Multithreaded Programming

Advanced Topics

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/java5.html

Agenda

- Timers
- wait/notify
- Deadlock

Timers

- For simple periodic triggering of actions, Java defines a Timer class
  – A Timer can ring for a single cycle or fire periodically
  – At each ring an ActionEvent is fired
  – Timer is in Swing package (javax.swing.Timer)
    - And uses classes from java.awt.event.*
  – Useful for animations, simulations, timing out secure network connections, and other cases with actions at fixed intervals
- Approach

```
Timer timer = new Timer(milliseconds, listener);
timer.start();
...
timer.stop();
```

Useful Timer Methods

- **start/stop**
  - Starts or stops the timing sequence
- **restart**
  - Cancels any undelivered time events and starts the timer again
- **setCoalesce**
  - Turns coalescing off or on
  - By default, if a timer event is already in the event queue (coalesce true), a new ActionEvent is not created at the next firing interval
- **setRepeats**
  - Sets the timer to ring once (false) or to ring periodically (true)
  - Default behavior is to ring periodically

Timer: Example

```
import java.awt.*;
import javax.swing.*;

public class TimedAnimation extends JApplet {
  private static final int NUMDUKES = 2;
  private TimedDuke[] dukes;
  private int i, index;

  public void init() {
    dukes = new TimedDuke[NUMDUKES];
    setBackground(Color.white);
    dukes[0] = new TimedDuke( 1, 100, this);
    dukes[1] = new TimedDuke(-1, 500, this);
  }

  //  Start each Duke timer.
  public void start() {
    for (int i=0; i<NUMDUKES ; i++) {
      dukes[i].start();
    }
  } ...
```

Timer Example (Continued)

```
  ...
  public void paint(Graphics g) {
    for (i=0 ; i<NUMDUKES ; i++) {
      if (dukes[i] != null) {
        index = dukes[i].getIndex();
        g.drawImage(TimedDuke.images[index], 200*i, 0, this);
      }
    }
  }

  //  Stop each Duke timer.
  public void stop() {
    for (int i=0; i<NUMDUKES ; i++) {
      dukes[i].stop();
    }
  }
}
```

Timer Example (Continued)

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TimedDuke extends Timer
                       implements ActionListener {
  private static final int NUMIMAGES = 15;
  private static boolean loaded = false;
  private static Object lock = new Object();
  private int tumbleDirection;
  private int index = 0;
  private Applet parent;
  public static Image[] images = new Image[NUMIMAGES];

  public TimedDuke(int tumbleDirection, int msec,
                                        Applet parent) {
    super(msec, null);
    addActionListener(this);
    this.tumbleDirection = tumbleDirection;
    this.parent = parent; ...
```

Timer Example (Continued)

```
    synchronized (lock) {
      if (!loaded) {
         // Load images using MediaTracker
          ...
      }
    }
  }

  public int getIndex() { return index; }

  public void actionPerformed(ActionEvent event) {
    index += tumbleDirection;
    if (index < 0){
      index = NUMIMAGES - 1;
    }
    if (index >= NUMIMAGES) {
      index = 0;
    }
    parent.repaint();
  }
}
```

Timer Example: Result

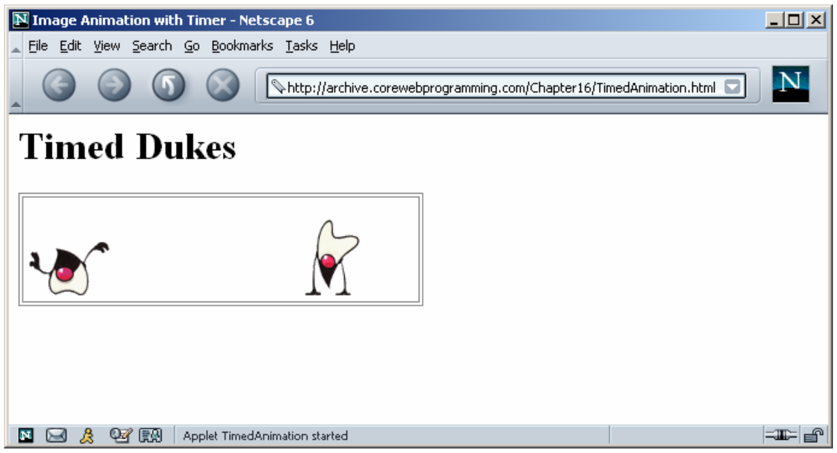
Image Animation with Timer - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://archive.corewebprogramming.com/Chapter16/TimedAnimation.html
Timed Dukes
Applet TimedAnimation started

Each Duke moves at a different speed

Duke is a registered trademark of Sun Microsystems, Inc.
All restrictions apply (http://www.sun.com/policies/trademarks/).
Used with permission.

wait/notify: Idea

- **What if threads need some resource to proceed, but resource is not available?**
  - They could sleep and recheck every so often, but checking too often or not often enough are both wasteful.
  - Instead, the threads say "put me to sleep (wait) and then wake me up whenever anything happens that could potentially free up the resource (notify)".
    - You can call notify to wake up one waiting thread, or call notifyAll to wake up all of them (and perhaps let them fight for the resource)

wait/notify: Basic Code

- **wait**

```
synchronized(this) {
  ...
  while(!someNecessaryCondition()) {
    wait();
  }
  doStuffThatRequiresCondition();
}
```

- **notify (or notifyAll)**

```
synchronized(this) {
  ...
  freeUpSomeResource();
  notifyAll();
  ...
}
```

wait/notify Example: Database Connection Pool (Background)

- **Creating a new connection to a database is much slower than sending a query over an existing connection**
- **So you typically allocate and recycle several connections**
- **Issues**
  - Connections can time out
  - Pools are sometimes of variable size (initial size plus some upper limit)

Connection Pooling: Continued

- **Problem**
  - What do you do if you want a connection, none is available, but upper limit is not yet reached?
    - Directly opening a new connection is wrong, because a thread might return an existing connection before a new connection is made (so you would wait too long)
- **Solution: wait/notify**
  - You say "start a Thread that will open a new connection, then put me to sleep (wait)".
  - Then, whenever a connection becomes available for any reason (new connection done being established or existing connection returned to the pool), say "wake up everyone waiting for connections and let them try again to get one (notifyAll)".

Connection Pooling: Code

```
public synchronized Connection getConnection()
    throws SQLException {
  if (!availableConnections.isEmpty()) {
    int lastIndex = availableConnections.size() - 1;
    Connection existingConnection =
      (Connection)availableConnections.get(lastIndex);
    availableConnections.remove(lastIndex);
    if (existingConnection.isClosed()) {
      notifyAll(); // Freed up a spot for anybody waiting
      return(getConnection());
    } else {
      busyConnections.add(existingConnection);
      return(existingConnection);
    }
  } else {
    if ((totalConnections() < maxConnections) &&
        !connectionPending) {
      makeBackgroundConnection();
    try {
      wait();
    } catch(InterruptedException ie) {}
    // Someone freed up a connection, so try again.
    return(getConnection());
  }
}
```

Connection Pooling: Code

```
private void makeBackgroundConnection() {
  connectionPending = true;
  try {
    Thread connectThread = new Thread(this);
    connectThread.start();
  } catch(OutOfMemoryError oome) {
    // Give up on new connection
  }
}

public void run() {
  try {
    Connection connection = makeNewConnection();
    synchronized(this) {
      availableConnections.add(connection);
      connectionPending = false;
      notifyAll();
    }
  } catch(Exception e) { // SQLException or OutOfMemory
    // Give up on new connection and wait for existing one
    // to free up.
  }
}
```

Deadlock: Circular Waiting

- **What if threads need two resources to operate**
  - And Thread1 gets the first one of them (waiting for the second)
  - And Thread2 gets the second one of them (waiting for the first)
- **Dining Philosophers**
- **Not deadlock**

```
synchronized(this) {          public synchronized
  ...                                   void foo() {
  foo();                          ...
}                             }
```

Summary

- **Timers**
  - Generate ActionEvent when triggered
    - Use actionPerformed to respond
- **wait/notify**
  - If thread cannot proceed ... (test condition again)
- **Deadlock**
  - Avoid applications where threads require multiple resources to proceed, but where resources can be acquired in any order

Questions?