

## 2. JavaFX

### 2.1. List of Important Features:

#### 2.1.1 Incremental dependency-based Evaluation (Bind)

Attribute values can be declared to be dependent on (bound to) expressions involving other attributes. Thus, when a referenced attribute changes its value all attributes directly or indirectly dependent on them will change their values accordingly with no intervening procedural logic. This is similar to the way formula cells in a spreadsheet change their values whenever one or more cells they depend upon change their values. It is especially useful for GUI development where maintaining model and view attributes in sync usually requires complex procedural logic.

Consider the following Model/View type example, the value of the View's *title* is dependent upon the Model's *saying* attribute. This dependency is bi-directional.

```
import javafx.ui.*;

class HelloWorldModel {
    attribute saying: String;
}
var model = HelloWorldModel {
    saying: "Hello World"
};
Frame {
    title: bind "{model.saying} JavaFX"
    width: 200
    content: TextField {
        value: bind model.saying
    }
    visible: true
};
```

Initially the *title* and TextField's *value* are set by model's *saying* attribute. When the user updates the TextField's *value* the model's *saying* attribute is automatically updated (which in turn updates the *title*).

In the next example we show how to bind a variable to an expression involving other variables.

```
var ItemCost = 50;
var NumberOfItems = 10;
var Total = bind ItemCost * NumberOfItems;
assert Total == 500;
```

In the code above, whenever the value of *ItemCost* or *NumberOfItems* changes the *Total* is updated. Binding with expressions is unidirectional so change in *Total* will not reflect on the variables bound to it.

## Bind with Functions Vs. Operation

The bind operator works well with functions too. Functions fall in the purely functional aspect of JavaFX. The body of a function may only contain variable declarations and a return statement. Conditional operations or loops are not allowed inside the body of a function. Because functions incrementally update their results whenever either their arguments or referenced variables change, binding to a function works just as well as binding to a single attribute. In fact, functions really are designed to be used with bindings. The following is an example of binding with a function.

```
/*TO DO: Add sample code*/
```

Operations are more like class methods of Java. Unlike functions, operations may contain any number of statements such as conditional statements, looping statements, `try` and `catch` statements, and so on. The body of a `function` is always incrementally evaluated without requiring the `bind` operator, however the body of an `operation` is not. Unlike a function, changes to local variables, inside an operation, do not trigger incremental evaluation. Incremental evaluation is not performed inside the body of an `operation` except for expressions explicitly prefixed by `bind`.

Although when you call an operation from an incremental evaluation context the call itself is incrementally evaluated. This means that if the values of any arguments to the call change a new call to that operation will be made and a new value is returned.

By contrast, `function`'s are only called once and the result of the evaluation is incorporated into the callers evaluation tree.

## Compiled JavaFX Changes

JavaFX is being converted from the interpreter to the compiler and some changes to features and syntax are taking place in the process.

1. The language used to differentiate between functions and operations (procedures). The current syntax merges these two concepts into just one: `function`. Functions are no longer restricted to variable declarations and return expressions, and basically old-style operations with a new name.
2. If you want to use the bidirectional binding feature, you have to use new 'with inverse' syntax in compiled JavaFX.
3. If a declared attribute had an initial value, it was set outside of the class body. Attribute initial values are now set as part of the declaration, like Java. (also `functions/operations defined within class?`)
4. Replace triggers were defined outside of the class body. Replace triggers are now part of the attribute declaration. Also, the syntax has changed, such that the trigger function now follows the keywords "on replace".