2

Java Fundamentals (Java SE 8)

Data, Variables, and Operators

Presented by :

Eng Marwa Ali Eissa

keywords

***Keywords** are words that :*

- Are reserved for use by Java.
- May not be used to name Java applications or objects ,such as classes ,methods or variables.
- Are case-sensitive and in lowercase

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

Identifiers

*An **identifier** is a name that:*

- Identifies a variable, class or method
- Identifiers must start with either an uppercase or lowercase letter, an underscore (_), or a dollar sign ($).
- Identifiers cannot contain punctuation, spaces, dashes, or any of the Java technology keywords.
- Most importantly identifiers are case sensitive.
- Identifiers cannot begin with a digit (0-9)
- White space is not permitted.

- Examples of legal identifiers: age, $salary, _value, __1_value
- Examples of illegal identifiers : 123abc, -salary , max value

Naming Convention in Java

- **packages** names are in lowercase .
  E.g :java.lang,java.util and com.ourglobalcompany.myproject.model

- **Classes**: Names should be in CamelCase is where each new word begins with a capital letter
  E.g. :CustomerAccount , Employee

- **Methods & Variables** : is the same as CamelCase except the first letter of the name is in lowercase
  E.g.: firstName ,orderNumber
  void calculateTax() , String getSalary()

- **Constants**: Names should be in uppercase.
  E.g.: DEFAULT_WIDTH , MAX_HEIGHT

Statements and Blocks

Statements are roughly equivalent to sentences in natural languages. A *statement* forms a complete unit of execution. The following types of expressions can be made into a statement by terminating the expression with a semicolon (;).

- Assignment expressions
- Any use of ++ or --
- Method invocations
- Object creation expressions

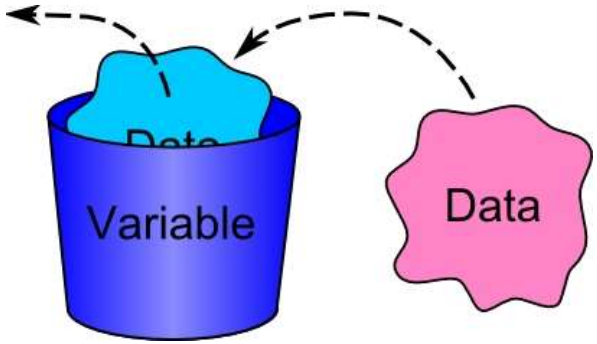
```
// Each of the following lines is a programming statement, which ends with a semi-colon (;)
int number1 = 10;
int number2=5, number3=99;
int product;
product = number1 * number2 * number3;
System.out.println("Hello");
```

Statements and Blocks

Multiple Java statements may be grouped using braces to form ***a block statement.*** Block statements are logical units of statements ,usually associated with a specific statement container such as a method or a loop .Java allows to place a block statement within another block

```
public class BlockDemo {
      public static void main(String[] args) {
            boolean condition = true;
            if (condition) { // begin block 1
                  System.out.println("Condition is true.");
            } // end block one
            else { // begin block 2
                  System.out.println("Condition is false.");
            } // end block 2
      }
}
```

Variables



Variables are named memory locations that are used for storing data items of a *value* of a particular data type or a reference to an object.

- Variable lives confined to the scope of their definition, which cab be :
  - At the local evlevel inside a method or block of code , they are called *local variables*
  - At the object instance lel when defined as a *Non-static* attribute , also called *instance variables*
  - At the class level when defined as a *static* attribute , also called *class variables*
  - Variables in method declarations—these are called *parameters.*

Variables

Variable Declaration

- To declare a variable, you use the syntax :
  datatype varableName;

- To Declare multiple variables using A single statement :
  datatype variable1,variable2 ,variable3 ,….. variableN;

Variable Naming Conventions

The variable naming conversions are the same as those of identifiers
Java developers must not use the dollar symbol in a variable. Also, variable names must not start with an underscore

Initializing Variables

You can assign values to variables in your program during variable declaration.

datatype variableName = initvalue;

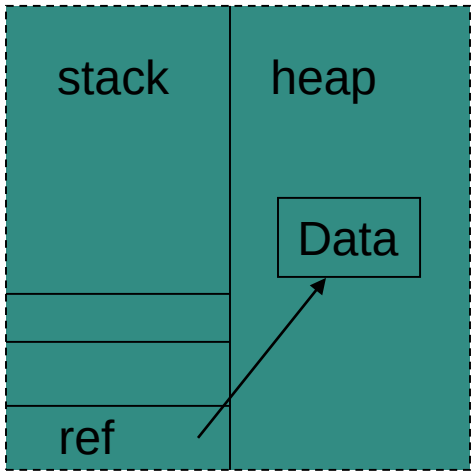
datatype variable1 = initvalue , variable2= initvalue2 , …. , variableN= initvalueN;

Uninitialized Variables

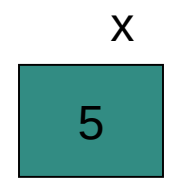
Attributes (instance or static) are always initialized to a default value
local variable must be initialized otherwise the Program does not compile

Data Types in JAVA

- ***Value Types*** (Built in data types – Primitive data types Like: int , float , ..)
- ***Reference Types*** (any other type Like: objects , Interface , array, Enum ..)



Memory



int x =5;

Primitive Data Types

- There are eight primitive data types supported by Java, and they enable you to define variables for storing data that fall into one of three categories:

1. **Numeric values**
   - ✓ integer (byte, short, int, and long)
   - ✓ floating-point (float and double)
2. **A single Unicode character** (char)
3. **Logical values** (boolean) that can be true or false

Integer Data Types

Data Type	Size	Range	Default Value
byte	1 byte (8 bits)	**-2^7** to **2^7-1** (-128 to +127 or 256 possible values)	0
short	2 bytes (16 bits)	**-2^{15}** to **$2^{15}-1$** (-32,768 to 32,767 or 65,535 possible values)	0
int	4 bytes (32 bits)	**-2^{31}** to **$2^{31}-1$** (-2,147,483,648 to 2,147,483,647 or 4,294,967,296 possible Values)	0
long	8 bytes (64 bits)	**-2^{63}** to **$2^{63}-1$** (-9,223,372,036854,775,808 to 9,223,372,036854,775,807, or 18,446,744,073,709,551,616 possible values)	0L

Floating-Point Data Types

Data Type	Size	Default Value
float	32-bit	0.0f
double	64-bit	0.0d

- Float is mainly used to save memory in large arrays of floating point numbers.
- Float data type is never used for precise values such as currency.

- Double data type is generally used as the default data type for decimal values. generally the default choice.
- Double data type should never be used for precise values such as currency.

Character Type

- Char data type is used to store any character.
- Java uses Unicode to represent characters
- char data type is a single 16-bit Unicode character.
- Minimum value is '\u0000' (or 0).
- Maximum value is '\uffff' (or 65,535 inclusive).
- Default value : '\u0000'
- **Example** : char letterA ='A'

Boolean Type

- ✓ boolean data type represents one bit of information.
- ✓ There are only two possible values : true and false.
- ✓ This data type is used for simple flags that track true/false conditions.
- ✓ Used to hold the result of an expression that evaluates to either true or false.
- ✓ Default value : false
- ✓ The size of boolean is not defined in the Java specification, but requires at least one bit.
- ✓ **Example** : boolean one = true

Primitive Casting

byte → short →int →long →float →double

Implicit Casting (Automatic Type Conversions)

Explicit Casting

```
public class CastDemo {

    public static void main(String[] args) {
        int i = (int) 17.5f;
        System.out.println("The value after casting is: " + i);
        float f = i;
        System.out.println("The value after implicit casting is: " + f);
        long l = 1334564544;
        double d = l;
        System.out.println("The value after casting is: " + d);
    }
}
```

Explicit casting

Implicit casting

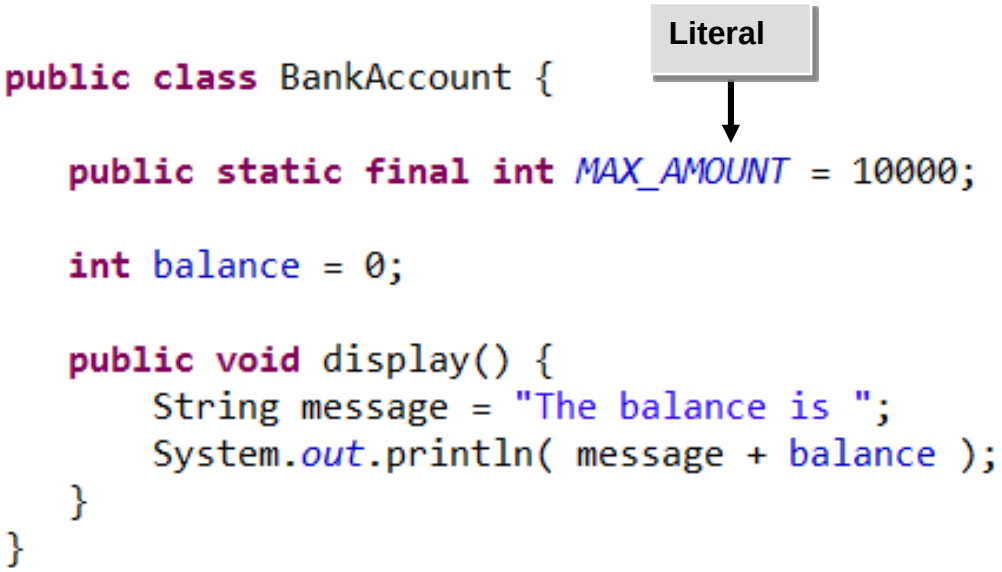
Console

```
<terminated> CastDemo [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (14-Sep-2011 10:13:54 PM)
The value after casting is: 17
The value after implicit casting is: 17.0
The value after casting is: 1.334564544E9
```

Data loss due to explicit down casting

Literals

A literal in java is any item that directly represents a particular value . Variables belonging to any primitive types can also be assigned values and treated as literals .
Each variable type has a corresponding literal type , such as integer, character, string and boolean



Types of Literals

There are four types of literals in java, which are based on the types of variables present

- Numeric Literal
  - ✓ int ➔ ex: 7
  - ✓ double ➔ ex: 12.87 , 12e22 , 19E-95
  - ✓ float ➔ ex: 12.87f , 123.988F
- Boolean Literal
  - ✓ true or false
- Character Literal
  - ✓ ‘a’ ,’A’,’#’,’3’ ➔ ASCII
- String Literal
  - ✓ “hi from Marwa” , “Welcome \n in New Horizons”

Escape Sequences for Character Literals

Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a formfeed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.
\d	Octal
\xd	Hexadecimal
\ud	Unicode Character

Declaring Constants

- A Constant is a variable type in java whose value does not change .

- Constants are defined using the final keyword followed by the variable declaration. By convention ,constant variable names are in uppercase .If the name is composed of more than one word, the words are separated by an underscore (_).

- Constants defined in this way cannot be reassigned, and it is a compile - time error if your program tries to do so.

EX: final double PI=3.141592653589793;
EX: final int FEET_PER_YARD = 3;

Expressions

An expression is a combination of operators (such as addition '+', subtraction '-', multiplication '*', division '/') and operands (variables or literals), that can be evaluated to yield a single value of a certain type

```
1 + 2 * 3 // evaluated to int 7
int sum, number; sum + number // evaluated to an int value
// Evaluated to a double value
double principal, interestRate; principal * (1 + interestRate)
```

Operators

- ✓ *Arithmetic Operators*
- ✓ *Assignment Operators*
- ✓ *Relational Operators*
- ✓ *Logical Operators*
- ✓ *Increment and Decrement Operators*
- ✓ *Bitwise Operators*
- ✓ *Operator Precedence*

Arithmetic Operators

Arithmetic operators are used in mathematical expressions

Operator	Meaning	Syntax
+	Addition	X + y
–	Subtraction	X - y
*	Multiplication	X * y
/	Division	X / y
%	Modulus The reminder from a division operation	X % y

String Concatenation

The + operator may used to concatenate the strings together

System.out.println("Account Balance is"+ balance);

Compound Arithmetic Assignment Operators

= Simple assignment operator, Assigns values from right side operands to left side operand

Operator	Expression	Meaning
+=	X += y	X=X + y
-=	X -= y	X=X – y
*=	X *= y	X=X * y
/=	X /= y	X=X / y
%=	X %= y	X=X % y

Relational Operators

Relational Operators are symbols user for determining relational comparisons between operands. all expressions created using relational operators will return a boolean value , depending on whether the comparison is true

Operator	Meaning	Syntax
==	equal to	X == y
!=	not equal to	X != y
>	greater than	X > y
<	less than	X < y
>=	greater than or equal to	X >= y
<=	less than or equal to	X <= y

Logical Operators

Logical Operators evaluate expressions that carry boolean values . The result returned by logical operators is also a boolean value.

Operator	Meaning
&	logical AND
&&	conditional AND
|	logical OR
||	Conditional OR
^	exclusive OR (XOR)
!	logical (NOT)

Logical Operators - The truth tables

AND (&&)	true	false
true	true	false
false	false	false

OR (||)	true	false
true	true	true
false	true	false

NOT (!)	true	false
Result	false	true

XOR (^)	true	false
true	false	true
false	true	false

Increment and Decrement Operators

Operator	Meaning
++	Increment operator; increase the value of a numeric variable or array element by 1
--	Decrement operator; reduce the value of a numeric variable or array element by 1

Prefix and Postfix

- **Prefix** :

Placing the operator before the operand causes increment or decrement to occur before the value of the operand is used to evaluate an expression
For example : int a=5;
int x =++a;
both x and a will have the value 6

- **Postfix** :

Placing the operator after the operand causes increment or decrement to occur after the value of the operand is used in an expression
For example : int a=5;
int x =a++;
x will have the value of 5 and a will have the value 6

Bitwise Operators

Operator	Description
Binary bitwise AND (&)	Returns a 1 if both bits are 1
Binary bitwise OR (|)	Returns a 1 if either bit is 1
Binary bitwise XOR (^)	Returns a 1 if both bits have different values
Unary bitwise complement (~)	Changes each bit in its operand to the opposite value
Binary left shift (<<)	Shifts bits on the left to the left by a distance denoted by the right operand .Fills in zeros
Binary right shift (>>)	Shifts bits on the left to the right by a distance denoted by the right operand .Fills in the highest bit on the left side.
Binary arithmetic and logical shift (>>>)	Shifts bits on the left to the right by a distance denoted by the right operand .Fills in zeros

Bitwise Operators

X=10
y=25

00001010
&
00011001

00001000

z=8

00001010
|
00011001

00011011

z=27

00001010
~

11110101

z=-11

00001010
<<
2

00101000

z=40

X=10

00001010
>>
2

00000010

z=2

X=-11

11110101
>>
2

11111101

z=-3

X=10

00001010
>>>
2

00000010

z=2

X=-1

0X ff ff ff ff ff
>>>
42

0X 00 00 00 ff

z=255

Operator Precedence

Operator precedence is the order in which operators are evaluated in an expression containing two or more operators

```
public class Kiosk_1A {

    public static void main(String[] args) {

        // Assigns value 20 to a
        int a = 3 + 5 + 14 - 7 + 5;

        // Assigns value 72 to b
        int b = 3 + 5 * 14 - 7 / 5;

        // Assigns value 111 to c
        int c = (3 + 5) * 14 - 7 / 5;

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }

}
```

Operators with same precedence are calculated from left to right

Parentheses alter the order of calculations

Operator Precedence

precedence	Operator
1	(..) [..] . (dot operator)
2	++ -- ! ~ instanceof
3	new (type) expression
4	* / %
5	+ -
6	<< >> >>>
7	< > <= >=
8	= !=
9	&
10	^
11	|
12	&&
13	||

Thank You !