Java: An Eventful Approach

Chapter 16

* Adapted from Lecture Notes by K.B. Bruce, A.P. Danyluk, and T.P. Murtagh

Textual Data

Many computer applications manipulate textual data

- word processors
- web browsers
- online dictionaries

Java’s String Class

- in simplest form, just quoted text
  "This is a string"
  "So is this"
  "hi"
- used as parameters to
  - Text constructor
  - System.out.println

Strings are Objects

- String is a class, not a primitive type
- Java provides many methods for manipulating them
- compare with equals method
- find length with length method

Manipulating Strings

- Java also provides String literals and + operator
  - special features because strings used in many programs

The Empty String

- smallest possible string
- made up of no characters at all (length is 0)
- ""
- typically used when we want to build something from nothing
Building a String "From Nothing"

Example Morse code
- Allow user to display a series of dots and dashes
- Long mouse click signifies dash
- Short click signifies dot

```java
private String currentCode = " ";
• currentCode is empty until user begins to enter dots and dashes
```

Long Strings

- Strings can be arbitrarily long
  - String chapter in your Java text can be 1 big string
- Practical issue for long strings: Readability
  - Might want line breaks in a string
  - newline character 'n'

Example. Let's add instructions to the Morse Code program

Morse Code Instructions

This program will allow you to enter a message in Morse Code.

To enter your message:
Click the mouse quickly to generate a dot;
Depress the mouse longer to generate a dash.

Printing Instructions

1. Series of 5 System.out.println instructions, or
2. Define String constant INSTRUCTIONS; print INSTRUCTIONS

```java
private static final String INSTRUCTIONS =
"This program will allow you to enter a message in Morse code.

To enter your message:
Click the mouse quickly to generate a dot;
Depress the mouse longer to generate a dash."
```

Note "n" just has length one!!

Readability and Legality

Java does not allow us to write a String literal with actual line breaks in it!

```java
System.out.println("The message that you have entered contains characters that cannot be translated.");
```

is illegal

```java
System.out.println("The message that you have entered contains " +
characters that cannot be translated.");
```

is legal

Many String Methods

- `someString.length()` returns an int that is number of characters in someString
- `someString.endsWith( otherString )` returns true if and only if otherString is a suffix of someString
- `someString.startsWith( otherString )` returns true if and only if otherString is a prefix of someString
More Useful Methods

- Example. Web browsers offer automatic address completion
  I type "http://www.a"
  My browser suggests "http://www.aol.com"
- Keep track of URLs typed in by users
- Use this to provide suggestions

Finding a Substring

- `someString.indexOf( otherString )`
  - think of otherString as a pattern to be found
  - returns an int giving first index in someString
    where otherString found
- Example. if sentence is
  "Strings are objects in Java."
  and pattern is "in", then
  sentence.indexOf(pattern)
  returns 3.

Using indexOf to find URLs

```java
public boolean contains( String aURL ) {
    // Look for URL terminated by newline separator
    return urlString.indexOf( aURL + "\n" ) >= 0;
}
```

Why must we add newline to the URL to be found?

Another indexOf

- `someString.indexOf( pattern, startIndex )`
  - Searches for pattern in someString, beginning at index given by
    startIndex
- If someString is
  "Strings are objects in Java."
  and pattern is "ing", then
  `someString.indexOf( pattern, 0 )`
  returns 3
  `someString.indexOf( pattern, 5 )`
  returns -1
  `someString.indexOf( "in", 5 )`
  returns 20

Case Sensitivity

```java
someString.indexOf( "IN" )
yields -1
if someString is
"Strings are objects in Java."
```
Dealing with Lower and Upper Case

• sometimes useful and important to distinguish between lower and upper case
• sometimes not
if "http://www.cs.williams.edu" in our history
surely we want to recognize
"HTTP://www.cs.williams.edu"
as the same

Note: part of URL after domain name may be case sensitive. Will ignore that here.

Methods for Handling Case

• someString.equalsIgnoreCase( otherString )
returns true if someString and otherString are composed of the same sequence of characters
ignoring diffs in case
• someString.toLowerCase()
returns a copy of someString with upper case chars replaced by lower case
• someString.toUpperCase()

Improving our contains method

// Return true if and only if the history contains the given URL
public boolean contains( String aURL ) {
String lowerUrlString = urlString.toLowerCase();
// Look for URL terminated by newline separator
return lowerUrlString.indexOf( aURL.toLowerCase() + "\n" ) >=0;
}

Alternative: Maintain URL History in lower case

Cutting and Pasting

• can paste strings together with concatenation operator (+)
• can also extract substrings
• somestring.substring( startIndex, endIndex )
returns substring of somestring beginning at startIndex and up to, but not including, endIndex
Ex. If urlString is "http://www.cs.williams.edu"
urlString.substring( 7, 10 )
returns "www" and
urlString.substring( 0, 7 )
returns "http://" and
urlString.substring( 7, urlString.length() )
returns "www.cs.williams.edu."

Rules for substring

• startIndex must be a valid index in the string
• endIndex may not be greater than the length of someString

Will use substring to help us find URL completions
• Let prefix be URL entered so far.
• Use indexOf to find prefix in urlString
• Extract full URL from urlString (up to newline)
• Add full URL to list of all possible completions.
Trimming Strings

- often want to ignore leading and trailing blanks in a string
  "http://www.cs.williams.edu"
  vs.
  "http://www.cs.williams.edu  
- `someString.trim()`
  returns a copy of `someString` with white space removed from both ends

Comparing Strings

- `equals` and `equalsIgnoreCase`
- `someString.compareTo(anotherString)` returns
  - 0, if `someString` and `anotherString` are equal
  - positive int, if `someString` appears after `anotherString` in lexicographic ordering
  - negative int, if `someString` appears before `anotherString` in lexicographic ordering

Lexicographic Ordering

if
- 2 strings are made up of alphabetic characters and
- both all lower case or upper case
then
  lexicographic ordering = alphabetical ordering

StringBuffer

- Java Strings are immutable.
- StringBuffer is essentially a mutable String
- Various ways to construct them
  // empty with initial capacity 1000
  StringBuffer urlStringBuffer = new StringBuffer(1000);
  // create StringBuffer from existing String
  StringBuffer urlStringBuffer = new StringBuffer(urlString);
- Many useful methods (append, replace, delete)
- Some String methods missing (toLowerCase, toUpperCase)

Characters

- Strings are sequences of characters
- Java data type `char` represents characters
- a primitive data type
- char literal written by putting character in single quotes
  'a', 'A', '?', '?', 'n'
Note: these are not the same as
  "a", "A", "?", "?", "n"

Declaration and Use

- To declare variable letter of type char
  char letter;
- chars in Java represented internally as integers
- can perform arithmetic operations on them
- can compare them with operators like < and >
1. Determine whether a char represents a digit in the range 0-9.
   ```java
   if (mysteryChar >= '0' && mysteryChar <= '9')
   works because integers representing '0' to '9' are consecutive numbers
   ```

2. Determine whether mysteryChar is a lower-case alphabetical character
   ```java
   if (mysteryChar >= 'a' && mysteryChar <= 'z')
   works because ints representing 'a' to 'z' are consecutive
   ```

### Extracting chars from Strings

- `aString.charAt(index)` returns the char at the specified index in aString
- If `aString` is "Coffee", then `aString.charAt(1)` returns '0'
- common use for charAt: check whether the characters in a string have some property

### Using charAt

#### Consider a medical record management program

- Want to treat weight as an int
- If `weightField` is the weight text field:
  ```java
  String weight = weightField.getText();
  int weightValue = Integer.parseInt(weight);
  But this only works if weight entered looks like an int
  ```

### Checking for Integer Conversation

Valid: "154", "016"
Not valid: "154lbs", "12"

```java
public boolean validInt(String number) {
    for (int i = 0; i < number.length(); i++) {
        char digit = number.charAt(i);
        if (digit < '0' || digit > '9') {
            return false;
        }
    }
    return true;
}
```

### Operations on chars

- ability to perform arithmetic on chars can be extremely useful.

**Example. A program that will translate a message into Morse code.**
- Make it simple: alphabetic messages only
- Assume all characters upper case.
Translating to Morse Code

I LOVE JAVA
.. .-. . --- ...- . .--- .- ...- . -

High-level Translation

// Converts an alphabetic string into Morse Code
public String toMorseCode( String message ) {
    String morseMessage = "";
    for (int i = 0; i < message.length(); i++) {
        char letter = message.charAt(i);
        if (letter == ' ') {
            morseMessage = morseMessage + WORD_SPACE;
        } else {
            morseMessage = morseMessage + morseCode(letter) + " ";
        }
    }
    return morseMessage;
}

How Does morseCode work?

• look up code in array
• would be convenient if int value of 'A'
  was 0, but it isn't
  – can calculate appropriate index!
      [letter - 'A']
  – if letter is 'A', gives 0
  – if letter is 'B', gives 1
  etc.

Translating a Character to Morse Code

// Returns the sequence of dots and dashes corresponding to
// a letter of the alphabet
public String morseCode( char letter ) {
    return letterCode[letter - 'A'];
}

Chapter Review

• Java provides String literals and + operator
• But Strings are objects!
• Many useful methods
  – equals, equalsIgnoreCase
  – compareTo
  – toUpperCase, toLowerCase
  – indexOf
  – substring
  – trim
  – startsWith, endsWith
  and many others

char

• allows us to manipulate characters
• written as individual characters between
  single quotes
• represented internally as integers - can
  perform arithmetic on them