



JavaFX™ for Java, JavaScript programmers



Agenda

- Introduction
- JavaFX programming language – big picture
- Compare and contrast JavaFX, Java and JavaScript language features to **learn JavaFX**

Introduction

- Audience assumption
 - Knowledge of Java
 - Knowledge of JavaScript
- Covering **only the language aspects** of JavaFX
- Please attend other talks to learn how to create create stunning multimedia/GUI applications
- Language comparison is a great way to learn a new programming language

JavaFX - big picture

- “scripting language” - but statically typed
- Type inferred rather than specified always
- “global” functions, variables in addition to classes and methods
- Multiple inheritance (may change to mixins!)
- Object Literals - create/initialize a new object
- Bind - simpler way to do observable/observer
- Sequences - flat, dynamic list with interesting operations

Comments

- Java

 - `// this is single line comment`

 - `/* this is multi-line comment */`

 - `/** this is doc-comment */`

- JavaScript

 - `// this is single line comment`

 - `/* this is multi-line comment */`

- JavaFX

 - Nothing new! Same as Java!

Variables, Constants

- Java

```
String str = "hello"; // Type name = init_value
```

```
final double PI = 3.14; // "final" keyword => unmodifiable
```

- JavaScript

```
var str = "hello"; // "var" is a keyword
```

```
const PI = 3.14; // "const" is a keyword
```

- JavaFX

```
var str = "hello"; // "var" is a keyword
```

```
var str : String = "hello"; // optionally specify type
```

```
def PI = 3.14; // "def" keyword => unmodifiable
```

Primitive Types

- Java
 - char, byte, short, int, long, float, double, boolean
 - void (as return type)
- JavaScript
 - Number, boolean, string, null, undefined
- JavaFX
 - Value types – non-null default value, immutable types
 - String, Number, Integer, Boolean, Duration
 - Void (as return type)

String Literals

- Java
 - “This is a String”
- JavaScript
 - 'single-quoted string'
 - “double-quoted string”
- JavaFX
 - “double quoted like Java”
 - 'single quoted too works like JavaScript!'
 - “x is {x}” // string with embedded expressions

Other Literals

- Java
 - 22, 3.14, 2.41e10, 23.2e-2, 'c', true, false, null
- JavaScript
 - Similar to Java. Adds **undefined**
- JavaFX
 - Similar to Java
 - Duration Literals (of type Duration)
 - 2ms** // 2 milliseconds
 - 5s** // 5 seconds

Operators

- The usual suspects: +, -, *, /, ++, -- etc.
 - Reminder is “**mod**” and not % (unlike Java, JavaScript)
- Usual comparisons: <, >, <=, >=, ==, !=
- No conditional operator. Instead there are **more expressions than statements**. Use “if” expression

```
var v = if (x mod 2 == 0) “even” else “odd”;
```

```
// instead of the Java, JavaScript expression
```

```
// (x % 2 == 0)? “even” : “odd”
```

Operators - contd.

- Logical (short-circuit) and, or are keyword operators

// instead of the &&, || as in Java and JavaScript,

```
var inRange : Boolean = (x >= 0) and (x <= 100);
```

```
var done : Boolean = (timeRemaining <= 0ms) or  
cancelled;
```

- Logical not is also a keyword operator

// instead of ! In Java, JavaScript,

```
var notInRange = not ((x >= 0) and (x <= 100));
```

Operators - contd.

- Assignment =, +=, -=, *=, /=
- Dynamic type check – **instanceof**
 - Java, JavaFX – same “**instanceof**” operator
 - JavaScript - “**typeof**” operator to differentiate between primitives, object, function, array etc. And “**instanceof**” to check against object types.
- Cast operation – not applicable to JavaScript
 - Java: **ArrayList** al = (**ArrayList**) list;
 - JavaFX: **var** al : **ArrayList** = list **as** ArrayList;
- No bitwise, no shift operators

Operators - contd.

- No String + (concatenation) operator as in Java and JavaScript
- But, we can use embedded expressions in JavaFX
// instead of “hello, “ + name, we use
var welcome = “hello, {str}”
- String expressions support format specifiers (like `java.util.Formatter`)
- Support for localization

Expressions

- JavaFX is an expression rich language
- “**if**” is expression – use it for conditional expression as well as “if” statement

```
var str = if (x mod 2 == 0) then “even” else “odd”
```

- **while** expression
- **break**, **continue** expression
- **return** expression
- **try** .. **catch** .. **finally** expression
- **throw** expression

Expressions - contd.

- JavaFX does not have labeled break and continue
- JavaFX does not have **switch...case** statements
- No **do...while** statements
- Block are expressions

// block is evaluated and last expression

// "33" is assigned to "v"

```
var v = { println("hello"); 33 }
```

Functions

- Java

No free standing functions, use static methods

- JavaScript

```
function Hello() {  
    alert("hello"); // alert is built-in  
}
```

- JavaFX

```
function Hello(str) {  
    println("hello"); // println is built-in  
}
```

Functions - contd.

- JavaScript

```
function add(x, y) {  
    return x + y;  
}
```

- JavaFX

```
function add(x : Number, y : Number) : Number {  
    return x + y;  
}
```

Functions - contd.

- JavaScript – anonymous functions, function values

// function valued variable

```
var func = function (x) { alert ("x = " + x); }
```

```
func(10);
```

- JavaFX

// function valued variable

```
var func = function (x) { println("x = {x}"); }
```

```
func(10);
```

Classes

- Java

```
class Person {  
    private String name; // instance var  
    private String likes; // instance var  
    public void hello() {  
        System.out.println("Hello, I'm " + name);  
    }  
}
```

Classes - contd.

- JavaScript - No classes, use constructors.

```
function Person(name, likes) {
```

```
    this.name = name; // introduce "name" instance var
```

```
    this.likes = likes; // introduce "likes" instance var
```

```
}
```

```
Person.prototype.hello = function() {
```

```
    alert("Hello, I'm " + this.name);
```

```
}
```

Classes - contd.

- JavaFX

```
class Person {
```

```
// instance variables are variables inside classes
```

```
var name : String;
```

```
var likes: String;
```

```
// methods are functions with classes
```

```
public function hello() {
```

```
    println("Hello, I'm {name}");
```

```
}
```

```
}
```

Object Creation

- Java

```
List list = new ArrayList(10);
```

- Use “new” and call constructor together.
- Additional initialization through “setters”

```
Person p = new Person();
```

```
p.setName(“Sundar”);
```

```
p.setLikes(“JavaFX”); // and so on
```

- Or use static methods of some factory class

Object Creation

- JavaScript
 - Java style creation – call constructor function with appropriate arguments.
- Object literals – a series of initialization within “{” “}”

```
var v = new Date();
```

```
var p = {  
    name: "Sundar", // initialize "name" property  
    likes: "JavaFX" // initialize "likes" property  
};
```

Object Creatation - JavaFX

- Java-style creation - Mostly used with Java classes

```
var a = new ArrayList(20);
```

- Object Literals

```
var p = Person {
```

```
    // initialize each "property" here
```

```
    name: "Sundar", // similar p.setName("Sundar")
```

```
    likes: "JavaFX"
```

```
};
```

Inheritance

- Java
 - Single inheritance “**extends**”
 - No multiple implementation inheritance
 - Interfaces are pure abstract – no implementation
 - Multiple interface inheritance “**implements**”

```
class Employee extends Person
    implements Serializable, Runnable {
    // implementation here..
}
```

Inheritance - contd.

- JavaScript
 - Single inheritance by prototype chain
 - Every object has a prototype object. So, we have a prototype chain for every object.
 - We can simulate classical OO with this. See also <http://www.crockford.com/javascript/inheritance.html>
 - JavaScript 2 is adding classes and inheritance as first class constructs!

Inheritance - contd.

- JavaFX

```
class Employee extends Person, Serializable, Runnable {  
    // implementation here..  
}
```

- No explicit interfaces. Just use abstract class with abstract methods. Multiple concrete classes may be extended
- This may change in near future (Mixins?)

Method Overriding

- Java

```
class Person {  
    public String getName() {  
        return this.name;  
    }  
  
    // more code  
}
```

```
class Graduate extends Person {  
    @Override // optional annotation to flag is it overriding!  
    public String getName() { .....
```

Method Overriding - JavaFX

- In JavaFX, we use **override** keyword

```
class Graduate extends Person {  
    override function getName() : String {  
        .....  
    }  
}
```

Calling super method

- Java

`super.getName();`

- JavaFX

- Multiple super classes and so “super” can't be used
- `SuperClassName.getName();`
- It looks like as though we are calling a static method of super class. But, JavaFX does not have static fields, methods – and so no confusion.

Getter, Setter

- In Java, we make the instance variable private and add public “setFoo” and “getFoo” methods.
- JavaScript supports `__defineGetter__`, `__defineSetter__`
- In JavaFx, we implement “on replace” triggers

```
\var name : String on replace {  
    println(“setting name “);  
};
```

```
// setting “name” triggers on replace block
```

```
var p = Person { name: “Sundar” }
```

Package, Import

- Java
 - Define “**package** com.acme;” as first line in every compilation unit
 - The caller imports by
 - import** java.awt.*; // import on-demand
 - import** java.util.ArrayList; // specific import
 - import static** java.lang.Math.*; // static import

Package, Import - contd.

- JavaScript
 - No package, import concept in the language.
 - Can use object for that purpose. i.e., define all your library functions, variables in an object.
 - It is possible to use libraries like “dojo” to define packages/namespaces and use those with “imports”.

Package, Import - contd.

- JavaFX

- Package declaration is same as Java

- Imports

```
import java.util.ArrayList; // specific import
```

```
import javafx.scene.*; // import on-demand
```

```
import java.lang.Math.PI; // import static
```

```
import java.lang.Math.*; // import static on-demand
```

Globals?

- Java
 - No globals. Closest is static fields and methods.
 - Caller always qualifies with class name – `Math.PI`, `System.out`
- JavaScript
 - Global variables and functions live (and compete) in the same scope. Need to take care to avoid collision of globals from different files.
- JavaFX
 - File level variables and functions can be defined.
 - But, caller always qualifies with file name (or imports specifically).

bind

- Event listeners or in general observer/observable is implemented in Java as follows:
 - Observable maintains of observers to notify
 - Observable calls a specific listener method on observer whenever change occurs
 - Observer in turn re-computes it's dependent value based on new value notified
- GUI programs are full of event listeners
- Also, “model” changes are tracked by “view” by observer/observable

What is bind?

```
var v = 10;
```

```
def x = bind v * v;
```

```
// whenever "v" changes, "x" automatically
```

```
// changes to v * v
```

```
// In general,
```

```
def x = bind someExpression;
```

```
// means when someExpression changes, x
```

```
// will be updated to match
```

bind - contd.

```
var x = 10;
```

```
var v = bind 2 * x; // v is 20
```

```
x = 11; // v becomes 2*11 = 22
```

- There is also reverse “bind” (aka “bi-directional” bind)

```
var x = 10;
```

```
var z = bind x with inverse; // z is also 10
```

```
x = 11; // z is 11 now
```

```
z = 34; // x is 34 now - because it is bidirectional
```

bind - bound functions

- Consider binding function call expression

```
var x = 20;
```

```
var offset = 2;
```

```
function func(a) { return a + offset }
```

```
var v = bind func(x);
```

```
// when "x" changes, "v" is automatically recomputed
```

```
// by calling "func" with new "x" value
```

```
x = 3; // "v" is re-computed and becomes 3 + 2 = 6.0
```

bind - bound functions

- Consider no argument changes - but something else used by function "func" changes.

```
var offset = 2;
```

```
// function func(a) { return a + offset; }
```

```
// if we want "var v" to re-compute even when
```

```
// no argument change but "offset" changes, then
```

```
bound function func(a) { return a + offset; }
```

```
var v = bind func(x);
```

```
// now "v" is re-computed when ever "x" changes or
```

```
// whenever "offset" (used by 'func') changes
```

Sequences

- In Java, arrays are fixed length. `array.length` returns length. Arrays can be nested (can have arrays within arrays within arrays and so on)
- In JavaScript arrays are dynamic. We can add/remove elements to/from array. `array.length` returns current length of the array.
- In JavaFX, we use sequences. Sequences are dynamic, flat lists. Like JavaScript, sequence length is dynamic. Unlike Java and JavaScript, sequences are flat. There are no nested sequences.

Sequences - contd.

- Java

```
String[] args = { "hello", "world", "is", "great!" };
```

- JavaScript

```
var args = [ "hello", "world", "is", "great!" ];
```

- JavaFX

```
// optionally specify type of sequence like..
```

```
var args : String[] = [ "hello", "world", "is", "great!" ];
```

- Sequence element access syntax is same as Java and JavaScript array element access - **array[index]**

Sequences - contd.

- JavaFX range sequence

```
var seq = [1..10]; // sequence containing 1 to 10
```

```
var seq1 = [1..<10]; // sequence containing 1 to 9
```

```
var seq2 = [0..100 step 5]; // sequence 0, 5, 10.... 100
```

- Length of the sequence

```
// We use seq.length in Java and JavaScript to get length
```

```
// of arrays. In JavaFX, we use "sizeof" operator
```

```
var len = sizeof seq;
```

Sequences - contd.

- Inserting into sequence

```
var seq = [ "hello" ];
```

```
insert "world" into seq; // "seq" is ["hello", "world"]
```

- Deleting from sequence

```
var seq = [ "hello", "world", "hello" ];
```

```
delete "hello" from seq; // "seq" becomes ["world"]
```

- Deleting all elements from sequence

```
delete seq; // "seq" becomes empty sequence
```

Sequences - contd.

- Sequences are flat
- Inserting a sequence into another sequence results in a flat sequence

```
var s = [0..10];
```

```
var s1 = [11..100];
```

```
insert s1 into s; // makes "s" as flat sequence of [0..100]
```

- Filtering

```
def seq = [1..100];
```

```
// select only even numbers
```

```
def selected = seq[x | (x mod 2) == 0];
```

For Expressions

- Java “enhanced for-loop”

// where “args” is an array or an Iterable

```
for (String str : args) { .... }
```

- JavaScript for loop

```
for (v in array) { .. access array[v] ... }
```

- Works for arrays as well as objects
- Each time “v” gets property name or index (and not the n'th item in array!)
- Java and JavaScript support

```
for (<init>;<condition>;<increment>) style loop as well
```

For Expressions - JavaFX

- Similar to Java's enhanced for..loop

```
for (v in seq) {  
    //v is n'th element  
    println(v);  
}
```

- If you want to know the index of the current element, you can use “**indexof**” operator

```
for (v in seq) { println(indexof v); }
```

- No Java, JavaScript style “counting” for..loop

For Expressions - contd.

- “nested” looping

```
for (x in [0..3], y in [0..3]) {  
    // prints 0,0 0,1 0,2 .. 3,1, 3,2 3,3  
    println("{x}, {y}");  
}
```

- Optional “**where**” clause in for expressions

```
for (x in [0..3], y in [0..3] where x == y) {  
    // prints only 0, 0 1,1 2,2 3,3  
    println("{x}, {y}");  
}
```

Misc. - not covered

- Access modifiers – public, protected etc. Similar to Java except for few differences (see language spec)
- Tween expressions for animation
- “**init**” block in a class – executed as last step of instance initialization
- “**postinit**” block in a class – executed after instance initialization is completed

References

- JavaFX language
 - <http://openjfx.java.sun.com/current-build/doc/ref>
- Java Programming Language
 - <http://java.sun.com/docs/books/jls/>
- JavaScript 1.5 reference
 - https://developer.mozilla.org/En/Core_JavaScript_1.5_Reference_Guide
- JavaScript 2
 - <https://developer.mozilla.org/presentations/xtecl>