Threads in Swing

Using threads for long running tasks.

Other ways to use threads:

1) TimerTask in java.util and javax.swing

2) Executer - manage a thread pool

3) Future - return a result later

3 Kinds of Threads

In a Swing app:

❑ initial thread starts the application

❑ Event Dispatcher Thread
  ▪ handles all UI events, updates Swing UI

❑ Worker Threads (Background Threads)
  ▪ perform long running tasks

Why Bother with Threads?

❏ Prevent UI from *Freezing* while work is being done

- connecting to database

- downloading something

❏ Avoid *Thread Interference* and memory inconsistency

- Like in Homework 3

"main" method - the wrong way

- your "main" class runs in the initial thread (any thread)
- this code starts Swing UI on the same thread.

```java
public class PurseApp {
  public static void main(String[] args) {
      Purse purse = new Purse( 10 );
      // dependency injection
      PurseUI ui = new PurseUI( purse );
      ui.setVisible(true);
  }
}
```

Use SwingUtilities to launch UI

- Oracle says you should both <u>create</u> and <u>launch</u> the UI on the *Event Dispatcher thread*.

Use `SwingUtilities.invokeLater( runnable );`

```java
public class PurseApp {
  public static void main(String[] args) {
    SwingUtilities.invokeLater(
      new Runnable( ) {
        public void run() {
            // create and start Swing UI
        }
      }
    );
}
```

SwingUtilities Example

- Create PurseUI inside the Runnable task, because the UI will create Swing components.

- purse is final so it can be accessed in anonymous class

```java
public static void main(String[] args) {
    final int capacity = 10;
    final Purse purse = new Purse( capacity );
    SwingUtilities.invokeLater(
      new Runnable( ) {
        public void run() {
            // Must CREATE PurseUI inside Runnable
            PurseUI pui = new PurseUI( purse );
            pui.setVisible(true);
        }
      }
    );
```

SwingUtilities

SwingUtilities

invokeLater(Runnable): void

invokeAndWait(Runnable): void

isEventdispatcherThread(): bool

many more methods

Rules for Event Dispatcher Thread

To prevent UI from freezing and to prevent memory inconsistency:

1) operations on UI components should be done *only* in the Event Dispatcher thread

2) time-consuming operations should <u>never</u> be done on the Event Dispatcher thread

Note that all UI events (button press, state change) invoke event handlers on the event dispatcher thread.

Example: Fibonacci

- as an example of a slow operation, let's compute Fibonacci numbers by recursion.
- fib(0) = 1, fib(1) = 1, fib(n) = fib(n-1) + fib(n-2)

```java
public class Ribonacci {

    // this method could be static
    public long fibonacci(int n) {
        if (n < 0) return 0;
        if (n <= 1) return 1;
        return fibonacci(n-2) + fibonacci(n-1);
    }


    //TODO: test this code
}
```

UI for Fibonacci

Fibonacci Madness

Fibonacci([ ]) = [   ]   OK

Enter an integer
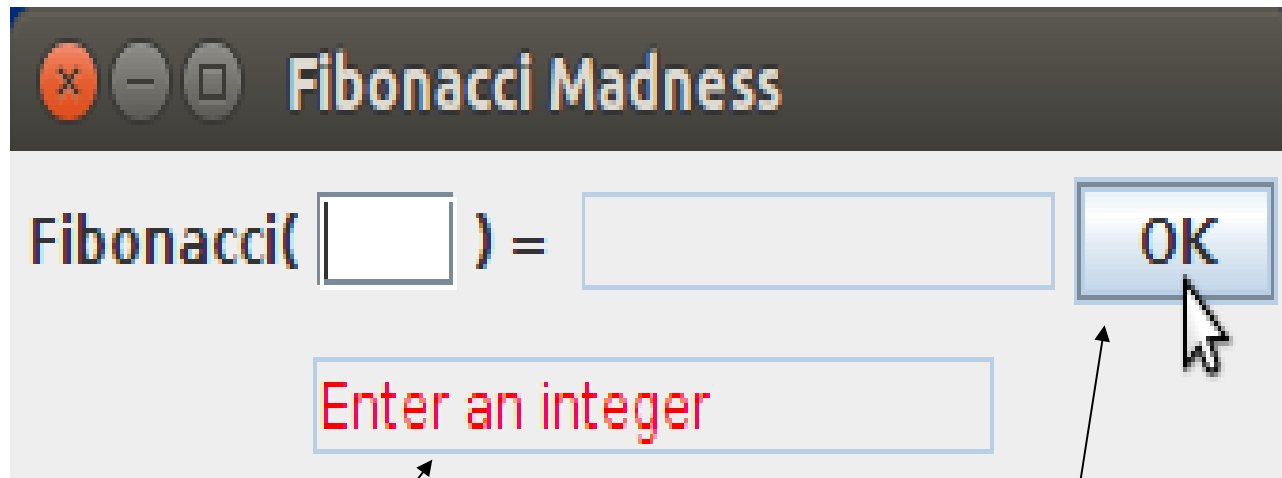
Status or error message.
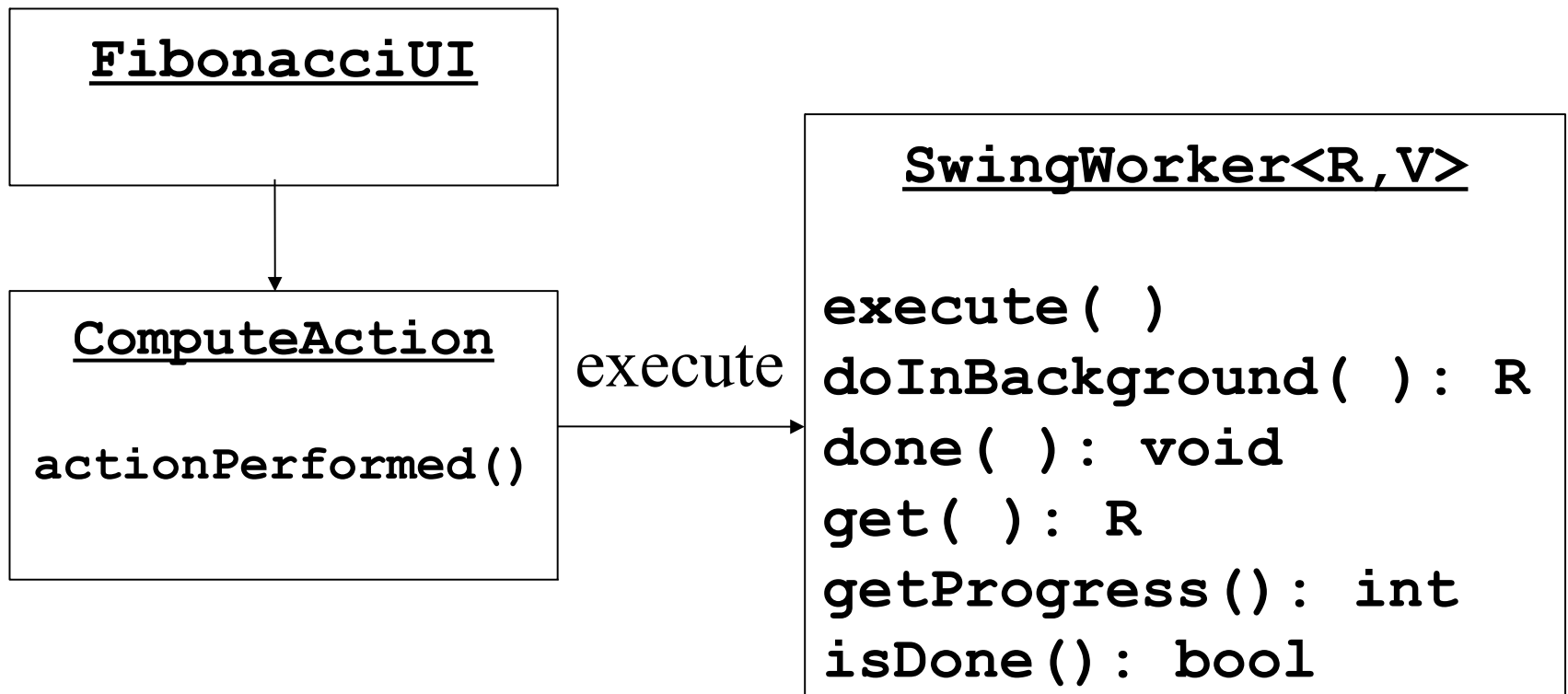
TODO:
add ActionListener

Frozen UI

UI freezes (unresponsive) if you use try to compute large fibonacci numbers on the event dispatcher thread.

```java
// ActionListener method for fibonacci UI
public void actionPerformed(ActionEvent evt) {
    String value = inputField.getText().trim();
    if (value.isEmpty()) return;
    int n = Integer.parseInt( value );
    setMessage( "working" );
    long result = Fibonacci.fibonacci( n );
    outputField.setText( Long.toString(result) );
    setMessage( "" );
}
```

SwingWorker

SwingWorker runs a task in a background thread.

SwingWorker communicates result to Event Dispatcher Thread.

```
FibonacciUI
```

```
ComputeAction

actionPerformed()
```

execute

```
SwingWorker<R,V>

execute( )
doInBackground( ): R
done( ): void
get( ): R
getProgress(): int
isDone(): bool
```

How to Use SwingWorker

1) Create a subclass of SwingWorker for your task.

2) Override 2 methods:

`doInBackground( )` - do the work (on background thread)

`done( )` - communicate the result to UI (this method runs on event dispatcher thread)

Optional:

`publish( V stuff )` - publish intermediate results

Fibonacci Worker

R = return type = Long,

use constructor param to pass value of n for Fibonacci

```
class FibonacciWorker
            extends SwingWorker<Long,Void> {
    private int n;
    public FibonacciWorker(int n) { this.n = n; }

    @Override
    protected Long doInBackground() throws … {
        long result = Fibonnaci.fibonacci(n);
        return result;
    }
```

Getting Result of Fibonacci Worker

Use SwingWorker.done( ) to display the result.

get( ) throws exception if the result is not ready yet.

```java
@Override
protected void done( ) {
  try {
      result = this.get( );
      // update the UI
      outputfield.setText(result.toString());
      setMessage(""); // clear status msg
  } catch(ExecutionException|
          InterruptedException ex) {
      setMessage("Execution Error");
```

More About SwingWorker

1) Can invoke only one time. Create a new instance each time you need to do a task.

2) Can "cancel" a SwingWorker, but requires cooperation of the task. See *Java Tutorial*.

3) Status methods:

```
getProgress( )
isDone( )
isCancelled( )
```

References

The Java Tutorial:
https://docs.oracle.com/javase/tutorial

Concurrency in Swing

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

Concurrency (general)

https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html