Object Rexx Collection Classes

If all you have is a hammer...

Everything looks like a nail...

Understanding the collection classes

- How to choose the different types of collection
- Understanding object indices
- Other language uses of collections

The different types of collections

- Ordered collections
  - Array, List, Queue
- Key-indexed collections
  - Table, Relation, Directory
- Set-like collections
  - Set, Bag

Ordered collections

- Elements are inserted in a particular order
- Element indexing is specific to the collection type

Arrays

- Ordered list of references indexed by numeric order
  - Can be created with an initial size, but will grow automatically
  - Arrays can be sparsely populated, but “skipped” elements still take up index positions

Array usage

```
a = .array~new(10)   -- creates 10 initial slots
a = .array~of("Fred", "George") -- fills array

a[1] = "Rick"   -- assigns the first array position
a[a~last+1] = "McGuire" -- adds new last item
a[1,2,3] = "RexxLA" -- multi-dimensional
```

List and Queue

- Very similar collections. Queue is a list with PUSH/PULL/PEEK methods
- Ordered collections using index values assigned with an object is inserted
- Unbounded in size, never sparse. Removing elements will “close up” the gap.

List/Queue examples

```
list~insert(“Fred”)  -- adds to end of list
          -- inserts after first item of list
list~insert(“Rick”, list~first)

queue~push(“Fred”)  -- insert at head of queue
queue~queue(“Lee”) -- add to the end
queue~pull                -- get the head element
```

Key-indexed collections

- Items are stored in the collection using a “key” value
- Key lookups are based on equality
- The Directory class has some useful extra features

Object keys

- Table and Relation key lookups determine equality using the “==” method
  - If string objects are used for keys, string value equality is used
  - For other objects, equality is determined by “object identity”
  - BUT, it is possible to change this using a custom “==” method

Overriding “==”

```
::method “==”
expose name
use arg other
-- override the hashcode form
if arg() = 0 then return name~”==”
-- compare the names of the two employees
return name == other~name
```

Overriding “==”

- The “==” method is used both to retrieve a hash code for the object and perform comparisons
- Table and Directory are implemented as hash tables, so a constant hash code is required

Directory keys

- Directories are indexed only by string keys
  - Using non-string objects is an error

Table and Relation

- Table and Relation implement one-to-one and one-to-many mappings.
  - Table can have just a single value associated with a key
  - Relation can have multiple values associated with a single key value

Using relations

```
byLocation = .relation~new

do employee over employees
    byLocation[employee~location] = employee
end

sandyHookers = byLocation~allAt(“Sandy Hook”)
```

Directories

- Keys must be strings
- Directory implements an UNKNOWN method that allows values to be set/retrieved using method invocations

  – dir~fred = employee -- sets “FRED”
  – This form uppercases the index string.

Adding active code to Directories

- SETMETHOD allows methods to be set as directory keys:
  - dir~setmethod(foo, “use arg key; return value(key,,'ENVIRONMENT'”)
- SETMETHOD can also override the UNKNOWN method.

Set and Bag

- Keyless collections (or more precisely, the objects added are their own keys)
- Set will eliminate duplicates
- Bag allows duplicates to be added
  - "duplication" is determined using same rules used for Table/Relation key matches
- Implement UNION, INTERSECTION, XOR, DIFFERENCE, and SUBSET

```
unique = .set~new

do while text != “”
    parse var text word text
    unique~put(word)
end

do word over unique
     say word
end
```

Other collection facilities

- MAKEARRAY
- DO OVER
- Suppliers

MAKEARRAY

- All collections implement a MAKEARRAY method. Each collections defines what that operation means
  - Array -- returns a non-sparse array with items in order
  - List, Queue -- returns array with items in list order
  - Table, Relation, Directory – returns array of index objects (no defined order)
  - Set, Bag – returns array with all contained objects (no defined order)

DO OVER

- Sends a MAKEARRAY message to the OVER expression result, then iterates over each of the returned array items.
- DO OVER works off of a snapshot of the object, so the iteration set is safe from alterations to the base collection.
- Any object that implements a MAKEARRAY method can be “done over”, not just stems or collections.

Suppliers

- All collections implement a SUPPLIER method
- Suppliers allow iteration over collections, providing access to both the indices and the values.
- Like MAKEARRAY, this works off of a snapshot of the values.

Supplier example

```
sup = byLocation~supplier
while sup~available
    say sup~item~name “works at“ sup~index
    sup~next
end
```

Questions?