Lecture 27:

this reference

toString() method

Strings as objects

Inner classes

Apr 7 2015
The **this** reference

You can refer to any field (member) of the current object from within a method or a constructor by using the keyword `this`.

```java
public class Point {
    public int x = 0;
    public int y = 0;

    Point(int x, int y) { //constructor
        this.x = x;
        this.y = y;
    }
}
```
The **this** reference

From within a constructor, you can use this keyword to call another constructor in the same class. This is called **explicit constructor invocation**. If present, the invocation of another constructor must be the *first line* in the constructor.

```java
public class Rectangle {
    private int x, y;
    private int width, height;
    public Rectangle() {
        this(0, 0, 1, 1);
    }
    public Rectangle(int width, int height) {
        this(0, 0, width, height);
    }
    public Rectangle(int x, int y, int width, int height) {
        this.x = x; this.y = y;
        this.width = width; this.height = height;
    }
    ...
}
```
The `toString()` method

The `toString()` method returns the string representation of the object.

If you print any object, java compiler internally invokes the `toString()` method on the object.

By overriding the `toString()` method, it returns the desired output, i.e. in your own format. (Instead of the default, which usually returns the reference value!)
The **String** Class

Constructing a String:

```java
String message1 = "Welcome to Java";
String message2 = new String("Welcome to Java");
String s = new String();
```

What string objects can do:

- Obtaining String length and Retrieving Individual Characters in a string
- String Concatenation (concat)
- Substrings (substring(index), substring(start, end))
- Comparisons (equals, compareTo)
- String Conversions
- Finding a Character or a Substring in a String
- Conversions between Strings and Arrays
- Converting Characters and Numeric Values to Strings
Strings are **Immutable**

A String object is **immutable**, meaning its contents cannot be changed once the string has been created.

The following code DOES NOT change the contents of any string objects; it just changes which string object is *referenced* by `s`:

```java
String s = "Java";
s = "HTML";
```
Interned Strings

Since strings are immutable and are frequently used, to improve efficiency and save memory, the JVM uses a unique instance for string literals with the same character sequence. Such an instance is called interned.
Examples (of String variables and the objects they reference)

String s1 = "Welcome to Java";
String s2 = new String("Welcome to Java");
String s3 = "Welcome to Java";
System.out.println("s1 == s2 is " + (s1 == s2));
System.out.println("s1 == s3 is " + (s1 == s3));

displays:
   s1 == s2 is false
   s1 == s3 is true

A new object is created if you use the new operator.
If you use the string initializer, no new object is created if the interned object is already created.
# String Comparisons

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+equals(s1: Object): boolean</code></td>
<td>Returns true if this string is equal to string s1.</td>
</tr>
<tr>
<td><code>+equalsIgnoreCase(s1: String): boolean</code></td>
<td>Returns true if this string is equal to string s1 case-insensitive.</td>
</tr>
<tr>
<td><code>+compareTo(s1: String): int</code></td>
<td>Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than s1.</td>
</tr>
<tr>
<td><code>+compareToIgnoreCase(s1: String): int</code></td>
<td>Same as compareTo except that the comparison is case-insensitive.</td>
</tr>
<tr>
<td><code>+regionMatches(toffset: int, s1: String, offset: int, len: int): boolean</code></td>
<td>Returns true if the specified subregion of this string exactly matches the specified subregion in string s1.</td>
</tr>
<tr>
<td><code>+regionMatches(ignoreCase: boolean, toffset: int, s1: String, offset: int, len: int): boolean</code></td>
<td>Same as the preceding method except that you can specify whether the match is case-sensitive.</td>
</tr>
<tr>
<td><code>+startsWith(prefix: String): boolean</code></td>
<td>Returns true if this string starts with the specified prefix.</td>
</tr>
<tr>
<td><code>+endsWith(suffix: String): boolean</code></td>
<td>Returns true if this string ends with the specified suffix.</td>
</tr>
</tbody>
</table>
String Comparison \texttt{s1.equals(s2)}

String \texttt{s1} = new String("Welcome");
String \texttt{s2} = "welcome";
if (s1.equals(s2)) {
    // \texttt{s1} and \texttt{s2} have the same contents
}
if (s1 == s2) {
    // \texttt{s1} and \texttt{s2} have the same reference
String Comparison `s1.compareTo(s2)`

```java
String s1 = new String("Welcome");
String s2 = "welcome";
if (s1.compareTo(s2) > 0) {
    // s1 is greater than s2
}
else if (s1.compareTo(s2) == 0) {
    // s1 and s2 have the same contents
}
else
    // s1 is less than s2
```

Length, Characters, and Combining Strings

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>Returns the number of characters in this string.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+length(): int</code></td>
<td>Returns the character at the specified index from this string.</td>
</tr>
<tr>
<td><code>+charAt(index: int): char</code></td>
<td>Returns a new string that concatenate this string with string s1.</td>
</tr>
<tr>
<td><code>+concat(s1: String): String</code></td>
<td></td>
</tr>
</tbody>
</table>