Threading Issues

The AWT Events Dispatching

- Swing GUI components rely on AWT support for events dispatching.

  The same well known events dispatching mechanism used in AWT is used in Swing based applications.

Three Threads

- Each time we run a swing application there are three main threads:

  ### The Application Thread

  This is the main thread of our application. It is the thread that runs our application.

  ### The Toolkit Thread

  This thread is responsible for capturing the system events, such as keys pressing and mouse movements. The captured events are sent over to the EDT thread.

Three Threads

The EDT Thread

This thread is in charge of dispatching the events captured by the toolkit thread to the relevant components and for calling the paint method. This is also the thread through which the events are hand held by their listeners.

Avoid Blocking The EDT

- Blocking the EDT for a long period of time will freezes the user interface and the user will think the application runs slowly.

  Sample for cases in which the EDT might be blocked include: reading / writing to files, connecting a database, connecting another computer etc.

- Trying to avoid it by initiating another separated thread that includes (in its end) a call to update the component is a violation to the swings single thread rule.

  Doing it might cause a dead lock.

Swing Single Thread Rule

- The EDT is responsible for executing any method that modifies the component, including the constructors.
- Swing is not “thread-safe” and the common rule states that every Swing related operation should be invoked on the EDT.

Swing Single Thread Rule

- Trying to interact with GUI components via threads other than the EDT might cause a dead lock.

  In most cases we won't experience any dead lock. Yet, the risk exists. This is also the reason for the famous recommendation to avoid creating the GUI components on the main application thread.

invokeLater()

- The SwingUtilities class includes the invokeLater() method that enables us to set a new thread to be called on the EDT.
- Using this method we can place the operations with the GUI components on the EDT.

invokeLater()

```
public static void invokeLater(Runnable doRun)
```

Causes *doRun.run()* to be executed asynchronously on the AWT event dispatching thread. This will happen after all pending AWT events have been processed. This method should be used when an application thread needs to update the GUI. In the following example the invokeLater call queues the Runnable object doHelloWorld on the event dispatching thread and then prints a message. Unlike the rest of Swing, this method can be invoked from any thread.

invokeLater()

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class KeyTextTester
{
    public static void main(String args[])
    {
        Runnable runner = new Runnable()
        {
            public void run()
            {
                JFrame frame = new JFrame("Key Text Sample");
                JTextField textField = new JtextField();
                frame.add(textField, BorderLayout.SOUTH);
                frame.setSize(300, 200);
                frame.setVisible(true);
            }
        };
        SwingUtilities.invokeLater(runner);
    }
}
```

isEventDispatcherThread()

- The SwingUtilities includes the isEventDispatcherThread() method. This method returns true if it was called on the EDT.

  We can use this method to avoid the creation of a new thread when the call is on the EDT.

isEventDispatcherThread()

```
public void doSomething()
{
    Runnable program = new Runnable()
    {
        public void run()
        {
            tf.setText("Bla Bla");
        }
    };
    if(SwingUtilities.isEventDispatcherThread()
    {
        program.run();
    }
    else
    {
        SwingUtilities.invokeLater(program);
    }
}
```

`invokeAndWait()`

- The `SwingUtilities` class includes the `invokeAndWait()` method that works similarly to `invokeLater()` except that it blocks the current thread and waits till the EDT completes executing the task.

`repaint()` & `revalidate()`

- The `repaint()` method passes the EDT a request to call `paint()`.
- The `revalidate()` method forms a component to lay out its children.
- Both methods can be called on any thread. Both methods do their job on the EDT.

The `Timer` Class

- The `Timer` class, both the one in `java.util` and the one in `javax.swing`, offer similar functionality allowing to schedule a specific thread to start.

The `Timer` Class

```
java.util.Timer timer = new java.util.Timer();
timer.schedule(new TimerTask()
{
    public void run()
    {
        SwingUtilities.invokeLater(new Runnable()
        {
            tf.setBackground(getRandomBackgroundColor());
        };
    }
},0,4000);
```

The `SwingWorker` Class

- The `SwingWorker` class was introduced in Java SE 6.
- Should you work with a previous Java SE version you can download this class at https://swingworker.dev.java.net/.
- This utility class simplifies Swing GUI applications development.

Threading Issues

The AWT Events Dispatching

- Swing GUI components rely on AWT support for events dispatching.

  The same well known events dispatching mechanism used in AWT is used in Swing based applications.

Three Threads

- Each time we run a swing application there are three main threads:

  The Application Thread

  This is the main thread of our application. It is the thread that runs our application.

  The Toolkit Thread

  This thread is responsible for capturing the system events, such as keys pressing and mouse movements. The captured events are sent over to the EDT thread.

Three Threads

The EDT Thread

This thread is in charge of dispatching the events captured by the toolkit thread to the relevant components and for calling the paint method. This is also the thread through which the events are hand held by their listeners.

Avoid Blocking The EDT

- Blocking the EDT for a long period of time will freezes the user interface and the user will think the application runs slowly.

  Sample for cases in which the EDT might be blocked include: reading / writing to files, connecting a database, connecting another computer etc.

- Trying to avoid it by initiating another separated thread that includes (in its end) a call to update the component is a violation to the swings single thread rule.

  Doing it might cause a dead lock.

Swing Single Thread Rule

- The EDT is responsible for executing any method that modifies the component, including the constructors.
- Swing is not “thread-safe” and the common rule states that every Swing related operation should be invoked on the EDT.

Swing Single Thread Rule

- Trying to interact with GUI components via threads other than the EDT might cause a dead lock.

  In most cases we won't experience any dead lock. Yet, the risk exists. This is also the reason for the famous recommendation to avoid creating the GUI components on the main application thread.

invokeLater()

- The SwingUtilities class includes the invokeLater() method that enables us to set a new thread to be called on the EDT.
- Using this method we can place the operations with the GUI components on the EDT.

invokeLater()

```
public static void invokeLater(Runnable doRun)
```

Causes *doRun.run()* to be executed asynchronously on the AWT event dispatching thread. This will happen after all pending AWT events have been processed. This method should be used when an application thread needs to update the GUI. In the following example the invokeLater call queues the Runnable object doHelloWorld on the event dispatching thread and then prints a message. Unlike the rest of Swing, this method can be invoked from any thread.

invokeLater()

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class KeyTextTester
{
    public static void main(String args[])
    {
        Runnable runner = new Runnable()
        {
            public void run()
            {
                JFrame frame = new JFrame("Key Text Sample");
                JTextField textField = new JtextField();
                frame.add(textField, BorderLayout.SOUTH);
                frame.setSize(300, 200);
                frame.setVisible(true);
            }
        };
        SwingUtilities.invokeLater(runner);
    }
}
```

isEventDispatcherThread()

- The SwingUtilities includes the isEventDispatcherThread() method. This method returns true if it was called on the EDT.

  We can use this method to avoid the creation of a new thread when the call is on the EDT.

isEventDispatcherThread()

```
public void doSomething()
{
    Runnable program = new Runnable()
    {
        public void run()
        {
            tf.setText("Bla Bla");
        }
    };
    if(SwingUtilities.isEventDispatcherThread()
    {
        program.run();
    }
    else
    {
        SwingUtilities.invokeLater(program);
    }
}
```

`invokeAndWait()`

- The `SwingUtilities` class includes the `invokeAndWait()` method that works similarly to `invokeLater()` except that it blocks the current thread and waits till the EDT completes executing the task.

`repaint()` & `revalidate()`

- The `repaint()` method passes the EDT a request to call `paint()`.
- The `revalidate()` method forms a component to lay out its children.
- Both methods can be called on any thread. Both methods do their job on the EDT.

The `Timer` Class

- The `Timer` class, both the one in `java.util` and the one in `javax.swing`, offer similar functionality allowing to schedule a specific thread to start.

The `Timer` Class

```
java.util.Timer timer = new java.util.Timer();
timer.schedule(new TimerTask()
{
    public void run()
    {
        SwingUtilities.invokeLater(new Runnable()
        {
            tf.setBackground(getRandomBackgroundColor());
        };
    }
},0,4000);
```


This code will causes the text field background to change its color every four seconds.

The `SwingWorker` Class

- The `SwingWorker` class was introduced in Java SE 6.
- Should you work with a previous Java SE version you can download this class at https://swingworker.dev.java.net/.
- This utility class simplifies Swing GUI applications development.