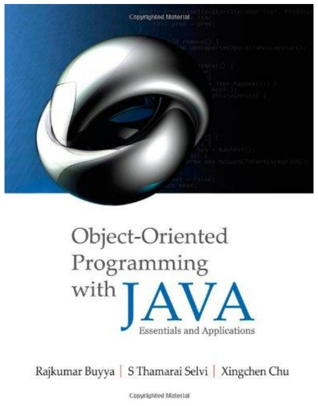
OS Support for Building Distributed Applications: Multithreaded Programming using Java Threads


Object-Oriented Programming with JAVA
Essentials and Applications
Rajkumar Buyya | S Thamarai Selvi | Xingchen Chu

Dr. Rajkumar Buyya

Cloud Computing and **D**istributed **S**ystems (CLOUDS) Laboratory

School of Computing and Information Systems

The University of Melbourne, Australia

http://www.buyya.com

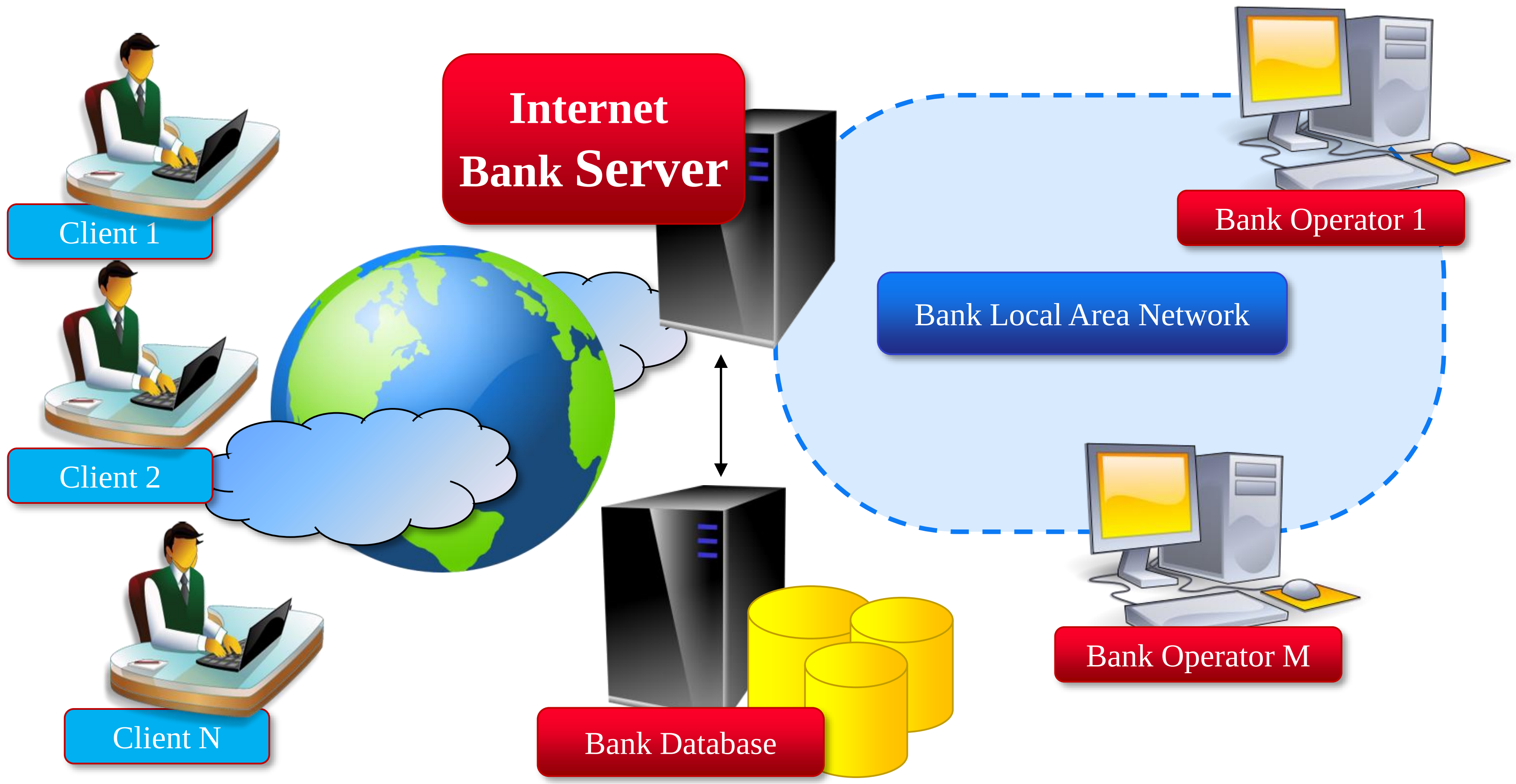
Agenda

- Introduction
- Thread Applications
- Defining Threads
- Java Threads and States
- **Examples**

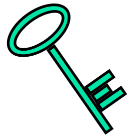
Accessing Shared Resources

- 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

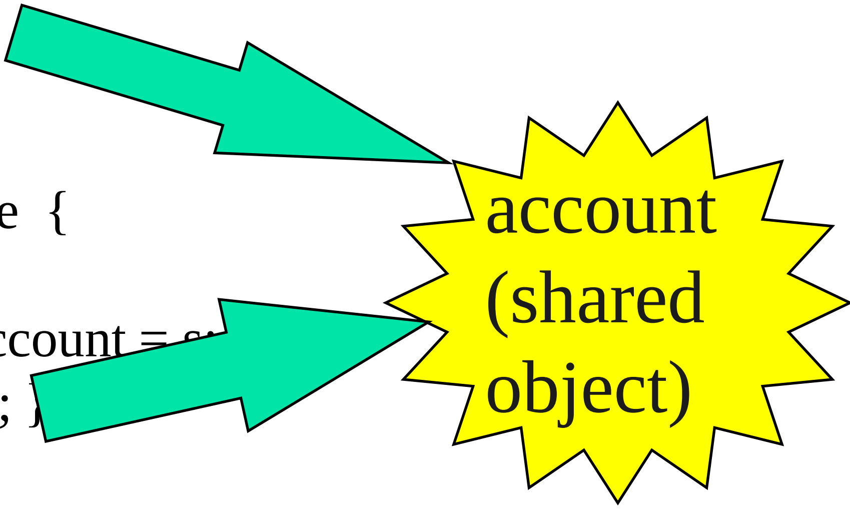


- If one thread tries to read the data and other thread tries to update the same data, it leads to inconsistent state.
- This can be prevented by synchronising access to the data.
- Use "synchronized" method:
  - public synchronized void update()
  - {
    - …
  - }

Shared account object between 3 threads

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

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



Producer and Consumer Problem

- Two threads, the producer and the consumer, share a common fixed-length buffer
- Producers generate a piece of data and put it into the buffer.
- The consumer is consuming data from the same buffer simultaneously
- Problem?
- Solution?

```
/* MessageQueue.java: A message queue with synchronized methods for queuing
and consuming messages. */

package com.javabook.threading;
import java.util.ArrayList;
import java.util.List;

public class MessageQueue {
   //the size of the buffer
   private int bufferSize;

   //the buffer list of the message, assuming the string message format
   private List<String> buffer = new ArrayList<String>();

   //construct the message queue with given buffer size
   public MessageQueue(int bufferSize){
       if(bufferSize<=0)
           throw new IllegalArgumentException("Size is illegal.");
       this.bufferSize = bufferSize;
   }
   //check whether the buffer is full
   public synchronized boolean isFull() {
       return buffer.size() == bufferSize;
}

//check whether the buffer is empty
public synchronized boolean isEmpty() {
    return buffer.isEmpty();
}

//put an income message into the queue, called by message producer
public synchronized void put(String message) {
    //wait until the queue is not full
    while (isFull()) {
        System.out.println("Queue is full.");
        try{
            //set the current thread to wait
            wait();
        }catch(InterruptedException ex){
            //someone wake me up.
        }
    }
    buffer.add(message);
    System.out.println("Queue receives message '"+message+"'");

    //wakeup all the waiting threads to proceed
    notifyAll();
}
```

```
    //get a message from the queue, called by the message consumer
    public synchronized String get(){
        String message = null;
        //wait until the queue is not empty
        while(isEmpty()){
            System.out.println("There is no message in queue.");
            try{
                //set the current thread to wait
                wait();
            }catch(InterruptedException ex){
                //someone wake me up.
            }
        }
        //consume the first message in the queue
        message = buffer.remove(0);

        //wakeup all the waiting thread to proceed
        notifyAll();
        return message;
    }
} //end MessageQueue class
```

The Producer

```
/* Producer.java: A producer that generates messages and put into a given
message queue. */

package com.javabook.threading;

public class Producer extends Thread{
   private static int count = 0;
   private MessageQueue queue = null;

   public Producer(MessageQueue queue){
       this.queue = queue;
   }

   public void run(){
       for(int i=0;i<10;i++){
           queue.put(generateMessage());
       }
   }

   private synchronized String generateMessage(){
       String msg = "MSG#"+count;
       count ++;

       return msg;
   }
} //end Producer class
```

The Consumer

```
/* Consumer.java: A consumer that consumes messages from the queue. */
package com.javabook.threading;

public class Consumer extends Thread {
   private MessageQueue queue = null;

   public Consumer(MessageQueue queue){
      this.queue = queue;
   }

   public void run(){
      for(int i=0;i<10;i++){
         System.out.println("Consumer downloads "
            +queue.get()+ " from the queue.");
      }
   }
} //end Consumer class
```

Main Program

```
/* MessageSystem.java: A message system that demonstrate the produce and
consumer problem with the message queue example. */

package com.javabook.threading;

public class MessageSystem {
   public static void main(String[] args) {
      MessageQueue queue = new MessageQueue(5);
      new Producer(queue).start();
      new Producer(queue).start();
      new Producer(queue).start();
      new Consumer(queue).start();
      new Consumer(queue).start();
      new Consumer(queue).start();
   }
} //end MessageSystem class
```

Required Readings

- Chapter 14: Multithread Programming
  - R. Buyya, S. Selvi, X. Chu, **"Object Oriented Programming with Java: Essentials and Applications",** McGraw Hill, New Delhi, India, 2009.