Burglar

An Oak Hill community couple discovered a thief in their home Saturday after a man told a joke and heard a laugh upstairs.
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
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
Multi-line 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.
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
Printing Instructions

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

private static final String INSTRUCTIONS =
    "This program will allow you to enter a message in Morse code.\n" +
    "\n" +
    "To enter your message:\n" +
    "Click the mouse quickly to generate a dot;\n" +
    "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!

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

is illegal

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

is legal
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
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`
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.
If sentence is
   "Strings are objects in Java."
and pattern is "primitive type", then
   sentence.indexOf(pattern)
returns -1
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
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
Cut it up

• How would I turn all words in 1 string into an array of strings, where each element is 1 word?
Case Sensitivity

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()`
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', '?', '7', '\n'

Note: these are *not* the same as

- "a", "A", "?", "7", "\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.
   
   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
   
   if ( mysteryChar >= 'a' && mysteryChar <= 'z')
   works because ints representing 'a' to 'z' are consecutive
Constructing Strings from chars

- can build a String from char components
  
  new String (characterArray)

- If example is the array of char

  `'a'    'n'    ' '    'e'    'x'    'a'    'm'    'p'    'l'    'e'`

  then

  String aString = new String(example);

  creates the String

  "an example"
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 'o'
- 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
• Given a string, how could we construct a character array?