Threads and concurrency in Java.

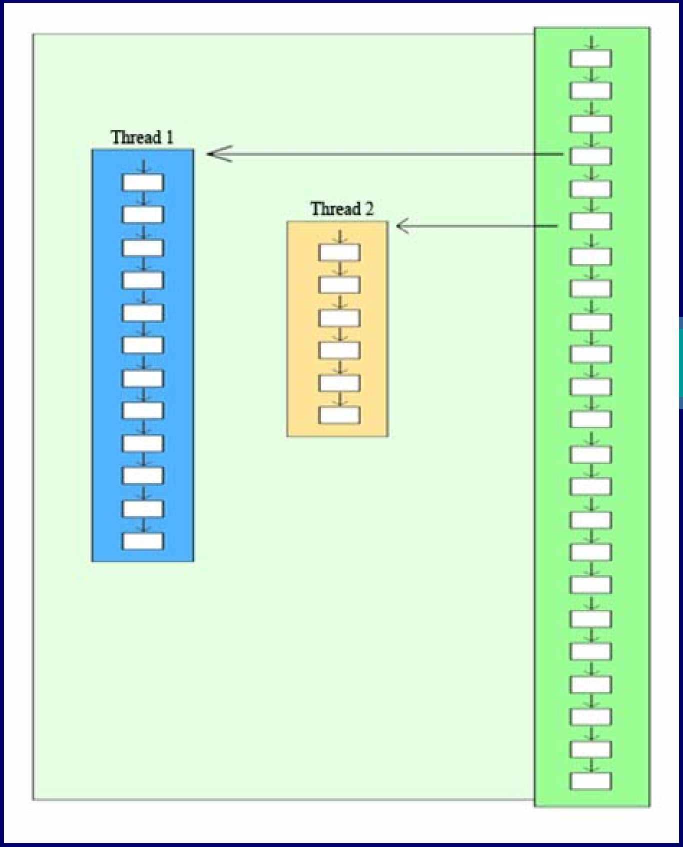
Martin Jarnes Olsen

This presentation

- Threads in the Java 1.5 API.
- Implementing threads.
- Controlling the thread.
- The life of a thread.
- Thread synchronization.
- Thread priority.

What is a thread?

- Def: Sequential flow of control within a program.
- Executes single instructions in a sequential order.
- Can run multiple threads at the same time.
- Runs within the same process.



Threads in the API

- High level – specific tasks
  - java.util.Timer
  - javax.swing.Timer
- Low level - implementing your own threads
  - java.lang.Thread
  - java.lang.Runnable

Implementing java.util.Timer

```
Timer t = new Timer();
t.schedule(new Clock(), 0, 1000);
:
class Clock extends TimerTask {
        public void run() {
            //will be executed each timeinterval
        }
    }
```

Implementing your own threads

- Two ways
  - Subclassing java.lang.Thread.
  - Why use Thread?
    - A class can only extend one class at a time.
    - If don’t need to extend other classes.
  - Implementing the Runnable interface.
  - Why use Runnable?
    - A class can implement multiple interfaces.
    - If need to extend other classes.

Subclassing java.lang.Thread

```
class ThreadExample extends Thread {
    public void run() {
        :
    }
}
:
ThreadExample te = new ThreadExample();
te.start();
```

Implementing java.lang.Runnable

```
class ThreadExample implements Runnable {
    public void run() {
        :
    }
}
:
Thread t = new Thread (new ThreadExample());
t.start();
```

Controlling the thread.

- start() automatically calls run()
- If decired task is repetative, use while loop inside run().
- Use a condition in the loop that is controllable from outside.
- Control speed/intensity of thread by using sleep or wait.

```
boolean running;
public void run() {
        running = true;
        While(running) {
            Try{ sleep(1000); } catch(Exception e) {}
                :
        }
}
Public void stopThread(boolean b) {
        running = b;
}
```

The life of a thread

- Different states (new to 5.0):
  - NEW – Before start() has been called.
  - RUNNABLE – After start() has been called.
  - WAITING – when calling wait().
  - TIMED_WAITING – when calling sleep().
  - TERMINATED – after run() is finished.
- getState() method.

Producer/consumer

- Consider the famous producer/consumer problem:
  - Two threads have access to the same stack.
  - One produces, one consumer.
  - Problem: Controlling the order of consummation and production.
  - Solution: Limiting the access to one thread at a time.

Thread synchronization

```
public synchronized void produce() {
      while(!producing)
          try{ wait(); } catch(Exception e) {}
      number++;
      System.out.println("Producing: " +number);
      producing = false;
      notifyAll();
   }
      :
public synchronized void consume() {
      while(producing)
          try{ wait(); } catch(Exception e) {}
      System.out.println("Consuming: " +number);
      producing = true;
      notifyAll();
   }
```

Thread priority

- Can be access with getPriority() and setPriority().
- A number between 1 and 10. (1 low, 10 high)
- 5 is default.
- Lowering the priority is not a smart way to schedule threads, as lower priority threads will simply not run most of the time.

Questions?