livework

Alex McLean - ma503am@gold.ac.uk

Copyright 2006

Package
livework

livework
Class Clock

```
java.lang.Object
   |
   +-livework.Clock
```

All Implemented Interfaces:
java.lang.Runnable

---

public class **Clock**
extends java.lang.Object
implements java.lang.Runnable

Threaded object to handle reasonably accurate timing for an application.
When instantiated a Clock goes at a default rate of 120 'ticks' per minute, changeable via the setTicksPerMinute() method. Every tick each member object in the triggerables array has its trigger() method called.

Author:
Alex Mclean - ma503am@gold.ac.uk

Fields

maxTriggerables

```
public static final int maxTriggerables
```

Maximum number of objects this object can handle.
Constant value: **256**

---

interval

```
private double interval
```

milliseconds per 'tick'

---

startTime

```
private long startTime
```

the time the clock starts

---

ticks

```
private int ticks
```

number of ticks since the clock started

---

triggerableCount

```
private int triggerableCount
```

number of objects in this object's triggerable array

(continued on next page)

(continued from last page)

triggerables

```
private livework.Triggerable triggerables
```

objects to trigger() every 'tick'

Constructors

Clock

```
public Clock()
```

Class constructor

Methods

setTicksPerMinute

```
void setTicksPerMinute(double tpm)
```

Parameters:

`tpm` - number of ticks per minute

getTicksPerMinute

```
double getTicksPerMinute()
```

addTriggerable

```
public boolean addTriggerable(Triggerable triggerable)
```

Adds an object to receive clock ticks via it's triggerable() method. The object must implement the Triggerable interface.

Parameters:

`triggerable`

Returns:

true if the object was successfully added

run

```
public void run()
```

Thread starting point. Just calls the tickLoop() method.

tick

```
private void tick()
```

called once per tick by the tickLoop(). Calls the trigger() array on each object in the triggerable array

(continued from last page)

tickLoop

```
private void tickLoop()
```

The thread loop. Calls tick() at the defined rate. Uses regulates itself using java.lang.nanoTime() for accuracy, particularly on linux based systems.

livework
Class CompileException

```
java.lang.Object
   |
   +-java.lang.Throwable
      |
      +-java.lang.Exception
         |
         +-livework.CompileException
```

All Implemented Interfaces:
java.io.Serializable

---

class **CompileException**
extends java.lang.Exception

A simple exception to represent a compile error.
Author:
Alex McLean - ma503am@gold.ac.uk

Fields

serialVersionUID

```
private static final long serialVersionUID
```

Constant value: **1**

Constructors

CompileException

```
CompileException()
```

CompileException

```
CompileException(java.lang.String message)
```

livework
Class Editor

```
java.lang.Object
   ┗-java.awt.Component
      ┗-java.awt.Container
         ┗-java.awt.Window
            ┗-java.awt.Frame
               ┗-javax.swing.JFrame
                  ┗-livework.Editor
```

All Implemented Interfaces:
java.io.Serializable, java.awt.MenuContainer, java.awt.image.ImageObserver, javax.accessibility.Accessible, java.awt.MenuContainer, javax.swing.RootPaneContainer, javax.accessibility.Accessible, javax.swing.WindowConstants

---

public class **Editor**
extends javax.swing.JFrame

The user interface for a livecoder using this application.
A frame with two components, one being a text editor (of the excellent JEditTextArea class) and the second a textarea any compiler errors (it actually captures everything sent to standard out).
Whenever the livecoder presses ALT-X, the sourcecode is saved, causing the LiveProxy to try to recompile and (if successful) reload it.

See Also:
LiveProxy, jedit.JEditTextArea, ErrTextArea

Author:
Alex McLean - ma503am@gold.ac.uk

Fields

sourcecode

```
private java.io.File sourcecode
```

File containing sourcecode that is being edited

---

textarea

```
private jedit.JEditTextArea textarea
```

The component for editing the sourcecode

---

errTextArea

```
private livework.ErrTextArea errTextArea
```

The textarea showing compiler output

(continued from last page)

serialVersionUID

```
private static final long serialVersionUID
```

I'm not going to serialize this class, but this avoids a compiler warning
Constant value: **1**

Constructors

Editor

```
public Editor(java.io.File sourcecode)
```

Constructor for this class

Parameters:

`sourcecode` - the file to edit

Methods

initErrorComponent

```
private javax.swing.JComponent initErrorComponent()
```

initialises the pane showing errors

Returns:

a pane containing the ErrTextarea object

---

initSourcecodeComponent

```
private java.awt.Component initSourcecodeComponent()
```

initialises the pane containing the text editor

Returns:

a pane containing the JEditTextArea object with the sourcecode loaded

---

save

```
void save()
```

saves the edited file

livework
Class EditorInputHandler

```
java.lang.Object
   |
   +-java.awt.event.KeyAdapter
      |
      +-jedit.InputHandler
         |
         +-jedit.DefaultInputHandler
            |
            +-livework.EditorInputHandler
```

All Implemented Interfaces:
java.awt.event.KeyListener

---

public class **EditorInputHandler**
extends jedit.DefaultInputHandler

A simple subclass of JEdit's DefaultInputHandler to bind ALT-x to the editor's save method.
Author:
Alex McLean - ma503am@gold.ac.uk

Fields

editor

```
private livework.Editor editor
```

Constructors

EditorInputHandler

```
public EditorInputHandler(Editor editor)
```

Contructor for this class

Parameters:
`editor` - The editor this is the input handler for

Methods

addDefaultKeyBindings

```
public void addDefaultKeyBindings()
```

Adds ALT-x to the defaults

livework
Class Envelope

```
java.lang.Object
   |
   +-livework.Envelope
```

---

public class **Envelope**
extends java.lang.Object

A simple but useful envelope class. Tweens between a given a set of discreet values that represent an envelope, so that a value can be returned for a floating point percentage. Tweening is linear.

This allows a set of envelopes to be applied to a visual element, for example to control brightness - fading in and out.

The given array of values should be considered as continuous, equally spaced points describing the envelope.

Fields

count

`private int` **`count`**

number of values

---

values

`private float` **`values`**

array of values

Constructors

Envelope

`Envelope``(float[] values)`

Constructor for this class

Parameters:

`values` - array of float values.

Methods

value

`float` **`value`**`(float percentage)`

Calculates a value of the given percentage

Parameters:

`percentage`

(continued from last page)

Returns:

the calculated value

livework
Class ErrTextArea

```
java.lang.Object
   ┗-java.awt.Component
      ┗-java.awt.Container
         ┗-javax.swing.JComponent
            ┗-javax.swing.text.JTextComponent
               ┗-javax.swing.JTextArea
                  ┗-livework.ErrTextArea
```

All Implemented Interfaces:
java.io.Serializable, java.awt.MenuContainer, java.awt.image.ImageObserver, java.io.Serializable, javax.accessibility.Accessible, javax.swing.Scrollable

---

public class **ErrTextArea**
extends javax.swing.JTextArea

A textarea that captures standard error and displays it in itself
Author:
Alex McLean - ma503am@gold.ac.uk

Fields

printStream

`java.io.PrintStream` **`printStream`**

Stream to replace standard error with

---

serialVersionUID

`private static final long` **`serialVersionUID`**

I'm not going to serialize this class, but this avoids a compiler warning
Constant value: **`1`**

Constructors

ErrTextArea

`ErrTextArea``()`

Constructor for this class

livework
Class ErrTextArea.FilteredStream

```
java.lang.Object
   |
   +-java.io.OutputStream
      |
      +-java.io.FilterOutputStream
         |
         +-livework.ErrTextArea.FilteredStream
```

All Implemented Interfaces:
java.io.Flushable, java.io.Closeable

---

class **ErrTextArea.FilteredStream**
extends java.io.FilterOutputStream

Inner class for our replacement standard error to use

Constructors

ErrTextArea.FilteredStream

```
public ErrTextArea.FilteredStream(java.io.OutputStream aStream)
```

Methods

write

```
public void write(byte[] b)
  throws java.io.IOException
```

---

write

```
public void write(byte[] b,
         int off,
         int len)
  throws java.io.IOException
```

livework
Class LiveProxy

```
java.lang.Object
   |
   +-java.lang.ClassLoader
      |
      +-livework.LiveProxy
```

All Implemented Interfaces:
java.lang.reflect.InvocationHandler

---

public class **LiveProxy**
extends java.lang.ClassLoader
implements java.lang.reflect.InvocationHandler

This class is the heart of this project. Refer to the project report for information about how I arrived at this implementation.

It employs a number of techniques towards the end of allowing a programmer to make changes to the behaviour of objects by changing their classes. This isn't actually possible in Java yet, you cannot add a method to a class that has already been loaded. This class achieves an approximate effect by using the following smoke and mirrors:

- A proxy object - the LiveProxy class acts as a proxy for the object that is being live-edited. This allows all calls to the object to be intercepted. When a call is made to the object via the proxy, before passing the call on, the proxy checks the proxied object's class sourcecode for changes. If the sourcecode has been changed, the class is dynamically compiled and loaded into a fresh package.
- Compilation - this is accomplished by calling javac via the "com.sun.tools.javac.Main" package. Before compilation the source is modified to include a unique package statement. This is necessary because a class can't be loaded twice into the same package.
- Dynamic loading - Because this class extends the ClassLoader class, it can load classes at runtime. Once the class of the proxied object has been successfully compiled it loads the class ready to be instantiated.
- Reflection - Because we can't really change the class of the proxied object (simplification - see project report for the full story), we've loaded the class into a fresh package. This way it can keep its original unqualified classname and therefore the names of its constructors and so on. Now an object can't be moved from one class to another, so we do the next best thing - make a new object, copy across all the fields we can, then forget about the original. This is akin to serialising an object, then deserialising it into a different class, all in one go. Not all the fields will be copyable, some fields might have been deleted by the programmer, or had been changed to a type incompatible with the original. All this is made possible, perhaps even *easy*, by Java reflection. The classes, methods are all accessible as objects.
- Exception handling - if any compilation errors are caught, execution continues with the original version of the object intact.

So that's a short overview, for further details refer to the methods.

Note that a LiveProxy object also starts an Editor to edit the sourcecode of the class that's being proxied. Once the programmer saves their changes to the source, the next call to the object via the proxy will cause the object to be recompiled as above.

Author:
Alex McLean - ma503am@gold.ac.uk

Fields

(continued from last page)

count

```
private int count
```

Count for the number of times this class has been recompiled, used for constructing unique package names.

filename

```
private java.lang.String filename
```

The filename of the sourcecode for the class. It lives within the folder defined by the 'liveclasses' setting.

See Also:
`Settings`

lastModified

```
private long lastModified
```

Records the last modified time of the sourcefile, so we can tell when it is modified again

sourcecode

```
private java.io.File sourcecode
```

Represents the fully qualified filename of the sourcefile

target

```
private java.lang.Object target
```

The compiled target object

Constructors

LiveProxy

```
private LiveProxy(java.lang.String filename)
```

The constructor for this class. It compiles and instantiates the class identified by the given filename. If the file doesn't exist, it is created with some default sourcecode.

Parameters:
`filename`

Methods

createProxy

```
public static java.lang.Object createProxy(java.lang.String filename)
```

Instantiates itself with the given sourcefile, instantiating the target object in the process, and returns a proxy class that acts as the target object but all accesses are passed to this object's invoke() method

Parameters:

(continued from last page)

`filename` - sourcefile of the class for the target object

Returns:

the proxy object

---

getInstanceVariables

```
static java.lang.reflect.Field[] getInstanceVariables(java.lang.Class cls)
```

The following method was taken from public domain book example - Forman and Forman, Java Reflection in Action.
Returns an array of fields for the given class

---

bumpCount

```
private void bumpCount()
```

Increases the count field by one and saves it to disk. This means that each edit of the java source is saved to a fresh packages across multiple invocations of this class. This isn't necessary at the moment but future versions of this application might allow the programmer to jump easily jump back to editing and reloading previous versions of the class.

---

classdir

```
private java.lang.String classdir()
```

Retrieves from the user settings the folder where the sourcefiles to be edited are located.

---

copyClass

```
private java.lang.String copyClass()
  throws CompileException
```

Copies the sourcefile to a fresh package and compiles it there

Returns:

the name of the new class qualified by the new package

Throws:

`CompileException` - thrown when the compile wasn't successful. Compile errors are sent to standard error, as captured by the editor and presented to the programmer to meditate over.

---

copyFields

```
private void copyFields(java.lang.Object newTarget,
         java.lang.Object source)
```

---

defaultSourcecode

```
private void defaultSourcecode()
```

---

findField

```
java.lang.reflect.Field findField(java.lang.Class myClass,
         java.lang.String fieldName)
```

(continued from last page)

getClassData

```
private byte[] getClassData(java.lang.String directory,
         java.lang.String name)
```

The following method was taken from a public domain example from the book - Forman and Forman, Java Reflection in Action.
It returns the byte compiled class data for the given file within the given directory.

initGlobals

```
private void initGlobals(java.lang.Object object)
  throws java.lang.Throwable
```

This method sidesteps a big problem with this whole approach - field initialisation. When you instantiate an object, fields are often initialised, ie:

```
int foo = 40;
```

Now if we changed that value to 41 and triggered a reload of the class, the field would get initialised to 41, but then overwritten back to 40 by the copyFields() method. This might be what the programmer wanted to happen, but then again, might not. Further, the initialisation might do something that the programmer really only wants to happen once, not every time the program is compiled.
To allow the programmer full control over field initialisation, this method is called *after* copyFields(). It reflectively looks for a method on the new target object called "initGlobals()". If it finds one, it invokes (calls) it. If it doesn't, it does nothing.
The programmer can therefore write code in such a method that initialises fields exactly as s/he wants.

Parameters:

`object` - The new target object

Throws:

`Throwable` - An exception raised by a initGlobals method on the new target object.

invoke

```
public java.lang.Object invoke(java.lang.Object proxy,
         java.lang.reflect.Method method,
         java.lang.Object[] args)
  throws java.lang.Throwable
```

Intercepts calls to the target object via this proxy object. If necessary, it reloads the target object invoking the method upon it.

loadObject

```
private java.lang.Object loadObject()
  throws CompileException
```

Loads the class and instantiates a new target object from it.

Returns:

the new target object

Throws:

`CompileException` - thrown when the compile failed

reloadTarget

```
public void reloadTarget()
```

(continued from last page)

Reloads the target object from its sourcecode.

livework
Class LiveWork

```
java.lang.Object
   |
   +-livework.LiveWork
```

public class **LiveWork**
extends java.lang.Object

Contains the main method for the application.

Constructors

LiveWork

```
public LiveWork()
```

The entry point for the livework application.
Makes a LiveProxy for the Bare class, and instantiates a Clock object that calls the trigger() method of the proxied object.

Methods

main

```
public static void main(java.lang.String[] args)
```

The main method for the LiveWork application, merely instantiates an object of this class.

Parameters:

`args` - not used

livework
Class MinimalTrigger

```
java.lang.Object
   |
   +-livework.MinimalTrigger
```

All Implemented Interfaces:
Triggerable

---

class **MinimalTrigger**
extends java.lang.Object
implements Triggerable

A class to instantiate a simple object from as a placeholder if a target object couldn't be loaded at runtime. This is only used if an object doesn't compile when the LiveProxy is first instantiated, where a previous compile isn't available to fall back on.

Author:
Alex McLean - ma503am@gold.ac.uk

Constructors

MinimalTrigger

```
MinimalTrigger()
```

Methods

trigger

```
public void trigger()
```

livework
Class Nourathar

```
java.lang.Object
   ┗-java.awt.Component
      ┗-java.awt.Container
         ┗-java.awt.Window
            ┗-java.awt.Frame
               ┗-javax.swing.JFrame
                  ┗-livework.Nourathar
```

All Implemented Interfaces:
java.lang.Runnable, java.io.Serializable, java.awt.MenuContainer, java.awt.image.ImageObserver, javax.accessibility.Accessible, java.awt.MenuContainer, javax.swing.RootPaneContainer, javax.accessibility.Accessible, javax.swing.WindowConstants

---

public class **Nourathar**
extends javax.swing.JFrame
implements javax.swing.WindowConstants, javax.accessibility.Accessible, javax.swing.RootPaneContainer, java.awt.MenuContainer, javax.accessibility.Accessible, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.lang.Runnable

A frame for producing simple live video animations. Intended as a demonstration for the rest of this application but explores some ideas inspired by Mary Hallock Greenwelt's concept of "Nourathar" - the playing of colour transitions as a performance.
I implemented an offscreen buffer to avoid flicker, but since then have readd that java2d can do this internally. If I develop this further I'll be sure to simplify things by removing the manual offscreen buffer.
Author:
Alex McLean - ma503am@gold.ac.uk

Fields

serialVersionUID

```
private static final long serialVersionUID
```

I'm not going to serialize this class, but this avoids a compiler warning
Constant value: **1**

---

colors

```
private java.util.LinkedList colors
```

to contain the currently active NourtharColor objects

---

envelopes

```
private java.util.Hashtable envelopes
```

a lookup table for available envelopes to apply to the brightness levels of the NouratharColor objects

g2d

```
private java.awt.Graphics2D g2d
```

The graphics context of the offscreen buffer

offscreenBuffer

```
private java.awt.Image offscreenBuffer
```

The offscreen buffer

Constructors

Nourathar

```
public Nourathar()
```

Contructor for this class. Initialises frame, offscreen buffer, envelopes and starts the thread for the object.

Methods

initEnvelopes

```
void initEnvelopes()
```

Initialises the envelopes.

paint

```
public void paint(java.awt.Graphics g)
```

Paints the offscreen buffer.

play

```
public NouratharColor play(java.awt.Color color,
        java.lang.String envelopeName,
        int ttl,
        double x,
        double y,
        double size)
```

Method for 'playing' the nourathar. Causes a NouratharColor visual element to appear of the given colour, location and size for the given length of time, with the brightness modulated by the given envelope. The parameters are passed straight to the constructor of the NouratharColor class, apart from the 'x', 'y' and 'size' parameters, which are given as proportion of the content pane size and converted to pixel values.

Parameters:

`color` - Colour of the element at it's maximum brightness

Returns:

The created NouratharColor object

(continued from last page)

render

```
void render()
```

Draws the active nouratharColor visual elements on the offscreen buffer

---

run

```
public void run()
```

Thread loop for this class. Renders and paints the window once per 50 milliseconds

livework
Class NouratharColor

```
java.lang.Object
   |
   +-java.awt.geom.RectangularShape
      |
      +-java.awt.geom.Ellipse2D
         |
         +-java.awt.geom.Ellipse2D.Double
            |
            +-livework.NouratharColor
```

All Implemented Interfaces:
java.lang.Cloneable, java.awt.Shape

---

public class **NouratharColor**
extends java.awt.geom.Ellipse2D.Double

A class representing a circle of a given colour, brightness envelope, lifespan (ttl), location and size.

Author:
Alex McLean - ma503am@gold.ac.uk

See Also:
Nourathar

Fields

age

`private int **age**`

age of this object in triggers

---

color

`private java.awt.Color **color**`

Color of this object

---

envelope

`private livework.Envelope **envelope**`

---

ttl

`private int **ttl**`

Constructors

(continued from last page)

NouratharColor

```
NouratharColor(java.awt.Color color,
               Envelope envelope,
               int ttl,
               double x,
               double y,
               double size)
```

Constructor for this object

Methods

dampenedColor

```
private java.awt.Color dampenedColor()
```

'Dampens' the colour of this object by finding the current value of the brightness envelope, and applying it to the alpha component of the colour.

Returns:

the new colour

---

draw

```
void draw(java.awt.Graphics2D g2d)
```

Draws the colour, at its predefined location and size, and using the colour calculated using the predefined brightness envelope.

Parameters:

`g2d`

---

getStrength

```
double getStrength()
```

Calculates the current strength (or brightness) of the colour according to the brightness envelope

Returns:

The calculated value

---

isExpired

```
boolean isExpired()
```

Calculates whether this object is as old or order than it's given ttl (lifespan)

Returns:

The calculated value

livework
Class Settings

```
java.lang.Object
   |
   +-livework.Settings
```

---

public class **Settings**
extends java.lang.Object

Singleton class for managing user settings for the application.
User settings strings can be set and retreived. On instantiantion the settings are checked for required fields, which are set with default values as necessary.
Settings are saved to a file "preferences.txt" under a directory called '.livework' under the user's home directory. This is conventional under UNIX based systems, but may annoy users of other operating systems.

Author:
Alex McLean - ma503@gold.ac.uk

Fields

singleton

```
private static livework.Settings singleton
```

Singleton object

---

table

```
private java.util.Hashtable table
```

Holds the settings in memory

---

settingsFile

```
private java.io.File settingsFile
```

File in which the settings are stored

---

dataFolder

```
private java.io.File dataFolder
```

Directory in which settings and other user data are saved.

Constructors

Settings

```
protected Settings()
```

Constructor for this class
Cannot be called externally, call getInstance() instead.

Methods

(continued from last page)

getInstance

```
public static Settings getInstance()
```

Retreives the singleton object for this class.

Returns:
The settings object.

---

initSettings

```
private void initSettings()
```

Makes the data folder and settings file if they're missing

---

save

```
public void save()
```

Saves settings to the settings file.

---

load

```
public void load()
  throws java.io.IOException
```

Loads the settings file. Would rarely need to be called externally, as it's called automatically when this object is first instantiated.

Throws:
`IOException` - thrown where there is a problem loading the settings file

---

get

```
public java.lang.String get(java.lang.String key)
```

Get a setting

Parameters:
`key` - The setting to be retrieved

Returns:
The value of the setting

---

put

```
public void put(java.lang.String key,
         java.lang.String value)
```

Set a setting

Parameters:
`key` - The setting to be set
`value` - The new value for the setting

---

sget

```
public static java.lang.String sget(java.lang.String key)
```

(continued from last page)

Static version of the set method

sput

```
public static void sput(java.lang.String key,
         java.lang.String value)
```

Static version of the put method

checkSettings

```
private void checkSettings()
```

Checks that the settings are valid, wiping any invalid values.

initRequired

```
private void initRequired()
```

Initialises required values and save()s if necessary.

initLiveClasspath

```
private void initLiveClasspath()
```

Initialises the "liveclasses" setting, where livecoded classes are saved and compiled. The directory is created if necessary.

livework
Interface Triggerable

All Known Implementing Classes:
MinimalTrigger

---

public interface **Triggerable**
extends

Interface for objects that have a trigger() method, and can therefore be passed to the Clock class.
See Also:
`Clock`

Methods

trigger

```
public void trigger()
```

livework
Class Util

```
java.lang.Object
   |
   +-livework.Util
```

public class **Util**
extends java.lang.Object

Class containing static utility methods
Author:
Alex McLean - ma503am@gold.ac.uk

Constructors

Util

```
protected Util()
```

This class cannot be instantiated

Methods

error

```
public static void error(java.lang.String title,
         java.lang.String message,
         java.lang.Exception e)
```

Displays an error dialog and exits

Parameters:
`title` - Title of the dialog (or null for the default)
`message` - Contents of the dialog (required)
`e` - Exception that caused the error (or null if none)