Session 5

Outline

- The **java.lang** package
  - The **Object** class
  - The **Math** class
  - The **Class** class
- The **String** class
- The **StringTokenizer** class

java.lang Package

- It is a core package in Java.
- When classes from this package are referenced there is no need for import statement.
- Contains core set of classes such as:
  - Object
  - Number
  - Math
  - System
  - Class
  - String
  - StringBuffer

Object Class

- Object class is the top of the class hierarchy in Java
  - ***Every class inherits from Object class***
  - Defines some default behavior that is mainly overridden in subclasses

- Commonly overridden methods from Object class are:
  - **toString()**
  - **equals()**
  - **hashCode()**
  - **clone()**

Method equals()

- Meant to return whether or not two objects are equal
  - Default implementation in Object class returns whether or not are objects identical
    - The == operator is used
  - Overriding method allows to change the equality criteria

Example equals() method

- An example of overriding the equals() method in the Policy class
  - Two policies are equal if their policy numbers are equal

```
public boolean equals(Object anObject){
   if ((anObject == null)||(anObject.getClass() != this.getClass()))
        return false;
   Policy policy = (Policy)anObject;
   return getPolicyNumber().equals(policy.getPolicyNumber());
}
```

Method hashCode()

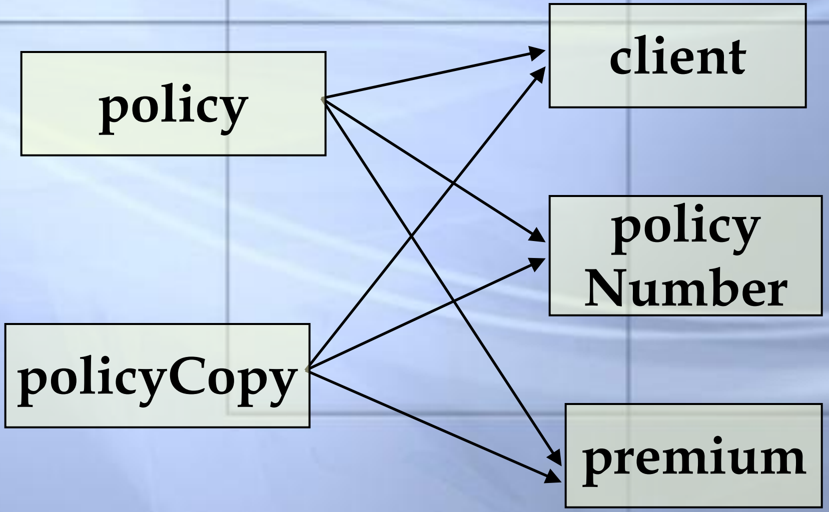
- Used by collections, primarily HashMap and HashSet
  - Returns an int for indexing
  - Hash codes must be identical for objects that are equal
  - For the Policy class implementation of the hash code method could be:

```
public int hashCode(){
   return getPolicyNumber().hashCode();
}
```

Method clone()

- Used to obtain copy of an object.
- The default implementation of the clone() method in Object class does shallow copy.
  - New instance of the class is created, but all containing fields are the same.

```
Policy policy;
policy = new Policy();
Policy policyCopy;
policyCopy = policy.clone();
```



Cloning Rules

- Classes must implement Cloneable interface to allow their instances to be cloned
  - CloneNotSupported exception thrown if objects cannot be cloned
  - In our example, both Client and Policy classes must implement Cloneable interface to support cloning

- Method clone() is protected in Object class
  - Usually made public when overridden to allow use everywhere

java.lang

- **Cloneable**
  - A class implements the Cloneable interface to indicate to the clone method in class Object that it is legal for that method to make a field-for-field copy of instances of that class.
- **SecurityManager**
  - The security manager is an abstract class that allows applications to implement a security policy
- **Exception**
  - The class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable application might want to catch.

java.lang

- **Errors**
  - An Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions

- **ClassLoader**
  - The class ClassLoader is an abstract class. Applications implement subclasses of ClassLoader in order to extend the manner in which the Java Virtual Machine dynamically loads classes.

The Supercomputing People

Corporate Profile

Java.lang.Math

java.lang.Math

- The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.
- **class is final**
- **constructor is private**
- **methods are static**

java.lang.Math (con’t)

- **public static final double E**
  - as close as Java can get to e, the base of natural logarithms
- **public static final double PI**
  - as close as Java can get to pi, the ratio of the circumference of a circle to its diameter

Methods

public static double	abs(double x)
public static native double	atan(double x)
public static native double	ceil(double x)
public static native double	cos(double x)
public static native double	exp(double x)
public static native double	floor(double x)
public static native double	log(double x)
public static native double	max(double x, double y)
public static native double	min(double x, double y)
public static native double	pow(double x, double y)
public static synchronized	double random()
public static long	round(double x)
public static native double	sin(double x)
public static native double	sqrt(double x)
public static native double	tan(double x)

Examples

Math.abs(-13.579)	=>	13.579
Math.floor(13.579)	=>	13.0
Math.ceil(13.579)	=>	14.0
Math.round(13.579)	=>	14
Math. pow(25.0,0.5)	=>	5.0
Math. sqrt(25.0)	=>	5.0
Math. random()	=>	0.5703404356417687

Important classes

- **System**
  - exit to terminate immediately.
  - get environmental variables
  - get time
  - stdin, stdout, stderr handling

- **Runtime**
  - Start new process
  - get total memory and free memory

java.lang.Class

class Class

- Java classes are not objects themselves.
  - They are templates for data and behavior and also factory for creating instances.

- Instances of Class class represent Java classes runtime.
  - They allow developers to find runtime information about the class.

Methods of class “Class”

- **forName**
  - give the Class object when the class name is knows as a string.
- **getName**
  - gives the name of the object.
- **getInterfaces**
  - give the interfaces for that class
- **newInstance**
  - create an object of that class

Methods of class "Class"

- **getFields**
  – get the variables of a class
- **getMethods**
  – get the methods of a class
- **getConstructors**
  – get the constructors of a class.

Example using ClassLoader and Class

```
ClassLoader cl=ClassLoader.getSystemClassLoader();
Class c=cl.loadClass("A");
System.out.println(c.getName());

Field[] f=c.getDeclaredFields();
for(int i=0; i<f.length; i++)
        System.out.println(f[i]);

Method[] m=c.getDeclaredMethods();
for(int i=0; i<m.length; i++)
        System.out.println(m[i]);
```

```
class A
{
    int a,b,c;
    void f1() { }
    void f1() { }
    void f1() { }
}
```

Outputs:

```
A
int A.a
int A.b
int A.c
void A.f1()
void A.f2()
void A.f3()
```

java.lang.String

java.lang.StringBuffer

Fundamentals of Strings

- **String**
  - Series of characters treated as single unit.
  - May include letters, digits, etc.
  - Object of class String

The Supercomputing People

String Constructors

- Class **String**
  - Provides nine constructors

CDAC
The Supercomputing People

StringConstructors.java

Line 25

Line 26

Line 27

```
1     // Fig. 10.1: StringConstructors.java
2     // This program demonstrates the String class constructors.
3
4     // Java extension packages
5     import javax.swing.*;
6
7     public class StringConstructors {
8
9        // test String constructors
10       public static void main( String args[] )
11       {
12          char charArray[] = { 'b', 'i', 'r', 't', 'h', '
13                               'd', 'a', 'y' };
14          byte byteArray[] = { ( byte ) 'n', ( byte ) 'e',
15             ( byte ) 'w', ( byte ) ' ', ( byte ) 'y',
16             ( byte ) 'e', ( byte ) 'a', ( byte ) 'r' };
17
18          StringBuffer buffer;
19          String s, s1, s2, s3, s4, s5, s6, s7, output;
20
21          s = new String( "hello" );
22          buffer = new StringBuffer( "Welcome to Java Progr
23
24          // use String constructors
25          s1 = new String();
26          s2 = new String( s );
27          s3 = new String( charArray );
28          s4 = new String( charArray, 6, 3 );
29          s5 = new String( byteArray, 4, 4 );
30          s6 = new String( byteArray );
31          s7 = new String( buffer );
32
```

String default constructor instantiates *empty string*

Constructor copies String

Constructor copies character array

Constructor copies character-array subset

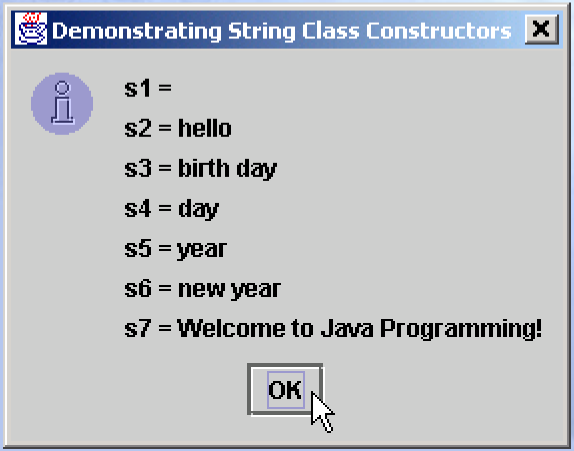
Constructor copies byte array

Constructor copies byte-array subset

Constructor copies StringBuffer

StringConstructors.java

```
        // append Strings to output
        output = "s1 = " + s1 + "\ns2 = " + s2 + "\ns3 = " + s3 +
           "\ns4 = " + s4 + "\ns5 = " + s5 + "\ns6 = " + s6 +
           "\ns7 = " + s7;

        JOptionPane.showMessageDialog( null, output,
           "Demonstrating String Class Constructors",
           JOptionPane.INFORMATION_MESSAGE );

        System.exit( 0 );
     }

  }  // end class StringConstructors
```



```
s1 =
s2 = hello
s3 = birth day
s4 = day
s5 = year
s6 = new year
s7 = Welcome to Java Programming!
```

String Methods length, charAt and getChars

- Method **length**
  - Determine **String** length
    - Like arrays, **String**s always "know" their size
    - Unlike array, **String**s do not have length instance variable

- Method **charAt**
  - Get character at specific location in **String**

- Method **getChars**
  - Get entire set of characters in **String**

```
1  // Fig. 10.2: StringMiscellaneous.java
2  // This program demonstrates the length, charAt and getChars
3  // methods of the String class.
4  //
5  // Note: Method getChars requires a starting point
6  // and ending point in the String. The starting point is the
7  // actual subscript from which copying starts. The ending point
8  // is one past the subscript at which the copying ends.
9
10 // Java extension packages
11 import javax.swing.*;
12
13 public class StringMiscellaneous {
14
15    // test miscellaneous String methods
16    public static void main( String args[] )
17    {
18       String s1, output;
19       char charArray[];
20
21       s1 = new String( "hello there" );
22       charArray = new char[ 5 ];
23
24       // output the string
25       output = "s1: " + s1;
26
27       // test length method
28       output += "\nLength of s1: " + s1.length();
29
30       // loop through characters in s1 and display reversed
31       output += "\nThe string reversed is: ";
32
33       for ( int count = s1.length() - 1; count >= 0; count-- )
34          output += s1.charAt( count ) + " ";
```

StringMiscellaneous.java

Line 28

Line 33

Determine number of characters in String s1

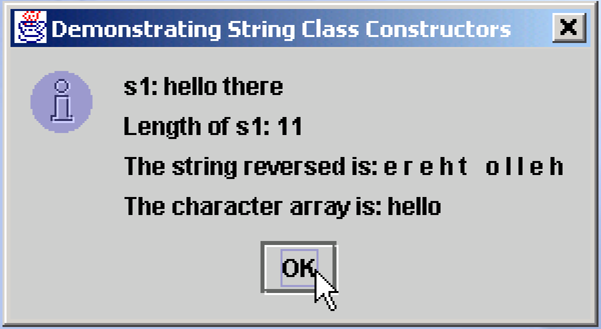
Append s1's characters in reverse order to String output

```
36         // copy characters from string into char array
37         s1.getChars( 0, 5, charArray, 0 );
38         output += "\nThe character array is: ";
39
40         for ( int count = 0; count < charArray.length; count++ )
41            output += charArray[ count ];
42
43         JOptionPane.showMessageDialog( null, output,
44            "Demonstrating String Class Constructors",
45            JOptionPane.INFORMATION_MESSAGE );
46
47         System.exit( 0 );
48      }
49
50   }  // end class StringMiscellaneous
```

Copy (some of) s1's characters to charArray

StringMiscellaneous.java

Line 37



```
s1: hello there
Length of s1: 11
The string reversed is: e r e h t   o l l e h
The character array is: hello
```

Comparing Strings

- Comparing **String** objects
  - Method **equals**
  - Method **equalsIgnoreCase**
  - Method **compareTo**
  - Method **regionMatches**

```
1     // Fig. 10.3: StringCompare.java
2     // This program demonstrates the methods equals,
3     // equalsIgnoreCase, compareTo, and regionMatches
4     // of the String class.
5
6     // Java extension packages
7     import javax.swing.JOptionPane;
8
9     public class StringCompare {
10
11       // test String class comparison methods
12       public static void main( String args[] )
13       {
14          String s1, s2, s3, s4, output;
15
16          s1 = new String( "hello" );
17          s2 = new String( "good bye" );
18          s3 = new String( "Happy Birthday" );
19          s4 = new String( "happy birthday" );
20
21          output = "s1 = " + s1 + "\ns2 = " + s2 +
22             "\ns3 = " + s3 + "\ns4 = " + s4 + "\n\n";
23
24          // test for equality
25          if ( s1.equals( "hello" ) )
26             output += "s1 equals \"hello\"\n";
27          else
28             output += "s1 does not equal \"hello\"\n";
29
30          // test for equality with ==
31          if ( s1 == "hello" )
32             output += "s1 equals \"hello\"\n";
33          else
34             output += "s1 does not equal \"hello\"\n";
35
```

StringCompare.java

Line 25

Line 31

Method equals tests two objects for equality using *lexicographical comparison*

Equality operator (==) tests if both references refer to same object in memory

CDAC
The Supercomputing People

```
36        // test for equality (ignore case)
37        if ( s3.equalsIgnoreCase( s4 ) )
38           output += "s3 equals s4\n";
39        else
40           output += "s3 does not equal s4\n";
41
42        // test compareTo
43        output +=
44           "\ns1.compareTo( s2 ) is " + s1.compareTo( s2 ) +
45           "\ns2.compareTo( s1 ) is " + s2.compareTo( s1 ) +
46           "\ns1.compareTo( s1 ) is " + s1.compareTo( s1 ) +
47           "\ns3.compareTo( s4 ) is " + s3.compareTo( s4 ) +
48           "\ns4.compareTo( s3 ) is " + s4.compareTo( s3 ) +
49           "\n\n";
50
51        // test regionMatches (case sensitive)
52        if ( s3.regionMatches( 0, s4, 0, 5 ) )
53           output += "First 5 characters of s3 and s4 match\n";
54        else
55           output +=
56              "First 5 characters of s3 and s4 do not match\n";
57
58        // test regionMatches (ignore case)
59        if ( s3.regionMatches( true, 0, s4, 0, 5 ) )
60           output += "First 5 characters of s3 and s4 match";
61        else
62           output +=
63              "First 5 characters of s3 and s4 do not match";
64
```

Test two objects for equality, but ignore case of letters in String

Method compareTo compares String objects

Method regionMatches compares portions of two String objects for equality

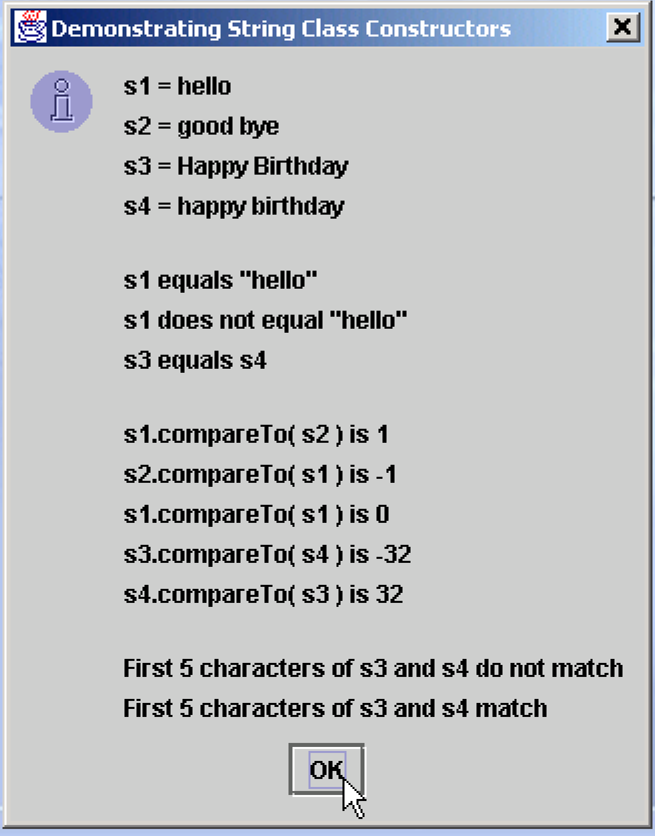
StringCompare.java

Line 37

Lines 44-48

```
         JOptionPane.showMessageDialog( null, output,
            "Demonstrating String Class Constructors",
            JOptionPane.INFORMATION_MESSAGE );

         System.exit( 0 );
      }

   }  // end class StringCompare
```

StringCompare.java



```
s1 = hello
s2 = good bye
s3 = Happy Birthday
s4 = happy birthday

s1 equals "hello"
s1 does not equal "hello"
s3 equals s4

s1.compareTo( s2 ) is 1
s2.compareTo( s1 ) is -1
s1.compareTo( s1 ) is 0
s3.compareTo( s4 ) is -32
s4.compareTo( s3 ) is 32

First 5 characters of s3 and s4 do not match
First 5 characters of s3 and s4 match
```

```java
1     // Fig. 10.4: StringStartEnd.java
2     // This program demonstrates the methods startsWith and
3     // endsWith of the String class.
4
5     // Java extension packages
6     import javax.swing.*;
7
8     public class StringStartEnd {
9
10       // test String comparison methods for beginning and end
11       // of a String
12       public static void main( String args[] )
13       {
14          String strings[] =
15             { "started", "starting", "ended", "ending" };
16          String output = "";
17
18          // test method startsWith
19          for ( int count = 0; count < strings.length; count++ )
20
21             if ( strings[ count ].startsWith( "st" ) )
22                output += "\"" + strings[ count ] +
23                   "\" starts with \"st\"\n";
24
25          output += "\n";
26
27          // test method startsWith starting from position
28          // 2 of the string
29          for ( int count = 0; count < strings.length; count++ )
30
31             if ( strings[ count ].startsWith( "art", 2 ) )
32                output += "\"" + strings[ count ] +
33                   "\" starts with \"art\" at position 2\n";
34
35          output += "\n";
```

StringStartEnd.java

Line 21

Line 31

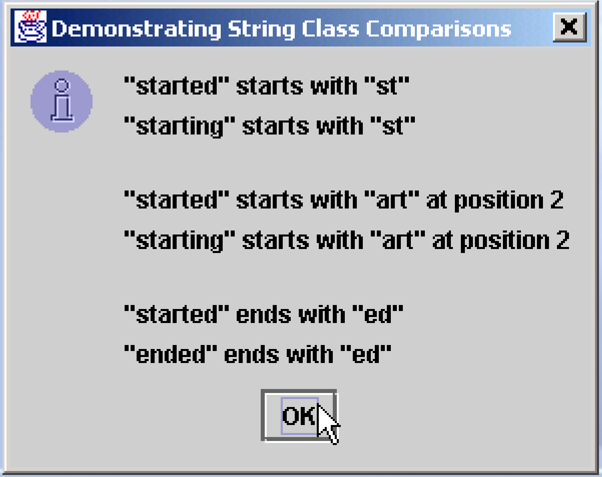
Method startsWith determines if String starts with specified characters

```
36
37       // test method endsWith
38       for ( int count = 0; count < strings.length; count++ )
39
40          if ( strings[ count ].endsWith( "ed" ) )
41             output += "\"" + strings[ count ] +
42                "\" ends with \"ed\"\n";
43
44       JOptionPane.showMessageDialog( null, output,
45          "Demonstrating String Class Comparisons",
46          JOptionPane.INFORMATION_MESSAGE );
47
48       System.exit( 0 );
49    }
50
51 }  // end class StringStartEnd
```

StringStartEnd.java

Line 40

Method endsWith determines if String ends with specified characters



```
Demonstrating String Class Comparisons

"started" starts with "st"
"starting" starts with "st"

"started" starts with "art" at position 2
"starting" starts with "art" at position 2

"started" ends with "ed"
"ended" ends with "ed"
```

Locating Characters and Substrings in Strings

- Search for characters in **String**
  - Method **indexOf**
  - Method **lastIndexOf**

```
1    // Fig. 10.6: StringIndexMethods.java
2    // This program demonstrates the String
3    // class index methods.
4
5    // Java extension packages
6    import javax.swing.*;
7
8    public class StringIndexMethods {
9
10      // String searching methods
11      public static void main( String args[] )
12      {
13         String letters = "abcdefghijklmabcdefghijklm";
14
15         // test indexOf to locate a character in a string
16         String output = "'c' is located at index " +
17            letters.indexOf( 'c' );
18
19         output += "\n'a' is located at index " +
20            letters.indexOf( 'a', 1 );
21
22         output += "\n'$' is located at index " +
23            letters.indexOf( '$' );
24
25         // test lastIndexOf to find a character in a string
26         output += "\n\nLast 'c' is located at index " +
27            letters.lastIndexOf( 'c' );
28
29         output += "\nLast 'a' is located at index " +
30            letters.lastIndexOf( 'a', 25 );
31
32         output += "\nLast '$' is located at index " +
33            letters.lastIndexOf( '$' );
34
```

StringIndexMethods.java

Lines 16-23

Lines 26-33

Method indexOf finds first occurrence of character in String

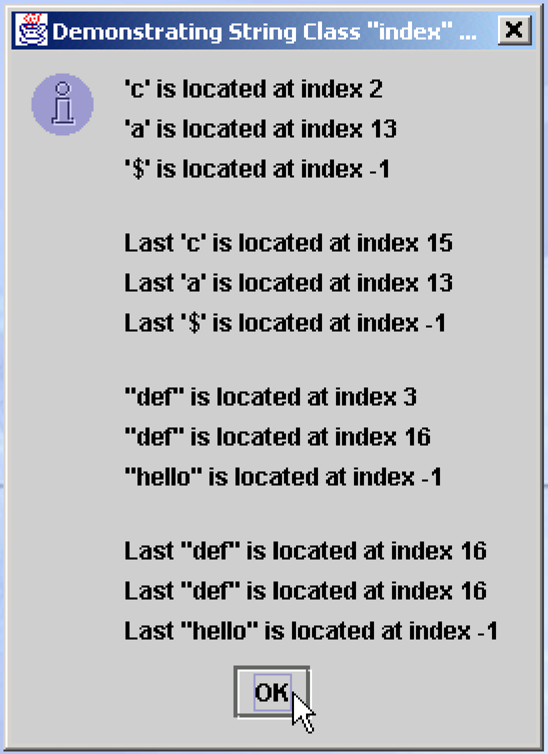
Method lastIndexOf finds last occurrence of character in String

StringIndexMethods.java

```
        // test indexOf to locate a substring in a string
        output += "\n\n\"def\" is located at index " +
           letters.indexOf( "def" );

        output += "\n\"def\" is located at index " +
           letters.indexOf( "def", 7 );

        output += "\n\"hello\" is located at index " +
           letters.indexOf( "hello" );

        // test lastIndexOf to find a substring in a string
        output += "\n\nLast \"def\" is located at index " +
           letters.lastIndexOf( "def" );

        output += "\nLast \"def\" is located at index " +
           letters.lastIndexOf( "def", 25 );

        output += "\nLast \"hello\" is located at index " +
           letters.lastIndexOf( "hello" );

        JOptionPane.showMessageDialog( null, output,
           "Demonstrating String Class \"index\" Methods",
           JOptionPane.INFORMATION_MESSAGE );

        System.exit( 0 );
     }

  }  // end class StringIndexMethods
```

Methods indexOf and lastIndexOf can also find occurrences of substrings

StringIndexMethods.java



```
'c' is located at index 2
'a' is located at index 13
'$' is located at index -1

Last 'c' is located at index 15
Last 'a' is located at index 13
Last '$' is located at index -1

"def" is located at index 3
"def" is located at index 16
"hello" is located at index -1

Last "def" is located at index 16
Last "def" is located at index 16
Last "hello" is located at index -1
```

Extracting Substrings from Strings

- Create **Strings** from other **Strings**
  - Extract substrings

```
1     // Fig. 10.7: SubString.java
2     // This program demonstrates the
3     // String class substring methods.
4
5     // Java extension packages
6     import javax.swing.*;
7
8     public class SubString {
9
10       // test String substring methods
11       public static void main( String args[] )
12       {
13          String letters = "abcdefghijklmabcdefghijklm";
14
15          // test substring methods
16          String output = "Substring from index 20 to end is " +
17             "\"" + letters.substring( 20 ) + "\"\n";
18
19          output += "Substring from index 0 up to 6 is " +
20             "\"" + letters.substring( 0, 6 ) + "\"";
21
22          JOptionPane.showMessageDialog( null, output,
23             "Demonstrating String Class Substring Methods",
24             JOptionPane.INFORMATION_MESSAGE );
25
26          System.exit( 0 );
27       }
28
29    }  // end class SubString
```

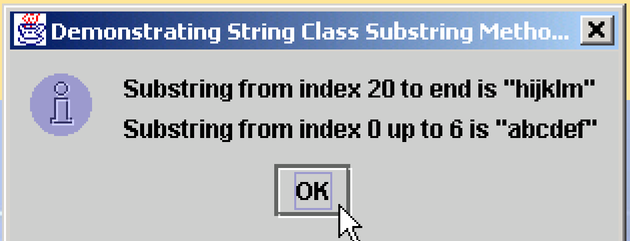
SubString.java

Line 17

Line 20

Beginning at index 20, extract characters from String letters

Extract characters from index 0 to 6 from String letters



```
Substring from index 20 to end is "hijklm"
Substring from index 0 up to 6 is "abcdef"
```

Concatenating Strings

- Method **concat**
  - Concatenate two **String** objects

```
1     // Fig. 10.8: StringConcatenation.java
2     // This program demonstrates the String class concat method.
3     // Note that the concat method returns a new String object. It
4     // does not modify the object that invoked the concat method.
5
6     // Java extension packages
7     import javax.swing.*;
8
9     public class StringConcatenation {
10
11       // test String method concat
12       public static void main( String args[] )
13       {
14          String s1 = new String( "Happy " ),
15                 s2 = new String( "Birthday" );
16
17          String output = "s1 = " + s1 + "\ns2 = " + s2;
18
19          output += "\n\nResult of s1.concat( s2 ) = " +
20             s1.concat( s2 );
21
22          output += "\ns1 after concatenation = " + s1;
23
24          JOptionPane.showMessageDialog( null, output,
25             "Demonstrating String Method concat",
26             JOptionPane.INFORMATION_MESSAGE );
27
28          System.exit( 0 );
29       }
30
31    }  // end class StringConcatenation
```

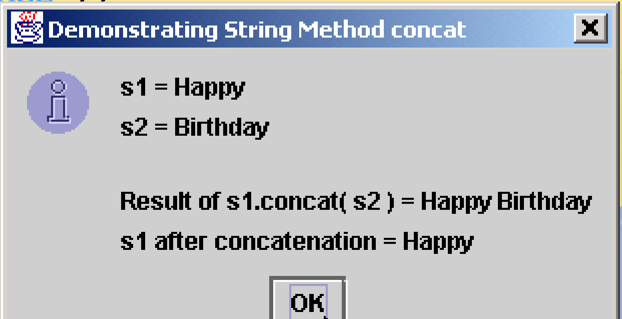
StringConcatenation.java

Line 20

Line 22

Concatenate String s2 to String s1

However, String s1 is not modified by method concat



```
s1 = Happy
s2 = Birthday

Result of s1.concat( s2 ) = Happy Birthday
s1 after concatenation = Happy
```

Miscellaneous String Methods

- Miscellaneous **String** methods
  - Return modified copies of **String**
  - Return character array

```
1     // Fig. 10.9: StringMiscellaneous2.java
2     // This program demonstrates the String methods replace,
3     // toLowerCase, toUpperCase, trim, toString and toCharArray
4
5     // Java extension packages
6     import javax.swing.*;
7
8     public class StringMiscellaneous2 {
9
10       // test miscellaneous String methods
11       public static void main( String args[] )
12       {
13          String s1 = new String( "hello" ),
14             s2 = new String( "GOOD BYE" ),
15             s3 = new String( "   spaces   " );
16
17          String output = "s1 = " + s1 + "\ns2 = " + s2 +
18             "\ns3 = " + s3;
19
20          // test method replace
21          output += "\n\nReplace 'l' with 'L' in s1: " +
22             s1.replace( 'l', 'L' );
23
24          // test toLowerCase and toUpperCase
25          output +=
26             "\n\ns1.toUpperCase() = " + s1.toUpperCase() +
27             "\ns2.toLowerCase() = " + s2.toLowerCase();
28
29          // test trim method
30          output += "\n\ns3 after trim = \"" + s3.trim() + "\"";
31
32          // test toString method
33          output += "\n\ns1 = " + s1.toString();
34
```

StringMiscellaneous2.java

Line 22

Line 26

Line 27

Line 30

Use method **replace** to return **s1** copy in which every occurrence of ‘**l**’ is replaced with ‘**L**’

Use method **toUpperCase** to return **s1** copy in which every character is uppercase

Use method **toLowerCase** to return **s2** copy in which every character is uppercase

Use method **trim** to return **s3** copy in which whitespace is eliminated

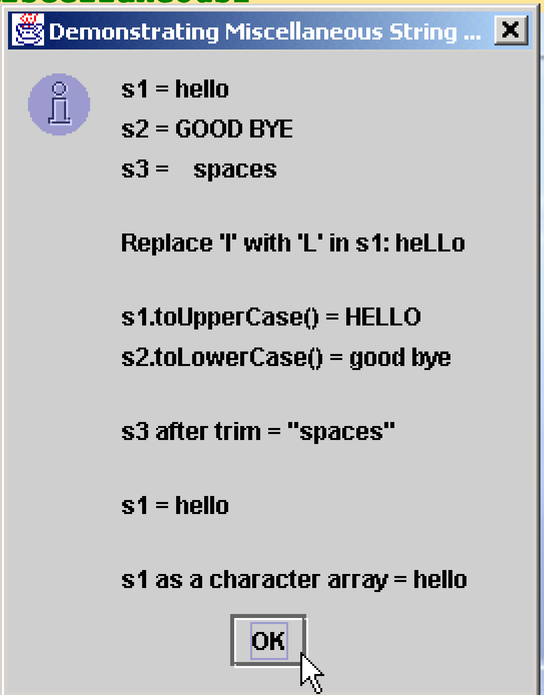
Use method **toString** to return **s1**

```java
35          // test toCharArray method
36          char charArray[] = s1.toCharArray();
37
38          output += "\n\ns1 as a character array = ";
39
40          for ( int count = 0; count < charArray.length; ++count )
41             output += charArray[ count ];
42
43          JOptionPane.showMessageDialog( null, output,
44             "Demonstrating Miscellaneous String Methods",
45             JOptionPane.INFORMATION_MESSAGE );
46
47          System.exit( 0 );
48       }
49
50    }  // end class StringMiscellaneous2
```

Use method `toCharArray` to return character array of `s1`

`StringMiscellaneous2.java`

Line 36



```
s1 = hello
s2 = GOOD BYE
s3 =   spaces

Replace 'l' with 'L' in s1: heLLo

s1.toUpperCase() = HELLO
s2.toLowerCase() = good bye

s3 after trim = "spaces"

s1 = hello

s1 as a character array = hello
```

Using String Method valueOf

- **String** provides **static** class methods
  - Method **valueOf**
    - Returns **String** representation of object, data type, etc.

The Supercomputing People

```
1     // Fig. 10.10: StringValueOf.java
2     // This program demonstrates the String class valueOf methods.
3
4     // Java extension packages
5     import javax.swing.*;
6
7     public class StringValueOf {
8
9        // test String valueOf methods
10       public static void main( String args[] )
11       {
12          char charArray[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
13          boolean b = true;
14          char c = 'Z';
15          int i = 7;
16          long l = 10000000;
17          float f = 2.5f;
18          double d = 33.333;
19
20          Object o = "hello";  // assign to an Object reference
21          String output;
22
23          output = "char array = " + String.valueOf( charArray ) +
24             "\npart of char array = " +
25             String.valueOf( charArray, 3, 3 ) +
26             "\nboolean = " + String.valueOf( b ) +
27             "\nchar = " + String.valueOf( c ) +
28             "\nint = " + String.valueOf( i ) +
29             "\nlong = " + String.valueOf( l ) +
30             "\nfloat = " + String.valueOf( f ) +
31             "\ndouble = " + String.valueOf( d ) +
32             "\nObject = " + String.valueOf( o );
33
```

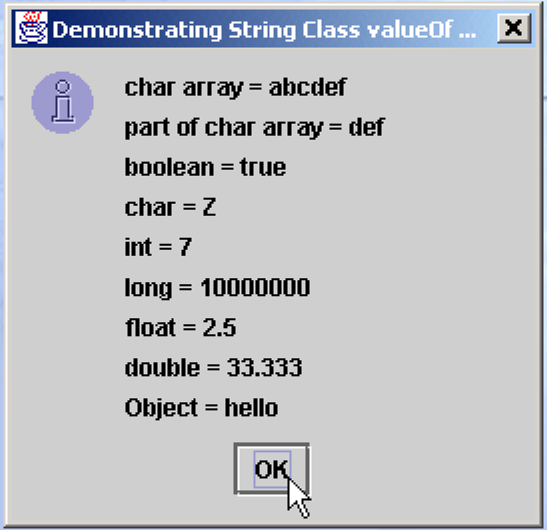
StringValueOf.java

Lines 26-32

static method valueOf of class String returns String representation of various types

StringValueOf.java

```
         JOptionPane.showMessageDialog( null, output,
            "Demonstrating String Class valueOf Methods",
            JOptionPane.INFORMATION_MESSAGE );

         System.exit( 0 );
      }

   }  // end class StringValueOf
```



```
char array = abcdef
part of char array = def
boolean = true
char = Z
int = 7
long = 10000000
float = 2.5
double = 33.333
Object = hello
```

String Method intern

- String comparisons
  - Slow operation
  - Method **intern** improves this performance
    - Returns reference to **String**
    - Guarantees reference has same contents as original **String**

```
1     // Fig. 10.11: StringIntern.java
2     // This program demonstrates the intern method
3     // of the String class.
4
5     // Java extension packages
6     import javax.swing.*;
7
8     public class StringIntern {
9
10       // test String method intern
11       public static void main( String args[] )
12       {
13          String s1, s2, s3, s4, output;
14
15          s1 = new String( "hello" );
16          s2 = new String( "hello" );
17
18          // test strings to determine if they are same
19          // String object in memory
20          if ( s1 == s2 )
21             output = "s1 and s2 are the same object in memory";
22          else
23             output = "s1 and s2 are not the same object in memory";
24
25          // test strings for equality of contents
26          if ( s1.equals( s2 ) )
27             output += "\ns1 and s2 are equal";
28          else
29             output += "\ns1 and s2 are not equal";
30
31          // use String intern method to get a unique copy of
32          // "hello" referred to by both s3 and s4
33          s3 = s1.intern();
34          s4 = s2.intern();
```

String s1 and String s2 occupy different memory locations

String s1 and String s2 have same content

Reference returned by s1.intern() is same as that returned by s2.intern()

StringIntern.java

Lines 15-20

Line 26

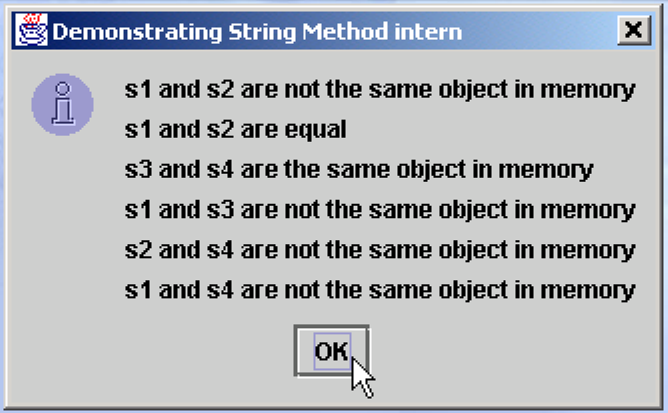
Lines 33-34

StringIntern.java

```
        // test strings to determine if they are same
        // String object in memory
        if ( s3 == s4 )
           output += "\ns3 and s4 are the same object in memory";
        else
           output +=
              "\ns3 and s4 are not the same object in memory";

        // determine if s1 and s3 refer to same object
        if ( s1 == s3 )
           output +=
              "\ns1 and s3 are the same object in memory";
        else
           output +=
              "\ns1 and s3 are not the same object in memory";

        // determine if s2 and s4 refer to same object
        if ( s2 == s4 )
           output += "\ns2 and s4 are the same object in memory";
        else
           output +=
              "\ns2 and s4 are not the same object in memory";

        // determine if s1 and s4 refer to same object
        if ( s1 == s4 )
           output += "\ns1 and s4 are the same object in memory";
        else
           output +=
              "\ns1 and s4 are not the same object in memory";

```

StringIntern.java

```java
        JOptionPane.showMessageDialog( null, output,
           "Demonstrating String Method intern",
           JOptionPane.INFORMATION_MESSAGE );

        System.exit( 0 );
     }

  }  // end class StringIntern
```



Demonstrating String Method intern

```
s1 and s2 are not the same object in memory
s1 and s2 are equal
s3 and s4 are the same object in memory
s1 and s3 are not the same object in memory
s2 and s4 are not the same object in memory
s1 and s4 are not the same object in memory
```

StringBuffer Class

- Class **StringBuffer**
  - When **String** object is created, its contents cannot change
  - Used for creating and manipulating dynamic string data
    - i.e., modifiable **String**s
  - Can store characters based on capacity
    - Capacity expands dynamically to handle additional characters
  - Uses operators **+** and **+=** for **String** concatenation

StringBuffer Constructors

- Three **StringBuffer** constructors
  - Default creates **StringBuffer** with no characters
    - Capacity of 16 characters

```java
// Fig. 10.12: StringBufferConstructors.java
// This program demonstrates the StringBuffer constructors.

// Java extension packages
import javax.swing.*;

public class StringBufferConstructors {

   // test StringBuffer constructors
   public static void main( String args[] )
   {
      StringBuffer buffer1, buffer2, buffer3;

      buffer1 = new StringBuffer();
      buffer2 = new StringBuffer( 10 );
      buffer3 = new StringBuffer( "hello" );

      String output =
         "buffer1 = \"" + buffer1.toString() + "\"" +
         "\nbuffer2 = \"" + buffer2.toString() + "\"" +
         "\nbuffer3 = \"" + buffer3.toString() + "\"";

      JOptionPane.showMessageDialog( null, output,
         "Demonstrating StringBuffer Class Constructors",
         JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );
   }

}  // end class StringBufferConstructors
```

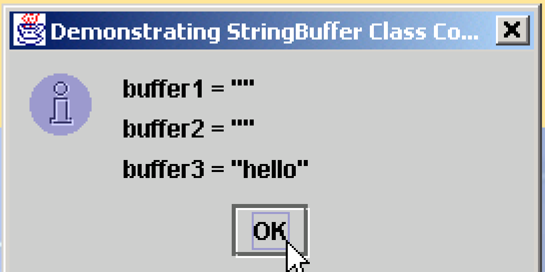
StringBufferConstructors

Default constructor creates empty StringBuffer with capacity of 16 characters

Second constructor creates empty StringBuffer with capacity of specified (10) characters

Third constructor creates StringBuffer with String "hello" and capacity of 16 characters

Method toString returns String representation of StringBuffer



```
Demonstrating StringBuffer Class Co...

buffer1 = ""
buffer2 = ""
buffer3 = "hello"
```

StringBuffer Methods

- Method **length**
  - Return **StringBuffer** length
- Method **capacity**
  - Return **StringBuffer** capacity
- Method **setLength**
  - Increase or decrease **StringBuffer** length
- Method **ensureCapacity**
  - Set **StringBuffer** capacity
  - Guarantee that **StringBuffer** has minimum capacity

```
1   // Fig. 10.13: StringBufferCapLen.java
2   // This program demonstrates the length and
3   // capacity methods of the StringBuffer class.
4
5   // Java extension packages
6   import javax.swing.*;
7
8   public class StringBufferCapLen {
9
10     // test StringBuffer methods for capacity and length
11     public static void main( String args[] )
12     {
13        StringBuffer buffer =
14           new StringBuffer( "Hello, how are you?" );
15
16        String output = "buffer = " + buffer.toString() +
17           "\nlength = " + buffer.length() +
18           "\ncapacity = " + buffer.capacity();
19
20        buffer.ensureCapacity( 75 );
21        output += "\n\nNew capacity = " + buffer.capacity();
22
23        buffer.setLength( 10 );
24        output += "\n\nNew length = " + buffer.length() +
25           "\nbuf = " + buffer.toString();
26
27        JOptionPane.showMessageDialog( null, output,
28           "StringBuffer length and capacity Methods",
29           JOptionPane.INFORMATION_MESSAGE );
30
31        System.exit( 0 );
32     }
33
34  }  // end class StringBufferCapLen
```

StringBufferCapLen.java

Line 17

Line 18

Line 20

Method length returns StringBuffer length

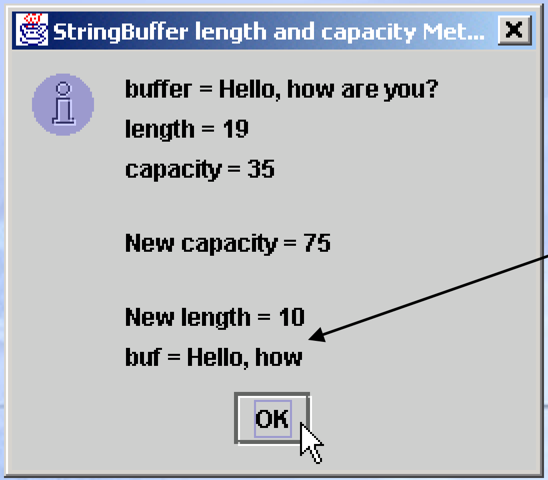
Method capacity returns StringBuffer capacity

Use method ensureCapacity to set capacity to 75

Use method setLength to set length to 10

StringBufferCapLen.java

Only **10** characters from **StringBuffer** are printed



```
buffer = Hello, how are you?
length = 19
capacity = 35

New capacity = 75

New length = 10
buf = Hello, how
```

Only 10 characters from StringBuffer are printed

StringBuffer Methods

- Manipulating **StringBuffer** characters
  - Method **charAt**
    - Return **StringBuffer** character at specified index
  - Method **setCharAt**
    - Set **StringBuffer** character at specified index
  - Method **getChars**
    - Return character array from **StringBuffer**
  - Method **reverse**
    - Reverse **StringBuffer** contents

```
1   // Fig. 10.14: StringBufferChars.java
2   // The charAt, setCharAt, getChars, and reverse methods
3   // of class StringBuffer.
4
5   // Java extension packages
6   import javax.swing.*;
7
8   public class StringBufferChars {
9
10     // test StringBuffer character methods
11     public static void main( String args[] )
12     {
13        StringBuffer buffer = new StringBuffer( "hello there" );
14
15        String output = "buffer = " + buffer.toString() +
16           "\nCharacter at 0: " + buffer.charAt( 0 ) +
17           "\nCharacter at 4: " + buffer.charAt( 4 );
18
19        char charArray[] = new char[ buffer.length() ];
20        buffer.getChars( 0, buffer.length(), charArray, 0 );
21        output += "\n\nThe characters are: ";
22
23        for ( int count = 0; count < charArray.length; ++count )
24           output += charArray[ count ];
25
26        buffer.setCharAt( 0, 'H' );
27        buffer.setCharAt( 6, 'T' );
28        output += "\n\nbuf = " + buffer.toString();
29
30        buffer.reverse();
31        output += "\n\nbuf = " + buffer.toString();
32
```

StringBufferChars.java

Lines 16-17

Line 20

Lines 26-27

Return `StringBuffer` characters at indices `0` and `4`, respectively

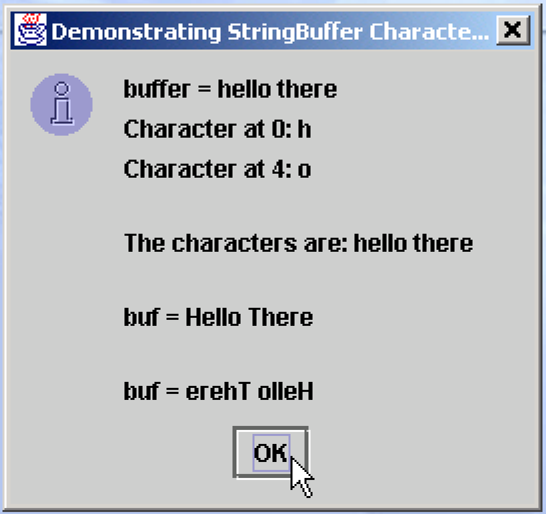
Return character array from `StringBuffer`

Replace characters at indices `0` and `6` with ‘`H`’ and ‘`T`,’ respectively

Reverse characters in `StringBuffer`

StringBufferChars.java

```
        JOptionPane.showMessageDialog( null, output,
           "Demonstrating StringBuffer Character Methods",
           JOptionPane.INFORMATION_MESSAGE );

        System.exit( 0 );
     }

  }  // end class StringBufferChars
```



```
buffer = hello there
Character at 0: h
Character at 4: o

The characters are: hello there

buf = Hello There

buf = erehT olleH
```

StringBuffer append Methods

- Method **append**
  - Allow data-type values to be added to **StringBuffer**

```java
1    // Fig. 10.15: StringBufferAppend.java
2    // This program demonstrates the append
3    // methods of the StringBuffer class.
4
5    // Java extension packages
6    import javax.swing.*;
7
8    public class StringBufferAppend {
9
10      // test StringBuffer append methods
11      public static void main( String args[] )
12      {
13         Object o = "hello";
14         String s = "good bye";
15         char charArray[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
16         boolean b = true;
17         char c = 'Z';
18         int i = 7;
19         long l = 10000000;
20         float f = 2.5f;
21         double d = 33.333;
22         StringBuffer buffer = new StringBuffer();
23
24         buffer.append( o );
25         buffer.append( "  " );
26
```

StringBufferAppend.java

Line 24

Append String “hello” to StringBuffer

```
27        buffer.append( s );
28        buffer.append( "  " );
29        buffer.append( charArray );
30        buffer.append( "  " );
31        buffer.append( charArray, 0, 3 );
32        buffer.append( "  " );
33        buffer.append( b );
34        buffer.append( "  " );
35        buffer.append( c );
36        buffer.append( "  " );
37        buffer.append( i );
38        buffer.append( "  " );
39        buffer.append( l );
40        buffer.append( "  " );
41        buffer.append( f );
42        buffer.append( "  " );
43        buffer.append( d );
44
45        JOptionPane.showMessageDialog( null,
46           "buffer = " + buffer.toString(),
47           "Demonstrating StringBuffer append Methods",
48           JOptionPane.INFORMATION_MESSAGE );
49
50        System.exit( 0 );
51     }
52
53  }  // end StringBufferAppend
```

Append String "good bye"

Append "a b c d e f"

Append "a b c"

Append boolean, char, int, long, float and double

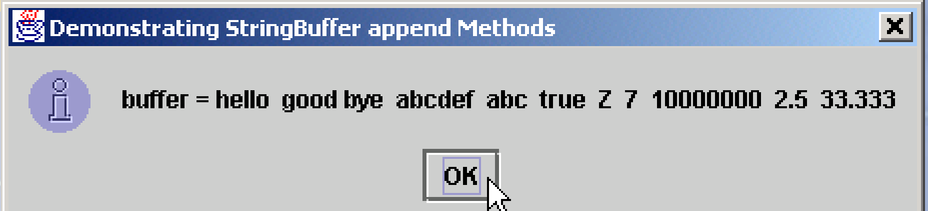
StringBufferAppend.java

Line 27

Line 29

Line 31

Lines 33-43



Demonstrating StringBuffer append Methods

```
buffer = hello  good bye  abcdef  abc  true  Z  7  10000000  2.5  33.333
```

StringBuffer Insertion and Deletion Methods

- Method **insert**
  - Allow data-type values to be inserted into **StringBuffer**
- Methods **delete** and **deleteCharAt**
  - Allow characters to be removed from **StringBuffer**

StringBufferInsert.java

```java
// Fig. 10.16: StringBufferInsert.java
// This program demonstrates the insert and delete
// methods of class StringBuffer.

// Java extension packages
import javax.swing.*;

public class StringBufferInsert {

   // test StringBuffer insert methods
   public static void main( String args[] )
   {
      Object o = "hello";
      String s = "good bye";
      char charArray[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
      boolean b = true;
      char c = 'K';
      int i = 7;
      long l = 10000000;
      float f = 2.5f;
      double d = 33.333;
      StringBuffer buffer = new StringBuffer();

24          buffer.insert( 0, o );
25          buffer.insert( 0, "  " );
26          buffer.insert( 0, s );
27          buffer.insert( 0, "  " );
28          buffer.insert( 0, charArray );
29          buffer.insert( 0, "  " );
30          buffer.insert( 0, b );
31          buffer.insert( 0, "  " );
32          buffer.insert( 0, c );
33          buffer.insert( 0, "  " );
34          buffer.insert( 0, i );
35          buffer.insert( 0, "  " );
36          buffer.insert( 0, l );
37          buffer.insert( 0, "  " );
38          buffer.insert( 0, f );
39          buffer.insert( 0, "  " );
40          buffer.insert( 0, d );
41
42          String output =
43             "buffer after inserts:\n" + buffer.toString();
44
45          buffer.deleteCharAt( 10 );  // delete 5 in 2.5
46          buffer.delete( 2, 6 );      // delete .333 in 33.333
47
48          output +=
49             "\n\nbuffer after deletes:\n" + buffer.toString();
50
51          JOptionPane.showMessageDialog( null, output,
52             "Demonstrating StringBufferer Inserts and Deletes",
53             JOptionPane.INFORMATION_MESSAGE );
54
55          System.exit( 0 );
56       }
57
```

Use method insert to insert data types in beginning of StringBuffer

Use method deleteCharAt to remove character from index 10 in StringBuffer

Remove characters from indices 2 through 5 (inclusive)

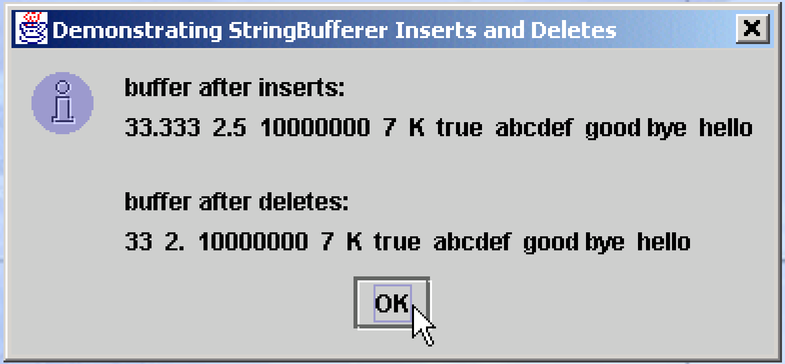
StringBufferInsert.java

Lines 24-40

Line 45

Line 46

StringBufferInsert.java



```
buffer after inserts:
33.333  2.5  10000000  7  K  true  abcdef  good bye  hello

buffer after deletes:
33  2.  10000000  7  K  true  abcdef  good bye  hello
```

java.util.StringTokenizer

Class StringTokenizer

- Tokenizer
  - Partition **String** into individual substrings
  - Use *delimiter*
  - Java offers **java.util.StringTokenizer**

Example using StringTokenizer

```
import java.util.StringTokenizer;
public class Test {
public static void main(String[] args) {
    String s=new String("Hello Welcome to IMCEITS!");
    StringTokenizer stk=new StringTokenizer(s,".,! ?");
    while(stk.hasMoreTokens())
        System.out.println(stk.nextToken());
    }
}
```

Outputs:

```
Hello
Welcome
to
IMCEITS
```

Thank you!