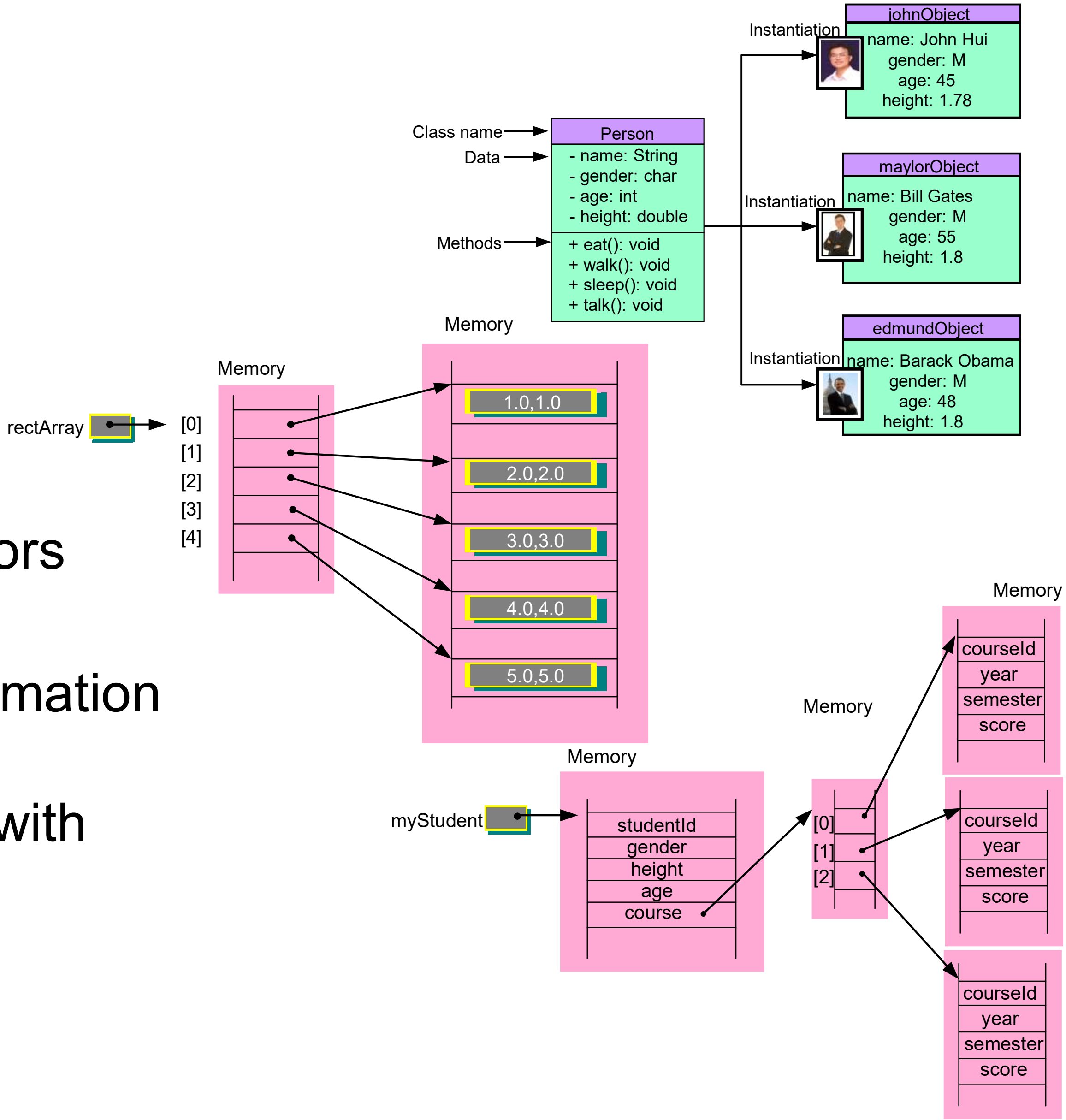
Chapter 11: Strings and Characters

- The String Class
- String Constructors
- String Input and Output
- String Class: Instance Methods
- The Character Class
- Conversion Methods
- The StringBuffer Class
- The StringTokenizer Class
- The Scanner Class
- Command Line Arguments
- Case Studies

Review Ch 10: Classes and Objects

- Objects and Classes
- Class Definition
- Message Sending
- Copying Objects
- The Keyword this
- Passing Objects to Methods
- Accessors and Mutators
- The Keyword static
- Encapsulation & Information Hiding
- Designing Programs with Classes
- Array of Objects
- Object Composition
- Case Study



Chapter 11: Strings and Characters

- The String Class
- String Constructors
- String Input and Output
- String Class: Instance Methods
- The Character Class
- Conversion Methods
- The StringBuffer Class
- The StringTokenizer Class
- The Scanner Class
- Command Line Arguments
- Case Studies

The String Class

- **Java.lang.String** is a class representing strings.
- A string or string constant is a series of characters in double quotes, e.g. **"Java Programming"**.
- In Java, strings are always created as **objects**.
- Syntax to declare a string:

```
        String Variable_Name ;
```

Since it is an object, memory will only be allocated when

```
        Variable_Name = new String( String_Value );
```

Combined into one:

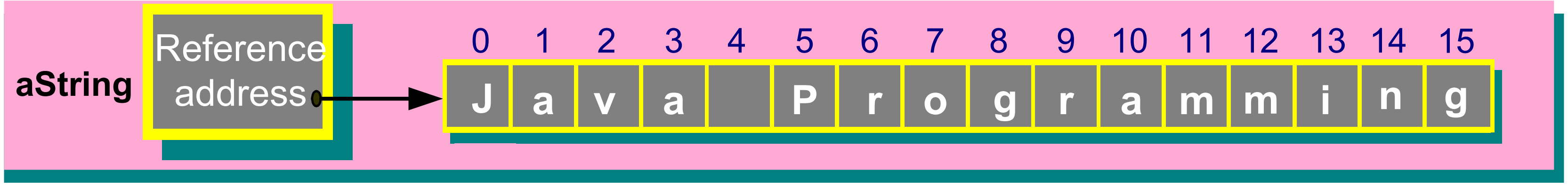
```
   String Variable_Name = new String( String_Value ) ;
```

or

```
   String Variable_Name = String_Value ;
```

Examples

String aString = new String(“Java Programming”); OR
String aString = “Java Programming” ;



aString

Reference address

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
J	a	v	a		P	r	o	g	r	a	m	m	i	n	g

The variable aString contains **reference address** of the string object

The **length** of the string is also stored in the string object’s storage.

String can be used as **arguments** for System.out.println()

System.out.println(“Java Programming”); OR
System.out.println(**aString**);

Chapter 11: Strings and Characters

- The String Class
- String Constructors
- String Input and Output
- String Class: Instance Methods
- The Character Class
- Conversion Methods
- The StringBuffer Class
- The StringTokenizer Class
- The Scanner Class
- Command Line Arguments
- Case Studies

String Constructors

```
public class StringConstructors {
 public static void main( String[] args )
 {
  char[] charData = { 'S' , 'C' , '1' , '0' , '2' };
  byte[] byteData = { (byte)'S' , (byte)'t' , (byte)'r' ,
                      (byte)'i' , (byte)'n' , (byte)'g' };
  StringBuffer strBuffer =
    new StringBuffer( "Introduction to Programming" );

  String str1 = new String();
  String str2 = new String( "Java Programming" );
  String str3 = new String( charData           );
  String str4 = new String( charData , 2 , 3   );
  String str5 = new String( byteData           );
  String str6 = new String( byteData , 2 , 4   );
  String str7 = new String( strBuffer          );
 }
}
```

position   length

```
Program Output
str1 =
str2 = Java Programming
str3 = SC102
str4 = 102
str5 = String
str6 = ring
str7 = Introduction to Programming
```

Chapter 11: Strings and Characters

- The String Class
- String Constructors
- String Input and Output
- String Class: Instance Methods
- The Character Class
- Conversion Methods
- The StringBuffer Class
- The StringTokenizer Class
- The Scanner Class
- Command Line Arguments
- Case Studies

String Input and Output

String Input Methods (from keyboard):

(1) **System.in.read()**

- Reads a character (in integer format) at a time from the keyboard.

(2) **Scanner class**

- Using methods next(), nextLine(), etc.

String Output Methods (to screen):

(1) **System.out.println()**

- Writes a string to the screen with carriage return.

(2) **System.out.print()**

- Writes a string to the screen.

Chapter 11: Strings and Characters

- The String Class
- String Constructors
- String Input and Output
- String Class: Instance Methods
- The Character Class
- Conversion Methods
- The StringBuffer Class
- The StringTokenizer Class
- The Scanner Class
- Command Line Arguments
- Case Studies

Class String: Instance Methods

Method	Description
`length()`	Returns the length of the string.
`charAt()`	Returns the character at the specified index.
`equals()`	Compares two strings for equality.
`equalsIgnoreCase()`	Compares two strings for equality, ignoring whether characters are in uppercase or lowercase.
`compareTo()`	Compares two strings.
`compareToIgnoreCase()`	Compares two strings, ignoring whether characters are in uppercase or lowercase.
`indexOf()`	Returns the position of the **first** occurrence of a specified input character or substring.
`lastIndexOf()`	Returns the position of the **last** occurrence of a specified input character or substring.
`substring()`	Generates and returns a new string that is a substring of an original input string. The substring **starts** from a specified position, and **ends** with a specified position or towards the end of the string.

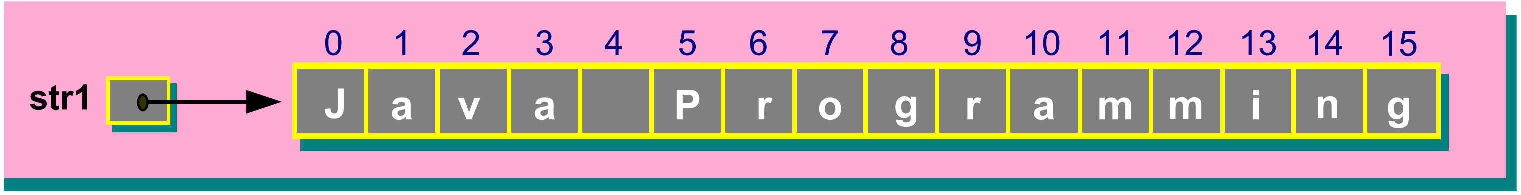
Method	Description
`concat()`	Concatenates another string to the end of a string.
`getChars()`	Copies characters from a string into a character array.
`toUpperCase()`	Generates a new string (given an input string) with all characters in uppercase.
`toLowerCase()`	Generates a new string (given an input string) with all characters in lowercase.
`trim()`	Removes whitespace (given an input string) from the **beginning** and **end** of the string.
`replace()`	Generates a new string with all occurrences of an old character replaced with a new character.
`hashCode()`	Generates an integer number from the string.
`startsWith()`	Tests whether a string starts with a specified set of characters.
`endsWith()`	Tests whether a string ends with a specified set of characters.

http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html

Method – length

- Determine the **number of characters** in a string
- For arrays, **length** is an instance variable, but for String, **length** is a method!!!

String str1 = “Java Programming”;

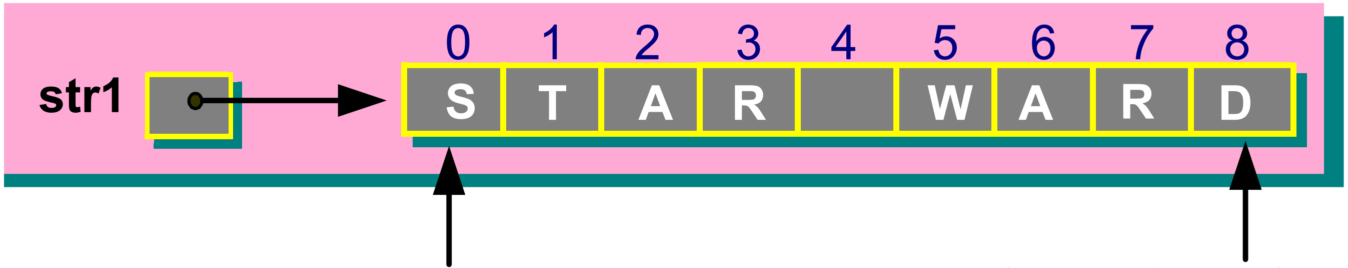


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
J	a	v	a		P	r	o	g	r	a	m	m	i	n	g

str1.length() = 16

Method – charAt

- Retrieve character at specific location in String



0	1	2	3	4	5	6	7	8
S	T	A	R		W	A	R	D

str1.charAt(0)

str1.charAt(str1.length() -1)

```
public class UsingCharAt {
  public static void main( String[] args ) {
    String str1   = "STAR WARD" ;
    int    vowels = 0 ;
    System.out.println( "The string is: " + str1  );
    System.out.print(   "The reverse string is: " );
    for ( int i = str1.length()-1 ; i>=0 ; i-- ) {
      System.out.print( str1.charAt(i) );
      switch ( str1.charAt(i) ) {
        case 'a': case 'A': case 'e': case 'E':
        case 'i': case 'I': case 'o': case 'O':
        case 'u': case 'U':
            vowels++; }
    }
    System.out.println( "The string has "
                        + vowels + " vowels" );
  }
}
```

Program Output

```
The string is: STAR WARD
The reverse string is: DRAW RATS
The string has 2 vowels
```

Methods – Comparisons

- equals()   , equalsIgnoreCase()
- compareTo() , compareToIgnoreCase()

- Compared according to **lexicographic order** based on Java's Unicode character set in the following order:

  space < digits (0-9) < letters (A-Z, a-z)

- Examples:

  " " < "$" < "1" < "9" < "A" < "Ape" < "Apeman" < "Z" < "Zebra" < "a" < "an" < "z" < "zero"

equals() string1.equals(string2)

Expression	Return value
`"ABC".equals("XYZ")`	`false`
`"ABC".equals("abc")`	`false`
`"ABC".equals("ABC")`	`true`
`"ABC".equals("ABCD")`	`false`

compareTo()string1.compareTo(string2)

Return value	Description
0	if the two strings are equal
> 0	if string1 is lexicographically greater than string2
< 0	if string1 is lexicographically less than string2

compareTo()string1.compareTo(string2)

Expression	Value
`"ABC".compareTo(   "XYZ"   )`	< 0
`"ABC".compareTo(   "abc"   )`	< 0
`"ABC".compareTo(   "ABC"   )`	0
`"123".compareTo(   "abc"   )`	< 0
`"abcd".compareTo( "abc"   )`	> 0
`"abc".compareTo(   "abcd" )`	< 0

Example: String Comparison

```
String str1 = “hello” ;
String str2 = “Hello” ;
String str3 = “HELLO” ;

-> str1.equals( str2 ) is false

-> str1.compareTo( str2 ) is  32
-> str2.compareTo( str1 ) is -32
-> str1.compareTo( str3 ) is  32
-> str2.compareTo( str3 ) is  32

-> str1.equalsIgnoreCase( str2 ) is true

-> str1.compareToIgnoreCase( str2 ) is 0
-> str1.compareToIgnoreCase( str3 ) is 0
-> str2.compareToIgnoreCase( str3 ) is 0
```

Methods – indexOf() and lastIndexOf()

- Search and return the index position of a given character or a given substring within an input string.

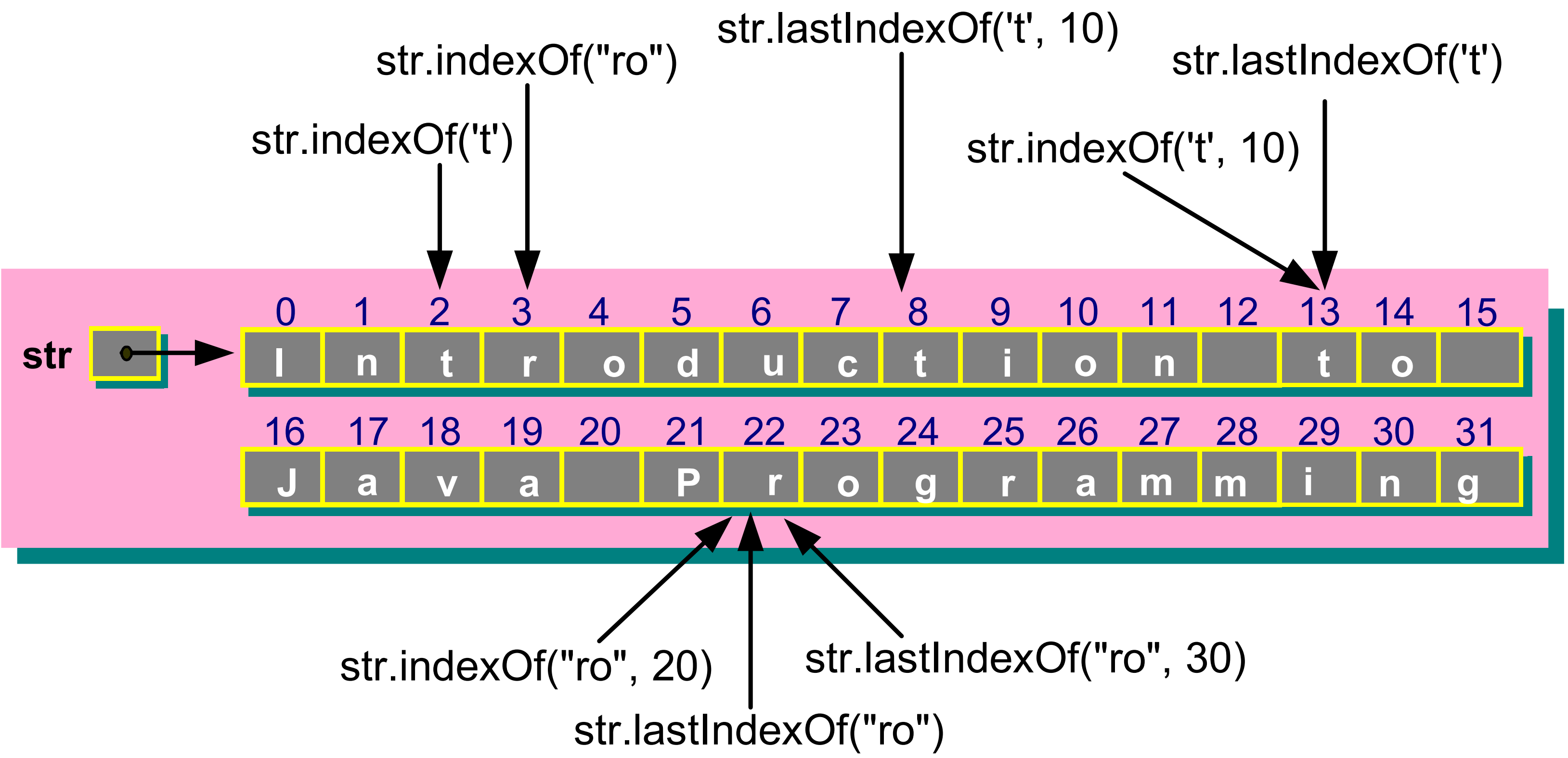
  - indexOf()   – locates the first occurrence
  - lastIndexOf() – locates the last occurrence

```
int indexOf( int    ch                      )
int indexOf( int    ch  , int startingIndex )
int indexOf( String str                     )
int indexOf( String str , int startingIndex )

int lastIndexOf( int    ch                      )
int lastIndexOf( int    ch  , int startingIndex )
int lastIndexOf( String str                     )
int lastIndexOf( String str , int startingIndex )
```

Example

```
String str = “Introduction to Java Programming” ;
str.indexOf(      ‘t’        )      location = 2
str.indexOf(      ‘t’   , 10 )      location = 13
str.indexOf(      “ro”       )      location = 3
str.indexOf(      “ro” , 20 )      location = 22
str.lastIndexOf( ‘t’         )      location = 13
str.lastIndexOf( ‘t’    , 10 )      location = 8
str.lastIndexOf( “ro”        )      location = 22
str.lastIndexOf( “ro” , 30 )        location = 22
```



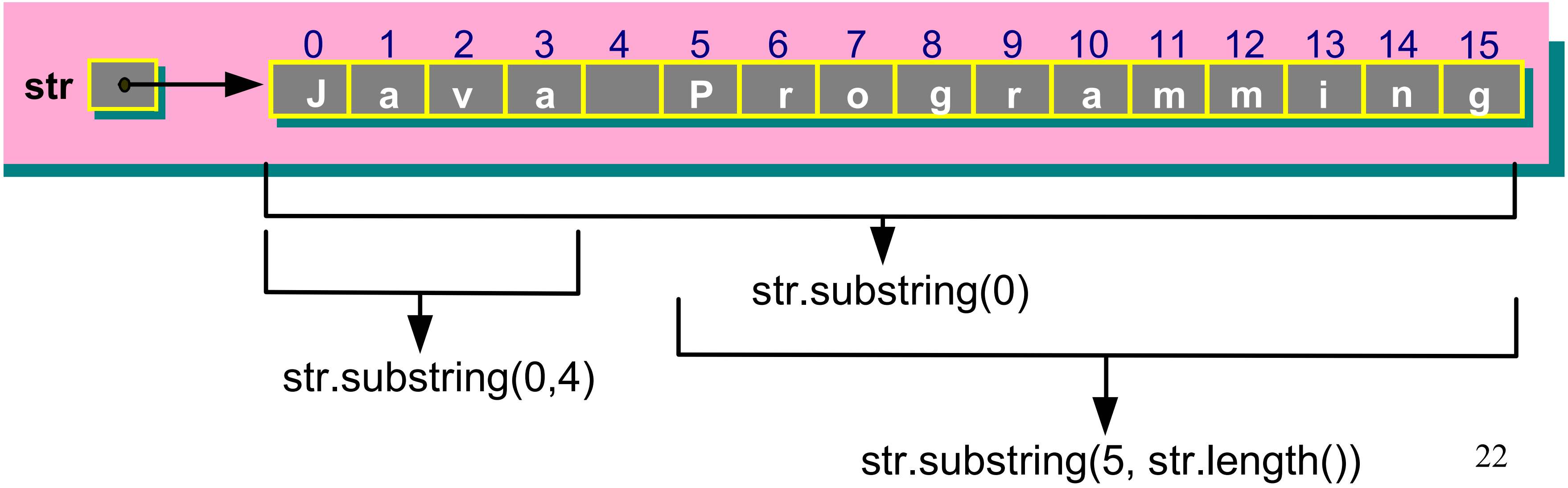
Methods - Extracting Substrings from Strings

- Extract a substring from a string.

```
int substring( int startIndex              )
int substring( int startIndex , int endIndex )
```

Example

```
String str = “Java Programming” ;
str.substring( 0     ) = “Java Programming”
str.substring( 0 , 4 ) = “Java”
str.substring( 5 , str.length() ) = “Programming”
```



Methods – Concatenating Strings

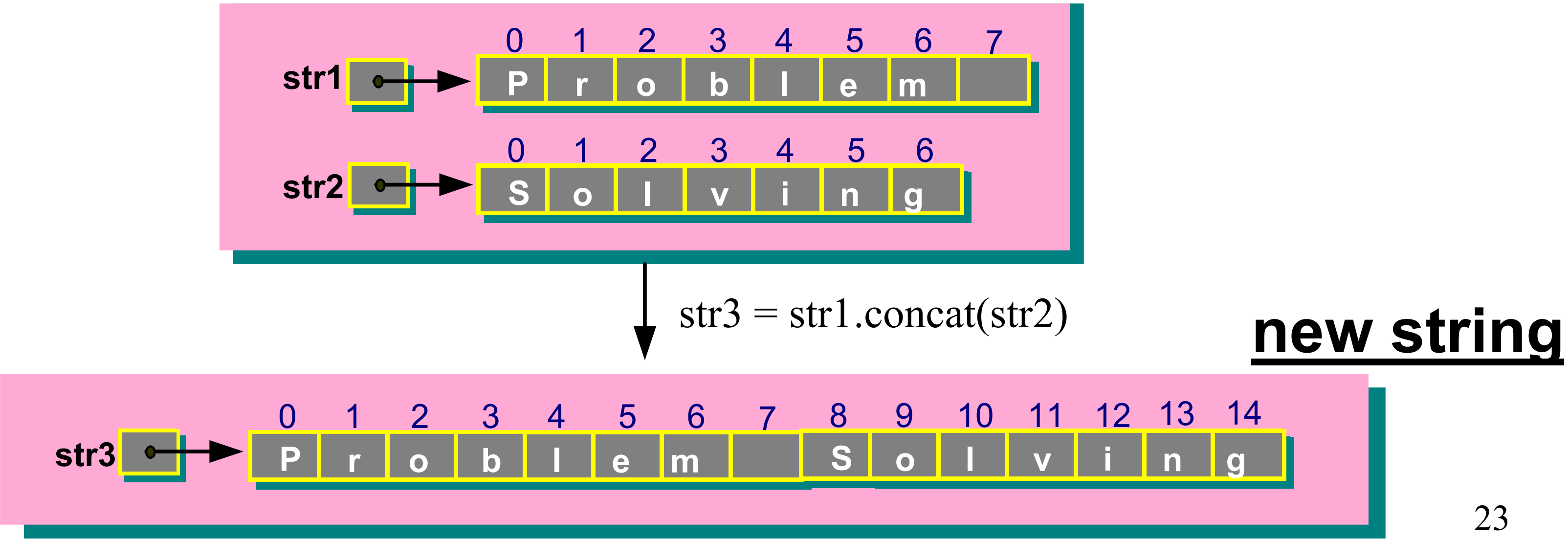
- Concatenates two strings to form a new string.

```
String concat(String aString)
```

Example

String in Java is immutable

```
String str1 = “Problem ” ;
String str2 = “Solving”  ;
String str3 = str1.concat( str2 );
OR String str3 = str1 + str2 ;
OR String str3 = “Problem ” + “Solving” ;
// then, str3 = “Problem Solving”
```



Methods – Replacement of Strings

String in Java is immutable

`String toLowerCase()`

- `String str2 = str1.toLowerCase();`

`String toUpperCase()`

- `String str2 = str1.toUpperCase();`
- returns a new string formed by converting the characters to upper (lower) case

`String trim()`

- `String str2 = str1.trim();`
- returns a new string str2 formed by removing whitespace from the beginning and end of string str1

`String replace( char oldChar , char newChar )`

- `String str2 = str1.replace( oldChar , newChar );`
- returns a new string formed by replacing all occurrences of an old character with a new character

Chapter 11: Strings and Characters

- The String Class
- String Constructors
- String Input and Output
- String Class: Instance Methods
- The Character Class
- Conversion Methods
- The StringBuffer Class
- The StringTokenizer Class
- The Scanner Class
- Command Line Arguments
- Case Studies

Character Class Methods

- **java.lang.Character** is a class that represents char as objects
- Class Methods for **testing** characters:

Method name	Returns a true if Argument is
boolean isDigit (char ch)	a digit, i.e. '0' - '9'.
boolean isLetter (char ch)	a letter, i.e. 'A' - 'Z', 'a' - 'z'.
boolean isSpaceChar (char ch)	a whitespace character, i.e. space, newline, formfeed, carriage return.
boolean isWhiteSpace (char ch)	a Java-defined whitespace character.
boolean isLowerCase (char ch)	a lowercase character, i.e. 'a' - 'z'
boolean isUpperCase (char ch)	an uppercase character, i.e. 'A' - 'Z'

- Methods for **converting** characters:

toLowerCase() and **toUpperCase()**

```
public class ConvertChar {
 public static void main( String[] args ) {
   int  i        ;
   char nextChar ;
   String str = "Introduction to Java Programming";
   System.out.println( "The string is: " + str );
   System.out.print(   "The new string is: "   );
   for ( i = 0 ; i < str.length() ; i++ )
   {
     nextChar = str.charAt( i ) ;
     if ( Character.isUpperCase( nextChar ) )
       nextChar = Character.toLowerCase( nextChar );
     else
     if ( Character.isLowerCase( nextChar ) )
       nextChar = Character.toUpperCase( nextChar );
     System.out.print( nextChar );
   }
   System.out.println();
 }
}
```

Program Output

```
The string is: Introduction to Java Programming
The new string is: iNTRODUCTION TO jAVA pROGRAMMING
```

http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Character.html

Chapter 11: Strings and Characters

- The String Class
- String Constructors
- String Input and Output
- String Class: Instance Methods
- The Character Class
- Conversion Methods
- The StringBuffer Class
- The StringTokenizer Class
- The Scanner Class
- Command Line Arguments
- Case Studies

Conversion Methods

#1: Numbers to Strings Conversion

- Three ways to convert a number into a string:

(1) String str = “” + num ;

(2) **Integer, Long, Float** and **Double** are **Wrapper** classes that represent numbers as objects. They provide static methods:

String str = **Integer**.**toString**(i); // where int i ;
String str = **Double**.**toString**(d); // where double d ;

(3) Using the String class method:

```
String valueOf(Type Value)
```

Examples:

String str1 = new String(String.**valueOf**(123));
String str2 = new String(String.valueOf(1233.56));
String str3 = new String(String.valueOf('A‘));
String str4 = new String(String.valueOf(true));

Conversion Methods

#2: Strings to Numbers Conversion

- Integer, Long, Float and Double are **wrapper classes** that represent numbers as objects.
- They provide useful static methods for conversions

```
int     parseInt(    String str )
long    parseLong(   String str )
float   parseFloat(  String str )
double  parseDouble( String str )
```

Examples:

```
String str1        = new String( "123"     );
int    intValue    = Integer.parseInt(str1);
String str2        = new String( "123456"  );
long   longValue   = Long.parseLong(str2);
String str3        = new String( "12.34"   );
float  floatValue  = Float.parseFloat(str3);
String str4        = new String( "1234.56" );
double doubleValue = Double.parseDouble(str4);
```

```
public class ConversionMethod {
  public static void main(String[] args) {
    String str1 = new String( String.valueOf( 123     ) );
    String str2 = new String( String.valueOf( 1233.56 ) );
    String str3 = new String( String.valueOf( 'A'     ) );
    String str4 = new String( String.valueOf( true    ) );
    System.out.println(       str1       );
    System.out.println( "[" + str2 + "]" );
    System.out.println( "[" + str3 + "]" );
    System.out.println( "[" + str4 + "]" );
  }
}
```

Program Output

```
123
[1233.56]
[A]
[true]
```

Chapter 11: Strings and Characters

- The String Class
- String Constructors
- String Input and Output
- String Class: Instance Methods
- The Character Class
- Conversion Methods
- The StringBuffer Class
- The StringTokenizer Class
- The Scanner Class
- Command Line Arguments
- Case Studies

The StringBuffer Class

- **String Class** – provides many methods for processing strings, the string **cannot** be changed once it is created.

- **StringBuffer Class** – provides methods for creating **dynamic** string information, and strings can be modified and extended.

These are the key difference between them.

Instance Methods in StringBuffer Class

Method	Description
`append()`	Appends values of various data type (e.g. **`boolean`**, **`char`**, **`int`**, **`float`**, **`double`**, etc.) to the end of a **`StringBuffer`**.
`insert()`	Inserts values of various data type (e.g. **`boolean`**, **`char`**, **`int`**, **`float`**, **`double`**, etc.) to any position in a **`StringBuffer`**.
`delete()`	Removes character(s) at any position in a **`StringBuffer`**.
`capacity()`	Returns the number of characters that can be stored in a **`StringBuffer`**.
`ensureCapacity()`	Ensures a **`StringBuffer`** has a minimum capacity.
`length()`	Returns the number of characters in a **`StringBuffer`**.
`setLength()`	Increases or decreases the maximum length of a **`StringBuffer`**.
`charAt()`	Returns the character at the specified index (position).

Instance Methods in StringBuffer Class

Method	Description
`setCharAt()`	Sets the character at the specified position in the **`StringBuffer`** to the character argument.
`getChars()`	Copies character(s) from a source string in the **`StringBuffer`** into a character array.
`reverse()`	Reverses the contents of a **`StringBuffer`**.
`replace()`	Replaces the characters starting at the specified start position and ending at one position less than the specified end position in the **`StringBuffer`**, with the characters in the string argument. The number of characters replaced need not be the same as that in the string argument.
`substring()`	Generates a new string that is a substring of the original string in the **`StringBuffer`**. The substring starts from a specified position, and ends with a specified position or towards the end of the string in the **`StringBuffer`**.
`toString()`	Converts a **`StringBuffer`** into a string.

StringBuffer Constructors

Constructor	Description
`StringBuffer()`	Creates a **`StringBuffer`** **with no character** and an initial capacity of **16 characters**.
`StringBuffer(int length)`	Creates a **`StringBuffer`** **with no character** and an initial capacity of **`length`**.
`StringBuffer( String aString )`	Creates a **`StringBuffer`** with the same characters as **`aString`** and an **additional** capacity of **16 characters**.

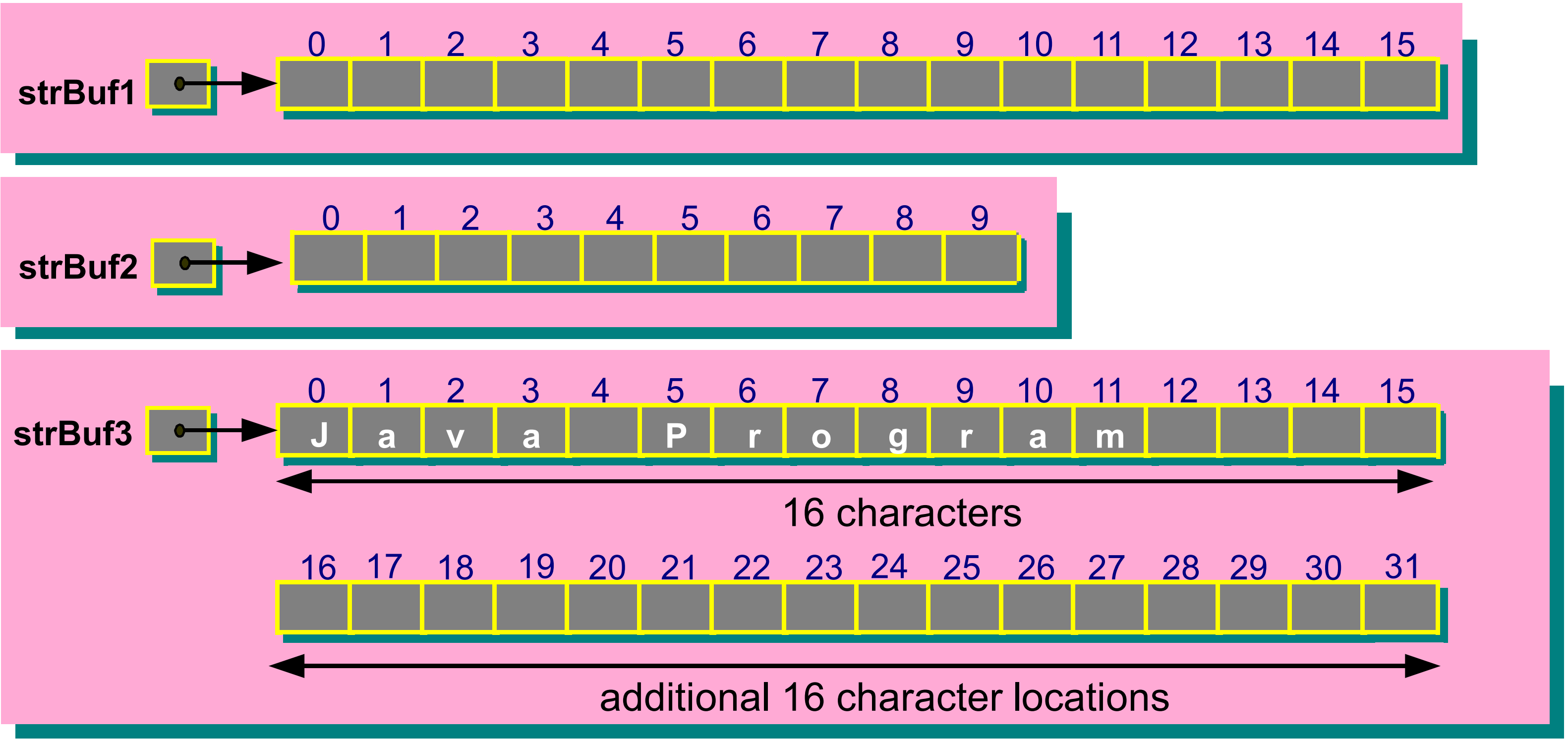
StringBuffer Constructors

```
public class StringBufferConstructors
{
  public static void main(String[] args)
  {
    StringBuffer strBuf1 = new StringBuffer();
    StringBuffer strBuf2 = new StringBuffer( 10 );
    StringBuffer strBuf3 =
      new StringBuffer( "Java Program    " );

    System.out.println( "strBuf1 = ["
                         + strBuf1.toString() + "]" );
    System.out.println( "strBuf2 = ["
                         + strBuf2.toString() + "]" );
    System.out.println( "strBuf3 = ["
                         + strBuf3.toString() + "]" );
  }
}
```

StringBuffer Constructors

```
Program Output
strBuf1 = []
strBuf2 = []
strBuf3 = [Java Program    ]
```



StringBuffer Methods - Append

- Adds a data item to the end of a StringBuffer.

```
StringBuffer append( Type Value )
```

where parameter Value is of type Type and and Type can be of any primitive data types

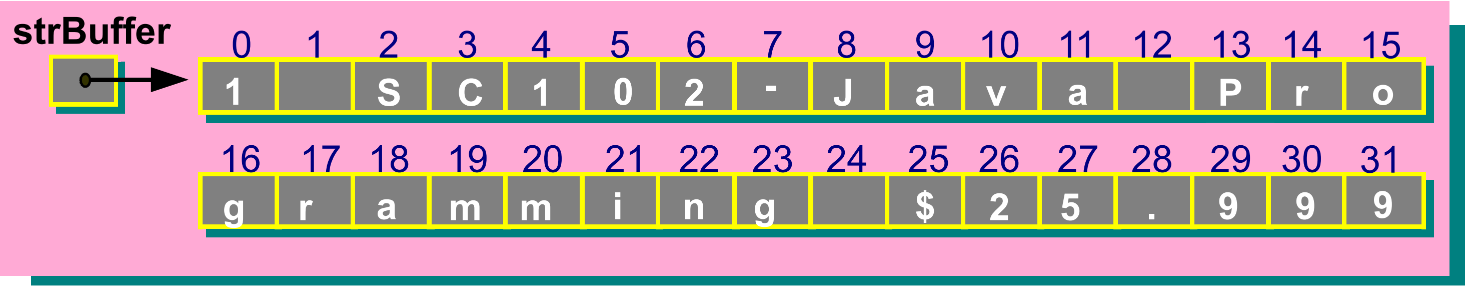
```
public class UsingAppendMethod {
  public static void main( String[] args ) {
    int    i        = 1                            ;
    char[] charData = { 'S', 'C', '1', '0', '2' } ;
    Object o        = "Java"                      ;
    String s        = "Programming"               ;
    char   c        = '$'                         ;
    double d        = 25.999                       ;

    StringBuffer strBuffer = new StringBuffer(32);
    strBuffer.append(i);        strBuffer.append(" ");
    strBuffer.append(charData); strBuffer.append("-");
    strBuffer.append(o);        strBuffer.append(" ");
    strBuffer.append(s);        strBuffer.append(" ");
    strBuffer.append(c);
    strBuffer.append(d);
    System.out.println( "[" + strBuffer.toString() + "]" );
  }
}
```

Program Output

```
[1 SC102-Java Programming $25.999]
```

The final string after the append operations in StringBuffer



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1		S	C	1	0	2	-	J	a	v	a		P	r	o

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
g	r	a	m	m	i	n	g		$	2	5	.	9	9	9

StringBuffer Methods - Insert

- Inserts a data item to a StringBuffer at the specified index position.
- 9 versions (overloaded) of the insert() method to support various data type values

```
StringBuffer insert(int offset, Type Value)
```

where

- offset specifies the specified **index** position and
- Value is Type that can be of any primitive data types.

StringBuffer Methods - Delete

```
StringBuffer delete(int start, int end)
StringBuffer deleteCharAt(int index)
```

- start and end specifies the starting position and the ending at one position less than end to delete
- index specifies the index position of the character to delete

```
public class InsertDeleteMethods {
   public static void main( String[] args ){
   int    i        = 1                       ;
   char[] charData = { 'S','C','1','0','2' } ;
   Object o        = "Java"                  ;
   String s        = "Programming"           ;
   char   c        = '$'                     ;
   double d        = 25.999                  ;
   StringBuffer strBuffer = new StringBuffer(32);
   strBuffer.insert(0,d);
   strBuffer.insert(0,c);   strBuffer.insert(0," ");
   strBuffer.insert(0,s);   strBuffer.insert(0," ");
   strBuffer.insert(0,o);
   strBuffer.insert(0,"-"); strBuffer.insert(0,charData);
   strBuffer.insert(0," "); strBuffer.insert(0,i);
   System.out.println( strBuffer.toString() );
   strBuffer.deleteCharAt(3);
   strBuffer.delete(12,24);
   System.out.println( strBuffer );
  }
}
```

```
Program Output
1 SC102-Java Programming $25.999
1 S102-Java $25.999
```

StringBuffer Methods - Capacity Methods

Length()

- Returns the number of characters currently in a StringBuffer.

```
int length()
```

Capacity()

- Returns the number of characters that can be stored in a StringBuffer.

```
int capacity()
```

StringBuffer Methods - Capacity Methods

setLength()

- Increases and decreases the maximum length of the StringBuffer.

```
int setLength( int newLength )
```

ensureCapacity()

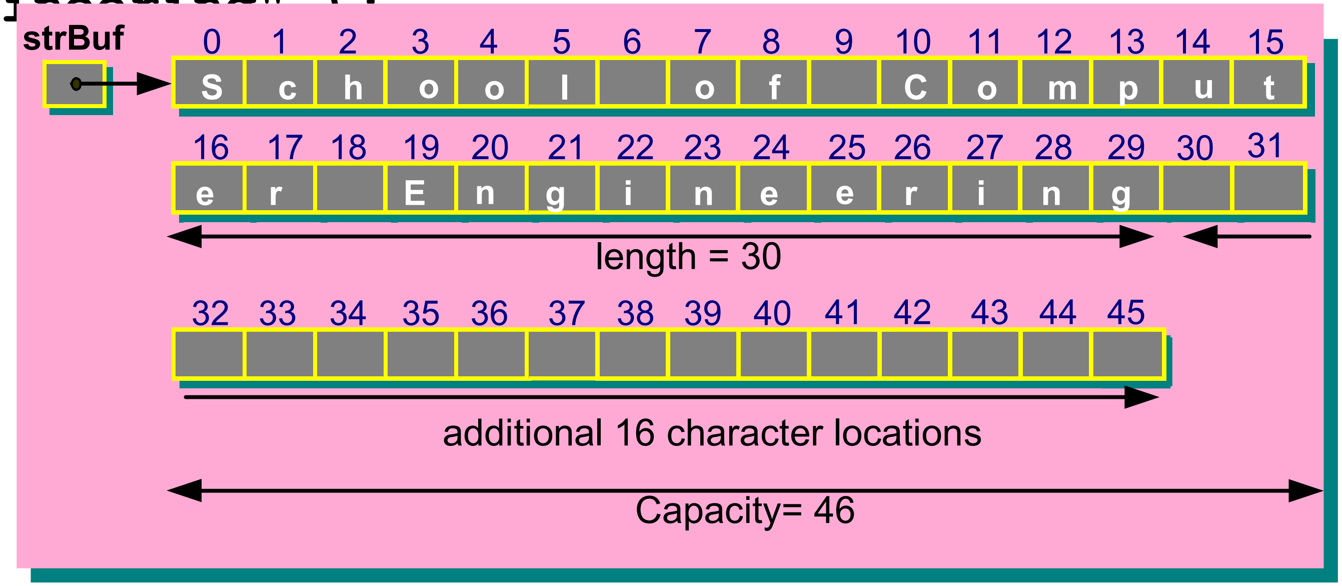
- Ensures that the StringBuffer a minimum capacity.

```
int ensureCapacity( int minCap )
```

```java
public class UsingCapacity {
  public static void main( String[] args ) {
    StringBuffer strBuf = new
        StringBuffer( "School of Computer Engineering" );
    System.out.println( "StrBuf = " + strBuf.toString()
        + "\nLength = "   + strBuf.length()
        + "\nCapacity = " + strBuf.capacity() );
    strBuf.setLength(18);
    System.out.println( "StrBuf = " + strBuf.toString()
        + "\nLength = "   + strBuf.length()
        + "\nCapacity = " + strBuf.capacity() );
  }
}
```

```
Program Output
StrBuf = School of Computer Engineering
Length = 30
Capacity = 46
StrBuf = School of Computer
Length = 18
Capacity = 46
```

```
 StringBuffer strBuf = new
   StringBuffer( "School of Computer
Engineering" );
```



strBuf

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	c	h	o	o	l		o	f		C	o	m	p	u	t

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
e	r		E	n	g	i	n	e	e	r	i	n	g		

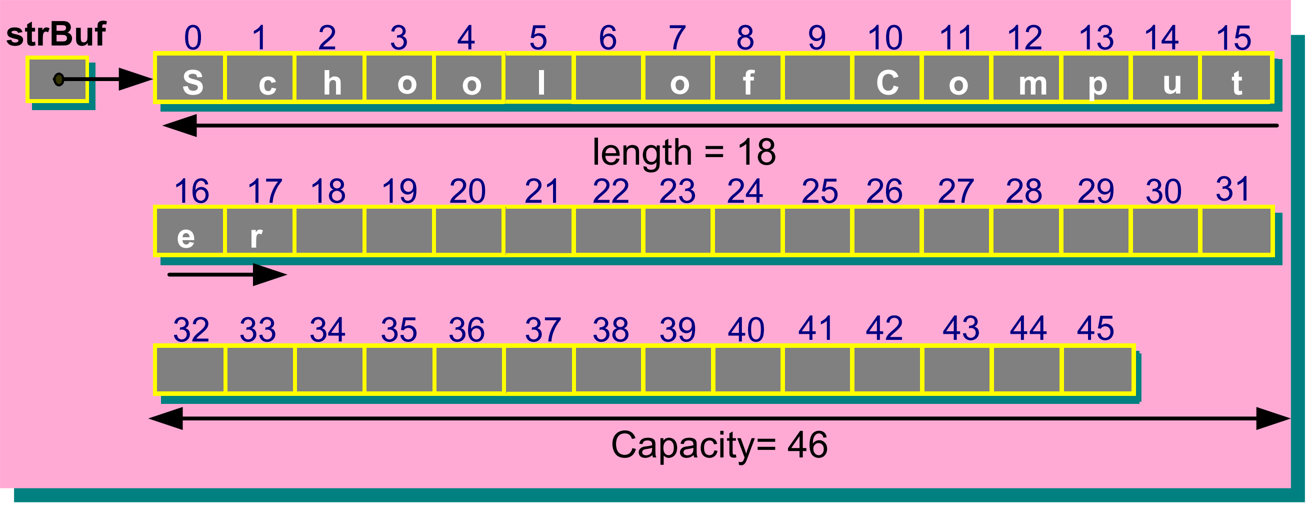
length = 30

32	33	34	35	36	37	38	39	40	41	42	43	44	45

additional 16 character locations

Capacity= 46

```
strBuffer.setLength(18);
```



strBuf

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	c	h	o	o	l		o	f		C	o	m	p	u	t

length = 18

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
e	r														

32	33	34	35	36	37	38	39	40	41	42	43	44	45

Capacity= 46

StringBuffer Methods - Character Manipulation

charAt()

```
    char charAt()
```

setCharAt()

```
    void setCharAt( int position , char ch )
```

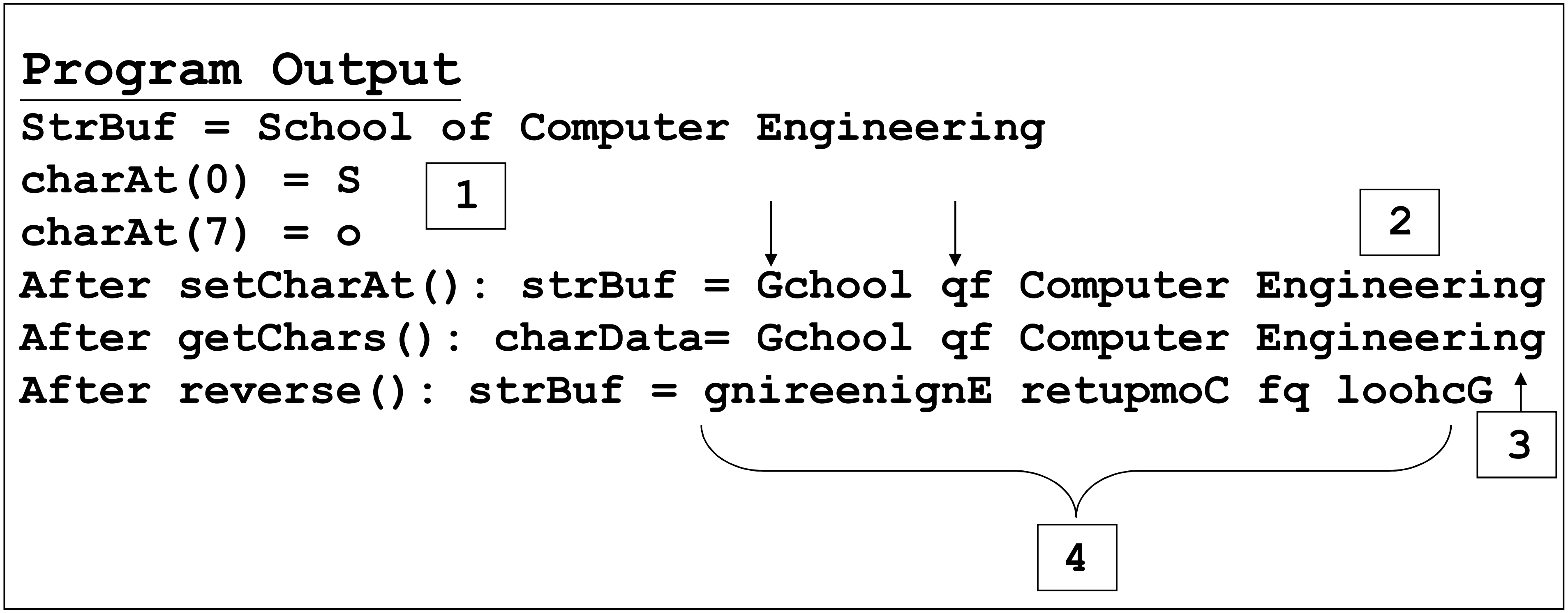
getChars() – output to destArray

```
    void getChars( int     startIndex      ,
                   int     endIndex        ,
                   char[]  destArray       ,
                   int     destStartIndex )
```

reverse()

```
    StringBuffer reverse()
```

```
public class UsingCharMan {
  public static void main( String[] args ) {
    int i;
    StringBuffer strBuf =
      new StringBuffer( "School of Computer Engineering" );
    System.out.println( "StrBuf = "
                        + strBuf.toString() );
1   System.out.println( "charAt(0) = "   + strBuf.charAt(0)
                      + "\ncharAt(7) = " + strBuf.charAt(7) );
    strBuf.setCharAt( 0 , 'G' );
2   strBuf.setCharAt( 7 , 'q' );
    System.out.println( "After setCharAt(): strBuf = "
                        + strBuf.toString() );
    char[] charData = new char[ strBuf.length() ];
3   strBuf.getChars( 0 , strBuf.length(),  charData , 0 );
    System.out.print( "After getChars(): charData= " );
    for ( i = 0 ; i < strBuf.length() ; i++ )
      System.out.print( charData[i] );
4   strBuf.reverse() ;
    System.out.println() ;
    System.out.println( "After reverse(): strBuf = "
                        + strBuf.toString() );
  }
}
```



Program Output

```
StrBuf = School of Computer Engineering
charAt(0) = S
charAt(7) = o
After setCharAt(): strBuf = Gchool qf Computer Engineering
After getChars(): charData= Gchool qf Computer Engineering
After reverse(): strBuf = gnireenignE retupmoC fq loohcG
```

http://java.sun.com/j2se/1.5.0/docs/api/java/lang/StringBuffer.html

Chapter 11: Strings and Characters

- The String Class
- String Constructors
- String Input and Output
- String Class: Instance Methods
- The Character Class
- Conversion Methods
- The StringBuffer Class
- The StringTokenizer Class
- The Scanner Class
- Command Line Arguments
- Case Studies

The StringTokenizer Class

- From package **java.util**
  - Breaks up a string into components or tokens, e.g., A sentence may be broken into **words**
  - Tokens are separated from one another by delimiters. The default delimiters are white-space characters (i.e. tabs, blanks, line feeds, etc.)

- **Two constructors:**

1 `StringTokenizer( String str )`

– Constructs a StringTokenizer object for string str with **default** delimiters (i.e. white-spaces).

2 `StringTokenizer( String str , String delim )`

– You can **specify** delimiters delim

http://java.sun.com/j2se/1.5.0/docs/api/java/util/StringTokenizer.html

StringTokenizer Methods

hasMoreTokens()

`boolean hasMoreTokens()`

- Returns true if there is token remaining in the string.

nextTokens()

`String nextToken()`

- Returns the next token in the string.

`String nextToken( String delim )`

- Reset the delimiter and return the next token in the string.

countTokens()

`int countTokens()`

- Returns the number of tokens remaining in the StringTokenizer.

```
import java.util.StringTokenizer ;
public class StringTokenizerApp {
 public static void main( String[] args )
 {
  StringTokenizer str1 = new StringTokenizer(
    "This is SC102 course on Java programming." );
  System.out.println( "countTokens() = "
                      + str1.countTokens() );
  while ( str1.hasMoreTokens() )
    System.out.println( str1.nextToken() );

  StringTokenizer str2 = new StringTokenizer(
    "http://www.ntu.edu.sg/sce/asschui.html", ":/" );
  System.out.println( "countTokens() = "
                      + str2.countTokens() );
  while ( str2.hasMoreTokens() )
    System.out.println( str2.nextToken() );
 }
}
```

```
Program Output
countTokens() = 7
This
is
SC102
course
on
Java
programming.

countTokens() = 4
http
www.ntu.edu.sg
sce
asschui.html
```

```java
import java.util.Scanner;
import java.util.StringTokenizer;
public class StringTokenizer2 {
  public static void main( String[] args ) {
    String  inputString , aString ;
    int     total = 0 ;
    Scanner sc    = new Scanner( System.in );
    System.out.print( "Enter your number strings: " );
    inputString = sc.nextLine();
    StringTokenizer str = new
        StringTokenizer( inputString );
    System.out.println( "countTokens() = "
                        + str.countTokens() );
    while ( str.hasMoreTokens() ) {
      aString = str.nextToken();
      total += Integer.parseInt( aString );
    }
    System.out.println( "The total is " + total );
  }
}
```

Using with Scanner class

```
Program Output
Enter your number strings: 13 45 67
countTokens() = 3
The total is 125
```

Chapter 11: Strings and Characters

- The String Class
- String Constructors
- String Input and Output
- String Class: Instance Methods
- The Character Class
- Conversion Methods
- The StringBuffer Class
- The StringTokenizer Class
- The Scanner Class
- Command Line Arguments
- Case Studies

The Scanner Class

In the Scanner class, a word can be used as a delimiter.

```
import java.util.Scanner;
public class ScannerApp
{
 public static void main(String[] args)
 {
  Scanner str1 = new
   Scanner( "This is SC102 course on Java programming." );
  str1.useDelimiter( "course" );
  while ( str1.hasNext() )
     System.out.println( "[" + str1.next() + "]" );
 }
}
```

Program Output

```
[This is SC102]
[ on Java programming.]
```

http://java.sun.com/j2se/1.5.0/docs/api/java/util/Scanner.html

Chapter 11: Strings and Characters

- The String Class
- String Constructors
- String Input and Output
- String Class: Instance Methods
- The Character Class
- Conversion Methods
- The StringBuffer Class
- The StringTokenizer Class
- The Scanner Class
- Command Line Arguments
- Case Studies

Command Line Arguments

The command line is the string of characters we type in order to run a program.

Arguments can be given to commands as options. For example, on our Linux platform (or other Unix system):

- ls
- ls –las
- cat file1 file2 file3 …

where *-las* is the argument for ls and *file1 file2 file3* ... are the arguments for cat.

- User can also supply arguments to his program, e.g.

```
$java CommandLineApp argument1, argument2,  ...
```

- Arguments of main() methods receive these inputs:

```
… main( String[ ] args )
{
    ...
}
```

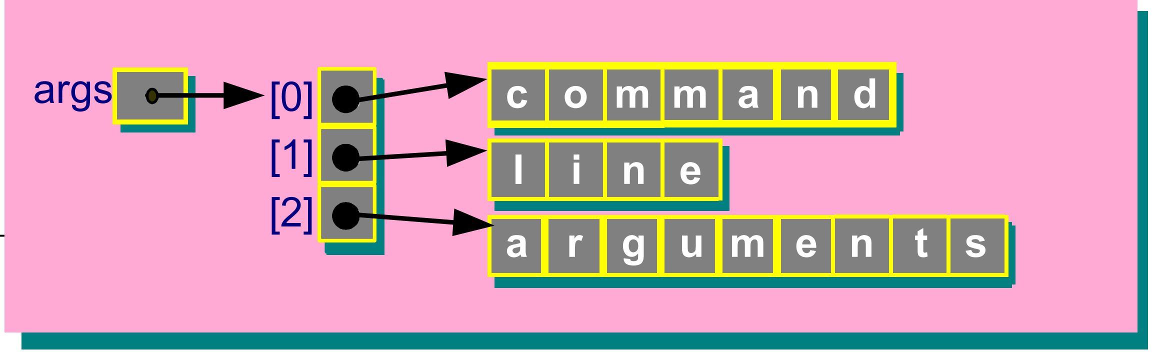
where args is the array of argument strings which stores the user-input arguments from the command line.

However, the command itself is not counted, i.e., args[0] stores argument1, args[1] stores argument2, etc.

```java
import java.util.StringTokenizer ;
public class CommandLineApp {
 public static void main( String[] args ) {
  int i;
  System.out.println( "args.length = " + args.length );
  for ( i = 0 ; i < args.length ; i++)
   System.out.println( "args " + i + " = ["
                       + args[i] + "]" );
  System.out.println();
 }
}
```

Program Output

```
$ java CommandLineApp command line arguments
args.length = 3
args 0 = command
args 1 = line
args 2 = arguments
```



Chapter 11: Strings and Characters

- The String Class
- String Constructors
- String Input and Output
- String Class: Instance Methods
- The Character Class
- Conversion Methods
- The StringBuffer Class
- The StringTokenizer Class
- The Scanner Class
- Command Line Arguments
- Case Studies

Case Studies

Two case studies in the textbook:

(1) Text Editor
– which demonstrates the use of the StringBuffer class.

(2) String Sorting
– which demonstrates the use of the String class and array of strings manipulation.

Case Study: Text Editor Application

Problem Specification

Write a program to implement a text editor application.

The text editor class should contain methods for text operations including insert, replace, delete, find, reverse and case change.

In addition, write an application class to test the text editor class. In the application class, it should allow the user to specify the required text editing operation, and execute the corresponding operation.

Text Editor Application

Class name:

- TextEditor

Data:

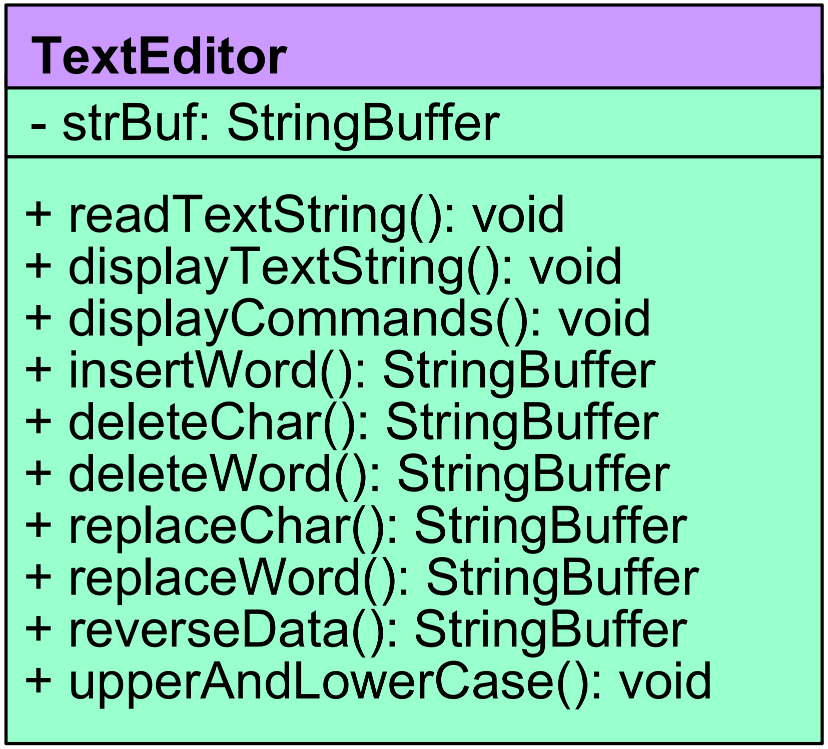
- strBuf: An instance variable to store the string on input sentence

Methods:

- readTextString: A method to read the input string
- displayTextString: A method to display the current string in buffer
- displayCommands: A method to display the command menu
- insertWord: A method to insert a word or character
- deleteChar: A method to delete a character
- deleteWord: A method to delete a word
- replaceChar: A method replace a character
- replaceWord: A method replace a word
- reverseData: A method to reverse the content of the string
- upperAndLowerCase: A method to display the string content from uppercase to lowercase or from lowercase to uppercase

Text Editor Application

Class diagram:



TextEditor

- strBuf: StringBuffer

+ readTextString(): void
+ displayTextString(): void
+ displayCommands(): void
+ insertWord(): StringBuffer
+ deleteChar(): StringBuffer
+ deleteWord(): StringBuffer
+ replaceChar(): StringBuffer
+ replaceWord(): StringBuffer
+ reverseData(): StringBuffer
+ upperAndLowerCase(): void

Application Class name:

- **TextEditorApp**

Method:

- **main: A method to start the text editor application.**

[Refer to the textbook for the implementation]

Testing: Program input and output

```
Enter a string sentence to edit:
NTU Computer Engineering
====== Text Editor ======
To DELETE a character/word
To INSERT a character/word
To REPLACE a character/word
To REVERSE the sentence
...
To QUIT
Command> INSERT
Enter the words to insert: School
At index: 3
NTU School Computer Engineering
Command> REPLACE
Enter 1 to replace character or 2 to replace word: 2
Enter the new word: Material
Enter the start and end indexes: 11 19
NTU School Material Engineering
Command> DELETE
Enter 1 to delete character or 2 to delete word: 2
Enter start and end indexes: 19 28
NTU School Materialing
Command> QUIT
```

Case Study: String Sorting Application

Problem Specification

Write a Java program to implement a string sorting application.

The SortingString class contains methods to support string sorting operations including

(1) readStrings()   to read a number of strings

(2) displayStrings() to display the strings

(3) sortStrings()   to sort the stored strings

Write an application class to test the application.

SortingStrings Application

Class name:

- SortingStrings

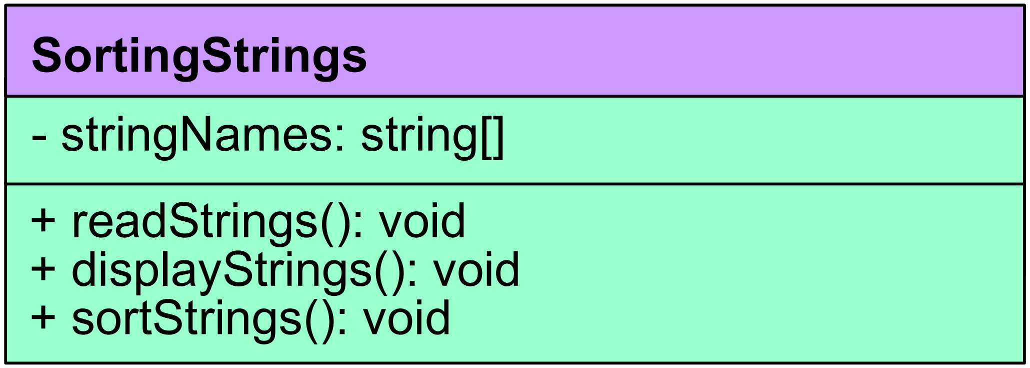
Data:

- stringNames: An instance variable to store an array of string names.

Methods:

- sortStrings: To sort the strings in the array.
- displayStrings: To display the sorted strings.
- readStrings: To read the strings.

Class Diagram:



SortingStrings

- stringNames: string[]

+ readStrings(): void
+ displayStrings(): void
+ sortStrings(): void

Application Class name:

- SortStringsApp

Method:

- main: A method to start the string sorting application.

[Refer to the textbook for the implementation]

Testing: Program input and output

```
Enter names: Kim
Enter names: Ken
Enter names: Tom
Enter names: Kathy
Enter names: Brad
Before sorting:
Kim
Ken
Tom
Kathy
Brad
After sorting:
Brad
Kathy
Ken
Kim
Tom
```

Further Reading

- Read Chapter 11 on “Strings and Characters” of the textbook.
- Read the case studies from the chapter.

Thank you !!!