWriter(connection.getOutputStream(), true); BufferedReader in = new BufferedReader(new

InputStreamReader(connection.getInputStream())); String msg;

System.out.println("To stop chatting with server type STOP");

do {

System.out.print("Client Says: "); msg = br.readLine(); out.println(msg);

if (msg.equals("STOP")) {

break;

}

String response = in.readLine(); System.out.println("Server Says : " + response);

} while (true); br.close(); in.close(); out.close(); connection.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}