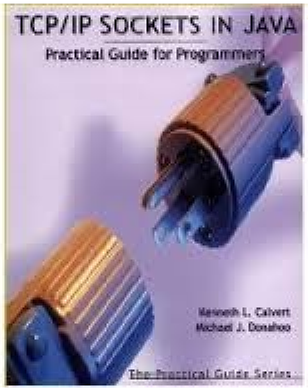
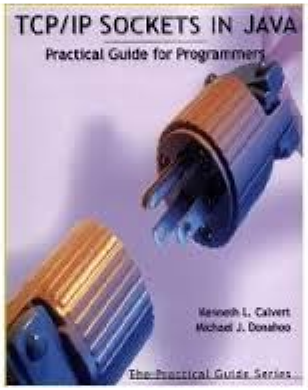
Socket Programming #2
Java Thread Basics
for Socket Programming



Most slides are from web site of the textbook "Computer Networking: A Top Down Approach," written by Jim Kurose and Keith Ross and "**TCP/IP Sockets in Java: Practical Guide for Programmers**", written by Kenneth L. Calvert and Michael J. Donahoo for Socket Programming

Java Thread Basics



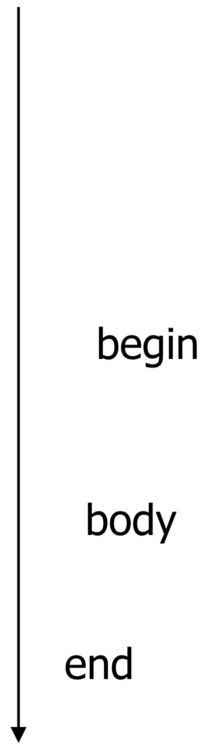
Most slides are from *Prof. Rajkumar Buyya*
Cloud Computing and Distributed Systems (CLOUDS) Laboratory
Dept. of Computer Science and Software Engineering
University of Melbourne, Australia http://www.cloudbus.org/~raj or http://www.buyya.com

Agenda

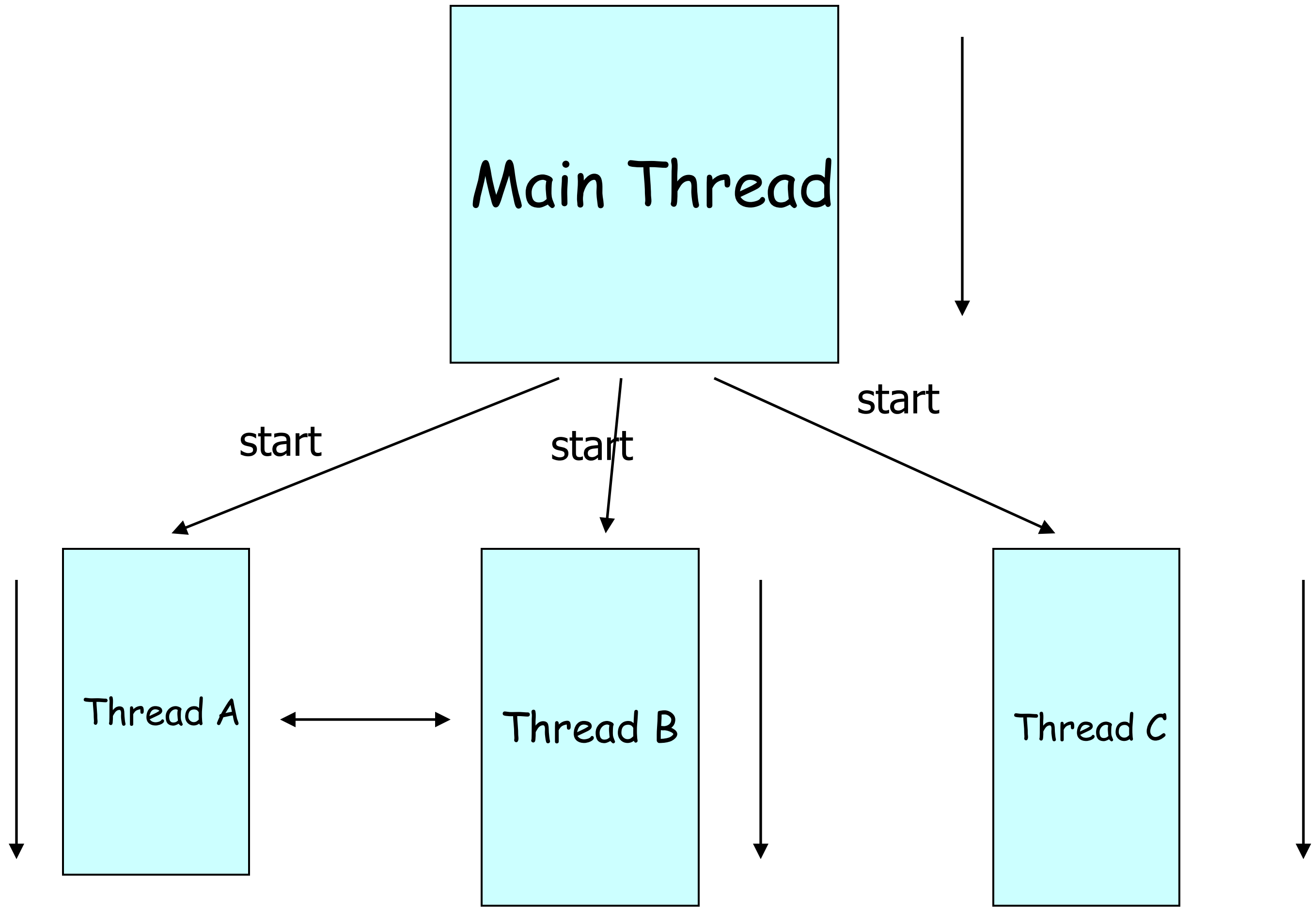
- Introduction
- Thread Applications
- Defining Threads
- Java Threads and States
  - Priorities
- Accessing Shared Resources
  - Synchronization

A single threaded program

```
class ABC
{
….
     public void main(..)
     {
     …
     ..
     }
}
```



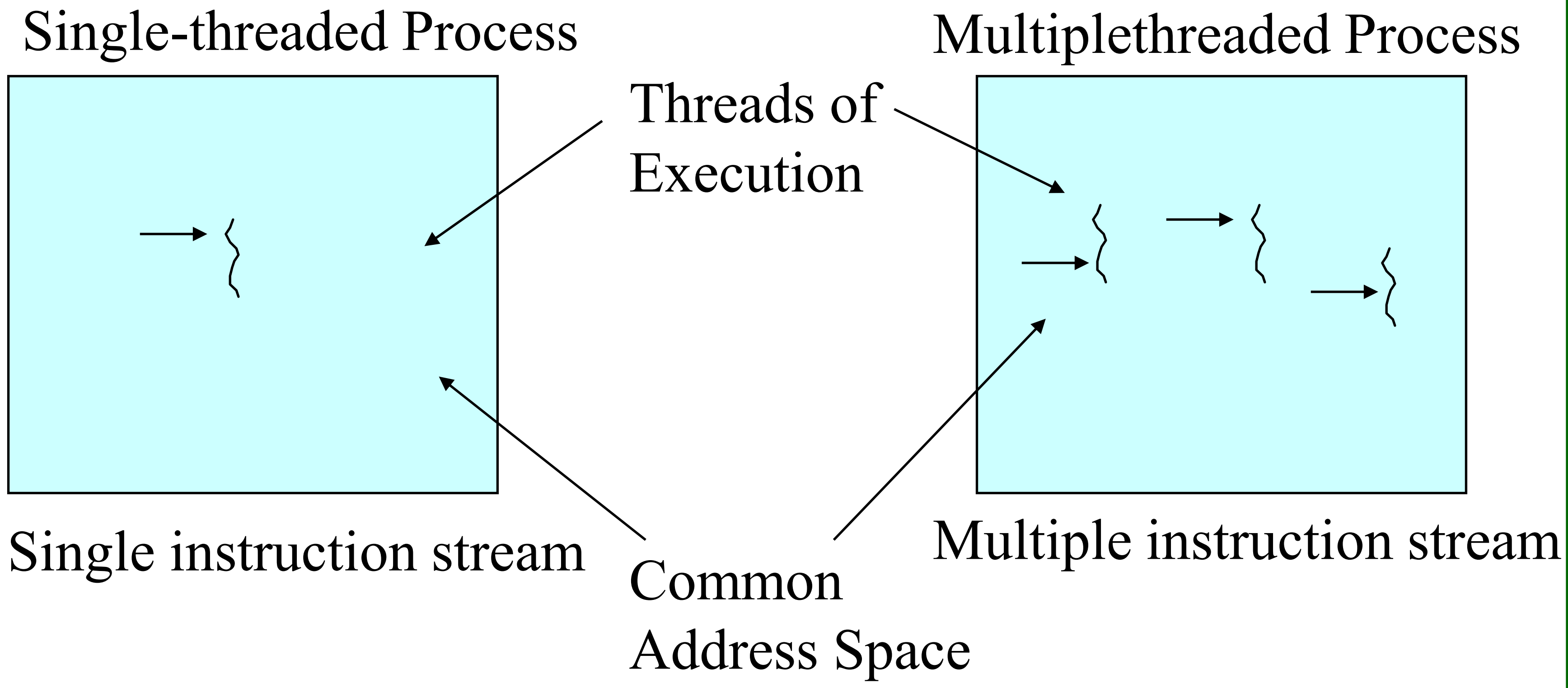
A Multithreaded Program



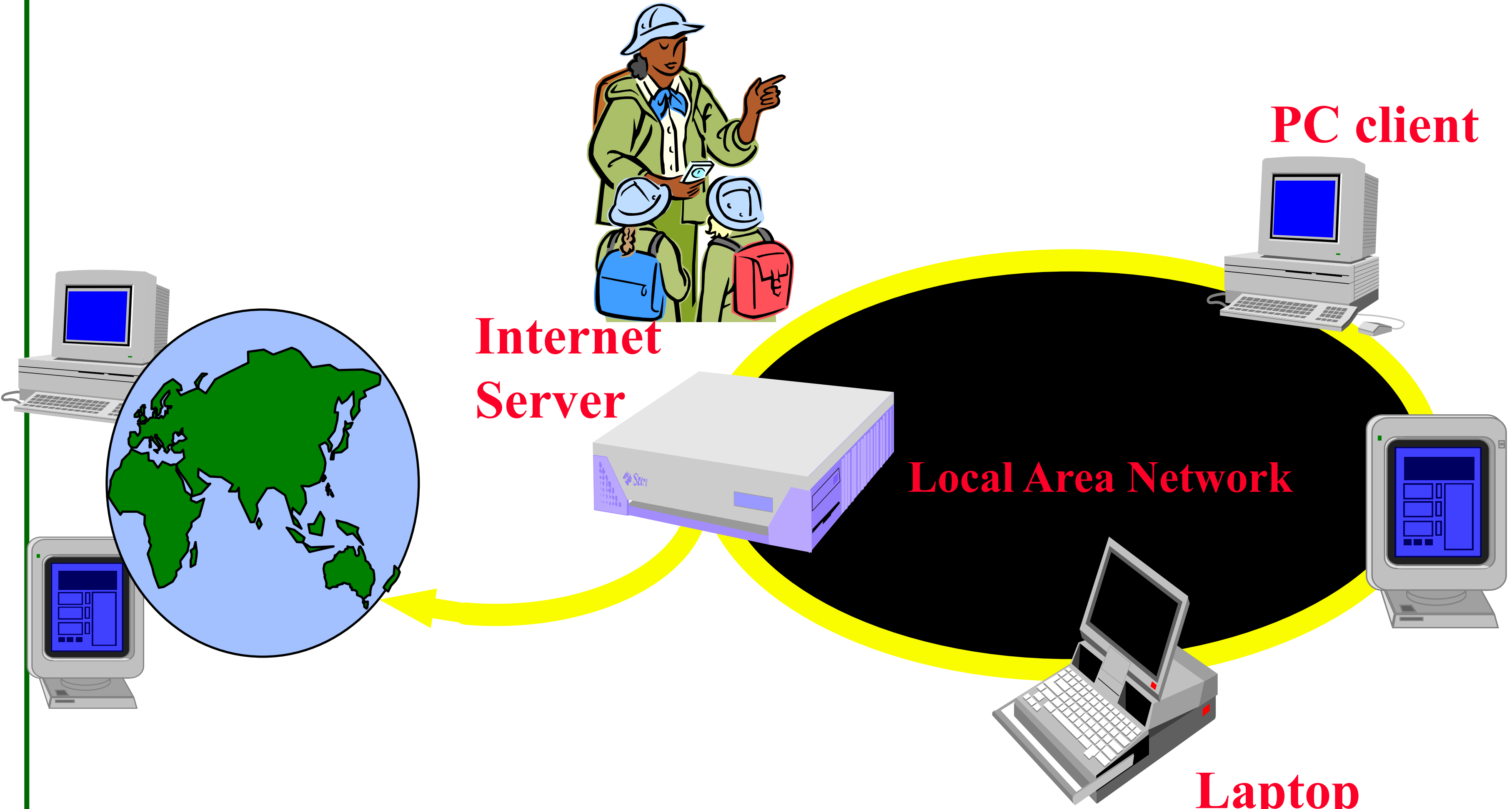
Threads may switch or exchange data/results

Single and Multithreaded Processes

threads are light-weight processes within a process



Web/Internet Applications: Serving Many Users Simultaneously



We have observed..

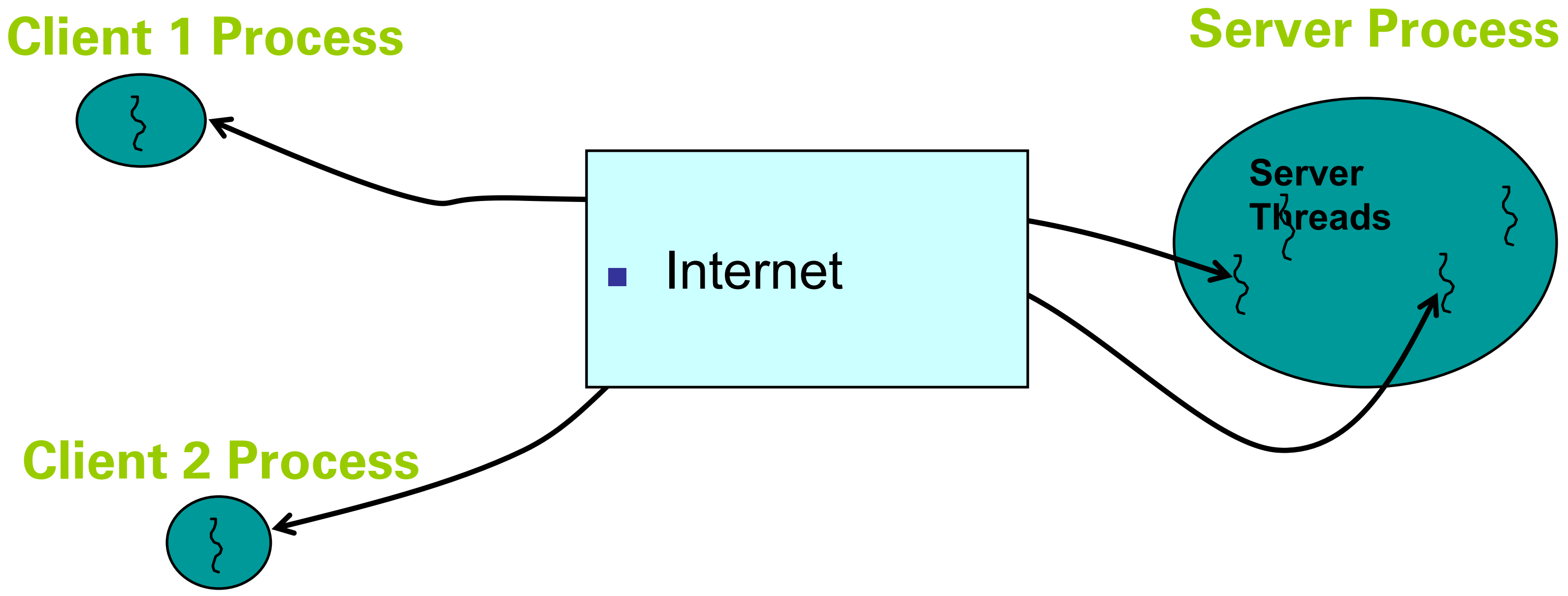
TCPServer.java

```
import java.io.*;
import java.net.*;
class TCPServer {
   public static void main(String argv[]) throws Exception
      {
        String clientSentence;
        String capitalizedSentence;

         ServerSocket welcomeSocket = new ServerSocket(6789);

         while(true) {

           Socket connectionSocket = welcomeSocket.accept();

           BufferedReader inFromClient = new BufferedReader(new
                InputStreamReader(connectionSocket.getInputStream()));

           DataOutputStream  outToClient =
              new DataOutputStream( connectionSocket.getOutputStream());

              clientSentence = inFromClient.readLine();

              capitalizedSentence = clientSentence.toUpperCase() + '₩n';

              outToClient.writeBytes(capitalizedSentence);

         }
      }
  }
```

The server is **blocked** until the client will send any message

So, all the other clients can't get any responses from the server

server should be able to serve Multiple Clients Concurrently: **Multithreaded Server**



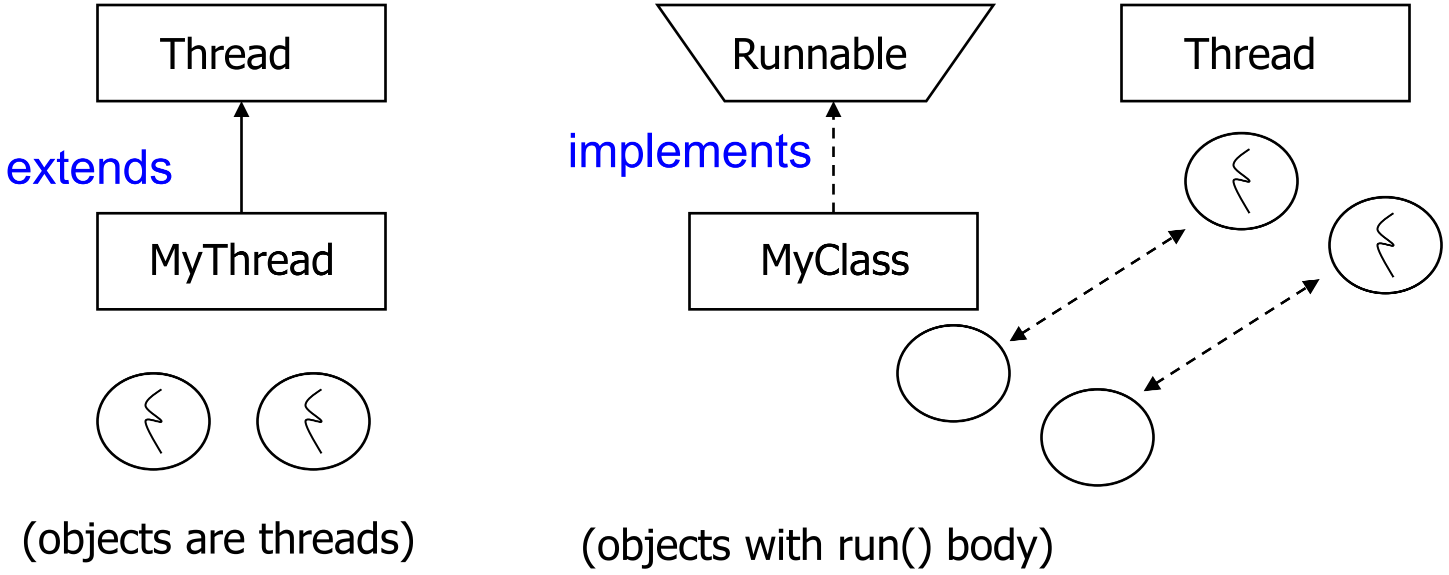
Java Threads

- Java has built in thread support for Multithreading
- Synchronization
- Thread Scheduling
- Inter-Thread Communication:

– currentThread	start	setPriority
– yield	run	getPriority
– sleep	stop	suspend
– resume		

Two ways to use Thread

- [a] Create a class that extends the **`Thread class`**
- [b] Create a class that implements the **`Runnable`** interface



[a]

[b]

1st method: Extending Thread class

- Create a class by extending Thread class and override run() method:

```
class MyThread extends Thread
{
     public void run()
    {
        // thread body of execution
    }
}
```

- **Create a thread:**

```
MyThread thr1 = new MyThread();
```

- **Start Execution of threads:**

```
thr1.start();
```

- or Create and Execute together:

```
new MyThread().start();
```

An example

```
class MyThread extends Thread {
        public void run() {
              System.out.println(" this thread is running ... ");
        }
}

class ThreadEx1 {
        public static void main(String [] args ) {
            MyThread t = new MyThread();
            t.start();
        }
}
```

Example 2

```
class MyThreadA extends Thread {
   public void run() { // entry point for thread
         for (;;) {
                  System.out.println("hello world1");
         }
   }
}

class MyThreadB extends Thread {
   public void run() { // entry point for thread
         for (;;) {
                  System.out.println("hello world2");
         }
   }
}

public class Main1 {
   public static void main(String [] args) {
         MyThreadA t1 = new MyThreadA();
         MyThreadB t2 = new MyThreadB();
         t1.start();
         t2.start();
         // main terminates, but in Java the other threads keep running
         // and hence Java program continues running
   }
}
```

2nd method: Threads by implementing Runnable interface

- Create a class that implements the interface Runnable and override run() method:

```
class MyThread implements Runnable
{
  .....
  public void run()
  {
     // thread body of execution
  }
}
```

  - Creating Object:

```
MyThread myObject = new MyThread();
```

  - Creating Thread Object:

```
Thread thr1 = new Thread( myObject );
```

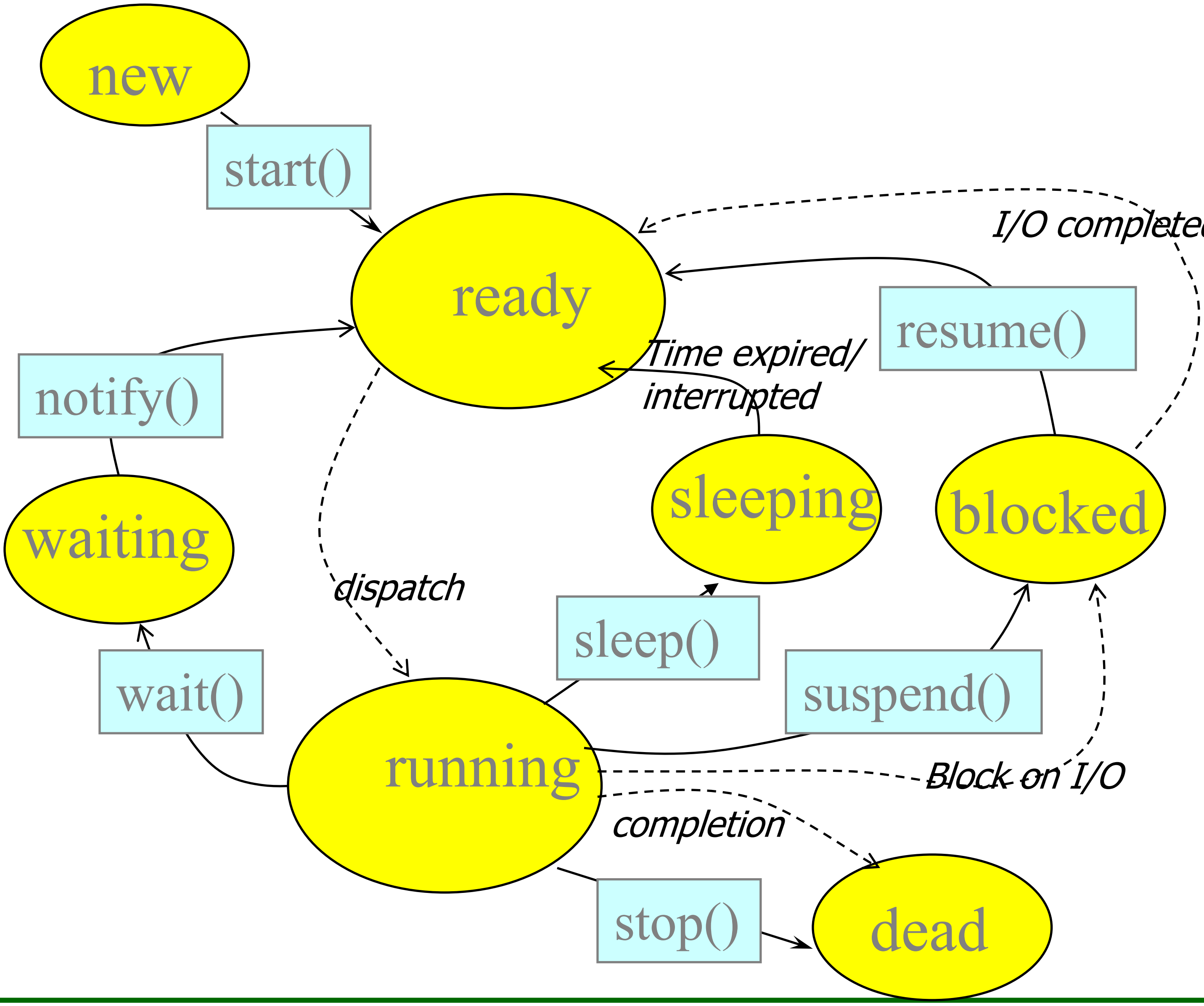
  - Start Execution:

```
thr1.start();
```

An example

```
class MyThread implements Runnable  {
        public void run() {
                System.out.println(" this thread is running ... ");
        }
}

class ThreadEx2 {
        public static void main(String [] args  ) {
                Thread t = new Thread(new MyThread());
                 t.start();
        }
}
```

Life Cycle of Thread



TRY : Three threads example

- Write a program that creates 3 threads

Three threads example

```
class A extends Thread
{
       public void run()
        {
              for(int i=1;i<=500;i++)
                {
                      System.out.println("\t From ThreadA: i= "+i);
                }
                 System.out.println("Exit from A");
        }
}

class B extends Thread
{
       public void run()
        {
              for(int j=1;j<=500;j++)
                {
                      System.out.println("\t From ThreadB: j= "+j);
                }
                 System.out.println("Exit from B");
        }
}
```

```
class C extends Thread
{
      public void run()
        {
            for(int k=1;k<=500;k++)
              {
                   System.out.println("\t From ThreadC: k= "+k);
              }

               System.out.println("Exit from C");
        }
}

class ThreadTest
{
         public static void main(String args[])
          {
                  A threadA = new A();
                  B threadB = new B();
                  C threadC = new C();
                  threadA.start();
                 threadB.start();
                 threadC.start();
                 System.out.println("Main() is terminated\n");
          }
}
```

Run!

```
        From ThreadA: i= 1
         From ThreadA: i= 2
         From ThreadA: i= 3
         From ThreadA: i= 4
         From ThreadA: i= 5
         From ThreadC: k= 1
         From ThreadC: k= 2
….
Main() is terminated
….
Exit from C
        From ThreadB: j= 495
        From ThreadB: j= 496
        From ThreadB: j= 497
        From ThreadB: j= 498
        From ThreadB: j= 499
        From ThreadB: j= 500
Exit from B
```

Add a line in the example

```
class ThreadTest
{
        public static void main(String args[])
         {
                A threadA = new A();
                B threadB = new B();
                C threadC = new C();
                threadA.start();
               threadB.start();
               threadC.start();

               System.out.println("Main() has
 terminated\n");
         }
}
```

```
try {
        threadA.join();
        threadB.join();
        threadC.join();
 } catch(InterruptedException e)  {

 }
```

What happens ?

Thread Method

- public final void join();
  - Wait until the thread is “not alive”
  - Threads that have completed are “not alive” as are threads that have not yet been started

Thread Priority

- In Java, each thread is assigned priority, which affects the order in which it is scheduled for running. The threads so far had same default priority (NORM_PRIORITY) and they are served using FCFS policy.
  - Java allows users to change priority:
    - ThreadName.setPriority(intNumber)
      - MIN_PRIORITY = 1
      - NORM_PRIORITY=5
      - MAX_PRIORITY=10

Thread Priority Example

```
class A extends Thread
{
        public void run()
          {
                System.out.println("Thread A started");
                for(int i=1;i<=4;i++)
                  {
                        System.out.println("\t From ThreadA: i= "+i);
                  }
                  System.out.println("Exit from A");
          }
}
class B extends Thread
{
        public void run()
          {
                System.out.println("Thread B started");
                for(int j=1;j<=4;j++)
                  {
                        System.out.println("\t From ThreadB: j= "+j);
                  }
                  System.out.println("Exit from B");
          }
}
```

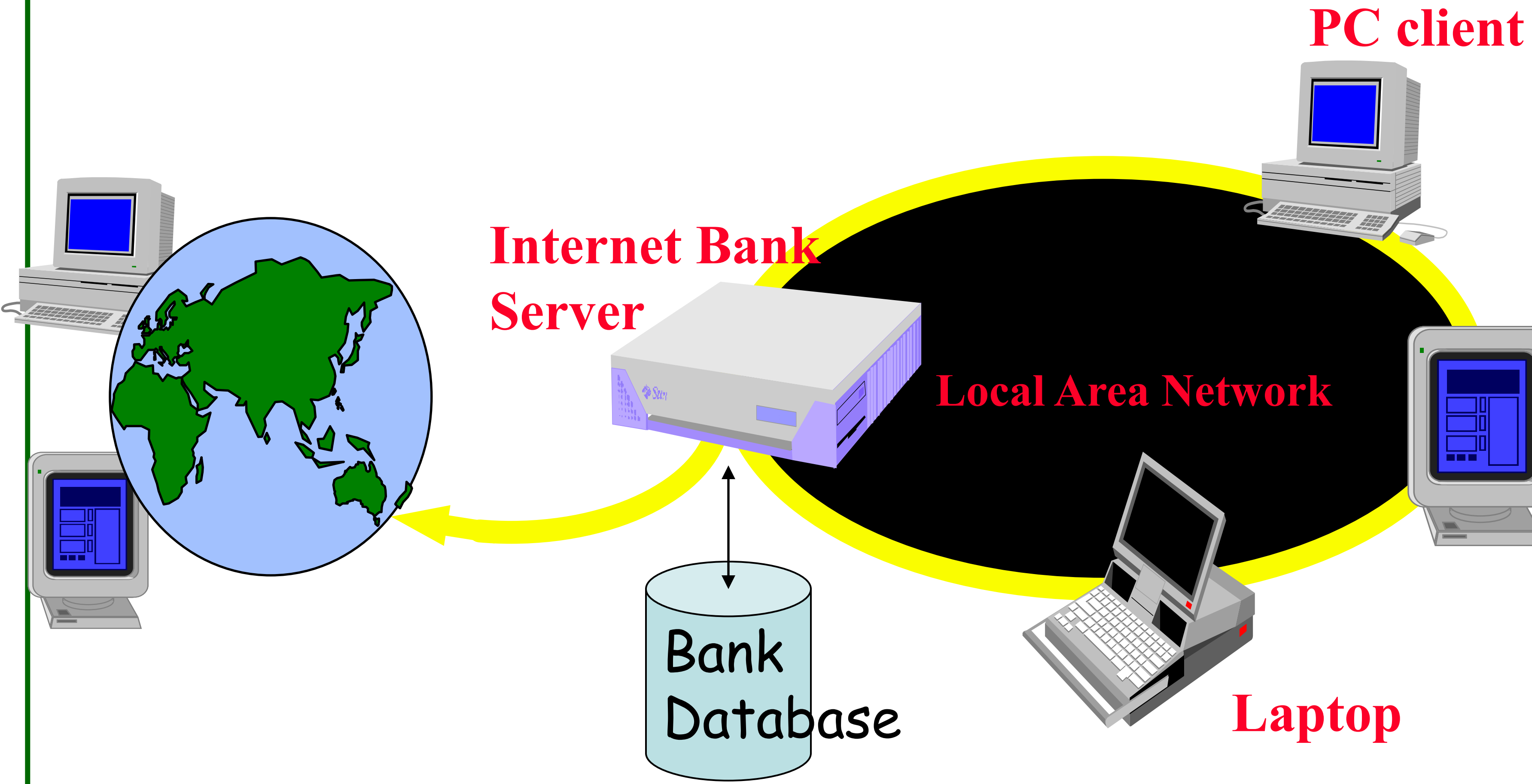
Thread Priority Example

```
class C extends Thread
{
       public void run()
        {
            System.out.println("Thread C started");
            for(int k=1;k<=4;k++)
             {
                 System.out.println("\t From ThreadC: k= "+k);
             }
             System.out.println("Exit from C");
        }
}
class ThreadPriority
{
          public static void main(String args[])
           {
                 A threadA=new A();
                 B threadB=new B();
                 C threadC=new C();
                 threadC.setPriority(Thread.MAX_PRIORITY);
                 threadB.setPriority(threadA.getPriority()+1);
                 threadA.setPriority(Thread.MIN_PRIORITY);
               System.out.println("Started Thread A");
                threadA.start();
               System.out.println("Started Thread B");
                threadB.start();
               System.out.println("Started Thread C");
                threadC.start();
                System.out.println("End of main thread");
           }
}
```

Accessing Shared Resources

- Applications Access to Shared Resources need to be coordinated.
  - Printer (two person jobs cannot be printed at the same time)
  - Simultaneous operations on your bank account.
  - Can the following operations be done at the same time on the same account?
    - Deposit()
    - Withdraw()
    - Enquire()

Online Bank: Serving Many Customers and Operations



Shared Resources

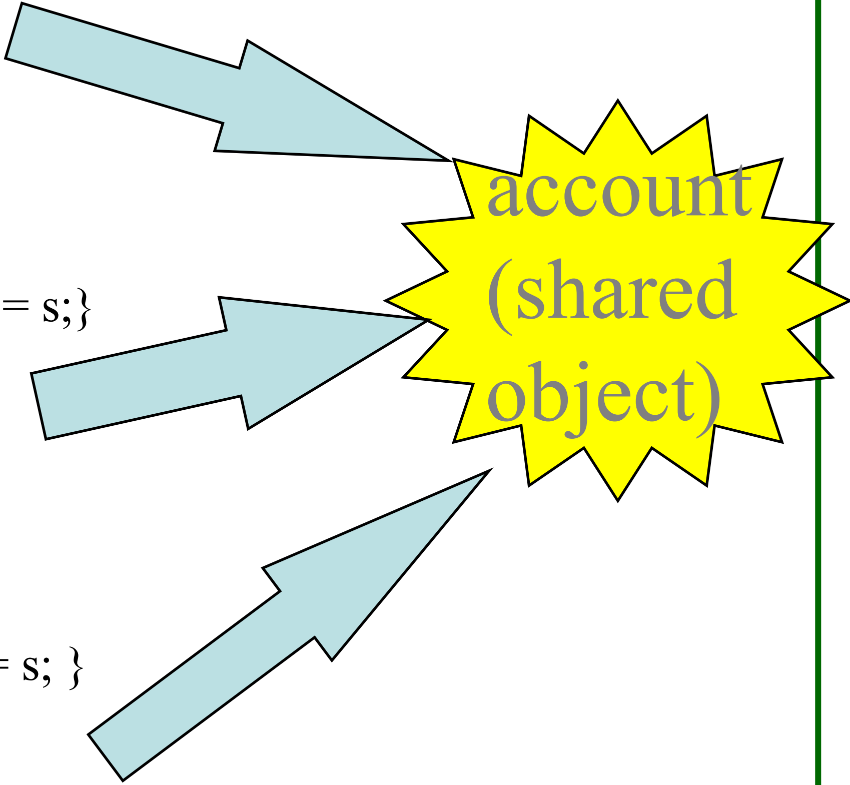
- If one thread tries to read the data and other thread tries to update the same data, it leads to inconsistent state.
- This can be prevented by synchronising access to the data.
- Use “Synchronized” method:
  - public synchronized void update()
  - {
    - …
  - }

the driver: 3 Threads sharing the same object

```
class InternetBankingSystem {
        public static void main(String [] args  ) {
            Account accountObject = new Account ();
            Thread t1 = new Thread(new MyThread(accountObject));
            Thread t2 = new Thread(new YourThread(accountObject));
            Thread t3 = new Thread(new HerThread(accountObject));
            t1.start();
            t2.start();
            t3.start();
          // DO some other operation
        } // end main()
}
```

Shared account object between 3 threads

```
class MyThread implements Runnable  {
 Account account;
       public MyThread (Account s) {  account = s;}
       public void run() { account.deposit(); }
} // end class MyThread

class YourThread implements Runnable  {
 Account account;
       public YourThread (Account s) { account = s;}
       public void run() { account.withdraw(); }
} // end class YourThread

class HerThread implements Runnable  {
 Account account;
       public HerThread (Account s) { account = s; }
       public void run() {account.enquire(); }
} // end class HerThread
```



Monitor (shared object access): serializes operation on shared object

```
class Account {   // the 'monitor'
   int balance;

       // if 'synchronized' is removed, the outcome is unpredictable
        public synchronized void deposit( ) {
           // METHOD BODY : balance += deposit_amount;
         }

          public synchronized void withdraw( ) {
            // METHOD BODY: balance -= deposit_amount;
          }
          public synchronized void enquire( ) {
            // METHOD BODY: display balance.
          }
}
```

References

- Rajkumar Buyya, Thamarai Selvi, Xingchen Chu, **Mastering OOP with Java**, McGraw Hill (I) Press, New Delhi, India, 2009.
- Sun Java Tutorial – Concurrency:
  - http://java.sun.com/docs/books/tutorial/essential/concurrency/

Ex1: Single-thread Server

```
import java.io.DataInputStream;
import java.io.PrintStream;
import java.io.IOException;
import java.net.Socket;
import java.net.ServerSocket;

public class Server {
  public static void main(String args[]) {

    ServerSocket echoServer = null;
    String line;
    DataInputStream is;
    PrintStream os;
    Socket clientSocket = null;

   /*
    * Open a server socket on port 2222. Note that we can't choose a port less
    * than 1023 if we are not privileged users (root).
    */
    try {
      echoServer = new ServerSocket(2222);
    } catch (IOException e) {
      System.out.println(e);
    }

    /*
     * Create a socket object from the ServerSocket to listen to and accept
     * connections. Open input and output streams.
     */
    System.out.println("The server started. To stop it press <CTRL><C>.");
    try {
      clientSocket = echoServer.accept();
      is = new DataInputStream(clientSocket.getInputStream());
      os = new PrintStream(clientSocket.getOutputStream());

      /* As long as we receive data, echo that data back to the client. */
      while (true) {
        line = is.readLine();
        os.println("From server: " + line);
      }
    } catch (IOException e) {
      System.out.println(e);
    }
  }
}
```

Ex-1: Simple Client

```
import java.io.DataInputStream;
import java.io.PrintStream;
import java.io.BufferedInputStream;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;

public class Client {
  public static void main(String[] args) {

    Socket clientSocket = null;
    DataInputStream is = null;
    PrintStream os = null;
    DataInputStream inputLine = null;

    /*
     * Open a socket on port 2222. Open the input and the output streams.
     */
    try {
      clientSocket = new Socket("localhost", 2222);
      os = new PrintStream(clientSocket.getOutputStream());
      is = new DataInputStream(clientSocket.getInputStream());
      inputLine = new DataInputStream(new BufferedInputStream(System.in));
    } catch (UnknownHostException e) {
      System.err.println("Don't know about host");
    } catch (IOException e) {
      System.err.println("Couldn't get I/O for the connection to host");
    }

    /*
     * If everything has been initialized then we want to write some data to the
     * socket we have opened a connection to on port 2222.
     */
```

Continues on the next page

Ex1: Client (cont.)

```
if (clientSocket != null && os != null && is != null) {
      try {

        /*
         * Keep on reading from/to the socket till we receive the "Ok" from the
         * server, once we received that then we break.
         */
        System.out.println("The client started. Type any text. To quit it type 'Ok'.");
        String responseLine;
        os.println(inputLine.readLine());
        while ((responseLine = is.readLine()) != null) {
          System.out.println(responseLine);
          if (responseLine.indexOf("Ok") != -1) {
            break;
          }
          os.println(inputLine.readLine());
        }

        /*
         * Close the output stream, close the input stream, close the socket.
         */
        os.close();
        is.close();
        clientSocket.close();
      } catch (UnknownHostException e) {
        System.err.println("Trying to connect to unknown host: " + e);
      } catch (IOException e) {
        System.err.println("IOException:  " + e);
      }
    }
  }
}
```

Ex2:
Multithreaded Chat Server/Client

- Enclosed sample java files
- The chat server
  - It uses a separate thread for each client.
  - It spawns a new client thread every time a new connection from a client is accepted.
  - This thread opens the input and the output streams for a particular client, it ask the client's name, it informs all clients about the fact that a new client has joined the chat room and, as long as it receive data, echos that data back to all other clients.
  - When the client leaves the chat room, this thread informs also the clients about that and terminates.

Important Classes

1. **InetAddress**

2. **Socket**
   ServerSocket

3. **DatagramSocket**
   DatagramPacket

4. **URL, URLConnection ..**

Question

- How to obtain the IP address from a domain name ?

Question

- How to obtain your local IP address ?

InetAddress class

- static methods you can use to create new InetAddress objects.
  - static InetAddress **getByName(String host)**
  - static InetAddress[] **getAllByName(String host)**
    - e.g. daum.net, naver.com, …
  - static InetAddress **getLocalHost()**

```
InetAddress x  = InetAddress.getByName(
                                 "sw.gachon.ac.kr");
InetAddress local = InetAddress.getLocalHost();
```

❖ Throws `UnknownHostException`

InetAddress class (Example)

```
try {

  InetAddress ad = InetAddress.getByName(hostname);
  System.out.println(hostname + ":" +
                   ad.getHostAddress());

} catch (UnknownHostException e) {

  System.out.println("No address found for " +
                          hostname);

}
```

Example : TRY !!

```
import java.net.*;

public class InetAddressExample {

        public static void main(String[] args) {

                // Get name and IP address of the local host
                try {
                        InetAddress address = InetAddress.getLocalHost();
                        System.out.println("Local Host:");
                        System.out.println("\t" + address.getHostName());
                        System.out.println("\t" + address.getHostAddress());
                } catch (UnknownHostException e) {
                        System.out.println("Unable to determine this host's address");
                }

                for (int i = 0; i < args.length; i++) {
                        // Get name(s)/address(es) of hosts given on command line
                        try {
                                InetAddress[] addressList = InetAddress.getAllByName(args[i]);
                                System.out.println(args[i] + ":");
                                // Print the first name. Assume array contains at least one entry.
                                System.out.println("\t" + addressList[0].getHostName());
                                for (int j = 0; j < addressList.length; j++)
                                        System.out.println("\t" + addressList[j].getHostAddress());
                        } catch (UnknownHostException e) {
                                System.out.println("Unable to find address for " + args[i]);
                        }
                }
        }
}
```

TRY it out

- Run with the following arguments
  - sw.gachon.ac.kr   www.naver.com www.google.com

End.