

G5100P



6. Strings

Object Oriented Programming

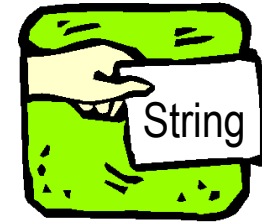
Colin Higgins

Strings

- This section illustrates how to manipulate character data using the `String` and `StringBuilder` classes. It also teaches you about accessor methods and how the compiler uses `Strings` and `StringBuilders` behind the scenes.
- Why Two String Classes?
- The Java Dev. Env. provides two classes that store and manipulate character data:
 - **`String`**, for constant strings, and
 - **`StringBuilder`**, for strings that can change.
- To creating `Strings` and `StringBuilders` :
- Same way for creating an object of any type. E.g.

```
StringBuilder dest = new StringBuilder( len );
```

The String Class



- A character string in Java is an object, defined by the `String` class

```
String name = new String ("Hello World");
```

- Because strings are so common, Java includes them in its syntax:

```
String name = "Ken Arnold";
```

- Java strings are *immutable*; once a string object has a value, it cannot be changed

The String Class

- A character in a string can be referred to by its position, or *index*
- The index of the first character is zero
- The `String` class is defined in the `java.lang` package (and is implicitly imported by default)
- Many helpful methods are defined in the `String` class such as :
 - `length()`
 - `equals(String)`
 - `startsWith(String)`
 - `toUpperCase()`
 - `toLowerCase()`
 - `indexOf(String)`
 - `substring(int begin)`
 - `substring(int begin, int end)`
 - `charAt(int index)`

Example : ReverseString

- The reverseIt method accepts an argument of type String called source that contains the String data to be reversed. The method creates a StringBuilder, dest, the same size as source. It then loops backwards over all the characters in source and appends them to dest, thereby reversing the String. Finally, the method converts dest, a StringBuilder, to a String.

Example : ReverseString

```
public class ReverseString {  
  
    public static String reverseIt( String source ) {  
        int len = source.length();  
        StringBuilder dest = new StringBuilder( len );  
        for ( int i = len - 1; i >= 0; i-- ) {  
            dest.append( source.charAt( i ) );  
  
        }  
        return dest.toString();  
    } // end method reverseIt  
} // end ReverseString
```

Accessor Methods

- The `reverse` method uses two accessor methods to obtain information about the source string: `charAt` and `length`.
- The `reverse` method uses `StringBuilder`'s `append` method to add characters to `dest`. In addition to `append`, `StringBuilder` provides methods to insert characters into the buffer, modify a character at a specific location within the buffer, and so on.
- `reverse` converts the resulting `StringBuilder` to a `String` and returns the string.
- You can convert several different data types to `Strings` using `String`'s `valueOf` method.
- You can also use methods from the `Integer`, `Float`, `Double`, and `Long` classes to convert the contents of a `String` to a number.

Accessor Methods

- As we know, an object's instance variables are encapsulated within the object, hidden inside, safe from inspection or manipulation by other objects.
- With certain well-defined exceptions, the object's methods are the only means by which other objects can inspect or alter an object's instance variables.
- The reverseIt method uses two of String's accessor methods to obtain information about the source string.

```
class ReverseString {  
    public static String reverseIt( String source ) {  
        int i, len = source.length() ;  
        StringBuilder dest = new StringBuilder( len );  
        for ( i = ( len - 1 ); i >= 0; i-- ) {  
            dest.append( source.charAt( i ) );  
        }  
        return dest.toString();  
    }  
}
```


Accessor Methods

- First, reverselt uses String's length accessor method to obtain the length of the String source.

```
int len = source.length();
```

- Note that reverselt doesn't care if String maintains its length attribute as an integer, as a floating point number, or even if String computes its length on the fly.
- reverselt simply relies on the public interface of the length method, which returns the length of the String as an integer. That's all reverselt needs to know.
- Second, reverselt uses the charAt accessor, which returns the character at the position specified in the parameter i (`source.charAt(i)`)
- The character returned by charAt is then appended to the StringBuilder dest.
- Since the loop variable i begins at the end of source and proceeds backwards over the string, the characters are appended in reverse order to the StringBuilder, thereby reversing the string.

Accessor Methods for String objects

- In addition to length and charAt, String supports a number of other accessor methods that provide access to substrings and the indices of specific characters in the String.

```
public class StringExample {  
    public static void main(String[] argv) {  
        String str = new String("my name is Thanassis");  
        System.out.println("The string contains:" + str );  
        System.out.println("its length is :" + str.length() + " characters");  
        System.out.println("uppercase :" + str.toUpperCase());  
        System.out.println("lowercase :" + str.toLowerCase());  
        System.out.println("-is- starts at :" + str.indexOf("is") );  
        if (!str.equals("my name is Thanasis")) System.out.println("not true");  
    }  
}
```

```
The string contains:my name is Thanassis  
its length is :20 characters  
uppercase :MY NAME IS THANASSIS  
lowercase :my name is thanassis  
-is- starts at :8  
not true
```

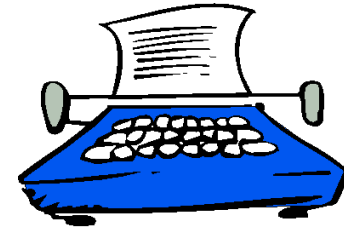
Strings and the Java Compiler

- The Java compiler uses Strings and StringBuilders behind the scenes to handle literal strings and concatenation.
- String and StringBuilder provide several other useful ways to manipulate string data, including concatenation, comparison, substitution, and conversion to upper and lower case.
- `java.lang.String` and `java.lang.StringBuilder` summarise and list all of the methods and variables supported by these two classes.

Using Strings vs Using StringBuilders

- You use Strings when you don't want the value of the String to change. For example, if you write a method that requires String data and the method is not going to modify the string in any way, use a String object.
- Typically, you'll want to use Strings to pass character data into methods and return character data from methods. The `reverse` method takes a String argument and returns a String value.
- The `StringBuilder` class provides for strings that will be modified; you use `StringBuilders` when you know that the value of the character data will change.
- You typically use `StringBuilders` for constructing character data, as in the `reverse` method.
- Because they are constants, Strings are typically cheaper than `StringBuilders` and they can be shared. So it's important to use Strings when they're appropriate.

Creating StringBuilders



- The bold line in the reverseIt method creates a StringBuilder named dest whose initial length is the same as source.

```
class ReverseString {  
    public static String reverseIt( String source ) {  
        int i, len = source.length();  
        StringBuilder dest = new StringBuilder( len );  
        for ( i = ( len - 1 ); i >= 0; i-- ) {  
            dest.append( source.charAt( i ) );  
        }  
        return dest.toString();  
    }  
}
```

- The code `StringBuilder dest` declares to the compiler that `dest` will be used to refer to an object whose type is `StringBuilder`, the `new` operator allocates memory for a new object, and `StringBuilder(len)` initialises the object. These three steps - declaration, instantiation, and initialisation are - described in [Creating Objects](#).

Modifying StringBuilders



- The `reverseIt` method uses `StringBuilder`'s `append` method to add a character to the end of the destination string: `dest`.

```
class ReverseString {  
    public static String reverseIt( String source ) {  
        int i, len = source.length();  
        StringBuilder dest = new StringBuilder( len );  
        for ( i = ( len - 1 ); i >= 0; i-- ) {  
            dest.append( source.charAt( i ) );  
        }  
        return dest.toString();  
    }  
}
```

Modifying StringBuilders

- If the appended character causes the size of the StringBuilder to grow beyond its current capacity, the StringBuilder allocates more memory.
- Because memory allocation is a relatively expensive operation, you can make your code more efficient by initialising a StringBuilder's capacity to a reasonable first guess, thereby minimising the number of times memory must be allocated for it.
- For example, the `reverseIt` method constructs the StringBuilder with an initial capacity equal to the length of the source string, ensuring only one memory allocation for `dest`.
- The version of the `append` method used in `reverseIt` is only one of the StringBuilder methods that appends data to the end of a StringBuilder.
- There are several append methods that append data of various types, such as `float`, `int`, `boolean`, and even `Object`, to the end of the StringBuilder.
- The data is converted to a string before the append operation takes place.

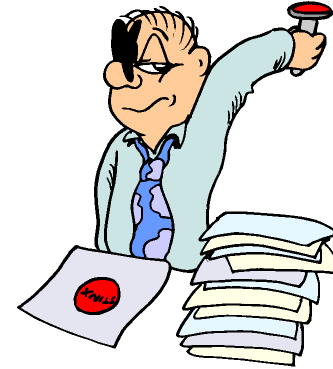
Inserting Characters



- At times, you may want to insert data into the middle of a `StringBuilder`. You do this with one of `StringBuffer`'s insert methods.
- This example illustrates how you would insert a string into a `StringBuilder`:

```
StringBuilder sb = new StringBuilder( "Drink  
Java!" );  
sb.insert( 6, "Hot " );  
System.out.println( sb.toString() );
```
- This code extract prints: Drink Hot Java!
- With `StringBuilder`'s many insert methods, you specify the index before which you want the data inserted. In this example, "Hot " needed to be inserted before the 'J' in "Java". Indices begin at 0, so the index for 'J' is 6.
- To insert data at the beginning of a `StringBuilder`, use an index of 0. To add data at the end of a `StringBuilder`, use an index equal to the current length of the `StringBuilder` or use `append`.

Setting Characters



- Another useful `StringBuilder` modifier is `setCharAt`, which replaces the character at a specific location in the `StringBuilder` with the character specified in the argument list.. `setCharAt` is useful when you want to reuse a `StringBuilder`.

Converting Objects to Strings

- Using the `toString()` method :
- It's often convenient or necessary to convert an object to a `String` because you need to pass it to a method that accepts only `String` values.
- The `reverseIt` method used earlier in this lesson uses `StringBuilder`'s `toString` method to convert the `StringBuilder` to a `String` object before returning the `String`.

```
class ReverseString {  
    public static String reverseIt( String source ) {  
        int i, len = source.length();  
        StringBuilder dest = new StringBuilder( len );  
        for ( i = ( len - 1 ); i >= 0; i-- ) {  
            dest.append( source.charAt( i ) );  
        }  
        return dest.toString() ;  
    }  
}
```

Converting Objects to Strings

- All classes inherit `toString` from the `Object` class and many classes in the `java.lang` package override this method to provide an implementation that is meaningful to that class.
- For example, the "type wrapper" classes - `Character`, `Integer`, `Boolean` etc- all override `toString()` and provide a `String` representation of the object.

The `valueOf` Method

- As a convenience, the `String` class provides the class method `valueOf()`. You can use `valueOf()` to convert variables of different types to `Strings`. For example, to print `x` and `y` next to each other (i.e. `56`):

```
int x = 5;
```

```
int y = 6;
```

```
System.out.println( String.valueOf( x ) + y ); // prints 56
```

```
// compare with :
```

```
System.out.println( x + y ); // prints 11
```

```
// and:
```

```
System.out.println( x + " " + y ); // prints 56
```

Converting Strings to Numbers



- The String class itself does not provide any methods for converting a String to a floating point, integer, or other numerical type.
- However, four of the "type wrapper" classes (Integer, Double, Float, and Long) provide a class method named `valueOf` that converts a String to a that an object of that type.
- Here's a small example of the Float class's `valueOf`:

```
String piStr = "3.14159";  
float pi = Float.valueOf( piStr );  
int i = Integer.parseInt( piStr );
```

trim()!

Concatenation and the + Operator



- In Java, you can use + to concatenate Strings together:

```
String cat = "cat";
```

```
System.out.println( "con" + cat + "enation" );
```

- This is a little deceptive because, as you know, Strings can't be changed. However, behind the scenes the compiler uses StringBuilders to implement concatenation. The above example compiles to:

```
String cat = "cat";
```

```
System.out.println( new StringBuilder().append(  
    "con" ).append( cat ).append( "enation" ) );
```

- You can also use the + operator to append values to a String that are not themselves Strings:

```
System.out.println( "Java's Number " + 1 );
```

- The compiler converts the non-String value (the integer 1 in the example) to a String object before performing the concatenation operation.

Class java.lang.String

```
public final class String extends Object implements Serializable
```

- The String class represents Character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.
- Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

String str = "abc"; is equivalent to:

```
char data[ ] = {'a', 'b', 'c'};
```

```
String str = new String( data );
```

- Here are some more examples of how strings can be used:

```
System.out.println( "abc" );
```

```
String cde = "cde";
```

```
System.out.println( "abc" + cde );
```

```
String c = "abc".substring( 2,3 );
```

```
String d = cde.substring( 1, 2 );
```

String' s Constructors

- `String()` : Allocates a new String containing no characters.
- `String(byte[])` : Construct a new String by converting the specified array of bytes using the platform's default character encoding.
- `String(byte[], int)` : Allocates a new String containing characters constructed from an array of 8-bit integer values. Deprecated.
- `String(byte[], int, int)` : Construct a new String by converting the specified subarray of bytes using the platform's default character encoding.

String' s Constructors

- `String(byte[], int, int, String)` : Construct a new String by converting the specified subarray of bytes using the specified character encoding.
- `String(byte[], String)` : Construct a new String by converting the specified array of bytes using the specified character encoding.
- `String(char[])` : Allocates a new String so that it represents the sequence of characters currently contained in the character array argument.
- `String(char[], int, int)` : Allocates a new String that contains characters from a subarray of the character array argument.
- `String(String)` : Allocates a new string that contains the same sequence of characters as the string argument.
- `String(StringBuilder)` : Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

String' s Methods

- `charAt(int)` : Returns the character at the specified index.
- `compareTo(String)` : Compares two strings lexicographically.
- `concat(String)` : Concatenates the specified string to the end of this string.
- `copyValueOf(char[])` : Returns a String using the specified character array.
- `copyValueOf(char[],int,int)` : Returns a String that is equivalent to the specified character array.
- `endsWith(String)` : Tests if this string ends with the specified suffix.
- `equals(Object)` : Compares this string to the specified object.
- `equalsIgnoreCase(String)` : Compares this String to another object.
- `getBytes()` : Convert this String into bytes according to the platform's default character encoding, storing the result into a new byte array.
- `getBytes(int, int, byte[], int)` : Copies characters from this string into the destination byte array. Deprecated.
- `getBytes(String)` : Convert this String into bytes according to the specified character encoding, storing the result into a new byte array.

String' s Methods

- `getChars(int, int, char[], int)` : Copies characters from this string into the destination character array.
- `hashCode()` : Returns a hashcode for this string.
- `indexOf(int)` : Returns the index within this string of the first occurrence of the specified character.
- `indexOf(int, int)` : Returns the index within this string of the first occurrence of the specified character, starting the search at index.
- `indexOf(String)` : Returns the index within this string of the first occurrence of the specified substring.
- `indexOf(String, int)` : Returns the index within this string of the first occurrence of the specified substring, starting at index.
- `intern()` : Returns a canonical representation for the string object.
- `lastIndexOf(int)` : Returns the index within this string of the last occurrence of the specified character.
- `lastIndexOf(int, int)` : Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.

String' s Methods

- `lastIndexOf(String)` : Returns the index within this string of the rightmost occurrence of the specified substring.
- `lastIndexOf(String, int)` : Returns the index within this string of the last occurrence of the specified substring.
- `length()` : Returns the length of this string.
- `regionMatches(boolean, int, String, int, int)` : Tests if two string regions are equal.
- `regionMatches(int, String, int, int)` : Tests if two string regions are equal.
- `replace(char, char)` : Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.
- `startsWith(String)` : Tests if this string starts with the specified prefix.
- `startsWith(String, int)` : Tests if this string starts with the specified prefix.
- `substring(int)` : Returns a new string that is a substring of this string.
- `substring(int, int)` : Returns a new string that is a substring of this string.
- `toCharArray()` : Converts this string to a new character array.
- `toLowerCase()` : Converts this String to lowercase.

String' s Methods

<code>toLowerCase(Locale)</code>	: Converts all of the characters in this String to lower case using the rules of the given locale.
<code>toString()</code>	: This object (which is already a string!) is itself returned.
<code>toUpperCase()</code>	: Converts this string to uppercase.
<code>toUpperCase(Locale)</code> upper case	: Converts all of the characters in this String to using the rules of the given locale.
<code>trim()</code>	: Removes white space from both ends of this string.
<code>valueOf(boolean)</code>	: Returns the string representation of the boolean argument.
<code>valueOf(char)</code>	: Returns the string representation of the char * argument.
<code>valueOf(char[])</code>	: Returns the string representation of the char array argument.
<code>valueOf(char[], int, int)</code>	: Returns the string representation of a specific subarray of the char array argument.
<code>valueOf(double)</code>	: Returns the string representation of the double argument.
<code>valueOf(float)</code>	: Returns the string representation of the float argument.
<code>valueOf(int)</code>	: Returns the string representation of the int argument.
<code>valueOf(long)</code>	: Returns the string representation of the long argument.
<code>valueOf(Object)</code>	: Returns the string representation of the Object argument.

Summary

- Strings and StringBuilder Intro
- The String Class
- Example : ReverseString
- Accessor Methods
- Strings - StringBuilders and the compiler
- Using String and StringBuilders
- Creating :
 - String
 - StringBuilder
- Modifying StringBuilder objects
 - Inserting Characters
 - Setting Characters
- Converting
 - Objects to Strings
 - Strings to Objects
- The valueOf method
- String Concatenation using +
- String' s
 - Constructors
 - Methods