LESSON II.
Java syntax basics

Vu Thi Huong Giang
SoICT (School of Information and Communication Technology)
HUST (Hanoi University of Science and Technology)

Objectives

- Understand the Java syntax basics and their semantics for writing a program.

Structure of a Java program

Program
- Package
  - Class
    - Methods
      - block
        - » Statement
          - Expression
            - Token

Book
- Part
  - Chapter
    - section
      - Paragraph
        - » Sentence
          - Phrase
            - Word

Content

- White space
- Separators
- Comments
- Identifiers and keywords
- Data types and literals
- Comment line arguments

I. White spaces

- A whitespace is a space, a tab or a new line.
- It is used to separate the tokens in a source file and to improve readability
- Java don't require indentation rules
  - Example: following codes are equivalent

```
package firstpackage;
public class SayHello {
   public static void main(String[] args) {
      // Display Chao! on the screen
      System.out.println ("Chao!");
   }
}
```

```
package firstpackage; public class SayHello { public static void main(String[] args) { // Display Chao! on the screen System.out.println ("Chao!");}}
```

II. Separators

- Help define the structure of program

()	parenthesis	lists of parameters in method definitions and calls, precedence in expressions, type casts
{}	braces	block of code, local scope, class definitions, method definitions, automatically initialized arrays
[]	brackets	declaring array types, referring to array values brackets
;	semicolon	terminating statements, chain statements inside the "for" statement
,	comma	separating multiple identifiers in a variable declaration, chains statements in the test, expression of a for loop
.	period (dot)	separate package names from sub packages and classes, separating an object variable from its attribute or method
:	colon	using after loop labels

III. Comment

- Comments are used to
  - give overviews of code
  - provide additional information that is not readily available in the code itself
- Comments should contain only information that is relevant to reading and understanding the program.
- Two kinds of comments:
  - Implementation comment
  - Documentation comment

Comments

Implementation comment:

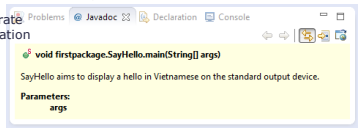
- `/* text */`
  - Ignore the text between `/*` and `*/`
- `// text`
  - Ignore all text from `//` to the end of the line

Documentation comment:

- `/** documentation */`
  - `javadoc` uses it to automatically generate software documentation

- Example

```
/** SayHello aims to display a hello in
*   Vietnamese on the standard output
*   device.
*/
class SayHello {
/* The program starts here.*/
public static void main(String[] args) {
     // Display Chao! on the screen
    System.out.println("Chao!");
    }
}
```



`javadoc` tool

- Used to automatically generate the API documentation or the implementation documentation for a set of Java source files.
  - Parse the declarations and documentation comments
  - Produce a corresponding set of HTML pages describing classes, interfaces, constructors, methods, and fields
- Frequent tags of `javadoc`:
  - `@param` : Explanation of argument name
  - `@return` : Explanation of the return value of the method of the object is described
  - `@exception class-name description` : Explanation of the exception `class-name` that may be thrown by the method
  - `@see reference` : Adds a "See Also" heading with a link or text entry that points to `reference`

IV. IDENTIFIERS AND KEYWORDS

1. Identifier
2. Naming rules
3. Keywords

1. Identifier

- Java identifiers are tokens that represent names of variables, constants, methods, objects, classes, files written in Java language
- Features:
  - Case-sensitive
    - identifier `Hello` <> identifier `hello` <> identifier `HELLO`
  - No limit of length
  - Not the same as Java keywords like `class`, `public`, `void`, etc.
  - Not be `true`, `false` or `null`
- Structure: Identifiers must begin with either a letter, an underscore "_" or a dollar sign "$".
  - Letters: lower or upper case
  - Subsequent characters: letters, digital numbers, etc.

2. Naming Rules

- Class: starts with upper case
  camel style (capitalize the first letter of each subsequent word)
  - `ThisIsAnExampleOfClassName, Circle, Account, ..`
- Method and variable: start with low case; camel style
  - `thisIsAnExampleOfMethodName, setColor, isFull, ..`
- Basic data type constant : all capitals
  - `MAX_LENGTH, TAX_VALUE ..`
- Object constant: all low cases
  - `Color.red, System.out ..`

Choose meaning and descriptive names

Consistent naming in the whole program

3. Java keywords

- Keywords are predefined identifiers reserved by Java for a specific purpose.
- 53 Java reserved words:

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

- **You cannot use reserved words as names for your variables, classes, methods, etc.**

13

V. DATA TYPES AND LITERALS

1. Data type categories
   a. Primitive data types
   b. Reference data types
   c. Literals
2. Variables
3. Type compatibility
4. Strong typing
5. Array

1. Data type categories

- Primitive data types : 8
- Reference data types: 2
- Literals: 6

8 primitive data types

	Type	Size	Min value	Max value	Default
Integer	byte	8 bits	-128	127	0
	short	16 bits	-32768	32767	0
	int	32 bits	-2,147,483,648	2,147,483,647	0
	long	64 bits	-9×10^{18}	9×10^{18}	0
Floating point number	float	32 bits	-3.4×10^{38} with 7 significant digits	3.4×10^{38} with 7 significant digits	0.0
	double	64 bits	-1.7×10^{308} with 15 significant digits	1.7×10^{308} with 15 significant digits	0.0
Unicode character	char	16 bits	\u0000 (0)	\uffff (65636)	\u0000
Boolean values	boolean	1 bits	false	true	false

Composite data types (reference types)

- Composite data type: constructed out of primitive data types and other composite date types.
  - Object: value of every type
  - Array: ordered list of values of the same data type
- Data of composite types don't have well-defined standard size, so they are handled by reference.
- The default value of this type is null (no data should be referenced).

Literals

- Literals are constant value of 6 following types:
  - Integer literals
    - Decimal: `100` or `100L` (long type)
    - Octal: `013` or `013L`
    - Hexadecimal: `0x52` or `0xFF1`
  - Floating-point literals (default is double type)
    - Decimal expression:
      - `0.23445, 0.655d` (double type) or `0.23445F` (float type)
    - Exponent (scientific) expression: `6.02E13`
  - Boolean literals: `true` or `false`
  - Character literals: `'a'`, `'\t'`, `'\u3042'`
  - String literals: `"Hello"`, `"Have a nice day!"`
  - Null literal: `null` (there no data should be referenced)

18

2. Variables

- A variable is a name that refers to a value located in memory
  → Used to store data
- Declaration:
  - variable's name
  - the type of information that it will hold (data type)

  →Multiple variables of the same type can be created in one declaration

```
int myFirstNumber;
int mySecondNumber;

or
int myFirstNumber, mySecondNumber;
```

Assignment and initialization

- Assignment: associate a value to a variable
  `variable = <expression>;`

```
int myFirstNumber, mySecondNumber;
myFirstNumber = 100;
mySecondNumber = myFirstNumber*2;
```

- Initialization: give an initial value in the declaration
  - Static: initialize with a literal
  - Dynamic: initialize with an expression available at the moment

```
int myThirdNumber = 10;
```

```
int myFourthNumber =
        myThirdNumber + 5;
```

20

Primitive variable

- A primitive variable is a variable of one of primitive data types
- A primitive variable stores the data in the actual memory location where it is

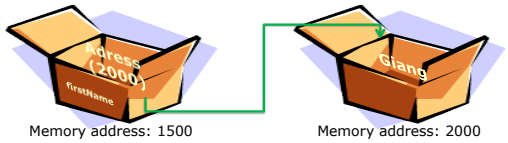
```
int myFirstNumber;
myFirstNumber = 100;
```

Memory address: 1001

Reference variable

- Reference: the memory address at which a composite data (object or array) is stored.
- Reference variable:
  - Store the address in the memory location
  - Point to another memory location where the actual data is
  - When a variable is referenced in a program, its current value is used
- Example:

```
String firstName = « Giang »;
```



Memory address: 1500 Memory address: 2000

3. Type compatibility

- Integer types and floating-point types are compatible with each other
- Numeric types are not compatible with **char** or **boolean**
- **char** and **boolean** are not compatible with each other
- What happens if a value of type **T1** is assigned to a variable of another type **T2** ?

```
T1 t1;
T2 t2 = t1;
```

  - types **T1** and **T2** are incompatible → not possible
  - types **T1** and **T2** are compatible:
    - **T1** and **T2** are the same
    - **T1** is larger than **T2**
    - **T2** is larger than **T1**

    → type casting

Type casting

Implicit type casting

- When operating on values of different data types, the lower one is promoted to the type of the higher one.
- Example:

```
int intVal = 123;
long longVal = 213000L;
longVal = intVal; //ok
intVal = longVal; //na
```

Explicit type casting

- The type is put in parentheses in front of the value being converted.
- Example: if we want a floating-point result of the division, we can cast the variable **total**:

```
int count, total;
float result;
result =
   (float) total / count;
```

24

4. Strong typing

- Every variable and expression has a type
  - All assignments are checked for type-compatibility
- Every type is strictly defined
  - No automatic conversion of non-compatible, conflicting types
- Java compiler checks the type of all expressions and parameters
  - Any typing errors must be corrected for a successful compilation

5. Array

- Array: ordered list of elements which hold values of the same data type
- Declaration:

```
data_type[] array_name;
data_type array_name[];
```

- Creation:

```
data_type[] array_name = new data_type[SIZE];
data_type array_name[] = new data_type[SIZE];
```

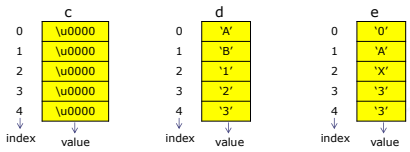
- Initialization:

```
data_type[] array_name = {value1, value2,.., valuen};
data_type array_name[] = {value1, value2,.., valuen};
```

Example

```
char c[] = new char[5];     //creation
//initialization all at once
char d[] = {'A','B','1','2','3'};

char e[] = new char [3];  // creation
//assignment for each element
e[0] = '0'; e[1] = 'A'; e[2] = 'X'; e[3] = '3'; e[4] = '3';
```



Array of reference type

- Be aware that the array of the reference type is "array of the variables that refer to the objects", and that it is not "array of the objects".

```
String array[] = new String[5];
array[0] = "Chao";
```


Multi-dimensional array

- Multidimensional arrays are implemented as arrays of arrays.
- Example

```
// integer array 512 x 128 elements
int[][] twoD = new int[512][128];
// character array 8 x 16 x 24
char[][][] threeD = new char[8][16][24];
// String array 4 rows x 2 columns
String[][] dogs = {{ "terry", "brown" },
 { "Kristin", "white" }
 { "toby", "gray"},
 { "fido", "black"}
};
```


VI. Comment line arguments

- When a java application is launched, the runtime system starts by calling the **`main()`** method of its main class
- The **`main()`** method then calls all methods found in his body to run this application
- Syntax:

```
public static void main(String args[]){
    // body
}
```

  - The runtime system passes arguments to the application through an array of Strings.
    - Arguments can be passed from the command line
    - Any number of arguments can be accepted as configuration information of the application.

Comment line arguments

- Launching a java application without configuration information

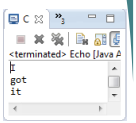
prompt> java class_name

- Launching a java application with configuration information

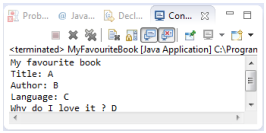
prompt> java class_name arg[0] arg[1] .. arg[n]

Example

```
public class Echo {
   public static void main (String[] args) {
        for (String s: args) { System.out.println(s); }
   }
}
>java Echo I got it
```




```
public class MyFavouriteBook {
    public static void main(String[] args) {
       System.out.println("My favourite book");
       System.out.println("Title: " + args[0]);
       System.out.println("Author: " + args[1]);
       System.out.println("Language: " + args[2]);
       System.out.println(
         "Why do I love it ?" + args[3]);
    }
}
> java MyFavouriteBook A B C D
```



Quiz: Javadoc

Given a Java source file named
virtual_parse_stack.java

This file can be found at JDK installation folder

C:\Program Files\Java\jdk1.7.0_03\src.zip
\com\sun\java_cup\internal\runtime

1. Run the following command for producing different documentations.

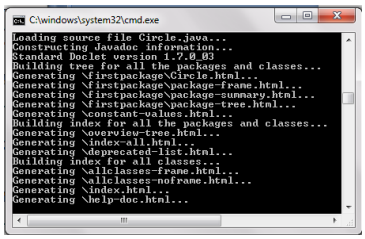
javadoc -private virtual_parse_stack.java
javadoc -author -version virtual_parse_stack.java

2. Explain the obtained results
3. Try to get the same results by using Eclipse.

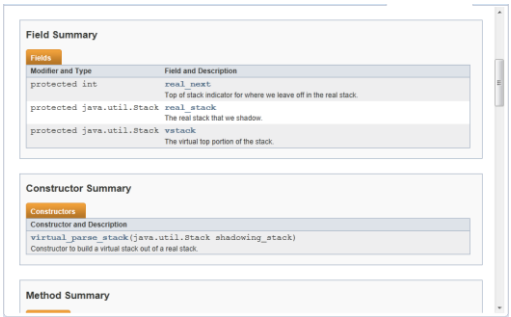
Quiz 1+2 - Solution

1. Explain the result

javadoc -private Circle.java
Show all class and member
javadoc -author -version Circle.java
Include the tags @author and @version



Quiz 3 - Solution



Quiz: array

4. Given a one dimension array of int type, write a program named OneDimensionArrayUsage that calculates its sum total (int) and its mean value (float)

```
int data[] =
    {1, 80,22, 134, 0, 33, 93,45,33, 12} ;
```

- The expected output is

```
sum: 453
mean: 45.3
```

5. Extend the program so that it displays the minimal and maximal value of the array.

Quiz 3 - Solution

```
public class OneDimensionArrayUsage {
    public static void main(String[] args) {
      int data[] = {1, 80,22, 134, 0, 33, 93,45,33, 12};
      int sum = 0; // Total sum
      for(int i=0; i<data.length; i++){
       // Calculation of total
       sum += data[i];
      }
      float mean = sum / 10.0f; // Mean
      System.out.println("Total sum :" + sum);
      System.out.println("Mean :" + mean);
      }
}
```


Quiz 4 - Solution

```
public class Arithmetic {
    public static void main(String[] args) {
    //...
    int min = 0; int max = 0;
    for(int i=0; i<data.length; i++){
    // Calculation of total
    sum += data[i];
    if (data[i] < min) min = data[i];
    if (data[i] > max) max = data[i];
    }
    //...
    System.out.println("Minimal value :" + min);
    System.out.println("Maximal value :" + max);
    }
}
```

Review

- White space: separate the tokens
- Separators: help define the structure of program
- Comment: improve readability + auto generate documentation
- Identifier: naming
- Data types: every variable, literal and expression has a type
- Comment line arguments: pass arguments to the application through an array of Strings.