

# **02\_1 Java Basics**

Object-Oriented Programming

# Basic structure of a Java program (1/5)

```
/**
 * This is a JavaDoc comment to generate documentation
 */
public class HelloWorld1 {
    public static void main(String[] args) {
        // Single statement
        System.out.println("Hello, World!");

        // Block of statements
        {
            System.out.println("This is a block.");
            System.out.println("It contains multiple statements.");
        }
    }
}
```

- **public**: access modifier
  - The class can be used anywhere
- **class**: keyword for class definition
- **HelloWorld1**: the name of the class
- **begins** with **{** and **ends** with **}**

# Basic structure of a Java program (2/5)

```
/**
 * This is a JavaDoc comment to generate documentation
 */
public class HelloWorld1 {
    public static void main(String[] args) {
        // Single statement
        System.out.println("Hello, World!");

        // Block of statements
        {
            System.out.println("This is a block.");
            System.out.println("It contains multiple statements.");
        }
    }
}
```

- Method "main"
- The first method executed when the program starts
- Should be included in any executable program

# Basic structure of a Java program (3/5)

```
/**
 * This is a JavaDoc comment to generate documentation
 */
public class HelloWorld1 {
    public static void main(String[] args) {
        // Single statement
        System.out.println("Hello, World!");

        // Block of statements
        {
            System.out.println("This is a block.");
            System.out.println("It contains multiple statements.");
        }
    }
}
```

- **public**: access modifier
  - main method can be called from anywhere
- **static**: method belongs class not instance
- **void**: no return type
- **main**: method name
- **Strings[] args**: array of Strings, command line arguments

# Basic structure of a Java program (4/5)

```
/**
 * This is a JavaDoc comment to generate documentation
 */
public class HelloWorld1 {
    public static void main(String[] args) {
        // Single statement
        System.out.println("Hello, World!");

        // Block of statements
        {
            System.out.println("This is a block.");
            System.out.println("It contains multiple statements.");
        }
    }
}
```

**multi-line comments:** `/* ... */`

`/** ... */`: can be captured by JavaDoc tool (automatic manual generation)

**single-line comments:** `// .....`

# Basic structure of a Java program (5/5)

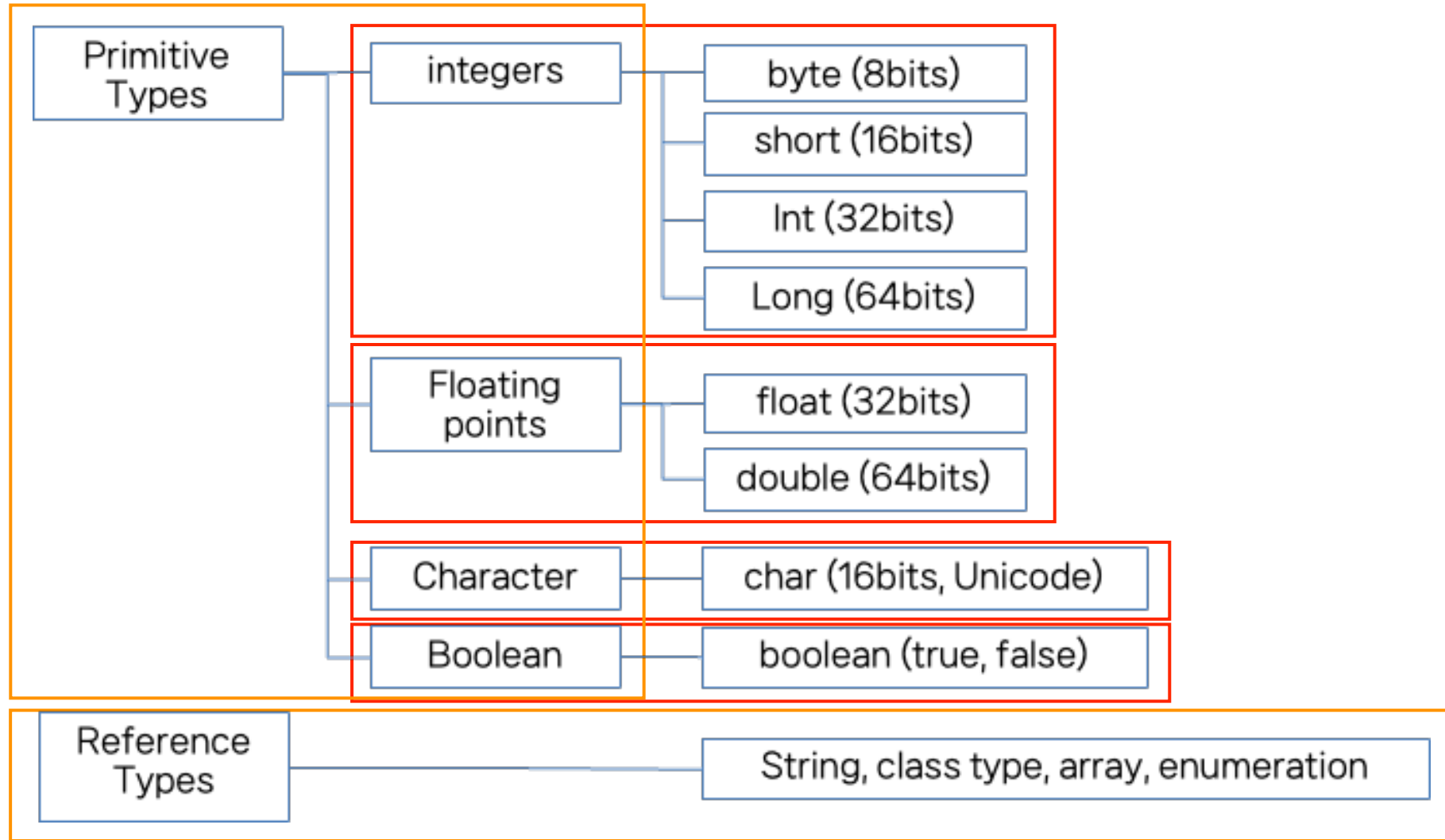
```
/**
 * This is a JavaDoc comment to generate documentation
 */
public class HelloWorld1 {
    public static void main(String[] args) {
        // Single statement
        System.out.println("Hello, World!");

        // Block of statements
        {
            System.out.println("This is a block.");
            System.out.println("It contains multiple statements.");
        }
    }
}
```

# Identifiers

- Identifiers
  - The name of variables, constants, methods, parameters, classes, ...
- Identifier Rules
  - **Case sensitivity:** MyVariable  $\neq$  myVariable
  - **Reserved words (keywords) cannot be used:** class, public, void, ...
  - **General Naming Rules:**
    - Cannot start with numeric characters
    - Cannot use special characters except \$ and \_
  - **Naming Conventions:**
    - camelCase for variables and methods: myVariable
    - PascalCase for classes: MyClass

# Data Types





# Data Types: Example

```
public class DataTypesExample {  
    public static void main(String[] args) {  
        int myNumber; // variable declaration  
        myNumber = 10; // initialization  
        int yourNumber = 10; // declaration with initialization  
        float f1 = 3.151492f; // float type literal should be ended with the suffix 'f'  
        double d1 = 3.151492; // double type literal doesn't have to have any suffix.  
  
        System.out.println("myNumber=" + myNumber + " yourNumber=" + yourNumber +  
            " f1=" + f1 + " d1=" + d1);  
    }  
}
```

OUTPUT:

```
myNumber=10 yourNumber=10 f1=3.151492 d1=3.151492
```

# Type Conversion

```
public class TypeConversion {  
    public static void main(String[] args) {  
  
        int myInt = 10;  
        double myDouble = myInt;    // Implicit conversion from int to double  
                                     // range of (double) > range of (int)  
        System.out.println("myInt(" + myInt + ") myDouble(" + myDouble + ")");  
  
        myDouble = 9.78;  
        myInt = (int) myDouble;    // Explicit conversion from double to int  
  
        System.out.println("myInt(" + myInt + ") myDouble(" + myDouble + ")");  
    }  
}
```

OUTPUT:

```
myInt(10) myDouble(10.0)  
myInt(9) myDouble(9.78)
```

# Named Constants

- Named constants having names:

```
public static final int INCHES_PER_FOOT = 12;  
public static final double RATE = 0.14;
```

- Cannot change the value in the program
- Naming convention for constants: Use all uppercase letters, and designate word boundaries with an underscore character

# Strings

- A class used to handle text

```
public class StringClass {  
    public static void main(String[] args) {  
        String greeting = "Hello, World!";  
        String firstName = "John";  
        String lastName = "Doe";  
  
        // Concatenation  
        String fullName = greeting + " " + firstName + " " + lastName;  
  
        System.out.println(fullName); // Outputs "Hello, World! John Doe"  
    }  
}
```

# Concatenation of Strings

```
String str4 = "The answer is " + 42    // "The answer is 42"  
int k = 35;  
String str5 = "Yes " + k;              // "Yes 35"
```

# String Indexes

"Java is fun."

0	1	2	3	4	5	6	7	8	9	10	11
J	a	v	a		i	s		f	u	n	.

# String Methods (1/3)

```
public class StringMethods {  
    public static void main(String[] args) {  
  
        String str = "Hello, World!";  
  
        int length = str.length(); // length of the string: 13  
        char ch = str.charAt(0); // character at a specific index position: 'H'  
  
        // substring from the given begin index to the end: "World!"  
        String substr1 = str.substring(7);  
  
        // substring from index1 to index2: "Hello"  
        String substr2 = str.substring(0, 5);  
  
        // str과 given String의 내용 비교 (reference, 즉, 주소 비교 아님) : true  
        boolean isEqual = str.equals("Hello, World!");  
  
        // 대소문자 구분없이 내용 비교: true  
        boolean isEqualIgnoreCase = str.equalsIgnoreCase("hello, world!");  
    }  
}
```

# String Methods (2/3)

```
// dictionary order로 str > "Hello" 이면 positive, str < "Hello" 이면 negative
// str == "Hello" 이면 0을 return
int comparison = str.compareTo("Hello"); // Positive value
int comparisonIgnoreCase = str.compareToIgnoreCase("hello"); // Positive value

int index = str.indexOf("World"); // 처음 출현하는 World의 W의 index: 7
int lastIndex = str.lastIndexOf("o"); // 마지막 출현하는 o의 index: 8
boolean contains = str.contains("Hello"); // 주어진 substring을 포함하는가? true

String replacedStr = str.replace('o', 'a'); // "Hella, World!"
String replacedStr2 = str.replace("World", "Java"); // "Hello, Java!"
String replacedAllStr = str.replaceAll("l", "L"); // "HeLlO, WorLd!"
String replacedFirstStr = str.replaceFirst("l", "L"); // "HeLlO, World!"

String upper = str.toUpperCase(); // 대문자로: "HELLO, WORLD!"
String lower = str.toLowerCase(); // 소문자로: "hello, world!"
```



# String Methods (3/3)

```
String trimmedStr = str.trim(); // 앞뒤 공백이 제거된 문자열
String[] words = str.split(", "); // words[0] = "Hello", words[1] = "World!"
String joinedStr = String.join(", ", "Hello", "World"); // "Hello, World"

String intStr = String.valueOf(123); // integer 123을 String "123" 으로
String boolStr = String.valueOf(true); // boolean true를 String "true"로

boolean startsWith = str.startsWith("Hello"); // true
boolean endsWith = str.endsWith("!"); // true
boolean isEmpty = str.isEmpty(); // false
}
```

# Escape Sequences

- 1) 프로그램에서 특별한 의도로 사용되는 문자
- 2) 눈에 안보이는 특수 문자

```
\ " Double quote.  
\ ' Single quote.  
\ \ Backslash.  
\ n New line. Go to the beginning of the next line.  
\ r Carriage return. Go to the beginning of the current line.  
\ t Tab. White space up to the next tab stop.
```

Ex) `System.out.print("Hey Guys\\\n\"0h\t\'Yes!!");`

```
Hey Guys\  
"0h      \'Yes!!
```

# String Processing

- A **String** object in Java
  - immutable, i.e., the characters it contains cannot be changed
- **StringBuffer**
  - Can be changed
- [NOTE] Possible to change the value of a **String** variable by using an assignment statement

```
String name = "Soprano";  
name = "Anthony " + name;
```

# Character Sets - ASCII

- ASCII: 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
  - Each character is represented by a particular number
  - 1 byte (8 bits)

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

\* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

# Character Sets - Unicode

- Unicode: A character set used by the Java language
  - includes all the ASCII characters plus many of the characters used in languages with a different alphabet from English (ex. Korean)
  - 2 bytes (16 bits)