JavaFX Your Way

Building JavaFX Applications with Alternative Languages

Stephen Chin
GXS
steve@widgetfx.org
tweet: @steveonjava

Jonathan Giles
Oracle
Jonathan.giles@oracle.com
tweet: @jonathangiles

JavaOne

OR: Holy Cow – What Just happened to JavaFX Script!??!

Stephen Chin
GXS
steve@widgetfx.org
tweet: @steveonjava

Jonathan Giles
Oracle
Jonathan.giles@oracle.com
tweet: @jonathangiles

Meet the Presenters…

Steve

Jonathan

Disclaimer:
This is proof of concept

Disclaimer #2: This is code-heavy

Overall Presentation Goal

Demonstrate the future potential of the JavaFX platform.

Agenda

- Catch up on latest news
- JavaFX in Java
- Explore alternative languages
  - JRuby
  - Clojure
  - Groovy
  - Scala

Todays News

- JavaFX Script no longer required to write JavaFX applications
- Benefits:
  - Easier integration with business logic on JVM
  - Access to generics, annotations, (closures), etc
  - Java has great IDE support
- Downsides:
  - JavaFX Script was kind to us

JavaFX With Java

JavaFX in Java

- JavaFX API follows JavaBeans approach
- Similar in feel to other UI toolkits (Swing, etc)
- Researching approaches to minimize boilerplate

Binding

- Unquestionably the biggest JavaFX Script innovation
- Researching ways to incorporate into Java
- Will not be as succinct as JavaFX Script
- Genius' are deployed in underground labs pondering this right now

Observable Pseudo-Properties

- Supports watching for changes to properties
- Implemented via anonymous inner classes
- Maybe closures in the future

Observable Pseudo-Properties

```
Rectangle rect = new Rectangle();
rect.setX(40);
rect.setY(40);
rect.setWidth(100);
rect.setHeight(200);


rect.addChangedListener(Rectangle.HOVER, new BooleanListener() {



});
```

Observable Pseudo-Properties

```
Rectangle rect = new Rectangle();
rect.setX(40);
rect.setY(40);
rect.setWidth(100);
rect.setHeight(200);


rect.addChangedListener(Rectangle.HOVER, new BooleanListener() {



});
```

Before: ‘addChangingListener’
After: ‘addChangedListener’

Observable Pseudo-Properties

```
Rectangle rect = new Rectangle();
rect.setX(40);
rect.setY(40);
rect.setWidth(100);
rect.setHeight(200);


rect.addChangedListener(Rectangle.HOVER, new BooleanListener() {



});
```

The property we are wanting to watch

Observable Pseudo-Properties

```
Rectangle rect = new Rectangle();
rect.setX(40);
rect.setY(40);
rect.setWidth(100);
rect.setHeight(200);


rect.addChangedListener(Rectangle.HOVER, new BooleanListener() {



});
```

Listener for each primitive type, and Objects in general

Observable Pseudo-Properties

```
Rectangle rect = new Rectangle();
rect.setX(40);
rect.setY(40);
rect.setWidth(100);
rect.setHeight(200);


rect.addChangedListener(Rectangle.HOVER, new BooleanListener() {
  public void handle(Bean bean, PropertyReference pr, boolean oldHover) {

  }
});
```

Rectangle is a Bean

Observable Pseudo-Properties

```
Rectangle rect = new Rectangle();
rect.setX(40);
rect.setY(40);
rect.setWidth(100);
rect.setHeight(200);


rect.addChangedListener(Rectangle.HOVER, new BooleanListener() {
  public void handle(Bean bean, PropertyReference pr, boolean oldHover) {

  }
});
```

Refers to the Rectangle.hover ‘property’

Observable Pseudo-Properties

```
Rectangle rect = new Rectangle();
rect.setX(40);
rect.setY(40);
rect.setWidth(100);
rect.setHeight(200);


rect.addChangedListener(Rectangle.HOVER, new BooleanListener() {
  public void handle(Bean bean, PropertyReference pr, boolean oldHover) {

  }
});
```

For performance reasons, this is the old value.

Observable Pseudo-Properties

```
Rectangle rect = new Rectangle();
rect.setX(40);
rect.setY(40);
rect.setWidth(100);
rect.setHeight(200);


rect.addChangedListener(Rectangle.HOVER, new BooleanListener() {
  public void handle(Bean bean, PropertyReference pr, boolean oldHover) {
    rect.setFill(rect.isHover() ? Color.GREEN : Color.RED);
  }
});
```

Sequences in Java

- Sequence class available
  - Essentially an observable List
- Public API is still sequence-based
- Internal code can use lighter collections API

Example Application

```
public class HelloStage implements Runnable {

  public void run() {
    Stage stage = new Stage();
    stage.setTitle("Hello Stage");
    stage.setWidth(600);
    stage.setHeight(450);

    Scene scene = new Scene();
    scene.setFill(Color.LIGHTGREEN);

    stage.setScene(scene);
    stage.setVisible(true);
  }

  public static void main(String[] args) {
    FX.start(new HelloStage());
  }
}
```

Summary

- The JVM has a modern UI toolkit coming to it
- Total port to Java – no hacks or kludges
- Many languages to choose from
- Alternate languages == exciting possibilities
- Choose the best language for your needs

Major Question

How can alternative languages make developing JavaFX user interfaces easier & more productive?

JavaFX With JRuby

Why JRuby?

- Direct access to Java APIs
- Dynamic Typing
- Closures
  - ‘Closure conversion’ for interfaces

Java in JRuby

- Accessing Properties

```
timeline.setAutoReverse(true)
timeline.autoReverse = true
timeline.auto_reverse = true


timeline.getKeyFrames().add(kf)
timeline.key_frames.add(kf)
timeline.key_frames.add kf
```

JRuby Example 1: Simple Stage

```
require 'java'

FX = Java::javafx.lang.FX
Stage = Java::javafx.stage.Stage
Scene = Java::javafx.scene.Scene
Color = Java::javafx.scene.paint.Color

class HelloStage
  include java.lang.Runnable

  def run

  end
end

FX.start(HelloStage.new);
```

```
stage = Stage.new
stage.title = 'Hello Stage (JRuby)'
stage.width = 600
stage.height = 450

scene = Scene.new
scene.fill = Color::LIGHTGREEN
stage.scene = scene

stage.visible = true;
```

JRuby Example 2

```
rect = Rectangle.new
rect.x = 25
rect.y = 40
rect.width = 100
rect.height = 50
rect.fill = Color::RED
scene.content.add(rect)

timeline = Timeline.new
timeline.repeat_count = Timeline::INDEFINITE
timeline.auto_reverse = true
kv = KeyValue.new(rect, Rectangle::X, 200);
kf = KeyFrame.new(Duration.valueOf(500), kv);
timeline.key_frames.add kf;
timeline.play();
```

JRuby Closure Conversion

```
rect.add_changed_listener(Rectangle::HOVER) do |bean, pr, bln|
  rect.fill = rect.hover ? Color::GREEN : Color::RED;
end
```

JRuby Swiby

```
require 'swiby'

class HelloWorldModel
  attr_accessor :saying
end
model = HelloWorldModel.new
model.saying = "Hello World"

Frame {
  title "Hello World“
  width 200
  content {
    Label {
      text bind(model,:saying)
    }
  }
  visible true
}
```

JavaFX With Clojure

Artwork by Augusto Sellhorn http://sellmic.com/

A Little About Clojure

- Started in 2007 by Rich Hickey
- Functional Programming Language
- Derived from LISP
- Optimized for High Concurrency

```
(def hello (fn [] "Hello world"))
(hello)
```

- … and looks nothing like Java!

Clojure Syntax in One Slide

Symbols

- numbers – 2.178
- ratios – 355/113
- strings – “clojure”, “rocks”
- characters – \a \b \c \d
- symbols – a b c d
- keywords – :alpha :beta
- boolean – true, false
- null - nil

Collections

(commas optional)

- Lists

(1, 2, 3, 4, 5)

- Vectors

[1, 2, 3, 4, 5]

- Maps

{:a 1, :b 2, :c 3, :d 4}

- Sets

#{:a :b :c :d :e}

(plus macros that are syntactic sugar wrapping the above)

Clojure GUI Example

```
(defn javafxapp []
  (let [stage (Stage. "JavaFX Stage")
        scene (Scene.)]
    (.setFill scene Color/LIGHTGREEN)
    (.setWidth stage 600)
    (.setHeight stage 450)
    (.setScene stage scene)
    (.setVisible stage true)))
(javafxapp)
```

Clojure GUI Example

Create a Function for the Application

```
(defn javafxapp []
  (let [stage (Stage. "JavaFX Stage")
        scene (Scene.)]
    (.setFill scene Color/LIGHTGREEN)
    (.setWidth stage 600)
    (.setHeight stage 450)
    (.setScene stage scene)
    (.setVisible stage true)))
(javafxapp)
```

Clojure GUI Example

```
(defn javafxapp []
  (let [stage (Stage. "JavaFX Stage")
        scene (Scene.)]
    (.setFill scene Color/L
    (.setWidth stage 600)
    (.setHeight stage 450)
    (.setScene stage scene)
    (.setVisible stage true)))
(javafxapp)
```

Initialize the Stage and Scene Variables

Clojure GUI Example

```
(defn javafxapp []
  (let [stage (Stage. "Jav
        scene (Scene.)]
    (.setFill scene Color/LIGHTGREEN)
    (.setWidth stage 600)
    (.setHeight stage 450)
    (.setScene stage scene)
    (.setVisible stage true)))
(javafxapp)
```

Call Setter Methods on Scene and Stage

Clojure GUI Example

```
(defn javafxapp []
  (let [stage (Stage. "JavaFX Stage")
        scene (Scene.)]
    (.setFill scene Color/LIGHTGREEN)
    (.setWidth stage 600)
    (.setHeight stage 450)
    (.setScene stage scene)
    (.setVisible stage true)))
(javafxapp)
```

Java Constant Syntax

Java Method Syntax

Simpler Code Using **doto**

```
(defn javafxapp []
  (let [stage (Stage. "JavaFX Stage")
        scene (Scene.)]
    (doto scene
      (.setFill Color/LIGHTGREEN))
    (doto stage
      (.setWidth 600)
      (.setHeight 450)
      (.setScene scene)
      (.setVisible true))))
(javafxapp)
```

Simpler Code Using **doto**

```
(defn javafxapp []
  (let [stage (Stage. "JavaFX Stage")
        scene (Scene.)]
    (doto scene
      (.setFill Color/LIGHTGREEN))
    (doto stage
      (.setWidth 600)
      (.setHeight 450)
      (.setScene scene
      (.setVisible tru
(javafxapp)
```

doto form:

```
(doto symbol
   (.method params))
```

equals:

```
(.method symbol params)
```

Refined Clojure GUI Example

```
(defn javafxapp []
  (doto (Stage. "JavaFX Stage")
    (.setWidth 600)
    (.setHeight 450)
    (.setScene (doto (Scene.)
      (.setFill Color/LIGHTGREEN)
      (.setContent (list (doto (Rectangle.)
        (.setX 25)
        (.setY 40)
        (.setWidth 100)
        (.setHeight 50)
        (.setFill Color/RED))))))
    (.setVisible true)))
(javafxapp)
```

Refined Clojure GUI Example

```
(defn javafxapp []
  (doto (Stage. "JavaFX Stage")
    (.setWidth 600)
    (.setHeight 450)
    (.setScene (doto (Scene.)
      (.setFill Color/LIGHTGREEN)
      (.setContent (list (doto (Rectangle.)
        (.setX 25)
        (.setY 40)
        (.setWidth 100)
        (.setHeight 50)
        (.setFill Color/RED))))))
    (.setVisible true)))
(javafxapp)
```

Let replaced with inline declarations

Refined Clojure GUI Example

```
(defn javafxapp []
  (doto (Stage. "JavaFX Stage")
    (.setWidth 600)
    (.setHeight 450)
    (.setScene (doto (Scene.)
      (.setFill Color/LIGHTGREEN)
      (.setContent (list (doto (Rectangle.)
        (.setX 25)
        (.setY 40)
        (.setWidth 100)
        (.setHeight 50)
        (.setFill Color/RED))))))
    (.setVisible true)))
(javafxapp)
```

Doto allows nested data structures

Refined Clojure GUI Example

```
(defn javafxapp []
  (doto (Stage. "JavaFX Stage")
    (.setWidth 600)
    (.setHeight 450)
    (.setScene (doto (Scene.)
      (.setFill Color/LIGHTGREEN)
      (.setContent (list (doto (Rectangle.)
        (.setX 25)
        (.setY 40)
        (.setWidth 100)
        (.setHeight 50)
        (.setFill Color/RED)))))))
    (.setVisible true)))
(javafxapp)
```

Now a nested Rectangle fits!

Closures in Clojure

- Inner classes can be created using `proxy`

```
(.addChangeListener rect Rectangle/HOVER
  (proxy [BooleanListener] []
    (handle [b, p, o]
      (.setFill rect
        (if (.isHover rect) Color/GREEN Color/RED)))))
```

Closures in Clojure

- Inner classes can be created using proxy

Proxy form:

```
(proxy [class] [args] fs+)
f => (name [params*] body)
```

```
(.addChangeListener rect Rectangle/HOVER
  (proxy [BooleanListener] []
    (handle [b, p, o]
      (.setFill rect
        (if (.isHover rect) Color/GREEN Color/RED)))))
```

JavaFX With Groovy

Features of Groovy

- Tight integration with Java
  - Very easy to port from Java to Groovy
- Declarative syntax
  - Familiar to JavaFX Script developers
- Builders

Example 1: Simple FX Script to Groovy

Step 1: Lazy conversion to Groovy

```
class HelloStage implements Runnable {

  void run() {
    Stage stage = new Stage();
    stage.setTitle("Hello Stage (Groovy)“);
    stage.setWidth(600);
    stage.setHeight(450);

    Scene scene = new Scene();
    scene.setFill(Color.LIGHTSKYBLUE);
    stage.setScene(scene);

    stage.setVisible(true);
  }

  static void main(args) {
    FX.start(new HelloStage());
  }
}
```

Step 2: Slightly More Groovy

```
class HelloStage implements Runnable {

    void run() {
        new Stage(
            title: "Hello Stage (Groovy)",
            width: 600,
            height: 450,
            visible: true,
            scene: new Scene(
                fill: Color.LIGHTSKYBLUE,
            )
        );
    }

    static void main(args) {
        FX.start(new HelloStage());
    }
}
```

Slight Aside: Groovy Builders

- Groovy builders make writing custom DSLs easy
- For the next slide, I am using a builder I defined
- Hopefully the community will improve upon this

Step 3: Using a Groovy Builder

```
FxBuilder.build {
    stage = stage(
        title: "Hello World",
        width: 600,
        height: 450,
        scene: scene(fill: Color.LIGHTSKYBLUE) {
            ...
        }
    )

    stage.visible = true;
}
```

Step 4: With Content

```
FxBuilder.build {
    stage = stage(
        title: "Hello Rectangle (Groovy FxBuilder 2)",
        width: 600,
        height: 450,
        scene: scene(fill: Color.LIGHTSKYBLUE) {
            rectangle(
                x: 25, y: 40,
                width: 100, height: 50,
                fill: Color.RED
            )
        }
    )

    stage.visible = true;
}
```

Example 2: FX Script Animation in Groovy

Step 1: JavaFX Script

```
def timeline = Timeline {
  repeatCount: Timeline.INDEFINITE
  autoReverse: true
  keyFrames: [
    KeyFrame {
      time: 750ms
      values : [
        rect1.x => 200.0 tween Interpolator.LINEAR,
        rect2.y => 200.0 tween Interpolator.LINEAR,
        circle1.radius => 200.0 tween Interpolator.LINEAR
      ]
    }
  ];
}

timeline.play();
```

Step 1a: JavaFX Script Simplification

```
def timeline = Timeline {
  repeatCount: Timeline.INDEFINITE
  autoReverse: true
  keyFrames: [
    at (750ms) {
      rect1.x => 200.0 tween Interpolator.LINEAR;
      rect2.y => 200.0 tween Interpolator.LINEAR;
      circle1.radius => 200.0 tween Interpolator.LINEAR;
    }
  ];
}

timeline.play();
```

Step 2: Java-ish Groovy Animations

```
final Timeline timeline = new Timeline(
  repeatCount: Timeline.INDEFINITE,
  autoReverse: true
)

final KeyValue kv1 = new KeyValue (rect1, Rectangle.X, 200);
final KeyValue kv2 = new KeyValue (rect2, Rectangle.Y, 200);
final KeyValue kv3 = new KeyValue (circle1, Circle.RADIUS, 200);

final KeyFrame kf = new KeyFrame(Duration.valueOf(750), kv1, kv2, kv3);

timeline.getKeyFrames().add(kf);

timeline.play();
```

Step 3: JavaFX Animation in Groovy (Using Builders)

```
timeline = timeline(repeatCount: Timeline.INDEFINITE, autoReverse: true) {
  keyframes {
    keyframe(time: 750) {
      keyvalue(target: rect1, property: Rectangle.Y, endValue: 200);
      keyvalue(target: rect2, property: Rectangle.X, endValue: 200);
      keyvalue(target: circle1, property: Circle.RADIUS, endValue: 200);
    }
  }
}

timeline.play();
```

Groovy Closures

- With interface coercion

```
def f = {
  bean, pr, bln -> rect.setFill(rect.isHover() ? Color.GREEN : Color.RED);
} as BooleanListener;

rect.addChangedListener(Rectangle.HOVER, f);
```

Groovy Closures

- Interfaces with multiple methods

```
def keyValueTarget = [
  getType: { Type.FLOAT },
  unwrap: { this },
  getValue: { Float.valueOf(rect.getX()) },
  setValue: { Object value -> rect.setX((Float)value) }
] as KeyValueTarget;
```

JavaFX With Scala

What is Scala

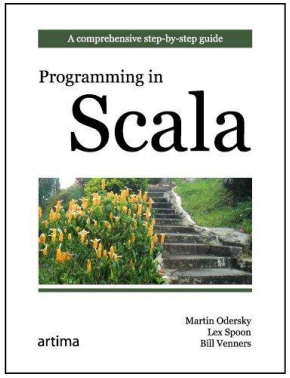
2001
- Scala Started

2003/2004
- Scala v1.0

2006
- Scala v2.0

2010
- Scala 2.8.0 (latest)

- Started in 2001 by Martin Odersky
- Compiles to Java bytecodes
- Pure object-oriented language
- Also a functional programming language



Why Scala?

- Shares many language features with JavaFX Script that make GUI programming easier:
  - Static type checking – Catch your errors at compile time
  - Closures – Wrap behavior and pass it by reference
  - Declarative – Express the UI by describing what it should look like
- Scala also supports DSLs!

Java vs. Scala DSL

```
public class HelloStage implements Runnable {

  public void run() {
    Stage stage = new Stage();
    stage.setTitle("Hello Stage");
    stage.setWidth(600);
    stage.setHeight(450);
    Scene scene = new Scene();
    scene.setFill(Color.LIGHTGREEN);
    Rectangle rect = new Rectangle();
    rect.setX(25);
    rect.setY(40);
    rect.setWidth(100);
    rect.setHeight(50);
    rect.setFill(Color.RED);
    stage.add(rect);
    stage.setScene(scene);
    stage.setVisible(true);
  }

  public static void main(String[] args) {
    FX.start(new HelloStage());
  }
}
```

22 Lines
545 Characters

```
object HelloJavaFX extends JavaFXApplication {
  def stage = new Stage {
    title = "Hello Stage"
    width = 600
    height = 450
    scene = new Scene {
      fill = Color.LIGHTGREEN
      content = List(new Rectangle {
        x = 25
        y = 40
        width = 100
        height = 50
        fill = Color.RED
      })
    }
  }
}
```

17 Lines
324 Characters

```
object HelloJavaFX extends JavaFXApplication {
  def stage = new Stage {
    title = "Hello Stage"
    width = 600
    height = 450
    scene = new Scene {
      fill = Color.LIGHTGREEN
      content = List(new Rectangle {
        x = 25
        y = 40
        width = 100
        height = 50
        fill = Color.RED
      })
    }
  }
}
```

```
object HelloJavaFX extends JavaFXApplication {
  def stage = new Stage {
    title = "Hello Stage"
    wi
    he
    sc
      fill = Color.LIGHTGREEN
      content = List(new Rectangle {
        x = 25
        y = 40
        width = 100
        height = 50
        fill = Color.RED
      })
    }
  }
}
```

Base class for JavaFX applications

```
object HelloJavaFX extends JavaFXApplication {
  def stage = new Stage {
    title = "Hello Stage"
    width = 600
    height = 450
    scene = new Scene {
      fill = Color.LIGHTGREEN
      content = List(new Rectangle {
        x = 25
        y = 40
        width = 100
        height = 50
        fill = Color.RED
      })
    }
  }
}
```

Declarative Stage definition

```
object HelloJavaFX extends JavaFXApplication {
  def stage = new Stage {
    title = "Hello Stage"
    width = 600
    height = 450
    scene = new Scene {
      fill = Color.LIGHTGREEN
      content = List(new Rectangle {
        x = 25
        y = 40
        width = 100
        height = 50
        fill = Color.RED
      })
    }
  }
}
```

Inline property definitions

```
object HelloJavaFX extends JavaFXApplication {
  def stage = new Stage {
    title = "Hello Stage"
    width = 600
    height = 450
    scene = new Scene {
      fill = Color.LIGHTGREEN
      content = List(new Rectangle {
        x = 25
        y = 40
        width = 100
        height = 50
        fill = Color.RED
      })
    }
  }
}
```

List Construction Syntax

Animation in Scala

```
def timeline = new Timeline {
    repeatCount = INDEFINITE
    autoReverse = true
    keyFrames = List(
      new KeyFrame(50) {
        values = List(
          new KeyValue(rect1, Rectangle.X -> 300),
          new KeyValue(rect2, Rectangle.Y -> 500),
          new KeyValue(rect2, Rectangle.Width -> 150)
        )
      }
    )
  }
```

Animation in Scala

```
def timeline = new Timeline {
    repeatCount = INDEFINITE
    autoReverse = true
    keyFrames = List(
      new KeyFrame(50) {
        values = List(
          new KeyValue(rect1, Rectangle.X -> 300),
          new KeyValue(rect2, Rectangle.Y -> 500),
          new KeyValue(rect2, Rectangle.WIDTH -> 150)
        )
      }
    )
  }
```

Duration set by Constructor Parameter

Animation in Scala

```
def timeline = new Timeline {
    repeatCount = INDEFINITE
    autoReverse = true
    keyFrames = List(
      new KeyFrame(50) {
        values = List(
          new KeyValue(rect1, Rectangle.X -> 300),
          new KeyValue(rect2, Rectangle.Y -> 500),
          new KeyValue(rect2, Rectangle.WIDTH -> 150)
        )
      }
    )
  }
```

Operator overloading for animation syntax

Closures in Scala

- Closures are also supported in Scala
- And they are 100% type-safe

```
rect.addChangedListener(Node.HOVER, (b, p, o) => {
  rect.fill = if (rect.hover) Color.GREEN else Color.RED
})
```

Closures in Scala

- Closures are also supported in Scala
- And they are 100% type-safe

```
rect.addChangedListener(Node.HOVER, (b, p, o) => {
  rect.fill = if (rect.hover) Color.GREEN else Color.RED
})
```

Compact syntax
(params) => {body}

Conclusion

- JavaFX as Java APIs is great
- Usable in alternate languages
- Over time improved support is possible
  - E.g. Groovy builders, Scala DSL

Remember: This is a proof of concept only – you can not leave this session and do this today.

JavaOne

Thank You

Stephen Chin
steve@widgetfx.org
tweet: @steveonjava

Jonathan Giles
Jonathan.giles@oracle.com
tweet: @jonathangiles