Write Multi-Thread Swing Program Right

Huanwen Qu

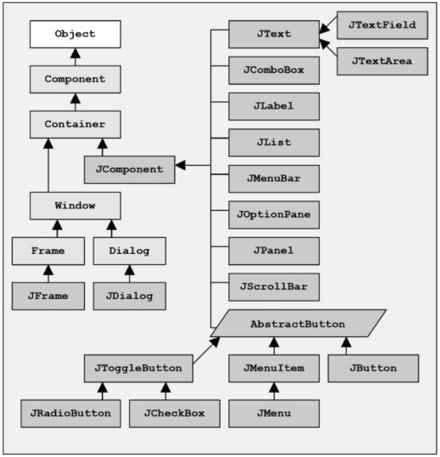
CSTS

April 2, 2009

Outline

1. EDT
   - Swing Architecture
   - EDT
   - Thread-Safe
2. Worker Thread
   - Threads in Swing
   - SwingWorker class
   - Swing Timer
3. Issues
   - Slowness of TEW
   - Window resource release issue
4. Case Study
5. Demo

Swing Class Hierarchy



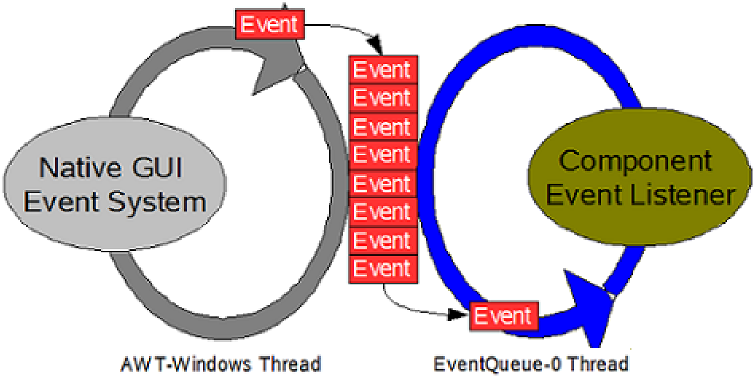
EDT

Event Dispatch Thread

Swing applications have a single EDT for the UI. This thread draws GUI components, updates them, and responds to user interactions.

- AWT-Windows, AWT-Linux, AWT-Solaris
- EventQueue-0

Event Eueue


Event
Event
Event
Event
Event
Event
Event
Event
Event
Native GUI Event System
Component Event Listener
Event
AWT-Windows Thread
EventQueue-0 Thread

Realized

Realized

- A Swing widget is realized when pack(), show() or setVisible(true) is called, and the widget is marked as displayable.
- Call dispose() to release native screen resources and make the widget undisplayable.
- All access to displayable widget must be in EDT except the operation is specified as multithread-safe.

The thread-safe exceptions

The thread-safe exceptions

- Some methods of JComponent : repaint, revalidate, invalidate
- All addXXXListener and removeXXXListener methods
- All methods explicitly documented as thread-safe

Threads in Swing

Swing applications have three types of threads

- An initial thread
- A UI event dispatch thread (EDT)
- Worker threads

A common way to start a Swing Program

A common way to start a Swing Program

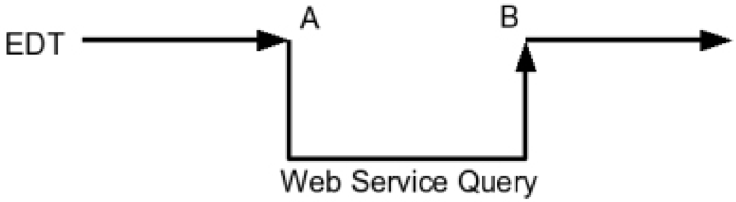
```
public class MainFrame extends javax.swing.JFrame {
  ...

  public static void main(String[] args) {
    new MainFrame().setVisible(true);
  }
}
```

The right way

The right way

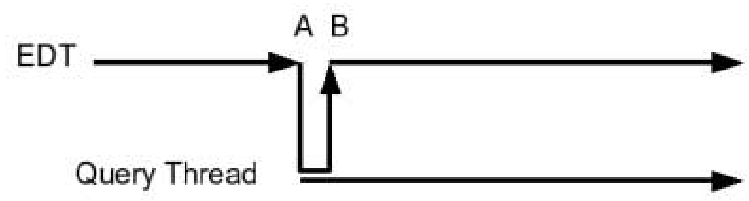
```
public class MainFrame extends javax.swing.JFrame {
  ...

  public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        new MainFrame().setVisible(true);
      }
    });
  }
}
```

Single thread



The EDT cannot process UI events between points A and B.

Two threads



The EDT is able to continue processing UI events without a long delay.

A simple solution

Every invocation on widget's method should be wrapped as:

A simple solution

```
SwingUtilities.invokeLater(new Runnable() {
    public void run() {
        widget.method(...);
      }
    });
```

Problems

Problems of the simple solution

- Inconvinient
- Inflexible
- Ugly
- Hard to manage the worker thread

SwingWorker class

SwingWorker class

```
public abstract class SwingWorker<T,V>
extends Object
implements RunnableFuture
```

- The T type indicates that an implementation's doInBackground and get methods will return values of T type.
- The V type indicates that an implementation's publish and process methods will operate on values of type V.

Main methods

Main methods

- boolean cancel(boolean mayInterruptIfRunning)
- T get()
- T get(long timeout, TimeUnit unit)
- boolean isCancelled()
- boolean isDone()
- protected final void publish(V... chunks)

User implemented methoeds

User implemented methoeds

- protected abstract T doInBackground() throws Exception
- protected void done()
- protected void process(List<V> chunks)

My experience

My experience

- Add **isCancelled()** checkpoints before doing background task, publishing interim results and finishing the task.
- For long time task, publishing the interim results or progress is more user friendly.
- A SwingWorker instance is not reusable.
- Notify the result receivers via event listener.
- User should have the ability to stop the worker thread.

Swing Timer

When to use timer?

- To perform a task once, after a delay.
- To perform a task repeatedly.

java.util.timer vs javax.swing.timer

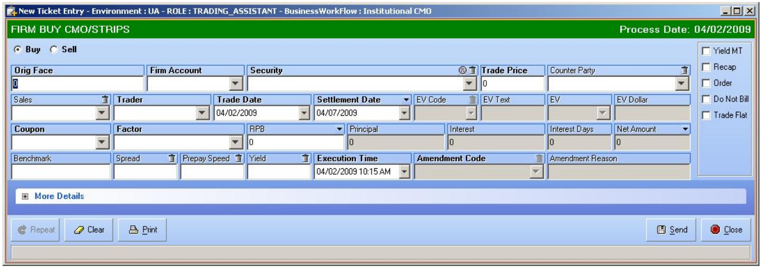
In general, we recommend using Swing timers rather than general-purpose timers for GUI-related tasks because Swing timers all share the same, pre-existing timer thread and the GUI-related task automatically executes on the event-dispatch thread. However, you might use a general-purpose timer if you don't plan on touching the GUI from the timer, or need to perform lengthy processing.

Issues encountered in TM

Issues encountered in TM

- Slowness of TEW creation
- Window resoure release issue

Original TEW


New Ticket Entry - Environment : UA - ROLE : TRADING_ASSISTANT - BusinessWorkFlow : Institutional CMO

FIRM BUY CMO/STRIPS Process Date: 04/02/2009

(● Buy) ○ Sell

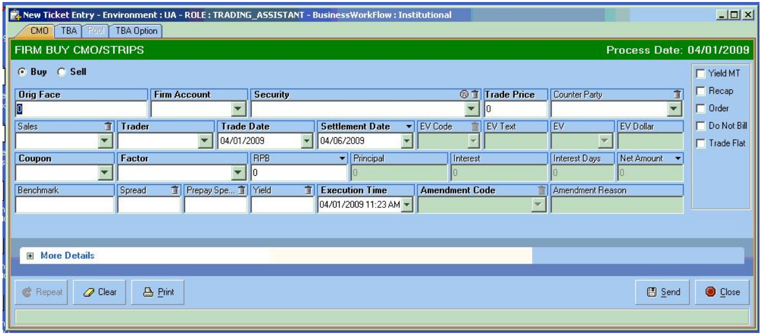
Orig Face	Firm Account	Security	Trade Price	Counter Party	
0			0		
Sales	Trader	Trade Date	Settlement Date	EV Code	EV Text
		04/02/2009	04/07/2009		
Coupon	Factor	RPB	Principal	Interest	Interest Days
		0	0	0	0
Benchmark	Spread	Prepay Speed	Yield	Execution Time	Amendment Code
				04/02/2009 10:15 AM	

EV	EV Dollar	Net Amount	Amendment Reason
		0	

☐ Yield MT
☐ Recap
☐ Order
☐ Do Not Bill
☐ Trade Flat

More Details

New TEW


New Ticket Entry - Environment : UA - ROLE : TRADING_ASSISTANT - BusinessWorkFlow : Institutional

FIRM BUY CMO/STRIPS Process Date: 04/01/2009

[✓ Buy] Sell

Orig Face: 0	Firm Account:	Security:	Trade Price: 0	Counter Party:
Sales:	Trader:	Trade Date: 04/01/2009	Settlement Date: 04/06/2009	EV Code:
EV Text:	EV:	EV Dollar:		
Coupon:	Factor:	RPB: 0	Principal: 0	Interest: 0
Interest Days: 0	Net Amount: 0			
Benchmark:	Spread:	Prepay Spe...:	Yield:	Execution Time: 04/01/2009 11:23 AM
Amendment Code:	Amendment Reason:			

Yield MT, Recap, Order, Do Not Bill, Trade Flat

New frame creation code

```
public TraderTicketingEntryWindow
    newDefaultWindow(TicketMode mode) {
    initTEWCache();
    InstitutionalTraderTicketEntryBasePanel[] panel
        new InstitutionalTraderTicketEntryBasePanel
    panels[3]=new
        InstitutionalTBAOptionTraderTicketEntryPane
    panels[2]=new InstitutionalTBAPoolTraderTicketE
    panels[1]=new InstitutionalTBATraderTicketEntry
    panels[0]=new InstitutionalTraderTicketEntryPan
    populateTEW(panels);
    return instance;
}
```

One improvement solution

One improvement solution

```
Thread t[i] = new Thread(){
    public void run(){
        panel[i] = new xxxx();
    }
}
```

Put the construction functions in four threads. We gained nearly 4 times performance improvement. But. . .

Issue

Sometimes we got the following exception:

```
Exception in thread "New CMO Panel" java.lang.NullP
  at javax.swing.plaf.basic.BasicPopupMenuUI.uninsta
  at javax.swing.plaf.basic.BasicPopupMenuUI.uninsta
  at javax.swing.JPopupMenu.setInvoker(Unknown Sourc
  at javax.swing.JMenu.ensurePopupMenuCreated(Unknow
  at javax.swing.JMenu.getPopupMenu(Unknown Source)
  at com.jidesoft.plaf.vsnet.VsnetMenuUI.installList
  at com.jidesoft.plaf.basic.BasicJideSplitButtonUI.
  at com.jidesoft.plaf.vsnet.VsnetMenuItemUI.install
  at javax.swing.JComponent.setUI(Unknown Source)
  at com.jidesoft.swing.JideSplitButton.updateUI(Unk
  at javax.swing.JMenuItem.init(Unknown Source)
  at javax.swing.JMenuItem.<init>(Unknown Source)
  at javax.swing.JMenuItem.<init>(Unknown Source)
  at javax.swing.JMenu.<init>(Unknown Source)
```

What’s wrong?

What’s wrong?

Swing is not multithread-safe. Dozens of widgets are created in the construction function. Conflicts occur among these initialization threads.
We will look into detail in case study section.

Window resource release issue

Window resource release issue

In order to optimize the GC performance, when TEW frame is closed, the frame is disposed and the sub entry panels will be cleaned. All components in sub panels will be removed from the parents. But when the initialization process is running, exceptions will ocurr due to the worker thread try to set values on removed widgets.

Case Study

Cases

- Create widgets in user threads and reproduce the TM issue.
- Improve the program by using worker thread
- A timer example

A Picasa photo searcher

Soutu

- Search photos via Google Picasa API
- There are two kinds workers:
  - A worker retrieves the search result from Picasa
  - A worker downloads the thumbnails and images according to the search result.
- Follow the MVC principal. There is no directi dependency between UI and Model, Model and Searcher.
- Most operations are asynchronous.
- The UI is smooth. No stuck point.