Key point: *Java has six numeric types for integers and floating-point numbers with operators +, -, *, /, and %.*

2.9 Numeric Data Types and Operations

2.9.1 Numeric Types

Table 2.1 Numeric Data Types

Name	*Range*	*Storage Size*	
byte	$-2\text{^}7$to $2\text{^}7-1(-128$ to $127)$	8-bit signed	byte type
short	$-2\text{^}15$to $2\text{^}15-1(-32768$ to $32767)$	16-bit signed	short type
int	$-2\text{^}31$to $2\text{^}31-1(-2147483648$ to $2147483647)$	32-bit signed	int type
long	$-2\text{^}63$to $2\text{^}63-1$ (-9223372036854775808 to 9223372036854775807)	64-bit signed	long type
float	Negative range: $-[\![3.4028235]\!]\text{^}38$ to $[\![-1.4]\!]\text{^}(-45)$ Positive range: $[\![1.4]\!]\text{^}(-45)$ to $[\![3.4028235]\!]\text{^}38$	32-bit IEEE 754	float type
double	Negative range: $-[\![1.7976931348623157]\!]\text{^}308$ to $[\![-4.9]\!]\text{^}(-324)$ Positive range: $[\![4.9]\!]\text{^}(-324)$ to $[\![1.7976931348623157]\!]\text{^}308$	64-bit IEEE 754	double type

2.9.2 Reading Numbers from the Keyboard

Table 2.2 Methods for Scanner Objects

Method	*Description*
nextByte()	reads and integer of the **byte** type
nextShort()	reads and integer of the **short** type
nextInt()	reads and integer of the **int** type
nextLong()	reads and integer of the **long** type
nextFloat()	reads and integer of the **float** type
nextDouble()	reads and integer of the **double** type

```
1  Scanner input = new Scanner(System.in);
2  System.out.print("Enter a byte value: ");
3  byte byteValue = input.nextByte();
4
5  System.out.print("Enter a short value: ");
6  short shortValue = input.nextShort();
7
8  System.out.print("Enter a int value: ");
9  int intValue = input.nextInt();
10
11 System.out.print("Enter a long value: ");
12 long longValue = input.nextLong();
```

```
13
14  System.out.print("Enter a float value: ");
15  float floatValue = input.nextFloat();
```

2.9.3 Numeric Operators

Table 2.3 Numeric Operators

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 fractional portion of the quotient is truncated when integer division is employed.
  - Example 1: Both the 1 and the 3 are integer constants forcing the /-operator to be integer division
    $1 / 3 = 0.3333 = 0$
  - Example 2: The numerator 1.0 is a floating-point value forcing the /-operator to perform real division
    $1.0 / 3 = 0.3333$
- Modulo arithmetic. The %-operator finds the remainder. Both operands must be integers. The operand on the left is the numerator (dividend) and the operand on the right is the denominator (divisor).
  - Positive examples:
    - $7 \% 3 = 1$ // $7 \div 3 = 2$ r 1
    - $3 \% 7 = 3$ //$3 \div 7 = 0$ r 3
    - $22 \% 7 = 1$ //$22 \div 7 = 3$ r 1
  - Negative examples:
    - $7 \% 3 = 1$ // $7 \div 3 = 2$ r 1
    - $3 \% 7 = 3$ //$3 \div 7 = 0$ r 3
    - $22 \% 7 = 1$ //$22 \div 7 = 3$ r 1

Listing 2.5 DisplayTime.java

```
1  import java.util.Scanner;
2
3  public class DisplayTime {
4      public static void main(String[] args) {
5          Scanner input = new Scanner(System.in);
6          //Prompt the user for input
7          System.out.print("Enter an integer for seconds: ");
8          int seconds = input.nextInt();
9
10         int minutes = seconds / 60;              //Find minutes in seconds
11         int remainingSeconds = seconds % 60;     //Seconds remaining
12         System.out.println(seconds + " seconds is " + minutes +
13             " minutes and " + remainingSeconds + " seconds");
14     }
15 }
```

```
run:
Enter an integer for seconds: 500
500 seconds is 8 minutes and 20 seconds
BUILD SUCCESSFUL (total time: 8 seconds)
```

2.9.4 Exponent Operations

- The **Math.pow(a,b)** method can be used to compute a^b.
- Example: compute 4^3;
  Math.pow(4,3)
- 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
```