Java language syntax

Topics

- Basics
- Comments
- Variable naming and keywords
- Statements
- Expressions and operators
- Arithmetic manipulations

Unit objectives

After completing this unit, you should be able to:

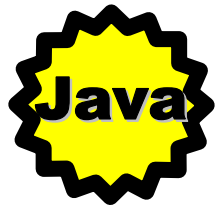
- Define a Java program using statements, variables, expressions, and comments
- Use Java's inline documentation feature
- Describe the Java operators used to manipulate Java variables

What has been covered and what is next

- Here is what has been covered so far:
  - What Java is and the reasons for using it
  - How Java applications compare to RPG applications
  - Java flow of control compared to the RPG IV language
- Coming up next:
  - Basics
  - Comments
  - Variable Naming and Keywords
  - Statements
  - Expressions and Operators
  - Arithmetic Manipulations

Need to be free!

- Unlike RPG, Java is free form!
  - All white space is ignored




```
void myMethod(int parameter1)
{
    return;
}
```

```
void myMethod(int parameter1)
{ return; }
```

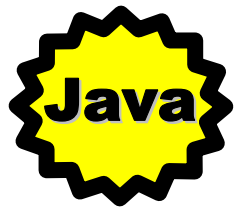
```
void myMethod(int parameter1) {
  return;
}
```

```
return;
```

```
return
;
```

Coding standards

- Naming conventions (industry conventions):
  - Package names: all.lower.case
  - Class names: UppercaseFirstLetterOfEachWord
  - Variable and method names: uppercaseFirstLetterExceptFirstWord
  - Constants: ALL_UPPERCASE
- Brace alignment (IBM preference):
  - Lined up directly under first letter of block, and always on lines by themselves
- Instance and class variable locations (IBM's):
  - Always before method definitions
- Indentation (IBM's):
  - Declarations of variables and methods inside classes: four spaces
  - Blocks inside methods, flow-of-control statements: four spaces
- White space (IBM's):
  - Around expression operators: 1
  - Between parameter declarations on a method: 1
  - Between expressions on a for loop: 1



Java language syntax

- Java language syntax is C-like.
- Semi-colons (;) end each statement.
- Statements are free-format: white space is ignored.
- Uses braces { } to delimit scope blocks, begin and end of methods, begin and end of classes.
- Names are *very* case-sensitive!

Comments

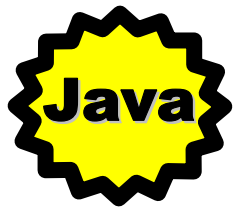
- Continue the tradition of commenting your code (or perhaps start it).
- Java has three kinds of comments.

Comment Type	Description
/* comment */	Multi-line comments surrounded by '/*' and '*/'
// comment	Single line comments using the double-slashes
/** comment */	JavaDoc comments surrounded by '/**' and '*/'

More on comments

- Multiline comment:

```
/* this is a multi
 line comment */
```



- Single line comment:

```
 // This whole line is a comment
int myVariable = 10; // Only this part is a comment
```

- JavaDoc comment:

Note double asterisk

```
/** This is the <U>scan package</U>
 *  this is the second line.
 *  @author  George & Phil
 *  @version 1.0
 */
```

JavaDoc comments (1 of 2)

- Java is all about code reuse.
  - Commenting your classes (APIs) is very important.
- JavaDoc is a command line tool in the JDK.
  - It extracts all javadoc-style comments and produces HTML files from them (external documentation).
  - c:\> javadoc MyJava.java

File type	File names
Main entry point	index.html
Package list	package-list.html
Index	index-all.html
class hierarchy	overview-tree.html
List of methods	One file per class that lists the class and its methods. In subdirectory tree matching package name

JavaDoc comments (2 of 2)

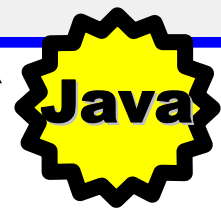
- Javadoc recognizes a special set of tags (keywords) that start with '@'
  - Use in your class and method comment header blocks to generate good HTML
  - Can also use any HTML tags like bold

Tag	Description
@author	Author of this class or method
@see	References another related class or method. Generates a link
@version	Version number of this class or method
@since	Release or version this class or method has existed since
@deprecated	This is an obsolete method
@return	Describes what this method returns
@param	Describes a parameter to this method
@throws	Describes an exception thrown by this method. @throws class-name
@serial, @serialData,@serialField	Describes information related to "serializing" this class to disk
{@link xxx}	Identical to @see but generates inline hypertext links versus separate section

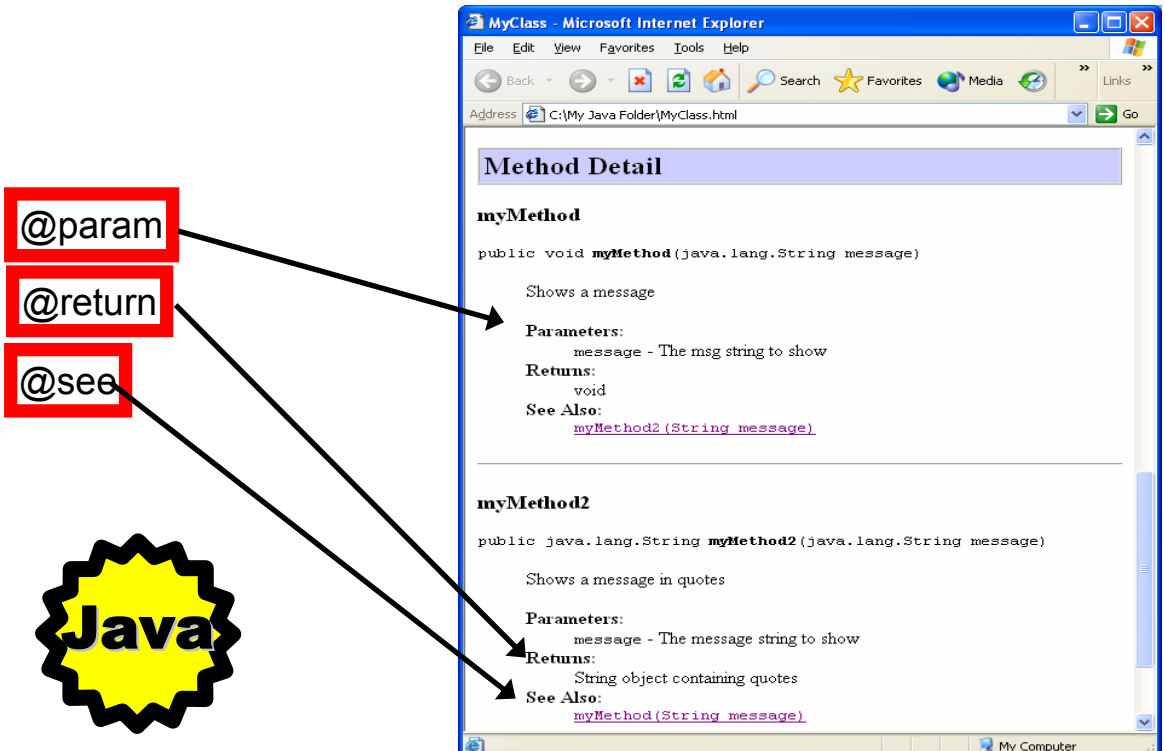
JavaDoc example

```
/**
 * A cool class.
 *
 * @author Phil Coult
 * @version 1.0
 * @see YourClass
 */
public class MyClass
{
    /**
     * Constructor
    */
    public MyClass()
    { } // empty for
    /**
     * Shows a message
     * @param  message The msg string to show
     * @return void
     * @see    MyClass#myMethod2(String message)
    */
```

```
  public void myMethod(String message)
  {
      System.out.println(message);
  }
  /**
   * Shows a message in quotes
   * @param  message The message string to show
   * @return String object containing quotes
   * @see    MyClass#myMethod(String message)
   */
  public String myMethod2(String message)
  {
      String newMessage = "'" + message + "'";
      System.out.println(newMessage);
      return newMessage;
  }
} // end class MyClass
```

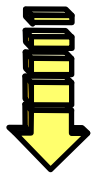


Example output



Variable naming (1 of 2)

- RPG IV limits names to 4096 characters.
- Java allows variable names to have an unlimited length.
- Java names are *very* case sensitive.



- Yes, you must be more sensitive when dealing with Java variables!

Variable naming (2 of 2)

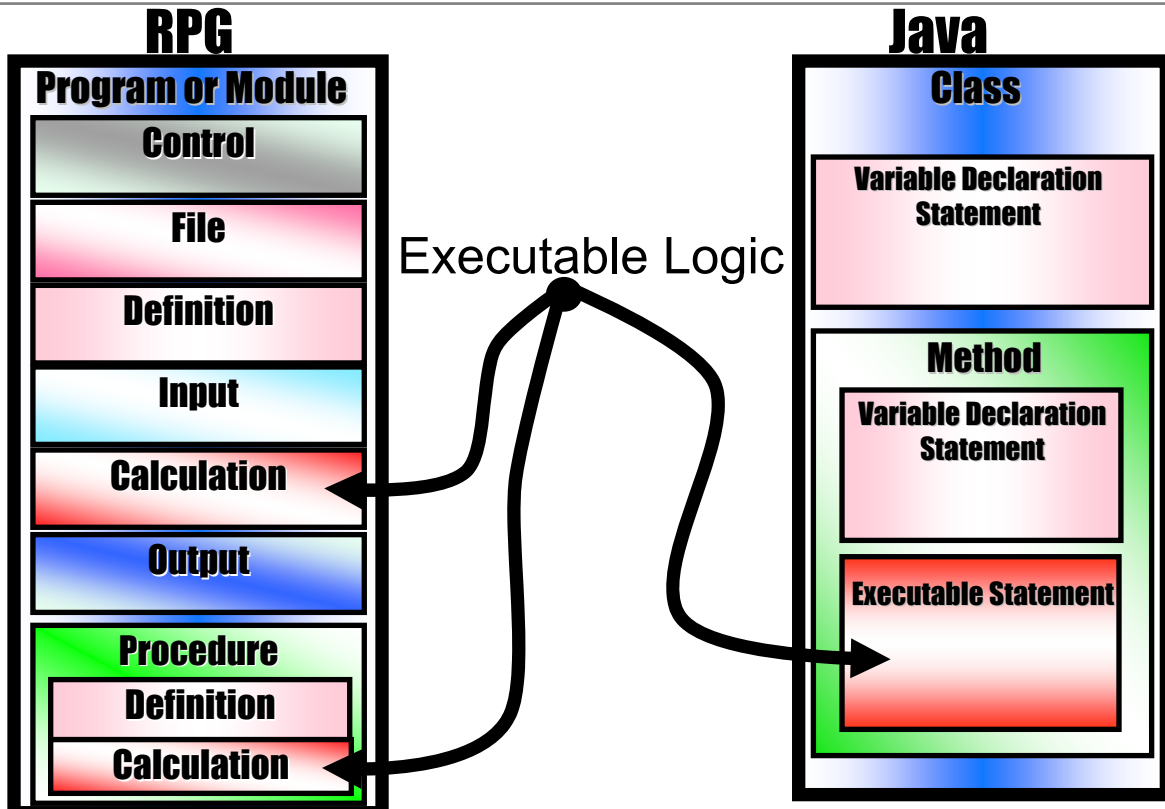
- For RPG IV:
  - First character must be alphabetic.
  - It can include the special characters, such as $, #, and @.
  - The remaining characters can be alphabetic or numeric including the '_' character.
- For Java:
  - First character can be any valid letter, the underscore, or dollar sign.
  - Remaining characters can be letters or digits.
  - Also can include UNICODE!

Keywords

- RPG IV has keywords:
  - Examples: CONST, DIM, OCCUR, INZ
- So does Java:

abstract	boolean	break	byte
case	catch	char	class
const	continue	default	do
double	else	extends	false
final	finally	float	for
goto	if	implements	import
instanceof	int	interface	long
native	new	null	package
private	protected	public	return
short	static	super	switch
synchronized	this	throw	throws
transient	true	try	void
volatile	while		

Anatomies



Statements

RPG Calc Statements

- Fixed Format OPCODEs
  - Traditional op-codes like **MOVE**
- Free Format OPCODEs
  - Newer opcodes:
    - ✓**CALLP and RETURN**
    - ✓**DOU and DOW and FOR**
    - ✓**EVAL and EVALR**
    - ✓**IF and WHEN**

RPG Free-Form Factor 2 Expressions

- Conditional
  - Simple or complex, results in true or false
- Computational
  - Operands, operators, procedure calls, and intrinsic function calls that result in a value

Java Statements

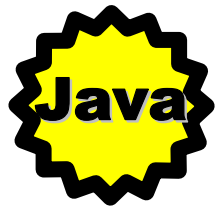
- Variable Declaration
  - Covered in Chapter 5
- Expression
  - Expression with a semicolon
- Block
  - Statements between braces
- Empty
  - Semi-colon by itself
- Labeled
  - Target of **break** / **continue** statements
- Flow of Control
  - Covered in Chapter 4

Java Expressions

- Fetch
  - Conditional and computative
- Computational
  - **NULL**, method call, and array indexing

Expression statement

- Expression statement in Java:
  - A statement executed for the side effect of the expression assignment expression statement
  - For example: total = price * tax;
- Post and pre-increment expression statement
  - For example: ++total; total--;
- Method call expression statement
  - For example: payroll.runWeeklyPayroll();
- New expression statement
  - For example: new Payroll();



More details and examples coming

All statements in Java end in a semi colon.

Block statement

- Block statement in Java:
  - Delimited by braces: { }
    - No semi colon after either brace
  - Example:

```
{
  tax = total * 0.15;
  totalPrice = total + tax;
}
```

Typically used with **if**, **while**, **do-while**, and **for** statements

  - Groups one or more individual statements
    - Allows multiple statements where only one is allowed:

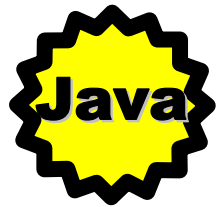
```
if (expression)
  statement;
```

```
if (expression)
{
    statement 1;
    statement 2;
    statement 3;
}
```

Empty statement

- Empty statement in Java
  - Just a semi colon by itself
    - Where a statement is expected

```
;
```



- Not very interesting; sometimes used:
  - In an **if** statement when only interested in **else** case

```
if (x > 2)
  ;
else
  // interesting statements
```

  - In a for loop when all work is done in expressions

```
for (i=0; name.charAt(i)!=' '; i = i+1)
   ; // nothing to do in the body
```

Note: The **if** and **for** statements are covered in Chapter 4.

Labeled statement

- Labeled statements in Java
  - A statement labeled prefixed by a name and colon
    - Similar to RPG TAG operation code and labels


```
aLabel: a statement;
```

- However:
  - Labels in Java are only valid on loop statements
    - while, do-while, for
  - Essentially only an identifier for the loop
- Why a label?
  - You can use break and continue to exit and iterate the labeled loop.

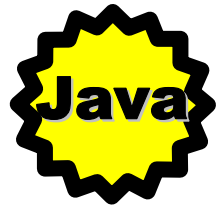
```
int index = 1;
OutHere: while(true)
{
  index = index + 1;
  if (index == 3)
    break OutHere;
}
```

A

A

You do not need to specify a label on **break** and **continue** for current loop
►*Just needed for outer loop*

Expressions



- Expressions in Java:
  - Are combinations of:
    - Operators
    - Operands
    - Method calls
    - Optional sub parentheses for enforcing priority in evaluation
  - Are used for computing or fetching values
  - Have two flavors in Java
- Fetch expressions:
  - Conditional
    - Return true or false
  - Computative
    - Return a non-boolean value
- Computational expressions:
  - Strictly speaking are:
    - NULL keyword, method calls, new operator

In practice, *computational expression* usually refers to both of these!

Expression examples

- Conditional expressions:

RPG IV	Java
`C DOW *IN99 = *ON`	`while (in99==true)`

- Computative expressions:

RPG	Java
`C EVAL x = x + 1`	`x = x + 1;`

Arithmetic operators

Java	RPG Op-Code	RPG Expression
+ (addition)	ADD or Z-ADD	+
- (subtraction)	SUB or Z-SUB	-
* (multiplication)	MULT	*
/ (division)	DIV	/
% (modulus or remainder)	MVR	%REM bif
		** (exponent)

- **Examples:**

RPG			Java
C	EVAL	X = X+1	X = X + 1;
C	EVAL	X = X / 2	X = X / 2;
C	EVAL	A = A + 2 * 3	A = A + 2 * 3;
C	EVAL	A = (A-3) / 2	A = (A - 3) / 2

Operations: Math examples

RPG

```
C*  A  =  B+C
C              B          ADD   C          A         50
C*  A  =  (B+C)/12
C              B          ADD   C          A         50
C              A          DIV   12         A
```

RPG IV

```
C     EVAL     a = b + c
C     EVAL     a = (b+c)/12
```

Java

```
a = b + c;
a = (b+c)/12;
```

Assignment

- Simple and similar in both languages:

RPG	Java
`C EVAL X = 0`	`X = 0;`

- To reduce the number of assignment statements, Java allows stringing, as in:

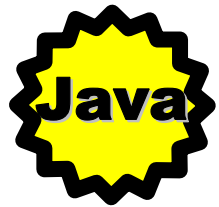
```
A = B = C = 25;
```

- New set of assignment statements:

```
X += 1; // or
Y -= 3;
```

see next page...

Contracted assignment



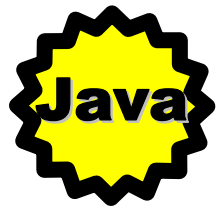
- What does this mean?
  - X += 10;
- Answer: short form for
  - X = X + 10;
- All binary operators supported, for example:
  - X *= 10; X /= 2; Y -= 1;
- Same as using ADD op-code in RPG and *not* specifying factor 1 value

Contracted operators

Operation	Java Operator	Example	Full
Add	+=	a += b	a = a + b
Subtract	-=	a -= b	a = a - b
Multiply	*=	a *= b	a = a * b
Divide	/=	a /= b	a = a / b
Modulus	%=	a %= b	a = a % b
Bit And	&=	a &= b	a = a & b
Bit Or	|=	a |= b	a = a | b
Bit XOr	^=	a ^= b	a = a ^ b
Shift Left, Right	<<,>>	a >>= b	a = a >> b
Shift Right, zero fill	>>>	a >>>= b	a = a >>> b

Increment, decrement

- What does this mean?
  - X++;
- Answer: short form for
  - X = X + 1;
- Also supports decrementing:
  - X--;
- Can be before or after variable:
  - ++X; --X; X++; X--;
  - Same as C and C++



Increment++

- Always changes variable
  - if (X++ > 10) ...

X is incremented

X==10?
result == false

- Prefix:
  - Increment variable, use value
    - X = 10;
    - Y = ++X + 2;

x=x+1;
y=x+2;

Y == 13
X == 11

- Suffix:
  - Use value, increment variable
    - X = 10;
    - Y = X++ + 2;

y=x+2;
x=x+1;

Y == 12
X == 11

Incrementing and decrementing

- Increment and decrement operators:
  - Are only for convenience (you do not have to use them)
  - Are most often used in loops on index variable

RPG	Java

`C EVAL IDX = 1`
`C DOW (IDX <= 10)`
`C* : Some code`
`C EVAL IDX = IDX +1`
`C ENDDO`
	`idx = 0;`
	`while (idx < 10)`
	`{`
	`  // some code`
	`  idx ++;`
	`}`

```
a = 5;
b = 5;
x = ++a + 50; // x is 56 after execution
y = 50 + b++; // y is 55 after execution
```

Unary operators

- RPG and Java's arithmetic unary operators:

Operation	Java	RPG	Java example	Comment
Increment before	++		`j = ++i;`	*i incremented, then assigned*
Increment after	++		`j = i++;`	*i assigned, then incremented*
Decrement before	--		`j = --i;`	*i decremented, then assigned*
Decrement after	--		`j = i--;`	*i assigned, then decremented*
Unary minus	-	-	`j = -i;`	*j = i * (-1)*
Unary plus	+	+	`j = +i;`	*j = i * (+1)*
Logical negation	!	NOT	`boolean f = !(i > j);`	***true** => **false**, **false** => **true***
Bitwise complement	~		`byte b = ~c;`	*00001111 => 11110000*

Relational and logical (1 of 2)

Note how comparisons are done

RPG	Java

`C IF (prefix = 'A')`
`C* some code`
`C* more code`
`C ENDIF`
	`if (prefix == 'A')`
	`{`
	`// code`
	`}`

- Expressions resulting in true or false values can be combined using logical operators.

Relational and logical (2 of 2)

Relational

RPG XX	RPG Expression	Java	Description
EQ	=	==	equal
NE	NOT =	!=	not equal
GT	>	>	greater than
LT	<	<	less than
GE	>=	>=	greater than or equal
LE	<=	<=	less than or equal

Logical

RPG XX opCodes	RPG opCodes	Java	Description
ORxx	OR	|| or |	logical or
ANDxx	AND	&& or &	logical and
NOT		!	logical not, or negation
		^	exclusive or

Relational example

RPG

```
C     A          IFGE      2
C     A          ANDLE     20
C     X          ANDEQ     330.0
```

Java

```
if ((a>=2)&& (a<=20)
    && (x == 330.0))
```

Note double equals signs: ==

RPG IV

```
C          IF        ((a>=2) AND (a<=20) AND
C                     (x=330.0) )
```

DeMorgan's rule

- It is useful in any language to know how to propagate a negation through a relational expression.
  - DeMorgan's Rule explains how to do this
    - To negate an AND expression:
      - Negate each operand; change AND to OR
    - To negate an OR expression:
      - Negate each operand; change OR to AND

```
if ( ! ( (day == MONDAY) && (age >= 65) ) )
```

```
if ( (day != MONDAY) || (age < 65) )
```

```
if ( ! ( (day != MONDAY) || (age < 65) ) )
```

```
if ( (day == MONDAY) && (age >= 65) )
```

Bitwise operators

- RPG has TESTB, BITON, BITOFF op-codes.
- Java has operators.

Bitwise operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive OR
~	Bitwise negation
< <	Left Shift
> >	Right Shift
> > >	Zero fill right shift

- They work only on integer types.

A “bit wiser"

- Use AND (&) and OR (|) operators plus a "bit mask" to emulate RPG bit op-codes.

RPG Opcode	Java Operator	Mask
TESTB	&	Bit to be tested is '1', rest are '0'
BITON	|	Bit to be set on is '1', rest are '0'
BITOFF	&	Bit to be set off is '0', rest are '1'

```
if ((variable & 0x0002)>0) // is bit 2 on?
  // the second bit is on, do something...
variable = variable | 0x0002; // bit 2 on
variable = variable & 0xFFFD; // bit 2 off
```

0000 0000 0000 0010

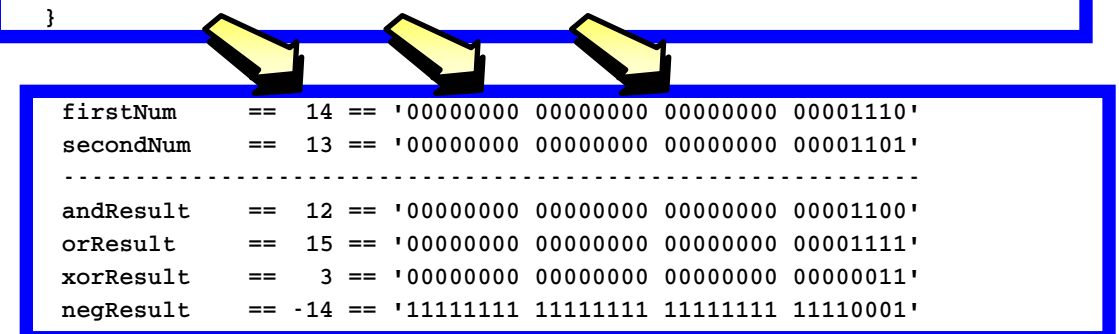
0000 0000 0000 0010

1111 1111 1111 1101

Bitwise examples

```
public static void main(String args[])
{
    int firstNum  = 14;
    int secondNum = 13;
    int andResult, orResult, xorResult, negResult;

    andResult = (byte) (firstNum & secondNum);
    System.out.println("AND result: " +  andResult);
    orResult  = (byte) (firstNum | secondNum);
    System.out.println("OR  result: " +  orResult);
    xorResult = (byte) (firstNum ^ secondNum);
    System.out.println("XOR result: " +  xorResult);
    negResult = (byte) (~firstNum);
    System.out.println("NEG result: " +  negResult);
}
```



Example: Shift

```
public class Shift
{
    public static void main(String args[])
    {
        int firstNum = 2;
        int firstResult, secondResult;

        firstResult = (firstNum << 2);
        System.out.println("The first result is : " + firstResult);

        secondResult = (firstResult >> 1);
        System.out.println("The second result is : " +  secondResult);
    }
}
```

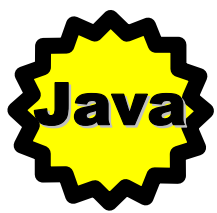

```
firstNum     == 2 == '00000000 00000000 00000000 00000010'
-------------------------------------------------------
firstResult  == 8 == '00000000 00000000 00000000 00001000'
secondResult == 4 == '00000000 00000000 00000000 00000100'
```

Conditional operator

- Conditional operator '? :'
  - Also called a ternary operator
- Short form for **if** statement
  - When only binary decision to make

```
result = (idx == 20) ? 30 : 35;

// same as...
if (idx == 20)
  result = 30;
else
  result = 35;
```



Operator precedence (1 of 2)

- For RPG, here it is the operator precedence:

( )
Built-in functions, user-defined procedures
unary +, unary -, NOT
**
*, /
binary +, and Binary -
=, <>, >, >=, <, <=
AND
OR

Operator precedence (2 of 2)

- For Java, here is the operator precedence:

Operators	Associativity
++, --, ~, !, -, (type cast)	Right-to-left
*, /, %	Left-to-right
+, -	Left-to-right
+ (concat)	Left-to-right
>>>	Left-to-right
>>, <<	Left-to-right
<, <=, >, >=	Left-to-right
instanceof	Left-to-right
= =, !=	Left-to-right
&	Left-to-right
^	Left-to-right
&& |	Left-to-right
| |	Left-to-right
?:	Right-to-left
=, *=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |=	Right-to-left

Java

Math functions

```
D FirstNum        S              4S 0 INZ(40)
D SecondNum       S              4S 0 INZ(65)
D Result          S              4S 0
C                   EVAL       Result = FirstNum - SecondNum
C     Result        DSPLY      -25
C                   EVAL       Result = %ABS(FirstNum - SecondNum)
C     Result        DSPLY      25
```

RPG IV

```
int firstNum  = 40;
int secondNum = 65;
int result1, result2;

result1 = firstNum - secondNum;
result2 = Math.abs(firstNum - secondNum);

System.out.println(result1);
System.out.println(result2);
```

Java

-25

25

More math methods (1 of 2)

Method	Description
abs(double)	Returns the absolute value of a double value.
abs(float)	Returns the absolute value of a float value.
abs(int)	Returns the absolute value of an int value.
abs(long)	Returns the absolute value of a long value.
acos(double)	Returns the arc cosine of an angle, in the range of 0.0 through pi.
asin(double)	Returns the arc sine of an angle, in the range of -pi/2 through pi/2.
atan(double)	Returns the arc tangent of an angle, in the range of -pi/2 through pi/2.
atan2(double,double)	rectangular coordinates (b, a) to polar (r, theta).
ceil(double)	Returns the smallest (closest to negative infinity) double value that is not less than the argument and is equal to a mathematical integer.
cos(double)	Returns the trigonometric cosine of an angle.
exp(double)	Returns the exponential number (eg, 2.718) raised to the power of a double value.
floor(double)	Returns the largest (closest to positive infinity) double value that is not greater than the argument and returns the greater of two double values.

More math methods (2 of 2)

Method	Description
max(float, float)	Returns the greater of two float values.
max(int, int)	Returns the greater of two int values.
max(long, long)	Returns the greater of two long values.
min(double, double)	Returns the smaller of two double values.
min(float, float)	Returns the smaller of two float values.
min(int, int)	Returns the smaller of two int values.
min(long, long)	Returns the smaller of two long values.
pow(double, double)	Returns of value of the first argument raised to the power of the second argument.
random()	Returns a random number between 0.0 and 1.0.
rint(double)	Returns the closest integer to the argument.
round(double)	Returns the closest long to the argument.
round(float)	Returns the closest int to the argument.
sin(double)	Returns the trigonometric sine of an angle.
sqrt(double)	Returns the square root of a double value.

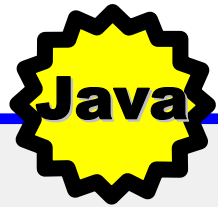
Maximum in RPG

```
D*Name+++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++
D First           S               4S 0 INZ(9)
D Second          S               4S 0 INZ(22)
D Third           S               4S 0 INZ(2)
D Result          S               4S 0
D MAX             PR              4S 0
D  First                          4S 0
D  Second                         4S 0
C*
C                   EVAL      Result = MAX(First:Second)
C     Result        DSPLY
C                   EVAL      Result = MAX(First:Third)
C     Result        DSPLY
C                   MOVE      *ON            *INLR
 * Start of procedure MAX...
P MAX             B
D                 PI              4S 0
D  First                          4S 0
D  Second                         4S 0
C                   IF        First>Second
C                   RETURN    First
C                   ELSE
C                   RETURN    Second
C                   EndIF
P MAX             E
```

RPG IV

max procedure in RPG

Maximum in Java



Using Java-supplied max method

```
public class Max
{
      public static void main (String args[])
      {
          double firstNum = 9;
          double secondNum = 22;
          double result;
          result = Math.max(firstNum, secondNum);
          System.out.println("The result is: " + result);
      }
}
```

Topics covered

- Basics
- Comments
- Variable naming and keywords
- Statements
- Expressions and operators
- Arithmetic manipulations

Unit summary

Having completed this unit, you should be able to:

- Define a Java program using statements, variables, expressions, and comments
- Use Java's inline documentation feature
- Describe the Java operators used to manipulate Java variables