Overview

• Introduction to Threads
• Class Thread and Thread methods
  • Example of simple thread
• Thread Synchronization
  • Example of rendezvous
• The Runnable Interface
• Thread Groups

Threads

• A thread is an execution path in a program.
• We refer to the concurrent execution of more than one thread in a program as multithreading.
• Java supports multithreading in the language constructs.
• Unique among many general purpose programming languages, including C and C++.
Priorities and Scheduling

• There is one CPU: which thread gets executed?
• Each thread is assigned a priority: higher priority gets execution before lower priority.
• The scheduling policy of the threads by their priorities is slightly different on different platforms: Solaris vs Win32.

Solaris: non-preemptive multithreading
• Higher priority thread gets execution first.
• Change priority threads on a first come first serve basis.
• Once a thread gets execution, it runs until it is blocked or voluntarily releases.

Win32: time-sliced multithreading
• Higher priority thread gets execution first.
• Same priority threads share in a round-robin basis, each getting a time-slice.
• Once a thread gets execution, it runs until it is blocked, it voluntarily gives up, or when its allotted time-slice expires.

Java Thread Priority Scheduling

<table>
<thead>
<tr>
<th>Priority 9</th>
<th>Priority 8</th>
<th>Priority 7</th>
<th>Priority 2</th>
<th>Priority 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
</tbody>
</table>
Life Cycle of a Thread

born
ready
running
sleeping
waiting
blocked
dead
start
priority
scheduling
yield
wait
notify (or notifyAll)
sleep
completion
i/o
sleep time
expires…

Class Thread

• java.lang.Thread
• Class Thread encapsulates the primitive constructs for multi-threading in Java.
• Constructors…
• public Thread(String threadName);
• public Thread(); // use default name.
• …override the run method.
• public void run();

class MyThread extends Thread
{public void run()
{     System.err.println("I am running!"); }
}

• To launch a new thread, use the start method:
public void start();
MyThread mt = new MyThread();
mt.start(); // starts a new thread to call run()
… // returns immediately after new thread
// is launched; to be running concurrently.
Class Thread

• static method sleep:
  public static void sleep(long millis);
  ... to sleep for a number of milliseconds.
  ... will not contend for CPU for a while.

• static method yield:
  public static void yield();
  ... to yield CPU to other running threads.
  ... will not contend for CPU now.

Class Thread

• public final void setName(String name);
  ... specifies the name of this thread.

• public final String getName();
  ... returns the name of this thread.

• public static Thread currentThread();
  ... returns the thread currently running.

• public final void join();
  ... waits for this thread to terminate.

Class Thread

• public int getPriority();
  ... returns the priority of this thread.

• public void setPriority(int newPriority);
  ... sets the priority of this thread to newPriority.

• public static final int MAX_PRIORITY;
• public static final int MIN_PRIORITY;
• public static final int NORM_PRIORITY;
  ... for use in the system to get/set priorities.
Simple Thread – Driver

```java
public class SimpleThreadExample extends JFrame {
    public SimpleThreadExample() {
        JTextArea printspace = new JTextArea();
        getContentPane().add(printspace);
        AThread[] at = new AThread[10];
        for (int i=0;i<10;i++)
            at[i] = new AThread(new String("Thread"+i), printspace);
        for (int i=0;i<10;i++)
            at[i].start();
    }
    public static void main(String[] args) {
        new SimpleThreadExample();
    }
}
```

Simple Thread – Proper

```java
class AThread extends Thread {
    private String name;
    private JTextArea output;
    public AThread(String name, JTextArea output) {
        this.name = name;
        output = output;
        output.append("starting " + getname() + "\n");
    }
    public void run() {
        try {Thread.sleep((int) (Math.random() * 10000));}
        catch (InterruptedException e) {
            System.err.println(e.toString());
        }
        output.append("\t" + getname() + " exiting\n");
    }
}
```

Synchronization

- For threads to cooperate, sharing access to some common facilities... we need synchronization!
- Java uses monitors (CACM paper by Hoare, 1974)
  "Monitors: an operating system structuring concept."
- Synchronization is implemented with a special concept called the class Thread, but built into all Java objects.
- Any method can be declared synchronized.
- Every object with synchronized methods is a monitor.
- It allows only ONE thread at a time to execute a synchronized method on the object.
- synchronized(Obj) { ... } is just a block of code...
synchronization: Object methods

- `public final void wait();`
- `public final void wait(long millis);`
- `public final void notify();`
- `public final void notifyAll();`

Rendezvous Example

- In order to manage a rendezvous, we need four classes:
  - A driver class
  - One or more producer threads
  - One or more consumer threads
  - A synchronization monitor

Rendezvous Driver

```java
public class SynchThreads extends JFrame {
    public SynchThreads() {
        getContentPane().setLayout(new BorderLayout());
        JTextArea printspace = new JTextArea();
        getContentPane().add(printspace, BorderLayout.CENTER);
        SynchMsgMon smm = new SynchMsgMon(printspace);
        CThread reader = new CThread("Reader", printspace, smm);
        reader.start();
        PThread[] at = new PThread[10];
        for (int i = 0; i < 10; i++) {
            at[i] = new PThread(new String("PT" + i), printspace, smm);
            at[i].start();
        }
        public static void main(String args[]) {
            SynchThreadsf = new SynchThreads();
        }
    }
```
public class SynchMessageMonitor
{
    private String ma[] = new String[10];
    private boolean messages = false;
    private boolean writable = true;
    private boolean readable = false;
    private int nmsgs = 0;
    private int lastread = 0;
    private int lastwritten = 0;
    private JTextArea output;
    public SynchMessageMonitor(JTextArea o)
    {
        output = o;
    }

    public synchronized void writeMessage(String name, String msg)
    {
        while (!writable)
        {
            try
            {
                output.insert(name + " waiting to store msg\n", 0);
                wait();
            }
            catch (InterruptedException e)
            {
                System.err.println(e.toString());
            }
        }
        readable = true;
        ma[lastwritten] = new String(msg);
        lastwritten = (lastwritten + 1) % 10;
        nmsgs++;
        output.insert("There are " + nmsgs + " msgs\n", 0);
        if (nmsgs == 10)
        {
            writable = false;
            output.insert("BUFFER IS FULL\n", 0);
            notify();
        }
    }

    public synchronized String readMessage(String name)
    {
        while (!readable)
        {
            try
            {
                output.insert(name + " waiting to read message\n", 0);
                wait();
            }
            catch (InterruptedException e)
            {
                System.err.println(e.toString());
            }
        }
        writeable = true;
        String msg = new String(ma[lastread]);
        lastread = (lastread + 1) % 10;
        nmsgs--;
        if (nmsgs == 0)
        {
            readable = false;
            output.insert("BUFFER IS EMPTY\n", 0);
            notify();
            return msg;
        }
Rendezvous Consumer

Rendezvous Producer

The Runnable Interface

• We extend class Thread to create a new class to support multi-threading. But if we have a class already, and it is not derived from class Thread, how can we support multi-threading in that class?

• We must implement Runnable interface in that class.

interface Runnable
{
    public void run();
}

• We only need to implement the run method to implement the Runnable interface.
a thread in a Runnable object...

- We can then create a thread for the Runnable object, using these Thread constructors...

  ```java
  public Thread(Runnable obj);
  public Thread(Runnable obj, String name);
  ```

- We then call the start method on the thread...

  ```java
  class X implements Runnable {
      ...
      
      X obj = new X();
      Thread tx = new Thread(obj, "MyThread");
      tx.start();    // to start the thread
  }
  ```

Thread Groups

- When we have many threads, it would be useful to organize threads into groups.

- In Java, the class ThreadGroup provides the facility:

  ```java
  ThreadGroup threadGroup = new ThreadGroup("My Group");
  Thread t = new Thread(new Runnable() {
      public void run() {
          // thread body
      }
  }, "Thread 1", threadGroup);
  t.start();
  ```

- A ThreadGroup is a group of Threads, but can also be parent to other ThreadGroups.

- ThreadGroup constructors...

  ```java
  public ThreadGroup(String name);
  public ThreadGroup(ThreadGroup tg, String name);
  ```

associating a thread to a group...

- Class Thread has these 3 constructors...

  ```java
  public Thread(ThreadGroup tg, String name);
  public Thread(ThreadGroup tg, Runnable obj);
  public Thread(ThreadGroup tg, Runnable obj, String name);
  ```

- The constructors allow us to create a thread associated to a particular thread group.

- The class ThreadGroup provides methods to manage groups of threads.
ThreadGroup methods

• public int activeCount();
• public int enumerate(Thread[] list);
• public int enumerate(Thread[] list, boolean all);
• public int enumerate(ThreadGroup[] list);
• public int enumerate(ThreadGroup[] list, boolean all);

ThreadGroup methods

• public int getMaxPriority();
• public void setMaxPriority(int pri);
• public String getName();
• public ThreadGroup getParent();
• public boolean parentOf(ThreadGroup tg);