How to cancel network request

Client code opens connection to server

Client sends request to server

Client calls read method on a writer/stream connected to socket

Server is slow in responding

User decides they want to cancel the request

How?
Problem - Blocking IO

Read methods in Reader & InputStream

    public int read(char[] cbuf)
        throws IOException

    Reads characters into an array. This method will block until some input is available, an I/O error occurs, or the end of the stream is reached.

Blocking IO

    When you call read() your code has to wait until read returns

So how to cancel the request???
Inputstream - available

available()

Returns number of bytes you can read without blocking
Common Bad Idea - Polling

have flag userCanceledRequest
Set flag when user cancels request

```java
while (in.available() == 0 ) {
    if (userCanceledRequest) return;
}

in.read(buffer);
```
Why Polling is bad

Tight spin loop consumes all CPU cycles available

while (in.available() == 0 ) {
    if (userCanceledRequest) return;
}

Putting a delay in the loop can reduce the CPU consumption, but there are better ideas
Second idea - Use a Thread

Put your read inside a thread

When user wants to cancel interrupt/kill thread
About Threads
Processes verses Threads

Processes (Heavy Weight)
- Child process gets a copy of parent’s variables
- Relatively expensive to start
- No concurrent access to variables

Thread (Light Weight Process)
- Child process shares parents variables
- Relatively cheap to start
- Concurrent access to variables is an issue
Thread Topics

Creating & Running Threads

Thread Scheduling

Deamon Threads

yield, sleep, join, interrupt

Deprecated methods - suspend, resume, stop, destroy

wait, notify (covered later)
class ExtendingThreadExample extends Thread {
    public void run() {
        for (int count = 0; count < 4; count++)
            System.out.println( "Message " + count + " From: Mom" );
    }

    public static void main( String[] args ) {
        ExtendingThreadExample parallel =
            new ExtendingThreadExample();
        System.out.println( "Create the thread" );
        parallel.start();
        System.out.println( "Started the thread " + parallel.getId() );
        System.out.println( "End" );
    }
}
class SecondMethod implements Runnable {
    public void run() {
        for (int count = 0; count < 4; count++)
            System.out.println("Message " + count + " From: Dad");
    }

    public static void main(String[] args) {
        SecondMethod notAThread = new SecondMethod();
        Thread parallel = new Thread(notAThread);
        System.out.println("Create the thread");
        parallel.start();
        System.out.println("Started the thread");
        System.out.println("End");
    }
}
public class WithNames implements Runnable {
    public void run() {
        for (int count = 0; count < 2; count++)
            System.out.println("Message " + count + " From: " + Thread.currentThread().getName());
    }

    public static void main( String[] args ) {
        Thread a = new Thread(new WithNames(), "Mom");
        Thread b = new Thread(new WithNames(), "Dad");

        System.out.println("Create the thread");
        a.start();
        b.start();
        System.out.println("End");
    }
}
Threads Run Once

Can't restart a thread

```java
public class RunOnceExample extends Thread {
    public void run() {
        System.out.println( "I ran" );
    }

    public static void main( String args[] ) throws Exception {
        RunOnceExample onceOnly = new RunOnceExample();
        onceOnly.setPriority( 6 );
        onceOnly.start();

        System.out.println( "Try restart" );
        onceOnly.start(); // Causes Exception

        System.out.println( "The End" );
    }
}
```

When the run method of a thread ends the thread is dead.
public class SimpleThread extends Thread {
    private int maxCount = 32;

    public SimpleThread( String name ) {
        super( name );
    }

    public SimpleThread( String name, int repetitions ) {
        super( name );
        maxCount = repetitions;
    }

    public SimpleThread( int repetitions ) {
        maxCount = repetitions;
    }

    public void run() {
        for ( int count = 0; count < maxCount; count++ ) {
            System.out.println( count + " From: " + getName() );
        }
    }
}

For Future Examples
Some Parallelism

public class RunSimpleThread {
    public static void main( String[] args ) {
        SimpleThread first = new SimpleThread( 5 );
        SimpleThread second = new SimpleThread( 5 );
        first.start();
        second.start();
        System.out.println( "End" );
    }
}

Output On Rohan
End
0 From: Thread-0
1 From: Thread-0
2 From: Thread-0
0 From: Thread-1
1 From: Thread-1
2 From: Thread-1
3 From: Thread-0
3 From: Thread-1
4 From: Thread-0
4 From: Thread-1

Java on a Solaris machine with multiple processors can run threads on different processors
Thread Scheduling

Priorities

Time-slicing
Priorities

Each thread has a priority

If there are two or more active threads
    If one has higher priority than others
    The higher priority thread is run until it is done or not active

Java Thread Priorities

<table>
<thead>
<tr>
<th>java.lang.Thread field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread.MAX_PRIORITY</td>
<td>10</td>
</tr>
<tr>
<td>Thread.NORM_PRIORITY</td>
<td>5</td>
</tr>
<tr>
<td>Thread.MIN_PRIORITY</td>
<td>0</td>
</tr>
</tbody>
</table>
Java Priority

public class PriorityExample {
    public static void main( String[] args ) {
        SimpleThread first = new SimpleThread( 5 );
        SimpleThread second = new SimpleThread( 5 );
        second.setPriority( 8 );
        first.start();
        second.start();
        System.out.println( "End" );
    }
}

On Single Processor
0 From: Thread-5
1 From: Thread-5
2 From: Thread-5
3 From: Thread-5
4 From: Thread-5
0 From: Thread-4
1 From: Thread-4
2 From: Thread-4
3 From: Thread-4
4 From: Thread-4
End

If you do not set the priority of a thread it has the same priority as the thread it was created in.
Time-Slicing

A thread is run for a short time slice and suspended, 
It resumes only when it gets its next "turn"

Threads of the same priority share turns

Non time-sliced threads run until:
  They end
  They are terminated
  They are interrupted
  Higher priority threads interrupts lower priority threads
  They go to sleep
  They block on some call
  Reading a socket
  Waiting for another thread

Java spec allows time-sliced or non-time-sliced threads
Testing for Time-slicing

If time-sliced output will be mixed

public class InfinityThread extends Thread
{
    public void run()
    {
        while ( true )
            System.out.println( "From: " + getName() );
    }

    public static void main( String[] args )
    {
        InfinityThread first = new InfinityThread( );
        InfinityThread second = new InfinityThread( );
        first.start();
        second.start();
    }
}
Java user & daemon Threads

Daemon thread
   Expendable
   When all user threads are done
       the program ends
       all daemon threads are stopped

User thread
   Not expendable
   Execute until
       Their run method ends or
       An exception propagates beyond the run method.
When a Java Program Ends

Runtime.exit(int) has been called and the security manager permits the exit operation to take place.

or

Only daemon threads are running
Daemon Example

public class DaemonExample extends Thread {
    public static void main(String args[]) {
        DaemonExample shortLived = new DaemonExample();
        shortLived.setDaemon(true);
        shortLived.start();
        System.out.println("Bye");
    }

    public void run() {
        while (true) {
            System.out.println("From: " + getName());
            System.out.flush();
        }
    }
}

Output

From: Thread-0 (Repeated many times)
Bye
From: Thread-0 (Repeated some more, then the program ends)
Thread States

Executing

Only one thread per processor can be running at a time

Runnable

A thread is ready to run but is not currently running

Not Runnable

A thread that is suspended or waiting for a resource
Yield

Allow another thread of the same priority to run
Thread is still runnable

```java
public class YieldThread extends Thread {
    public void run() {
        for (int count = 0; count < 4; count++) {
            System.out.println(count + " From: " + getName());
            yield();
        }
    }
}

public static void main(String[] args) {
    YieldThread first = new YieldThread();
    YieldThread second = new YieldThread();
    first.setPriority(1);
    second.setPriority(1);
    first.start();
    second.start();
    System.out.println("End");
}
```

Output (Explain this)
0 From: Thread-0
0 From: Thread-1
1 From: Thread-0
1 From: Thread-1
2 From: Thread-0
2 From: Thread-1
3 From: Thread-0
End
3 From: Thread-1
Java sleep
Put calling thread in not-runnable state for specified milliseconds

```java
public class NiceThread extends Thread {
    public void run() {
        try {
            System.out.println( "Thread started" );
            sleep( 5 );
            System.out.println( "From: " + getName() );
            System.out.println( "Clean up operations" );
        } catch ( InterruptedException interrupted ) {
            System.out.println( "In catch" );
        }
    }
    public static void main( String args[] ) {
        NiceThread missManners = new NiceThread();
        missManners.start();
        System.out.println( "Main after start" );
    }
}
```

**Output**
- Thread started
- Main after start
- From: Thread-0
- Clean up operations
Java sleep

Put **calling** thread in not-runnable state for specified milliseconds

```java
public class NiceThread extends Thread {
    public void run() {
        System.out.println( "Thread started" );
        System.out.println( "From: " + getName() );
        System.out.println( "Clean up operations" );
    }

    public static void main( String args[] ) throws InterruptedException {
        NiceThread missManners = new NiceThread( );
        missManners.start();
        missManners.sleep(50);  //Who is sleeping
        System.out.println( "Main after start" );
    }
}
```

**Output**
Thread started
From: Thread-0
Clean up operations
Main after start
Java deprecated Thread methods

The following Thread methods are not thread safe

suspend
resume
stop
destroy
The following program does not end
The interrupt just sets the interrupt flag!

```java
public class NoInterruptThread extends Thread {
    public void run() {
        while (true) {
            System.out.println( "From: " + getName() );
        }
    }
}

public static void main(String args[]) throws InterruptedException{
    NoInterruptThread focused  = new NoInterruptThread();
    focused.setPriority( 2 );
    focused.start();
    Thread.currentThread().sleep( 5 ); // Let other thread run
    focused.interrupt();
    System.out.println( "End of main" );
}
```

**Output**

From: Thread-0  (repeated many times)
End of main
From: Thread-0  (repeated until program is killed)
Using Thread.interrupted

```java
public class RepeatableNiceThread extends Thread {
    public void run() {
        while ( true ) {
            while ( !Thread.interrupted() )
                System.out.println( "From: " + getName() );

            System.out.println( "Clean up operations" );
        }
    }

    public static void main(String args[]) throws InterruptedException{
        RepeatableNiceThread  missManners  =
            new RepeatableNiceThread( );
        missManners.setPriority( 2 );
        missManners.start();
        Thread.currentThread().sleep( 5 );
        missManners.interrupt();
    }
}
```

**Output**

```
From: Thread-0
Clean up operations
From: Thread-0
From: Thread-0 (repeated)
```
public class NiceThread extends Thread {
    public void run() {
        try {
            System.out.println( "Thread started" );
            while ( !isInterrupted() ) {
                sleep( 5 );
                System.out.println( "From: " + getName() );
            }
            System.out.println( "Clean up operations" );
        } catch ( InterruptedException interrupted ) {
            System.out.println( "In catch" );
        }
    }
}

public static void main( String args[] ) {
    NiceThread missManners = new NiceThread( );
    missManners.setPriority( 6 );
    missManners.start();
    missManners.interrupt();
}

Output
Thread started
From: Thread-0
From: Thread-0
In catch
Java interrupt ()

Sent to a thread to interrupt it

If thread is blocked on a call to wait, join or sleep
   InterruptedException is thrown &
   The interrupted status flag is cleared

if the thread is blocked on I/O operation on an interruptible channel (NIO)
   ClosedByInterruptException is thrown
   The interrupted status flag is set

If the thread is blocked by a selector (NIO)
   Interrupt status is set
   The thread returns from the selector call as normal

If none of the other conditions hold then the thread’s interrupt status is set
Details

If thread is blocked on a call to wait, join or sleep
  InterruptedException is thrown &
  The interrupted status flag is cleared

if the thread is blocked on I/O operation on an interruptible channel (NIO)
  ClosedByInterruptException is thrown
  The interrupted status flag is set

If the thread is blocked by a selector (NIO)
  Interrupt status is set
  The thread returns from the selector call as normal

If none of the other conditions hold then the thread’s interrupt status is set
Interrupt and Pre JDK 1.4 NIO operations

If a thread is blocked on a read/write to a:
   Stream
   Reader/Writer
   Pre-JDK 1.4 style socket read/write

The interrupt does not interrupt the read/write operation!

The threads interrupt flag is set

Until the IO is complete the interrupt has no effect

This is one motivation for the NIO package
Example

public class SomeClientThread extends Thread {
    private Socket connection;

    public SomeClientThread(Socket toServer) {
        connection = toServer;
    }

    public run() {
        InputStream rawIn = connection.getInputStream();
        BufferedReader in = new BufferedReader(new InputStreamReader(rawIn));
        while ( !isInterrupted() ) {
            String answer = in.readLine();
            process input here
        }
        in.close();
    }
}

If code is blocked in the readLine() then interrupting the thread has no effect until readLine is done.
In short

Using stream IO there is no safe way to always cancel a request to the server

You have to use NIO
NIO - New IO

Supports
  Blocking I/O
  Non-blocking I/O

Buffers
  For data of primitive types
Character set encoders and decoders
A pattern-matching facility based on Perl-style regular expressions

Channels
  Interruptible I/O
  Blocking & non-blocking I/O
A file interface that supports locks and memory mapping of files
A multiplexed, non-blocking I/O facility for writing scalable servers
Channels (java.nio.channels)

Open connection to an entity such as
  hardware device
  file
  network socket
  program component

that is capable of performing I/O operations
Buffer (java.nio)

Buffers for different types
   ByteBuffer
   CharBuffer
   DoubleBuffer
   FloatBuffer
   IntBuffer
   LongBuffer
   MappedByteBuffer
   ShortBuffer
What is new - nio Buffers

One reads from and writes to nio buffers

nio Buffers have
  capacity
    Maximum elements buffer can hold
  limit
    Last position in buffer that can hold data
  In ByteBuffers limit start out = capacity
  position
    Current position in buffer
  reads and writes start a position
  In ByteBuffers position starts out = 0
  mark

array holding the actual data (usually)
Basic nio Buffer operations - ByteBuffer

put(byte)
put(byte[])  
putChar(char)  
...
  writes to buffer
  Write starts at position
  Moves position location after last byte written
  Exception if not enough room in buffer

get()  
get(byte[])  
getChar()  
  Reads from position up to limit
  Moves position to location after last byte read
Basic nio Buffer operations

flip()
   Sets limit to position
   Set position to zero
   After writing to a buffer call flip to read contents

clear()
   Sets limit to capacity
   Set position to zero
   Call clear() when you want to reuse a buffer, need to write first

rewind()
   Sets position to zero
   limit is not changed
   Call when you want to reread buffer
SocketChannel Important methods

open()
close()
connect(SocketAddress)
configureBlocking(boolean)
    True means reads block until there is data to return

read(ByteBuffer)
    Returns number of byte read or -1 if at end of stream

write(ByteBuffer)
    Returns number of byte written
Example Writing

SocketChannel sdChatServer = SocketChannel.open();
sdChatServer.configureBlocking(true);
sdChatServer.connect(new InetSocketAddress("bismarck.sdsu.edu", 8009));
ByteBuffer ioBuffer = ByteBuffer.allocate(1024);

try {
    String message = "nickname:foo;;";
    ioBuffer.put(message.getBytes("UTF8"));
    ioBuffer.flip();
    int bytesWritten = sdChatServer.write(ioBuffer);
} catch (IOException e) {
    System.out.println("Socket write error" + e.message());
}
Example Reading

```java
try {
    ioBuffer.clear();
    int numberBytesRead = sdChatServer.read(ioBuffer);

    if (numberBytesRead == -1) {
        sdChatServer.close();
    } else {
        ioBuffer.flip();
        byte[] responseBytes = new byte[numberBytesRead];

        ioBuffer.get(responseBytes, 0, numberBytesRead - 1);
        String response = new String(responseBytes, "UTF8");
        System.out.println(response);
    }
} catch (IOException e) {
    System.out.println("Socket read error");
}
sdChatServer.close();
```

Not that there should be a loop here to make sure that we have read the entire message from the server. The loop is not included as I am trying to keep the example on a few slides as possible
SocketChannel read/write Exceptions

NotYetConnectedException
   If this channel is not yet connected

ClosedChannelException
   If this channel is closed

AsynchronousCloseException
   If another thread closes this channel while reading

ClosedByInterruptException
   If another thread interrupts the current thread while reading is in progress,
   Channel is closed and setting the current thread's interrupt status

IOException
   If some other I/O error occurs
So Using NIO we can stop a Read request

Threads

- Put code in a thread
- When user want to cancel operation call interrupt() on thread object
- Thread has to check interrupted() calls
- NIO blocking reads/writes will end with exception

AsyncTask

- Put code in a doInBackground()
- When user want to cancel operation call cancel(true) on asyncTask object
- doInBackground() has to check isCancelled()
- NIO blocking reads/writes will end with exception
**MarsClient using NIO**

public class MarsClient {
    SocketChannel serverConnection;
    InetSocketAddress serverAddress;
    ByteBuffer ioBuffer;

    public MarsClient(String serverHost, int port) {
        serverAddress = new InetSocketAddress(
            serverHost, port);
        ioBuffer = ByteBuffer.allocate(1024);
    }
}
Sending

private void send(String message) throws IOException {
    if (serverConnection == null) {
        connect();
    }
    ioBuffer.clear();
    ioBuffer.put(message.getBytes("UTF8"));
    ioBuffer.flip();
    int bytesSent = serverConnection.write(ioBuffer);
    while (bytesSent < message.getBytes("UTF8").length)
        bytesSent += serverConnection.write(ioBuffer);
}

private void connect() throws IOException {
    serverConnection = SocketChannel.open(serverAddress);
    
}
private String readResponse() throws UnsupportedEncodingException, IOException {
    String response = "";
    ioBuffer.clear();
    int bytesRead;
    while ((bytesRead = serverConnection.read(ioBuffer)) != -1) {
        ioBuffer.flip();
        byte[] responseBytes = new byte[bytesRead];
        ioBuffer.get(responseBytes, 0, bytesRead);
        ioBuffer.clear();
        response += new String(responseBytes, "UTF8");
        if (response.contains(";;"))
            return response;
    }
    if (bytesRead == 0) serverConnection.close();
    return response;
}
private Hashtable<String, Float> parseToKeyValues(String message) throws IOException {
    Hashtable<String, Float> keyValues = new Hashtable<String, Float>();
    UpToReader parser = new UpToReader(new StringReader(message));
    for (int k = 1; k <= 2; k++) {
        String key = parser.upto(':'_consts::Ends
        String value = parser.upto(';'_consts::Ends
        keyValues.put(key, new Float(value));
    }
    return keyValues;
}
public static String tripMessage(int people, float weight, float mpg, float malesPerYear) {
    return "trip;destination:mars;people:" + people + ";weight:" + weight
        + ";mpg:" + mpg + ";milesperyear:" + malesPerYear + ";;";
}

public Hashtable<String, Float> trip(int people, float weight, float mpg, float milesPerYear) throws IOException{
    send(tripMessage(people,weight,mpg,milesPerYear));
    String response = readResponse();
    return parseToKeyValues(response);
}