Motivations

► In the preceding chapter, you learned how to create, compile, and run a Java program.

► Starting from chapter 2, you will learn how to solve practical problems programmatically.

► Through these problems, you will learn Java primitive data types and related subjects, such as variables, constants, data types, operators, expressions, and input and output.

Chapter 2
Elementary Programming

Objectives

► Use identifiers to name variables, constants, methods, and classes

► Use variables to store data

► Assignment statements

► Use constants to store permanent data

► Declare Java primitive data types: <u>byte</u>, <u>short</u>, <u>int</u>, <u>long</u>, <u>float</u>, <u>double</u>, and <u>char</u>

► Represent characters using the <u>char</u> type

► Become familiar with Java documentation, programming style, and naming conventions

Introducing Programming with an Example

► Computing the Area of a Circle

► This program computes the area of the circle.

3

Trace a Program Execution

```java
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius" +
      radius + " is " + area);
  }
}
```
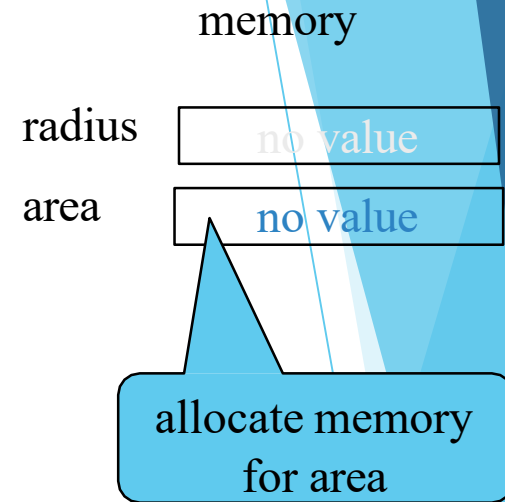
allocate memory for radius

radius | no value

Trace a Program Execution

```java
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius
      +
      radius + " is " + area);
  }
}
```

memory

radius | no value

area | no value

allocate memory for area

Trace a Program Execution

```java
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius "
      +
      radius + " is " + area);
  }
}
```

assign 20 to radius

radius   20

area   no value

Trace a Program Execution

```java
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius
      +
      radius + " is " + area);
  }
}
```

memory

radius [20]

area [1256.636]

compute area and assign it to variable area

Trace a Program Execution

```java
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```
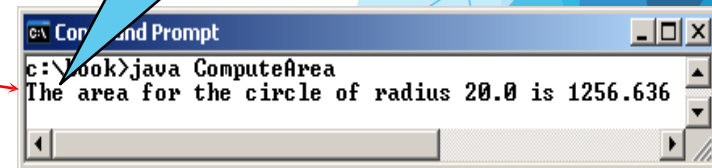
memory

radius   20

area   1256.636

print a message to the console

```
c:\book>java ComputeArea
The area for the circle of radius 20.0 is 1256.636
```

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, "Java Keywords," for a list of reserved words).

- An identifier cannot be `true`, `false`, or `null`.

- An identifier can be of any length.

Variables

```java
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is " +
  area + " for radius "+radius);

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is " +
  area + " for radius "+radius);
```

Declaring Variables

```
int x;              // Declare x to be an
                    // integer variable;
double radius;      // Declare radius to
                    // be a double variable;
char a;             // Declare a to be a
                    // character variable;
```

Assignment Statements

```
x = 1;              // Assign 1 to x;

radius = 1.0;       // Assign 1.0 to radius;

a = 'A';            // Assign 'A' to a;
```

Declaring and Initializing in One Step

► ```
int x = 1;
```

► ```
double d = 1.4;
```

13

Constants

```
final datatype CONSTANTNAME = VALUE;


final double PI = 3.14159;
final int SIZE = 3;
```

Naming Conventions

▶ Choose meaningful and descriptive names.

▶ Variables and method names:

  ▶ Use lowercase. 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. For example, the variables `radius` and `area`, and the method `computeArea`.

15

Naming Conventions, cont.

► Class names:

   ► Capitalize the first letter of each word in the name. For example, the class name `ComputeArea`.

► Constants:

   ► Capitalize all letters in constants, and use underscores to connect words. For example, the constant `PI and MAX_VALUE`

Numerical Data Types

Name	Range	Storage Size
byte	-2^7 (-128) to 2^7-1 (127)	8-bit signed
short	-2^{15} (-32768) to $2^{15}-1$ (32767)	16-bit signed
int	-2^{31} (-2147483648) to $2^{31}-1$ (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

Number Literals

A *literal* is a constant value that appears directly in the program. For example, 34, 1000000, and 5.0 are literals in the following statements:

int i = 34;

long x = 1000000;

double d = 5.0;

Integer Literals

▶ 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.

▶ For example, the statement <u>byte b = 1000</u> would cause a compilation error, because 1000 cannot be stored in a variable of the <u>byte</u> type.

▶ To denote an integer literal of the <u>long</u> type, append it with the letter <u>L</u> or <u>l</u>. L is preferred because l (lowercase L) can easily be confused with 1 (the digit one).

Character Data Type

Four hexadecimal digits.

char letter = 'A'; (ASCII)

char numChar = '4'; (ASCII)

char letter = '\u0041'; (Unicode)

char numChar = '\u0034'; (Unicode)

Unicode Format

Java characters use *Unicode*, a 16-bit encoding scheme established by the Unicode Consortium to support the interchange, processing, and display of written texts in the world's diverse languages. Unicode takes two bytes, preceded by \u, expressed in four hexadecimal numbers that run from '\u0000' to '\uFFFF'. Unicode \u03b1 \u03b2 \u03b3 for three Greek letters

22
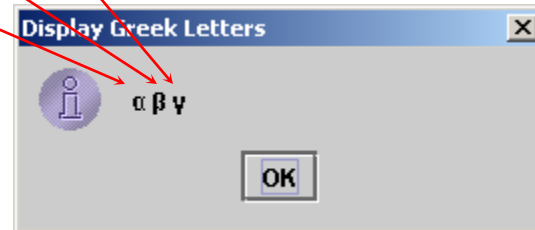
Escape Sequences for Special Characters

Description	Escape Sequence	Unicode
Backspace	\b	\u0008
Tab	\t	\u0009
Linefeed	\n	\u000A
Carriage return	\r	\u000D
Backslash	\\	\u005C
Single Quote	\'	\u0027
Double Quote	\"	\u0022

ASCII Character Set, cont.

ASCII Character Set is a subset of the Unicode from \u0000 to \u007f

TABLE B.2   ASCII Character Set in the Hexadecimal Index

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht	nl	vt	ff	cr	so
1	dle	dcl	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs
2	sp	!	"	#	$	%	&	'	(	)	*	+	,	-	.
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n
7	p	q	r	s	t	u	v	w	x	y	z	{			}	