Chapter 2
Data and Expressions

Java Software Solutions
Foundations of Program Design
9th Edition

John Lewis
William Loftus

Copyright © 2017 Pearson Education, Inc.
Life is like riding a bicycle. To keep your balance, you must keep moving.
A. Einstein
Data and Expressions

• Let's explore some other fundamental programming concepts

• Chapter 2 focuses on:
  – character strings
  – primitive data
  – the declaration and use of variables
  – expressions and operator precedence
  – data conversions
  – accepting input from the user

Copyright © 2017 Pearson Education, Inc.
Character Strings

• A *string literal* is represented by putting double quotes around the text

• A *literal* is an explicit data value used in a program – e.g., 2, "pippo", 'C"

• Examples:
  
  "This is a string literal."
  
  "123 Main Street"
  
  "X"

• *Are these strings Java objects?*

• Every string literal is an object in Java, defined by the *String* class
The println Method

• In the Lincoln program from Chapter 1, we invoked the println method to print a character string.

• The System.out object represents a destination (the monitor screen) to which we can send output.

```java
System.out.println ("Whatever you are, be a good one.");
```

- **object**: System.out
- **method name**: println
- **information provided to the method (parameters)**: "Whatever you are, be a good one."
The print Method

• The System.out object provides another service as well

• The print method is similar to the println method, except that it does not advance to the next line

• Therefore anything printed after a print statement will appear on the same line

• See Countdown.java
public class Countdown {
    public static void main (String[] args) {
        System.out.print ("Three... ");
        System.out.print ("Two... ");
        System.out.print ("One... ");
        System.out.print ("Zero... ");
        System.out.println ("Liftoff!"); // appears on first output line
        System.out.println ("Houston, we have a problem.");
    }
}
```java
public class Countdown {
    // Prints two lines of output representing a rocket countdown.
    public static void main (String[] args) {
        System.out.print ("Three... ");
        System.out.print ("Two... ");
        System.out.print ("One... ");
        System.out.print ("Zero... ");
        System.out.println ("Liftoff!");  // appears on first output line
        System.out.println ("Houston, we have a problem.");
    }
}
```

**Output**

```
Three... Two... One... Zero... Liftoff!
Houston, we have a problem.
```
String Concatenation

• The string concatenation operator (+) is used to append one string to the end of another
  "Peanut butter " + "and jelly"

• It can also be used to append a number to a string

• A string literal cannot be broken across two lines in a program:

• The following generates a compiler error!
  System.out.println ("We present the following facts for your extracurricular edification:");

• See Facts.java
public class Facts {
    // Prints various facts.
    public static void main (String[] args) {
        // Strings can be concatenated into one long string
        System.out.println ("We present the following facts for your "
                           + "extracurricular edification:");

        System.out.println ("Letters in the Hawaiian alphabet: 12");

        continue
    }
}
// A numeric value can be concatenated to a string
System.out.println("Dialing code for Antarctica: " + 672);

System.out.println("Year in which Leonardo da Vinci invented "
                             + "the parachute: " + 1515);

System.out.println("Speed of ketchup: " + 40 + " km per year");
Output

We present the following facts for your extracurricular edification:

Letters in the Hawaiian alphabet: 12
Dialing code for Antarctica: 672
Year in which Leonardo da Vinci invented the parachute: 1515
Speed of ketchup: 40 km per year

System.out.println ("Speed of ketchup: " + 40 + " km per year");
}
String Concatenation

• The + operator is also used for arithmetic addition

• The function that it performs depends on the type of the information on which it operates

• If both operands are strings, or if one is a string and the other is a number, it performs string concatenation

• If both operands are numeric, it adds them

• The + operator is evaluated left to right, but parentheses can be used to force the order

• See Addition.java
// Addition.java       Author: Lewis/Loftus
//
// Demonstrates the difference between the addition and string
// concatenation operators.
//*******************************************************************************

public class Addition
{
    //-----------------------------------------------------------------------------
    // Concatenates and adds two numbers and prints the results.
    //-----------------------------------------------------------------------------
    public static void main (String[] args)
    {
        System.out.println ("24 and 45 concatenated: " + 24 + 45);
        System.out.println ("24 and 45 added: " + (24 + 45));
    }
}
public class Addition {
    public static void main (String[] args) {
        System.out.println ("24 and 45 concatenated: " + 24 + 45);
        System.out.println ("24 and 45 added: " + (24 + 45));
    }
}
Quick Check

What output is produced by the following?

```java
System.out.println("X: "+25);
System.out.println("Y: "+(15+50));
System.out.println("Z: "+300+50);
```
Quick Check

What output is produced by the following?

System.out.println ("X: " + 25);
System.out.println ("Y: " + (15 + 50));
System.out.println ("Z: " + 300 + 50);

X: 25
Y: 65
Z: 30050
Quiz *

• A string literal, for instance, "John" is an object.
• Hence, is it possible to invoke a method of the object "john"?

"John".concat(" is lazy!");

"*" means that is a difficult exercise – beginners should not be worried if they do not catch it 😊
Quiz

• A string literal, for instance, "John" is an object.
• Hence, is it possible to invoke a method of "john" object?

"John".concat(" is lazy!");

• YES, so for instance:
System.out.println("John".concat(" is lazy!"));

Will print:
> John is lazy!
Escape Sequences

• What if we wanted to print the quote character?

• The following line would confuse the compiler because it would interpret the second quote as the end of the string

  System.out.println ("I said "Hello" to you.");

• An escape sequence is a series of characters that represents a special character

• An escape sequence begins with a backslash character (\)

  System.out.println ("I said \"Hello\" to you.");
Escape Sequences

• Some Java escape sequences:

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>backspace</td>
</tr>
<tr>
<td>\t</td>
<td>tab</td>
</tr>
<tr>
<td>\n</td>
<td>newline</td>
</tr>
<tr>
<td>\r</td>
<td>carriage return</td>
</tr>
<tr>
<td>&quot;</td>
<td>double quote</td>
</tr>
<tr>
<td>'</td>
<td>single quote</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
</tr>
</tbody>
</table>

• See Roses.java
//********************************************************************
//  Roses.java       Author: Lewis/Loftus
//
//  Demonstrates the use of escape sequences.
//********************************************************************

public class Roses
{
  //----------------------------------------------------------------------
  //  Prints a poem (of sorts) on multiple lines.
  //----------------------------------------------------------------------
  public static void main (String[] args)
  {
    System.out.println ("Roses are red,\nViolets are blue,\n" +
                        "Sugar is sweet,\nBut I have " +
                        "commitment issues",\n                        "So I'd rather just be friends\nAt this point in our " +
                        "relationship.");
  }
}
public class Roses {
    public static void main (String[] args) {
        System.out.println ("Roses are red,
                            Violets are blue,
                            Sugar is sweet,
                            But I have "commitment issues",
                            So I'd rather just be friends
                            At this point in our relationship.");
    }
}
Quick Check

Write a single `println` statement that produces the following output:

"Thank you all for coming to my home tonight," he said mysteriously.
Quick Check

Write a single `println` statement that produces the following output:

"Thank you all for coming to my home tonight," he said mysteriously.

```java
System.out.println("""Thank you all for " + "coming to my home\ntonight," he said " + "mysteriously."');
```
Outline

Character Strings
Variables and Assignment
Primitive Data Types
Expressions
Data Conversion
Interactive Programs
Buttered Toast – with Variables

- **LET A** = a loaf of bread
- **LET T** = a toaster
- **LET P** = a plate
- **LET B** = some butter
- **LET S** = *cut slice from A*

get A

- move the S to T

turn on T

wait for T

- move the S to P

spread S with B
Variables

• A **variable** is a name for a location in memory that holds a value

• A **variable declaration** specifies the variable's name and the **type** of information that it will hold

```
int total;
int count, temp, result;
```

Multiple variables can be created in one declaration

After declaration the type of a variable cannot be changed!
A variable must be declared only ONCE!
Variable Initialization

• A variable can be given an **initial value** in the declaration

```java
int sum = 0;
int base = 32, max = 149;
```

• When a variable is referenced in a program, its current value is used

• **See** PianoKeys.java
public class PianoKeys
{
    public static void main (String[] args)
    {
        int keys = 88;
        System.out.println("A piano has " + keys + " keys.");
    }
}
public class PianoKeys
{
    // Prints the number of keys on a piano.
    public static void main (String[] args)
    {
        int keys = 88;
        System.out.println ("A piano has " + keys + " keys.");
    }
}
Assignment

• An *assignment statement* changes the value of a variable – *it assigns a value to a variable*

• The assignment operator is the '*=' sign

```java
    total = 55;
```

• The value that was in `total` is overwritten

• You can only assign a value to a variable that is consistent with the variable's declared type

• See `Geometry.java`
public class Geometry
{
    public static void main (String[] args)
    {
        int sides = 7; // declaration with initialization
        System.out.println ("A heptagon has " + sides + " sides.");

        sides = 10; // assignment statement
        System.out.println ("A decagon has " + sides + " sides.");

        sides = 12;
        System.out.println ("A dodecagon has " + sides + " sides.");
    }
}
public class Geometry
{
    public static void main (String[] args)
    {
        int sides = 7; // declaration with initialization
        System.out.println ("A heptagon has " + sides + " sides.");

        sides = 10; // assignment statement
        System.out.println ("A decagon has " + sides + " sides.");

        sides = 12;
        System.out.println ("A dodecagon has " + sides + " sides.");
    }
}
Constants

• A constant is an identifier that is similar to a variable except that it holds the same value during its entire existence

• As the name implies, it is constant, not variable

• The compiler will issue an error if you try to change the value of a constant

• In Java, we use the final modifier to declare a constant

    final int MIN_HEIGHT = 69;
Constants

- Constants are useful for three important reasons
- First, they give meaning to otherwise unclear literal values
  - Example: `MAX_LOAD` means more than the literal 250
- Second, they facilitate program maintenance
  - If a constant is used in multiple places, its value need only be set in one place
- Third, they formally establish that a value should not change, avoiding inadvertent errors by other programmers
Primitive Data

• There are **eight** primitive data types in Java

• Four of them represent integers:
  – byte, short, int, long

• Two of them represent floating point numbers:
  – float, double

• One of them represents characters:
  – char

• And one of them represents boolean values:
  – boolean
Numeric Primitive Data

- The difference between the numeric primitive types is their size and the values they can store:

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>&lt; -9 x 10^{18}</td>
<td>&gt; 9 x 10^{18}</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>+/- 3.4 x 10^{38} with 7 significant digits</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>+/- 1.7 x 10^{308} with 15 significant digits</td>
<td></td>
</tr>
</tbody>
</table>
Characters

• A char variable stores a single character.

• Character literals are delimited by single quotes:
  'a'   'X'    '7'    '$'    ','    '

• Example declarations:
  char topGrade = 'A';
  char terminator = ';', separator = ' ';

• Note the difference between a primitive character
  variable, which holds only one character, and a
  String object, which can hold multiple characters.
Character Sets

- A *character set* is an ordered list of characters, with each character corresponding to a unique number.

- A `char` variable in Java can store any character from the *Unicode character set*.

- The Unicode character set uses *sixteen bits* per character, allowing for 65,536 unique characters.

- It is an international character set, containing symbols and characters from many world languages.
Characters

- The ASCII character set is older and smaller than Unicode, but is still quite popular.
- The ASCII characters are a subset of the Unicode character set, including:

  - uppercase letters: A, B, C, …
  - lowercase letters: a, b, c, …
  - punctuation: period, semi-colon, …
  - digits: 0, 1, 2, …
  - special symbols: &, |, \, …
  - control characters: carriage return, tab, …
Characters

char c = '\u005B'; //assign to c the
    //character with that Unicode value
System.out.println(c);
char d = 91; //assign to d the
    //character with that Unicode value
System.out.println(d);

What is printing? Look at the Unicode table for the corresponding character.
Convert the hexadecimal number 005B in decimal format.

Unicode table
Characters

```java
char c = '\u005B';
System.out.println(c);
char d = 91;
System.out.println(d);
```

What is printing? Look at the unicode table. Convert the hexadecimal number 005B in decimal format.
Boolean

• A boolean value represents a true or false condition

• The reserved words true and false are the only valid values for a boolean type

        boolean done = false;

• A boolean variable can also be used to represent any two states, such as a light bulb being on or off
Expressions

• An expression is a combination of one or more operators and operands that produces a result

• Arithmetic expressions compute numeric results and make use of the arithmetic operators:
  
  Addition +
  Subtraction -
  Multiplication *
  Division /
  Remainder %

• If either or both operands are floating point values, then the result is a floating point value
Division and Remainder

• If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

\[
14 \div 3 \quad \text{equals} \quad 4 \\
8 \div 12 \quad \text{equals} \quad 0
\]

• The remainder operator (%) returns the remainder after dividing the first operand by the second

\[
14 \mod 3 \quad \text{equals} \quad 2 \\
8 \mod 12 \quad \text{equals} \quad 8
\]
Quick Check

What are the results of the following expressions?

\[ 12 \div 2 \]
\[ 12.0 \div 2.0 \]
\[ 10 \div 4 \]
\[ 10 \div 4.0 \]
\[ 4 \div 10 \]
\[ 4.0 \div 10 \]
\[ 12 \mod 3 \]
\[ 10 \mod 3 \]
\[ 3 \mod 10 \]
Quick Check

What are the results of the following expressions?

\[
\begin{align*}
12 / 2 &= 6 \\
12.0 / 2.0 &= 6.0 \\
10 / 4 &= 2 \\
10 / 4.0 &= 2.5 \\
4 / 10 &= 0 \\
4.0 / 10 &= 0.4 \\
12 \mod 3 &= 0 \\
10 \mod 3 &= 1 \\
3 \mod 10 &= 3
\end{align*}
\]
Operator Precedence

• Operators can be combined into larger expressions

\[
\text{result} = \text{total} + \frac{\text{count}}{\text{max}} - \text{offset};
\]

• Operators have a well-defined precedence which determines the order in which they are evaluated

• Multiplication, division, and remainder are evaluated before addition, subtraction, and string concatenation

• Parentheses have a higher precedence than the other operators

• Arithmetic operators with the same precedence are evaluated from left to right, but parentheses can be used to force the evaluation order
Quick Check

In what order are the operators evaluated in the following expressions?

\[ a + b + c + d + e \quad a + b * c - d / e \]

\[ a / (b + c) - d \% e \]

\[ a / (b * (c + (d - e))) \]
Quick Check

In what order are the operators evaluated in the following expressions?

\[ a + b + c + d + e \]

1  2  3  4

\[ a + b \times c - d \div e \]

2  1  4  3

\[ a \div (b + c) - d \% e \]

2  1  4  3

\[ a \div (b \times (c + (d - e))) \]

4  3  2  1
Quick Check

• How this expression is evaluated:

1 + 2 + 3 * 4
Quick Check

• How this expression is evaluated:

\[ 1 + 2 + 3 \times 4 \]

1) \( 1 + 2 + 3 \times 4 \)
2) \( 3 + 3 \times 4 \)
3) \( 3 + 12 \)
Expression Trees

• The evaluation of a particular expression can be shown using an expression tree
• The operators lower in the tree have higher precedence for that expression

\[ a + \frac{(b - c)}{d} \]
Assignment Revisited

• The assignment operator has a **lower precedence** than the arithmetic operators.

First the expression on the right hand side of the `=` operator is evaluated:

\[
\text{answer} = \frac{\text{sum}}{4} + \text{MAX} \times \text{lowest};
\]

Then the result is stored in the variable on the left hand side.
Assignment Revisited

• The right and left hand sides of an assignment statement can contain the same variable

  First, one is added to the original value of count

  \[
  \text{count} = \text{count} + 1;
  \]

  Then the result is stored back into count (overwriting the original value)
Quiz

• What will be the results of the following assignment statement?
• Assume \( b=5, \ c=10 \)

\[
\text{int } a = b \times (-c + 2) / 2
\]
Quiz

• What will be the results of the following assignment statement?
• Assume b=5, c=10

```
int a = b * (-c + 2) / 2
```

-20
Quiz

• Assume that x, y, and z are all integers (int) equal to 50, 20, and 6 respectively.
• What is the result of: x / y / z

a) 0  
b) 0.4166  
c) 15  
d) A syntax error since it is syntactically invalid  
e) A run-time error because this is a division by 0
Quiz

• Assume that x, y, and z are all integers (int) equal to 50, 20, and 6 respectively.
• What is the result of: $x \div y \div z$

a) 0 $\leftarrow$ division is performed left to right and $50/20 = 2, 2/6 = 0$

b) 0.4166

c) 15

d) A syntax error since it is syntactically invalid

e) A run-time error because this is a division by 0
Increment and Decrement

- The increment (++) and decrement (--) operators use only one operand
- The statement
  \[
  \text{count}++; \\
  \]
  is functionally equivalent to
  \[
  \text{count} = \text{count} + 1; \\
  \]
Increment and Decrement

• The increment and decrement operators can be applied in *postfix form*:
  
  \[ \text{count}++ \]

• or *prefix form*:
  
  \[ ++\text{count} \]

• When used as part of a larger expression, the two forms can have different effects

• Because of their subtleties, the increment and decrement operators should be used with care
Quick Check

What is printed?

```java
int count = 3, value = 0;
value = count++;
System.out.println(value);
System.out.println(count);
value = ++count;
System.out.println(value);
System.out.println(count);
```
Quick Check

What is printed?

```java
int count = 3, value = 0;
value = count++;
System.out.println(value);
System.out.println(count);
value = ++count;
System.out.println(value);
System.out.println(count);
3
4
5
5
```
Quick Check

• What is printed?

```java
int a = 0;
a = a++ + 2;
System.out.println(a);
a = 0;
a = ++a + 2;
System.out.println(a);
```
Quick Check

• What is printed?

```java
int a = 0;
a = a++ + 2;
System.out.println(a);
a = 0;
a = ++a + 2;
System.out.println(a);
```

2

3
Quick Check

• What is printed?

```java
int yy = 3;
System.out.println(++yy + yy*yy);
yy=3;
System.out.println(yy*yy + ++yy);
```
Quick Check

• What is printed?

```java
int yy = 3;
System.out.println(++yy + yy*yy);
yy=3;
System.out.println(yy*yy + ++yy);
```

20

Is the sum not commutative in JAVA ???

13
Assignment Operators

• Often we perform an operation on a variable, and then store the result back into that variable

• Java provides assignment operators to simplify that process

• For example, the statement

\[ \text{num} += \text{count}; \]

is equivalent to

\[ \text{num} = \text{num} + \text{count}; \]
Assignment Operators

- There are many assignment operators in Java, including the following:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent To</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>x += y</td>
<td>x = x + y</td>
</tr>
<tr>
<td>-=</td>
<td>x -= y</td>
<td>x = x - y</td>
</tr>
<tr>
<td>*=</td>
<td>x *= y</td>
<td>x = x * y</td>
</tr>
<tr>
<td>/=</td>
<td>x /= y</td>
<td>x = x / y</td>
</tr>
<tr>
<td>%=</td>
<td>x %= y</td>
<td>x = x % y</td>
</tr>
</tbody>
</table>
Assignment Operators

• The right hand side of an assignment operator can be a complex expression

• The entire right-hand expression is evaluated first, then the result is combined with the original variable

• Therefore

\[
\text{result} /= (\text{total} - \text{MIN}) \% \text{num};
\]

is equivalent to

\[
\text{result} = \text{result} / ((\text{total} - \text{MIN}) \% \text{num});
\]
Assignment Operators

• The behavior of some assignment operators depends on the types of the operands

• If the operands to the `+=` operator are strings, the assignment operator performs string concatenation

• The behavior of an assignment operator (`+=`) is always consistent with the behavior of the corresponding operator (`+`)
Quiz

• What value is contained in the integer variable length after the following statements are executed?

```java
length = 5;
length *= 2;
length *= length;
length /= 100;
```
Quiz

• What value is contained in the integer variable `length` after the following statements are executed?

```java
length = 5;
length *= 2;
length *= length;
length /= 100;
```

1
Quiz

• What will be printed?

```java
int count, apples;
apples = (count = 3) + 1;
System.out.println("count = " + count);
System.out.println("apples= " + apples);
```
Quiz

• What will be printed?

```java
int count, apples;
apples = (count = 3) + 1;
System.out.println("count = " + count);
System.out.println("apples= " + apples);

count = 3
apples= 4
```
Outline

- Character Strings
- Variables and Assignment
- Primitive Data Types
- Expressions
- Data Conversion
- Interactive Programs
Data Conversion

• Sometimes it is convenient to convert data from one type to another

• For example, in a particular situation we may want to treat an integer (e.g. the number of students) as a floating point value (e.g. for calculating the probability of females)

• These conversions do not change the type of a variable or the value that's stored in it – they only convert a value as part of a computation
## Data Conversion

### Widening Conversions

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>short, int, long, float, or double</td>
</tr>
<tr>
<td>short</td>
<td>int, long, float, or double</td>
</tr>
<tr>
<td>char</td>
<td>int, long, float, or double</td>
</tr>
<tr>
<td>int</td>
<td>long, float, or double</td>
</tr>
<tr>
<td>long</td>
<td>float or double</td>
</tr>
<tr>
<td>float</td>
<td>double</td>
</tr>
</tbody>
</table>

### Narrowing Conversions

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>char</td>
</tr>
<tr>
<td>short</td>
<td>byte or char</td>
</tr>
<tr>
<td>char</td>
<td>byte or short</td>
</tr>
<tr>
<td>int</td>
<td>byte, short, or char</td>
</tr>
<tr>
<td>long</td>
<td>byte, short, char, or int</td>
</tr>
<tr>
<td>float</td>
<td>byte, short, char, int, or long</td>
</tr>
<tr>
<td>double</td>
<td>byte, short, char, int, long, or float</td>
</tr>
</tbody>
</table>
Data Conversion

- **Widening conversions** are safest because they tend to go from a small data type to a larger one (such as a short to an int)

- **Narrowing conversions** can lose information because they tend to go from a large data type to a smaller one (such as an int to a short)

- In Java, data conversions can occur in three ways:
  - **Assignment conversion** (only widening)
  - **Promotion** (only widening)
  - **Casting** (both widening and narrowing)
Assignment Conversion

- **Assignment conversion** occurs when a value of one type is assigned to a variable of another type.

- Example:
  
  ```java
  int dollars = 20;
  double money = dollars;
  ```

- Only widening conversions can happen via assignment (… why?)

- Note that the value or type of `dollars` did not change.
Promotion

• *Promotion* happens automatically when *operators* in *expressions* convert their operands

• Example:

  ```java
  int count = 12;
  double sum = 490.27;
  result = sum / count;
  ```

• The value of `count` is converted to a floating point (`double`) value to perform the division calculation

• This is independent from the type of the variable `result`
Casting

- **Casting** is the most powerful, and dangerous, technique for conversion

- Both widening and narrowing conversions can be accomplished by explicitly casting a value

- To cast, the type is put in parentheses in front of the value being converted

```java
int total = 50;
float result = (float) total / 6;
```

- Without the cast, the fractional part of the answer would be lost
Quiz

• What is the value of the result variable after the execution of the following statements:

```java
long total = 5;
float result = total / 2;
```
Quiz

• What is the value of the result variable after the execution of the following statements:

```java
long total = 5;
float result = total / 2;
```

2.0
Quiz

double d = 5.0;
int i = d / 3;
System.out.println(i)

• Is this ok?
• What is printing?
Quiz

double d = 5.0;
int i = d / 3;
System.out.println(i)

• Is this ok?
• What is printing?
• Nothing is printed; there is a syntax error. Automatic narrowing is not performed. You must cast to int

int i = (int) d / 3;
Quiz

- What is printed and why?

c = 65;
System.out.println(c);
c = (char) -65;  //attention! What char should we get?
int l = c;
System.out.println(l);
Quiz

1. What is printed and why?

```java
char c = 65;
System.out.println(c);
c = (char) -65;
int l = c;
System.out.println(l);
```

A
65471

Because the ASCII and Unicode (decimal) number of A is 65. A narrowing is performed in the second case and we obtain an unpredictable integer (not 65 as one could imagine)
Interactive Programs

- Programs generally need input on which to operate
  
  - The `Scanner` class provides convenient methods for reading input values of various types

  - A `Scanner` object can be set up to read input from various sources, including the user typing values on the keyboard

  - Keyboard input is represented by the `System.in` object
Reading Input

• The following line creates a Scanner object that reads from the keyboard:

    Scanner scan = new Scanner (System.in);

• The new operator creates the Scanner object

• Once created, the Scanner object can be used to invoke various input methods, such as:

    answer = scan.nextLine();
Reading Input

- The `Scanner` class is part of the `java.util` class library, and must be imported into a program to be used.

- The `nextLine` method reads all of the input until the end of the line is found.

- See `Echo.java`.

- The details of object creation and class libraries are discussed further in Chapter 3.
import java.util.Scanner;

public class Echo {
    //---
    //  Reads a character string from the user and prints it.
    //---
    public static void main (String[] args) {
        String message;
        Scanner scan = new Scanner (System.in);

        System.out.println ("Enter a line of text:");
        message = scan.nextLine();

        System.out.println ("You entered: \\
        " + message + \\
        \");
    }
}
import java.util.Scanner;

public class Echo
{
    //---
    // Reads a character string from the user and prints it.
    //---
    public static void main (String[] args)
    {
        String message;
        Scanner scan = new Scanner (System.in);

        System.out.println ("Enter a line of text: ");

        message = scan.nextLine();

        System.out.println ("You entered: \\
                           " + message + \\
                           ");
    }
}
Input Tokens

• Unless specified otherwise, *white space* is used to separate the elements (called *tokens*) of the input

• White space includes space characters, tabs, new line characters

• The *next method* of the *Scanner class* reads the next input token and returns it as a string

• Methods such as *nextInt* and *nextDouble* read data of particular types

• See *GasMileage.java*
```java
import java.util.Scanner;

public class GasMileage {

    // Calculates fuel efficiency based on values entered by the user.

    public static void main (String[] args) {
        int miles;
        double gallons, mpg;

        Scanner scan = new Scanner (System.in);
    }
}
```
continue

    System.out.print ("Enter the number of miles: ");
miles = scan.nextInt();

    System.out.print ("Enter the gallons of fuel used: ");
gallons = scan.nextDouble();

    mpg = miles / gallons;

    System.out.println ("Miles Per Gallon: " + mpg);
}
System.out.print("Enter the number of miles: ");
miles = scan.nextInt();

System.out.print("Enter the gallons of fuel used: ");
gallons = scan.nextDouble();

mpg = miles / gallons;

System.out.println("Miles Per Gallon: "+ mpg);

Sample Run

Enter the number of miles: 328
Enter the gallons of fuel used: 11.2
Miles Per Gallon: 29.28571428571429
Summary

• Chapter 2 focused on:
  – character strings
  – primitive data
  – the declaration and use of variables
  – expressions and operator precedence
  – data conversions
  – accepting input from the user
Variables I

- **Instance Variables (Non-Static Fields):** objects store their individual states in (non-static) fields are also known as *instance variables* because their values are unique to each *instance* of a class (to each object, in other words)

- **Class Variables (Static Fields):** A *class variable* is any field declared with the static modifier; this tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated
Variables II

- **Local Variables** Similar to how an object stores its state in fields, a method will often store its temporary state in *local variables*. The syntax for declaring a local variable is similar to declaring a field (for example, `int count = 0;`)

- **Parameters** You've already seen an example of parameters, in the main method of the "Lincoln" application. The *signature* for the main method is `public static void main(String[] args)`. Here, the `args` variable is the *parameter* to this method.
Quiz

• What value will \( z \) have if we execute the following assignment statement?
  \[
  \text{int } z = 50 \div 10.00;
  \]

  – A) 5
  – B) 5.0
  – C) 50
  – D) 10
  – E) none of the above, a run-time error arises because \( z \) is an \text{int} and 50 \div 10.00 is not
Quiz

• What value will \( z \) have if we execute the following assignment statement?

\[
\text{int } z = 50 / 10.00;
\]

- A) 5
- B) 5.0
- C) 50
- D) 10
- E) none of the above, a compile-time error arises because \( z \) is an int and \( 50 / 10.00 \) is not

• Because 10.00 is a double, the division produces a double precision value which cannot be stored in the int \( z \). Either cast to int 10.00 or the result of the division.
Quiz

• What value is contained in the integer variable size after the following statements are executed?

```java
size = 18;
size = size + 12;
size = size * 2;
size = size / 50;
```
Quiz

What value is contained in the integer variable `size` after the following statements are executed?

```java
size = 18;
size = size + 12;
size = size * 2;
size = size / 50;
```

1
Quiz

• Write four different program statements that increment the value of an integer variable `total`.
Quiz

• Write four different program statements that increment the value of an integer variable `total`.

```plaintext
total = total + 1;
total += 1;
total++;
++total;
```
Quiz

• What is the difference between the literals 4, 4.0, '4', and "4"?

• Explain the following programming statement in terms of objects and the services they provide.
  – System.out.println("I gotta be me!");

• What is wrong with the following program statement? How can it be fixed?
  – System.out.println("To be or not to be, that is the question.");
Quiz

• What output is produced by the following statement? Explain.
  – System.out.println("50 plus 25 is " + 50 + 25);

• What value is contained in the integer variable size after the following statements are executed?
  size = 18;
  size = size + 12;
  size = size * 2;
  size = size / 4;
Quiz

• How many bits are needed to store a color picture that is 400 pixels wide and 250 pixels high? Assume color is represented using the RGB technique and that no special compression is done.

• For each of the following expressions, indicate the order in which the operators will be evaluated by writing a number beneath each operator.
  – a – b – c – d
  – a / b * c * d
  – a – ((b – c) – d)
  – a % (b % c) * d * e
  – (a + b) * (c / d) % e
Quiz

• What do each of the following print?

```java
System.out.println('b');
System.out.println('b' + 'c');
System.out.println((int) 'b');
System.out.println((int) 'c');
System.out.println((char) ('a' + 4));
```
Quiz

• What do each of the following print?

```java
System.out.println('b');
System.out.println('b' + 'c');
System.out.println((int) 'b');
System.out.println((int) 'c');
System.out.println((char) ('a' + 4));
b
197
98
99
e
```