ANSWERS

Emma Arfelt Kock, ekoc

Anders Fischer, afin

Exercise 7

Exercise 7.1

7.1.1

The documentation at: https://docs.oracle.com/javase/tutorial/uiswing/concurrency/index.html

"The programmer does not need to provide code that explicitly creates these threads: they are provided by the runtime or the Swing framework. The programmer's job is to utilize these threads to create a responsive, maintainable Swing program.

This lesson discusses each of the three kinds of threads in turn. Worker threads require the most discussion because tasks that run on them are created using javax.swing.SwingWorker. This class has many useful features, including communication and coordination between worker thread tasks and the tasks on other threads."

Therefore we should use multiple SwingWorkers and not delegate work using an executor in a single SwingWorker.

Alterations in the code (Class DownloadWorker):

```
public String doInBackground() {
     StringBuilder sb = new StringBuilder();
     int count = 0;
       if (isCancelled())                          // (3)
          return sb.toString();
             System.out.println("Fetching " + url);
       String page = getPage(url, 200);
       String result = String.format("%-40s%7d%n", url, page.length());
       sb.append(result); // (1)
       setProgress((100 * ++count) / urls.length); // (2)
       publish(result); // (4)
       textArea.append(result);
       return sb.toString();
   }
```

and in Class TestFetchWebGui:

```
    // (1) Use a background thread, not the event thread, for work
    List<DownloadWorker> downloadWorkers = new ArrayList<DownloadWorker>
```

```
();
    for (String url : urls) {
      downloadWorkers.add(new DownloadWorker(textArea, url));
    }

    fetchButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          for (DownloadWorker d : downloadWorkers)
          d.execute();
        }});
```

7.1.2

```
 // (3) Enable cancellation
    cancelButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          for (DownloadWorker downloadTask : downloadWorkers) {
            downloadTask.cancel(false);
          }
        }});
```

7.1.3

Alteration in Class TestFetchWebGui (in method goodFetch()):

```
 JProgressBar progressBar = new JProgressBar(0, 100);
    //    progressBar.setValue(50);
    outerPanel.add(progressBar, BorderLayout.SOUTH);

    for (DownloadWorker downloadTask : downloadWorkers) {
      downloadTask.addPropertyChangeListener(new PropertyChangeListener()
{
        public void propertyChange(PropertyChangeEvent e) {
          if ("progress".equals(e.getPropertyName())) {
            int count = (Integer)e.getNewValue();
            progressBar.setValue(100 * count / downloadWorkers.size());
          }}});
    }
```

and

```
private static final AtomicInteger count = new AtomicInteger(1);
...
public String doInBackground() {
    ...
```

```
    setProgress(count.getAndIncrement()); // (2)
    ...
```

7.2

7.2.1

Every lift has it's own thread and every method on the Runnable is synchronized.

7.2.2

Hotel lift with four elevators and floors from -2 to 10. Alterations in code:

```
  public final int lowFloor = -2, highFloor = 10;
```

and main-method:

```
public static void main(String[] args) {
    // The lift model and associated graphics
    final LiftShaft shaft1 = new LiftShaft(),
      shaft2 = new LiftShaft(),
      shaft3 = new LiftShaft(),
      shaft4 = new LiftShaft();
    final Lift lift1 = new Lift("Lift1", shaft1),
      lift2 = new Lift("Lift2", shaft2),
      lift3 = new Lift("Lift3", shaft3),
      lift4 = new Lift("Lift4", shaft4);
    final LiftDisplay lift1Display = new LiftDisplay(lift1, true),
      lift2Display = new LiftDisplay(lift2, false),
      lift3Display = new LiftDisplay(lift3, true),
      lift4Display = new LiftDisplay(lift4, false);
    LiftController controller = new LiftController(lift1, lift2, lift3,
lift4);
    Thread t1 = new Thread(lift1), t2 = new Thread(lift2), t3 = new
Thread(lift3), t4 = new Thread(lift4);
    t1.start(); t2.start(); t3.start(); t4.start();

    // The graphical presentation
    final JFrame frame = new JFrame("TestLiftGui");
    final JPanel framePanel = new JPanel();
    framePanel.setLayout(new BorderLayout());

    final JPanel panel = new JPanel();
    panel.setLayout(new BorderLayout());
    panel.add(lift1Display, BorderLayout.WEST);
    panel.add(lift2Display, BorderLayout.EAST);
    framePanel.add(panel, BorderLayout.WEST);

    framePanel.add(new OutsideLiftButtons(controller),
```

```
BorderLayout.CENTER);

    final JPanel panel2 = new JPanel();
    panel2.setLayout(new BorderLayout());
    panel2.add(lift3Display, BorderLayout.WEST);
    panel2.add(lift4Display, BorderLayout.EAST);
    framePanel.add(panel2, BorderLayout.EAST);

    frame.add(framePanel);
    frame.pack(); frame.setVisible(true);
  }
}
```

7.2.3

We removed the Thread creation in the `main()` method, and replaced it with:

```
    final int rate = 16;
    final ScheduledThreadPoolExecutor scheduler = new
ScheduledThreadPoolExecutor(4);
    scheduler.scheduleAtFixedRate(lift1, rate, rate,
TimeUnit.MILLISECONDS);
    scheduler.scheduleAtFixedRate(lift2, rate, rate,
TimeUnit.MILLISECONDS);
    scheduler.scheduleAtFixedRate(lift3, rate, rate,
TimeUnit.MILLISECONDS);
    scheduler.scheduleAtFixedRate(lift4, rate, rate,
TimeUnit.MILLISECONDS);
```

We also removed the sleep() calls and the related try/catch statements.

7.2.4

We added the function `anyLiftAt(double floor)` to the Controller as seen below.

```
  public boolean anyLiftAt(double floor) {
    return Arrays.stream(lifts).anyMatch(l -> l.getFloor() == floor);
  }
```

Then we added the following to UpDownButtons:

```
  up.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      controller.someLiftTo(atFloor, Direction.Up);
      while (!controller.anyLiftAt(atFloor)) up.setBackground(Color.RED);
    }});
  down.addActionListener(new ActionListener() {
```

```
    public void actionPerformed(ActionEvent e) {
      controller.someLiftTo(atFloor, Direction.Down);
      while (!controller.anyLiftAt(atFloor))
down.setBackground(Color.RED);
    }});
```