Basic Elements of Java

Topics

- Data types
- Variable Naming Convention
- Reading input from console
- Conversions
- Keywords/Reserved words
- Common Errors
- Arithmetic Operators
- Shift operators
- Conditional Logical Operators
- Conditional statement
- For loop omitting options
- Labelled continue/break statements

Datatypes

Data types in java specify different sizes and values that can be stored in a variable.

There are two types of data types available in Java:

Primitive data types: The primitive data types are basic datatype. There are 8 Primitive data type, they are:

- Integer types: **byte, short, int, long**
- Floating point types: **float, double**
- Character data type: **char**
- Boolean data type: **Boolean**

Non-primitive data types: The non-primitive data types include Classes, Interfaces and Arrays.

Ranges of Primitive data types

Type	Default Size	Default Value	Min Range	Max Range
byte	1 Byte	0	**-2^{7}**	-2^{7}-1
short	2 Bytes	0	**-2^{15}**	-2^{15}-1
Int	4 Bytes	0	**-2^{31}**	-2^{31}-1
long	8 Bytes	0L	**-2^{63}**	-2^{63}-1
char	2 Bytes	‘\u0000’	0	-2^{16}-1
Float	4 Bytes	0.0f	The float data type is a single-precision 32-bit IEEE 754 floating point. **Its value range**	

			is unlimited. It is recommended to use float instead of double if you need to save memory in large arrays of floating point numbers. The float data should never be used for precise value like currency.
Double	8 Bytes	0.0d	The double data type is double precision 64-bit IEEE 754 floating point. **Its value is unlimited.** The double data type generally used for decimal values just like float. The double data type should never be used for precise values, such as currency.
boolean	1 Bit	False	The Boolean data type is used to store only two possible values: true or false

Unicode Character

- Unicode is a 16 bit character
- They are generally represented in hexadecimal format
- ‘\u’ in beginning of character is used to represent hexadecimal character
- The characters represented include all basic English letters, numbers, special characters and characters from other languages too.
- ‘A’ -> ‘\u0041’
- The Unicode Standard encodes characters in the range U+0000…U+10FFFF

Variables

- Variable names must begin with
  - A letter (A-Z, a-z, or any other language letters supported by UFT 16)
  - An underscore (_)
  - A dollar ($)
    - After which it can be a sequence of letters/digits.
- After which it can be sequence of letters/digits.
  - Digits: 0-9 or any Unicode that represents digit.
  - Length of the variable name is unlimited
- ~~Java reserved words should not be used for variable names~~

Variable Naming Convention

- Must begin with lower case
- Must always begin with a letter, not “$” or “_”
- Avoid abbreviations, use meaningful names.

- If a variable consists of two or more words, then the second and subsequent words should start with upper case.
- For example, firstName

Variable Declarations

- Local declarations
- Class declarations
    - Instance declaration
    - Static declaration

Reading input from console

To read input from the console, JDK provides a friendly class called java.util.Scanner.

```
Import java.util.Scanner;

Class A {

        Public static void main(String args[]) {

                Scanner scan = new Scanner(System.in);

                Int I = sc.nextInt();

                System.out.println(i);

        }

}
```

Note:

To read any primitive types, replace XXX in nextXXX() with the primitive type. To read a string use next().

Constants

- Final double PI = 3.1415;
- Constant Naming Convention
    - All the rules that apply for a variable, applies to constants as well except for the one given below.
    - Convention for constants is that they must all be in UPPER CASE. In cases where you have more than one word making up constant, words must be separated by underscore.

Literals

- Integer Literal
    - Integer literals are by default int (32 bits).
    - They can be decimal, octal , hexadecimal or long
    - 56, 010, 0XF, 4000000000l, 4000000000L
- Floating Point Literal
    - Floating-point literals are by default double (64 bits)
    - 45.6, 89.32D, 85.05d, 75.33f, 46.38F
- Character Literal
    - They can be characters inside single quote, Unicode char or octal character or escape sequence.
    - ‘a’, ‘\u0041’, ‘\101’, ‘\n’
- Boolean Literal: true, false
- String Literal: “Hello World”, “\u0041\u0069”

Escape Sequence

Escape Sequence	**Character**
\n	newline
\t	tab
\b	Backspace
\f	Form feed
\r	return
\”	**“**
\’	**‘**
\\	\

Conversions

1. Widening conversions (achieved by automatic or implicit conversion)
2. Norrowing Conversions (achieved by casting or explicit conversion) (Overall magnitude not lost)
3. All primitives are convertible to each other except for **Boolean**

Automatic or implicit conversion

- The conversion in the direction indicated happens automatically.
    - byte -> short -> int -> long -> float -> double
- When an arithmetic operation happens between different numeric types, the result of the operation is always of the higher numeric data type.
- When the integer literals are within the range of the specified integer dat types, the conversion happens automatically. In case the literal is beyond the range, the compiler flags an error.

- When arithmetic operation happens between floating points, the result is double.

Loss of information

There could be loss of some information during conversion of the following:

a) int -> float
b) long->float
c) long-> double

Example

```
Int i = 76543210

 Float f = I -> 7.6543208E7
```

Assignment conversions-integers

```
int I = 10;

          int b = 10

          int k = I + b; // will work


 byte b1 = 20; // will work

 short s = 10; // will work

 But byte b = 789; // will be an error

  But final int b = 10;


   Byte b1 = 20, b2 = 30;

         int i = b1 + b2; // will work

   But byte b = b1 + b2; // will be an error
```

In the case of short

```
          byte b = b + 1; // will be an error

 But byte b += 1; // will work
```

Assignment conversions – double

```
Int f = 32;
float t = f; // will work
Int k = t; // won’t work

float f = 32.3; // Compilation error
```

Char conversions

1. char c = ‘A’;
   int i = c; // ASCII
2. char c = 65; // will work
3. char c = -65; // will result in error
4. int number = 65;
   a. char c = number; // will result in error
5. char c = ‘A’;
   a. short s = c; // error

Arithmetic Operators

Unary: + - ++ --

Examples: -5, +5

```
char ch = ‘X’;
ch++;  // (ch = ‘Y’)
```

Binary: + - * / %

Example: int I = j + 5;

Relational Operators

<, >, <=, >=, ==, !=

Returns true or false

Example:

```
Int I = 10;
Int j = 20;
System.out.println(I > j);   // output is false
```

Integer Bitwise Operators

Operand 1	**Operand 2**	&	|	^	~ Operand 1
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

They convert the integral data types into their binary form and then perform operations according to the table.

Example: Integer Bitwise Operators

```
Public class AndOrNotEx {
   Public static void main(String[] args) {
       Byte x = 1;
       Byte y = 3;
       System.out.println(x & y); // prints 1
       System.out.println(x | y); // prints 3
       System.out.println(x ^ y); // prints 2
       System.out.println(~x); // prints 2
   }
 }
```

Logical Operators

& | ^

Operand 1	Operand 2	&	|	^
True	True	True	True	False
True	False	False	True	True
False	True	False	True	True
False	False	False	False	False

Conditional Logical Operators

Operand 1	**Operand 2**	**&&**	**||**	**!Operand 1**
True	True	true	true	false
true	False	False	True	False
False	True	False	True	true
False	False	False	False	true

Logical operators are binary o>>perators that require Boolean values as operands.

&& and || are also called short circuit operators because they are optimized.

Ternary operator

Syntax of ?:

<variable>=(boolena expression) ? <value to assign if true>:<value to assign if false>

Example:

int i = 10;

double j = 10.1;

int k = (i > j) ? 10: 20;

System.out.println(k); // outputs 20

Shift Operators

- >> the shift right operator (Arithmetic shift)
- << the shift left operator
- >>> the unsigned shift right operator (Logical shift)

System.out.println(2<<1); // prints 4

System.out.println(-1<<2); // prints -1

System.out.println(-1>>>2); // prints 1073741823

Activity

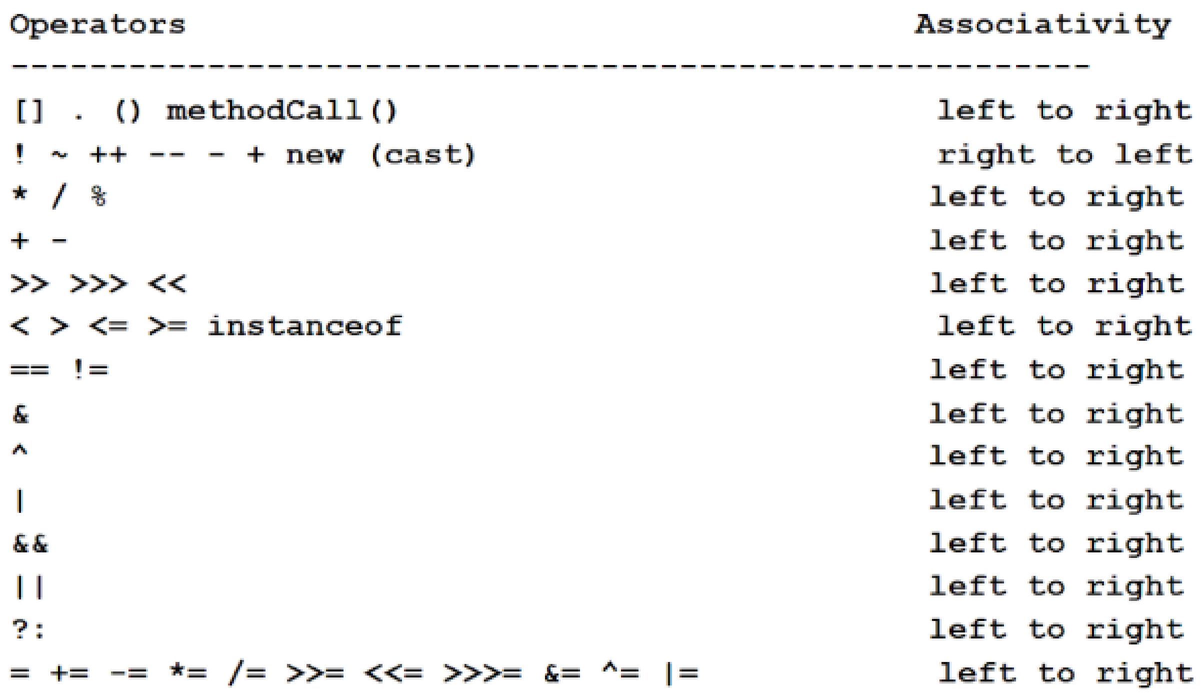
Find out what happens if you try to shift more than the number of bits in an int

System.out.println(2<<32);

System.out.println(2>>32);

System.out.println(2>>>32);

Precedence



Operators	Associativity
[] . () methodCall()	left to right
! ~ ++ -- - + new (cast)	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
?:	left to right
= += -= *= /= >>= <<= >>>= &= ^= |=	left to right

Explicit conversion or casting

Any conversion between primitives (excluding boolen) that is not possible implicitly can be done explicitly.

Conversions like

- double to long
- char to byte etc.,
- These are done through casting

Example:

Int k = 10;

char b = k; // error

Casting makes the error disappear:

Char b = (char) k;

Question:

Byte b = (byte) 128;

What will be the value when you print b? Compute and arrive the solution.

Flow control statements

Conditional Statements

- **If** statement
- **Switch** statement

Loops

- **For** statement
- Enhanced **for** statement
- **While** statement
- **Do-while** statement

Loops can have **break or continue.**

All of these statements except (enhanced for statement) are same as that in C

But note that for Java conditons must always evaluate to **boolean** value.

The **switch** expression should be in integer (not long) or char or String or Enum and case expression must evalulate to a constant/final value.

Common Errors

Can you figure out why the following statements are erroneous?

- If (1) System.out.println("OK");
- While(1.1) { // some statements

  }
- For (int I = 1; (i<5); i++) { System.out.println(i+);}

  I = 12;