Language Fundamentals

The objectives of this chapter are:

- To explain Java statements and expressions
- To provide an overview of the fundamental data types
- To explain operators and precedence
- To discuss the general syntax of Java

Java Statements

- Statements in Java can be one of the following:
  - Variable Declaration
  - Control statement (if, switch, while, for, method invocation)
  - Expression
  - Block
  - Comment

- Variables are symbolic names for memory storage.
- Control statements affect the execution flow of the program.
- Expressions produce a value.
- Blocks group statements and define variable scope.
- Comments are discarded during compilation. They are necessary for documenting code.

Comments

- Java supports three types of comments
  - Line comments
  - C style comments
  - javadoc
- The general forms are as follows:

```
// line comment.  All text from the first // to the end of the
// line is a comment.

/* C-Style Comment.  These comments can span multiple lines.
The compiler ignores all text up until */

/** Javadoc comment.  The compiler ignores this text too.
However, the javadoc program looks for these comments and
interprets tags for documentation generation purposes:

     @author Craig Schock
     @version 1.7
     @see java.lang.Object
*/
```

A quick note about javadoc

- javadoc comments must immediately precede the item they are documenting (class, method, attribute, etc)
- Some Javadoc tags:
  - @see *class-name*
  - @see *full-class-name*
  - @see *full-class-name#method-name*
  - @version *text* (class def only)
  - @author *text* (class def only)
  - @param *parameter-name description* (method def only)
  - @return *description* (method def only)
  - @exception *full-class-name description* (method def only)
  - @deprecated *explanation*
  - @since *version*

Variable Declarations

- Like most compiled languages, variables must be declared before they can be used.
- Variables are a symbolic name given to a memory location.
- All variables have a type which is enforced by the compiler
  - However, Java does support polymorphism which will be discussed later
- The general form of a variable declaration is:

```
type variable-name [= value][,variable-name[= value]];
```

- Examples:

```
int total;
float xValue = 0.0;
boolean isFinished;
String name;
```

initialization

note the semicolon

Types in Java

- In a variable declaration, the *type* can be:
  - a fundamental data type
  - a class
  - an array

- Java has 8 fundamental data types.

- Fundamental data types are not Object-Oriented. They are included as part of the language primarily for efficiency reasons.

- The eight types are**: byte, char, short, int, long, float, double, and boolean.**

Fundamental Data Types

- All primitive types in Java have a defined size (in bits). This is needed for cross platform compatibility.
- Each type has a defined set of values and mathematical behaviour.
- Six of the types are numeric (byte, short, int, long, float, double)
- The *char* type holds characters
- The *boolean* type holds truth values

Integral Data Types

- 4 types based on integral values: byte, short, int, long
- All numeric types are signed. There are NO unsigned types in Java.

Type	Size	Range
byte	8 bits	-128 through +127
short	16 bits	-32768 through +32767
int	32 bits	-2147483648 through +2147483647
long	64 bits	-9223372036854775808 through +9223372036854775807

Floating point Data Types

- 2 types based on floating point values: float and double
- Storage conforms to IEEE 754 standard
- floats store 7 significant digits. doubles store 15.

Type	Size	Range
float	32 bits	$-3.4 * 10^{38}$ through $+3.4 * 10^{38}$
double	64 bits	$-1.7 * 10^{308}$ through $+1.7 * 10^{308}$

Character data type

- The char type defines a single character
- In many other programming languages, character types are 8-bits (they store ASCII values). In Java, character types are 16-bits.
- Java characters store characters in *unicode* format.
- Unicode is an international character set which defines characters and symbols from several different world languages.
  - Unicode includes ASCII at its low range (0-255)
- Characters can be converted to integers to perform mathematical functions on them.

Boolean data type

- The boolean type defines a truth value: *true or false*.
- Booleans are often used in control structures to represent a condition or state.
- booleans CANNOT be converted to an integer type.

Class data type

- When a fundamental data type is declared, the memory necessary to hold an element of that type is reserved by the compiler. The compiler knows how much memory to reserve based on the size of the type (8 bits, 16 bits, etc).
- The variable name refers to the memory that was reserved by the compiler.
- If the type is a class, the compiler reserves enough memory to hold a *reference* to an instance of the class. The compiler DOES NOT reserve memory to hold the object. Objects must be created dynamically.
- More on this in a later chapter.

Variable/Identifier names

- Java has a series of rules which define valid variable names and identifiers.
  - Identifiers can contain letters, numbers, the underscore (_) character
  - Identifiers must start with a letter, underscore or dollar sign.
  - Identifiers are case sensitive
  - Identifiers cannot be the same as reserved Java keywords.

valid:

myName	total	total5
_myName	_total	___total5

invalid:

1myName	total#	default	My-Name

Reserved Words in Java

boolean
byte
char
short
int
long
float
double
void

false
null
true

abstract
final
native
private
protected
public
static
synchronized
transient
volatile

break
case
catch
continue
default
do
else
finally
for
if
return
switch
throw
try
while

class
extends
implements
interface
throws

import
package

instanceof
new
super
this

byvalue
cast
const
future
generic
goto
inner
operator
outer
rest
var

reserved for future use.

Reserved Symbols in Java

- ; Semi-colon indicates the end of a statement.
- () Parentheses used in several places including:
  - Overrides the default precedence in an expression that contains multiple operators.
  - Indicate a method or casting operator.
  - Surround the logic test in an if statement, e.g.if(*test*)...
- [] Array declaration.
  Array element specification
- {} Curly braces
  - enclose the fields and methods of a class.
  - enclose the code for a method.
  - Enclose the code body for an if statement, a for loop statement, a synchronized block, and initial values for an array declaration.
- // Indicates a single line comment
- /*..*/ Bracket a set of comments that can span more than one line.
- /**..*/ Same as /* */ except that the double asterisks tell the javadoc program to use the comments in its output.

Reserved Symbols in Java Cont'd

- : Colon is used in switch statements and the conditional operator.
- "xx“ Double quotes surround a string literal.
- 'x‘ Single quotes surround a character literal.
- +,-,etc Operator symbols.
- % Used with the Formatter class and print to specify output formats. (J2SE 5.0)
- ? Used with the conditional operator
  x = boolean ? y : z;

Tips for good variable names

- Use a naming convention
- Use names which are meaningful within their context
- Start **Class** names with an **Upper case letter**. **Variables** and other identifiers should start with a **lower case letter**.
- Avoid using _ and $.
- Avoid prefixing variable names (eg. _myAge, btnOk)
  - This is often done in languages where type is not strongly enforced.
  - If you do this in Java, it is often an indication that you have not chosen names meaningful within their context.
- Separate words with a capital letter, not an underscore (_)
  - myAccount, okButton, aLongVariableName
  - avoid: my_account, ok_button, and a_long_variable_name

Initializing variables

- Although Java provides ALL variables with an initial value, variables should be initialized before being used.
- Java allows for initializing variables upon declaration.
- It is considered good practice to initialize variables upon declaration.
- Variables declared within a method must be initialized before use or the compiler with issue an error.

```
int total = 100;
float xValue = 0.0;
boolean isFinished = false;
String name = "Zippy The Pinhead";
```

Constant Values

- A variable can be made **constant** by including the keyword ***final*** in its declaration.
- By convention, the names of variables defined as **final** are **UPPER CASE**.
- Constants allow for more readable code and reduced maintenance costs.
- Final variables must be initialized upon declaration.

```
final int MAX_BUFFER_SIZE
= 256;
final float PI=3.14159;
```

Literal Definitions

- Integrals can be defined in decimal, octal, or hexadecimal
  - Integrals can be long or int (L or l)
  - Integrals are int by default
- Floating point numbers can be defined using standard or scientific notation
  - double by default
  - F indicates float
- Single characters are defined within single quotes
  - can be defined as unicode
  - can be "special" character (eg. '\n')
- Strings are defined by double quotes.

```
Decimal: 0 1 10 56 -35685
Octal: 01 056 07735
Hex: 0x1 0x6F 0xFFFF
long: 7L  071L  0x4FFL
```

```
Standard: 3.14 9.9  -37.1
Scientific: 6.79e29
float: 7.0F  -3.2F
```

```
'c'   '\n'   '\r'   '\025'   \u34F6
```

```
"this is a String."
```

Special Characters

- Java defines several "special" characters. All are preceded by a backslash (\) character:

\n	Newline (linefeed character)
\r	Return (carriage return character)
\t	Horizontal Tab
\\	Back slash
\'	Single Quote
\"	Double Quote
\###	Octal represented by octal number
\u####	Unicode character (hex)

Expressions

- An expression is anything which evaluates to something.
- Expressions are a combination of operators and operands
- Operators are defined by symbols:
  - Arithmetic (+, -, *, /, %)
  - Assignment (=, +=, -=, *=, /=)
  - Increment and decrement (++, --)
  - relational operators (==, !=, <, <=, >, >=)
  - logical operators (||, &&) (note: logical or, logical and)
- The order of operations is defined through precedence.

Operator Precedence

Order	Operators	Name
1	. [] (parameters)	array indexes, parms
2	++ -- ! ~ instanceof	unary operators
3	new (type)expr	creation and cast
4	* / %	multiply and divide
5	+ -	addition and subtraction
6	<< >> >>>	bitwise shifts
7	< > <= >=	relational operators
8	!= ==	equality operators
9	&	bitwise and
10	^	bitwise xor
11	|	bitwise or
12	&&	logical and
13	||	logical or
14	?:	conditional
(ternary) operator		
15	= += -= *= /=	assignment

Note: two operators of the same precedence will be evaluated based on their associativity Usually, associativity is evaluated from left to right. Associativity of assignment is right to left

Assignment

- The assignment operator has the lowest precedence of all operators
  - It is always evaluated last
- Assignments can be used to assign the value of an expression to a variable: `variable = expression;`
- In an assignment, the previous value of the variable is overwritten by the value of the expression.
- Examples:

```
x = x + 1;
isVisible = true;
etaInSeconds = distance/speedOfLight;
```

Arithmetic Operators

- Arithmetic operators in Java behave as one would expect

Addition: 3+x+7+9
Subtraction: 17-2-a-35
Multiplication: x*y
Division: 100/percent
Modulus: 10%3 (remainder after division)

Compound expressions:
3*x + 5*y - 37 principle + (principle*interest)
3*x*x + 2*x + d

Brackets can be used to override precedence
x*(3+y) / (z-27)

Assignment -Revisited

- Java also defines assignment operators which have an implied mathematical function

```
x = x + 1;          x += 1;
x = x + y + 5;      x += y + 5;
x = x * (z * 50);   x *= z * 50;
x = x / 10;         x /= 10;
```

- These were originally added to the C language so that the programmer could help the compiler optimise expressions. It is generally better to avoid using these assignment operators.

- Be aware of the precedence issues:

```
x *= y + 5;    does not equal        x = x * y + 5;
          instead, it equals     x = x * (y + 5);
```

- Assignment ALWAYS has the lowest precedence.

Integer Arithmetic

- What are the problems with the following code?

```
int x;
long y;
int z;

[... x and y are initialized with some values]

z = x + y;
```

- Can we add x to y?
  - They are different types
  - but they are both integral.

- Can we assign the results of the expression x+y to z?
  - z is an int (32 bits)
  - y is a long (64 bits).

Type Conversions

- Java allows values of one type to be converted to differing types.

- Conversion occurs in the following situations:
  - During assignment
  - During arithmetic evaluation
  - When the programmer explicitly requests a conversion

- Not all conversions are possible
  - booleans cannot be converted to any other type

- Some conversions are not desirable
  - If a 64 bit value is put into a 32 bit variable, there will be a loss of information

Widening Conversions

- A widening conversion occurs when a value stored in a smaller space is converted to a type of a larger space.
  - There will never be a loss of information
- Widening conversions occur automatically when needed.

Original Type	Automatically converted to:
byte (8 bits)	char, short, int, long, float or double
char (16 bits)	int, long, float, or double
short (16 bits)	int, long, float, or double
int (32 bits)	long, float, double
float (32 bits)	double

Automatic Conversions

- Automatic conversions occur during arithmetic and assignment operations

```
byte x = 100;
int y = x;          // the value in x is promoted to int
                    // and assigned to y.
long z = y;         // the value in y is promoted to long
                    // and assigned to z.
float a = z;        // the value in z is promoted to float
                    // and assigned to a.
```

```
int x = 50;
long y = 100;
float z = 200.0;

double a = x + y + z
```

x promoted to long
expression evaluates to long

value of (x + y) promoted to float
expression evaluates to float.

a is double.
expression value promoted to double for assignment

Narrowing Conversions

- A narrowing conversion occurs when a value stored in a larger space is converted to a type of a smaller space.
  - Information may be lost
  - Never occurs automatically. Must be explicitly requested by the programmer using a cast.

Original Type	Narrowing conversions to:
char (16 bits)	byte or short
short (16 bits)	byte or char
int (32 bits)	byte, char, or short
long	byte, char, short, or int
float (32 bits)	byte, char, short, int, or long
double (32 bits)	byte, char, short, int, long, or float

Casting

- Casting is what a programmer does to explicitly convert a value from one type to another.

- The general syntax for a cast is:

```
(result_type) value;
```

- Examples

```
float price = 37.53;
int dollars = (int) price;          // fractional portion lost
                          // dollars = 37

char response = 'A';
byte temp =(byte) response;  // temp = 65 (ASCII value
                          // for 'a')
```

Increment and Decrement Operators

- Contains operators for increment and decrement (++ , --)
- For each, there is a prefix and postfix notation:
  - x++ (postfix increment : x is incremented by 1)
  - ++x (prefix increment : x is incremented by 1)
  - x-- (postfix decrement : x is decremented by 1)
  - --x (prefix decrement : x is decremented by 1)
- Used in isolation, postfix and prefix are essentially the same.
- However, used within expressions, the results are very different.

Postfix versus Prefix notation

- When prefix notation is used, the operation is performed first and the result is evaluated.
- When postfix notation is used, the result is evaluated first and then the operation is performed.

```
int x = 5;          // x initialized to 5
int y = x++;        // y assigned the value of x, x incremented
                // ie x = 6, y = 5

int z = ++x;        // x incremented, z assigned the value of x
                // ie x = 7, z = 7
```

- Often used when accessing array indices:

```
int[] grades = {96, 74, 88, 56};
int index = 0;

     int firstGrade = grades[x++];
```

Be aware of code readability issues

- As always, code readability is essential.
- Avoid using postfix and prefix notation in such a manner which obscures the intent of the code:

```
int x = 5;
int y = 26;
int z = 91;

int a = (++x * y++)/ --z + (x++ + ++y)*z++;
```

- What does the value of a represent?
- Remember, the increment and decrement operators are unary operators and have a higher precedence than all mathematical operators.

Blocks

- Java groups statements into a single unit called a block.
- Block boundaries are defined with curly braces {}
- Blocks help to define scope.
  - Generally speaking, variables defined within a block are only known within that block.
- Using blocks can make code much more readable.

```
if (x % 2 == 0)
{
     System.out.println("x is even");
}
else
{
     System.out.println("x is odd");
}
```

Review

- Describe statements in Java
- How are variables Declared? What are valid variable names?
- What are the fundamental data types?
- How are variables made constant in Java?
- What are the operators in Java? Describe the precedence list.
- What kinds of type conversions occur automatically?
- What is a narrowing conversion and how is it done?
- What is a block?