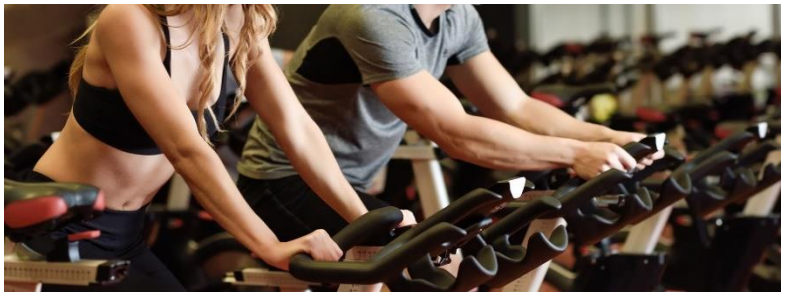
IECS273/274 [1111 3670/3671]
物件導向設計與實習

02#1 Java Fundamentals
Simple Expression Statement

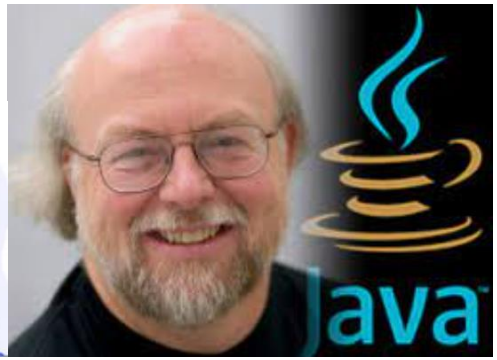


01

Introduction of Java

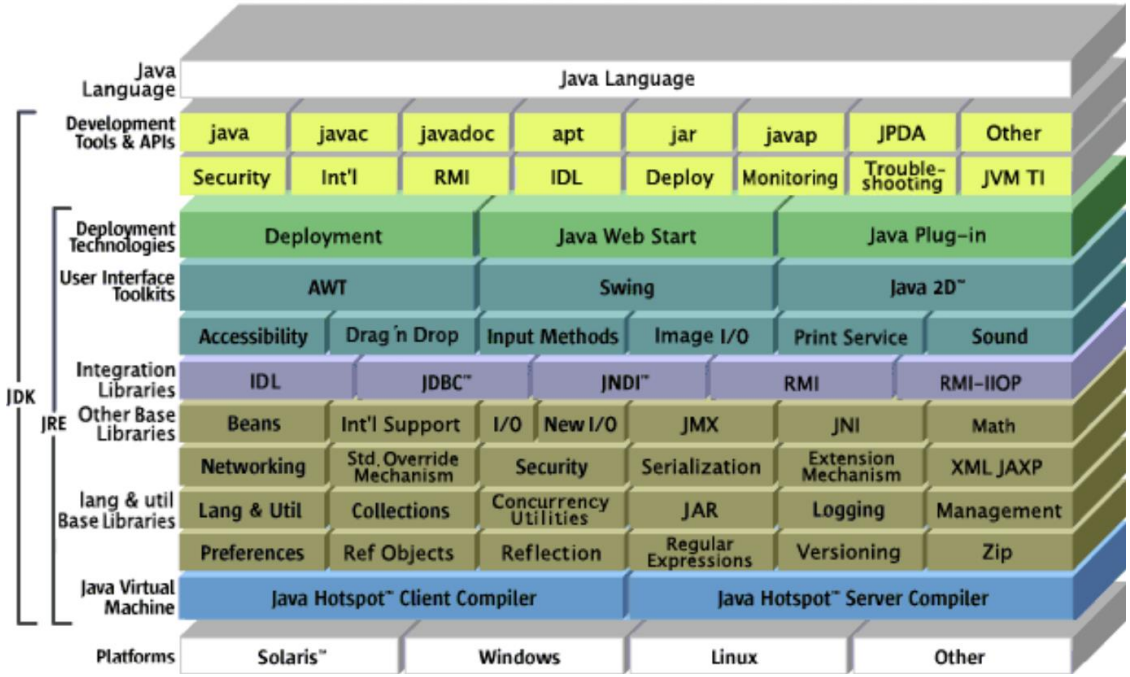
Java Application Programs

- Java is created by Sun Microsystems team led by *James Gosling*.
- There are two main types of Java programs: *applications* and *applets*.
  - A Java application program is a class with a method named **main**.
    - ✓ All Java application programs start with the main Method
    - ✓ When a Java application program is run, the run-time system automatically invokes the method named main
    - ✓ Application programs may use a windowing interface or console I/O.
  - A Java applet is a Java program that is meant to be run from a Web browser.
    - ✓ Can be run from a location on the Internet.
    - ✓ Can also be run with an applet viewer program for debugging.
    - ✓ Applets always use a windowing interface.



Java Frameworks

- Java is a robust language and when combined with a framework, Java can provide the best solutions for e-commerce, banking, cloud computing, finance, big data, stock market, IT, and more.
- A **Java framework** is specific to the Java programming language, used as a platform for developing Java programs. It may include predefined classes and functions used to process, input, and manage hardware devices, as well as interact with system software.



02

Java Foundation

Java Foundation

- The Java programming language is an object-oriented programming language that contains language constructs similar to the C language:
  - **Identifier:** A symbol that marks a name, such as the name of a package, class, method, parameter, constant, variable, attribute, etc.
    - ✓ *The name of a variable (or other item you might define in a program) is called an identifier.*
    - ✓ An identifier is a string consisting of English letters A--Z and a--z, digits 0--9, underscore (_), and dollar sign ($), but the first letter cannot be a digit, and cannot be the same as a keyword in the Java language。
    - ✓ *Java is a case-sensitive language; that is, it distinguishes between upper- and lowercase letters in the spelling of identifiers.*
  - **Primitive data types:** eight primitive types in the Java language

Type	Memory Size (bits)	Value Range	Type	Memory Size (bits)	Value Range
boolean	--	**true, false**	**int**	32	-2^{31} to $2^{31}-1$
byte	8	-128 to 127	**long**	64	-2^{63} to $2^{63}-1$
char	16	0000_X to $FFFF_X$	**float**	32	2^{-149} to $(2-2^{-23})\times 2^{127}$
short	16	-2^{15} to $2^{15}-1$	**double**	64	2^{-1074} to $(2-2^{-52})\times 2^{1023}$

Java Foundation

- **Keyword**: a word with special meaning in Java programming language that cannot be used as an identifier.

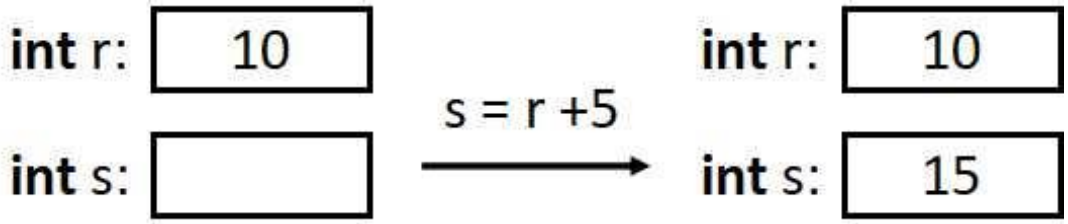
abstract	boolean	break	byte	case
catch	char	class	continue	default
do	double	else	extends	false
final	finally	float	for	if
implements	import	instanceof	int	interface
long	native	new	null	package
private	protected	public	return	short
static	super	switch	synchronized	this
throw	throws	transient	true	try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

Java Foundation

- **Constant** : a value that does not change throughout the execution of the program. *A **named constant** is an identifier that represents a permanent value, use all uppercase letters and designate word boundaries with an underscore character*
  - **Integer constant :**
    - ✓ **Binary**: 0 and 1 binary (base-2) numbers starting with 0b or 0B, e.g., 0b01101110, 0B10100011.
    - ✓ **Octal**: a base-2 number of digits from 0 to 7 with a leading 0, e.g., 06217.
    - ✓ **Decimal**: a base-10 number of digits from 0 to 9, e.g., -128, 0995
    - ✓ **Hexadecimal**: a hexadecimal (base-16) number from 0 to 9 and A to F (a to f) starting with 0x or 0X, e.g., 0x1c50e9, 0XFF00ABCD.
    - ✓ **Floating point constant**:
      - single precision: 3.14159265359F (π), -15e4f (-15×10^{4}), 12.5E-2F (12.5×10^{-2})
      - double precision: 1.414213562373095048 80D ($2^{1/2}$), 2.1111112e-10d (2.1111112×10^{-10})
  - **Character constant:** a fixed English letter, number, punctuation mark, special character, Chinese character
    - ✓ 'a', 'Z', '5', ':', '\n' (newline), '\u0000'(whitespace), '\u4e2d' (中)
  - **String constant: a sequence of consecutive characters**
    - ✓ “abcd”, “1234xyz4321”, “逢甲大學”
  - **Boolean constant: two values, true and false**
  - **Null constant: there is only one null value**

Java Foundation

- **Variable:** A variable represents a **memory space** and a **variable value.** *Variables are used to represent values that may be changed in the program. Every variable in a Java program must be declared before it is used.*
  - Variable declaration syntax: ⟨**variable**⟩ **::=** ⟨**class**⟩ ⟨**identifier**⟩



  - **Primitive data types**： **bool, byte, char, short, int, long, float, double**

  - Variable type conversion**: byte** a=5; **int** n=5249;
    - ✓ **Type coercion** (automatic, implicit): from a "small" type to a "large" type, n = a;
      - A compilation error may occur when changing from a "large" type to a "small" type.
    - ✓ **Type casting** (explicit): from a "large" type to a "small" type, a = (**byte**) n;
      - You can also switch from a "small" type to a "large" type.

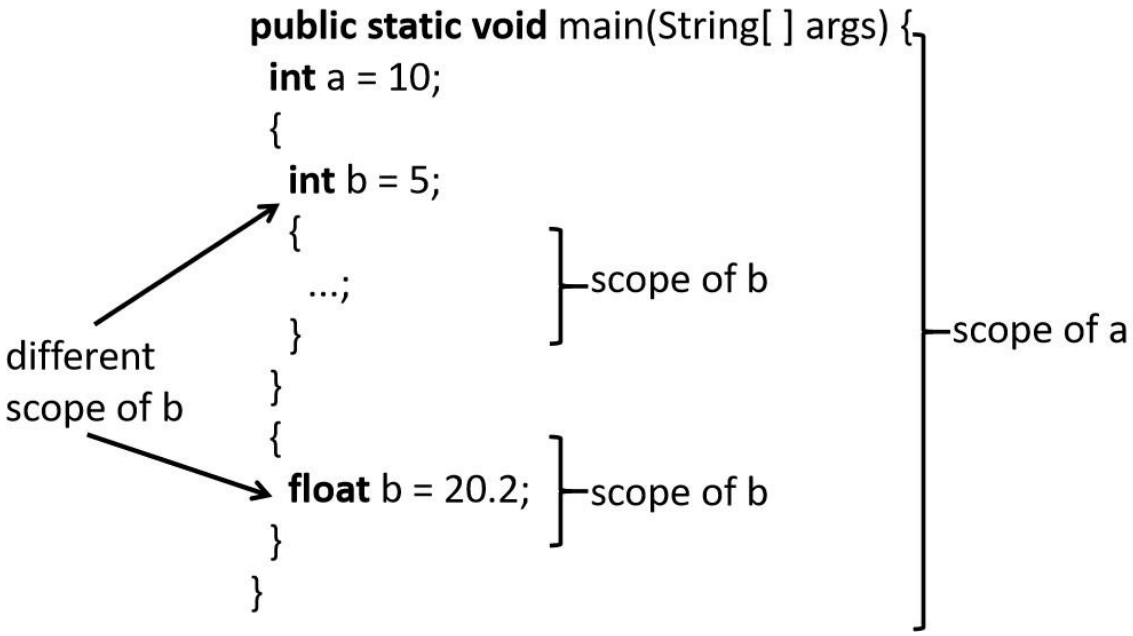
Java Foundation

- **Variable scope**: A variable has effect only in the block statement in which the variable is declared.
  - ✓ **Block statement**: a declarative statement in a pair of curly braces, Syntax:

⟨block statement⟩ ::= { ⟨statement sequence⟩ }

⟨statement sequence⟩ ::= ⟨statement⟩; | ⟨statement⟩; ⟨statement sequence⟩

- ✓ Block statements can be multi-level nested structures.



Java Foundation

Operations

- **arithmetic operations**: operands are of integer or floating point number type.
  - ✓ **uniary operator**: + (positive sign)，- (negative sign)
  - ✓ **binary operator**: + (addition)，- (subtraction)，* (multiplication)，/ (division)，% (modulo/ remainder)
  - ✓ **prefix/postfix operator**: ++ (add1)，-- (sub1)
- **logical operation:** operands are of boolean type
  - ✓ **uniary operator**: ! (not, negation)
  - ✓ **binary operator**: & (and, cojunction), | (or, disjunction), ^ (exclusive or), && (short-circuit and)，|| (short-circuit or)

Java Foundation

- **bit operation:** operands are of integer type
  - ✓ uniary operator: ~ (complement)
  - ✓ biary operator: & (bitwise and, conjuction)，| (bitwise or, disjunction)，^ (bitwise exclusive or), << (left shit)，>> (right shit)，>>> (unsigned right shift)

BitwiseOperation.java

```
public class test {
	public static void main(String arg[]){
		int a=20, b = -45; // two integer data
		System.out.print("a     = "); // integer a
		printBinary(a);
		System.out.print("b     = "); // integer b
		printBinary(b);
		System.out.print("~a    = "); // negation of a
		printBinary(~a);
		// unsigned right shift three bit of b
		System.out.print("b>>>3  = ");
		printBinary(b>>>3);
	}

	private static void printBinary(int a) {
		for (int i=0; i<32; i++) {
			System.out.print((a>>(31-i))&1);
			if (i>0 & (i+1)%4==0) System.out.print(" ");
		}
		System.out.println();
	}
}
```

BitwiseOperation.class

```
a      = 0000 0000 0000 0000 0000 0000 0001 0100
b      = 1111 1111 1111 1111 1111 1111 1101 0011
~a     = 1111 1111 1111 1111 1111 1111 1110 1011
b>>>3  = 0001 1111 1111 1111 1111 1111 1111 1010
```

Java Foundation

- **Short circuit evaluation**
  - ✓ In the case of AND(&&), the expression is evaluated until we get one false result because the result will always be false, independent of the further conditions.
  - ✓ In the case of OR(||), the expression is evaluated until we get one true result because the result will always be true, independent of the further conditions.
  - ✓ Ex:Sequential searching, find key in an integer array a[n].

```
1
2  public class test {
3          public static void main(String[] args) {
4              int key = 110; // The value of key to be searched for.
5              int[] a = new int[10]; // Data array.
6              int i; // Loop variable.
7
8              for (i=0; i<10; i++) a[i] = 100 + i; // Initial array values.
9
10             // Searching operation.
11             i = 0;
12             while (i<10 & a[i]!=key) i++; // without short-circuit evaluation
13
14             if (i<10) System.out.println("Search succeeds!");
15             else System.out.println("Search fails!");
16     }
17 }
18
```

```
1
2  public class test {
3          public static void main(String[] args) {
4              int key = 110; // The value of key to be searched for.
5              int[] a = new int[10]; // Data array.
6              int i; // Loop variable.
7
8              for (i=0; i<10; i++) a[i] = 100 + i; // Initial array values.
9
10             // Searching operation.
11             i = 0;
12             while (i<10 && a[i]!=key) i++; // with short-circuit evaluation
13
14             if (i<10) System.out.println("Search succeeds!");
15             else System.out.println("Search fails!");
16     }
17 }
18
```

Short Circuit AND test.java

```
import java.io.*;
public class test {
// Java code to demonstrate the short circuiting using &&
    public static void main(String arg[])
    {
          // Since first operand is false and operator is &&,
        // Evaluation stops and false is returned.
        if (false && true && true)
            System.out.println("This output will not be printed");
        else
            System.out.println("This output got printed actually, "
                               + " due to short circuit");

        // Whole expression will be evaluated, as no false is encountered
        // before last condition Therefore no Short circuit
        if (true && true && true)
            System.out.println("This output gets print"
                               + " as there will be no Short circuit");
        else

            System.out.println("This output will not be printed");
    }
}
```

Short Circuit OR test.java

```
import java.io.*;
public class test {
// Java code to demonstrate the short circuiting using ||
    public static void main(String arg[])
    {
       // Since first operand is true and operator is ||,
        // Evaluation stops and true is returned.
        if (true || false || false)
            System.out.println("This output got printed actually, "
                               + " due to short circuit");
        else
            System.out.println("This output will not be printed");

        // Whole expression will be evaluated,
        // as no true is encountered before last condition
        // Therefore no Short circuit
        if (false || false || true)
            System.out.println("This output gets print"
                               + " as there will be no Short circuit");
        else
            System.out.println("This output will not be printed");
  }
}
```

Short Circuit AND test.java

```
This output got printed actually,  due to short circuit
This output gets print as there will be no Short circuit.
```

Short Circuit OR test.java

```
This output got printed actually,  due to short circuit
This output gets print as there will be no Short circuit
```

Java Foundation

- **Assignment operation**: Assign the value of the on the right of the assignment operator to the variable on the left
  - ✓ Basic assignment operator: =

    *x=100; a=b; m=2*n+p/3;*
  - ✓ Extended assignment operator: +=, -=, *=, /=, %=, &= |=, &&=, ||=, <<=, >>=, >>>=

    *x += 10; a >>= b; m *= n + p * q;*
- **Comparison operation**: The operands are of various basic types, and the result of the operation is a boolean type
  - ✓ biary operator: == (qual to), != (not equal to), < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to)

Java Foundation

Operation Precedence

Precedence	Operators
1	. [] ()
2	++ -- + - ! ~
3	* / %
4	+ -
5	<< >> >>>
6	< > <= >=
7	== !=
8	&
9	^
10	|
11	&&
12	||
13	?:
14	= *= /= %= += -= != …

Association Rules

- ✓ binary operators of equal precedence are grouped left to right.
- ✓ unary operators of equal precedence are grouped right to left
- ✓ A string of assignment operators is grouped right to left

```
a = b + c + d;
// a= ( b + c ) + d

+-+number;
//+(-(+number))

x = y = z;
// x = (y = z);
```

Java Foundation

Character Sets

- ✓ **ASCII**(American Standard Code for Information Interchange): A character set used by many programming languages that contains all the characters normally used on an English-language keyboard, plus a few special characters.

0	00	NUL	26	1A	SUB	52	34	4	78	4E	N	104	68	h
1	01	SOH	27	1B	ESC	53	35	5	79	4F	O	105	69	i
2	02	STX	28	1C	FS	54	36	6	80	50	P	106	6A	j
3	03	ETX	29	1D	GS	55	37	7	81	51	Q	107	6B	k
4	04	EOT	30	1E	RS	56	38	8	82	52	R	108	6C	l
5	05	ENQ	31	1F	US	57	39	9	83	53	S	109	6D	m
6	06	ACK	32	20	space	58	3A	:	84	54	T	110	6E	n
7	07	BEL	33	21	!	59	3B	;	85	55	U	111	6F	o
8	08	BS	34	22	"	60	3C	<	86	56	V	112	70	p
9	09	HT	35	23	#	61	3D	=	87	57	W	113	71	q
10	0A	LF	36	24	$	62	3E	>	88	58	X	114	72	r
11	0B	VT	37	25	%	63	3F	?	89	59	Y	115	73	s
12	0C	FF	38	26	&	64	40	@	90	5A	Z	116	74	t
13	0D	CR	39	27	'	65	41	A	91	5B	[	117	75	u
14	0E	SO	40	28	(	66	42	B	92	5C	\	118	76	v
15	0F	SI	41	29	)	67	43	C	93	5D	]	119	77	w
16	10	DLE	42	2A	*	68	44	D	94	5E	^	120	78	x
17	11	DC1	43	2B	+	69	45	E	95	5F	_	121	79	y
18	12	DC2	44	2C	,	70	46	F	96	60	`	122	7A	z
19	13	DC3	45	2D	-	71	47	G	97	61	a	123	7B	{
20	14	DC4	46	2E	.	72	48	H	98	62	b	124	7C	|
21	15	NAK	47	2F	/	73	49	I	99	63	c	125	7D	}
22	16	SYN	48	30	0	74	4A	J	100	64	d	126	7E	~
23	17	ETB	49	31	1	75	4B	K	101	65	e	127	7F	DEL
24	18	CAN	50	32	2	76	4C	L	102	66	f			
25	19	EM	51	33	3	77	4D	M	103	67	g			

- ✓ **Unicode**: A character set used by the Java language that includes all the ASCII characters plus many of the characters used in languages with a different alphabet from English.

Java Foundation

- **Comment**
  - ✓ A line comment begins with the symbols //, and causes the compiler to ignore the remainder of the line
  - ✓ A block comment begins with the symbol pair /* and ends with the symbol pair */
- **Escape**
  - ✓ A backslash (\) immediately preceding a character denotes an escape sequence or an escape character.
  - ✓ The character following the backslash does not have its usual meaning.

```
// line comment sample

/*
   block comment sample
   a =  b + c + d;
*/

\r\n
```

03 Console I/O

Screen Output

- **Packages** are Java libraries of classes.
- Import statements make classes from a package available to your program.
- **System.out** is an object that is part of the Java language, and **println** and **print** are methods invoked by that object.
  - with **println**, the next output goes on a new line
  - with **print**, the next output goes on the same line
  - with **printf**, the output goes with a specific format
  - **Syntax**

    System.out.println(*Item_1 + Item_2 + ... + Last_Item*);
    System.out.print(*Item_1 + Item_2 + ... + Last_Item*);
  - **Examples**
    - ✓ *System.out.println("Welcome to Java World.");*
    - ✓ *System.out.println("The Elapsed time = " + time + " seconds");*
    - ✓ *System.out.print("God ");*
      System.out.print(" helps ");
      System.out.println(" those who ");
      System.out.println(" help themselves .");
    - ✓ *System.out.print("The cost is %d dollars", n);*

Screen Output

- **Syntax**

  System.out.printf(“print_*format*”, *Item_1,Item_2, …, Last_Item*);

- **Examples**
  - ✓ *double cost = 14.6;*
    System.out.printf(“The total cost is $%6.2f", cost);

CONVERSION CHARACTER	TYPE OF OUTPUT	FORMAT EXAMPLES
d	Decimal integer	%d %6d %4d
f	Fixed-point floating point	%f %4.1f %6.3f
e	E-notation floating point	%e %8.2e %7.4e
g	General floating point	%g %8.4g
s	String	%s %16s
c	Character	%c %10c

OutputTest.java

```
public class PrintFormatTest  {
        public static void main(String[] args) {
                double testNumber = 28.135;
                System.out.printf("Left %7.2f Right", testNumber);
                System.out.println();
                System.out.printf("Left %-7.2f Right", testNumber);
                System.out.println();
                System.out.println();

                int number = 3;
                System.out.printf("There are %2d apples.", number);
                System.out.println();
        }
}
```

OutputTest.java

```
Left   28.14 Right
Left 28.14   Right

There are  3 apples.
```

Console Input

- Java includes a class for doing simple keyboard input named the **Scanner** class.
  - In order to use the Scanner class, a program must

    import java.util.Scanner
  - This line specifies the Scanner class is in the java.util package; a package is simply a library of classes. The import statement makes the Scanner class available to your program.
  - **Syntax**

    Scanner *Object_Name* = new Scanner(System.in);
    ***Int_Variable* = *Object_Name*.nextInt()**
    ***Double_Variable* = *Object_Name*.nextDouble();**
    ***String_Variable* = *Object_Name*.next();**
    ***String_Variable* = *Object_Name*.nextLine();**
  - **Examples**
    - ✓ *Scanner keyboard = new Scanner(System.in);*
    - ✓ *int num;*

      num = keyboard.nextInt();
    - ✓ *String answer;*

      answer = keyboard.nextLine();

InputTest.java

```
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAverage {
        public static void main(String[] args) {

        // Create a Scanner object
        Scanner keyboard = new Scanner(System.in);

        // Prompt the user to enter three numbers
        System.out.print("Enter two numbers: ");
        double number1 = keyboard.nextDouble();
        double number2 = keyboard.nextDouble();

        // Compute average
        double average = (number1 + number2) / 2;

        // Display results
        System.out.println("The average of " + number1 + “and" + number2
                + " is " + average);
   }
}
```

InputTest.class

```
Enter two numbers: 10 20
The average of 10.0 and 20.0 is 15.0

Enter two numbers: 16.4
12.4
The average of 16.4 and 12.4 is 14.4
```

Text File Input

- The **Scanner** class can also be used to read data from a text file. To do this, we must create a Scanner object and link it to the file on the disk.
- To read from a text file, we need to import the classes **FileInputStream** and **FileNotFoundException** in addition to the Scanner class:

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
```

  - The FileInputStream class handles the connection between a Java program and a file on the disk.
  - The FileNotFoundException class is used if a program attempts to open a file that doesn't exist.

RefAJ_Display 2.12.java

```
import java.util.Scanner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

public class TextFileDemo
{
    public static void main(String[] args)
   {
                Scanner fileIn = null; // Initializes fileIn to empty
                try
                {
                // Attempt to open the file
                fileIn = new Scanner(
                new FileInputStream(“student.txt"));
                }
                catch (FileNotFoundException e)
                {
                // This block executed if the file is not found
                // and then the program exits
                System.out.println("File not found.");
                System.exit(0);
    }
```

RefAJ_Display 2.12.java

```
   // If the program gets here then
   // the file was opened successfully
   int studentID;
   String name;

   System.out.println("Text left to read? " +
   fileIn.hasNextLine());
   studentID = fileIn.nextInt();
   fileIn.nextLine(); // Read newline left from nextInt()
   name = fileIn.nextLine();

   System.out.println(“Student ID: " + studentID);
   System.out.println("Name: " + name);
   System.out.println("Text left to read? " +
   fileIn.hasNextLine());
   fileIn.close();
   }
}
```

RefAJ_Display 2.12.class

```
Text left to read? true
Student ID: 11102
Name: Michael Lee
Text left to read? False
```

student.txt

```
1102
Michael Lee
```

04 Java Statement - Simple Expression

Java Statement

- **Statement**: basic control structure of a programming language.

  There are four basic control structures. Syntax:

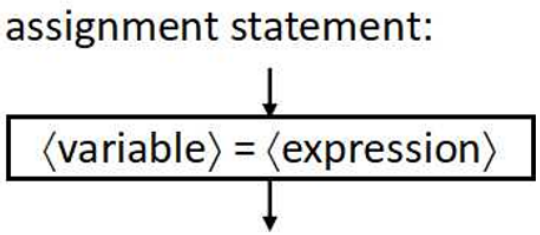
  ⟨statement⟩ ::= ⟨assignment statement⟩ | ⟨sequential statement⟩ |
  ⟨selection statement⟩ | ⟨iterative statement⟩

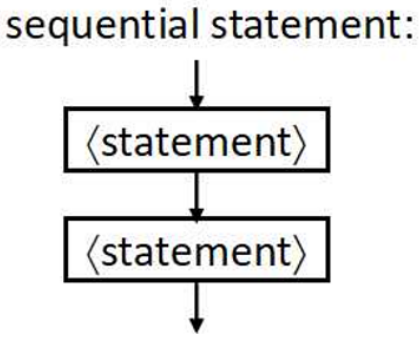
  - **Assignment statement**, syntax:

    ⟨assignment statement⟩ ::= ⟨variable⟩ ⟨assignment operator⟩ ⟨expression⟩

  - **Sequential statement**: Statements combined with **semicolons** (;). syntax:

    ⟨sequential statement⟩ ::= ⟨statement⟩; ⟨sequential statement⟩ | ⟨statement⟩;





Variable Declaration Statement

- Variables are used to represent values that may be changed in the program.
  - Variables are for representing data of a certain type.
  - To use a variable, you declare it by telling the compiler its name as well as what type of data it can store.
  - The **variable declaration** tells the compiler to allocate appropriate memory space for the variable based on its data type.
  - The syntax for declaring a variable is

    datatype variableName;

```
int score;              // Declare score to be an integer variable
double height;          // Declare height to be a double variable
double interest;        // Declare interest to be a double variable
```

Source: Y. D. Liang, Introduction to Java Programming and Data Structures (11th Edition), Pearson

Assignment Statement

- An assignment statement designates a value for a variable.
- An assignment statement can be used as an expression in Java.
  - After a variable is declared, you can assign a value to it by using an **assignment statement**.
  - In Java, the equal sign (=) is used as the assignment operator.
  - The syntax for assignment statements is as follows:

    variable = expression;

```
int z = 2;                          // Assign 2 to variable z
int y = 5 * (3 / 2);                // Assign the value of the expression to y
x = z + 1;                          // Assign the addition of z and 3 to x
double width = 2.1;                 // Assign 2.1 to variable width
double area = width * width;         // Compute area
```

Source: Y. D. Liang, Introduction to Java Programming and Data Structures (11th Edition), Pearson

Sequential Statement (Simple Expression)

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius " + radius);

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius " + radius);
```

Source: Y. D. Liang, Introduction to Java Programming and Data Structures (11th Edition), Pearson

RefIJ_List 2.1.java

```
import java.util.Scanner;

public class ComputeArea {
        public static void main(String[] args) {
                // Create a Scanner object
                Scanner keyboardInput = new Scanner(System.in);

                // Variable declare
                double radius;
                double area;

                 // Prompt the user to enter a radius
                System.out.print ("Enter a number for radius: ");
                radius = keyboardInput.nextDouble();

                // Compute area
                area = radius * radius * 3.1415928;

                // Display results
                System.out.println ("The area for the circle of radius " +
                radius + " is " + area);
        }
}
```

RefIJ_List 2.1.class

```
Enter a number for radius: 24
The area for the circle of radius 24 is 1809.5574528
```

Algorithm Design

```
public class ComputeArea {
        public static void main(String[] args) {
                double radius;
                double area;

                // Step 1: Read in radius

                // Step 2: Compute area

                // Step 3: Display the area
        }
}
```

Arithmetic Operators and Statements

- Most of simple statements in Java can be formed using variables, constants, and arithmetic operators.
  - These operators are + (addition), - (subtraction), *(multiplication), / (division), and % (modulo remainder)
  - A statement can be used anyplace it is legal to use a value of the type produced by the expression.
  - A statement can be **fully parenthesized** in order to specify exactly what subexpressions are combined with each operator.

- More generally, a value of any type in the following list can be assigned to a variable of any type that appears to the right of it

  `byte` → `short` → `int` → `long` → `float` → `double`

  `char` →

  the list from left to right allows values for the types becomes larger

- A **type cast** takes a value of one type and produces a value of another type with an "equivalent" value
  - An **explicit type cast** is required to assign a value of one type to a variable whose type appears to the left of it on the above list (e.g., float to int).

Shorthand Assignment Statement

- Shorthand assignment is the combination of an assignment operator (=) and an arithmetic operator. It is used to change the value of a variable by adding, subtracting, multiplying, or dividing by a specified value.
  - The syntax for shorthand assignment statements is as follows:

    variable op= expression;

    which is equivalent to

    variable = variable op (expression);
  - The **expression** can be another variable, a constant, or a more complicated expression
  - **op** means the operation such as +, -, *, /, or %

```
sum += 3;                    // sum = sum + 2;
price -= discount;           // price = price - discount;
amount *= x + y;             // amount = amount * (x + y)
```

The Class String

- There is no primitive type for strings in Java. The class **String** is a predefined class that is used to store and process strings
  - The String class contains many useful methods for string-processing applications.
  - Objects of type **String** are made up of strings of characters that are written within double quotes (“ “).
  - Always count from zero when referring to the position or index of a character in a string.

```
• String blessing = “Happy Birthday“;
• String ans;
  ans = "The answer is " + “blowing in the wind”;
• String greeting = "Hello";
  int count = greeting.length();
  System.out.println("Length is " + greeting.length());
• String helloJava = "Java Programming!";
  System.out.println(str.toUpperCase());
  System.out.println(str.substring(6,9));
  System.out.println(str.lastIndexOf(“a"));
```

05

Programming Practice

Practice 1： Bit Operations

- Write a program to calculate the following bitwise operations of two input integer data. (Refer to BitwiseOperation.java)
  - The algorithm is designed as follows:
    - ✓ Input two integer data and use *printBinary* function to generate the binary format of each input data.
    - ✓ Use the following bitwise operations to calculate and generate output binary format.
      - Disjunction
      - Conjunction
      - Exclusive
      - right shift three bits of the first input integer
      - left shift three bits of the second input integer
  - ✓ The Output Format: use the following method – *printBinary(int)*

```
private static void printBinary(int a) {
   for (int i=0; i<32; i++) {
   System.out.print((a>>(31-i))&1);
   if (i>0 & (i+1)%4==0) System.out.print(" ");
    }
   System.out.println();
}
```

Practice 2：Arithmetic Operations

① **Fahrenheit to Celsius**

Write a program that converts a Fahrenheit degree to Celsius using the formula Celsius = (5/9) * (Fahrenheit - 32).
Input: Fahrenheit degree
Output: Celsius degree with %6.2f format

② **BMI**

Write a program to calculate the Body Mass Index (BMI) using the formula
BMI = 體重(公斤) 除以 身高(公尺)的平方
Input: Height in meters, Weight in kilograms
Output: BMI with %4.1f format

③ **GPA Conversion**

Write a program to convert the Percentage to GPA using the formula
GPA = (Percentage /100) * 4
Input: Percentage
Output: GPA with %4.2f format