ooRexx Group Therapy

The Fear of Objects
When Object Rexx was first designed during the Peak of Inflated Expectations era
- ...it's now the Plateau of Productivity.
- ...Resistance is Futile!
Nouns and Verbs

- Design your application by identifying the entities you need to manipulate (the “nouns”) and the operations you need to perform on the entities (the “verbs”)
  - These are your starting classes and methods
  - Each class is a specialist at an individual task
  - Fine-grained objects working together to create a whole is the goal
  - Note that “Interitance” is NOT the starting point here.
A **DSECT is not an object**

- This
  
  ::class tablem
  ::attribute stcknum
  ::attribute artist
  ::attribute title
  ::attribute instock
  ::attribute price

- Is little different than this
  
  TABELEM DSECT
  STCKNUM DS F
  ARTIST DS CL24
  TITLE DS CL24
  INSTOCK DS F
  PRICE DS F

- An object is more than just data!
Design the operations first

- Define the nouns, decide on the verbs...
  - Then decide in the data you need internally to implement the above
  - ::attribute methods define a “set” and “get” operation. It is part of your object interface.
    - Not all variables used inside the object are appropriate to expose as part of the interface.
Don't design your objects as collections

• Separate the implementation of the object from its presence in a collection:
  – “a~setTitle(i, “This is the title”)”
  vs.
  – “a[i]~setTitle(“This is the title”)”
The factory is not the car!

- Classes are the factories that make objects
  - There is one factory, which can make many objects.
  - New objects are ordered from the factory ("new")
  - Classes are themselves objects, so they can have their own methods defined

- Object instances are created by the factories
  - Object customization finishes when the factory calls "init" on the new object
  - One factory, many object instances
Keep the function close to the data

• If code that uses a class is making many calls to object methods or changing many attributes, perhaps code should be refactored into a method of the target class.
  – This is particularly true if this occurs in more than one place!
Understanding References

• Everything in ooRexx is done using references ("pointers") to objects
  – All variables. An assignment just updates the object reference
  – All expressions evaluate to a result object
  – All method/function arguments are passed as references
  – Some objects inherently contain references to other objects (e.g., the Collection classes)
Variables ≠ Objects

- A variable is NOT the same as the object it references
  - A variable in an expression evaluates to an object reference, just like any other expression term
  - When used as a function/method argument, the receiving function/method only sees the evaluated object reference, not the originating variable
  - Multiple variables may point to the same object reference...
    - This is where “Immutability” becomes an important concept
Immutability

- Some objects contain references to other objects that can be updated
- When referenced by multiple variables, the update is seen in multiple places.
  - None of the variables are changed...they still point to the original object
  - Assigning something to the variable updates the variable reference, severing the connection
- String objects are “immutable”, so you cannot see this effect with strings
Consider this...

\[
a = .\text{myclass}\text{~new}(\text{“Fred”})
b = a \quad \text{-- “B” and “A” point to same object}\n\]
\[
a\text{~value} = \text{“Mike”} \quad \text{-- updates variable inside object}\n\]
\[
say a\text{~value} b\text{~value} \quad \text{-- displays “Mike Mike”}\n\]
\[
a = .\text{myclass}\text{~new}(\text{“Rick”}) \quad \text{-- “A” points to different object}\n\]
\[
say a\text{~value} b\text{~value} \quad \text{-- displays “Rick Mike”}\n\]

::class myclass
::method init
expose value
use arg value

::attribute value