Long-Running Tasks

Long Tasks

- What should you do when a task will take significant time?
  - Fetching a large image or long list over a (slow) internet connection
  - Factoring a large number
  - Reading a large file
  - Searching a directory structure
  - etc
- Demo of what *not* to do...

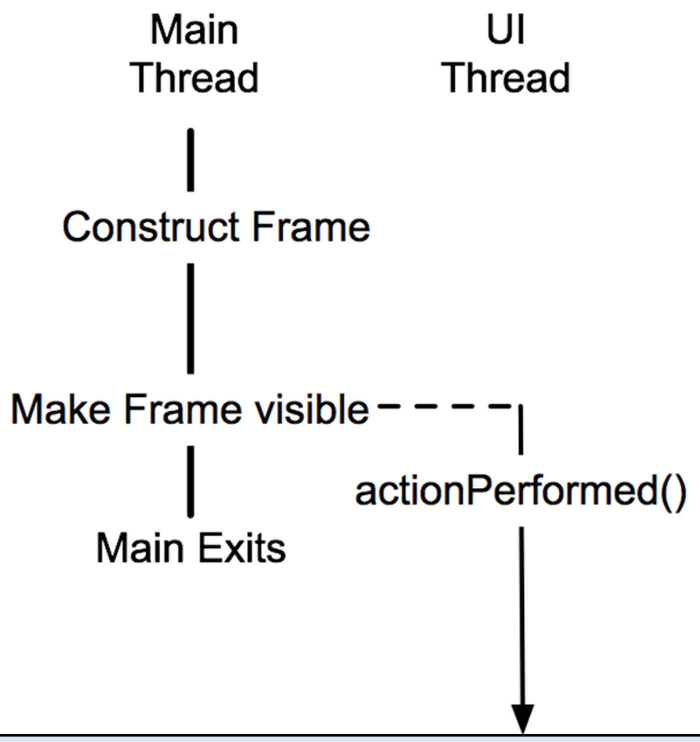
What is wrong?

```
protected void registerControllers() {
        // Handle presses of the start button
        this.startStop.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        model.calculatePrimes();
                }
        });
```

Find primes in [1, 250000]
Takes ~10 seconds to complete

What went wrong?

- Like your X program, Java has a single dispatch thread.
- Almost all of the processing happens in response to events.
- As long as each piece takes just a little time, this is OK.
- But… we clicked "Start" and we started to find the primes in the same thread that handles the events.



A long task causes subsequent events to queue up behind it and the interface becomes unresponsive.

Handling Long Tasks

- Goal is to maintain a highly interactive application
- Providing feedback, maintaining responsiveness keeps users happy
  - Even if it takes longer to complete the task
- In particular, it is usually a good idea to provide users with affordances to pause or cancel tasks, and to view progress

Strategies for Long Tasks

- Option 1. Run in the Event Dispatch Thread
  - Break the task into smaller subtasks
  - Periodically execute each subtask on the Event Dispatch thread (between handling regular events)
- Option 2. Run in a Separate Thread
  - Execute the long-running method on a separate thread
- BUT
  - we would like to be able to pause/cancel the tasks and report progress
  - should periodically check for cancellation, and report progress

Common Functionality

- Regardless of specific approach, should provide methods to execute task, cancel task, check whether task was completed successfully, and query progress:
  - run()
  - cancel()
  - isDone()
  - wasCancelled()
  - progress()

Option 1 - Subtasks on Event Thread

- Task object keeps track of current task progress
- Subtasks periodically called on Swing event thread
  - See SwingUtilities.invokeLater() for way to execute on Swing thread
  - Alternatively, see javax.swing.Timer
    (there is also java.util.Timer; use the Swing version)
- Every time object told to “run,” it checks current progress, executes subtask, updates progress, yields

```
class FindPrimesMP extends AbstractPrimesModel {
        private boolean cancelled = false;
        private boolean running = false;
        private int current = 0;          // progress so far

        public FindPrimesMP(int min, int max) {  super(min, max);  }

        /* Calculate a some primes in the event thread. If necessary,
        * schedule ourselves to calculate some more a little bit
        * later. */
        public void calculatePrimes() {
                this.running = true;
                SwingUtilities.invokeLater(new Runnable() {
                        public void run() {
                                calculateSomePrimes();
                                if (!cancelled && current <= max) {
                                        calculatePrimes();
                                }
                        }
                });
        }
```

```
/** Calculate some prime numbers. Quit when we run out of
* time or we're cancelled or we've reached the maximum
* prime to look for.  */
private void calculateSomePrimes() {
        long start = System.currentTimeMillis();
        while (true) {
                if (this.current > this.max) {
                        this.running = false;
                        updateAllViews();
                        return;

                } else if (System.currentTimeMillis() - start >= 100) {
                        updateAllViews();
                        return;

                } else if (isPrime(this.current)) {
                        this.addPrime(current);
                }
                current += 1;
        }
}
```

Option 1 - Subtasks on Event Thread

- Advantages:
  - Can more naturally handle “pausing” (stopping/restarting) task because it maintains information on progress of overall task
  - Can be run in Swing event thread or separate thread
  - Useful in single-threaded platforms (e.g., iPhone, iPad, etc.)
- Disadvantages:
  - Tricky to predict length of time for subtasks
  - Not all tasks can easily break down into subtasks (e.g., Blocking I/O)

Option 1 - Subtasks on Event Thread

These are some nasty disadvantages!
It’s better to use threads (Method 2) when possible!

- Disadvantages:
  - Tricky to predict length of time for subtasks
  - Not all tasks can easily break down into subtasks (e.g., Blocking I/O)

Option 2 – Using a Separate Thread

- Long method runs in a separate thread
  – Typically implemented via Runnable object
- Method regularly checks if task should be cancelled
- Demo…

```
class FindPrimesT extends AbstractPrimesModel {
...
        public void calculatePrimes() {
                new Thread() {
                        public void run() {
                                running = true;
                                long start = System.currentTimeMillis();
                                while (true) {
                                  if (cancelled || current > max) {
                                        running = false;
                                        updateSwing();
                                        return;
                                  } else if (isPrime(current)) {
                                        addPrime(current);
                                  }
                                  current += 1;
                                  if (System.currentTimeMillis() - start >= 100) {
                                        updateSwing();
                                        start = System.currentTimeMillis();
                                  }
                        }
                }
                private void updateSwing() {
                        SwingUtilities.invokeLater(new Runnable() {
                                public void run() {
                                        updateAllViews();
                                }
                        });
                }
        }.start();
}
```

Option 2 – Using a Separate Thread

- Advantages:
  - Conceptually, the easiest to implement
  - Takes advantage of multi-core architectures
- Disadvantages:
  - Extra code required to be able to pause/restart method
  - All the usual Thread baggage
    - Race conditions
    - Deadlocks
    - Etc.

Option 2 – Using a Separate Thread

- WARNING: Swing is not thread safe!
- Don’t call Swing methods or access Swing components from outside the Event Dispatch thread
- From task thread, use invokeLater to schedule code to run in the Event Dispatch thread
- Use synchronized keyword to protect critical sections

“synchronized” keyword

- Java’s strategy for handling concurrency is to use the “Monitor” abstraction
  - Conceptually higher level than semaphores and mutexs
  - Overall goal is the same: provide mutually exclusive access to critical sections
- Methods marked with the “synchronized” keyword can only be access by one thread a time.
  - Synchronizing both run() and cancel() methods means cancel() can’t execute until after run() has finished

“synchronized” keyword

```
public class ThreadSafeCounter {
    private int c = 0;

    public synchronized void increment() {
        this.c++;
    }

    public synchronized void decrement() {
         this.c--;
    }

    public synchronized int value() {
        return this.c;
    }
}
```

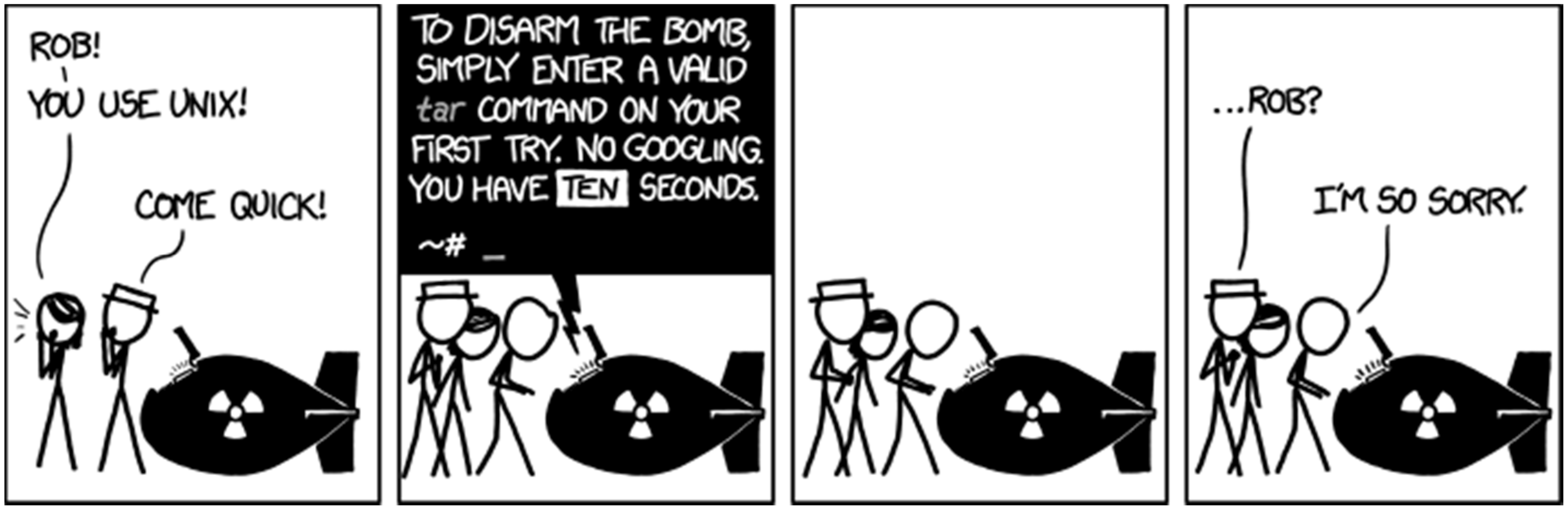
Long Tasks and MVC

- MVC strives to have a complete separation between model and view
- What do you see happening as we break task up?

Long Tasks and MVC

- Long tasks start to break clean separation of MVC
- Model’s methods need to be designed to allow user to stop them, to maintain interactivity
  - Needed to service event queue
  - Needed to allow user to stop method
- May need methods to inquire about length of task completion
  - Not part of “model” – part of interaction
- Usability concerns are thus directly influencing design of model to accommodate user interaction

Up Next: Design


ROB!
YOU USE UNIX!
COME QUICK!
TO DISARM THE BOMB, SIMPLY ENTER A VALID tar COMMAND ON YOUR FIRST TRY. NO GOOGLING. YOU HAVE TEN SECONDS.
~# _
...ROB?
I'M SO SORRY.