Handling User Cancel Requests

How to cancel an AsyncTask

Call cancel(boolean) on AsyncTask
to interrupt AsyncTask thread

In AsyncTask
Call isCancelled() to see if have been cancelled
If in read on SocketChannel catch exceptions
  NotYetConnectedException
  ClosedChannelException
  AsynchronousCloseException
  ClosedByInterruptException
  IOException

Implement onCancelled() method
  Run on UI thread
  Used to clean up
Mars Example

- Passengers
- Total Weight
- MPG
- Mile per Year
- Years to drive

Compute Cost  Cancel

Passengers  1
Total Weight  200
MPG  44
Mile per Year  10000
Years to drive

Compute Cost  Cancel
delayedtrip Message

Command on Mars Server

Waits 30 seconds before sending back response

For testing Cancel operation
public class MainActivity extends Activity {
    TripDetailsTask getResult;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }

    public void cancel(View aView) {
        boolean wasCancelled = getResult.cancel(true);
    }
}

MainActivity
MainActivity

public void compute(View aView) {
    Log.i("mars","compute now");
    TextView peopleView = (TextView) this.findViewById(R.id.people);
    TextView weightView = (TextView) this.findViewById(R.id.weight);
    TextView mpgView = (TextView) this.findViewById(R.id.mpg);
    TextView milesPerYearView = (TextView) this.findViewById(R.id.milePerYear);
    int peopleCount = Integer.parseInt(peopleView.getText().toString());
    float weight = Float.parseFloat(weightView.getText().toString());
    float mpg = Float.parseFloat(mpgView.getText().toString());
    float milesPerYear = Float.parseFloat(milesPerYearView.getText().toString());
    TextView totalMiles = (TextView) this.findViewById(R.id.totalMiles);
    ProgressBar showWait = (ProgressBar) this.findViewById(R.id.serverWait);
    MarsClient client = new MarsClient("bismarck.sdsu.edu",8009);
    getResult = new TripDetailsTask(totalMiles, showWait,client);
    getResult.execute((float)peopleCount,weight, mpg, milesPerYear);
}
public class TripDetailsTask extends AsyncTask<Float, Void, Float> {
    TextView totalMiles;
    ProgressBar waiting;
    MarsClient client;

    public TripDetailsTask(TextView out, ProgressBar bar, MarsClient client) {
        super();
        totalMiles = out;
        waiting = bar;
        this.client = client;
    }

    protected void onPreExecute () {
        waiting.setVisibility(View.VISIBLE);
    }
}
protected Float doInBackground(Float... params) {
    int people = (int)(float)params[0];
    float weight = params[1];
    float mpg = params[2];
    float milesPerYear = params[3];
    Hashtable<String, Float> result;
    if (isCancelled()) return (float) -1;
    try {
        result = client.trip(people, weight, mpg, milesPerYear);
        client.quit();
        return result.get("food") + result.get("weight");
    } catch (ClosedByInterruptException e) {
        Log.i("mars", "User canceled operation");
    } catch (IOException e) {
        Log.e("mars", "error on get result from server", e);
    } finally {
        client.quit();
    }
    return (float) -1;
}
protected void onCancelled(Float result) {
    waiting.setVisibility(View.INVISIBLE);
}

protected void onPostExecute(Float result) {
    Log.i("mars", "in Post");
    waiting.setVisibility(View.INVISIBLE);
    if (result == -1) return;
    DecimalFormat format = new DecimalFormat("#.#");
    totalMiles.setText(format.format(result));
}

}
Handling Multiple Request
Basic Situation

AsyncTask runs once

We want to use one connection to handle multiple requests

How to do that?
First Solution

Activity creates SocketChannel (or MarsClient in example)
   But do not open the SocketChannel

Pass the SocketChannel to each AsyncTask

Make sure that only one AsyncTask uses channel at a time
   Only one task active a time
   Synchronize critical section
Second Solution

Requires some background
Java Safety - Synchronize

A call to a synchronized method locks the object
   Object remains locked until synchronized method is done

Any other thread's call to any synchronized method on the same object
   will block until the object is unlocked
class SynchronizeExample {
    int[] data;

    public String toString() {
        return "array length " + data.length + " array values " + data[0];
    }

    public synchronized void initialize( int size, int startValue) {
        data = new int[size];
        for ( int index = 0; index < size; index++ )
            data[index] = (int) Math.sin(index * startValue);
    }

    public void unSafeSetValue( int newValue) {
        for ( int index = 0; index < data.length; index++ )
            data[index] = (int) Math.sin(index * newValue);
    }

    public synchronized void safeSetValue( int newValue) {
        for ( int index = 0; index < data.length; index++ )
            data[index] = (int) Math.sin(index * newValue);
    }
}
class SynchronizeExample {
    int[] data;

    public String toString() {
        return "array length " + data.length + " array values " + data[0];
    }

    public synchronized void initialize(int size, int startValue) {
        data = new int[size];
        for (int index = 0; index < size; index++)
            data[index] = (int) Math.sin(index * startValue);
    }

    public void unsafeSetValue(int newValue) {
        for (int index = 0; index < data.length; index++)
            data[index] = (int) Math.sin(index * newValue);
    }

    public synchronized void safeSetValue(int newValue) {
        for (int index = 0; index < data.length; index++)
            data[index] = (int) Math.sin(index * newValue);
    }
}

Synchronized Static Methods

Locks class

Blocks other synchronized class methods
Synchronized Statements

synchronized ( expression ) {
    statements
}

expression must evaluate to an object
That object is locked

class LockTest {
    public synchronized void enter() {
        System.out.println( "In enter" );
    }
}

class LockTest {
    public void enter() {
        synchronized ( this ) {
            System.out.println( "In enter" );
        }
    }
}
public class LockExample extends Thread {
    private Lock myLock;

    public LockExample( Lock aLock ) {
        myLock = aLock;
    }

    public void run() {
        System.out.println( "Start run" );
        myLock.enter();
        System.out.println( "End run" );
    }

    public static void main( String args[] ) throws Exception {
        Lock aLock = new Lock();
        LockExample tester = new LockExample( aLock );

        synchronized ( aLock ) {
            System.out.println( "In Block" );
            tester.start();
            System.out.println( "Before sleep" );
            Thread.currentThread().sleep( 5000 );
            System.out.println( "End Block" );
        }
    }
}

class Lock {
    public synchronized void enter() {
        System.out.println( "In enter" );
    }
}

Output
In Block
Start run
Before sleep
End Block
In enter
End run (why is this at the end?)
Synchronized and Inheritance

class Top {
    public void synchronized left() {
        // do stuff
    }

    public void synchronized right() {
        // do stuff
    }
}

class Bottom extends Top {
    public void left() {
        // not synchronized
    }

    public void right() {
        // do stuff not synchronized
        super.right(); // synchronized here
        // do stuff not synchronized
    }
}

Methods do not inherit synchronized
How to use to allow multiple AsyncTasks?

Make sure all access to server is in one synchronized block

In MarsClient example just need to synchronize MarsClient methods
   As only need to call one method
wait and notify

public final void wait(timeout) throws InterruptedException
public final void wait(timeout, nanos) throws InterruptedException
public final void wait() throws InterruptedException
   Causes a thread to wait until it is notified or the specified timeout expires.

   Throws: IllegalMonitorStateException
   If the current thread is not the owner of the Object's monitor.

   Throws: InterruptedException
   Another thread has interrupted this thread.

public final void notify()
public final void notifyAll()
   Notifies threads waiting for a condition to change.
**wait - How to use**

The thread waiting for a condition should look like:

```java
synchronized void waitingMethod()
{
    while ( ! condition )
        wait();

    Now do what you need to do when condition is true
}
```

Everything is executed in a synchronized method

The test condition is in loop not in an if statement

The wait suspends the thread it atomically releases the lock on the object
synchronized void changeMethod()
{
    Change some value used in a condition test

    notify();
}

wait and notify Example

When can Consumer read from queue?
Blocking Queue Basic operations

put
Adds an element to the end of the queue
If needed blocks until there is room
After adding element call notify

take
Removes element from queue
Only one thread at a time can remove
If queue is empty
   Calling thread blocks (ie take uses wait())
   When another thread add element (ie calls notify)
      Calling thread can proceed
import java.util.concurrent.*;

public class Producer extends Thread {
    BlockingQueue<String> factory;
    int workSpeed;

    public Producer( String name, BlockingQueue<String> output, int speed ) {
        setName(name);
        factory = output;
        workSpeed = speed;
    }

    public void run() {
        try {
            int product = 0;
            while (true) {
                System.out.println( getName() + " produced " + product);
                factory.put( getName() + String.valueOf( product) );
                product++;
                sleep( workSpeed);
            }
        } catch ( InterruptedException workedToDeath ) {
            return;
        }
    }
}
import java.util.concurrent.*;

class Consumer extends Thread {
    BlockingQueue<String> localMall;
    int sleepDuration;

    public Consumer( String name, BlockingQueue<String> input, int speed ) {
        setName(name);
        localMall = input;
        sleepDuration = speed;
    }

    public void run() {
        try {
            try {
                while (true) {
                    System.out.println( getName() + " got " + localMall.take());
                    sleep( sleepDuration );
                }
            }
            catch ( InterruptedException endOfCreditCard ) {
                return;
            }
        }
    }
}
import java.util.concurrent.*;

public class ProducerConsumerExample {
    public static void main( String args[] ) throws Exception {
        BlockingQueue<String> wallmart = new ArrayBlockingQueue(100, true);
        Producer nike = new Producer( "Nike", wallmart, 500 );
        Producer honda = new Producer( "Honda", wallmart, 1200 );
        Consumer valleyGirl = new Consumer( "Sue", wallmart, 400);
        Consumer valleyBoy = new Consumer( "Bob", wallmart, 900);
        Consumer dink = new Consumer( "Sam", wallmart, 2200);
        nike.start();
        honda.start();
        valleyGirl.start();
        valleyBoy.start();
        dink.start();
    }
}

wait and notify - Driver Program
Java Blocking Queues

ArrayBlockingQueue
DelayQueue
LinkedBlockingQueue
PriorityBlockingQueue
SynchronousQueue
How to use Blocking Queues in AsyncTask

AsyncTasks are designed to handle only one request

So need to use Thread
class Consumer extends Thread {
    BlockingQueue<String> requests;

    public Consumer( BlockingQueue<String> input ) {
        requests = input;
    }

    public void run() {
        try {
            while (true) {
                Request current = (Request) requests.take();
                connect to server, get response
                Send answer back to Activity
            }
        } catch ( InterruptedException endOfCreditCard ) {
            return;
        }
    }
}
How to Send Request to Thread?

Easier if thread just does the same thing for each request

Need to do more work if thread handle different requests
From Parsing Lecture

InputStream rawIn = connection.getInputStream();
SDwitterReader in = new SDwitterReader(rawIn);
Message answer = in.next();
Message Responsibilities

Hide all message syntax

Read message and convert to object

    TransmitMessage message =
    TransmitMessage.from("transmitMessage:duh;now what;;");

Create message from values

    TransmitMessage message = new TransmitMessage("duh;now what");

Convert object to required protocol string

    message.toString()  // returns "transmitMessage:duh;now what;;"

Access information about message

    message.isLogin();
    message.name();
Put Message objects in Queue

class Consumer extends Thread {
    BlockingQueue<String> requests;

    public Consumer( BlockingQueue<String> input ) { requests = input; }

    public void run() {
        try {
            while (true) {
                Message current = (Message) requests.take();
                if (!interrupted()) {
                    send(current.toString());
                    response = readServerResponse();
                    Send answer back to Activity
                }
            }
        } catch ( InterruptedException e ) {
            return;
        }
    }
}

Thursday, October 11, 12
How to send Response back to Activity

Thread can not change UI widgets

Thread can use following to have code run on UI thread

Activity.runOnUiThread(Runnable)
View.post(Runnable)
View.postDelayed(Runnable, long)
But each Request requires different response

So have each Message object produce the correct Runnable

Runnable is sent to UI thread to update Activity/Views
Multiple AsyncTasks Verses One Thread

Multiple AsyncTasks
  More Direct
  Clear what each AsyncTask is doing
  Need AsyncTask for each operation

One Worker Thread
  More general
  Potentially fewer classes
  Details in Message classes not AsyncTasks
  Harder to understand and debug