refere~1.mkr Page 35 Tuesday, July 14, 1998 8:48 PM

2.3.3 Comparing Strings

Use *equals* and *compareTo* to perform string comparison.

Since the basic assignment operator works for `String`s, it is tempting to assume that the relational and equality operators also work. This is not true.

In accordance with the ban on operator overloading, relational operators (`<`, `>`, `<=`, and `>=`) are not defined for the `String` object. Further, `==` and `!=` have the typical meaning for reference variables. For two `String` objects `lhs` and `rhs`, for example `lhs==rhs` is `true` only if `lhs` and `rhs` refer to the same `String` object. Thus, if they refer to different objects that have identical contents, `lhs==rhs` is `false`. Similar logic applies for `!=`.

To compare two `String` objects for equality, we use `equals`. `lhs.equals(rhs)` is `true` if `lhs` and `rhs` reference `String`s that store identical values.

A more general test can be performed with the method `compareTo`. `lhs.compareTo(rhs)` compares two `String` objects, `lhs` and `rhs`. It returns a negative number, zero, or a positive number, depending on whether `lhs` is lexicographically less than, equal to, or greater than `rhs`, respectively.

2.3.4 Other `String` Methods

Use *length*, *charAt*, and *substring* to compute string length, get a single character, and get a substring, respectively.

The length of a `String` object (an empty string has length zero) can be obtained with the method `length`. Since `length` is a method, parentheses are required.

Two methods are defined to access individual characters in a `String`. The method `charAt` gets a single character by specifying a position (the first position is position 0). The method `substring` returns a reference to a newly constructed `String`. The call is made by specifying the starting point and the first nonincluded position.

Here is an example of these three methods:

```
String greeting = "hello";
int len    = greeting.length( );          // len is 5
char ch    = greeting.charAt( 1 );        // ch  is 'e'
String sub = greeting.substring( 2, 4 );  // sub is "ll"
```

2.3.5 Converting between Strings and Primitive Types

toString converts primitive types (and objects) to a `String`.

An appropriate method `toString` can be used to convert any primitive type to a `String`. As an example, `Integer.toString(45)` returns a reference to the newly constructed `String` `"45"`. Many objects also provide an implementation of `toString`. In fact, when operator `+` has only one `String` argument, the non`String` argument is converted to a `String` by automatically applying `toString`. For the integer types, an alternative form of `toString` allows the specification of a radix. Thus