

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Chapter 3 - 1

Chapter 3

Numerical Data



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.
- Generate pseudo random numbers.
- Use the `GregorianCalendar` class in manipulating date information such as year, month, and day.
- Use the `DecimalFormat` class to format numerical data
- Input and output numerical data by using `System.in` and `System.out`

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Chapter 3 - 2



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;
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Chapter 3 - 3



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;
```

©The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Chapter 3 - 4



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



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;
```



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; ← A
number = 237; ← B
number = 35; ← C
```

Code

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.

State of Memory

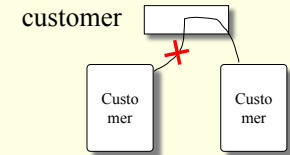


Assigning Objects

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

```
Customer customer; ← A
customer = new Customer(); ← B
customer = new Customer(); ← C
```

Code



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**.

State of Memory

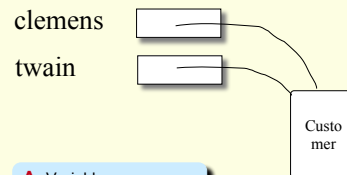


Having Two References to a Single Object

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

```
Customer clemens, twain; ← A
clemens = new Customer(); ← B
twain = clemens; ← C
```

Code



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 **twain**.

State of Memory



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.



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 ↑ ↓ Low	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.
	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.



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**.



Displaying Numerical Values

- In Chapter 2, we showed how to output text (String) to the standard output
- We use the same **print** and **println** methods to output numerical data to the standard output.

```
int num = 15;
System.out.print(num); //print a variable
System.out.print(" "); //print a string
System.out.print(10); //print a constant
```

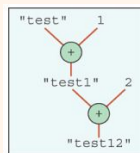
```
15 10
```



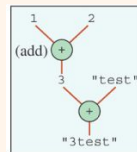
Overloaded Operator +

- 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";



Sample Code Fragment

```

//code fragment to input radius and output
//area and circumference
final double PI = 3.14159;

double radius, area, circumference;

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

System.out.println("Given Radius: " + radius);
System.out.println("Area: " + area);
System.out.println("Circumference: " + circumference);

```



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

```



Getting Numerical Input

- In Chapter 2, we learned how to input strings using the Scanner class.
- We can use the same Scanner class to input numerical values

```

Scanner scanner = new Scanner(System.in);
int age;
System.out.print("Enter your age: ");
age = scanner.nextInt();

```



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);
```



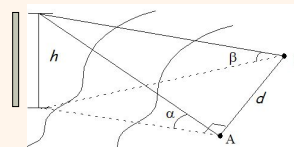
Some Math Class Methods

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

Table 3.7 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)
                   );
```



Random Number Generation

- We can use the `nextInt(n)` method of the `Random` class to generate a random number between 0 and `n-1`, inclusive.

```
import java.util.Random;
...
Random random = new Random();
...
int number = random.nextInt(11); //return x, 0 <= x <= 10
```

- To return a random integer in `[min, max]` inclusively, where `min <= max`

```
...
int number = random.nextInt(max - min + 1) + min;
```



Random Number Generation - 2

- The `Math.random` method is called a pseudo random number generator and returns a number (double) `X`, where $0.0 \leq X < 1.0$
- To return a pseudo random integer in `[min, max]` inclusively, where `min <= max`, use the formula

$$\lfloor X \times (\text{max} - \text{min} + 1) \rfloor + \text{min}$$

```
int randomNumber = (int) (Math.floor(Math.random()
    * (max - min + 1)) + min);
```



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 Dec 18, 2008  
  
System.out.print("Today is " +  
    (cal.get(Calendar.MONTH)+1) + "/" +  
    cal.get(Calendar.DATE) + "/" +  
    cal.get(Calendar.YEAR));
```

Output

Today is 12/18/2008



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.



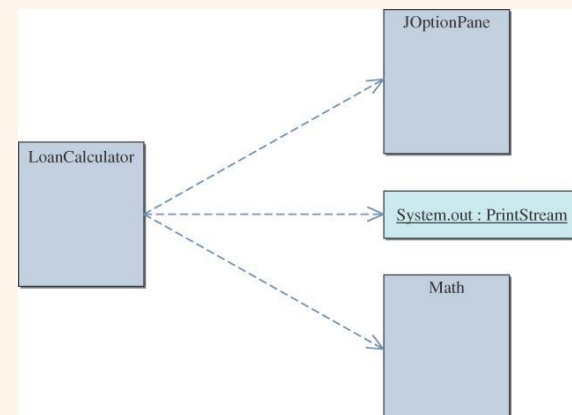
Overall Plan

- **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	<table> <tr> <td>Monthly payment:</td><td>\$ 143.47</td></tr> <tr> <td>Total payment:</td><td>\$ 17216.50</td></tr> </table>	Monthly payment:	\$ 143.47	Total payment:	\$ 17216.50								
Monthly payment:	\$ 143.47												
Total payment:	\$ 17216.50												
Both the input and computed values (and their labels) are shown.	<table> <tr> <td>For</td><td></td></tr> <tr> <td>Loan Amount:</td><td>\$ 10000.00</td></tr> <tr> <td>Annual Interest Rate:</td><td>12.0%</td></tr> <tr> <td>Loan Period (years):</td><td>10</td></tr> <tr> <td>Monthly payment is</td><td>\$ 143.47</td></tr> <tr> <td>TOTAL payment is</td><td>\$ 17216.50</td></tr> </table>	For		Loan Amount:	\$ 10000.00	Annual Interest Rate:	12.0%	Loan Period (years):	10	Monthly payment is	\$ 143.47	TOTAL payment is	\$ 17216.50
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