Basic Computation

Chapter 2
Increment and Decrement Operators

- Used to increase (or decrease) the value of a variable by 1
- Easy to use, important to recognize
- The increment operator
  \[ \text{count++ or ++count} \]
- The decrement operator
  \[ \text{count-- or --count} \]
Increment and Decrement Operators

- equivalent operations

```java
count++;  
++count;  
count = count + 1;

count--;  
--count;  
count = count - 1;
```
Increment and Decrement Operators in Expressions

- after executing
  ```java
  int m = 4;
  int result = 3 * (++m)
  result has a value of 15 and m has a value of 5
  ```
- after executing
  ```java
  int m = 4;
  int result = 3 * (m++)
  result has a value of 12 and m has a value of 5
  ```
The Class **String**

- We've used constants of type **String** already.
  
  "Enter a whole number from 1 to 99."

- A value of type **String** is a
  
  - Sequence of characters
  - Treated as a single item.
String Constants and Variables

• Declaring

```java
String greeting;
greeting = "Hello!";
```

or

```java
String greeting = "Hello!";
```

or

```java
String greeting = new String("Hello!");
```

• Printing

```java
System.out.println(greeting);
```
String Methods

- An object of the `String` class stores data consisting of a sequence of characters.
- Objects have methods as well as data.
- The `length()` method returns the number of characters in a particular `String` object.

```java
String greeting = "Hello";
int n = greeting.length();
```
The Method length()

- The method `length()` returns an `int`.
- You can use a call to method `length()` anywhere an `int` can be used.

```java
int count = command.length();
System.out.println("Length is " + command.length());
count = command.length() + 3;
```
String Indices

- Figure 2.4

- Positions start with 0, not 1.
  - The 'J' in "Java is fun." is in position 0
  - A position is referred to an an *index*.
  - The 'f' in "Java is fun." is at index 8.
# String Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>charAt(Index)</code></td>
<td>Returns the character at <code>Index</code> in this string. Index numbers begin at 0.</td>
</tr>
<tr>
<td><code>compareTo(A_String)</code></td>
<td>Compares this string with <code>A_String</code> to see which string comes first in the lexicographic ordering. (Lexicographic ordering is the same as alphabetical ordering when both strings are either all uppercase letters or all lowercase letters.) Returns a negative integer if this string is first, returns zero if the two strings are equal, and returns a positive integer if <code>A_String</code> is first.</td>
</tr>
<tr>
<td><code>concat(A_String)</code></td>
<td>Returns a new string having the same characters as this string concatenated with the characters in <code>A_String</code>. You can use the <code>↓</code> operator instead of <code>concat</code>.</td>
</tr>
<tr>
<td><code>equals(Other_String)</code></td>
<td>Returns true if this string and <code>Other_String</code> are equal. Otherwise, returns false.</td>
</tr>
</tbody>
</table>
### String Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equalsIgnoreCase(Other_String)</td>
<td>Behaves like the method <code>equals</code>, but considers uppercase and lowercase versions of a letter to be the same.</td>
</tr>
<tr>
<td>indexOf(A_String)</td>
<td>Returns the index of the first occurrence of the substring <code>A_String</code> within this string. Returns -1 if <code>A_String</code> is not found. Index numbers begin at 0.</td>
</tr>
<tr>
<td>lastIndexOf(A_String)</td>
<td>Returns the index of the last occurrence of the substring <code>A_String</code> within this string. Returns -1 if <code>A_String</code> is not found. Index numbers begin at 0.</td>
</tr>
</tbody>
</table>
# String Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>length()</code></td>
<td>Returns the length of this string.</td>
</tr>
<tr>
<td><code>toLowerCase()</code></td>
<td>Returns a new string having the same characters as this string, but with any uppercase letters converted to lowercase.</td>
</tr>
<tr>
<td><code>toUpperCase()</code></td>
<td>Returns a new string having the same characters as this string, but with any lowercase letters converted to uppercase.</td>
</tr>
</tbody>
</table>

---

**Figure 2.5c**
### String Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>replace(OldChar, NewChar)</code></td>
<td>Returns a new string having the same characters as this string, but with each occurrence of <em>OldChar</em> replaced by <em>NewChar</em>.</td>
</tr>
<tr>
<td><code>substring(Start)</code></td>
<td>Returns a new string having the same characters as the substring that begins at index <em>Start</em> of this string through to the end of the string. Index numbers begin at 0.</td>
</tr>
<tr>
<td><code>substring(Start, End)</code></td>
<td>Returns a new string having the same characters as the substring that begins at index <em>Start</em> of this string through, but not including, index <em>End</em> of the string. Index numbers begin at 0.</td>
</tr>
<tr>
<td><code>trim()</code></td>
<td>Returns a new string having the same characters as this string, but with leading and trailing whitespace removed.</td>
</tr>
</tbody>
</table>

**Figure 2.5d**

*JAVA: An Introduction to Problem Solving & Programming, 6th Ed. By Walter Savitch*  
String Processing

• No methods allow you to change the value of a String object.
• But you can change the value of a String variable.
• View sample program StringDemo listing 2.4

Sample Screen Output

Text processing is hard!
012345678901234567890123
The word "hard" starts at index 19
The changed string is:
TEXT PROCESSING IS EASY!
Escape Characters

• How would you print

   "Java" refers to a language. ?

• The compiler needs to be told that the quotation marks (") do not signal the start or end of a string, but instead are to be printed.

   System.out.println(
   "\\"Java\\" refers to a language.");
Escape Characters

- " Double quote.
- ' Single quote.
- \ Backslash.
- \n New line. Go to the beginning of the next line.
- \r Carriage return. Go to the beginning of the current line.
- \t Tab. Add whitespace up to the next tab stop.

• Figure 2.6
• Each escape sequence is a single character even though it is written with two symbols.
System.out.println("abc\\def");

\abc\def

System.out.println("new\\nline");

\new\\nline

char singleQuote = '\'';
System.out.println(singleQuote);

\'
Keyboard and Screen I/O: Outline

• Screen Output
• Keyboard Input
Screen Output

• We've seen several examples of screen output already.
  • `System.out` is an object that is part of Java.
  • `println()` is one of the methods available to the `System.out` object.
  • Alternatively, there is `print()`
Keyboard Input

• Java has reasonable facilities for handling keyboard input.

• These facilities are provided by the **Scanner** class in the **java.util** package.
  ▪ A *package* is a library of classes.
Using the Scanner Class

• Near the beginning of your program, insert
  
  ```java
  import java.util.Scanner;
  ```

• Create an object of the `Scanner` class
  
  ```java
  Scanner keyboard =
      new Scanner (System.in)
  ```

• Read data (an `int` or a `double`, for example)
  
  ```java
  int n1 = keyboard.nextInt();
  double d1 = keyboard.nextDouble();
  ```
Keyboard Input Demonstration

• View sample program

```
class ScannerDemo, listing 2.5
```

Enter two whole numbers separated by one or more spaces:
  42   43
You entered 42 and 43
Next enter two numbers.
A decimal point is OK.
  9.99  21
You entered 9.99 and 21.0
Next enter two words:
plastic spoons
You entered "plastic" and "spoons"
Next enter a line of text:
May the hair on your toes grow long and curly.
You entered "May the hair on your toes grow long and curly."
Some **Scanner** Class Methods

- **Figure 2.7a**

  - `Scanner_Object_Name.next()`
    Returns the `String` value consisting of the next keyboard characters up to, but not including, the first delimiter character. The default delimiters are whitespace characters.

  - `Scanner_Object_Name.nextLine()`
    Reads the rest of the current keyboard input line and returns the characters read as a value of type `String`. Note that the line terminator '\n' is read and discarded; it is not included in the string returned.

  - `Scanner_Object_Name.nextInt()`
    Returns the next keyboard input as a value of type `int`.

  - `Scanner_Object_Name.nextDouble()`
    Returns the next keyboard input as a value of type `double`.

  - `Scanner_Object_Name.nextFloat()`
    Returns the next keyboard input as a value of type `float`. 
Some **Scanner** Class Methods

- **Figure 2.7b**

```java
Scanner_Object_Name.nextLong()
   Returns the next keyboard input as a value of type **long**.

Scanner_Object_Name.nextByte()
   Returns the next keyboard input as a value of type **byte**.

Scanner_Object_Name.nextShort()
   Returns the next keyboard input as a value of type **short**.

Scanner_Object_Name.nextBoolean()
   Returns the next keyboard input as a value of type **boolean**. The values of **true** and **false** are entered as the words **true** and **false**. Any combination of uppercase and lowercase letters is allowed in spelling **true** and **false**.

Scanner_Object_Name.useDelimiter(Delimiter_Word);
   Makes the string **Delimiter_Word** the only delimiter used to separate input. Only the exact word will be a delimiter. In particular, blanks, line breaks, and other whitespace will no longer be delimiters unless they are a part of **Delimiter_Word**.

   This is a simple case of the use of the **useDelimiter** method. There are many ways to set the delimiters to various combinations of characters and words, but we will not go into them in this book.
```
**nextLine() Method Caution**

- The `nextLine()` method reads
  - The remainder of the current line,
  - Even if it is empty.
nextLine() Method Caution

- Example – given following declaration.

```java
int n;
String s1, s2;
n = keyboard.nextInt();
s1 = keyboard.nextLine();
s2 = keyboard.nextLine();
```

- Assume input shown

```
42
and don't you forget it.
```

n is set to **42**

but s1 is set to the empty string.
The Empty String

- A string can have any number of characters, including zero.
- The string with zero characters is called the *empty* string.
- The empty string is useful and can be created in many ways including

  ```java
  String s3 = "";
  ```