Chapter 12 Strings

12.1 A String is…

- In computer science, a string is a sequence of characters (letters, numerals, punctuation marks etc.) such as might be entered at a keyboard or printed on a page.
- In Java, *String* is a class that is provided as part of the language. Each *String* object contains a sequence of characters (of type *char*).

12.2 Creating Java Strings

- A String is constructed by placing the desired characters between double-quotes, as in:

```
String name = "Joe";
```

- Two Strings can be concatenated (put together) by placing a plus-sign '+' between them:

```
String name = "Joe";
String fullName = name + " Smith";
```

fullName contains "Joe Smith". On the line above, the blank space immediately in front of Smith is important. Without this space, *fullname* would contain "JoeSmith" (no space).

- Any variable can be concatenated with a *String*:

```
Color darkBlue = new Color(0, 0, 50);
String description = "the variable darkBlue is " + darkBlue;
```

description contains "*the variable darkBlue is java.awt.Color[r=0,g=0,b=50]*".

Any variable will produce a *String* when concatenated with a *String*. The built-in types generally produce reasonably descriptive strings. Objects created from programmer created classes will produce quite uninformative strings unless a *toString* method is supplied. For examples of programmer created *toString* methods, see sections 4.6 and 10.1.

12.3 Breaking up Strings

Using the *StringTokenizer* class

Java contains the class *StringTokenizer* (in *java.util*) which is intended to break a string up into pieces (such as words). The *StringTokenizer* class can be used as long as each token (piece we are interested in) is separated from the next by at least one character that can be discarded. This is generally the case if we wish to extract words.

```
String s = "Apples, bananas and cherries.";
StringTokenizer st = new StringTokenizer(s, " ,.");
while (st.hasMoreTokens())
    {
    String token = st.nextToken();
    System.out.println(token);
    }
```

The example prints out the four words *Apples bananas and cherries*, each on a separate line. The string `" ,."` used in the construction of the tokenizer contains the delimiters, characters that separate words and are to be discarded. If you were to use this example to extract words typed by someone else, you would almost certainly use a larger delimiter set. For example: `" ,.!?()[]\n\r\t"`. Note the space at the beginning, this is required since we want to discard spaces. The symbols `\n`, `\r` and `\t` represent

commonly occurring invisible characters. The first two are alternate ways of expressing the break between lines (Java prefers `\n`, but text not created by Java may well contain `\r` and discarding it will cause no problems). The symbol `\t` represents the tab character.

Methods for breaking up Strings yourself

Sometimes, you must program the extraction of tokens. To do this you will need some of the following methods:

- The length of a *String* can be obtained by the *length* method.

```
String a = "Hello";
a.length()..... 5          (the number of characters in a)
```

- A single character can be obtained from a *String* with the *charAt* method. For the purposes of using *charAt* the characters of a *String* lie in the range *0…length–1*.

```
String a = "Hello";
a.charAt(0)..... 'H'       (the first position is 0)
a.charAt(4)..... 'o'       (the last position is a.length() – 1)
```

  The example below uses *length* and *charAt* to create a copy of the string *s* without the digits (0…9).

```
String s = "The 5 elephants and 17 monkeys ran for 8 hours.";
String noDigits = "";
for (int n = 0; n < s.length(); n++)
   if (s.charAt(n) < '0' || s.charAt(n) > '9')
      noDigits = noDigits + s.charAt(n);
```

  The result is:

```
   noDigits....."The  elephants and  monkeys ran for  hours."
```

- A portion of a *String* can be obtained by using the *substring* method. The positions between the characters are numbered as shown. It is an error to use a position outside of the range *0…length of string*.

```
String a = "Hello";
positions:  H  e  l  l  o
           ^  ^  ^  ^  ^  ^
           0  1  2  3  4  5

a.substring(1, 3)..... "el"      (between positions 1 and 3)
a.substring(2, 2)..... ""        (empty string — between 2 and 2)
a.substring(2)..... "llo"        (from 2 to the end)
a.substring(0, a.length())..... "Hello"  (between 0 and 5)
```

- The location in a *String* of the first (last) occurrence of another *String* (or a single *char*) can be found with the *indexOf* (*lastIndexOf*) method.

```
String a = "Hello, Helen and Herb!";
```

  positions: *Hello* starts at 0, *Helen* starts at 7, *and* starts at 13, *Herb* starts at 17

```
a.indexOf('e').....1          (the location of the 'e' in Hello)
a.indexOf("Hel")..... 0       (the start of the 'Hel' in Hello)
a.indexOf("Her")..... 17      (the start of the 'Her' in Herb)
a.lastIndexOf('e').....18     (the location of the 'e' in Herb)
a.lastIndexOf("Hel").....7    (the start of the 'Hel' in Helen)
a.indexOf("Heaven").....-1    (-1 is returned when no match is found)
a.lastIndexOf("Heaven")..... -1  (-1 is returned when no match is found)
```

Do-it-yourself extraction example

The *StringTokenizer* class cannot be used when tokens are adjacent. To analyze a *String* with both words and punctuation marks considered as tokens to be found, you must split up the *String* yourself. When the method *extractionExample* below analyses the *String* "Apples, bananas and cherries." of the example above, it should print out 6 items <Apples> <,> <bananas> <and> <cherries> <.>.

```
private boolean isALetter(char ch)
    {
    return ('a' <= ch && ch <= 'z') || ('A' <= ch && ch <= 'Z');
    }

private void extractionExample()
    {
    String s = "Apples, bananas and cherries.";
    String ss = s;
    while (ss.length() > 0)
        {
        ss = ss.trim();                              // throw out leading blanks*
        if (ss.length() > 0)
            {
            if (isALetter(ss.charAt(0)))             // the first character is a letter
                {
                int i = 1;
                while (i < ss.length() &&            // find the first non-letter**
                isALetter(ss.charAt(i)))
                i++;
                String token = ss.substring(0, i); // select characters to the non-letter
                System.out.println(token);           // print out the token
                ss = ss.substring(i);                // discard characters in the token
                }
            else                                     // the first character is not a letter
                {
                String token = "" + ss.charAt(0);  // select one character
                System.out.println(token);           // print out the token
                ss = ss.substring(1);                // discard one character
                }
            }
        }
    }
```

* The method *trim* removes 'white space' from both the beginning and the end of the string. The term 'white space' includes spaces, tabs and line separators.

** Checking for the length of the *String* is necessary because an attempt to use *charAt* with an index value greater than the length will cause an error.

Obtaining numbers from strings

Strings such as "23", "5.25" and "–1" clearly represent numbers. Java does not see these as numbers. To have Java recognize a *String* as a number, Java must be asked to convert the *String* to an *int* or *double* (or similar appropriate form). Examples of this conversion process appear in sections 3.8 & 9.4.2.