Filthy Rich Clients: Animated Effects in Swing Applications

Chet Haase
Kenneth Russell
Romain Guy

Sun Microsystems, Inc.

TS-1297

Goal

Learn how to use 2D, 3D, Swing, and animation to create more compelling, dynamic, and effective GUI applications

Agenda

Animation fundamentals

Into the third dimension

Putting it all together

Agenda

Animation fundamentals

Into the third dimension

Putting it all together

Animation: It's About Time

- Animations should be based on time
  - Not successive steps
  - Accounts for variable machine performance
  - Works the same across all environments
- Determine appropriate speed for animation
- For every animation frame
  - Calculate time delta from last time
  - Calculate change to object from time and speed
  - Render object with appropriate change

Timers: Wakeup Calls

- Timers are utilities for knowing when to render the next frame
- Create Timer with “resolution”
  - Determines frame rate (frames per second)
- Timer will call your code at this frame rate
  - Assuming your frame rate is achievable
- Timers in core JDK
  - **`java.util.Timer`**: for general usage
  - **`javax.swing.Timer`**: GUI specific, calls your code on Event Dispatch Thread

javax.swing.Timer

```
// Create and start timer
startTime = System.currentTime();
timer = new Timer(msBetweenCallbacks, actionListener);
timer.start();

// timer callbacks in actionPerformed() method
public void actionPerformed(ActionEvent ae) {
  // Get elapsed time
  long currentTime = ae.getWhen();
  long elapsedTime = currentTime - startTime;
  // Now do whatever you want with this information
  // ...
}
```

Beyond the Built-Ins

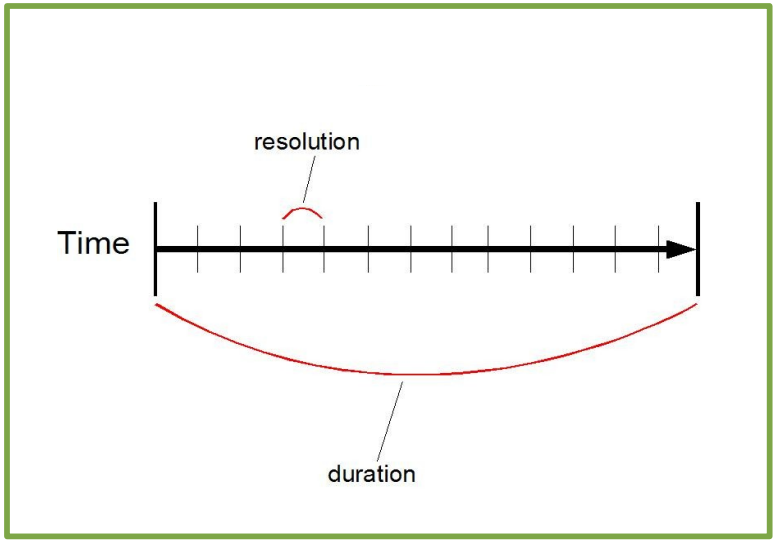
- Key functionality lacking in core timers for typical animation requirements
  - Duration: when to stop the animation?
  - Elapsed time: why not have the system tell you how much time has passed?
  - Repeat: what if you want the animation to repeat, or reverse?
  - Advanced
    - Property setters
    - Non-linear interpolation
    - Triggers

Timing Framework

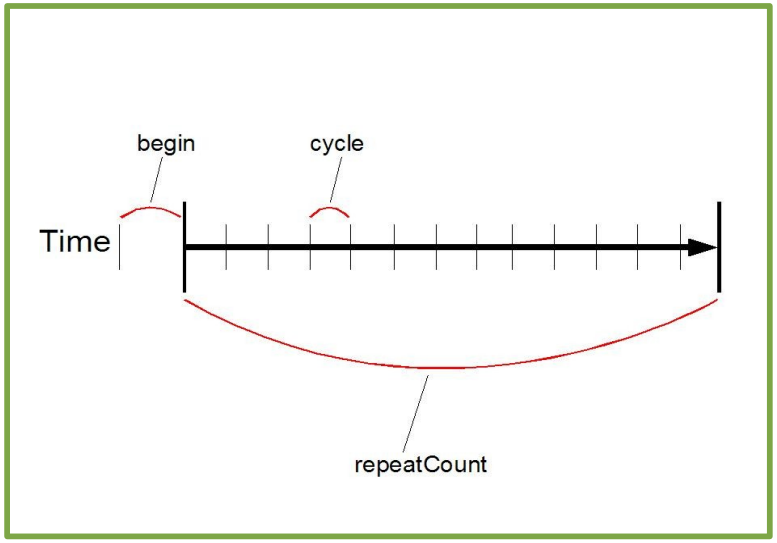
- http://timingframework.dev.java.net
  - Project in development over a year now
- Core concepts:
  - Cycle: basic animation loop
    - Duration, resolution
  - Envelope: contains one or more Cycles
    - Number of cycles, start delay, repeat behavior, end behavior
  - TimingTarget: Callback target
    - **`begin(), end(), timingEvent(fraction)`**
  - TimingController:
    - **`Cycle`**, **`Envelope`**, one or more **`TimingTarget`** objects

Timing Framework: Basics

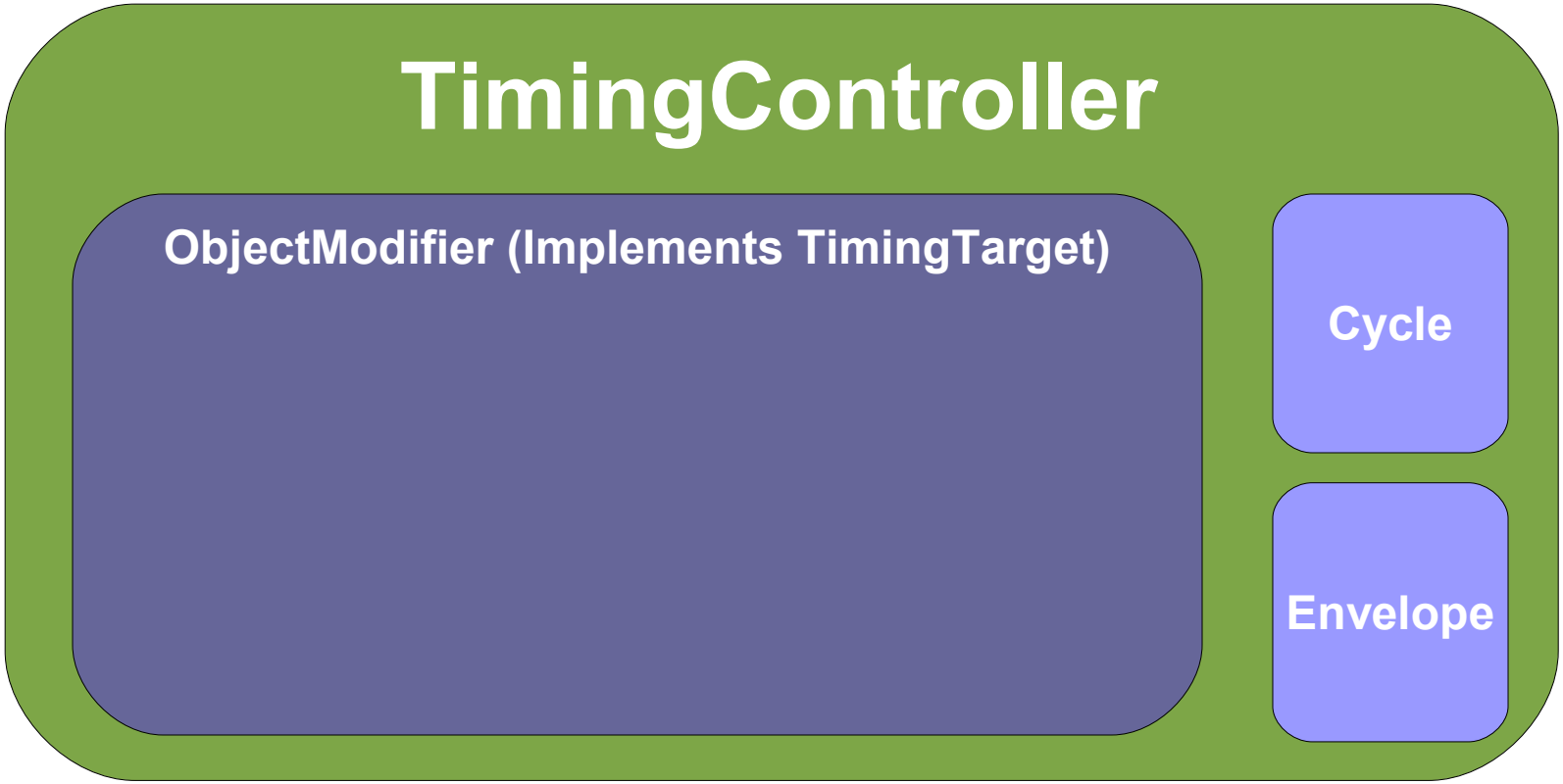
Cycle



Envelope
(contains one or more Cycles)



Timing Framework: Class Diagram



Timing Framework: Basics

```
// MyTarget implements begin/end/timingEvent methods
TimingTarget target = new MyTarget();

// duration = 5 seconds, resolution = 30 ms
Cycle cycle = new Cycle(5000, 30);

// repeats twice, begins after 1/2 second delay,
// reverses after each Cycle, holds final value at end
Envelope envelope = new Envelope(2, 500,
    RepeatBehavior.REVERSE, EndBehavior.HOLD);

// Now create and start timer with the given parameters
TimingController timer = new TimingController(cycle,
    envelope, target);
timer.start();
```

DEMO

BasicRace

BasicRace: The Code

```
// Starts/stops timer based on Go/Stop action events
public void actionPerformed(ActionEvent ae) {
  if (ae.getActionCommand().equals("Go")) {
    timer = new TimingController(RACE_TIME, this);
    timer.start();
  } else if (ae.getActionCommand().equals("Stop")) {
    timer.stop();
  }
}
// Callback: Linearly interpolate car position according
// to fraction of animation elapsed thus far
public void timingEvent(long cycleTime, long totalTime,
                        float fraction) {
  current.x = (int)(start.x + (end.x-start.x) * fraction);
  current.y = (int)(start.y + (end.y-start.y) * fraction);
  track.setCarPosition(current);
}
```

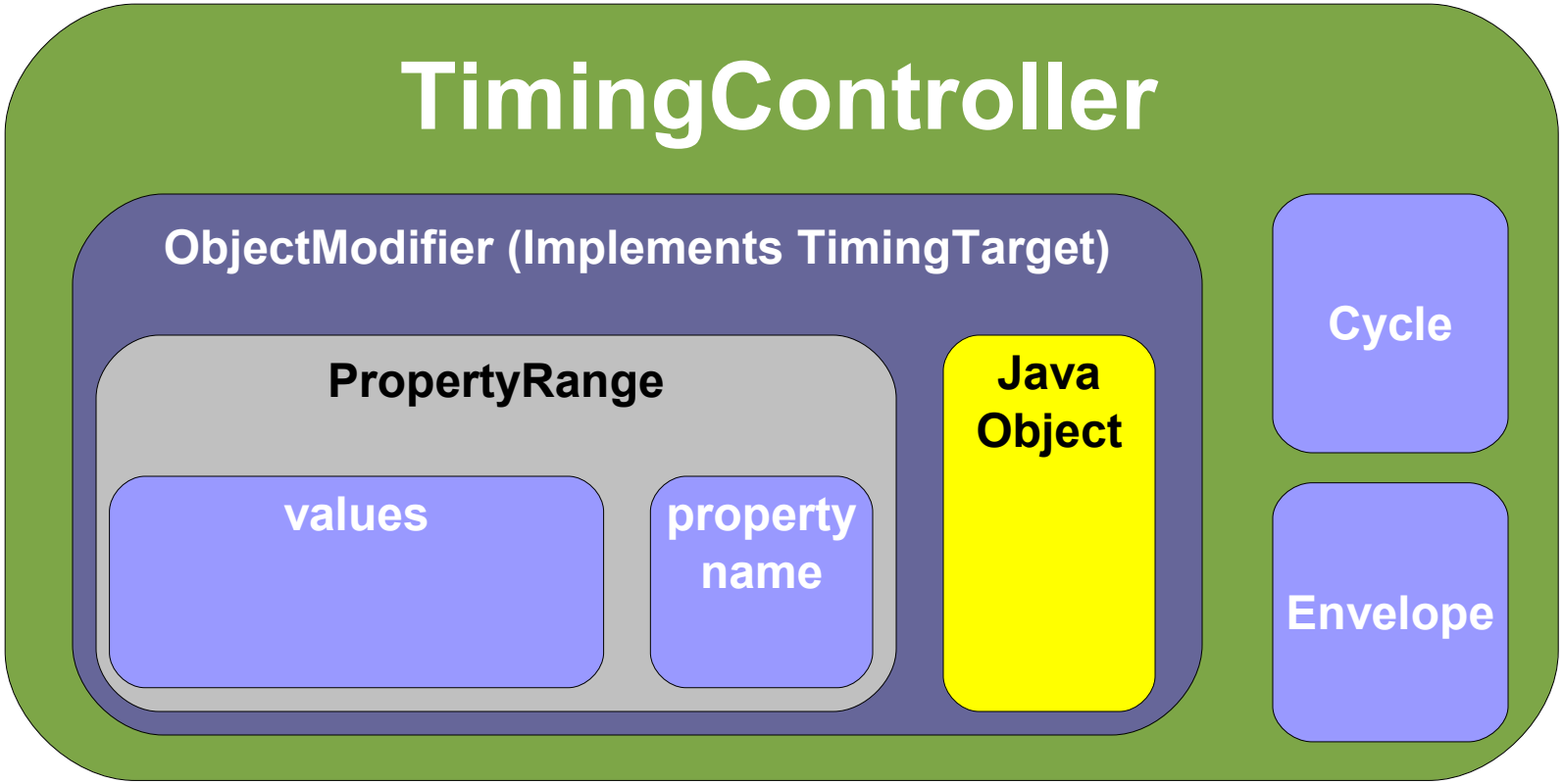
Timing Framework: Advanced

- Property Setters
  - Built-in TimingTargets that animate JavaBeans™ specification properties
- Triggers
  - Animations auto-fired based on various events
- Non-Linear Interpolation
  - More realistic movement
- Multi-step Animations
  - Not just “from -> to”
  - More like “from here…to there1…to there2…to there3…”
  - Or just “to”; animate from current position

Property Setters

- Built-in facility to animate JavaBeans specification properties of Objects
  - “location” of button
- Works for any property name (e.g., “prop”) that has related setter (e.g., “setProp”)
  - Component.size, Component.foreground, Component.location,…
- Custom components or delegators when no appropriate property exists
  - Opacity, rotation, scale

Property Setters: Class Diagram



Property Setters: The Classes

- PropertyRange
  - `range=createPropertyRangePoint("location", from, to);`
  - Defines JavaBean property to be modified
  - Range of values for property
- ObjectModifier
  - `modifier = new ObjectModifier(object, range);`
  - Declares object to be modified with range
  - Implements TimingTarget interface
- TimingController
  - `new TimingController(duration, modifier);`
  - Defines animation characteristics

SetterRace: The Code

```
// instead of manually calculating the car position
// during the animation, set up an ObjectModifier at
// construction time to handle it for us
public SetterRace() {
  PropertyRange range = PropertyRange.
      createPropertyRangePoint("carPosition", start, end);
  ObjectModifier modifier =
      new ObjectModifier(track, range);
  timer = new TimingController(RACE_TIME, modifier);
}

public void actionPerformed(ActionEvent ae) {
  // Same as in BasicRace: start/stop the timer
}
```

Triggers

- Idea: Make it easier to have canonical effects for rich components
  - Hover effect for buttons
  - Pulsating effect for focus events
- Also, simplify sequencing multiple animations
  - For example: Timer B should start when Timer A ends
- Triggers encapsulate EventListeners
  - And save your code the hassle

TriggerRace: The Code

```
// Constructor code is the same as in SetterRace, but
// this time we set up Triggers to handle the button
// events for us; no ActionListener required
public TriggerRace() {
  PropertyRange range = PropertyRange.
      createPropertyRangePoint("carPosition", start, end);
  ObjectModifier modifier =
      new ObjectModifier(track, range);
  timer = new TimingController(RACE_TIME, modifier);
  new ActionTrigger(timer, goButton, TriggerAction.START);
  new ActionTrigger(timer, stopButton,  TriggerAction.STOP);
}
```

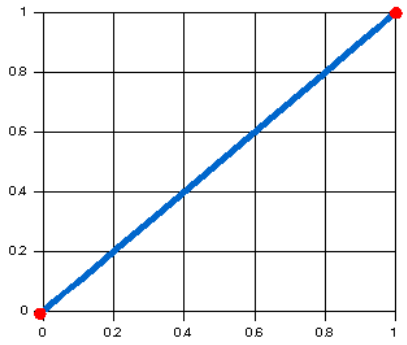
Non-Linear Interpolation

- We live in a non-linear world
  - Gravity, acceleration, deceleration, friction
    - …As well as tripping, stumbling, falling, crashing, settling
- …So our eyes expect to see non-linear movement
- Linear movement
  - Looks unnatural
  - Emphasizes rendering artifacts
    - Easy to track mistakes and hiccups when we are tracking linear movement

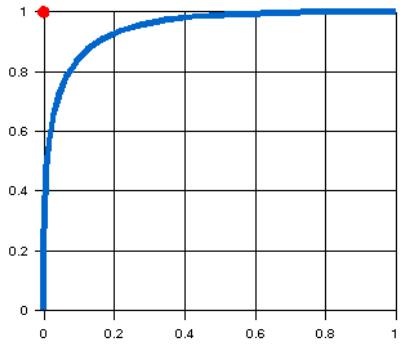
Non-Linear Interpolation: The Classes

- Acceleration/Deceleration: Simplest approach
  - `TimingController.setAcceleration(float);`
  - `TimingController.setDeceleration(float);`
  - Fraction of total time spent speeding up, slowing down
- Spline
  - Bezier curve that defines interpolation used between endpoints of animation

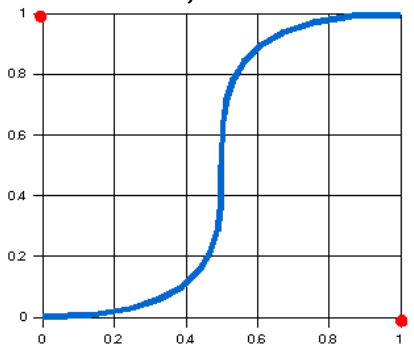
Linear



Fast Start



Ease In, Ease Out



DEMO

Non-Linear Interpolation

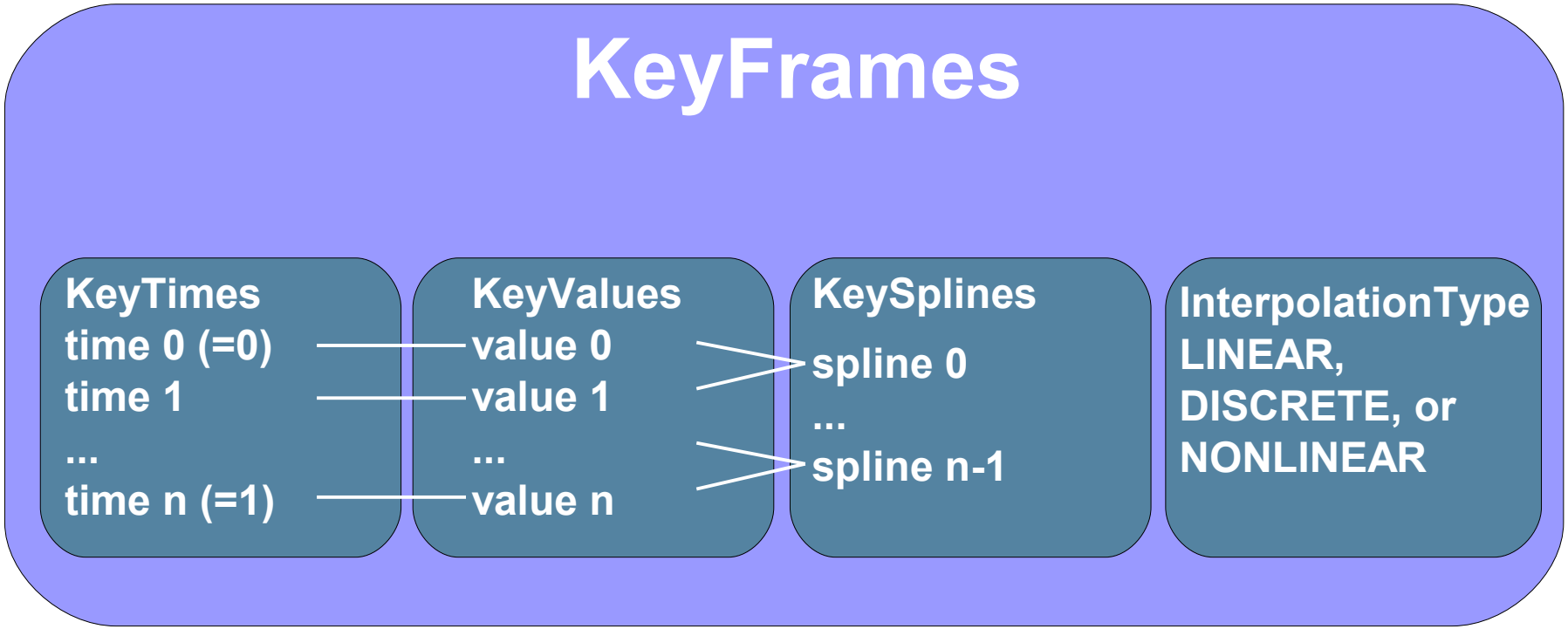
NonLinearRace: The Code

```
// The simplest variant of non-linear movement; use the
// acceleration property of TimingController.  This class
// subclasses the earlier SetterRace class but
// sets the acceleration property to change the timer
// behavior
public class NonLinearRace extends SetterRace {
  public NonLinearRace() {
    // Car will accelerate through the
    // first 70% of the total animation
    timer.setAcceleration(.7f);
  }
}
```

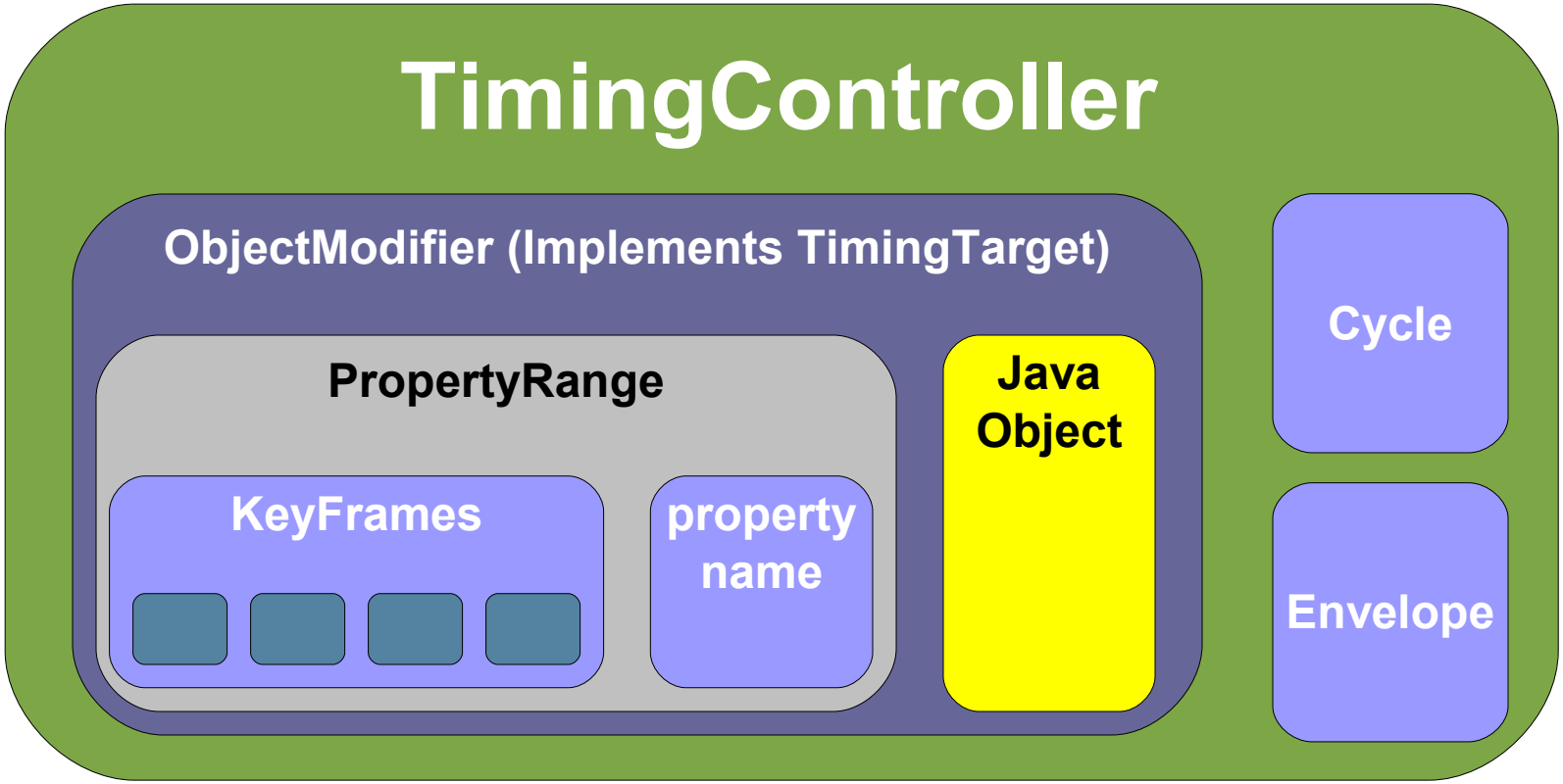
Multi-Step Animations

- May need more than simple “from here to there” movement
  - Objects may follow a complex path
    - A -> B -> C
  - Non-linearity applies: more realistic movement
    - Some movement inherently curve-based
      - Steps in walking,…
- May need simpler model: move “to” destination from wherever object is now
  - Useful for stringing together multiple animations
  - Or animating back from current position
    - When stopping and reversing a running animation

Multi-Step: Class Diagram (KeyFrames)



Multi-Step: Class Diagram (Everything)



Multi-Step Animations: The Classes

- KeyFrames hold information about:
  - values an animation can take during an animation
    - `KeyValues.createKeyValues(Point... values);`
  - times at which these values are assigned
    - `new KeyTimes(float... times);`
  - interpolation between the values
    - `new KeySplines(Spline... splines)`
    - `enum InterpolationType {`
      `LINEAR, DISCRETE, NONLINEAR};`
  - `new KeyFrames(keyValues, keySplines, keyTimes, interpolationType);`

Multi-Step Animations: More Classes

- PropertyRange can take KeyFrames
  - `PropertyRange range = createPropertyRange(prop, keyFrames);`
- ...or can implictly create KeyFrames for you
  - `createPropertyRangeFloat(prop, 5.0, 6.6, 9.7);`
  - will create:
    - `new KeyTimes(0.0f, 0.5f, 1.0f);`
    - `new KeyValuesDouble(5.0, 6.6, 9.7);`
    - `InterpolationType.LINEAR`
    - no KeySplines (not needed for LINEAR interpolation)

DEMO

Multi-Step Animations

MultiStepRace: The Code

```
// We use Points at the corners of the track for values
KeyValues keyValues = KeyValues.createKeyValues(...);
// We use times that will give us a constant travel time
// over each track segment
KeyTimes keyTimes = new KeyTimes(...);
// We use Splines for each segment which will make the
// car accelerate/decelerate appropriately
KeySplines keySplines = new KeySplines(...);
// We can now create KeyFrames from the above variables
KeyFrames keyFrames = new KeyFrames(keyValues, keySplines,
    keyTimes, KeyFrames.InterpolationType.NONLINEAR);
PropertyRange range =
    new PropertyRange("carPosition", keyFrames);
ObjectModifier modifier =
    new ObjectModifier(track, range);
timer = new TimingController(RACE_TIME, modifier);
```

Animation: Summary

- Animation is about varying values over time
- Possible with existing Java classes
  - `java.util.Timer`
  - `javax.swing.Timer`
- Easier with Timing Framework
  - Callbacks give more information (elapsed fraction)
  - Desirable animation behaviors built into framework
    - Repetition, duration, non-linear interpolation, multi-step
  - Natural tie-in to GUI animations
    - ObjectModifier for setting properties, Triggers for events
- API still in development; use it and let me know what changes you would like to see

Agenda

Animation fundamentals

Into the third dimension

Putting it all together

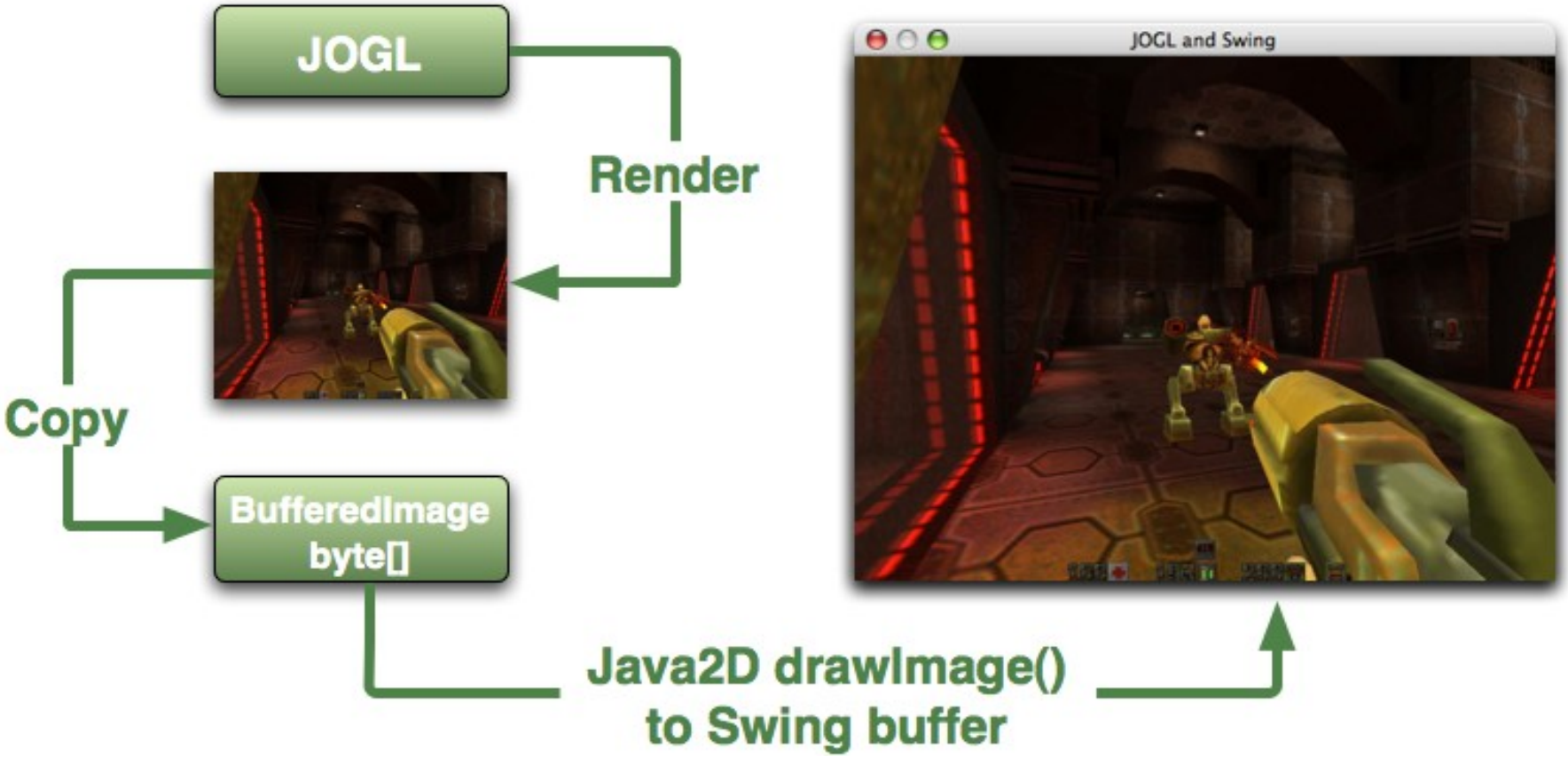
JOGL

- Development version of JSR-231 implementation (Java™ Binding for the OpenGL® API)
  - Not official reference implementation
  - `http://jogl.dev.java.net/`
- Supplies GLJPanel class supporting 100% correct Swing and 3D interaction
  - Performance not great, especially with large windows

Mixing Swing and 3D

- Historical problems
  - Highest 3D performance required heavyweight widget
  - Lightweight/heavyweight mixing issues
- 100% correct Swing integration expensive
  - Render to off-screen buffer (“pbuffer”)
  - Read back frame buffer into `byte[]`
  - Render BufferedImage using Java 2D™ API

Mixing Swing and 3D



New Experimental Work in Java SE 6 ("Mustang")

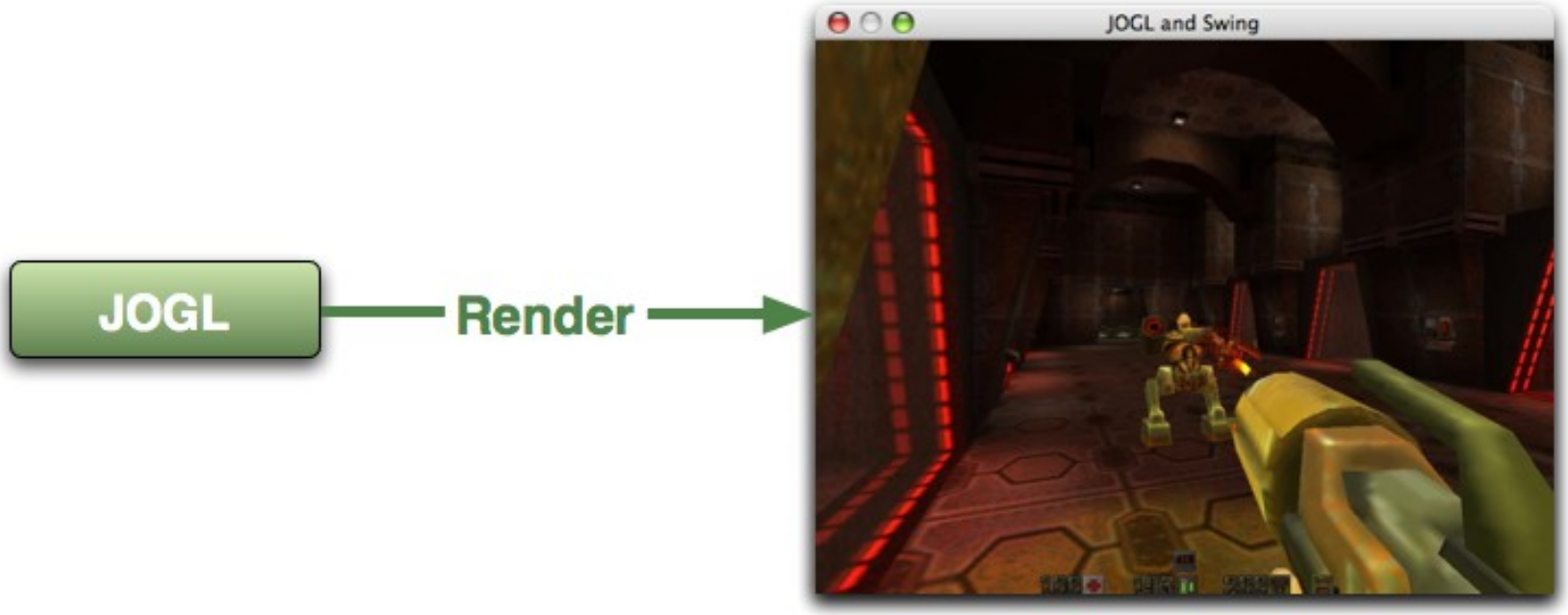
- Allow third-party libraries some access to internals of Java 2D/OpenGL® pipeline
- Access to OpenGL drawable, context, rendering thread
- Highly experimental; APIs not yet finalized
- More work to be done in Java SE 7 to standardize these APIs

Java 2D/JOGL Bridge

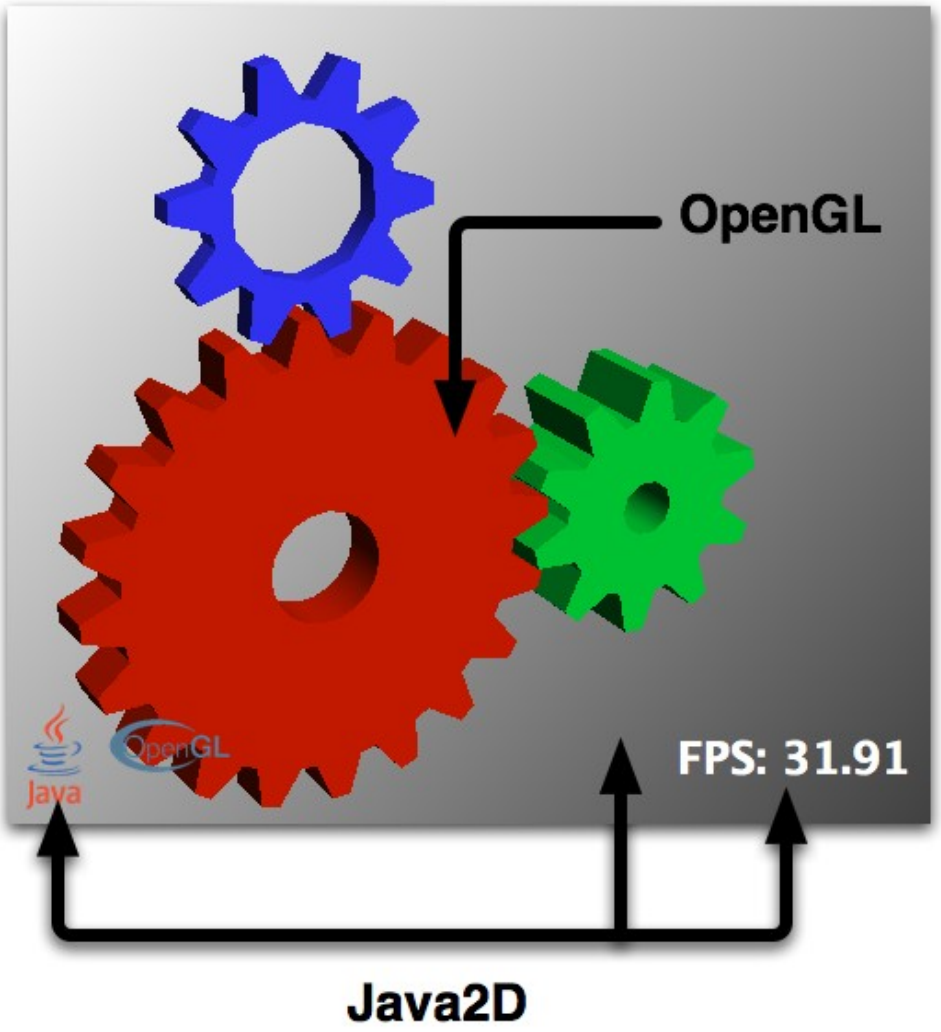
- GLJPanel now bridges to Java 2D/OpenGL pipeline when enabled
  - Much higher performance
  - Basically same speed as using heavyweight components
  - 100% correct Swing integration
  - No application changes
  - Windows and X11 support now; Mac OS X coming
- Implementation uses both experimental Java 2D APIs as well as new JSR 231 functionality
  - Interoperability with other OpenGL-based libraries

New GLJPanel: Cut Out the Middleman

- Render directly to Swing buffer



Demo



Agenda

Animation fundamentals

Into the third dimension

Putting it all together

Bring Life to Your Applications

- Life is restless
  - Transitions
  - Highlights
  - Progress Indicators
  - Motion
- Life is not flat
  - 3D visualization

Transitions

- When a value is changed
  - A label's text
  - A screen
- Fade in/out
  - Fade from/to a color (e.g., Fade to black)
  - Opacity change
- Cross-fade
  - Current value fades out
  - New value fades in

Fade to Black, Timing Code

```
cycle = new Cycle(1200, 10);
envelope = new Envelope(1, 0,
    RepeatBehavior.FORWARD, EndBehavior.HOLD);
fadeRange =
    PropertyRange.createPropertyRangeFloat(
        "fadeOut", 0.0f, 1.0f);
modifier = new ObjectModifier(this, fadeRange);
timer = new TimingController(cycle,
    envelope, modifier);
timer.setAcceleration(0.7f);
timer.setDeceleration(0.3f);
timer.start();
```

Fade to Black, Painting Code

```
public void setFadeOut(float fadeOut) {
  this.fadeOut = fadeOut;
  repaint();
}

protected void paintComponent(Graphics g) {
  g.setColor(
      new Color(0.0f, 0.0f, 0.0f, fadeOut));
  Rectangle r = g.getClipBounds();
  g.fillRect(r.x, r.y, r.width, r.height);
  // ...
}
```

Highlights

- Outline an interactive element
- Canonical effects
  - Brightness increase
  - Glow, Pulse
  - Spring
- Usually triggered by a user input

Highlights, Timing Code

```
public void mouseEntered(MouseEvent e) {
  if (timer != null && timer.isRunning()) {
    timer.stop();
  }
  range = PropertyRange.createPropertyRangeColor(
      "foreground", Color.WHITE);
  modifier = new ObjectModifier(button, range);
  timer =
      new TimingController(cycle, envelope, modifier);
  timer.start();
}
```

Progress Indicators

- Show that your UI is still alive
  - Otherwise the user might poke it with a stick
- Indeterminate progress indicators
  - Rotation
  - Glow/Pulse
- Short, repeated animations

Progress, Timing Code

```
cycle = new Cycle(800, 30);
envelope = new Envelope(
    TimingController.INFINITE, 0,
    RepeatBehavior.REVERSE, EndBehavior.RESET);
range = PropertyRange.createPropertyRangeFloat(
    "glow", 0.0f, 1.0f);
modifier = new ObjectModifier(this, range);
timer = new TimingController(cycle,
    envelope, modifier);
```

Progress, Painting Code

```
protected void paintComponent(Graphics g) {
  Graphics2D g2 = (Graphics2D) g;

  Composite composite = g2.getComposite();
  g2.setComposite(
      AlphaComposite.SrcOver.derive(glow));
  g2.drawImage(javaCupGlow, x, y, null);
  g2.setComposite(composite);

  g2.drawImage(javaCup, x, y, null);
}
```

Motion

- Helps the user understand what happened
  - When I miss a drop, the item goes back to its origin
  - When items change location, it is obvious
- No more “undo/redo” syndrome
- Realistic motion
  - Non-linear movements
- Implementation is simple
  - Use property setters
  - Use acceleration/deceleration

DEMO

Putting It All Together

Animated User Interfaces

- Use built-in properties
  - Foreground/background colors
  - Location, size
- Use advanced components
  - JXPanel from the SwingLabs project exposes an alpha property for easy fade in/out
- Keep animations short and simple
  - Do not bore the user!

Summary: Make Your Clients Rich

Swing

+

Animation

+

3D

Filthy Rich Clients

For More Information

- Sessions
  - TS-1548: Extreme GUI Makeover: Thursday, 2:45–3:45
  - Swing and 2D BOFS: Wednesday, 7:30–9:30
  - Desktop Futures Panel: Thursday, 7:30–8:30
- Websites
  - JOGL: http://jogl.dev.java.net
  - Timing Framework: http://timingframework.dev.java.net
  - SwingLabs: http://swinglabs.dev.java.net
- Articles, blogs
  - Romain's blog: http://www.jroller.com/page/gfx
  - Chet's blog: http://weblogs.java.net/blog/chet
  - Chris Campbell's blog: http://weblogs.java.net/blog/campbell
  - Timing Articles on java.net: "Timing is Everything", "Time Again"

Q&A

Filthy Rich Clients: Animated Effects in Swing Applications

Chet Haase
Kenneth Russell
Romain Guy

Sun Microsystems, Inc.

TS-1297