

CS4102: Network Programming

Mohamed R. Ghetas

E-mail: Mohghattas@gmail.com

Last update: 1-Dec.-2017

Threads

- A thread is a program's path of execution. Most programs written today run as a single thread, causing problems when multiple events or actions need to occur at the same time.
- For example, a program is not capable of drawing pictures while reading keystrokes. The program must give its full attention to the keyboard input **lacking the ability to handle more than one event** at a time.
- The ideal solution to this problem is the seamless execution of two or more sections of a program at the same time, which can be done through threads.
- TCP connection needs threads to handle more than one client at the same time.
- **Multithreading in java** is a process of executing multiple threads simultaneously.

Using Threads in Java

- Java has two common ways of creating threads:
 - Create a class that extends a class, i.e. Thread class.
 - Implement an interface, i.e. Runnable.
- Extending a class is the way Java inherits methods and variables from a parent class, i.e. limited to one parent class.
- Since extending Thread class is easier, it will be considered in the following.

Extending the Thread Class

- In this case, the **run method** specifies the actions that a thread is to execute like the main method does for the full program. Similarity to the main method, run method may not be called directly, but by calling the **start method** which then automatically calls run.
- Class Thread has seven constructors, the most common two are: **Thread()** and **Thread(String<name>)**. The second of these provides a name for the thread via its argument. If the first is used, the system generates a name of the form Thread-n, where n is an integer starting at zero and increasing in value for further threads.
- As an example, the following code,

```
1 Thread firstThread = new Thread();  
2 Thread secondThread = new Thread("namedThread");  
3 System.out.println(firstThread.getName());  
4 System.out.println(secondThread.getName());
```

will display,

```
1 Thread-0  
2 namedThread
```

- As an example for the multitasking by using threads, the following code creates two threads, but we shall have one thread display the message Hello five times and the other thread output integers 1 – 5. For the first thread, we shall create a class called HelloThread ; for the second, we shall create class CountThread . Note that it is not the main application class (ThreadHelloCount) here that extends class Thread this time, but each of the two subordinate classes, HelloThread and CountThread . Each has its own version of the run method. An example of the output is shown on the next slide.

```
1 public class ThreadHelloCount
2 {
3     public static void main(String[] args)
4     {
5         HelloThread hello = new HelloThread();
6         CountThread count = new CountThread();
7         hello.start();
8         count.start();
9     }
10 }
11 class HelloThread extends Thread
12 {
13     public void run()
14     {
15         int pause;
16         for (int i=0; i<5; i++)
17         {
18             try
```

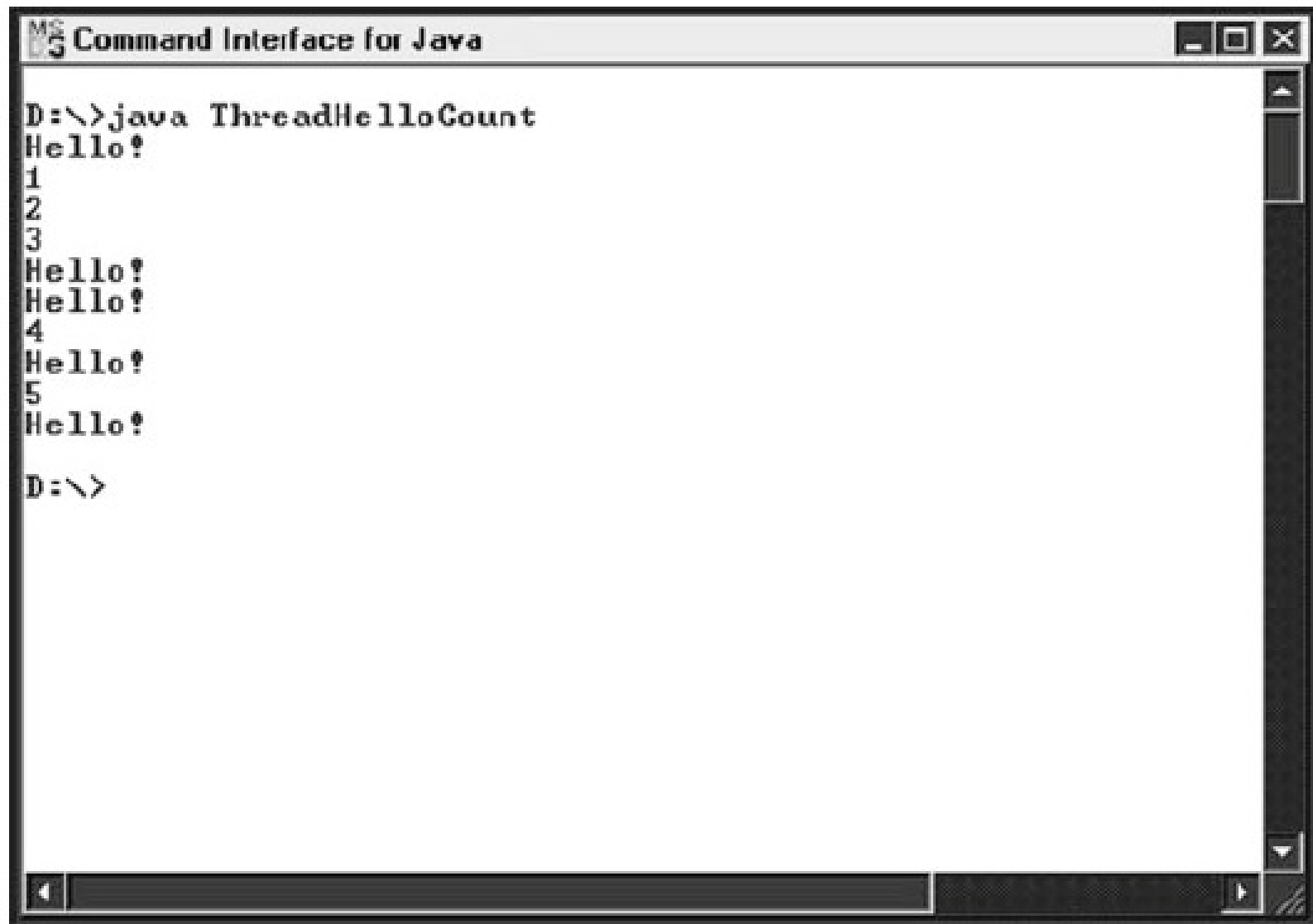
```
19 {
20 System.out.println("Hello!");
21 //Again, introduce an element
22 //of randomness
23 pause = (int)(Math.random()*3000);
24 sleep(pause);
25 }
26 catch (InterruptedException interruptEx)
27 {
28 System.out.println(interruptEx);
29 }
30 }
31 }
32 }
33 class CountThread extends Thread
34 {
35 int pause;
36 public void run()
37 {
38 for (int i=0; i<5; i++)
39 {
40 try
41 {
42 System.out.println(i);
43 pause=(int)(Math.random()*3000);
44 sleep (pause);
45 }
46 catch (InterruptedException interruptEx)
47 {
48 System.out.println(interruptEx);
```

```
49 }
```

```
50 }
```

```
51 }
```

```
52 }
```



The screenshot shows a window titled "MS Command Interface for Java". The command prompt displays the execution of the command `java ThreadHelloCount`. The output shows a sequence of numbers and "Hello?" messages, indicating that multiple threads are running concurrently. The output is as follows:

```
D:\>java ThreadHelloCount
Hello?
1
2
3
Hello?
Hello?
4
Hello?
5
Hello?
D:\>
```


Multithreaded TCP Servers

- The basic technique involves a two-stage process:
 - The main thread (the one running automatically in method `main`) allocates individual threads to incoming clients.
 - The thread allocated to each individual client then handles all subsequent interaction between that client and the server (via the threads `run` method).
- The code on the following slide is an example of the same previous Echo Server studied code by using multithreading to support multiple clients. Note that no changes are required to the client side.

Multithreaded TCP Server Example

```
1 import java.io.*;
2 import java.net.*;
3 import java.util.*;
4 public class MultiEchoServer
5 {
6     private static ServerSocket serverSocket;
7     private static final int PORT = 1234;
8     public static void main(String[] args) throws IOException
9     {
10     try
11     {
12         serverSocket = new ServerSocket(PORT);
13     }
14     catch (IOException ioEx)
15     {
16         System.out.println("\nUnable to set up port!");
17         System.exit(1);
18     }
19     do
20     {
21         //Wait for client
22         Socket client = serverSocket.accept();
23         System.out.println("\nNew client accepted.\n");
24         //Create a thread to handle communication with
25         //this client and pass the constructor for this
26         //thread a reference to the relevant socket
27         ClientHandler handler = new ClientHandler(client);
28         handler.start(); //As usual, method calls run .
29     } while (true);
```

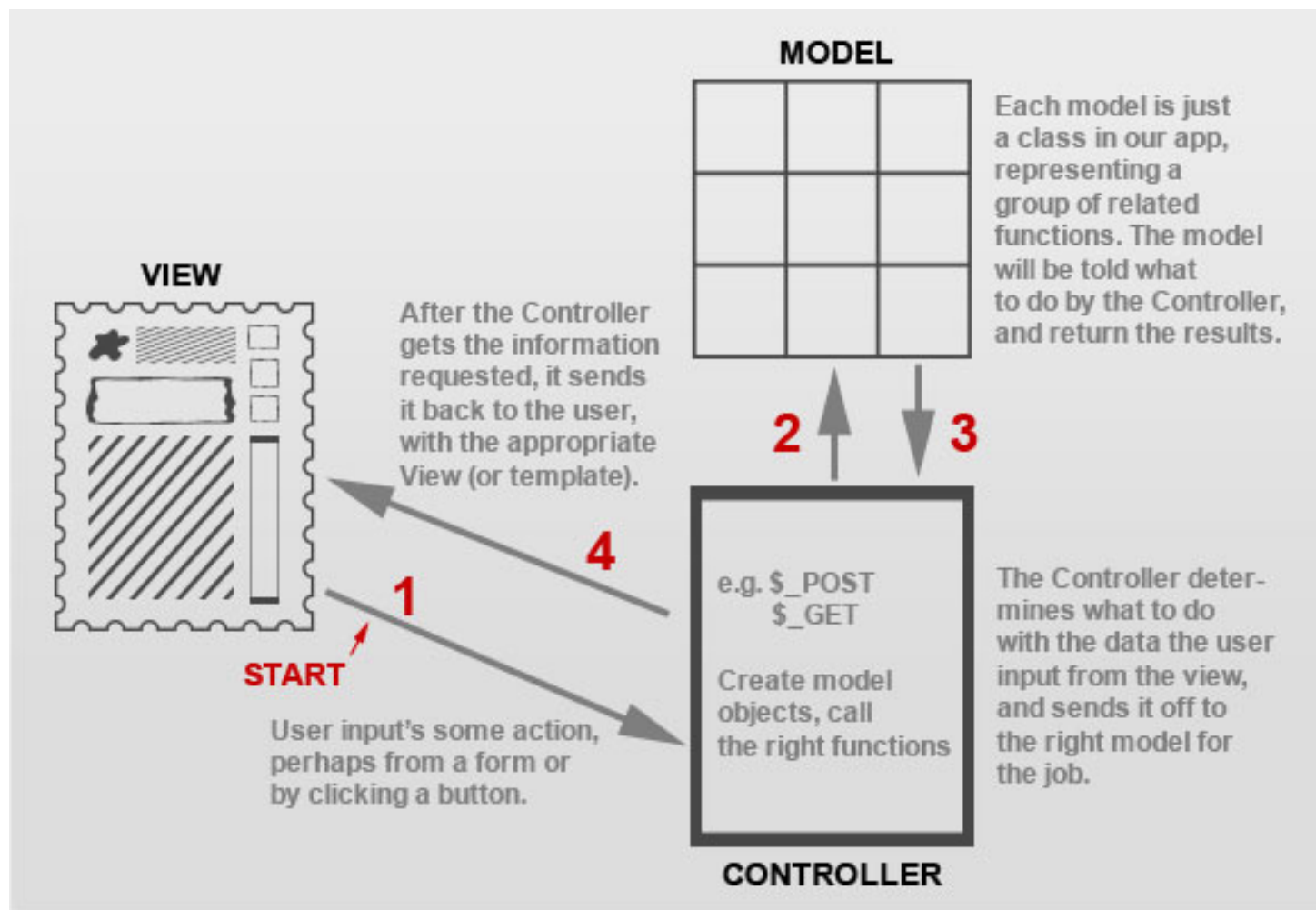
```
30 }
31 }
32 class ClientHandler extends Thread
33 {
34     private Socket client;
35     private Scanner input;
36     private PrintWriter output;
37     public ClientHandler(Socket socket)
38     {
39         //Set up reference to associated socket
40         client = socket;
41         try
42         {
43             input = new Scanner(client.getInputStream());
44             output = new PrintWriter(
45                 client.getOutputStream(), true);
46         }
47         catch(IOException ioEx)
48         {
49             ioEx.printStackTrace();
50         }
51     }
52     public void run()
53     {
54         String received;
55         do
56         {
57             //Accept message from client on
58             //the socket's input stream
59             received = input.nextLine();
```

```
60 //Echo message back to client on
61 //the socket's output stream
62 output.println("ECHO: " + received);
63 //Repeat above until 'QUIT' sent by client
64 }while (!received.equals("QUIT"));
65 try
66 {
67     if (client!=null)
68     {
69         System.out.println("Closing down connection ");
70         client.close();
71     }
72 }
73 catch(IOException ioEx)
74 {
75     System.out.println("Unable to disconnect!");
76 }
77 }
78 }
```

Modelviewcontroller (MVC) Models

- Model-view-controller (MVC) is a software architectural pattern for implementing user interfaces on computers. It divides a given application into three interconnected parts:
 - **Model:** is the central component of the pattern. It expresses the application's behavior in terms of the problem domain, independent of the user interface. It directly manages the data, logic and rules of the application.
 - **View:** can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
 - **Controller:** can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., scrolling through a document, movement of document)

- The interaction definitions in MVC model is slightly loose, so there are several possible implementation MVC models. However, the shown one in the figure is commonly used but sometimes model may decide the appropriate view and communicate with the viewer directly.



Servlets, JSP and Javabeans

- A Java servlet is a Java program that extends the capabilities of a web servers to generate dynamic Web content. It generates HTML in Java. Typically it is used as a controller in MVC. For further details: Check the Java packet javax.servlet.
- Java Server Page (JSP) is developed to enhance the view limitations in **Java Servlets**. It is a server side scripting language to embed Java codes in HTML. It has the capability of defining customized tags, write js pages. Typically it is used as the view part of MVC model.
- Servlets and JSP are commonly used together and they are the Java equivalent to other dynamic web content technologies like PHP and ASP.NET.
- JavaBeans are classes that encapsulate many objects into a single object called the bean. A bean can receive events from other objects and can generate events that are sent to those other objects and it can store/restore data. Thus, this type of classes if typically used as the model part of MVC model in Java.