Structured operations and statements

Topics

- Control structures
- If statement
- SELECT and switch
- For loop
- While, do-while loops
- Return statement

Unit objectives

After completing this unit, you should be able to:

- Write conditional processing code with Java's if - else, and switch control statements
- Define iterative processing with Java's for, while, and do - while iteration statements
- Alter control flow within iterative processing and methods

Control structures

- Control structures can be divided into four basic categories:
  - Sequence: Allows control to flow from one statement to the next in the order they are written
  - Decision: Allows control to flow in a different path depending on the result of a conditional statement
    - RPG examples: IF and SELECT op-codes
  - Loop: Transfer allows iterative control
    - RPG examples: DO, DOW, DOU op-codes
  - Transfer: Allow control to transfer to other parts of the program
    - RPG examples: GOTO, LEAVE, ITER, RETURN op-codes

Statements overview

RPG	Java	Description

IFxx/IF
 	
ENDIF
	if (*expression*)
	{
	// statements;
	}
		Very similar to RPG

SELECT
WHENxx/WHEN
OTHER
ENDSL
	switch (*expression*)
	{
	case value:
	default:
	}
		Choose action based on value of expression. Saves complex IF - ELSE logic.

DO
 	
ENDDO
(or FOR in V4R4!)
	for (*init*; *expression*; *increment*)
	{
	// statements
	}
		Loop a specific number of times based on an initial value, expression, and increment.

DOWxx/DOW
 	
ENDDO
	while (*expression*)
	{
	// statements
	}
		Loop as long as the expression is true.

DOUxx/DOU
 	
ENDDO
	do
	{
	// statements
	} while (*expression*);
		Loop while the expression is true. Body is executed at least once.

LEAVE ITER	break continue	Exit or iterate current or specified loop.

Control flow: If-else

```
C*    op-code   factor2
C     IF        expression
C*       Body
C     ELSE
C*       Body
C     ENDIF
C*       :
```

RPG

```
if (condition)
{
        //Body;
}
else
{
        //Body;
}
```

Java

- If-expression is evaluated
  - If true, body is executed.
  - If false, **else** statement or statement after **if** is executed.
  - **else** statement is optional for both languages.
- For RPG use free form or fixed form.
- Body can be compound or single statement.
  - Single statement bodies do not need braces in Java.

If example

```
C     AGE          IFLE      2
C                  MOVE      0     PRICE
C                  ELSE
C     AGE          IFLE      10
C                  MOVE      5     PRICE
C                  ELSE
C                  MOVE      10    PRICE
C                  ENDIF
C                  ENDIF
```

RPG

```
if (age <= 2)
  price = 0;
else if (age <= 10)
  price = 5;
else
  price = 10;
```

Java

```
C                  IF        AGE <= 2
C                  EVAL      PRICE = 0
C                  ELSE
C                  IF        AGE <= 10
C                  EVAL      PRICE = 5
C                  ELSE
C                  EVAL      PRICE = 10
C                  ENDIF
C                  ENDIF
```

RPG IV

Note single statement in body; so braces not required

RPG: Fixed versus free form IF

- **RPG IV Example:**


```
Version I   - Free Format Factor 2

...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5
    C                   IF        NUM<2 AND *IN99
    C                   :
    C                   :
    C                   ENDIF

Version II   - Fixed Format

...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5
    C     NUM           IFLT      2
    C     *IN99         ANDEQ     '1'
    C                   :
    C                   ENDIF
```

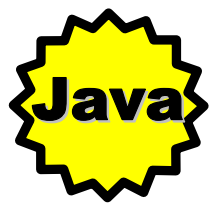
If blocks

- An if block can be one statement or multiple statements.

```
if (num<2  &&  IN99=='1')
{
                :
                .
}
```

Braces are optional if you have only one statement after the if.

Watch!!!




```
if (num<2 && IN99=='1')
  System.out.println("Hi George");
  System.out.println("Hi Phil");
```

New statement added later. Now you need braces.

Nesting if statements

```
C* Use IF and EVAL to assign
C* ticket prices for the zoo
C                   IF        Age>=65
C                   EVAL      Ticket=10
C                   ELSE
C                   IF        Age>12
C                   EVAL      Ticket=20
C                   ELSE
C                   IF        Age>2
C                   EVAL      Ticket=5
C                   ELSE
C                   EVAL      Ticket=0
C                   ENDIF
C                   ENDIF
C                   ENDIF
```

RPG IV

```
// Assign ticket prices
// for zoo based on age
if (age >= 65)
  ticket = 10;
else if (age > 12)
  ticket = 20;
else if (age > 2
  ticket = 5;
else
  ticket = 0;
```

Java

Convention is "else" and "if" on same line in Java.

Time to switch?

```
if (day == MON)
{
  // do something
}
else if (day == TUE)
{
  // do something
}
else if (day == WED)
{
  // do something
}
else if (day == THUR)
{
  // do something
}
...
```

You can do this logic with if, but switch and select are more elegant!

Assume these are previously defined integer constants.

Too much if? Switch!

RPG

RPG IV

```
C          SELECT
C          WHEN day = MON
C*            do something
C          WHEN day = TUE
C*            do something
C          WHEN day = WED
C*            do something
C          WHEN day = THU
C*            do something
C          OTHER
C*            do something
C          ENDSL
```

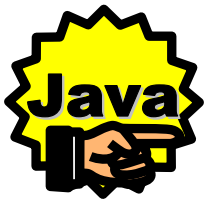
Java

Java

```
switch (day)
{
     case MON:
          // do something
           break;
     case TUE:
          // do something
           break;
     case WED:
          // do something
           break;
      ...
     default:
          // default code
} // end switch statement
```

- Improves readability.
- Structures are similar in both languages!

Same, but different



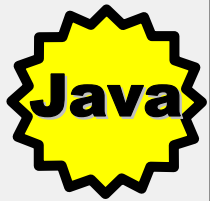
- Each WHEN expression evaluated until true
- Code executed until next WHEN

- Switch expression evaluated
- Result compared to each case
- In first match, code executed until "break;" or end of switch

RPG SELECT	Java Switch
SELECT	switch
WHEN or WHENxx	case
OTHER	default
ENDSL	end brace ‘}’

Break-less switch

```
switch (day)
{
    case MON:
    case TUE:
    case WED:
    case THUR:
    case FRI:
        // weekday code
        break;
    case SAT:
    case SUN:
        // weekend code
        break;
} // end switch statement
```



- Be careful! Remember the break statement!
  - Control goes to first "case" that matches the expression
  - Executes until "break" is encountered, or the end of switch
- Can put to your advantage
  - Clump common cases together with single break.

Looping around

- RPG and Java, like all other languages, have three main loops; they are as follows.

RPG	Java

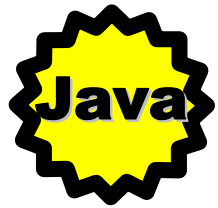
`C  start  DO  limit  index`
`C*  :`
`C  ENDDO`
DO
Note: RPG has FOR loop too now!
	`for (initialization;`
	`condition;`
	`increments)`
	`{`
	`// body`
	`}`
	FOR

`C  DOW  expression`
`C*  :`
`C  ENDDO`
DO-WHILE
	`while (expression)`
	`{`
	`// body`
	`}`
	WHILE

`C  DOU  expression`
`C*  :`
`C  ENDDO`
DO-UNTIL
	`do`
	`{`
	`// body`
	`} while (expression);`
	DO-WHILE

Control flow: For loop

- For loop in Java
  - Used when number of iterations is calculable
  - Known as determinant loop
  - Most often used when looping through arrays
- Need
  - Starting statement (initialize to starting value)
  - Conditional expression (loop while it is true)
  - Iterating statement<s>




```
for (expression)
{
  // statements
}
```

for (initialization; condition; increment)

For loop

- <declare> and initialize index variable
- Loop while true
- Increment or decrement index. Can comma separate multiple statements

```
static final int MAX = 10;
```

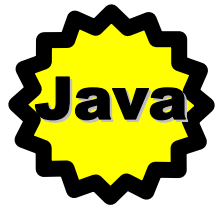

```
for (int i=0; i < MAX; i++)
{
    // body;
}
```


```
C          1          DO    10          I
C*                    ...
C                     ENDDO1
```

```
C*     initial-value DO          Limit-value    index
C      1             DO          10             I
C*                   ...
C*                   ENDDO       Increment-value
C                    ENDDO       1
```

For loop parts (1 of 2)

- All three parts are optional.
  - Only a convention that:
    - First part is for initing index variable.
    - Expression is for comparing index value.
    - Increment is for incrementing or decrementing index value.
- All three can even be empty!

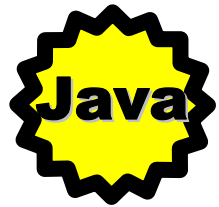



```
for ( ; ; )
  System.out.println("looping...");
```

Never-ending loop!

For loop parts (2 of 2)

- Simple bodies can be done in incrementing part versus body.
  - Comma-separated statements



Example: Blank out entire array

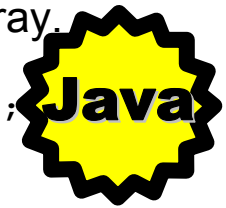
```
for (idx = 0;
     idx < myCharArray.length;
     myCharArray[idx] = ' ', idx++)
  ;
```

All work done in increment part. No need for body.

Two statements, comma separated

For loop example

- Find the first non-blank character in a character array.
  - Initialize the array:
    - `char myCharArray[]={' ', ' ','a', 'b', 'c'};`
  - Use for loop looking for first non-blank character.
  - Print the index with `System.out.println`.




```
char myCharArray[] = {' ', ' ', 'a', 'b', 'c'};
int idx;
for (idx = 0; myCharArray[idx] == ' '; idx++)
    ;
System.out.println("first blank char position = " + idx);
```

Note: Arrays are covered in detail in Chapter 6.

- But what if the array is all blanks? ...

Solution 2

```
char myCharArray[] = {' ', ' ', 'a', 'b', 'c'};
int idx;
for (idx = 0;
     idx < myCharArray.length && myCharArray[idx] == ' ';
     idx++)
   ;
if (idx >= myCharArray.length)
  System.out.println("All blank array!!! ");
else
  System.out.println("first blank char position = " + idx);
```

- Note:
  - “Length" is a built-in variable in arrays (arrays covered in chapter 6).
- Why is "idx" declared outside the `for` loop?
  - So it can be accessed after the `for` loop

New for loop in RPG

Example 1: n!

```
C*RN01Factor 1------Opcode----Factor 2------Result-Field
C*
C                   EVAL       Factorial = 1
C                   FOR        i = 1 to n
C                   EVAL       Factorial = Factorial * i
C                   ENDFOR
```

RPG IV

If n = 5,
n! = 5 * 4 * 3 * 2 * 1 = 120 ...

Example 2: Last non-blank character

```
C*RN01Factor 1------Opcode----Factor 2------Result-Field
C*
C                   FOR        i = %len(SayWhat) DOWNTO 1
C                   IF         %SUBST(SayWhat:i:1) <> ' '
C                   LEAVE
                    ENDIF
                    ENDFOR
```

if SayWhat =
'New For RPG4 ',
Last non-blank = 12

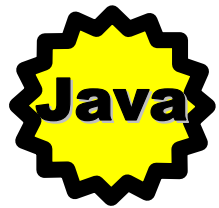
Java skills transfer

RPG IV for loop syntax

OP-CODE	FACTOR1	FREE-FORM FACTOR2
FOR		Index-name = start-value BY increment TO DOWNTO limit

Looping for a while

- while loop is a subset of for loop.
  - Takes only an expression
  - Loops until expression is false
- Use it when termination of the loop is not predetermined.
  - An "indeterminant" loop construct




```
while (expression)
{
        // statement(s)
}
```

While loop

- Loop while true.

- Set variable to force end of loop.
- Loop iterations >= 0

```
boolean in30 = false;
while (!in30)
{
   if (endOfFile())
     in30 = true;
   else
     readLine();
}
```


```
C          *IN30      DOWNE*OFF
C*                    ...
C                     END
```

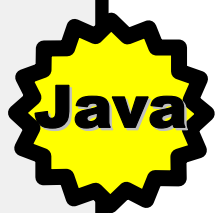

```
C                     DOW        *IN30 NE *OFF
C*                    ...
C                     END
```

Free Form Factor 2

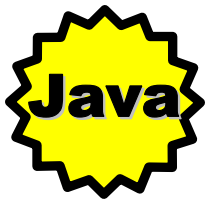
Looping until done

- do while loop similar to while loop.
- But expression is evaluated after the body of the loop.
  - This means loop executes at least once.
    - Versus zero or more times for while loop

```
do
{
   // statement(s)
} while (expression);
```



Do while loop



- Loop until true

- Set variable to force end of loop
- Loop iterations >= 1

```
boolean in30 = false;
do
{
   if (endOfFile())
     in30 = true;
   else
     readLine();
} while (!in30);
```


```
C          *IN30      DOUNE*OFF
C*                    ...
C                     END
```


```
C                     DOU        *IN30 NE *OFF
C*                    ...
C                     END
```

Free Form Factor 2

Continue, break

Label:

Note: `continue` and `break` can specify a labeled loop to explicitly iterate or leave.

```
out: for (int i=0; i < 10; i++)
{
    for (int j=0; j < 10; j++)
    {
        if (intArray[i][j] == -1)
        {
            // some code
            continue out;
        }
        if (intArray[i][j] == -2)
            break;
    } // end inner for-loop
    // outside inner loop
} // end outer for-loop
```

```
C          DOW       RECORDN = 2938174
C          IF        CODE='A1'
C          ITER
C          ENDIF
C          LEAVE
C          ENDDO
```

Example

```
int idx = 0;
while ( idx < 10 )
{
  if (!records[idx].equals("MIKE"))
  {
      idx++;
      continue;
  }
  else
  {
      salary[idx] = salary[idx] + 999;
      System.out.println(records[idx]);
      break;
  }
}
```

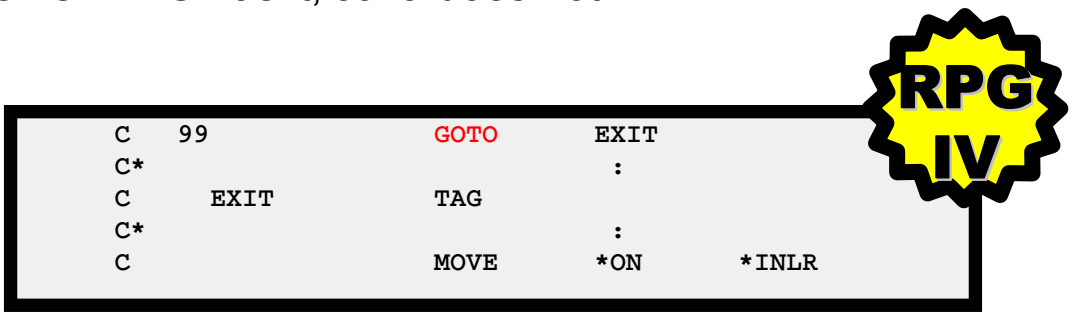
Java

```
C          MOVE      IDX            I
C          DOW       IDX<=10
C          IF        RECORDS(IDX)='MIKE'
C          EVAL      SALARY(IDX) = SALARY(IDX)+999
C          EXCEPT    HEAD1
C          LEAVE
C          ENDIF
C          ADD       1              IDX
```

RPG IV

Go to where?

- GOTO: RPG has it, Java does not.




```
C    99              GOTO      EXIT
C*                              :
C      EXIT          TAG
C*                              :
C                    MOVE      *ON       *INLR
```

- GOTO is not considered good programming practice.
  - Unstructured flow of control
  - Hard to maintain
  - Not in Java programming language, but it is a reserved word so you do not accidentally use it

Return to caller

RPG IV

Return type

```
C        CALLP      VAL = tripleval(FLD)
C* mainline code
C* --------------------
C* tripleval procedure:
C* --------------------
P tripleval        B
D                     PI          5I0
D   parm1                    5I0
C                            RETURN     parm1*3
P tripleval        E
```

```
val = tripleval(fld);
/**--------------------
 * tripleval method
 *--------------------*/
int tripleval(int parm1)
{
   return parm1*3;
}
```

Java

- `Return` must return value compatible with declared return type.
  - If no return type, can omit `return`.
  - You can have more than one per procedure or method.

New in V4R4: LEAVESR

- New in RPG IV V4R4 was LEAVESR opcode
  - Use it anywhere in subroutine to exit the subroutine.
  - Similar to return in Java!

```
C*0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++
 *
C     CheckCustName BEGSR
C     Name          CHAIN     CustFile
 *
 * Check if the name identifies a valid customer
 *
C                   IF        not %found(CustFile)
C                   EVAL      Result = CustNotFound
C                   LEAVESR
C                   ENDIF
 *
 * Check if the customer qualifies for discount program
C                   IF        Qualified = *OFF
C                   EVAL      Result = CustNotQualified
C                   LEAVESR
C                   ENDIF
 *
 * If we get here, customer can use the discount program
C                   EVAL      Result = CustOK
C                   ENDSR
```

Returning from main

- Both RPG and Java allow RETURN from mainline code:
  - Main C-spec code in RPG
  - Main method in Java
- In both cases it is also implied by reaching end of mainline.

Return from main in RPG

- The following occurs in RPG if RETURN is used in the main line of program:
  - The halt indicators are checked; if a halt indicator is on the program ends abnormally.
  - If the halt indicators are not on, the Last Record indicator (LR) is checked; if it is on, the program ends normally (and files are closed, fields cleared).
  - Finally, if none of the above, return goes back to the calling program and all data is preserved for the next time the program is called.

Return from main in Java

- The following occurs in Java if return is used in the main method:
  - If any non-daemon threads (chapter 11) are still running, exit is deferred until they end.
  - If no non-daemon threads are still running, control returns to the command line.
- A better alternative for returning from main (exiting the program) is using System.exit(n); where:
  - n = 0 (ok) or 1 (error) by convention
  - n is queryable by calling programs
- Forces end even if threads are running.

Topics covered

- Control structures
- If statement
- SELECT and switch
- For loop
- While, do-while loops
- Return statement

Unit summary

Having completed this unit, you should be able to:

- Write conditional processing code with Java's if - else, and switch control statements
- Define iterative processing with Java's for, while, and do - while iteration statements
- Alter control flow within iterative processing and methods