Chapter 8:
Strings and Things

Think Java:
How to Think Like a Computer Scientist

5.1.2

by Allen B. Downey
Word Of Fortune Program

Player 1, enter a secret word, from 3-10 letters
camel

Player 2, enter a guess, it should have 5 letters
bricks
wrong number of letters
Player 2, enter a guess, it should have 5 letters
brick
----
Player 2, enter a guess, it should have 5 letters
bread
-----
The first letter is c
Player 2, enter a guess, it should have 5 letters
craps
c----
Player 2, enter a guess, it should have 5 letters
camps
cam--
Player 2, enter a guess, it should have 5 letters
camel
camel
You got it in 6 guesses!
Chapter Topics

Chapter 8 discusses the following main topics:

– The char data type
– String methods: length, charAt
– Traversal
– Run-time Errors/Exceptions
– Reading Javadoc files
– Looping and Counting
– String methods: indexOf, compareTo, substring
– Working with input files
What is the data contained in a String object?

- The components of a String object are letters:
  - For example, 'a', 'b', 'X', 'Y', '@', '-', '.', and ''
- A better word for these is **characters**
- Java has a data type that stores single characters, called **char**
  - char works like the other types we have seen:
    ```java
    char ltr = 'c';
    if (ltr == 'c') {
        System.out.println(ltr);
    }
    ```
The `char` Data Type

- String values appear in double quotes, like "hello"
- Character values appear in single quotes, like 'x'
- `char` values can contain only a single letter or symbol.
- Don’t confuse `char` literals with string literals.
  - `char` literals: 'a', 'Z', '%'
  - String literals: "abc", "Z", "$4.50"
The `charAt` method for Strings

- Returns the letter or symbol located at some location in the String
  
  ```java
  String fruit = "banana";
  char letter = fruit.charAt(1);
  System.out.println(letter);
  ```
  - In the above case, letter contains the value 'a'
  - The actual first letter of a String is at location 0
    ```java
    char letter = fruit.charAt(0);
    ```
    - now letter contains 'b'

The `length` method for Strings

- Takes no arguments
- Returns the number of letters in a String

```java
String city = "New York";
int length = city.length();
System.out.println(length);
```

- Knowing length, can we find the last letter of a String?

```java
char last = city.charAt(length); //ERROR
```

  - The actual last letter of a String is at location `length-1`

```java
char last = city.charAt(length-1);
```

  - now `last` contains 'k'
Traversing Strings

• We often wish to process a String letter-by-letter
• This pattern of processing is called a traversal.
• A natural way to encode a traversal is with a while statement:

```java
int index = 0;
while (index < fruit.length()) {
    char letter = fruit.charAt(index);
    System.out.println(letter);
    index = index + 1;
}
```

• prints each letter on a line by itself
String fruit = banana;
int index = 0;
while (index < fruit.length()) {
    char letter = fruit.charAt(index);
    System.out.println(letter);
    index = index + 1;
}
Traversing Strings

• Notice the condition
  \[
  \text{index} < \text{fruit}.\text{length}() \]
• when \text{index} equals the length of the String
  – condition is false
  – last true value is when \text{index} = \text{length}-1
• Why the variable name \text{index} ?
• An \text{index} specifies a member in an ordered set
  – \text{index} \textit{indicates} which letter we are talking about
Run Time Errors

• In Java run-time errors are called **exceptions**.
• Exceptions are **thrown** for illegal actions
  – like asking for the `charAt` with a negative index
  – or an `index > length-1`

```java
public class BadString {

    public static void main(String[] args) {
        processWord("banana");
    }

    public static void processWord(String s) {
        char c = getLastLetter(s);
        System.out.println(c);
    }

    public static char getLastLetter(String s) {
        int index = s.length(); // WRONG!
        char c = s.charAt(index);
        return c;
    }
}
```
Run Time Errors

• In Java run-time errors are called **exceptions**.
• Exceptions are **thrown** for illegal actions
  – like asking for the `charAt` with a negative index
  – or an index > length-1

```java
public class BadString {

    public static void main(String[] args) {
        processWord("banana");
    }

    public static void processWord(String s) {
        char c = getLastLetter(s);
        System.out.println(c);
    }

    public static char getLastLetter(String s) {
        int index = s.length()-1  // RIGHT! 
        char c = s.charAt(index);
        return c;
    }
}
```
The exception thrown from the previous example

• Shows the type of exception and a stack trace
  
  `java.lang.StringIndexOutOfBoundsException: String index out of range: 6`
  
  `at java.lang.String.charAt(String.java:658)
  at BadString.getLastLetter(BadString.java:14)
  at BadString.processWord(BadString.java:8)
  at BadString.main(BadString.java:4)`

• Exceptions are like penalty flags in sports
  – If you know how to read them you'll get more out of the game!
In the previous slide

• What kind of Exception occurred, and what package is it defined in?
• What is the value of the index that caused the exception?
• What method threw the exception, and where is that method defined?
• What method invoked `charAt`?
• In `BadString.java`, what is the line number where `charAt` was invoked?
Reading Documentation

- Javadoc for String class
  - [http://docs.oracle.com/javase/6/docs/api/java/lang/String.html](http://docs.oracle.com/javase/6/docs/api/java/lang/String.html)
- Javadoc can be found easily with google search
  - "java string class" will take you straight to page

```
String(StringBuilder builder)
Allocates a new string that contains the sequence of characters currently contained
```

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>int</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>int</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
charAt

public char charAt(int index)

Returns the char value at the specified index. An index ranges from 0 to length() – 1. The first char value of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

If the char value specified by the index is a surrogate, the surrogate value is returned.

Specified by:
- charAt in interface CharSequence

Parameters:
- index - the index of the char value.

Returns:
- the char value at the specified index of this string. The first char value is at index 0.

Throws:
- IndexOutOfBoundsException - if the index argument is negative or not less than the length of this string.
Decoding the Javadoc info

• The method's *prototype*:

  \[
  \text{public char charAt(int index)}
  \]

  – Specifies the name of the method, type of parameters, and the return type

• What the method does:

  Returns the char value at the specified index.
Comparing Strings, equals method

• Strings are objects
  – Using == and <= doesn't work right with objects
  – use equals method to check for equality

```java
String name1 = "Alan Turing";
String name2 = "Ada Lovelace";

if (name1.equals (name2)) {
    System.out.println("The names are the same.");
}
```
Begin Lab 8 Part A

• Word of Fortune!
• BTW You will need to repeat the main game loop as long as the two words are NOT equal
• To test for INequality, put a ! in front of expression, for example:

```java
String name1 = "Alan Turing";
String name2 = "Ada Lovelace";

if (!name1.equals(name2)) {
    System.out.println("The names are not the same.");
}
```

• If you finish early, start on Assignment 8
String Topics ++

• More String methods
  – indexOf
  – toUpperCase, toLowerCase
  – substring
  – Comparing Strings (equals and compareTo)

• Looping and counting

• Reading from files
The `indexOf` method for Strings

- `indexOf` is the reverse of `charAt`:
  - `charAt` returns the character at the desired index
  - `indexOf` returns the index of the desired character

- Use it to find where a letter appears in a String
  
  ```java
  String fruit = "banana";
  int index = fruit.indexOf('a');
  ```
  
  - In the above example `index` gets the number 1
  - If the letter is not in the string, `indexOf` returns -1

Another version of `indexOf`

- Takes a second argument that indicates where in the string to start looking
  
  ```java
  int index = fruit.indexOf('a', 2);
  ```
  
  - In the above, `index` gets the number 3

- If the letter is at the start index, the start index is returned
  
  ```java
  int index = fruit.indexOf('a', 5);
  ```
  
  - In the above, `index` gets the number 5
Converting toUpperCase, toLowerCase

• Here's how we can convert the case of a String

```java
String greeting = "hello";
S...print(greeting.toUpperCase());
HELLO
```

• Note the original value of greeting is unchanged

```java
S...print(greeting);
hello
```
Strings are immutable

• Once created, the value of a String object cannot change

• When you invoke `toUpperCase` on a String, you get a new String as a return value. For example:

  ```java
  String name = "Alan Turing";
  String upperName = name.toUpperCase();
  ```

  – `upperName` contains "ALAN TURING"
  – `name` still contains "Alan Turing".
Selecting a substring of a String

- Returns a *new* String which is a subset of original

```java
String s1 = "hamburger";
String s2 = s1.substring(4, 8);  // urge

String s3 = "smiles";
String s4 = s2.substring(1, 5);  // mile
```

```java
public String substring(int beginIndex, int endIndex)

Returns a new string that is a substring of this string. The substring begins at the specified beginIndex and extends to the character at index endIndex - 1. Thus the length of the substring is endIndex - beginIndex.
```
Another version of `substring`

- Only takes a start index, goes to end of String

```java
String s1 = "hamburger";
String s2 = s1.substring(3);   //  burger
String s3 = "smiles";
String s4 = s2.substring(1);   //  miles
```

```java
public String substring(int beginIndex)
```

Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.
What value is returned by these calls?

String word = "onomotopoeia";
word.indexOf('m');
word.charAt(word.indexOf('p'));
word.indexOf('o',5);
word.substring(5,8);
word.substring(7);
word.substring(2,6).toUpperCase();
What value is returned by these calls?

```java
String word = "onomotopoeia";
word.indexOf('m');
word.charAt(word.indexOf('p'));
word.indexOf('o', 5);
word.substring(5, 8);
word.substring(7);
word.substring(2, 6).toUpperCase();
```
What value is returned by these calls?

String word = "onomotopoeia";
word.indexOf('m'); 3
word.charAt(word.indexOf('p')); p
word.indexOf('o',5); 6
top
word.substring(5,8); poeia
word.substring(7);
word.substring(2,6).toUpperCase(); OMOT
Looping and counting

• The following program counts the number of times the letter ‘a’ appears in a string:

```java
String fruit = "banana";
int length = fruit.length();
int count = 0;

int index = 0;
while (index < length) {
    if (fruit.charAt(index) == 'a') {
        count = count + 1;
    }
    index = index + 1;
}
System.out.println(count);
```
Previous example uses a **counter**

- The variable `count` is initialized to zero
  - and then incremented each time we find an ’a’.
  - To **increment** is to increase by one
  - To **decrement** is to decrease by one
  - After the loop, `count` contains the result: the total number of a’s.
Increment and Decrement Operators

• The ++ operator
  – adds one to the current value of an int or char

• The -- operator
  – subtracts one from the current value of an int/char

• Neither operator works on doubles, booleans or Strings.
Redoing the previous example

- Using the increment operators, we can rewrite the letter-counter:

```java
int index = 0;
while (index < length) {
    if (fruit.charAt(index) == 'a') {
        count++;
    }
    index++;
}
```
Trace a Counting Loop

```java
String fruit = banana;
int index = 0, count = 0;
while (index < fruit.length()) {
    if (fruit.charAt(index) == 'a') {
        count++;
    }
    index++;
}
System.out.println("number of a's = " + count);
```

<table>
<thead>
<tr>
<th>fruit</th>
<th>fruit.length()</th>
<th>index</th>
<th>fruit.charAt(index)</th>
<th>Console</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Warning with ++/--

• It is a common error to write something like
  \[
  \text{index} = \text{index}++; \quad // \quad \text{WRONG!!}
  \]

• Syntactically legal
  – compiler will not warn you.
  – leaves the value of \text{index} unchanged.
  – Often a difficult bug to track down.

• Remember, you can write \text{index} = \text{index} + 1, or you can write \text{index}++, but you shouldn’t mix them.
Comparing primitives

- Primitives types are:
  - `int`, `double`, `char`, `boolean`
- Only contain a single value, no methods
  - `4`, `3.14`, `'a'`, `true`
- Compare primitives using relational operators
  - `==`, `!=`, `>`, `<`, `>=`, `<=`
- Example:
  ```java
  int x = 5;
  if (x == 5){
      S...print("the condition is true");
  }
  ```
Comparing Strings, \texttt{equals} method

- Strings are objects
  - Using \texttt{==} and \texttt{<=} doesn't work right with objects
  - \texttt{use equals} method to check for equality

```java
String name1 = "Alan Turing";
String name2 = "Ada Lovelace";

if (name1.equals(name2)) {
    System.out.println("The names are the same.");
}
```
String `compareTo` method

• Lets you determine the order of strings

```java
int flag = name1.compareTo(name2);
if (flag == 0) {
    System.out.println("The names are the same.");
} else if (flag < 0) {
    System.out.println("name1 comes before name2.");
} else if (flag > 0) {
    System.out.println("name2 comes before name1.");
}
```

• `compareTo` returns:
  – positive number if first comes after second
  – negative number if first comes before second
  – zero if they are the same
What is the result of these conditions? (or say if it's an error)

String noun = "apple", verb = "apply";
int x = 4, y = 3;
noun.equals("apple")
noun.equals(verb)
x.equals(y)
noun < verb
noun.compareTo(verb)<0
x.compareTo(y)
noun.compareTo("apple")==0
What is the result of these conditions?
(or say if it's an error)

String noun = "apple", verb = "apply";
int x = 4, y = 3;
noun.equals("apple") true
noun.equals(verb) false
x.equals(y) ERROR!
noun < verb ERROR!
noun.compareTo(verb)<0 true
x.compareTo(y) ERROR
noun.compareTo("apple")==0 true
Working with Input Files

- First, set up a Scanner to point to the file
- then you can read from the file just as you have read from the keyboard.

```java
// Setup a Scanner to read from a text file
Scanner in = new Scanner(new File("in.txt"));
```

- The file being read is "in.txt"
- must be in same folder as your program files
Java Imports for Input Files

import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;

• the last one is to handle the run-time error where the file is not located where it is expected.
Throwing an Exception

- Also, your main method needs to be able to throw an exception if the file is not found:

```java
public static void main(String[] args) throws FileNotFoundException {
    // Needed for file operation
```
Example

```java
import java.util.Scanner; // Needed to use Scanner for input
import java.io.File; // Needed to use File
import java.io.FileNotFoundException; // Needed for file operation

public class FileScanner {
    public static void main(String[] args) throws FileNotFoundException { // Needed for file operation
        int num1;
        double num2;
        String name;
        double sum;

        // Setup a Scanner to read from a text file
        Scanner in = new Scanner(new File("in.txt"));
        num1 = in.nextInt(); // use nextInt() to read int
        num2 = in.nextDouble(); // use nextDouble() to read double
        name = in.next(); // use next() to read String

        // Display
        .......
    }
}
```

"in.txt" contents:
12
33.44
Peter
Reading to the end of file

- Scanner objects have a `hasNext()` method — true when there is still more in the file

```java
Scanner inputFile = 
    new Scanner(new File("words.txt"));
int count = 0;
while(inputFile.hasNext()){ 
    inputFile.next();
    count++;
}
S...println("File has " + count + " words");
```
Find out how many words start with 's'

Scanner in = new Scanner(new File("words.txt"));
int count = 0;
String word;
while(in.hasNext()){
    word = in.next();
    if (word.charAt(0) == 's'){
        count++;
    }
}
S...println(count + " words start with s");