Strings

• This section illustrates how to manipulate character data using the String and StringBuffer classes. It also teaches you about accessor methods and how the compiler uses Strings and StringBuffers 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
    – StringBuffer, for strings that can change.

• To creating Strings and StringBuffers:
  • Same way for creating an object of any type. E.g.
    ```java
    StringBuffer dest = new StringBuffer(len);
    ```

The String Class

• A character string in Java is an object, defined by the String class
  ```java
  String name = new String("Hello World");
  ```

• Because strings are so common, Java includes them in its syntax:
  ```java
  String name = "Ken Arnold";
  ```

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

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 StringBuffer, 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 StringBuffer, to a String.
Example: ReverseString

```java
public class ReverseString {
  public static String reverseIt(String source) {
    int len = source.length();
    StringBuffer dest = new StringBuffer(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 reverseIt method uses two accessor methods to obtain information about the source string: `charAt` and `length`.
- The reverseIt method uses StringBuffer's `append` method to add characters to dest. In addition to append, StringBuffer provides methods to insert characters into the buffer, modify a character at a specific location within the buffer, and so on.
- reverseIt converts the resulting StringBuffer to a String and returns the string.
- You can convert several different data types to Strings using `String.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.

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

**Strings and the Java Compiler**

- The Java compiler uses Strings and StringBuffers behind the scenes to handle literal strings and concatenation.
- String and StringBuffer 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.StringBuffer summarise and list all of the methods and variables supported by these two classes.
Using Strings vs Using StringBuffers

- 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 reverseIt method takes a String argument and returns a String value.
- The StringBuffer class provides for strings that will be modified; you use StringBuffers when you know that the value of the character data will change.
- You typically use StringBuffers for constructing character data, as in the reverseIt method.
- Because they are constants, Strings are typically cheaper than StringBuffers and they can be shared. So it’s important to use Strings when they’re appropriate.

Creating StringBuffers

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

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

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

Modifying StringBuffers

- The reverseIt method uses StringBuffer’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();
        StringBuffer dest = new StringBuffer( len );
        for ( i = ( len - 1 ); i >= 0; i-- ) {
            dest.append( source.charAt( i ) );
        }
        return dest.toString();
    }
}
```

- If the appended character causes the size of the StringBuffer to grow beyond its current capacity, the StringBuffer allocates more memory.
- Because memory allocation is a relatively expensive operation, you can make your code more efficient by initialising a StringBuffer’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 StringBuffer 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 StringBuffer methods that append data to the end of a StringBuffer.
- There are several append methods that append data of various types, such as float, int, boolean, and even Object, to the end of a StringBuffer.
- 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 StringBuffer. You do this with one of StringBuffer’s insert methods.
- This example illustrates how you would insert a string into a StringBuffer:

```
StringBuffer sb = new StringBuffer( “Drink Java!” );
sb.insert( 6, “Hot ” );
System.out.println( sb.toString() );
```

- This code extract prints: Drink Hot Java!
- With StringBuffer’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 StringBuffer, use an index of 0. To add data at the end of a StringBuffer, use an index equal to the current length of the StringBuffer or use append.

Setting Characters

- Another useful StringBuffer modifier is setCharAt, which replaces the character at a specific location in the StringBuffer with the character specified in the argument list. setCharAt is useful when you want to reuse a StringBuffer.
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 StringBuffers to store the String to be returned.

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

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):

```java
int x = 5;
int y = 6;
System.out.println(String.valueOf(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 an object of that type.

```java
String piStr = "3.14159";
float pi = Float.valueOf(piStr);
```

Concatenation and the + Operator

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

```java
String cat = "cat";
System.out.println("concatenation");
```
- However, behind the scenes, the compiler uses StringBuffers to implement concatenation. The above example compiles to:

```
String cat = "cat";
System.out.println(new StringBuffer("concatenation").append("cat").toString());
```
- You can also use the + operator to append values to a String that are not themselves Strings:

```
String a = "abc".substring(2,3);
String b = cde.substring(1,2);
```
String’s Constructors

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

String’s Methods

- `charAt(int)`: Returns the character at the specified index.
- `compareTo(String)`: Compares this string to another object.
- `concat(String)`: Concatenates the specified string to the end of this string.
- `copyOf(char[])`: Returns a String using the specified character array.
- `copyOf(byte[], int, int, byte[], int, int)`: Copies characters from this string into the destination byte array. Deprecated.
- `equals(Object)`: Compares this string to the specified object.
- `equalsRangeCaseInsensitive(String)`: Compares this string to another object.
- `getBytes(String)`: Converts this String into bytes according to the platform’s default character encoding, storing the result into a new byte array.
- `getBytes()`: Converts this string to a new character array.
- `getBytes(String)`: Converts this String into bytes according to the specified character encoding, storing the result into a new byte array.
- `getBytes(String, int, int, byte[], int)`: Copies characters from this string into the destination byte array. Deprecated.
- `hashCode()`: Returns a hashcode for this string.
- `indexOf(String, int)`: Returns the index within this string of the first occurrence of the specified substring, starting at index.
- `indexOf(int, int)`: Returns the index within this string of the first occurrence of the specified character, starting the search at index.
- `indexOf(int)`: Returns the index within this string of the first occurrence of the specified character, searching backward starting at the specified index.
- `indexOfAll(String, int, int)`: Occurrences of oldChar in this string with newChar.
- `lastIndexOf(String)`: Returns the index within this string of the last occurrence of the specified substring.
- `lastIndexOf(int, int)`: Returns the index within this string of the last occurrence of the specified character, starting the search at index.
- `lastIndexOf(int)`: Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
- `length()`: Returns the length of this string.
- `regionMatches(byte, String, int, int)`: Tests if this string starts with the specified prefix.
- `regionMatches(boolean, String, int, int)`: Tests if this string starts with the specified prefix.
- `regionMatches(int, String, int, int)`: Tests if this string starts with the specified prefix.
- `startsWith(String, int)`: Tests if this string begins with the specified prefix.
- `startsWith(String)`: Tests if this string begins with the specified prefix.
- `toCharArray()`: Converts this string to a new character array.
- `toString()`: This object (which is already a string!) is itself returned.
- `trim()`: Removes white space from both ends of this string.
- `trimToLeading()`: Removes white space from the leading end of this string.
- `trimToTrailing()`: Removes white space from the trailing end of this string.
- `trimToTrailingAndLeading()`: Removes white space from both ends of this string.
- `toUpperCase()`: Converts all of the characters in this String to upper case.
- `toUpperCase(Locale)`: Converts all of the characters in this String to upper case using the rules of the given locale.
- `toLowerCase()`: Converts all of the characters in this String to lower case.
- `toLowerCase(Locale)`: Converts all of the characters in this String to lower case using the rules of the given locale.
- `valueOf(char[])`: Returns the string representation of the char array argument.
- `valueOf(char)`: Returns the string representation of the char argument.
- `valueOf(char[])`: Returns the string representation of the char array argument.
- `valueOf(char)`: Returns the string representation of the char argument.
- `valueOf(double)`: Returns the string representation of the double argument.
- `valueOf(float)`: Returns the string representation of the float argument.
- `valueOf(int)`: Returns the string representation of the int argument.
- `valueOf(long)`: Returns the string representation of the long argument.
- `valueOf(Object)`: Returns the string representation of the Object argument.
Summary

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

From Nick Brunt Oct 2010….