Object oriented programming with COBOL

Robert Sales
Technical Architect
What is COBOL?

- **ANS85 COBOL**
  - Remains the current standard for structured procedural COBOL programming.
  - Some features rarely used (nested programs?), but widely accepted and respected as an industry standard

- **Mainframe variants**

- **Micro Focus, ACU, RM, Realia etc. etc.**

- **ISO2002**
What is COBOL’s future?

- Accept that COBOL is essentially a procedural programming language or...

- Embrace modern platforms and modern programming practices and make it easy and natural for COBOL to participate.
Option 1: COBOL as procedural language

- Widely used
- Portable (except for the plethora of different dialects)
- Efficient execution

BUT...

- MUCH less efficient development environment than those offered by modern platforms such as .NET and Java
- Dwindling numbers of COBOL programmers
Option 2: embrace modern platforms

- Far greater programming productivity
- Seamless integration into modern programming platforms

BUT...

- Can mean that programs are completely tied down to a single platform
So where do we go from here?

- Two current major development platforms
  - .NET
  - Java/JVM

- Allow fullest possible use of all features of target framework...
- But at the same time allow for the greatest possible degree of portability
Accessing .NET/JVM classes

- To access a .NET or JVM type (i.e. a class, interface, enum etc.), use the TYPE keyword followed by the name of the type.
- To access members of a type, use :: following the name of a type or an instance.

- So, to define a data item to be an instance of a type:
  01 fred type System.String value “Hello world”.
- To access an instance method:
  Display fred::Replace(“world”, “Dallas”)  
- To access an instance property:
  Display fred::Length  
- To access a static method:
  Display type System.String::Concat(fred, “ Texas”)
Built in types

- In addition to the use of the TYPE keyword, certain types have built in COBOL names, for instance:

<table>
<thead>
<tr>
<th>Built in type</th>
<th>.NET type</th>
<th>JVM type</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>System.Object</td>
<td>java.lang.Object</td>
</tr>
<tr>
<td>string</td>
<td>System.String</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>binary-long</td>
<td>System.Int32</td>
<td>int</td>
</tr>
<tr>
<td>condition-value</td>
<td>System.Boolean</td>
<td>boolean</td>
</tr>
</tbody>
</table>
Namespaces and abbreviated type names

- Most type names in .NET and JVM have complex names, such as the .NET type `System.Windows.Forms.MessageBox`
- The type is said to belong to the namespace `System.Windows.Forms`, and by *importing* the namespace into the project, it is possible to refer to the type using the short name `MessageBox`, i.e:
  - `type MessageBox`

...instead of:

- `type System.Windows.Forms.MessageBox`
Non-portable example

.NET
01 s1 type System.String value “Hello world”.
display s1::Substring(4) *> .NET System.String methods
display s1::Substring(6, 5)
display string::Concat (s1, “ and goodbye”)

JVM
01 s1 type java.lang.String value “Hello world”.
display s1::substring(4) *> Java java.lang.String methods
display s1::substring(6, 11)
display s1::concat (“ and goodbye”)
Portable example

01 s1 string value "Hello world".
   display s1(5:)   *> using reference modification
   display s1(7:5)
   display s1 & " and goodbye" *> using concatenation
Demo
New in 2.0 - Portable syntax for accessing lists and dictionaries

- LIST and DICTIONARY become predefined types (in a similar way to STRING and OBJECT), allowing the compiler to map implicitly onto the appropriate .NET or JVM type
- Uses variants of the WRITE, READ, REWRITE AND DELETE statements, making it possible to write portable code using these collection classes. The code is then portable between .NET and JVM
- New CREATE verb to create instances
- Specialized iteration syntax to normalize iteration through dictionaries
New in 2.0 – inline declaration of local variables

- Can now declare local variables within the body of a method’s (or procedural COBOL’s) procedure division

```cobol
declare i1 as binary-long = 3
display i1
perform varying i2 as binary-long from 1 by 1 until i2 = 8
    display i2
end-perform
perform varying s as string through string-collection
    display s
end-perform
```
New in 2.0 – access to indexers

- In .NET, an indexer is a kind of property, except that unlike a normal property it has parameters.
- It has a fixed name, and the idea is that you can access the indexer using subscript-like syntax on a type instance. We use square brackets to access indexer.

```
01 myList list[string].
01 myDict Dictionary[string string].
display myList[0]  /* access 1st element*/
display myDict["Max"]/* access value with key "Max"
```
New in 2.0 – zero based subscripting using [...] 

- For consistency with the use of [...] to access indexers, we also allow the use of square brackets to do standard table access, but zero based as opposed to one based

```plaintext
01 myArray string occurs 10.
display myArray[0]   -> first element of array
```

- equivalent to...
```plaintext
display myArray(1)
```
New in 2.0 – declaring properties

- PROPERTY-ID name *type-specifier*.
- PROCEDURE DIVISION.
- GETTER.
- Sets the returned value in *property-value*
- SETTER.
- Uses the incoming value in *property-value*
- END PROPERTY.
New in 2.0 – declaring indexers

• INDEXER-ID type-specifier.

• PROCEDURE DIVISION USING BY VALUE ...

• GETTER.

• Sets the returned value in property-value

• SETTER.

• Uses the incoming value in property-value

• END INDEXER.
In progress for 2.1 – simplified syntax for delegate creation

• Instead of:
  
  01 d type MyDelegate.
  
  set d to new MyDelegate(self::m)
  
• ...the method can be converted directly to a suitable delegate type:
  
  set d to method self::m

• Similarly for anonymous methods:
  
  set d to delegate display "Hello world" end-delegate

• ...instead of:
  
  set d to new MyDelegate(delegate display "Hello world" end-delegate)
In progress for 2.1 – delegates and events on JVM

- Unlike .NET, JVM has no direct support for delegates (or events, which are based on delegates)

- We hope to make further progress on the goal of portable language by enabling use of delegates in JVM, but details of implementation are not yet tied down
Questions

robert.sales@microfocus.com