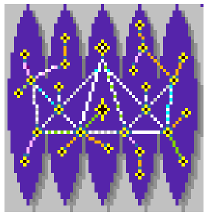
VM/ESA & VSE/ESA Technical Conference
May 31-June 3, 2000
Orlando, Florida

NetRexx Hands-on Lab

Sessions M62, M63, M64

Christine Casey
IBM VM Development
ccasey@vnet.ibm.com

Chuck Morse
IBM Washington Systems Center
morsec@us.ibm.com



RETURN TO INDEX

Disclaimer

Trademarks

The following are trademarks of International Business Machines Corporation. Those identified with an (*) are registered trademarks of International Business Machines

- IBM*
- S/390
- VM/ESA*
- OpenEdition
- OS/2
- AIX

The following are trademarks of the corporations identified

- Windows 95 - is a registered trademark of Mircrosoft Corporation
- JAVA - is a trademark of Sun Microsystems
- "Write once, run anywhere" - is a trademark of Sun Microsystems

Acronyms

AWT	Abstract Windowing Toolkit
JDBC	Java Database Connectivity
JDK	Java Developer’s Kit
JIT	Just-In-Time (compiler)
JVM	Java Virtual Machine
POSIX	Portable Operating System Interface

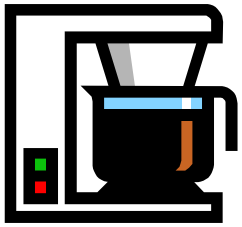
Agenda

- A Quick Review
- Using the OpenEdition/VM Shell
- Using the NetRexx Compiler and JDK
- The NetRexx Language
  - The Basics
  - Strings
  - Control Constructs & Exceptions
  - Subroutine & Function Methods

A Quick Review

NetRexx

- Roots in REXX and Java
- Ease of use from REXX
- Object model from Java
- Compiles to Java classes
- Java and NetRexx classes fully compatible
- Cross-platform portability

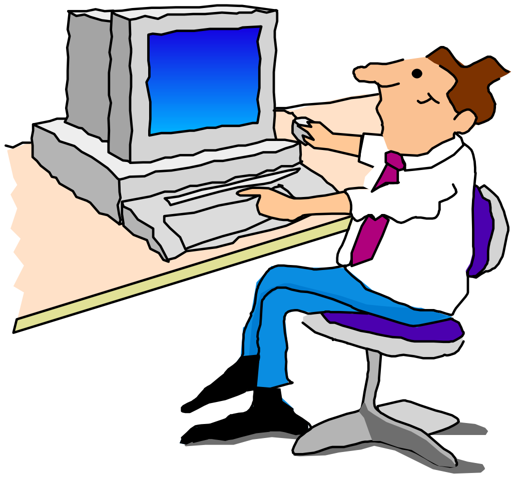


Object Oriented Programming

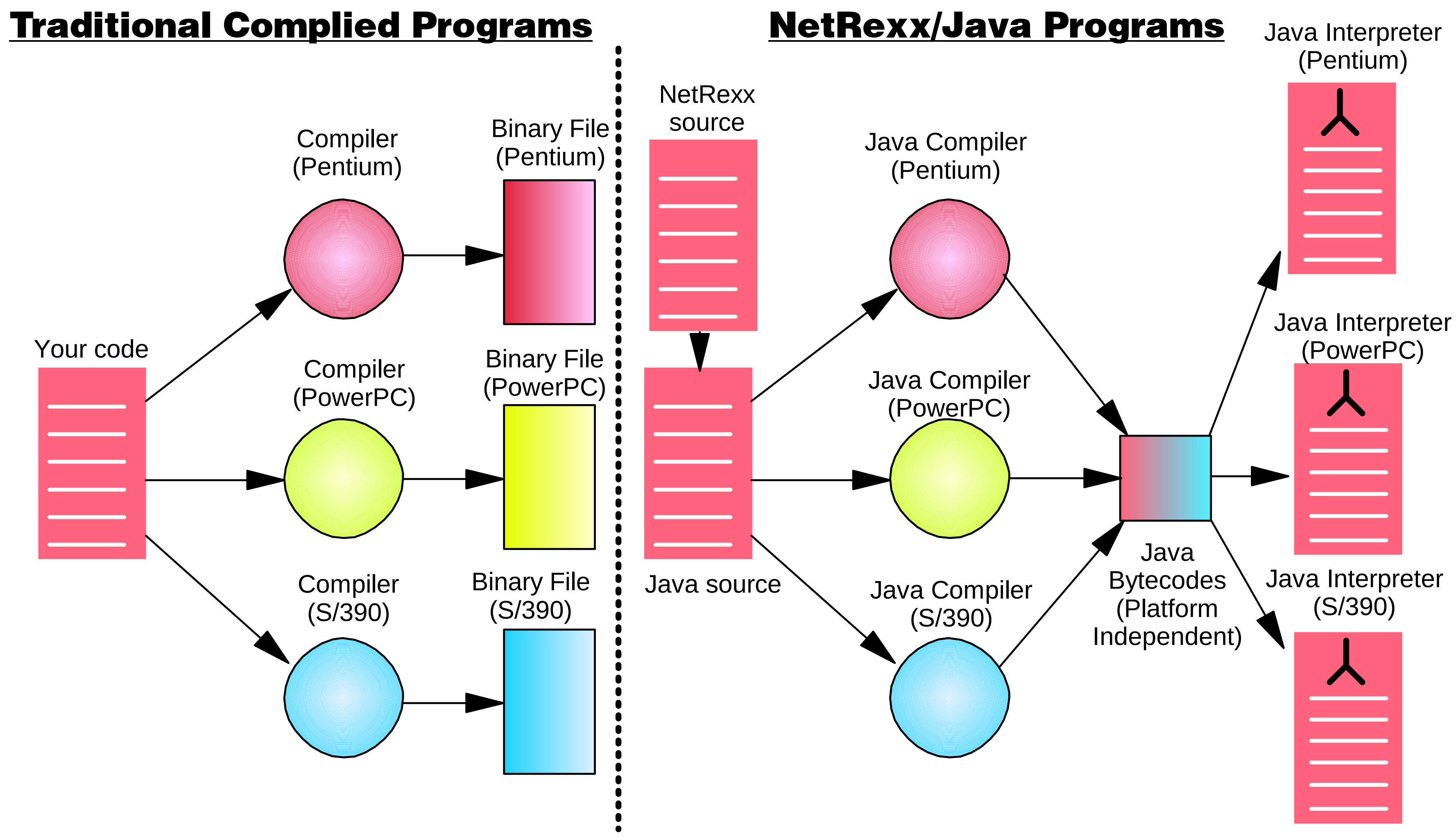
- Principles of OOP
  - Reuse
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Objects
  - Methods
  - Classes

The Java Virtual Machine

- A software microprocessor with its own instruction set and op-codes
  - Interprets bytecodes produced by the Java compiler
    - Architecture independent
    - Dynamically linked
  - Performs run time checking
    - Type and bounds checking
    - File I/O errors, exceptions



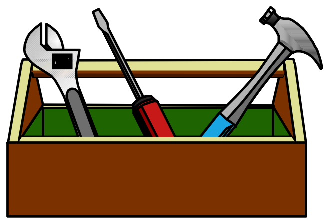
The JVM Concept



The Java Developer's Kit

Packages (class libraries)

- Common to every implementation
- Java source included
  - java.lang, java.util, java.math, java.text [strings, numbers, date/time...]
  - java.io, java.net [file and network I/O]
  - java.awt [abstract window toolkit], java.applet [animation, audio]
  - java.security [public keys, cryptography]
  - java.sql [database], java.rmi [remote methods]
  - java.beans [library of pluggable components]



Programs

javac	Compiles Java source into bytecodes
java	Invokes the JVM to run a compiled application
appletviewer	Previews a compiled applet
javadoc	Extracts interface documentation from source
javah	Generates C skeletons for native methods
javap	Disassembles Java class files
jdb	Runs the Java debugger

Samples and demos to illustrate usage

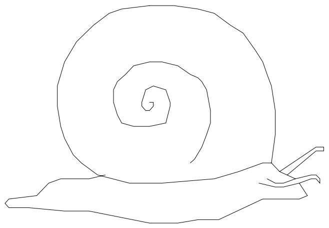
Java & NetRexx on VM/ESA

- JVM, JDK 1.1.6 and NetRexx 1.160
  - Requires VM/ESA V2R3.0+
    - Byte File System
    - OpenEdition Shell and Utilities
  - Can be obtained
    - From the Web
  - Do not support
    - Execution of JDBC classes
  - Can execute AWT classes via Remote AWT
    - Packaged with JDK 1.1.6
  - Includes Just-in-Time Compiler (JIT)

Getting Into the OpenEdition/VM Shell

- SHELL EXEC

```
/* By Richard Lewis */
'CP TERM LINEDEL OFF'
'CP TERM ESCAPE OFF'
'SET INPUT'
'OPENVM UNMOUNT /'
'OPENVM MOUNT /../VMBFS:VMSYS:ROOT/ /'
'EXEC LOADJAVA'
'OPENVM SHELL'
'CP TERM LINEDEL "'
'CP TERM ESCAPE "'
Exit
```



Getting Into the OpenEdition/VM Shell...

- LOADJAVA EXEC

```
/*  Pre-load the Java DLLs  */
Address OPENVM
flags = ''
libpath = ''
libpathlen = Length(libpath)
loadname.1 = '/usr/java/lib/openvm/native_threads/libagent.so'
loadname.2 = '/usr/java/lib/openvm/native_threads/libjava.so'
loadname.3 = '/usr/java/lib/openvm/native_threads/libjitc.so'
loadname.4 = '/usr/java/lib/openvm/native_threads/libjpeg.so'
loadname.5 = '/usr/java/lib/openvm/native_threads/libmath.so'
loadname.6 = '/usr/java/lib/openvm/native_threads/libmmedia.so'
loadname.7 = '/usr/java/lib/openvm/native_threads/libnet.so'
loadname.8 = '/usr/java/lib/openvm/native_threads/libsysresource.so'
loadname.9 = '/usr/lib/sockdll'
loadname.0 = 9
Do i = 1 To loadname.0
  the_loadname = loadname.i
  loadnamelen = Length(the_loadname)
  'BPX1LOD loadnamelen the_loadname flags libpathlen libpath rv rcode rs'
  End
Exit
```

User-Specific Login Profile

- .profile

```
export TZ=EST5EDT
set -o logical
export PS1='$PWD ($?)->'
export PATH=$HOME/bin:$PATH
export CLASSPATH=$HOME:$CLASSPATH
# Set a few useful functions
function xedit { cms "XEDIT '$1' ( NAMETYPE BFS" ; return }
function help { cms "HELP OSHELL $1" ; return }
# ... and a few useful synonyms
alias dir='ls -al'
alias x='xedit'
alias xw='xeditw'
alias man='help'
alias hh="cms help oshell '$1'"
alias erase='rm'
#cms 'set output ad ['
#cms 'set output bd ]'
#cms 'set input [ ad'
#cms 'set input ] bd'
cd $HOME
```

A Few Useful Commands

- In the Shell
  - list files in the current directory
    `ls -l`
  - copy a file
    `cp fromfile tofile`
  - change working directory
    `cd`
  - Erase a file
    `rm filename`
  - Rename a File
    `mv oldname newname`
  - Type the contents of a file
    `cat filename`
  - Cancel program execution
    `ctl-C`
    `(HX will abend the Shell)`
- Outside the Shell
  - Copy a file into the Byte File System
    `openvm putbfs fromfile tofile`
  - Make a Retrieve key
    `set pfnn retrieve`

All references to objects in the byte file system are case sensitive!

Compiling & Executing NetRexx Programs

```
NetRexxC name -option -option
NetRexxC name -run
```

- Translates NetRexx source to Java source
- Invokes Java compiler to create Java byte code

```
nrc name -option -option
nrc name -run
```

- Invokes NetRexxC
- Optionally runs program (if `-run` specified)

```
javac name.java
```

- Compiles Java source to bytecode

```
java name
```

- Executes java bytecode

Command Line Options

–Keep	NetRexxC saves the intermediate Java source file with **.java.keep** extension
–Nocompile	Stops NetRexxC after the first phase. Java source kept with **.java** extension so it can be compiled further with another compiler
–Time	Displays processing time (translation, compilation, total)

More Compiler Options

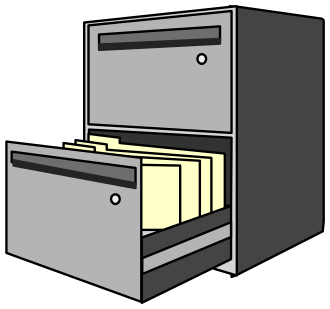
- Can be specified on command line or on OPTIONS statement

– **BINARY**	All classes treated as binary
– **CROSSREF**	X-ref of variables organized by class
– **DIAG**	Displays diagnostic information
– **FORMAT**	Adds spaces, newline chars to java source
– **LOGO**	Controls printing of the compiler logo
– **REPLACE**	Can overwrite an existing .java file
– **STRICTARGS**	( ) enforced for method invocations
– **STRICTASSIGN**	Checks that type of assignment and method args match
– **STRICTCASE**	Reference to java classes must match
– **STRICTIMPORT**	Prevents automatic class imports
– **STRICTSIGNAL**	Compiler complains if exceptions are missing
– **TRACE**	Enable/disable all trace instructions
– **UTF8**	Source is UTF-8 encoded
– **VERBOSE[n]**	Specify number of messages when executing

See page 15 for more details

File Types Used or Created

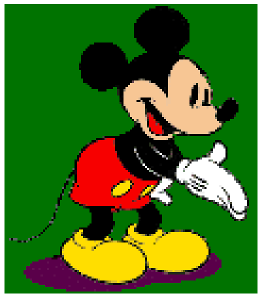
*.nrx	NetRexx program files
*.class	Compiled NetRexx or Java source
*.crossref	Variable cross reference file
*.java.keep	NetRexx pgm translated to Java
*.java	Temp generated Java program



Exercise #1
Using the Compiler

- Use NetRexxC to compile TOAST.nrx
  - enter the NetRexxC command
  - Visit Sea World
  - list the files created by the compiler
  - run the resulting Java class file

- Use nrc to compile & run TOAST.nrx
  - enter the nrc command
  - Say Hi to Mickey
  - list files created by compiler



- Create Java source for TOAST.nrx
  - use the nrc command
  - Take a look at the Space Shuttle
  - display Java source

- Compile and execute the java source created in #3

NetRexx Syntax

- Case Insensitivity

  Sea World is the same as **sea world**

- Comments
  - Rexx-Java style: /*   */

```
/* This is a comment */
```

  - Line comments: --

```
say "No comment"--The rest is a comment
```

- Continuation
  - Statements end at line-end or ; continued with hyphen

```
say 'This text is continued ' -
    'on the next line'
```

NetRexx Strings

- Any group of characters inside single or double quotation marks.

```
  "We are not trying to entertain the critics. I’ll take my" -
  ’chances with the public.’ -- Walt Disney
```

- Two " or ’ indicates a " or ’ in the string

```
  ’I only hope that we don’’t lose sight of one thing - that it’ -
’was all started by a mouse.’ -- Walt Disney
```

- The escape character \ can also be used

```
    ’There\’s enough land here to hold all the ideas and plans’ -
    ’ we can possibly imagine.’ -- Walt Disney
```

Escape Sequences

`\t`	tab
`\n`	new-line
`\r`	carriage return
`\f`	form feed
`\"`	double quote
`\’`	single quote
`\\`	backslash
`\-`	null
`\0`	null
`\xhh`	hex character
`\uhhhh`	hex character

Primitive Java Data Types

- All Primitive Java types available
  - boolean, char
  - byte, short, int, long
  - float, double
- All data converted to NetRexx strings before evaluation
- Automatic conversion between data types

See page 21 for details on Java data types

Operators & Expressions

- String Expressions

```
(blank)     "Sea"  "World"  --> "Sea World"
  ||        "Sea"||"World"  -->  "SeaWorld"
(abuttal)   abc = "Sea"
            abc"World"        -->  "SeaWorld"
```

- Arithmetic Expressions

```
+   -   *   /   % (int division)   //  (remainder)
** (power)     Prefix -    Prefix+
```

See page 22 for details

Operators & Expressions

- Comparative Expressions
  - Normal = \\= > < >= <=
    - comparison is not case sensitive
    - leading/trailing blanks removed before compare
    - shorter strings padded with blanks on right
  - Strict == \\== >> << >>= <<=
    - comparison is case sensitive
    - if 2 strings = except one is shorter, the shorter string is less than the longer string
- Logical Expressions

  & | && Prefix \\

See page 22 for details

Variables

- Named object whose value may change (but not its type)
- Variable names
  - case insensitive
  - cannot begin with a digit
  - cannot contain a period
- Defined by assignment

```
population = 176373
```

- Can be declared by assigning a type

```
population = int
```

Talking to a NetRexx Program and Getting it to Talk Back

- say [expression]
  - writes output to the user’s terminal

```
say 'Shamu eats an average of ' -
' 7 * 250 'pounds of fish per week'
```

  - terminating the string with a null character (\- or \0) suppresses the new line sequence

```
say "enter the orbiter's velocity\-"
```

- ask
  - reads input from the user’s terminal

```
velocity = ask
```

Tracing

- trace all
- trace methods
- trace results
- trace off

- output identifier tags:

`*=*`	1st source line of clause
`*-*`	continuation line
`>a>`	value assigned to arg
`>p>`	value assigned to property
`>v>`	value assigned to variable
`>>>`	result of expression
`+++`	error messages

Tracing -- example

- 3-line program:

```
trace results
number=1/7
parse number before '.' after
```

trace output:

```
 2 *=* number=1/7
   >v> number "0.142857143"
 3 *=* parse number before '.' after
   >v> before "0"
   >v> after "142857143"
```

Exercise #2
Say and Ask

- SHAMU eats an average of 250 lbs. of fish per day.

- Write a NetRexx program to:
  - prompt for a number of weeks
  - calculate the pounds of fish Shamu would eat in that time
  - display the number of weeks and the total consumption as:
    - **`'SHAMU eats 5250 pounds of fish in 3 weeks'`**
- Run the resulting Java class file with various numbers of weeks

Parsing Strings

- Very similar to Rexx

```
parse 'December 5, 1901 - Chicago' w1 w2 w3
```

  - w1 = 'December'
  - w2 = '5,'
  - w3 = '1901 - Chicago'

```
parse 'December 5, 1901' w1 . w2
```

  - w1 = 'December'
  - w2 = '1901 - Chicago'

```
parse 'December 30, 1890' w1 ',' w2
```

  - w1 = 'December 5'
  - w2 = ' 1901 - Chicago'

- Passing Arguments to a NetRexx Program

```
parse arg arg1 arg2 arg3
say arg1 arg2 arg3
```

Exercise #3
Passing Parameters

- The average temperature can be calculated by adding the high and the low temperature and dividing by 2.
- Write a NetRexx program to
  - take as an argument: a month, the average high and low temperatures (separated by commas)
  - calculate and display the average temperature as:

```
 'The average temperature for Orlando in January is
60.5 degrees.'
```

  - run the program using any of the following values

month	High(F)	Low(F)
February	73	50
May	88	66
August	92	73
October	84	65

String Methods

- Strings in NetRexx are of type *Rexx*
  - String functions invoked as object methods
  - Standard methods from Object Rexx

`’-17322’.abs`	`17322`
`’orbit ’.compare(’orbit’,’-’)`	`6`
`’17322’.datatype(’W’)`	`1`
`’STS-96’.length`	`6`
`’DISCOVERY’.lower(2)`	`’Discovery’`
`’39B’.pos(’B’)`	`3`
`’Discovery’.substr(4,4)`	`’cove’`
`"Launch Pad 39B".wordpos(’Pad’)`	`2`
`’15’.x2d`	`21`

See page 31 for a description of the built-in methods

Arrays

- Fixed size - must be constructed first
- Index is of type int and starts at 0
- Length is provided by length variable
  - `orbiter=Rexx[5]`
  - `orbiter[0]='Columbia'`
  - `orbiter[1]='Challenger'`
  - `orbiter[2]='Discovery'`
  - `orbiter[3]='Atlantis'`
  - `orbiter[4]='Endevour'`
  - `orbiter.length` $\rightarrow$ 5

Indexed Strings

- Strings with subvalues
- Similar to Rexx stem variables
  - Non-indexed value must be assigned first
  - Non-indexed value used for reference to non-existing value
    - `orbiter=’?’`
    - `orbiter[’STS-96’]=’Discovery’`
    - `orbiter[’STS-93’]=’Columbia’`
    - `orbiter[’STS-99’]=’Endevour’`
    - `say orbiter[’STS-93’]` → `Columbia`
    - `say orbiter[’STS-103’]` → `?`

Control Constructs - Selection

```
if height > 40 then say ’may ride Space Mt.’
                else say ’cannot ride Space Mt.’

select
  when height > 52 then say ’may ride all rides’
  when height < 40 then say ’cannot ride restricted’
  otherwise say ’may ride some restricted rides’
end
```

DO....END can be used to create a code block

```
if year > 1440 then do
     say ’This event occurred after the invention’
     say ’of the printing press’
   end
  else say ’before printing press’
```

Control Constructs - Loops

```
loop forever
  say ’You will get tired of this’
end

loop for 3
  say "It’s a small world after all, \-"
end

loop i=1 to 50 by 1
  say i
end
```

More Loops

```
i=30
loop until i > 21
   i=i+5
end
say i      → 35

i = 30
loop while i < 21
   i=i+5
end
say i      → 30
```

More Loops

```
orbiter='?'
orbiter['STS-96']='Discovery'
orbiter['STS-93']='Columbia'
orbiter['STS-99']='Endevour'
loop mission over orbiter
  say 'the orbiter on mission' mission -
    'is' orbiter[mission]
end
```

Exceptions

- Semantics from Java
- Generalized and simplified syntax (extends existing control constructs)

```
say 'Please enter a number:'
number=ask    -- read a line
do
  say 'reciprocal is:' 1/number
catch Exception
  say 'Sorry, could not divide'-
      '"'number'" into 1'
end
```

Iterate & Leave

- Iterate
  - causes a branch to the end of a control construct
- leave
  - exits the control construct

Exercise #4
Sorting Cards

- Convert the program CARDSORT EXEC to NetRexx
  - cardsort takes an argument of 13 words representing the values of playing cards and sorts them in descending order

CARDSORT EXEC

```
/* */
rank='2 3 4 5 6 7 8 9 10 J Q K A'
parse arg hand
num=words(hand)
do i=1 to num
  parse var hand item.i hand
end
do i=1 to num
  do j=i+1 to num
    if wordpos(item.j, rank) > wordpos(item.i,rank)
       then do
         temp=item.j
         item.j=item.i
         item.i=temp
       end
    end j
    hand = hand item.i
end
say hand
```

Subroutines & Functions

- Functions return values
- Subroutines do not
- Both Implemented as methods
  - defined with the method statement
    - method name(parameters) static returns classname
  - data must be passed as parameters
  - return statement exits the method and optionally returns a value

Method Example
Returning a value

```
/* spell a number from one to ten */
say "Enter a number from 1 to 10 \-"
number = int ask
say spellit(number)
method spellit(num=int) static returns rexx
numbers = ’one two three four five six seven eight nine ten’
spelling = numbers.word(num)
return spelling
```

Method Example
No Return Value

```
/* spell a number from one to ten */
say "Enter a number from 1 to 10 \-"
number = int ask
spellit(number)
method spellit(num=int) static
numbers = ’one two three four five six seven eight nine ten’
say numbers.word(num)
```

Exercise #5
Game.nrx

- Modify the program game.nrx
  1. Place the code that writes the results to the console in a method
  2. Write a method to keep asking the user for a number until an integer is entered
     - loop until the datatype of the user's input is W

Looking at the solution on page 51 is frowned upon

Game.nrx

```
/* small game to guess a number between 1 and 100 */
say "I’m choosing a number between 0 and 100"
number = 100*Math.random( ) % 1  -- %1 trans result to int
say ’Found a number’
guess = int                      -- force guess to int
loop count = 1 until guess = number -- loop until guessed
  say count  ’try:’          -- num tries
  guess = ask                    -- get player input
  select                         -- compare guess to number
   when guess > number then
     say guess  ’is too big’
    when guess < number then
     say guess  ’is too small’
    otherwise
     say ’Congratulations! You did it with ’ count ’tries.’
    end
catch RunTimeException         -- if guess not valid number
     say  ’Sorry, whole numbers only.  You lost the game.’
end
```

For More Information

IBM Centre for Java Technology Development:

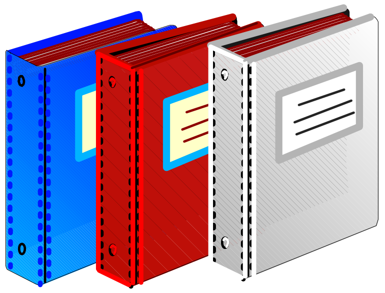
- http://ncc.hursley.ibm.com/javainfo/

Mike Cowlishaw's NetRexx Language page:

- http://www2.hursley.ibm.com/netrexx/

Mikes' new book: **The NetRexx Language**

- ISBN 0-13-806332-X,
- IBM Puborder SR23-8926



ITSO Redbooks:

- Creating Java Applications Using NetRexx, SG24-2216-00
- VM/ESA Network Computing with Java and NetRexx, SG24-5148

IBM's Java page: http://www.ibm.com/java

Sun's Java page: http://www.javasoft.com

Some Books on Java

- Ken Arnold and James Gosling, **The Java Programming Language, ISBN 0-201-63455-4**
- David Flanagan**, Java in a Nutshell, ISBN 1-56592-183-6**
- Peter van der Linden, **Just Java, ISBN 0-13-565839-X**

Acknowledgments

For help in providing content to portions of this presentation,
we would like to give special thanks to:

ITSO, San Jose Center -- Redbook for NetRexx

Mike Cowlishaw, IBM Fellow, Hursley, England

Pamela Taylor, Sterling Commerce, Dallas, TX

Richard Lewis, IBM Washington Systems Center