Lecture 2:
Fundamental Programming Structures In Java

Wholeness of the Lesson

Java is an object-oriented programming language that supports both primitive and object data types. These data types make it possible to store data in memory and modify it or perform computations on it to produce useful output. Execution of a program is an example of the "flow of knowledge"—the intelligence that has been coded into the program has a chance to be expressed when the program executes.

Maharishi's Science of Consciousness locates three components to any kind of knowledge: the knower, the object of knowledge, and the process of knowing. These can be found in the structure of a Java program: the "knower" aspect of the program is the intelligence underlying the creation of Java objects—a Java *class*. The *data* that a program works on, which is stored in program variables of either primitive or object type, is the "object of knowledge." And the Java methods, which act on the data, are the "process of knowing."

Outline of Topics

- Data Types:
  - The Primitive Types
  - Other data types at the basis of object creation
- Operators In Java
  - Arithmetic Operators
  - Increment and Decrement Operators
  - Relational And Boolean Operators
  - Bitwise Operators
- Control Flow:
  - Conditional Logic
  - While Loops
  - For loops
  - The switch Statement

Outline of Topics

- Data Types:
  - The Primitive Types
  - Other data types at the basis of object creation
- Operators In Java
  - Arithmetic Operators
  - Increment and Decrement Operators
  - Relational And Boolean Operators
  - Bitwise Operators
- Control Flow:
  - Conditional Logic
  - While Loops
  - For loops
  - The switch Statement

Data Types: The Primitive Types

- Every variable must have a declared type
- Eight primitive types: `int`, `short`, `long`, `byte`, `float`, `double`, `char`, `boolean`

Type	Storage Requirement	Range (Inclusive)
`int`	4 bytes	-2^{31} to $2^{31} - 1$
`short`	2 bytes	-2^{15} to $2^{15} - 1$
`long`	8 bytes	-2^{63} to $2^{63} - 1$
`byte`	1 byte	-2^{7} to $2^{7} - 1$ (-128 to 127)
`float`	4 bytes	6 - 7 significant (decimal) digits
`double`	8 bytes	15 significant (decimal) digits

- Note： 1 byte = 8 bits
- `boolean` has just 2 values: *true* and *false*.

Variables In Java

- Variables in Java store values, like numbers and other data
- A variable in Java always has a type; a variable is *declared* by displaying the type, followed by the variable name.
- Examples of declaring variables:

```
double salary;
int amount;
boolean found;
```

- Variable names consist of digits, letters and underscores, but may not begin with a digit.

Variables In Java

- *Variable Initialization.* A variable is initialized by using the *assignment operator* (=) to specify a value for a declared variable.

  Example:

  ```
  int sum;
  sum = 0;
  ```

  OR

  ```
  int sum = 0;
  ```

- *Coding Style:*
  - variable names begin with lower case letter
  - variable names composed of multiple words written so that each new word (except the first) begins with a capital letter, but all other letters are lower case
    Example:

    ```
    myExamScore
    ```

  - underscores should *not* be used typically in variable names
  - for *constants*, capitals and underscores are used (discussed later)

    ```
    CONSTANTS_LIKE_THIS
    ```

Characters

- In Java, single characters are represented using the data type **char**.
  - `char ch1;`
- Character constants are written as symbols enclosed in single quotes.
  - `char ch2 = 'X';`
- Characters are stored in a computer memory using some form of encoding.

ASCII Encoding

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
10	lf	vt	ff	cr	so	si	dle	dc1	dc2	dc3
20	cd4	nak	syn	etb	can	em	sub	esc	fs	gs
30	rs	us	sp	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	}	|	~	del		

For example, character 'O' is 79 (row value 70 + col value 9 = 79).

Character Processing

```
char ch1 = 'X';
```

Declaration and initialization

```
System.out.print("ASCII code of character X is " +
                 (int) 'X' );

System.out.print("Character with ASCII code 88 is "
        + (char)88 );
```

Type conversion between int and char.

```
'A' < 'c'
```

This comparison returns true because ASCII value of 'A' is 65 while that of 'c' is 99.

Introducing Other Types: Reading Console Input

- We have examined primitive data types (int, char, boolean, etc) and how variables are declared using these types. Most of the data types used in Java, however, are not built into the language, but are created through *class definitions*.

- Data types are special kinds of *types* that store data values. Other types may play other roles in an application. A `Scanner` is a type that makes it possible to work with console input.

- Types and class definitions will be the topic of Lesson 3.

Samples

- Scanner (as of j2se5.0)

```
//can think of Scanner as a special data type
Scanner sc = new Scanner(System.in);
System.out.print("Type your name: ");
System.out.println("you wrote: " + sc.nextLine());
System.out.print("Type your age: ");
System.out.println("your age: " + sc.nextInt());
sc.close();  //don't forget to close

//output
Type your name: Jim Stevens
you wrote: Jim Stevens
Type your age: 36
your age: 36
```

See package lesson2.scanner

Main Point

Variables in Java are *declared* and *initialized* to provide room in RAM for the data that is to be stored. Pure consciousness manifests as individuals in space.

Outline of Topics

- Data Types:
  - The Primitive Types
  - Other data types at the basis of object creation
- Operators In Java
  - Arithmetic Operators
  - Increment and Decrement Operators
  - Relational And Boolean Operators
  - Bitwise Operators
- Control Flow:
  - Conditional Logic
  - While Loops
  - For loops
  - The switch Statement

Operators In Java: Arithmetic Operators

- Standard binary operations represented in Java by `+`, `-`, `*`, `/`. Also the modulus operator `%`.
- Division by 0: for `int`s, an exception is thrown; for floating point numbers, the value is `NaN`
- The operators `+=`, `*=`, `/=`, `-=`, `%=`

Operator	Usage	Meaning
`+=`	`a += b;`	`a = a + b;`
`-=`	`a -= b;`	`a = a - b;`
`*=`	`a *= b;`	`a = a * b;`
`/=`	`a /= b;`	`a = a / b;`
`%=`	`a %= b;`	`a = a % b;`

Operators In Java: Increment and Decrement Operators

- Variables having primitive numeric type can be incremented and decremented using “++” and “--" respectively (char types can be incremented like this too, but it is not a good practice to do this)

- Example:

```
int k = 1;
k++;   //postfix form
++k;   //prefix form
```

Operators (continued)

- Differences between Postfix and Prefix forms:

Postfix form: variable k is incremented *after* expression is evaluated

```
int k = 0;
int m = 3 * k++;  //m equals 0, k equals 1
```

Prefix form: variable q is incremented *before* expression is evaluated

```
int q = 0;
int n = 3 * ++q;  //n equals 3, q equals 1
```

- Commonly used in for loops (coming up soon) but also in traversing arrays (also coming up soon).

Operators In Java: Relational And Boolean Operators

- **Relational:** == (equals), != (not equals), < (less than), <= (less than or equal to), >, >= (greater than, greater than or equal to)
- **Logical:** &&, ||, ! . Short-circuit evaluation.

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	
T	F
F	T

- **Ternary:** *condition* ? *expression1* : *expression2* – evaluates to expression1 if condition is true, expression2 otherwise
- Example:

```
CustomerStatus =
     (income > 100000) ?
           Gold : SILVER;
```

is equivalent to this logic:

```
if( income > 100000 )
       customerStatus = Gold
else
       customerStatus = SILVER
```

Operators In Java: Bitwise Operators

- & (and), | (or), ^ (xor), ~ (not), << (left shift), >> (right shift)

&	1	0
1	1	0
0	0	0

|	1	0
1	1	1
0	1	0

^	1	0
1	0	1
0	1	0

~1 = 0

~0 = 1

- Examples of << and >>

0000 1111 >> 2 = 0000 0011 (right shift by 2 is same as dividing by 4)
[15 >> 2 = 15/4 = 3]

0000 1111 << 2 = 0011 1100 (left shift by 2 is same as multiplying by 4)
[15 << 2 = 15 * 4 = 60]

Mathematical Constants And Functions

- Special math functions and constants are available in Java by using the syntax
  `Math.<constant>` and `Math.<function>`

- Examples:

```
Math.PI   (the number pi - approximately 3.14159)

Math.pow(a,x)  (the number a raised to the power x)

Math.sqrt(x)  (the square root of x)
```

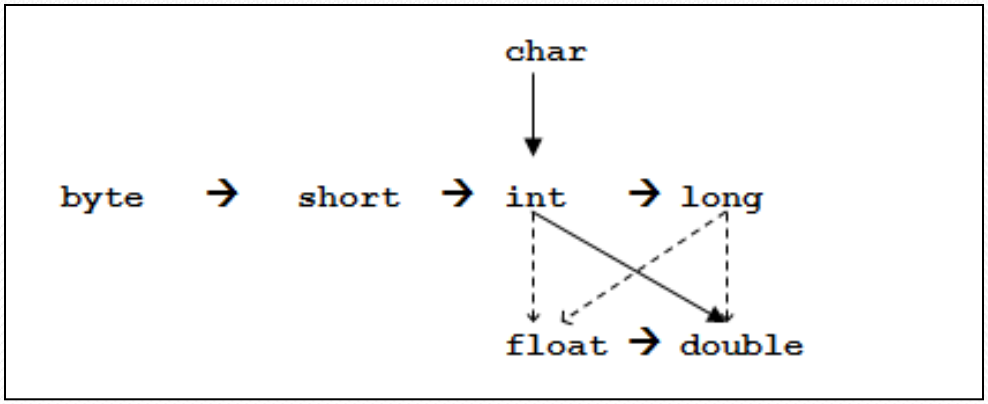
- For this course, we have a `RandomNumbers` class (which uses the Java Random class). Its methods can be accessed in the same way the methods of `Math` class can.

- Examples:

```
//produces a randomly generated int
int n = RandomNumbers.getRandomInt();

//produces a randomly generated int in the range 3..11, inclusive.
int m = RandomNumbers.getRandomInt(3,11);
```

Conversions Between Numeric Types



- Solid arrows indicate automatic type conversions that do not entail information loss
- `int` to `float`, `long` to `float`, and `long` to `double` are automatic, preserve number of digits to left of decimal, but may lose precision:

Example:

```
int n = 123456789;
float f = n;  //f is 123456792.000000,
```

See `lesson2.datatypeconversion`

- *Automatic promotion of integral types.* When a binary operation (like +, *, or any shift operator) is applied to values of type `byte` or `short`, the types are promoted to `int` before the computation is carried out.

Example: The following produces a compiler error. Why?

```
byte x = 5;
byte y = 7;
byte z = x + y;
```

- When values of different type are combined (via addition, multiplication or other operations), a type conversion occurs to arrive at just one common type.

  Most important cases:
  - a double combined with another primitive numeric type results in a double
  - an int combined with a smaller type (byte, short) results in an int.

- (A complete list of rules is given on p. 60 of Core Java, Vol 1, 10th ed.)

- Other conversions can be "forced" by casting.

  Example:

```
double x = 9.997;
int y = (int) x;  //y has value 9
```

- "Rounding" is usually preferable to casting and is done by using the round function of the Math class:

```
Example:
double x = 9.997;
int nx = (int)Math.round(x); //round returns a long,
 //so cast is necessary
```

- It is possible to cast a `long` to an `int`, an `int` to a `byte`, and so forth. In such cases, casting is accomplished by removing as many high-order bits as necessary to produce a number of the narrower type.

```
Example:
int x = 130;              ( = 000. . .00 1000 0010 as an int)
byte b = (byte)x;         ( = 1000 0010 as a byte = -126)
```

Operator Precedence and Association Conventions

Examples:

`a && b || c` means `(a && b) || c` (operator precedence)

`a += b += c` means `a += (b += c)` (association to the right)

- See page 64 of Core Java, Vol 1, 10th ed. for a list of all the rules
- Important precedence rules to know:
  - *, / , % precede + and –
  - ++ and -- have almost the highest precedence
  - = (assignment) associates to the right: `a = b = c` means `a = (b = c)`
  - *Tip:* be able to use the chart on p. 64 to read syntax

Example. Use the precedence rules table (below and on p.53) to evaluate:

```
7 & 13 >> 2 ^ 5
```

Solution:

```
7 & 13 >> 2 ^ 5
  =(7 & (13 >> 2)) ^ 5 (preced rules)
  =(7 & 3) ^ 5)       (since 13>>2 =3)
  = 00000011 ^ 00000101
  = 00000110
  = 6
```

Table 3-4. Operator precedence

Operators	Associativity
[] . () (method call)	left to right
! ~ ++ +(unary) -(unary) ()(cast) new	right to left
* / %	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
||	left to right
?: (ternary operator)	right to left
= += -= *= /= %= &= |= ^= <<= >>= >>>=	right to left

Main Point

Variables of primitive type can be combined to form expressions through the use of *operators*. The Java syntax requires one to observe rules for forming expressions – precedence rules, type conversion rules, and others. Pure consciousness, likewise, also has laws that govern its self-combining. The self-combining dynamics of pure consciousness – like the self-interacting dynamics of the unified field – give rise to "everything": All thoughts, all knowledge, and, if the unified field is really the same as pure consciousness, all manifest existence.

Outline of Topics

- Data Types:
  - The Primitive Types
  - Other data types at the basis of object creation
- Operators In Java
  - Arithmetic Operators
  - Increment and Decrement Operators
  - Relational And Boolean Operators
  - Bitwise Operators
- Control Flow:
  - Conditional Logic
  - While Loops
  - For loops
  - The switch Statement

Control Flow: Conditional Logic

- The conditional statement in Java has the following form:

```
if(condition) statement
```

Here, condition is any boolean statement (statement that evaluates to true or false)

- The *statement* may in fact be an entire block of code, in this form:

```
if(condition) {
     statement1;
     statement2;
     …
}
Example:

     if(sales >= target) {
            performance = "Satisfactory";
            bonus = 100;
     }
```

Control Flow: Conditional Logic

- Another form of conditionals is the *"if...else"* form:

```
if(condition) statement1
else statement2
```

Example:

```
    if(sales >= target) {
          performance = "Satisfactory";
          bonus = 100;
    }
    else {
          performance = "Unsatisfactory";
          bonus = 0;
    }
```

- Can have repeated “else if””s .

Example:

```
if(sales >= 2 * target) {
        performance = "Excellent";
        bonus = 100;
}
else if (sales >= target) {
        performance = "Satisfactory";
        bonus = 50;
}
else { //sales < target
        performance = "Unsatisfactory";
        bonus = 0;
}
```

Control Flow: While Loops

- The general form of a `while` loop is

  `while (`*`condition`*`)` *`statement`*

  where *condition* is a boolean expression.

- The general form of a `do…while` loop is

  ```
  do statement while (condition)
  ```

- Typically, `do…while` is used in place of `while` when it is necessary for `statement` to execute at least once (even if `condition` is always false).

Examples

```
//while loop
int balance = -50;
int goal = 100;
while(balance < goal) {
  //read payment from user input
  balance += payment;
}
```

```
//do..while loop
Scanner sc = new Scanner(System.in);
do{
  System.out.print("Payment amount? ");
  payment = sc.nextDouble();
  balance += payment;
  System.out.println("Make another payment?
(Y/N)");
  input = sc.next();
} while(input.equals("Y"));
```

Examples – the while(true) Construct

- Use the `while(true)` form when the `statement` requires processing before a condition can be evaluated. To exit the loop, use a `break` statement.

Example:

```
Scanner sc = new Scanner(System.in);
while(true) {
    System.out.print ("Enter a positive number: ");
    int value = sc.nextInt();
    if(value <= 0){
        break;
    }
}
System.out.println("The value you enter must be
positive.");
```

while(true) - continued

- Also used sometimes in creating a server (for a client/server system); in this case, the while loop never stops (until the server itself stops):
- *Using* `break` *in while loops.* When a `break` statement occurs, the while loop is exited and execution resumes as it would if the condition in the while loop had just failed. When possible, use `while` *without* a `break` statement (by selecting the condition for the `while` loop carefully) – sometimes though `break` statements are necessary.

Control Flow: for Loops

- General form of the `for` loop:

  for(*initialization*; *condition*; *increment*) *statement*

- Sample code: see the *reference example*, `Main` class.
- All three parts of the `for` expression are optional. The expression

  `for( ; ; )` *statement*

  means the same as

  `while(true)` *statement*

Examples

```
//standard
for(int i = 0; i < max; ++i) {
     //do something
}
```

Note: Since `i` is declared in the for loop, it cannot be referenced outside of the for block. If you need to use it outside the block, this code should be used:

```
int i;
for(i = 0; i < max; ++i) {
     //do something
}
//now i can be referenced here
```

or, equivalently,

```
int i=0;
for( ; i < max; ++i) {
     //do something
}
//now i can be referenced here
```

Examples - continued

More than one variable can be initialized, and more than one increment statement can be used; commas separate such statements.

```
for(int i = 1, j = max; i * j <= balance; i++, j--) {
     //do something
}
```

Complex conditions are allowed in the condition slot:

```
for(int i = 0; (i+1) * value > min && i * value < max; i = i +2) {
     //do something
}
```

Nested for loops – example

```
for(int i = 0; i < n; ++i){
    for(int j = 0; j < n; ++j){
          System.out.print(" *");
    }
    System.out.println();
}
//output for n = 5

* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

Control Flow: The switch Statement

- The `switch` statement is a convenient shorthand for writing "`if..else`" statements, when the values being tested are `int`s, `char`s, `String`s, or `enum`s. (Note: `enum`s will be discussed in Lesson 3.)
- General form of the `switch` statement:

```
switch(val) {
     case x:
               statement_x;
               break;
     case y:
               statement_y;
               break;
     …

     default:
               default_statement;
}
```

- The `break` in each case ensures that only one case is exectuted. If you forget to insert the break, later cases will continue to be tested and executed.
- A `default` case should typically be provided, to handle all cases not specified in the case statements.

“Fall-through” Behavior

- “Fallthrough behavior” occurs when break statements are omitted: Cases are examined and, as soon as a match is found, the corresponding statement is executed, and all subsequent case statements are also executed, until a break is encountered. If no matches are found, then the default statement is executed if there is one.
- Example of “fallthrough behavior”:

```
Scanner sc = new Scanner(System.in);
System.out.print("Pick an integer in the range 1..9");
int val = sc.nextInt();
System.out.println();
switch(val) {
    case 2:
    case 4:
    case 6:
    case 8:
        System.out.println("You chose an even number.");
        break;
    default:
        System.out.println("You chose an odd number.");
}
```

Main Point

Control flow is supported in Java via the *if..else, for, while, do..while, switch* [and also *for each*] language elements. Loops are the CS analogue to the self-referral performance at the basis of all creation, whereas branching logic mirrors the tree-like hierarchy of natural laws that guide the activity in each layer of creation.

Connecting the Parts of Knowledge With the Wholeness of Knowledge

1. In Java, variables of primitive type can be combined using operators to form expressions, which may be evaluated to produce well-defined output values.

2. On a broader scale, objects in Java are "combined" by way of "messages" between objects, which collectively result in the behavior of a Java application.

---

3. **Transcendental Consciousness:** Pure consciousness is the field beyond type and interaction; it is the field of *unbounded awareness* and *infinite silence*.
4. **Wholeness moving within itself**: In Unity Consciousness, one observes that this unbounded silent quality of awareness is spontaneously present at all levels of action in the world, and not just relegated to the transcendental field.