Lecture 2: Elementary Programming

Based on Chapter 2 in Introduction to Java Programming, Y. Daniel Liang, Brief Version, 9/E

You should be familiar with most of the concepts covered in this chapter. We will quickly go over the following topics:

- Problem, algorithm, pseudocode, code
- Variables: storage and memory, types, naming convention
- Reading input from console
- Named constants
- Assignment operator
- Mathematical operations: numerical operators, order of operations, shorthand operators, type conversions, casting, integer vs. double division
- Character data types: ASCII vs. Unicode, escape sequences, casting to/from integer types
- String type
- Programming style, documentation, indentation, use of white space, variable names

1 Problem, algorithm, pseudocode, code

We write programs to solve real life problems (well, maybe not at the beginning, but eventually). Writing such program involves:

- designing an algorithm (using natural language and pseudocode);
- writing source code that implements the algorithm.

An algorithm describes how the problem is solved by listing ordered steps that need to be taken (think of a cook book recipe, that is an algorithm for preparing a specific dish). An algorithm can be described by a natural language (preferably English for this course) and/or pseudocode which resembles code, but is not language specific). The source code should be obtained by implementing each step of the algorithm using proper syntax of a specific programming language (Java in this course).

WARNING: You should NEVER start by writing source code and then figuring out the algorithm once most of it is written. The initial description of the algorithm gives you an easy way of documenting the source code.

2 Variables: storage and memory, types, naming convention

A variable represents a value stored in the computer’s memory. Each variable has:

- name;
- type;
- memory location;
- value.

Variable declaration syntax:

    type variable_name;

Variable assignment syntax:

    variable_name = variable_value;
Name of a variable

Names should be descriptive, i.e., they should indicate what the variable is used for.

Names of variables have to follow rules for naming identifiers (elements such as classes, methods and variables in the Java program). An identifier is a sequence of characters that:

- consists of letters, digits, underscores (_), and dollar signs ($)
- starts with a letter, an underscore (_), or a dollar sign ($) (cannot start with a digit)
- cannot be a reserved word (see Appendix A, "Java Keywords," for a list of reserved words)
- cannot be true, false, or null
- can be of any length.

Example:
valid names:

$2, area, Area, _showMessage, num_of_students, numOfStudents

invalid names

2pounds, a+b, public, int

Type of a variable

Java provides several built-in primitive data types:

- byte
- short
- int
- long
- float
- double
- boolean
- char

For more on primitive data types in Java see: http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html. Later on in the semester we will learn how to create user defined data types.

Memory location of a variable

The memory for a variable is allocated when you declare it. The size of the chunk of memory associated with a given variable depends on its type and differs in different programming languages. Memory location of a variable cannot change after the variable is created.
Value of a variable

The value of a variable is stored in the memory associated with the given variable. The computer memory can only store sequences of zeros and ones (binary numbers). The actual value of a variable is determined using such a sequence of zeros and ones and the type of the variable. The exact same sequence of bits represents a different value when it is stored in memory locations associated with variables of different types.

3 Reading input from console

System.in refers to standard input; by default the keyboard input.

System.out refers to standard output; by default the output you see on the screen.

We saw System.out used previously to print text and numbers on the screen:

    System.out.print("Enter a number for radius:");
    System.out.println("Enter a number for radius:");

In order to read the input from the user, we need a variable/object of type Scanner (Scanner is not one of the primitive data types. It is a class and we refer to variables whose type is a class as objects.) Scanner is the class that provides many ways of reading the input from console. To create a Scanner object in your program, just write:

    Scanner input = new Scanner(System.in);

To Scanner, you need to import the java.util package, i.e., you need the line

    import java.util.Scanner;

or

    import java.util.*;

at the beginning of your file (the second option imports everything from the java.util package).

The non-primitive types come with tools that can be used with objects. Some of the tools provided by the Scanner class are:

- nextByte() reads an integer of the byte type
- nextShort() reads an integer of the short type
- nextInt() reads an integer of the int type
- nextLong() reads an integer of the long type
- nextFloat() reads a number of the float type
- nextDouble() reads a number of the double type
- next() reads a string that ends before a whitespace character
- nextLine() reads a line of text (i.e., a string ending with the Enter key pressed).

To use these tools, you specify the name of the object, followed immediately by a dot (.), followed by one of the names above, for example

    Scanner input = new Scanner(System.in);
    double number = input.nextDouble();

For more (maybe even too much information at this point) see [http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html](http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html)
4 Named constants

A named constant is an identifier that represents a permanent value, i.e., a variable whose value does not change.

Syntax:

```java
final datatype CONSTANTNAME = value;
```

The value of a constant has to be assigned during its declaration. `final` is the keyword that indicates that the value is constant and will not be changed.

Benefits of using constants: You assign the value once, without need to repeat it all over the code. If you need to change the value, it is done in one place. Descriptive name (assuming you use one) makes programs easier to read.
Assignment operator =

In mathematics

variable = value

and

c = variable

are often considered to be equivalent statements.

In Java, the variable has to be on the left hand side of the assignment operator and the value to be assigned to that variable has to be on the right hand side of the assignment operator:

variable = value;

For example, you can write:

```java
int radius;
radius = 5;
```

but NOT

```java
5 = radius; // This is WRONG
```

The compiler interprets the second expression as trying to change the value of the numeral 5 and generates an error.

Mathematical operations: numerical operators, order of operations, shorthand operators, type conversions, casting, integer vs. double division

Binary numeric operators

- + addition: 34 + 6, 34.5 + 2.1
- - subtraction: 34 - 6, 34.5 - 2.1
- * multiplication: 34 * 6, 34.5 * 2.1
- / integer division: 34 / 5, result: 6
- / real division: 34.0 / 5, result: 6.8
- % modulus/Remainder: 34 / 5, result: 4

WARNING: There is no exponentiation operator in Java. You can use repeated multiplications or the method Math.pow(a,b) to compute a^b.

Order of operations is the same as in math: multiplication, division and modulus are performed in order from left to right before addition and subtraction. You can use parenthesis in order to change that behavior.
Shorthand assignment operators:

```
+=  addition assignment: x += 8; short for: x = x + 8;
-=  subtraction assignment: x -= 8; short for: x = x - 8;
*=
|  multiplication assignment: x *= 8; short for: x = x * 8; |
/=
|  integer division assignment: x /= 8; short for: x = x / 8; |
/=  real division assignment: x /= 8; short for: x = x / 8;
%=
|  modulus/Remainder assignment: x %= 8; short for: x = x % 8; |
```

Shorthand increment/decrement operators:

```
++  preincrement: y= ++x; short for: x = x+1; y = x;
++  postincrement: y = x++; short for: y = x; x = x+1;
--  predecrement: y= –x; short for: x = x-1; y = x;
--  postdecrement: y = x–; short for: y = x; x = x-1;
```

Casting

Casting is an operation that converts the value of one data type to another data type (temporarily).

Can you use binary numerical operators with variables of different types? YES.

```
int a = 1;
double b = 2.0;
System.out.println( a / b);
```

The compiler promotes or widens a type of the variable a to perform the addition and obtain the value 0.5 for printing.

You can explicitly cast variable of one type to another:

```
int a = 1;
int b = 2;
System.out.println( a / b);
System.out.println( (double) a / b);
```

prints 0 in the first case, but 0.5 in the second case.

You can explicitly cast variable of one type to another:

```
double a = 1.0;
double b = 2.0;
System.out.println( a / b);
System.out.println( (int) a / (int) b);
```

prints 0.5 in the first case, but 0 in the second case.

**TODO:** Read ALL the sections on numeric operations in chapter 2 and make sure you understand it all (including the code examples). We can discuss other details if there are things that are not clear.
7 Character data types: ASCII vs. Unicode, casting to/from integer types

A character data type char represents a single Unicode character (UTF-8) to be exact.

ASCII (American Standard Code for Information Interchange) characters are a subset of Unicode characters with values from 0 to 127. There are 65,536 possible characters in Unicode (unextended).

How do these numbers correspond to alpha-numeric characters? Each letter, symbol, digit, etc. is mapped to a specific number in Unicode (for a listing of values corresponding to the ASCII subset see Appendix B in your textbook).

You can actually use values stored in variables of type integer to print characters when you use casting:

```java
char letter1 = 'A';
int number1 = 90;
System.out.println("Character " + letter1 + " has code " + (int)letter1);
System.out.println("Character " + (char)number1 + " has code " + number1);
```

prints

Character A has code 65
Character Z has code 90

8 String type

A string is a sequence of characters. In Java we use String type/class to represent strings in the code. Java’s String is not a primitive type.

```java
String message = "Welcome to Java"
```

You can use the + operator to concatenate strings:

```java
String part1 = "Welcome"
String part2 = " to Java"
String message = part1 + part2;
System.out.println(message);
```

prints "Welcome to Java" on the screen.

You can use next() method of the Scanner class to read strings from the user:

```java
Scanner in = new Scanner(System.in);
String name;
System.out.println("Enter you name: ");
name = input.next();
```

For more (definitely too much at this point) information see http://docs.oracle.com/javase/7/docs/api/java/lang/String.html
9 Programming style, documentation, indentation, use of white space, variable names

You always have to document your code,

- include the author’s name (that should be you!), date of creation of the file and purpose (short 2-3 sentence description) of the code;
- if you revise the file, add dates of revisions and short description of changes
- inside the code itself you should use comments to describe what you are doing (do not explain the code itself, but rather the algorithm that the code implements)

Use indentation and white space to make the code easier to read:

- the blocks of code should be indented (in addition to being surrounded by braces { ... } (NOTE: this is different than in Python, use of braces is necessary for multi-line blocks of code.)
- different sections of code should be separated by blank lines

Choose variable names that describe the purpose of the variable. This makes the code easier to read and reduces the need for comments.