CS-101 – WEEK 02 – JAVA BASICS

GIUSEPPE TURINI

Kettering UNIVERSITY

TABLE OF CONTENTS

STUDY GUIDE

Study Material:

- These slides.
- Your notes.
- "Java Illuminated (5th Ed.)", chapter 2, pp. 39-86. *

Practice Exercises:

- Exercises from *: 2.6.1.1-3, 2.6.2.4-22, 2.6.3.23-32, 2.6.4.33-39, 2.6.5.40-43, 2.6.6.44-46, 2.6.7.47-50.

Additional Resources:

- "Java Illuminated (5th Ed.)", appendix A, pp. 1147.
- "Java Illuminated (5th Ed.)": appendix B, pp. 1149.

INTRODUCTION

The main elements of any program are:

- Instructions.
- Data.

Typically, the structure of a program is:

- Receive the input data.
- Process the input data.
- Output the result of the processing.

Note: The input data may be different in each execution of the program, but the instructions stay the same. So, the instructions must be written to correctly handle any input data.

JAVA APPLICATION STRUCTURE

Every Java program consists of at least 1 class.

It is impossible to code a Java program without using classes.

Classes describe a logical entity that include data and methods/functions.

An object is a physical instantiation of a class, and an object contains its own specific data.

Example: We could design a class representing a car, including all data and methods/functions a generic car has/needs (e.g., brand, model, plate, etc.).

Objects of the class car can be generated, and each object will represent a specific car with its own details/data (e.g., ford, taurus, etc.).

Note: You can find more formal definitions of classes/objects in the appendices (even if these definitions will be clear only in the future).

JAVA APPLICATION STRUCTURE (2)

The following code is a shell/skeleton containing the basic format of a Java application:

```
// Note: The class name has to match the file name!
public class Example_02_01__ShellApplication {

   // The Java main method is the entry point of any Java program.
   // Its syntax is always this (you can only change the input variable name).
   public static void main( String[] args ) {

      // WRITE YOUR CODE HERE!

   }

}
```

Note: You can change the class name for each different program you code, but remember that in Java the class name has always to match the code file name where it is stored.

Example: This class (named Example_02_01__ShellApplication) must be stored in the code file Example_02_01__ShellApplication.java (with this exact capitalization).

JAVA STATEMENTS AND BLOCKS

Java statement: It is the basic building block of a Java program.
Each statement represents an instruction.
Each statement ends with a semicolon (;) and can span multiple lines.

White spaces: White space characters are the space, tab, newline, carriage return, etc.
Any number of white spaces is permitted between identifiers, keywords, operands, operators, and literals.

Java block: A Java block is bundled in curly braces ({}) and includes 0 or more statements.
Blocks can be used wherever a statement is legal.
Nesting blocks within blocks is legal.
Blocks are required for method and class definitions.

Note: It is good practice to properly use of white spaces to make your code more readable.

JAVA COMMENTS

Code comments are used to document your code by integrating in the code information useful to yourself (notes) and to other programmers reading your code (explanations).

Code comments are ignored by the compiler (and not compiled).

Java single-line comment: A code line starting with 2 forward slashes (//) is a comment.
A single-line comment ends at the end of that code line.

```
// This is a single-line code comment.
```

Java block comment: Any code part embedded between a block comment start (/*) and a block comment end (*/) is completely marked as a comment.
A block comment can span multiple code lines.

```
/* This is a block code comment
   spanning 2 code lines */
```

DATA IN JAVA

In this section we will focus on how to represent data in Java.

In particular we will discuss:

- How to create variables and constants to store the data we need in Java.
- How to specify the type of the data we need (numbers, characters, etc.).
- The differences between the various data types available in Java.
- How to initialize and modify the values stored in our variables.

In the next section, we will discuss how to process the data by using different operators.

JAVA IDENTIFIERS

Java identifiers are symbolic names that we assign to variables, methods, and classes.

Java identifiers must start with any character in a-z or A-Z, or the underscore (_), or the dollar sign ($). Them, the first character can be followed by any combination of characters and/or digits (0-9), but no space.

Java identifiers are case-sensitive, so “Name1” and “name1” are different identifiers.

None of the Java reserved keywords can be used as identifiers (e.g., you cannot use “class” as an identifier). You can find the complete list of Java reserved keywords in the appendices.

JAVA VARIABLES AND CONSTANTS

Java allows us represent data in a program by using variables and constants (and literals).

Java variable: A variable is a named location in memory where we can store values.
A variable can store 1 value at a time, but that value can be changed.
We can use variables and/or values in any expression.

```
int var1 = 123; // Integer variable var1 initialized to 123.
```

Java constant: A constant is a named location in memory where we can store values.
A constant can store 1 value at a time, but that value cannot be changed.
We can use constants and/or values in any expression.
Defining constants has the same syntax as defining variables, except that the data type is preceded by the Java keyword "final".

```
final int CONST1 = 123; // Constant CONST1 initialized to 123.
```

Java literal: A literal is a constant value that appears directly in the code.
Check the literal formats for the different Java data types in the appendices.

JAVA VARIABLES AND CONSTANTS (2)

Note: Variables need to be declared before they can be used in code.

Note: You cannot declare the same variable (same name) more than once in the same variable scope (an exception is variable shadowing, see appendices).

Note: Once a variable has been declared, its data type cannot change.

Scope: The area of a program where an item (a variable, a method/function, etc.), associated with an identifier, is recognized.

Variable scope: In Java, the scope of a variable is defined as follows:

- Starts where the variable is declared.
- Ends at the end of the code block where that variable is declared.
- (Special cases are class variables and iteration/loop variables).

NAMING CONVENTIONS

Naming convention: A set of rules for choosing the identifier name for a variable, a method/function, etc. in source code and documentation.

Variable naming: The identifier for a variable should start with a lowercase letter, and each word after the first should begin with a capital letter.

```
int numberOfStudentsInCS101; // Standard variable naming.
```

Constant naming: The identifier for a constant should be all uppercase with words separated by underscores (_).

```
final int CS101_COURSE_CAP = 24; // Standard constant naming.
```

Note: Consider that naming conventions cover several programming elements (e.g., class names), and that each language/company may use different naming conventions.

JAVA DATA TYPES

In Java, whenever we define a new variable/constant we need to specify the type of the value we will store in that variable/constant.

Java supports 8 primitive data types: byte, short, int, long, float, double, char, and boolean.

The data type we specify for a variable/constant will be used by the compiler to monitor our use of that variable/constant (to ensure there are no errors related to its data type).
Java is a strongly typed programming language, because it includes this kind of monitoring.

Note: For a more detailed description of all Java primitive data types check the appendices.

Note: In programming, we often refer definition/declaration of elements. Depending on the programming language, these terms may be synonyms or could be very different. More info on the difference between definition and declaration in the appendices.

JAVA DATA TYPES: INTEGERS

An integer data type is one that can store a signed (positive or negative) whole number.

Java provides 4 integer data types: int, short, long, and byte.
These 4 integer data types differ in the number of bytes of memory used to store each type.
The different byte sizes also mean different value ranges (min and max).
This table summarizes the details of the different integer data types available in Java.

INTEGER DATA TYPE	SIZE	MIN VALUE	MAX VALUE
byte	8-bit	-128	127
short	16-bit	-32768	32767
int	32-bit	-2147483648	2147483647
long	64-bit	-9233372036854775808	9233372036854775807

Note: More details on integers data types in the appendices of week 1.

JAVA DATA TYPES: FLOATS

A floating-point data type is one that can store a signed number with fractional parts.

Java provides 2 floating-point data types: float, and double.
A float is a single-precision floating-point signed value (32-bit).
A double is a double-precision floating-point signed value (64-bit).
This table summarizes the details of the different floating-point data types available in Java.

FLOATING-POINT DATA TYPE	SIZE	MIN POS NON-ZERO VALUE	MAX VALUE
float	32-bit	1.4E-45	3.4028235E38
double	64-bit	4.9E-324	1.79769313486231 57E308

Note: More details on floating-point data types in the appendices of week 1.

JAVA DATA TYPES: CHARACTERS

A character data type is one that can store a character.

Java stores characters using the Unicode standard.
Java provides 1 character data type: char.
Unicode characters are represented as unsigned integers using 16 bits.
This table summarizes the details of the character data type available in Java.

CHARACTER DATA TYPE	SIZE	MIN VALUE	MAX VALUE
char	16-bit	0000 0000 0000 0000 (null char)	1111 1111 1111 1111 (invalid char)

Note: More details on the character data type in the appendices of week 1.

Note: For text data (longer than a single character), Java provides the String class, that will be discussed later.

JAVA DATA TYPES: BOOLEANS

A boolean data type is one that can store only 2 values: true or false.

Java provides 1 boolean data type: boolean.
Boolean data types are usually used for decision-making and control flow.
This table summarizes the details of the boolean data type available in Java.

BOOLEAN DATA TYPE	SIZE	POSSIBLE VALUES
boolean	32-bit (depends on JVM)	true, false

Note: In Java, the size in memory of a boolean data type is not specified, and it depends on the specific JVM used (usually it is 32-bit but it could be as small as 1 byte).

ASSIGNMENT OPERATOR

When we declare a variable, we usually initialize it (see also default values).

To assign a value to a variable we use the assignment operator (=).

```
int var1 = 24; // Initialization of integer variable with a literal value.
int var2 = var1; // Initialization of integer variable with a variable.
```

Note: The assignment operator (=) is executed right-to-left, first the right-part (the expression) is evaluated, and then the left-part (the variable) is assigned the value.

Note: For the assignment operator (=) to be executed correctly, the right-part (the expression) must be valid (a computable expression and/or variables already defined).

Note: A common programming mistake is to incorrectly use the assignment operator (=) instead of the equality operator (==), and vice versa.

ASSIGNMENT OPERATOR: DATA TYPES

Note: For the assignment operator (=) to be executed correctly, the right-part (the expression) and the left-part (the variable) must have compatible data types (the left-part precision must be greater or equal than the right-part precision). See this table.

VARIABLE DATA TYPE (LEFT-PART)	COMPATIBLE DATA TYPES (RIGHT-PART)
byte	byte
short	byte, short
int	byte, short, int, char
long	byte, short, int, char, long
float	byte, short, int, char, long, float
double	byte, short, int, char, long, float, double
boolean	boolean
char	char

INTRODUCTION TO STRINGS

For text data (longer than a single character), Java provides the String class and String literals.

String literal: A String literal is an object of the String class.
A String literal is a sequence of characters enclosed by double quotes (").

```
String var1 = "ABC"; // Variable initialized with String literal.
```

String concatenation: The String concatenation operator (+) joins multiple String literals (and other variables, with values automatically converted to Strings).

```
String var1 = "ABC" + "DEF"; // String concatenation.
int var2 = 123;
String var3 = "ABC" + var2; // String concatenation.
String var4 = "ABC" + 456; // String concatenation.
```

Note: String literals cannot extend over multiple lines, use the newline character '\n' instead.
See a list of all escape sequences in appendices.

ARITHMETIC OPERATORS

In this section we will focus on arithmetic operators and expressions in Java.

We will start discussing 5 basic arithmetic operators:

- Addition operator (+) to add 2 numeric values.
- Subtraction operator (-) to subtract 2 numeric values.
- Multiplication operator (*) to multiply 2 numeric values.
- Division operator (/) to divide 2 numeric values.
- Modulus operator (%) to get the remainder of the division of 2 numeric values.

Note: The data type of the operands could affect the execution of an operator (for example a division between integers).

OPERATOR PRECEDENCE

Operator precedence: Set of rules used by the compiler (e.g., the Java compiler) to determine the execution order of the operations in a statement.

This figure shows the operator precedence for the Java operators discussed so far.

Operator Hierarchy	Order of Same-Statement Evaluation	Operation
()	left to right	parentheses for explicit grouping
*, /, %	left to right	multiplication, division, modulus
+, –	left to right	addition, subtraction
=	right to left	assignment

Note: It is a good programming practice to not memorize/use these rules, and instead use parentheses for readability/disambiguation.

Note: A full operator precedence table for all Java operators is included in the appendices.

INTEGER DIVISION AND MODULUS

Division (/) between floating-point numbers results in a floating-point number.

Division (/) between integer numbers results in an integer number, any fractional part is truncated (not rounded). The remainder of the division is available as an integer by using the modulus operator (%).

In other words, in Java, the integer division (/) calculates the quotient of the division, whereas the modulus (%) calculates the remainder.

Note: The integer division and the modulus are independent calculations. You can perform the division without calculating the modulus, and vice versa.

Note: The modulus is a useful operator, it can be used to determine if a number is even or odd, to check if a number is a factor of another, and for many other uses.

DIVISION BY ZERO

If an integer division by 0 is executed, the JVM generates an exception (runtime error) and an error message is displayed on the Java console (but no error at compilation time).

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

If a floating-point division by 0 is executed, the JVM does not generate an exception, but:

- If the dividend is non-zero (non-zero/zero), the result is Infinity.
- If the dividend is zero (0/0), the result is NaN (Not a Number).

Note: It is good programming practice to avoid dividing any number (integer or floating-point) by zero (by preventing that situation to happen).

MIXED-TYPE ARITHMETIC

When calculations of mixed types are performed, lower-precision operands are converted/promoted to the precision of the higher-precision operand, using the first (top-to-bottom) of these rules that fits the situation:

- If either operand is a double, the other operand is converted to a double.
- If either operand is a float, the other operand is converted to a float.
- If either operand is a long, the other operand is converted to a long.
- If either operand is an int, the other operand is converted to an int.
- If neither operand is a double, float, long, or an int, both operands are converted to int.

Note: These conversions/promotions change data types only temporarily (until the calculation is performed).

Note: This arithmetic conversion/promotion of operands is called implicit type casting (because the compiler performs these type conversions automatically).

EXPLICIT TYPE CASTING

Sometimes it is useful to instruct the compiler to convert (temporarily) the type of a variable, so that we can perform a calculation properly. This is called explicit type casting.

Casting: Formally called explicit type casting (or explicit type conversion), this is a special programming statement to specify the data type to use for a variable during the computation of an expression.

This conversion is only temporary, until the expression is computed.

```
int total = 99;
int count = 10;
double average = (double) (total) / count; // Explicit type casting.
```

Note: A cast (an explicit type cast) will always ignore extra information in the data type being casted (but will never add information to the data type being casted).

SHORTCUT OPERATORS

Java provides shortcut operators for each of the basic arithmetic operations (and more).

This figure shows a summary of the main shortcut operators (but there are many more).

Shortcut Operator	Example	Equivalent Statement
++	a++; or ++a;	a = a + 1;
−−	a−−; or −−a;	a = a − 1;
+=	a += 3;	a = a + 3;
−=	a −=10;	a = a − 10;
*=	a *= 4;	a = a * 4;
/=	a /= 7;	a = a / 7;
%=	a %= 10;	a = a % 10;

Note: Remember that prefix versions (++a) are different than postfix versions (a++), so be careful in using these shortcut operators.

APPENDICES

Appendices

- Classes and Objects
- Java Reserved Keywords
- Literal Formats for Data Types
- Variable Shadowing
- Java Primitive Data Types
- Definition vs Declaration
- Java Escape Sequences
- Operator Precedence Table
- Using "JShell" in jGRASP

CLASSES AND OBJECTS

Class: In Object-Oriented Programming (OOP), a class is a custom type.
This means that a class is a new type defined by the programmer to be used as an extensible template for creating objects, providing data for the object state and instructions (methods/functions) for the object behavior.

Object: In Object-Oriented Programming (OOP), an object is an instance of a class.
This means that an object is a specific item of the class type.
For example: in the real world we know what an apple is, but the concept of the apple (the class/type) is just an abstract concept that is different than a specific apple (the object/instance) with its own color, weight, etc.

Note: Each object/instance of a class/type has its own specific data and methods, but both the data and methods have been specified in the object class/type.

JAVA RESERVED KEYWORDS

KEYWORDS					LITERALS
abstract	continue	for	new	switch	true
assert ***	default	goto *	package	synchronized	false
boolean	do	if	private	this	null
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	

Note: All these keywords and literals cannot be used as identifiers. Some are not currently used (*), some have been added in Java 1.2 (**), or in Java 1.4 (***), or in Java 5.0 (****).

LITERAL FORMATS FOR DATA TYPES

PRIMITIVE DATA TYPE	LITERAL FORMAT
byte, short, int, long	Optional sign (+, –) followed by digits 0–9 in any combination. A byte/short/int/long literal that begins with a 0 digit is considered to be an octal number (base 8) and the remaining digits must be 0–7. A byte/short/int/long literal that begins with 0x is considered to be a hexadecimal number (base 16) and the remaining digits must be 0–9 or A–F. A long literal must terminate with an L or l (uppercase preferred).
float	Optional sign (+, –) followed by a floating-point number in fixed or scientific format, terminated by an F or f (uppercase preferred).
double	Optional sign (+, –) followed by a floating-point number in fixed or scientific format.
char	Any printable character enclosed in single quotes. A decimal value from 0 to 65,535. '\unnnn' where nnnn are hexadecimal digits. '\m' is an escape. '\n' is a newline. '\t' is a tab.
boolean	Only true or false.

VARIABLE SHADOWING

In programming, variable shadowing (or name masking) occurs when a variable (var_1) has the same name as another variable (var_2) declared in an outer scope (the scope of var_1 includes the scope of var_2).

Note: In this example, var_1 and var_2 have the same name (but different scopes).

```
// Example on variable shadowing (or name masking).
public class Example_02_99__VariableShadowing {

   static int var1 = 123; // Class variable.

   public static void main( String[] args ) {
      System.out.println(var1);
      int var1 = 456; // Local variable with same name (variable shadowing).
      System.out.println(var1);
   }

}
```

Note: Variable shadowing (or name masking) is not a good programming practice. Avoid it by using different variable names, or a different syntax (this, class name, etc.).

JAVA PRIMITIVE DATA TYPES

PRIMITIVE DATA TYPE	DESCRIPTION	DEFAULT VALUE
byte and short	8-bit and 16-bit signed two complement integers, respectively.	0 and 0
int	32-bit signed two complement integer.	0
long	64-bit signed two complement integer.	0L
float and double	32-bit and 64-bit IEEE 754 signed floating point, respectively.	0.0f and 0.0d
char	16-bit Unicode character (unsigned).	'\u0000'
boolean	Type with only 2 values: true, false (bit size depends on JVM).	false
object reference	Memory address (bit size depends on JVM).	null

DEFINITION VS DECLARATION

Declaration: In programming, declaring something (e.g., a method) usually means to provide only the minimal information necessary for that item to exist (so that it can be recognized by the compiler).

For example: the declaration of a variable may include only its data type and name, but not its initialization.

For example: the declaration of a method may include only its header/signature, but not its body.

```
float getAreaOfCircle( float radius ); // Declaration of a method.
```

Definition: In programming, defining something (e.g., a method) usually means to provide a full initialization of all its parts (so that it can be used in code).

```
float getAreaOfCircle( float radius ) { // Definition of a method.
   final double PI = 3.14159;
   return PI * radius * radius;
}
```

JAVA ESCAPE SEQUENCES

Escape sequence: In programming, an escape sequence is a combination of characters that has a meaning other than its literal characters.

This table lists all the escape sequences available in Java.

TEXT CHARACTER	JAVA ESCAPE SEQUENCE
newline	\n
tab	\t
double quotes	\"
single quote	\'
backslash	\\
backspace	\b
carriage return	\r
form feed	\f

OPERATOR PRECEDENCE TABLE

This table (part 1) shows all Java operators, along with their precedence and associativity.

PRECEDENCE	OPERATOR	DESCRIPTION	ASSOCIATIVITY
16 (max)	() [] .	Parentheses, array access, member access.	Left-to-right.
15	++ --	Post-increment, post-decrement.	Left-to-right.
14	+ - ! ~ ++ --	Unary: plus, minus, logical not, bitwise not, pre-increment, pre-decrement	Right-to-left.
13	() new	Cast, object instantiation.	Right-to-left.
12	* / %	Multiplication, division, modulus.	Left-to-right.
11	+ - +	Addition, subtraction, string concatenation.	Left-to-right.
10	<< >> >>>	Signed left shift, signed right shift, unsigned right shift.	Left-to-right.
9	< <= > >= instanceof	Less, less-equal, greater, greater-equal, instance of.	Left-to-right.

OPERATOR PRECEDENCE TABLE (2)

This table (part 2) shows all Java operators, along with their precedence and associativity.

PRECEDENCE	OPERATOR	DESCRIPTION	ASSOCIATIVITY
8	== !=	Equal, not-equal.	Left-to-right.
7	&	Bitwise and.	Left-to-right.
6	^	Bitwise xor.	Left-to-right.
5	|	Bitwise or.	Left-to-right.
4	&&	Logical and.	Left-to-right.
3	||	Logical or.	Left-to-right.
2	?:	Ternary conditional.	Right-to-left.
1	= += -= *= /= %= &= ^= |= <<= >>= >>>=	Assignments.	Right-to-left.
0 (min)	->	Lambda expression arrow.	Right-to-left.

USING "JSHELL" IN JGRASP

Starting with Java version 9, the JDK includes the JShell tool.

JShell: The Java Shell tool (JShell) is an interactive tool for learning Java and prototyping code. This tool is a Read-Evaluate-Print Loop (REPL), which evaluates declarations, statements, and expressions as they are entered and immediately shows the results. This tool is run from the command line.

jGRASP does not include an interface with JShell.

However, you can use the "Interactions" tab in jGRASP to access the same functionalities available in JShell.