GUI programming with threads

Threads and Swing

- Swing is *not* generally *thread-safe*: most methods are not synchronized
  - correct synchronization is difficult to implement and increases overheads
  - thread-safe and deadlock-free programming would hinder building new Swing components
- the most important threads in a Swing program
  - the *main* thread
  - the *event dispatch* thread that calls on listeners
  - *another* thread puts events into the event queue, etc.; to find out, run the following code:

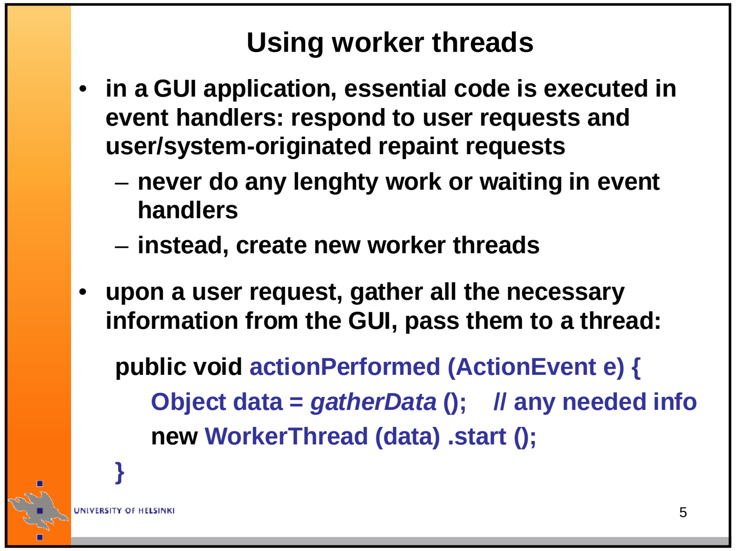
```
ThreadGroup group; group = Thread.currentThread
  ().getThreadGroup (); group.getParent ().list ();
```

The Swing *single-thread rule*

- *single thread rule* for Swing programming:
  - only the event dispatch thread may manipulate GUI components
- the event thread manages the GUI components of a program, executing event handlers (callbacks)
  - the event thread is *created lazily* by GUI methods
  - a Swing component can be freely manipulated until it is made visible as an active part of the visual user interface ("realized")
  - after that, other threads cannot manipulate Swing components (unless otherwise stated by API doc)

The Swing single-thread rule (cont.)

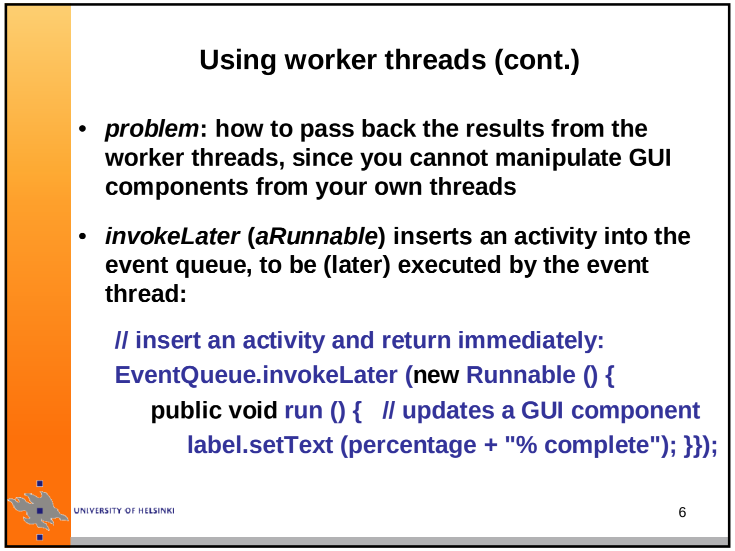
- *some* Swing methods are *thread-safe* :
  - *JTextComponent.setText* (*aString*)
  - *JTextArea*: *insert* (*aString*, *pos*), *append* (*aString*), *replaceRange* (*aString*, *start*, *end*)
  - in *JComponent*:
    - *repaint* () sends repaint into event queue
    - *revalidate* () sends a layout event that updates the position and size of the component
      - AWT-related components, such as *JFrame*, call the AWT *invalidatelvalidate* pair..
  - adding and removing event listeners
- but handler methods are always executed by the event thread, only



Using worker threads

- in a GUI application, essential code is executed in event handlers: respond to user requests and user/system-originated repaint requests
  - never do any lenghty work or waiting in event handlers
  - instead, create new worker threads
- upon a user request, gather all the necessary information from the GUI, pass them to a thread:

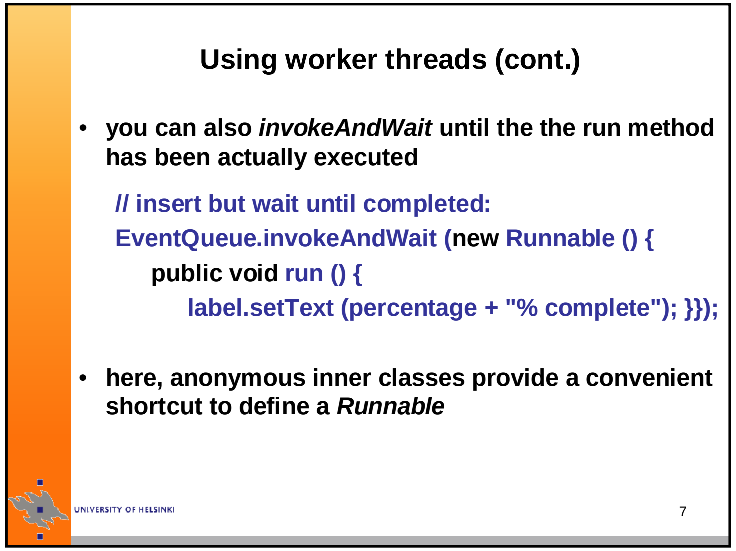
```
public void actionPerformed (ActionEvent e) {
    Object data = gatherData ();    // any needed info
    new WorkerThread (data) .start ();
}
```



Using worker threads (cont.)

- *problem*: how to pass back the results from the worker threads, since you cannot manipulate GUI components from your own threads
- *invokeLater* (*aRunnable*) inserts an activity into the event queue, to be (later) executed by the event thread:

```
// insert an activity and return immediately:
EventQueue.invokeLater (new Runnable () {
    public void run () {   // updates a GUI component
        label.setText (percentage + "% complete"); }});
```

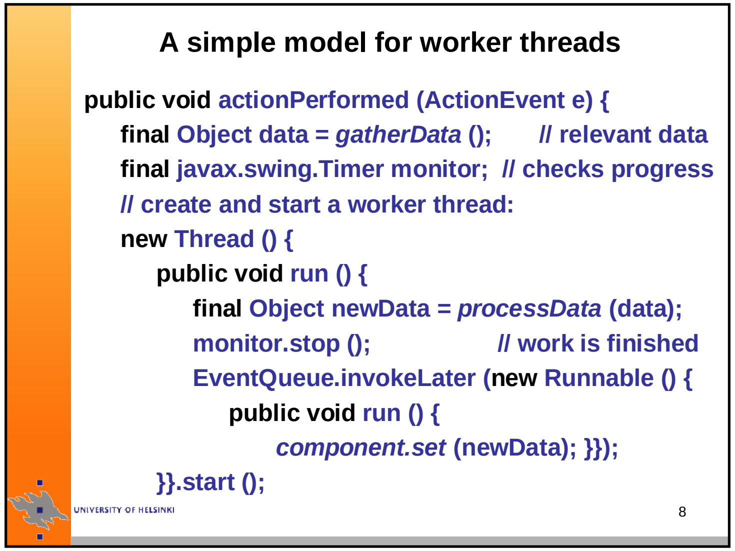


Using worker threads (cont.)

- you can also *invokeAndWait* until the the run method has been actually executed

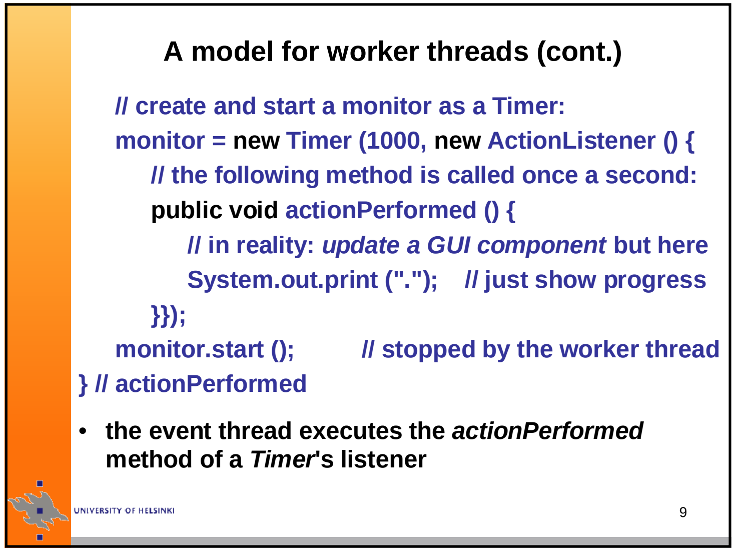
```
// insert but wait until completed:
EventQueue.invokeAndWait (new Runnable () {
    public void run () {
        label.setText (percentage + "% complete"); }});
```

- here, anonymous inner classes provide a convenient shortcut to define a *Runnable*



A simple model for worker threads

```
public void actionPerformed (ActionEvent e) {
    final Object data = gatherData ();      // relevant data
    final javax.swing.Timer monitor;  // checks progress
    // create and start a worker thread:
    new Thread () {
        public void run () {
            final Object newData = processData (data);
            monitor.stop ();                 // work is finished
            EventQueue.invokeLater (new Runnable () {
                public void run () {
                    component.set (newData); }});
        }}.start ();
```



A model for worker threads (cont.)

```
    // create and start a monitor as a Timer:
    monitor = new Timer (1000, new ActionListener () {
        // the following method is called once a second:
        public void actionPerformed () {
            // in reality: update a GUI component but here
            System.out.print (".");    // just show progress
        }});
    monitor.start ();          // stopped by the worker thread
} // actionPerformed
```

- the event thread executes the *actionPerformed* method of a *Timer*'s listener