Chapter 2

Data and Expressions
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
  - Java applets
  - introduction to graphics
Outline

- Character Strings
- Variables and Assignment
- Primitive Data Types
- Expressions
- Data Conversion
- Interactive Programs
- Graphics
- Applets
- Drawing Shapes
Character Strings

A string of characters can be represented as a string literal by putting double quotes around the text:

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

Every character string is an object in Java, defined by the String class

Every string literal represents a String object
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.");
```
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` (page 63)
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
- See **Facts.java** (page 65)
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 one 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 (page 67)
Escape Sequences

• What if we wanted to print a 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></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 (page 68)
Outline

Character Strings
Variables and Assignment
Primitive Data Types
Expressions
Data Conversion
Interactive Programs
Graphics
Applets
Drawing Shapes
Variables

• A *variable* is a name for a location in memory

• A variable must be *declared* by specifying the variable's name and the type of information that it will hold

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

Multiple variables can be created in one declaration
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 (page 70)
Assignment

• An assignment statement changes the value of a variable
• The assignment operator is the = sign

\[ \text{total} = 55; \]

• The expression on the right is evaluated and the result is stored in the variable on the left
• 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 (page 71)
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

  ```java
  final int MIN_HEIGHT = 69;
  ```
Constants

- Constants are useful for three important reasons
- First, they give meaning to otherwise unclear literal values
  - For 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 updated in one place
- Third, they formally establish that a value should not change, avoiding inadvertent errors by other programmers
Outline

Character Strings
Variables and Assignment
Primitive Data Types
Expressions
Data Conversion
Interactive Programs
Graphics
Applets
Drawing Shapes
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 various numeric primitive types is their size, and therefore 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'    '$'    ','    '\n'
• Example declarations:
  char topGrade = 'A';
  char terminator = ';', separator = ' ';
• Note the distinction 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, …
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

```java
boolean done = false;
```

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

Character Strings
Variables and Assignment
Primitive Data Types
Expressions
Data Conversion
Interactive Programs
Graphics
Applets
Drawing Shapes
Expressions

• An *expression* is a combination of one or more operators and operands

• *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
</tr>
<tr>
<td>Remainder</td>
<td>%</td>
</tr>
</tbody>
</table>

• If either or both operands used by an arithmetic operator are floating point, then the result is a floating point
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 second operand into the first
  
  \[
  14 \% 3 \quad \text{equals} \quad 2 \\
  8 \% 12 \quad \text{equals} \quad 8 
  \]
Operator Precedence

• Operators can be combined into complex expressions

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

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

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

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

• What is the order of evaluation in the following expressions?

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

1 2 3 4

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

3 1 4 2

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

2 1 4 3

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

4 3 2 1
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

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

```plaintext
count = count + 1;
```

Then the result is stored back into count (overwriting the original value)
Increment and Decrement

• The increment and decrement operators use only one operand

• The increment operator \((++)\) adds one to its operand

• The decrement operator \((--\)) subtracts one from its 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
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

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

is equivalent to

\[
\text{result} = \text{result} / ((\text{total}-\text{MIN}) \mod \text{num});
\]
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 (`+`).
Outline

Character Strings
Variables and Assignment
Primitive Data Types
Expressions
Data Conversion
Interactive Programs
Graphics
Applets
Drawing Shapes
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 as a floating point value
- 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

- Conversions must be handled carefully to avoid losing information.
- *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
  - promotion
  - casting
Assignment Conversion

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

- If `money` is a `float` variable and `dollars` is an `int` variable, the following assignment converts the value in `dollars` to a `float`:

  ```
  money = dollars
  ```

- Only widening conversions can happen via assignment.

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

- *Promotion* happens automatically when operators in expressions convert their operands.

- For example, if `sum` is a `float` and `count` is an `int`, the value of `count` is converted to a floating point value to perform the following calculation:

  
  ```
  result = sum / count;
  ```
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

- For example, if `total` and `count` are integers, but we want a floating point result when dividing them, we can cast `total`:

  ```java
  result = (float) total / count;
  ```
Outline

Character Strings
Variables and Assignment
Primitive Data Types
Expressions
Data Conversion
Interactive Programs
Graphics
Applets
Drawing Shapes
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:

```java
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:

```java
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.

• See Echo.java (page 91)

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

• The details of object creation and class libraries are discussed further in Chapter 3.
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](#) (page 92)
Outline

Character Strings
Variables and Assignment
Primitive Data Types
Expressions
Data Conversion
Interactive Programs
Graphics
Applets
Drawing Shapes
Introduction to Graphics

• The last few sections of each chapter of the textbook focus on graphics and graphical user interfaces

• A picture or drawing must be digitized for storage on a computer

• A picture is made up of *pixels* (picture elements), and each pixel is stored separately

• The number of pixels used to represent a picture is called the *picture resolution*

• The number of pixels that can be displayed by a monitor is called the *monitor resolution*
Coordinate Systems

- Each pixel can be identified using a two-dimensional coordinate system.
- When referring to a pixel in a Java program, we use a coordinate system with the origin in the top-left corner.

![Diagram of a coordinate system with the origin at (0, 0), the X-axis reaching 112 and the Y-axis reaching 40, with a pixel at (112, 40).]
Representing Color

- A black and white picture could be stored using one bit per pixel (0 = white and 1 = black)

- A colored picture requires more information; there are several techniques for representing colors

- For example, every color can be represented as a mixture of the three additive primary colors Red, Green, and Blue

- Each color is represented by three numbers between 0 and 255 that collectively are called an RGB value
The Color Class

- A color in a Java program is represented as an object created from the `Color` class.
- The `Color` class also contains several predefined colors, including the following:

<table>
<thead>
<tr>
<th>Object</th>
<th>RGB Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color.black</td>
<td>0, 0, 0</td>
</tr>
<tr>
<td>Color.blue</td>
<td>0, 0, 255</td>
</tr>
<tr>
<td>Color.cyan</td>
<td>0, 255, 255</td>
</tr>
<tr>
<td>Color.orange</td>
<td>255, 200, 0</td>
</tr>
<tr>
<td>Color.white</td>
<td>255, 255, 255</td>
</tr>
<tr>
<td>Color.yellow</td>
<td>255, 255, 0</td>
</tr>
</tbody>
</table>
Outline

- Character Strings
- Variables and Assignment
- Primitive Data Types
- Expressions
- Data Conversion
- Interactive Programs
- Graphics
- Applets
- Drawing Shapes
Applets

- A Java application is a stand-alone program with a main method (like the ones we've seen so far)
- A Java applet is a program that is intended to transported over the Web and executed using a web browser
- An applet also can be executed using the appletviewer tool of the Java Software Development Kit
- An applet doesn't have a main method
- Instead, there are several special methods that serve specific purposes
Applets

- The `paint` method, for instance, is executed automatically and is used to draw the applet’s contents.
- The `paint` method accepts a parameter that is an object of the `Graphics` class.
- A `Graphics` object defines a `graphics context` on which we can draw shapes and text.
- The `Graphics` class has several methods for drawing shapes.
Applets

- The class that defines an applet extends the Applet class
- This makes use of inheritance, which is explored in more detail in Chapter 8
- See Einstein.java (page 97)
- An applet is embedded into an HTML file using a tag that references the bytecode file of the applet
- The bytecode version of the program is transported across the web and executed by a Java interpreter that is part of the browser
The HTML applet Tag

<html>
  <head>
    <title>The Einstein Applet</title>
  </head>
  <body>
    <applet code="Einstein.class" width=350 height=175>
    </applet>
  </body>
</html>
Outline

Character Strings
Variables and Assignment
Primitive Data Types
Expressions
Data Conversion
Interactive Programs
Graphics
Applets
Drawing Shapes
Drawing Shapes

• Let's explore some of the methods of the Graphics class that draw shapes in more detail.

• A shape can be filled or unfilled, depending on which method is invoked.

• The method parameters specify coordinates and sizes.

• Shapes with curves, like an oval, are usually drawn by specifying the shape’s bounding rectangle.

• An arc can be thought of as a section of an oval.
page.drawLine (10, 20, 150, 45);

or

page.drawLine (150, 45, 10, 20);
Drawing a Rectangle

```java
page.drawRect(50, 20, 100, 40);
```
Drawing an Oval

bounding rectangle

page.drawOval (175, 20, 50, 80);
Drawing Shapes

• Every drawing surface has a *background color*
• Every graphics context has a current *foreground color*
• Both can be set explicitly
• See [Snowman.java](#) (page 103)
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
  ▪ Java applets
  ▪ introduction to graphics