1DV532 – Starting Out with Java

Lesson 2

Java Basics

Dr. Nadeem Abbas
nadeem.abbas@lnu.se

Getting Started

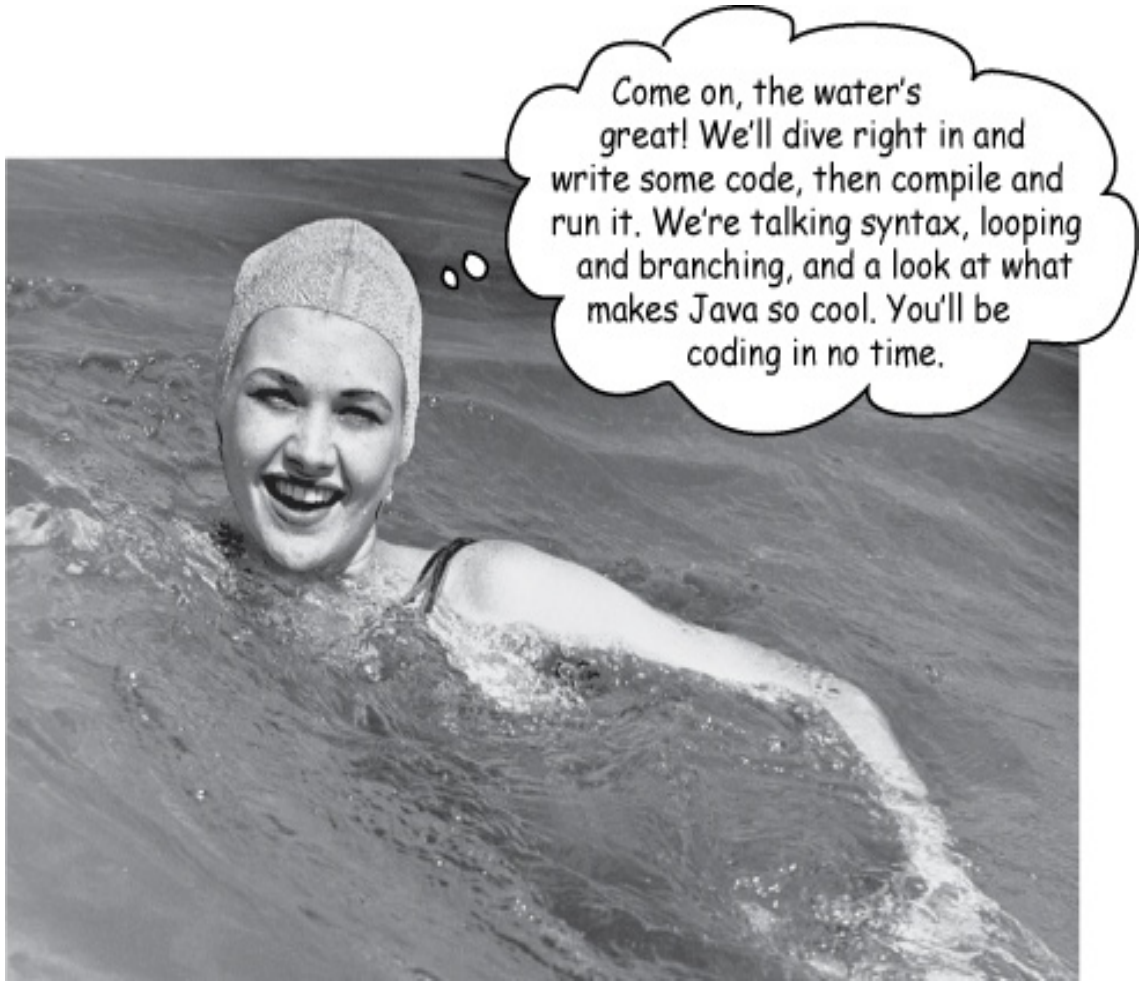

*Image taken from the book “Head First Java”, 2nd Edition by Bert Bates, Kathy Sierra

Introduction To Java

- **Java** is a **high-level object-oriented programming language**
  - It was developed by **Sun Microsystems**, team led by **James Gosling** in 1991 but **now owned by Oracle**.

- We have already learnt, *what a programming language is* in Lesson 1; this lesson introduces Java as a general purpose programming language
- we will learn about "object-oriented" in the next step of the course

Java Programming – Basic Concepts

Basic Concepts

- Java programs are always saved in files ending with (the postfix) ***.java,*** e.g., *Vehicle.java, Game.java, Player.java.*
- Every Java program must have at least one **Class**.
  - **Class** is a basic building block of an object-oriented language such as Java. It serves as a template that specifies states and behavior for objects (instances) of that class. We will learn more about objects and classes in next step.
- Each class has a **Name**, and should be saved in a file having same **Name** as that of the class, for instance, a class named *Game* should be saved in a file *Game.java*
- By convention, class names start with an **Uppercase** letter, such as *Vehicle.java, Game.java, Player.java*

Basic Concepts

- **Data** associated with a Class or Objects is stored in *variables* and the **behavior** associated with a class or objects is implemented by defining *methods*.

- A **method** is a collection of ***statements*** that are grouped together to perform an operation.

- A **statement** represents an action or a sequence of actions.
  - For instance, following statement declares a variable named *sum* of type *int*
    - `int sum;`
- Statements generally involve *variables* and *operators*, and Each statement in Java ends with a semicolon ;

Basic Concepts

- A Java program is basically a set of statements.
- In addition to statements, a java program may contain *comments*
- **Comments** are a type of *statements* that are not considered by the compiler and interpreter.
  - The comments are used for documentation purpose, to explain the code and make it more readable and understandable for programmers
- Java supports three types of comments
  1. **Single-Line** (to comment only one line)
  2. **Multi-Line** (to comment more than one lines)
  3. **Javadoc / Documentation** (to create formal API documentation)

Java Example Program - AddNumbers.java

```
/**
 * AddNumbers.java
 * Author: Nadeem Abbas
 */
package dv532.st19.lect01;// package declaration
/*
 * A program that adds two numbers and
 * displays their sum on the screen.
 */
public class AddNumbers {  // Class declaration
   public static void main(String[] args) { // main method
           int a=5, b=10;
           int sum = a+b;
           System.out.println("sum of "+a+" and "+b+" = "+sum);
   }
}
```

Note: See **Lesson 3** for details about *how to create, compile and execute* a java program.

Java Example Program - AddNumbers.java

```
/**
 * AddNumbers.java
 * Author: Nadeem Abbas
 */
package dv532.st19.lect01;// package declaration
/*
 * A program that adds two numbers and
 * displays their sum on the screen.
 */
public class AddNumbers {  // Class declaration
    public static void main(String[] args) { // main method
            int a=5, b=10;
            int sum = a+b;
            System.out.println("sum of "+a+" and "+b+" = "+sum);
    }
}
```

The program contains a class named "AddNumbers" thus it should be saved in a file named "**AddNumbers.java**"

This is an example of a package declaration ***statement***. **Packages** are used to group related classes.

This is an example of a **main method** which in an entry point for a java application (program).
Every executable java application is required to contain a main method with signatures as shown in the example.

Java Example Program - AddNumbers.java

```
/**
 * AddNumbers.java
 * Author: Nadeem Abbas
 */
```

This is an example of a **javadoc comment.**

```
package dv532.st19.lect01;// package declaration
```

```
/*
 * A program that adds two numbers and
 * displays their sum on the screen.
 */
```

This is an example of a **multi-lines comment.**

```
public class AddNumbers {  // Class declaration
    public static void main(String[] args) { // main method
        int a=5, b=10; // variable declaration and initialization
        int sum = a+b;
        System.out.println("sum of "+a+" and "+b+" = "+sum);
    }
}
```

The **System.*out.println*** *statement* is used for console output, i.e., to print output on screen.

This is an example of a **single-line comment.**

Variables

- A **variable** is the basic unit of storage to store data.
- Variable Declaration
  - *type identifier [= value];*
  - *type identifier [= value][, identifier [= value] ...];*

- Variable Declaration Examples
  - int age;
  - int count = 10;
  - double weight = 70.25, height=5.3, length= 2.5 ;
  - char = ‘x’;
  - boolean isStudent = true;

Data Types – Primitive Type

- Java is a statically typed language
  - All variable data types must be declared before they can be used

- Java data types are divided into two categories
  1. **Primitive Types**
  2. **Reference Types**

- Java has 8 Primitive Types
  - 4 integer types: ***byte, short, int, long*** (sizes 1 – 8 bytes)
  - 2 floating-point types: ***float, double*** (sizes 4 – 8 bytes)
  - 1 character type: ***char*** (size 2 bytes)
  - 1 logical type: ***boolean***. Size 1 byte, only two possible values true or false

Data Types – Reference Type

- A **reference type** is a data type that's based on a Class definition rather than the primitive types

- Each class defines a new *reference type*
- Reference types have a non-trivial behavior defined by their methods

- A **fundamental difference between primitive and reference types** is that primitive type variables store the actual values, whereas reference type variables store memory addresses of the objects they refer to.

Type Conversions

- Type conversions occurs when we assign a value of one type to a variable of another type. For instance,
  - `int count = 10;`
  - `double amount = count;`
- **Implicit (Automatic/Widening) Type Conversion**
  - Such type conversion take place automatically if
    - The two types are compatible.
    - The destination type is larger than the source type
  - Numeric data types are compatible with each other

    Byte –> Short –> Int –> Long – > Float –> Double
  - No automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other

Type Conversion

- **Explicit (Narrowing) Type Conversions**
  - Assigning a value of larger data type to a smaller data type, requires explicit type casting

    Double –> Float –> Long –> Int –> Short –> Byte

  - For instance
    - double d = 323.142;
    - int i = (int) d;
    - int num = 88;
    - ch = (char) num;
  - The *char* type in Java is a subset of type *int*

Operators

- An operator is a symbol that tells the compiler to perform a specific mathematical, logical, or some other operation.
- **Assignment Operator:** =
- **Arithmetic Operators:** +, -, *, /, %
  - Increment, Decrement Operators: ++, -- (postfix, prefix)
  - Compound Assignment Operators: +=, -=, *=, /=, %=
- **Unary Operators:** +, -, ++, --, !
- **Relational Operators:** ==, !=, >, <, >=, <=
- **Logical Operators:** & (AND), | (OR), ! (Not), ^ (XOR)
- **Bitwise Operators:** ~, &, |, ^, >>, >>>, << ,

Assignment Operator =

- Assignment operator is one of the most simple and common operators that is used to ***assign*** a value on its right to an operand on its left
- For example
  - `int area = 0;` // assigns a value 0 to the operand *area*
  - `int speed = 10;` // assigns a value 10 to the operand *speed*
  - `speed = 20;` // assigns a value 20 to the operand *speed*

Arithmetic Operators

- Arithmetic operators **+, -, *, /, and %** perform **addition, subtraction, multiplication, division, and modulo (remainder)** operations, respectively.
  - The arithmetic operators can be applied on any numeric type: byte, short, int, long, float, or double.

- We can combine the arithmetic operators with the simple assignment operator to create **compound assignments**.
  - we can use *x += 1;* in place of x = *x + 1;* both increment the value of x by 1.
  - *x -= 1;* in place of x = *x - 1;* both decrement the value of x by 1.
  - *x *= 2;* in place of x = *x * 2;* both multiply the value of x by 2.
  - *X /= 2;* in place of x = *x / 2;* both divide the value of x by 2.

ArithmeticDemo.java – Example

```
// An example program that demonstrates use of the arithmetic operators
class ArithmeticDemo {
    public static void main (String[] args) {
          int result = 1 + 2;
          System.out.println("1 + 2 = " + result);

          int original_result = result;
          result = original_result - 1;
          System.out.println(original_result + " - 1 = " + result);

          original_result = result;
          result = original_result * 8;
          System.out.println(original_result + " * 8 = " + result);

          original_result = result;
          result = original_result % 7;
          System.out.println(original_result + " % 7 = " + result);
    }
}
```

Output:

1 + 2 = 3
3 - 1 = 2
2 * 8 = 16
16 % 7 = 2

Unary Operators

- Unlike arithmetic and other binary operators, the unary operators require only one operand.
- Table below lists unary operates

Operator	Description
+	**Unary plus operator**; indicates positive value (numbers by default are positive)
-	**Unary minus operator**, indicates negative value or negates an expression
++	**Increment operator**, increments a value by one, used in following two forms: • **Prefix** (Value is incremented first and then result is computed) • **Postfix** (Value is first used for computing the result and then incremented)
--	**Decrement operator**, decrements a value by one, used in following two forms: • **Prefix** (Value is incremented first and then result is computed) • **Postfix** (Value is first used for computing the result and then decremented)
!	**Logical complement operator**; inverts a boolean value

UnaryDemo.java - Example

```
// A program that demonstrates use of the unary operators
class UnaryDemo {
    public static void main(String[] args) {
        int result = +1;
        System.out.println(result); // result is now 1

        result--;
        System.out.println(result); // result is now 0

        result++;
        System.out.println(result); // result is now 1

        result = -result;
        System.out.println(result); // result is now -1

        boolean success = false;
        System.out.println(success); // prints false
        System.out.println(!success); // prints true
    }
}
```

Relational Operators

- Java has six relational operators that compare two operands and return a boolean value.
  - The comparison is performed to determine if one operand is *greater than, less than, equal to, or not equal* to another operand
- The relational operators are: **<, >, <=, >=, ==,** and **!=**.

1.	x < y	returns	True if x is less than y, otherwise false.
2.	x > y	returns	True if x is greater than y, otherwise false.
3.	x <= y	returns	True if x is less than or equal to y, otherwise false.
4.	x >= y	returns	True if x is greater than or equal to y, otherwise false.
5.	x == y	returns	True if x equals y, otherwise false.
6.	x != y	returns	True if x is not equal to y, otherwise false.

Logical Operators

- Logical operators, also known as Boolean operators, operate on Boolean values to create a new Boolean value

- Table below lists and describes the logical operators

Operator	Description	Example	Result
!	Logical NOT	!a	True if a is false and false if a is true.
&&	Logical AND	a && b	True if **a** and **b** are both true and false otherwise
||	Logical OR	a || b	True if **a** or **b** or **both** are true and false otherwise

Bitwise Operators

- The bitwise operators operate **bit-by-bit** on individual bits (0s and 1s) of integer (byte, short, int, long) and *char* types.
- The bitwise operators are classified into two groups
  1. Bitwise Logical Operators
  2. Bitwise Shift Operators

Bitwise Logical Operators

Bitwise Operator	Name	Example	Description
&	Bitwise AND	10101110 & 10010010 yields 10000010	The AND of two corresponding bits yields a 1 if both bits are 1.
|	Bitwise (inclusive) OR	10101110 | 10010010 yields 10111110	The OR of two corresponding bits yields a 1 if either bit is 1.
^	Bitwise exclusive OR	10101110 ^ 10010010 yields 00111100	The XOR of two corresponding bits yields a 1 only if two bits are different
~	Bitwise NOT	~10101110 yields 01010001	The operator toggles each bit from 0 to 1 and from 1 to 0.

Bitwise Shift Operators

Operator	Name	Example	Description
<<	Left Shift	10101110 << 2 yields 10111000	The operator shifts bits in the first operand left by the number of bits specified in the second operand, filling with 0s on the right.
>>	Signed Right Shift	10101110 >> 2 yields 11101011 00101110 >> 2 yields 00001011	The operator shifts bit in the first operand right by the number of bits specified in the second operand, filling with the highest (sign) bit on the left
>>>	Unsigned Right Shift	10101110 >>> 2 yields 00101011 00101110 >>> 2 yields 00001011	The operator shifts bit in the first operand right by the number of bits specified in the second operand, filling with 0s on the left.

Operators Precedence (highest to lowest)

Operators	Precedence
postfix	*expr*++, *expr*--
unary	++*expr*, --*expr*, +*expr*, -*expr*, ~, !
multiplicative	*, /, %
additive	+, -
shift	<<, >>, >>>
relational	<, >, <=, >=,
equality	==, !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Strings and Strings Concatenation Operator +

- **String** is a sequence of characters.
- **Character** = letter, digit, whitespace, ... (All possible symbols)
- String Examples
  - "Hello World!" "Stockholm is in Sweden." "x"
  - "(GD&D .,~-{]HG()B(SG-,.M?PNI \\n\t\b”
- **Strings Concatenation**
  - We can create a new string by adding (concatenating) two strings using concatenation operator **“+”**
    - System.out.println("One string" + "Another string");
  - String + String = New String
  - String + Number = New String
    - System.out.println("Number: " + 144); //Output: Number: 144

Identifiers

- Identifiers are the **names** used in a program to identify the program elements such as classes, methods, and variables.
  - For example, **AddNumbers, main, sum** are the identifiers used in our first example program.
- An identifiers may consists of any combination of letters, digits and the characters (_) and ($) But cannot start with a digit.
  - Valid Identifiers: sum, $2 , ComputeArea , _area, circle_area, etc.
  - Invalid Identifiers: 1A, d+4, result@lnu.se, etc.
- An identifier cannot be a **keyword** or a **reserved word**.

Keywords/Reserved Words

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

*	not used
**	added in 1.2
***	added in 1.4
****	added in 5.0

Suggested Readings

- Absolute Java, Global Edition, 6/E by Walter J. Savitch, Chapter 1
- Introduction to Java Programming, Brief Version, Global Edition, 11/E Liang, Chapter 2
- Java Tuotrials:
  - **Language Basics**
    https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html

Lnu.se