

## < Elements of Java

- Statements and expressions
- Variables and data types
- Comments
- Literals
- Expressions and operators
- String arithmetic

## < Java

- Case sensitive
- Use plenty of whitespace
- Indent your code
- Indent and place your beginning and ending braces in same columns

## < Identifiers

- Are names given to a variable, class, or method
- Can start with a Unicode letter, underscore (\_) or dollar sign (\$)
  - Continue with letter, digit, underscore, or dollar sign
- Are case-sensitive and have no maximum length
- Examples:

- Identifier
- myName
- User\_name
- \_sys\_var1
- \$var

## < Naming an Identifier

- Identifier should be simple and descriptive
- An item ID could be ID, myID, itemID, itemNumber,
- one-letter identifier:
  - Quick, common usage -- index
  - Avoid, not recommended for unique items

## < Identifier Naming Rules

- First character can be A-Z, a-z, \_ or \$
- Subsequent characters can be any of the above and numeric
- Case-sensitive
- name, Name, NAME are all different, could be used, confusing
- Cannot be a keyword

## < Variable Identifier style

### Conventions

- Start with a lowercase letter, no separating character between words
- Start second and subsequent words with uppercase letter (Camel notation)

- Example: isFull (for a boolean value), price, itemID, myInt

## < Terminology

- In Java:
  - attribute = variable, characteristic
  - Method = operation, behavior
- Class – template/boilerplate for group of items
- Object – instance, example of a class, an actual item

## < Storing data in Variables Using

### Primitive Data Types

- primitive *data types* have predefined:
  - Storage sizes
  - Kinds of data they can store
  - Operations that they can use
- 8 primitive types are:
  - Integer types – *byte*, *short*, *int*, and *long*
  - Floating point types – *float* and *double* (default)
  - Textual type – *char*
  - Logical type – *boolean*

## < Integral Types

Type	Integer Length	Range	Examples of Allowed Literal
------	----------------	-------	-----------------------------

			Values
byte	1 byte	-27-27-1 (-128 to 127, or 256 possible values)	24 -107
Short	2 bytes	-215 to 215-1 (-32,768 to 32,767)	2 -44444
Int	4 bytes	-231 to 231-1 (-2,147,483,648 to 2,147,483,647)	2 323,123,456
Long	8 bytes	-263 to 263-1 (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)	5 - 3,456,789,123,456L 171L

## < Integral – byte, short, int, and long

- Numbers without decimals
- three forms – Decimal, octal, or hexadecimal
  - 2 The decimal value is 2
  - 066 The leading 0 indicates an octal value.
  - 0xABCD The leading 0x indicates a hexadecimal value
- Has a default int
- Defines long by using the letter L or l (not recommended – misread as ‘one’)

## < Represent numbers

- How to show negative numbers?
- They have the high order bit on (set to 1) to signify a negative sign

- 5 in binary -- 0000 0101
- How about -5? -- 1000 0101?
- If you add 5 and -5, would you get zero?

### < One's complement – procedure

Show the number in binary	Decimal 5	0000 0101
Convert all 1's to zeroes, all 0's to ones	One's complement of decimal 5	1111 1010

### < Two's complement – procedure

To get negative number, -5	Show binary positive number	0000 0101
Convert 1's to 0's and 0's to 1's		1111 1010
Add binary 1		0000 0001
Result is	-5 in two's complement	1111 1011

**< Application (+5 plus -5)**

Show number in binary	Decimal 5	0000 0101
Show negative number	Decimal -5	1111 1011
Add the numbers		0000 0000
	Result zero	

**< Application (20 + -5)**

Show number in binary	Decimal 20	0001 0100
Show negative number	Decimal -5	1111 1011
Add the numbers		0000 1111
	Result 15	

### < Application (-20 + 5)

Convert to -20	first, 20, convert 1's, 0's	0001 0100 1110 1011
Show negative number	Add 1, to get Decimal -20	1110 1100
Positive number	Decimal 5	0000 0101
Add -20 + 5	Result -15	1111 0001
To see if that is -15	Convert, add 1 – giving +15	0000 1110 0000 1111

### < Application (-15?)

What is this number		1111 0001
Change 1's to 0's, 0's to 1's		0000 1110
Add 1		0000 0001

Add the numbers	Result was -15	0000 1111

## < Floating Point – float and double

- Default is double
- Floating-point literal includes either a decimal point or one of the following:
  - E or e(add exponential value)
  - F or f (float)
  - D or d (double)

3.14 A simple floating-point value (a double)

6.02E23 A large floating-point value

2.123F A simple float size value

123.4E+306D A large double value with redundant D

## < Floating Point Types

Type	Length	Range	Allowed Literal Values
float	4 bytes	-3.4E38 to 3.4E38; 6 digits	57F -12,345.67F
double	8 bytes	-1.7E308 to 1.7E308; 14 digits	-123,456.78 12.34E+56

## < Decimal values

- Sign of number
- Mantissa – number normalized
  - $1.123 \times 10^3$  – normalized
  - $0.01123 \times 10^5$  is not normalized
- Characteristic – exponent
  - Number is positive with a bias subtracted from it
  - Exponent is 130, bias 126 is subtracted from it giving result of *4 for true exponent*
  - Exponent field is 120, bias 126 is subtracted giving a result of *-6 as true exponent*

## < Representation of decimal values

	Sign	Exponent	mantissa
Float	s	8 bits Bias - 126	23 bits
Double	s	11 bits Bias - 1023	52 bits

## < Textual – char

- Represents a 16-bit Unicode character
- Must have its literal enclosed in single quotes ( ' ' )
- Use the following notations:
  - 'a' The letter a
  - '\t' A tab
  - '\u????' A specific Unicode character, ?????, is replaced with exactly four hexadecimal digits (for example, '\u03A6' is the Greek letter phi [ ]) )

## < Textual – String

- Is not a primitive data type; it is a class (uppercase 'S' in String)
- Is enclosed in double quotes(“ ”)
  - “The quick brown fox jumps over the lazy dog.”

- Can be used as follows:  
String greeting = "Good Morning !! \n";  
String errorMessage = "Record Not Found !";

## < Logical – boolean

- The boolean data type has two literals, true (1) and false(0).
- For example, the statement:  
boolean truth = true;  
Declares the variable truth as boolean type and assigns it a value of true.

## < Primitive Variables

- Declare the variable (give its type and name).
- Assign a value to the variable

```
int var;  
var = 345;  
int var = 345;
```

## < Kinds of Values You Can Assign

- Literal values:  
float price = 1.23F;
- Other variables:  
float casePrice = 123.45F;  
float price = casePrice;
- The results of expressions:  
float numberOrdered = 234.5F;  
float casePrice = 23.45F;  
float price = (casePrice\*numberOrdered

## < Variables in a Program

- Use:  
System.out.println (ID);
- Re-use (assign another value later):  
ID = 123456;
- Combine:  
int ID = 123456;  
int saleID;  
saleID = ID;

## < Constants

- For values that cannot change once assigned:  
final double SALES\_TAX = 7.25;
- Cannot be changed except in original location.
- Use final keyword to make unchangeable
- Use all capital letters and underscores for identifier
- The compiler will give an error message if an attempt is made to change the constant's value

## < Comments

### • The three types of comment in a Java program:

```
// comment on one line
/* comment on one
Or more lines */
/** documentation comment */
```

## < Semicolons, Blocks, and White Space

- A statement is one or more lines of code terminated by a

```
semicolon(;):
totals = a + b + c
+ d + e + f;
```

- A block is a collection of statements bound by opening and

```
closing braces:
{
x = y + 1;
y = x + 1;
}
```

## < Semicolons, Blocks, and White Space

- block in a class definition:

```
Public class MyDate
{
Private int day;
Private int year;
}
```

- nest block statements.
- Any amount of white space is allowed

## < style

- When entering a beginning '{', place it on next line, enter a blank line, and place the ending '}' on that line in the same column position as its beginning brace



## < Arithmetic operators

*	Multiply
/	Divide
%	Modulus (remainder)
+	Add
-	Subtract

## < Assignment operators

- Left-hand-side assignment\_operator righthand-side
- LHS assignment\_operator RHS
- =
  - Whatever value RHS has is placed into LHS
- +=, -=, \*=, /=, %=
  - LHS is assigned value of ( LHS arithmetic\_operator RHS )

## < Incrementing, decrementing

- Increment – add one
- Decrement – subtract one
- Shown as j++ or ++j
- Postfix form – j++
  - Use variable, then add one

- Prefix form – ++j
  - Add one, use variable

## < Relational operators

- == equal
- != not equal
- < less than
- > greater than
- <= less than or equal to
- >= greater than or equal to

## < Logical Operators

- The boolean operators are:
  - ! - not
  - & - and
  - | - or
  - ^ - xor
- The short-circuit boolean operators are:
  - && - and
  - || - or

## < Not

	result
true	false
false	true

## &lt; And

	True	false
True	true	false
false	false	false

## &lt; Or

	True	false
True	true	true
false	true	false

## &lt; Xor

	True	false
True	false	true
false	true	false

## < example

- You can use these operators as follows:
  - MyDate d;
  - If ((d != null) && (d.day > 31))
  - {
  - // do something with d
  - }

## < Bitwise Operators

symbol	name	operation
&	Bitwise and	Only true if both bits true
	Bitwise or	True if any bits true
^	Bitwise xor	True if bits are different
~	Bitwise not	Change bit to other value

## < Using bitwise operators

purpose	use		Bits 7654 3210	result
Find the bits 3, 5	mask, &	A mask	0110 0101 0010 1000	0010 0000
Set bits 3, 5 to 1	mask, or	A mask	0110 0101 0010 1000	0110 1101
1's to 0's, 0's to 1's	Use ~	A	0110 0101	1001 1010

## < Shift Operators

Symbol	Name	operation
<<	Shift left	operation
>>	Shift right	Shift bits to left, zero filled on right
>>>	Unsigned shift right	Shift bits to right, extend sign

## < Shifting bits

<<	0011 1001 (x39, 57)	0111 0010 (x72, 114)
>>	0011 1001 (x39, 57)	0001 1100 (x1C, 28)
>>	1110 1100 (xEC, -20)	1111 0110 (xF6, -10)
>>>	1110 1100 (xEC)	0111 0110 (x76)

## < String Concatenation With +

- The + operator:

---

Sun, Sun Microsystems, the Sun Logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the united States and other countries. This Java Presentation was developed for the Sun Foundation by the East Side Union High School District in San Jose, CA, 2004.

- Performs String concatenation
- Produces a new String:  
String salutation = "Dr.";  
String name = "John" + " " + "Smith";  
String title = salutation + " " + name;
- One argument must be a String object.
- Non-strings are converted to String objects automatically

## < Operator Precedence

- Precedence rules:
  - Contents of parentheses
  - Multiplication and division, from left to right
  - Addition and subtraction
- Complex expression:  
C = 25 - 5 \* 4 / 2 - 10 + 4;
  - Result from left to right; 34
  - Result according to rules of precedence: 9
- Use parentheses to explicitly indicate structure

## < Casting

- If information is lost in an assignment, the programmer must confirm the assignment with a typecast.
- The assignment between long and int requires an explicit cast.

Long bigValue = 99L;

Int squashed = bigValue; // Wrong, needs a cast

```

Int squashed = (int) bigValue; // OK
Int squashed = 99L; //Wrong, needs a cast
Int squashed = (int) 99L; // OK, but...
Int squashed =99; //default integer literal

```

## < Promotion and Casting of Expressions

- Variables are automatically promoted to a longer form (such as int to long)
- Expression is assignment-compatible if the variable type is at least as large(the same number of bits) as the expression type.

```

Long bigval = 6; // 6 is an int type, OK
Int smallval = 99L; //99L is a long, illegal
Double z = 12.414F; // 12.414F if float, OK
Float z1 = 12.414; // 12.414 is double, illegal

```

## < Promotion and Typecasting

- Assignments and expressions can cause a mismatch between variable data types, and storage locations for results
- Examples of causes:
  - Multiplication can cause data to exceed data type size
  - Division can introduce decimal places

## < Mismatch of Data and Type

- Right-side data smaller than left:

```
byte num1 = 55;
```

```
byte num2 = 45;
long num3;
num3 = num1 * num2;
```

## < Mismatch of Data and Type

- Right-side data larger than left:

```
int num1 = 55;
int num2 = 45;
byte num3;
num3 = (num1 + num2);
```

## < Mismatch of Data and Type

- Right-side data and type larger than left:

```
int num1 = 55;
int num2 = 45;
byte num3;
num3 = num1 * num2;
```

## < Promotion

- Happens automatically when:
  - Assigning smaller type to large type
  - Assigning integer type to floating point type
- long big = 6; // legal
- int small = 99L; // illegal

## < Typecasting

- Lowers the range of a value (“chops it down”)

```
Variable_identifier = (target_type) value
int num1 = 53;
int num2 = 47;
byte num3;
num3 = (byte) (num1 + num2); // No data loss
int myInt;
long myLong = 99L;
```

```
myInt = (int) (myLong); // No data loss, only zeroes.
int myInt;
long myLong = 123987654321;
myInt = (int) (myLong); // Number is "chopped"
Typecasting floating point to integer removes
values to the right of the decimal point
```

## < Integer Data Types

- Adding integral types using + converts primitive data types to int (or higher)

```
Short a, b, c;
a = 1 ;
b = 2 ;
c = a + b ;
```

## < Floating point datatype

- Floating point values default to double if not specified float

```
float float1 = 12.3; // causes error
float float1 = 12.3F; //would work ok
float float1 = (float) 12.3; //would work ok
```