Chapter 3

Numerical Data

Animated Version

Objectives

After you have read and studied this chapter, you should be able to

- Select proper types for numerical data.
- Write arithmetic expressions in Java.
- Evaluate arithmetic expressions using the precedence rules.
- Describe how the memory allocation works for objects and primitive data values.
- Write mathematical expressions, using methods in the Math class.
- Use the GregorianCalendar class in manipulating date information such as year, month, and day.
- Use the DecimalFormat class to format numerical data
- Convert input string values to numerical data
- Perform input and output by using System.in and System.out

I. Introduction

- Major Callenge: C = C + 1

II. Types of Primitive Data Values

Manipulating Numbers

- In Java, to add two numbers x and y, we write

```
x + y
```

- But before the actual addition of the two numbers takes place, we must declare their data type. If x and y are integers, we write

```
int x, y;
```

or

```
int x;
int y;
```


II.1 Variables

Variables

- When the declaration is made, memory space is allocated to store the values of x and y.
- x and y are called *variables*. A variable has three properties:
  - A memory location to store the value,
  - The type of data stored in the memory location, and
  - The name used to refer to the memory location.
- Sample variable declarations:

```
int x;
int v, w, y;
```


Numerical Data Types

- There are six numerical data types: byte, short, int, long, float, and double.
- Sample variable declarations:

```
int     i, j, k;
float   numberOne, numberTwo;
long    bigInteger;
double  bigNumber;
```

- At the time a variable is declared, it also can be initialized. For example, we may initialize the integer variables count and height to 10 and 34 as

```
int count = 10, height = 34;
```


Data Type Precisions

The six data types differ in the precision of values they can store in memory.

Data Type	Content	Default Value†	Minimum Value	Maximum Value
byte	Integer	0	−128	127
short	Integer	0	−32768	32767
int	Integer	0	−2147483648	2147483647
long	Integer	0	−9223372036854775808	9223372036854775807
float	Real	0.0	−3.40282347E+38‡	3.40282347E+38
double	Real	0.0	−1.79769313486231570E+308	1.79769313486231570E+308

II.2 Constant

Constants

- We can change the value of a variable. If we want the value to remain the same, we use a *constant*.

```
final double PI               = 3.14159;
final int    MONTH_IN_YEAR    = 12;
final short  FARADAY_CONSTANT = 23060;
```

The reserved word **final** is used to declare constants.

These are constants, also called *named constant*.

These are called *literal constant*.

II.3 Primitive vs Reference

Primitive vs. Reference

- Numerical data are called *primitive data types*.
- Objects are called *reference data types*, because the contents are addresses that refer to memory locations where the objects are actually stored.

Primitive Data Declaration and Assignments

```
int firstNumber, secondNumber;
firstNumber  = 234;
secondNumber = 87;
```

A

```
int firstNumber, secondNumber;
firstNumber  = 234;
secondNumber = 87;
```

B

A. Variables are allocated in memory.

firstNumber 234

secondNumber 87

B. Values are assigned to variables.

Code **State of Memory**

Assigning Numerical Data

```
int number;
number = 237;
number = 35;
```

```
int number;
number = 237;
number = 35;
```

A B C

number 35

A. The variable is allocated in memory.

B. The value 237 is assigned to number.

C. The value 35 overwrites the previous value 237.

Code **State of Memory**

Assigning Objects

```
Customer customer;
customer = new Customer( );
customer = new Customer( );
```

```
Customer customer;
customer = new Customer( );
customer = new Customer( );
```

A B C

customer

Customer Customer

A. The variable is allocated in memory.

B. The reference to the new object is assigned to customer.

C. The reference to another object overwrites the reference in customer.

Code **State of Memory**

Having Two References to a Single Object

```
Customer clemens, twain;
clemens = new Customer( );
twain   = clemens;
```

```
Customer clemens, twain;
clemens = new Customer( );
twain   = clemens;
```

A B C

clemens

twain

Customer

A. Variables are allocated in memory.

B. The reference to the new object is assigned to clemens.

C. The reference in clemens is assigned to customer.

Code **State of Memory**

III. Arithmetic Expression

Assignment Statements

- We assign a value to a variable using an *assignment statements*.
- The syntax is

```
<variable> = <expression> ;
```

- Examples:

```
sum = firstNumber + secondNumber;
avg = (one + two + three) / 3.0;
```


Arithmetic Operators

- The following table summarizes the arithmetic operators available in Java.

Operation	Java Operator	Example	Value (x = 10, y = 7, z = 2.5)
Addition	+	x + y	17
Subtraction	-	x - y	3
Multiplication	*	x * y	70
Division	/	x / y	1
		x / z	4.0
Modulo division (remainder)	%	x % y	3

This is an integer division where the fractional part is truncated.

Arithmetic Expression

- How does the expression

```
x + 3 * y
```

  get evaluated? Answer: x is added to 3*y.
- We determine the order of evaluation by following the *precedence rules*.
- A higher precedence operator is evaluated before the lower one. If two operators are the same precedence, then they are evaluated left to right for most operators.

Precedence Rules

Order	Group	Operator	Rule
High	Subexpression	( )	Subexpressions are evaluated first. If parentheses are nested, the innermost subexpression is evaluated first. If two or more pairs of parentheses are on the same level, then they are evaluated from left to right.
	Unary operator	-, +	Unary minuses and pluses are evaluated second.
	Multiplicative operator	*, /, %	Multiplicative operators are evaluated third. If two or more multiplicative operators are in an expression, then they are evaluated from left to right.
Low	Additive operator	+, -	Additive operators are evaluated last. If two or more additive operators are in an expression, then they are evaluated from left to right.

Type Casting

- If **x** is a **float** and **y** is an **int**, what will be the data type of the following expression?

```
x * y
```

  The answer is **float**.
- The above expression is called a *mixed expression*.
- The data types of the operands in mixed expressions are converted based on the *promotion rules*. The promotion rules ensure that the data type of the expression will be the same as the data type of an operand whose type has the highest precision.

Explicit Type Casting

- Instead of relying on the promotion rules, we can make an explicit type cast by prefixing the operand with the data type using the following syntax:

```
( <data type> ) <expression>
```

- Example

```
(float) x / 3
```

Type case **x** to **float** and then divide it by 3.

```
(int) (x / y * 3.0)
```

Type cast the result of the expression **x / y * 3.0** to **int**.

Implicit Type Casting

- Consider the following expression:

```
double x = 3 + 5;
```

- The result of 3 + 5 is of type **int**. However, since the variable **x** is **double**, the value 8 (type **int**) is promoted to 8.0 (type **double**) before being assigned to **x**.
- Notice that it is a promotion. Demotion is not allowed.

```
int x = 3.5;
```

A higher precision value cannot be assigned to a lower precision variable.

Type Mismatch

- Suppose we want to input an age. Will this work?

```
int     age;

age = JOptionPane.showInputDialog(
            null, "Enter your age");
```

- No.
  String value cannot be assigned directly to an int variable.

Type Conversion

- *Wrapper classes* are used to perform necessary type conversions, such as converting a String object to a numerical value.

```
int     age;
String  inputStr;

inputStr = JOptionPane.showInputDialog(
             null, "Enter your age");

age = Integer.parseInt(inputStr);
```


Other Conversion Methods

Class	Method	Example
Integer	parseInt	Integer.parseInt('25') → 25 Integer.parseInt('25.3') → error
Long	parseLong	Long.parseLong('25') → 25L Long.parseLong('25.3') → error
Float	parseFloat	Float.parseFloat('25.3') → 25.3F Float.parseFloat('ab3') → error
Double	parseDouble	Double.parseDouble('25') → 25.0 Double.parseDouble('ab3') → error

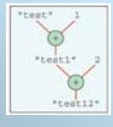
Sample Code Fragment

```
//code fragment to input radius and output
//area and circumference
double radius, area, circumference;

   radiusStr = JOptionPane.showInputDialog(
                         null, "Enter radius: " );

   radius = Double.parseDouble(radiusStr);

   //compute area and circumference
   area = PI * radius * radius;
   circumference = 2.0 * PI * radius;

   JOptionPane.showMessageDialog(null,
             "Given Radius:   " + radius + "\n" +
             "Area:           " + area   + "\n" +
             "Circumference:  " + circumference);
```

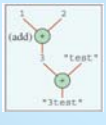

Overloaded Operator +

- The plus operator + can mean two different operations, depending on the context.
- <val1> + <val2> is an addition if both are numbers. If either one of them is a String, the it is a concatenation.
- Evaluation goes from left to right.

output = "test" + 1 + 2;



output = 1 + 2 + "test";



IV. Useful Classes

The DecimalFormat Class

- Use a DecimalFormat object to format the numerical output.

```
double num = 123.45789345;

DecimalFormat df = new DecimalFormat("0.000");
            //three decimal places
```

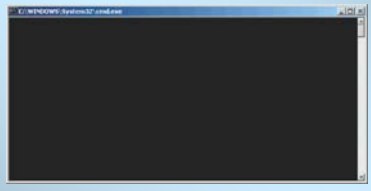
```
System.out.print(num);                    → 123.45789345

System.out.print(df.format(num));
                                          → 123.458
```


Standard Output

- The **showMessageDialog** method is intended for displaying short one-line messages, not for a general-purpose output mechanism.
- Using **System.out,** we can output multiple lines of text to the standard output window.

Standard Output Window

- A sample standard output window for displaying multiple lines of text.

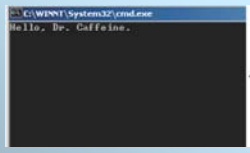


- The exact style of standard output window depends on the Java tool you use.

The print Method

- We use the **print** method to output a value to the standard output window.
- The **print** method will continue printing from the end of the currently displayed output.
- Example

```
System.out.print( "Hello, Dr. Caffeine." );
```

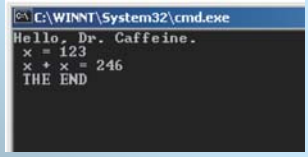


Note
Depending on the tool you use, you may see additional text such as
Press any key to continue...
or something similar to it. We will ignore any text that may be displayed automatically by the system.

The println Method

- We use **println** instead of **print** to skip a line.

```
int x = 123, y = x + x;
System.out.println( "Hello, Dr. Caffeine." );
System.out.print( " x = " );
System.out.println( x );
System.out.print( " x + x = " );
System.out.println( y );
System.out.println( " THE END" );
```



Standard Input

- The technique of using System.in to input data is called standard input.
- We can only input a single byte using System.in directly.
- To input primitive data values, we use the Scanner class (from Java 5.0).

```
Scanner scanner;

scanner = Scanner.create(System.in);

int num = scanner.nextInt();
```


Common Scanner Methods:

Method	Example
nextByte( )	byte b = scanner.nextByte( );
nextDouble( )	double d = scanner.nextDouble( );
nextFloat( )	float f = scanner.nextFloat( );
nextInt( )	int i = scanner.nextInt( );
nextLong( )	long l = scanner.nextLong( );
nextShort( )	short s = scanner.nextShort( );
next()	String str = scanner.next();

The Math class

- The **Math** class in the **java.lang** package contains class methods for commonly used mathematical functions.

```
double     num, x, y;

x = …;
y = …;

num = Math.sqrt(Math.max(x, y) + 12.4);
```

- Table 3.6 in the textbook contains a list of class methods defined in the **Math** class.

Some Math Class Methods

Method	Description
exp(a)	Natural number **e** raised to the power of **a**.
log(a)	Natural logarithm (base **e**) of **a**.
floor(a)	The largest whole number less than or equal to **a**.
max(a,b)	The larger of a and b.
pow(a,b)	The number **a** raised to the power of **b**.
sqrt(a)	The square root of **a**.
sin(a)	The sine of **a**. (Note: all trigonometric functions are computed in radians)

Table 3.8 page 113 in the textbook contains a list of class methods defined in the **Math** class.

Computing the Height of a Pole

$$h = \frac{d \sin\alpha \sin\beta}{\sqrt{\sin(\alpha+\beta)\sin(\alpha-\beta)}}$$

```
alphaRad = Math.toRadians(alpha);
betaRad  = Math.toRadians(beta);

height = ( distance * Math.sin(alphaRad) * Math.sin(betaRad) )
                    /
          Math.sqrt( Math.sin(alphaRad + betaRad) *
                     Math.sin(alphaRad - betaRad) );
```


The GregorianCalendar Class

- Use a GregorianCalendar object to manipulate calendar information

```
GregorianCalendar today, independenceDay;

today     = new GregorianCalendar();

independenceDay
          = new GregorianCalendar(1776, 6, 4);
             //month 6 means July; 0 means January
```

Retrieving Calendar Information

- This table shows the class constants for retrieving different pieces of calendar information from Date.

Constant	Description
YEAR	The year portion of the calendar date
MONTH	The month portion of the calendar date
DATE	The day of the month
DAY_OF_MONTH	Same as DATE
DAY_OF_YEAR	The day number within the year
DAY_OF_MONTH	The day number within the month
DAY_OF_WEEK	The day of the week (Sun — 1, Mon — 2, etc.)
WEEK_OF_YEAR	The week number within the year
WEEK_OF_MONTH	The week number within the month
AM_PM	The indicator for AM or PM (AM — 0 and PM — 1)
HOUR	The hour in 12-hour notation
HOUR_OF_DAY	The hour in 24-hour notation
MINUTE	The minute within the hour

Sample Calendar Retrieval

```
GregorianCalendar cal = new GregorianCalendar();
            //Assume today is Nov 9, 2003

System.out.print("Today is " +
      (cal.get(Calendar.MONTH)+1) + "/" +
      cal.get(Calendar.DATE) + "/" +
      cal.get(Calendar.YEAR));
```

Output

```
Today is 11/9/2003
```


V. Sample Development

Problem Statement

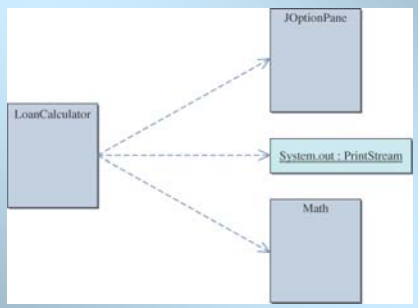
- Problem statement:

  Write a loan calculator program that computes both monthly and total payments for a given loan amount, annual interest rate, and loan period.

Overall Plan

- Tasks:
  - Get three input values: **loanAmount**, **interestRate**, and **loanPeriod**.
  - Compute the monthly and total payments.
  - Output the results.

Required Classes



Development Steps

- We will develop this program in four steps:

1. Start with code to accept three input values.
2. Add code to output the results.
3. Add code to compute the monthly and total payments.
4. Update or modify code and tie up any loose ends.

Step 1 Design

- Call the **showInputDialog** method to accept three input values:
  - loan amount,
  - annual interest rate,
  - loan period.
- Data types are

Input	Format	Data Type
loan amount	dollars and cents	double
annual interest rate	in percent (e.g.,12.5)	double
loan period	in years	int

Step 1 Code

Program source file is too big to list here. From now on, we ask you to view the source files using your Java IDE.

Directory: Chapter3/Step1

Source File: Ch3LoanCalculator.java

Step 1 Test

- In the testing phase, we run the program multiple times and verify that
  - we can enter three input values
  - we see the entered values echo-printed correctly on the standard output window

Step 2 Design

- We will consider the display format for out.
- Two possibilities are (among many others)

Only the computed values (and their labels) are shown

```
Monthly payment:        $ 143.47
Total payment:          $ 17216.50
```

Both the input and computed values (and their labels) are shown.

```
For
Loan Amount:            $ 10000.00
Annual Interest Rate:     12.0%
Loan Period (years):      10

Monthly payment is      $ 143.47
  TOTAL payment is      $ 17216.50
```


Step 2 Code

Directory: Chapter3/Step2

Source File: Ch3LoanCalculator.java

Step 2 Test

- We run the program numerous times with different types of input values and check the output display format.
- Adjust the formatting as appropriate

Step 3 Design

- The formula to compute the geometric progression is the one we can use to compute the monthly payment.
- The formula requires the loan period in months and interest rate as monthly interest rate.
- So we must convert the annual interest rate (input value) to a monthly interest rate (per the formula), and the loan period to the number of monthly payments.

Step 3 Code

Directory: Chapter3/Step3

Source File: Ch3LoanCalculator.java

Step 3 Test

- We run the program numerous times with different types of input values and check the results.

Input			Output (shown up to three decimal places only)	
Loan Amount	Annual Interest Rate	Loan Period (in years)	Monthly Payment	Total Payment
10000	10	10	132.151	15858.088
15000	7	15	134.824	24268.363
10000	12	10	143.471	17216.514
0	10	5	0.000	0.000
30	8.5	50	0.216	129.373

Step 4: Finalize

- We will add a program description
- We will format the monthly and total payments to two decimal places using DecimalFormat.

Directory: Chapter3/Step4

Source File: Ch3LoanCalculator.java