Numeric Data Types and Operations

Hui Chen

Department of Computer & Information Science

Brooklyn College

Objectives

- To program with assignment statements and assignment expressions (§2.6).
- To use constants to store permanent data (§2.7).
- To name classes, methods, variables, and constants by following their naming conventions (§2.8).
- To explore Java numeric primitive data types: **byte**, **short**, **int**, **long**, **float**, and **double** (§2.9.1).
- To read a **byte**, **short**, **int**, **long**, **float**, or **double** value from the keyboard (§2.9.2).
- To perform operations using operators **+**, **-**, *****, **/**, and **%** (§2.9.3).
- To perform exponent operations using **Math.pow(a, b)** (§2.9.4).
- To write integer literals, floating-point literals, and literals in scientific notation (§2.10).
- To write and evaluate numeric expressions (§2.11).

Outline

- Discussed
  - From “problem”, to “algorithm”, and to “implementation”
  - Design a program with input and output
    - Hardcode input
    - Read from users’ input (from console)
  - Dissecting the program
- This lesson covers
  - Naming convention (best practice)
  - Review: common errors and pitfalls
  - Numeric data types
  - Read numeric values from users’ input
  - Numeric operators (operating on numeric data types)

Using Identifiers

- What names are valid?
  - Identifiers
    - Variable names
    - Class names
    - Method names
    - Constants

Identifiers

- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
- An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- An identifier cannot be a reserved word.
  - See Appendix A of the textbook, “Java Keywords,” for a list of reserved words.
- An identifier cannot be true, false, or null (they are not keywords, but you cannot use them to name identifers).
- An identifier can be of any length.

Best Practice. Following Naming Convention

- Choose meaningful and descriptive names.
  - For classes, variables, constants, methods
    - We will create our own methods in the future
- Naming conventions for
  - Variables and method names
  - Class names
  - Constants

Variables and Method Names

- Begin with lowercase letters.
- If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name.
- Example
  - the variables `radius` and `area`, and
  - the method `computeArea`.

Class Names

- Begin with uppercase letters
- Capitalize the first letter of each word in the name.
- Example
  - the class name `ComputeArea`

Constants

- All caps!
- Capitalize all letters in constants, and use underscores to connect words.
- Example
  - the constant `PI`
  - the constant `MAX_VALUE`

Best Practice. Using Named Constants

- Why?
- Examples

```
final datatype CONSTANTNAME = VALUE;
final double PI = 3.14159;
final int SIZE = 3;
```

Questions?

- Identifiers?
- Naming convention?
- What lessons did we learn from the experience of writing and testing several programs?
  - Common errors and pitfalls
    - Compilation errors, e.g.,
      - A variable/method/constant/class must be declared before you can reference to it
    - Runtime errors
    - Logical errors
  - How to reduce/eliminate the common errors and pitfalls?

Numeric Data Types

Name	Range	Storage Size
`byte`	-2^7 to $2^7 - 1$ (-128 to 127)	8-bit signed
`short`	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed
`int`	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
`long`	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
`float`	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
`double`	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

Reading Numbers from the Console Input

- Using Scanner and its methods
- Example

```
Java.util.Scanner sc = new java.util.Scanner(System.in)

double d = sc.nextDouble()
```

Scanner and Methods

Method	Description
`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.

Let’s try these methods out

Questions?

Numeric Operations

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 – 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Integer Division

- The result is an integer. This is important!

  +, -, *, /, and %

  5 / 2 yields an integer 2.

  5.0 / 2 yields a double value 2.5

  5 % 2 yields 1 (the remainder of the division)

Remainder Operator

- Remainder is very useful in programming.
- Example
  - How to determine if a number is even or odd?
  - If we were going to meet in 10 days, what day would that day be?

What day is in 10 days?

- Today is Wednesday

Wednesday is the 3rd day in a week (Counting from 0)

A week has 7 days

(3 + 10) % 7 is 6

The 6th day in a week is Saturday

After 10 days

Questions?

Let’s try it out

- Let’s code this …

Problem. Convert Seconds to Minutes and Remaining Seconds

- Write a program to read *seconds* from the console, and obtain the *minutes* and *remaining seconds* from the *seconds*
- Algorithm
  - Read seconds from the console
  - Obtain the minutes in the seconds
  - Obtain the remaining seconds
  - Print out the minute and the remaining seconds in a nice format

Implementation

```
import java.util.Scanner;
public class DisplayTime {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter an integer for seconds: "); // Prompt the user for input
    int seconds = input.nextInt();
    int minutes = seconds / 60; // Obtain minutes in seconds
    int remainingSeconds = seconds % 60; // Obtain seconds remaining
    System.out.println(seconds + " seconds is " + minutes +
      " minutes and " + remainingSeconds + " seconds");  // print those nicely
  }
}
```

Questions?

Integers vs. Float-Point Numbers

- Integers stored exactly while float-point numbers approximately
  - Calculations involving floating-point numbers are approximated
  - Examples
    - `System.out.println(1.0-0.1-0.1-0.1-0.1-0.1);`
    - System.out.println(1.0 - 0.9);
    - (1.0-0.9) == 0.1?

Questions?

- More from our lab …
  - How about
    - System.out.println(Math.PI * 5.8 * 5.8);
    - System.out.println(Math.PI * (5.8 * 5.8));
    - System.out.println(5.8 * 5.8 * Math.PI);
    - …

Exponent Operations

- Use the pow method in the Math class
- Examples

```
System.out.println(Math.pow(2, 3));
// Displays 8.0
System.out.println(Math.pow(4, 0.5));
// Displays 2.0
System.out.println(Math.pow(2.5, 2));
// Displays 6.25
System.out.println(Math.pow(2.5, -2));
// Displays 0.16
```

Questions?

Number Literals

- A *literal* is a constant value that appears directly in the program.
- A *number literal* is a numeric value that appears directly in the program (hard coded numeric values).
- Examples
  - 34, 1,000,000, and 5.0 are literals in the following statements:

```
int i = 34;
long x = 1000000;
double d = 5.0;
```

Have you seen String literals?

Integer Literals and Variables

- An integer literal can be assigned to an integer variable as long as it can fit into the variable.
- A compilation error would occur if the literal were too large for the variable to hold.
- Example.
  - byte b = 1000
- Data types of integer literals
  - An integer literal is assumed to be of the ***int*** type, whose value is between -2^{31} (-2147483648) to $2^{31}-1$ (2147483647).
  - To denote an integer literal of the ***long*** type, append it with the letter L or l.
    - L is preferred because l (lowercase L) can easily be confused with 1 (the digit one).

Questions?

Floating-Point Literals

- Floating-point literals are written with a decimal point.
- Data types of float-point literals
  - By default, a floating-point literal is treated as a double type value.
    - For example, 5.0 is considered a ***double*** value, not a ***float*** value.
  - Make a number a ***float*** by appending the letter f or F, and make a number a ***double*** by appending the letter d or D.
    - For example, you can use 100.2f or 100.2F for a float number, and 100.2d or 100.2D for a double number.

double vs. float

- The double type values are more accurate than the float type values.
- Examples

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

displays `1.0 / 3.0 is 0.3333333333333333`

16 digits

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

displays `1.0F / 3.0F is 0.33333334`

7 digits

Scientific Notation

- Floating-point literals can also be specified in scientific notation
- Examples
  - 1.23456e+2, same as 1.23456e2, is equivalent to 123.456
  - 1.23456e-2 is equivalent to 0.0123456.
  - E (or e) represents an exponent and it can be either in lowercase or uppercase

Questions?

Writing Arithmetic Expressions

- Math

$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

- Java

```
(3+4*x)/5 – 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)
```

Evaluate Arithmetic Expressions

- Though Java has its own way to evaluate an expression behind the scene, the result of a Java expression and its corresponding arithmetic expression are the same.
- Therefore, you can safely apply the arithmetic rule for evaluating a Java expression

Example

```
3 + 4 * 4 + 5 * (4 + 3) - 1
```
(1) inside parentheses first
```
3 + 4 * 4 + 5 * 7 – 1
```
(2) multiplication
```
3 + 16 + 5 * 7 – 1
```
(3) multiplication
```
3 + 16 + 35 – 1
```
(4) addition
```
19 + 35 – 1
```
(5) addition
```
    54 - 1
```
(6) subtraction
```
    53
```

Questions?

Let’s try these out.

Problem. Converting Temperatures

- It is 70 degrees today, is it hot? Your Asian or European friends ask you.
- Convert Fahrenheit degree to Celsius
- Algorithm
  - Read a Fahrenheit degree from users' input on the console
  - Convert the Fahrenheit degree to the Celsius degree

$$celsius = (\tfrac{5}{9})(fahrenheit - 32)$$

  - Print nicely the result

Implementation

```
import java.util.Scanner;

public class FahrenheitToCelsius {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter a degree in Fahrenheit: ");
    double fahrenheit = input.nextDouble();
    // Convert Fahrenheit to Celsius
    double celsius = (5.0 / 9) * (fahrenheit - 32);
    System.out.println("Fahrenheit " + fahrenheit + " is " +
      celsius + " in Celsius");
  }
}
```

Questions?

Lab Exercise. Converting Celsius to Fahrenheit

- Write a program that reads a Celsius degree in a double value from the console, converts it to Fahrenheit, and displays the result with two digits after the decimal points.

$$\text{Fahrenheit} = \frac{9}{5}Celsius + 32$$

Lab Exercise. Compute Volume of Cylinder

- Write a program that reads in the radius and length of a cylinder from the console, compute the surface area and the volume of the cylinder, and display the results ***nicely***.

$$A = 2\pi r^2 + 2\pi r l$$

$$V = \pi r^2 l$$

- where A is the surface area, V the volume, r the radius, and l is length