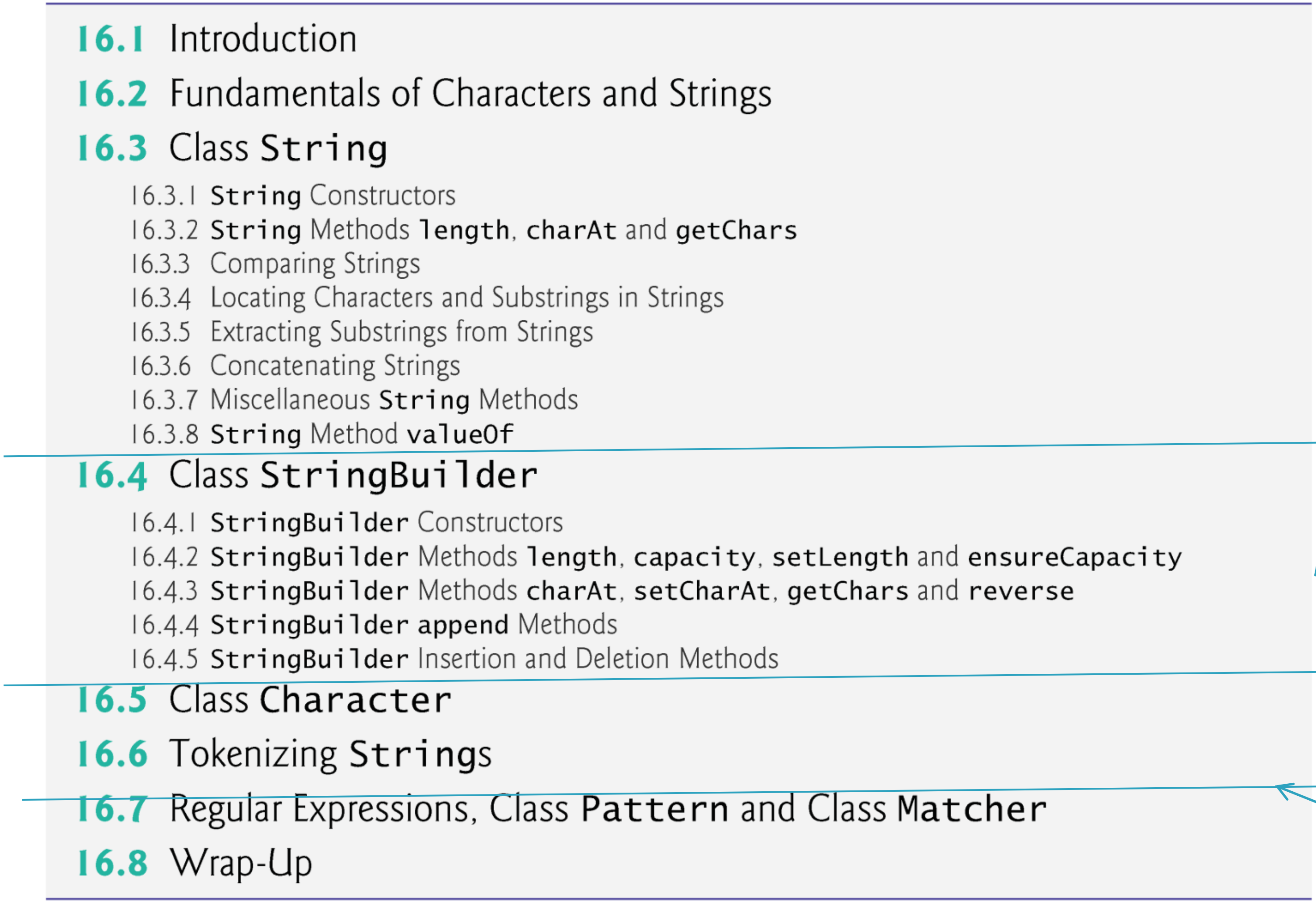
Lesson 6 – Part II Strings, Tokenization, Characters

Assoc. Prof. Marenglen Biba

In this Chapter you'll learn:

- To create and manipulate immutable character-string objects of class `String`.
- To create and manipulate mutable character-string objects of class `StringBuilder`.
- To create and manipulate objects of class `Character`.
- To break a `String` object into tokens using `String` method `split`.
- To use regular expressions to validate `String` data entered into an application.



Advanced Java

16.1 Introduction

- This chapter discusses class `String`, class `StringBuilder` and class `Character` from the `java.lang` package.

- These classes provide the foundation for string and character manipulation in Java.

16.2 Fundamentals of Characters and Strings

- A program may contain character literals.
  - An integer value represented as a character in single quotes.
  - The value of a character literal is the integer value of the character in the Unicode character set.
- String literals (stored in memory as `String` objects) are written as a sequence of characters in double quotation marks.

16.3 Class String

- Class String is used to represent strings in Java.
- The next several subsections cover many of class String's capabilities.

16.3.1 `String` Constructors

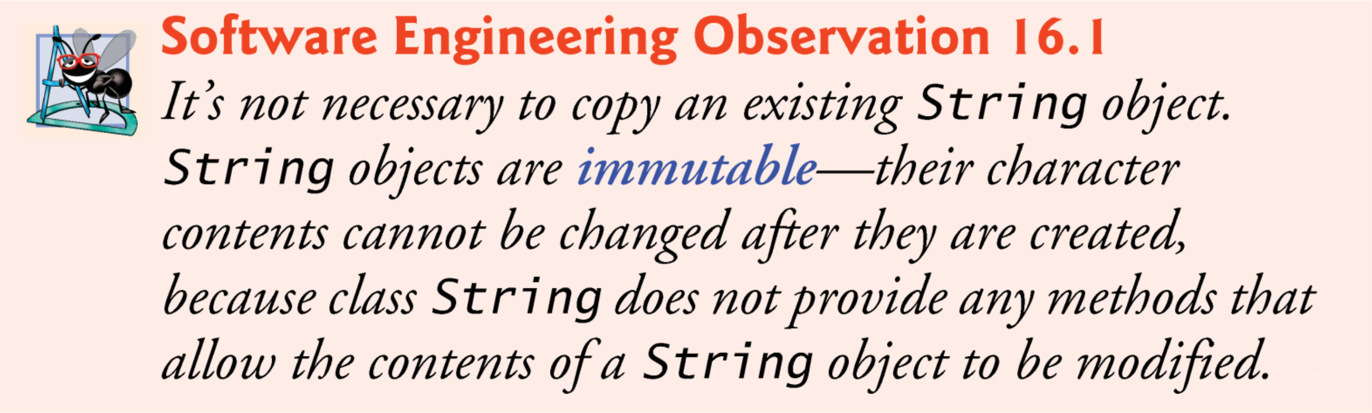
- No-argument constructor creates a `String` that contains no characters (i.e., the empty string, which can also be represented as `""`) and has a length of 0.
- Constructor that takes a `String` object copies the argument into the new `String`.
- Constructor that takes a `char` array creates a `String` containing a copy of the characters in the array.
- Constructor that takes a `char` array and two integers creates a `String` containing the specified portion of the array.

```java
// Fig. 16.1: StringConstructors.java
// String class constructors.

public class StringConstructors
{
   public static void main( String[] args )
   {
      char[] charArray = { 'b', 'i', 'r', 't', 'h', ' ', 'd', 'a', 'y' };
      String s = new String( "hello" );

      // use String constructors
      String s1 = new String();
      String s2 = new String( s );
      String s3 = new String( charArray );
      String s4 = new String( charArray, 6, 3 );

      System.out.printf(
         "s1 = %s\ns2 = %s\ns3 = %s\ns4 = %s\n",
         s1, s2, s3, s4 ); // display strings
   } // end main
} // end class StringConstructors
```

Fig. 16.1 | String class constructors. (Part 1 of 2.)

```
s1 =
s2 = hello
s3 = birth day
s4 = day
```

Fig. 16.1 | `String` class constructors. (Part 2 of 2.)



Software Engineering Observation 16.1

It's not necessary to copy an existing `String` *object.* `String` *objects are* ***immutable****—their character contents cannot be changed after they are created, because class* `String` *does not provide any methods that allow the contents of a* `String` *object to be modified.*

Common Programming Error 16.1

Accessing a character outside the bounds of a `String` *(i.e., an index less than 0 or an index greater than or equal to the* `String`*'s length) results in a* `StringIndexOutOfBoundsException`*.*

16.3.2 String Methods length, charAt and getChars

- String method length determines the number of characters in a string.
- String method charAt returns the character at a specific position in the String.
- String method getChars copies the characters of a String into a character array.
  - The first argument is the starting index in the String from which characters are to be copied.
  - The second argument is the index that is one past the last character to be copied from the String.
  - The third argument is the character array into which the characters are to be copied.
  - The last argument is the starting index where the copied characters are placed in the target character array.

```java
// Fig. 16.2: StringMiscellaneous.java
// This application demonstrates the length, charAt and getChars
// methods of the String class.

public class StringMiscellaneous
{
   public static void main( String[] args )
   {
      String s1 = "hello there";
      char[] charArray = new char[ 5 ];

      System.out.printf( "s1: %s", s1 );

      // test length method
      System.out.printf( "\nLength of s1: %d", s1.length() );

      // loop through characters in s1 with charAt and display reversed
      System.out.print( "\nThe string reversed is: " );

      for ( int count = s1.length() - 1; count >= 0; count-- )
         System.out.printf( "%c ", s1.charAt( count ) );

```

Fig. 16.2 | String class character-manipulation methods. (Part 1 of 2.)

```java
      // copy characters from string into charArray
      s1.getChars( 0, 5, charArray, 0 );
      System.out.print( "\nThe character array is: " );

      for ( char character : charArray )
         System.out.print( character );

      System.out.println();
   } // end main
} // end class StringMiscellaneous
```

```
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
```

Fig. 16.2 | String class character-manipulation methods. (Part 2 of 2.)

16.3.3 Comparing Strings

- Strings are compared using the numeric codes of the characters in the strings.
- Figure 16.3 demonstrates `String` methods `equals`, `equalsIgnoreCase`, `compareTo` and `regionMatches` and using the equality operator == to compare `String` objects (only compares objects).

```java
// Fig. 16.3: StringCompare.java
// String methods equals, equalsIgnoreCase, compareTo and regionMatches.

public class StringCompare
{
   public static void main( String[] args )
   {
      String s1 = new String( "hello" ); // s1 is a copy of "hello"
      String s2 = "goodbye";
      String s3 = "Happy Birthday";
      String s4 = "happy birthday";

      System.out.printf(
         "s1 = %s\ns2 = %s\ns3 = %s\ns4 = %s\n\n", s1, s2, s3, s4 );

      // test for equality
      if ( s1.equals( "hello" ) )  // true
         System.out.println( "s1 equals \"hello\"" );
      else
         System.out.println( "s1 does not equal \"hello\"" );

```

Fig. 16.3 | String methods equals, equalsIgnoreCase, compareTo and regionMatches. (Part 1 of 4.)

```java
      // test for equality with ==
      if ( s1 == "hello" ) // false; they are not the same object
         System.out.println( "s1 is the same object as \"hello\"" );
      else
         System.out.println( "s1 is not the same object as \"hello\"" );

      // test for equality (ignore case)
      if ( s3.equalsIgnoreCase( s4 ) ) // true
         System.out.printf( "%s equals %s with case ignored\n", s3, s4 );
      else
         System.out.println( "s3 does not equal s4" );

      // test compareTo
      System.out.printf(
         "\ns1.compareTo( s2 ) is %d", s1.compareTo( s2 ) );
      System.out.printf(
         "\ns2.compareTo( s1 ) is %d", s2.compareTo( s1 ) );
      System.out.printf(
         "\ns1.compareTo( s1 ) is %d", s1.compareTo( s1 ) );
      System.out.printf(
         "\ns3.compareTo( s4 ) is %d", s3.compareTo( s4 ) );
      System.out.printf(
         "\ns4.compareTo( s3 ) is %d\n\n", s4.compareTo( s3 ) );
```

Fig. 16.3 | `String` methods `equals`, `equalsIgnoreCase`, `compareTo` and `regionMatches`. (Part 2 of 4.)

```java

      // test regionMatches (case sensitive)
      if ( s3.regionMatches( 0, s4, 0, 5 ) )
         System.out.println( "First 5 characters of s3 and s4 match" );
      else
         System.out.println(
            "First 5 characters of s3 and s4 do not match" );

      // test regionMatches (ignore case)
      if ( s3.regionMatches( true, 0, s4, 0, 5 ) )
         System.out.println(
            "First 5 characters of s3 and s4 match with case ignored" );
      else
         System.out.println(
            "First 5 characters of s3 and s4 do not match" );
   } // end main
} // end class StringCompare
```

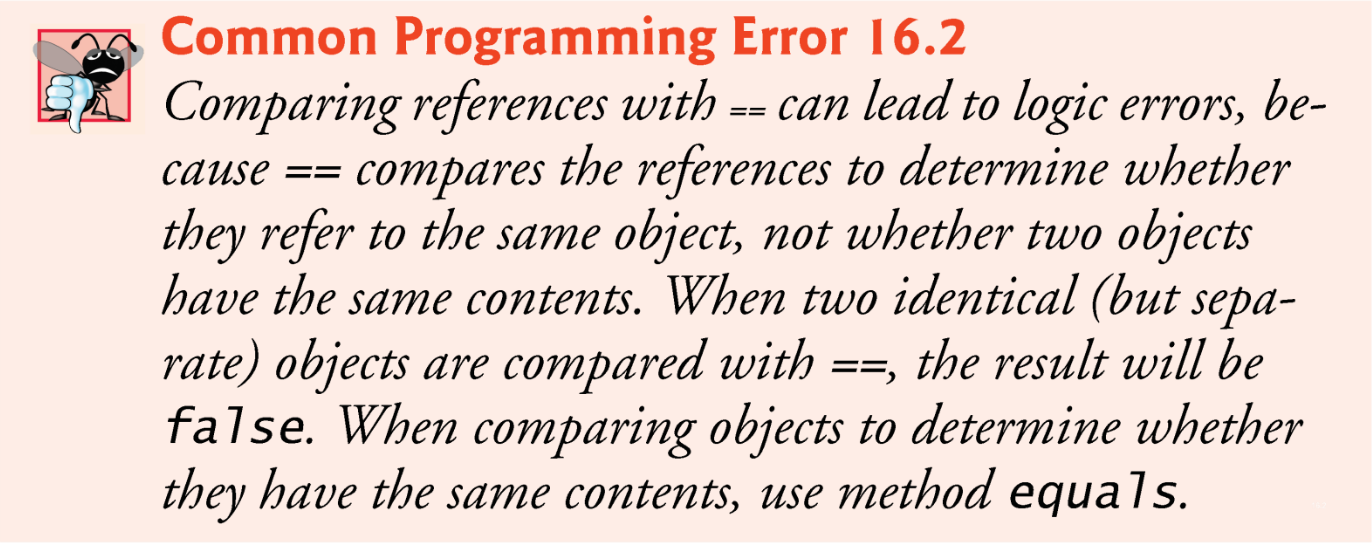
Fig. 16.3 | `String` methods `equals`, `equalsIgnoreCase`, `compareTo` and `regionMatches`. (Part 3 of 4.)

```
s1 = hello
s2 = goodbye
s3 = Happy Birthday
s4 = happy birthday

s1 equals "hello"
s1 is not the same object as "hello"
Happy Birthday equals happy birthday with case ignored

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 with case ignored
```

Fig. 16.3 | `String` methods `equals`, `equalsIgnoreCase`, `compareTo` and `regionMatches`. (Part 4 of 4.)

16.3.3 Comparing Strings (cont.)

- Method `equals` tests any two objects for equality
  - The method returns `true` if the contents of the objects are equal, and `false` otherwise.
  - Uses a lexicographical comparison.
- When primitive-type values are compared with ==, the result is `true` if both values are identical.
- When references are compared with ==, the result is `true` if both references refer to the same object in memory.
- Java treats all string literal objects with the same contents as one `String` object to which there can be many references.



Common Programming Error 16.2

Comparing references with `==` can lead to logic errors, because `==` compares the references to determine whether they refer to the same object, not whether two objects have the same contents. When two identical (but separate) objects are compared with `==`, the result will be `false`*. When comparing objects to determine whether they have the same contents, use method* `equals`*.*

16.3.3 Comparing Strings (cont.)

- `String` methods `startsWith` and `endsWith` determine whether strings start with or end with a particular set of characters

```java
// Fig. 16.4: StringStartEnd.java
// String methods startsWith and endsWith.

public class StringStartEnd
{
   public static void main( String[] args )
   {
      String[] strings = { "started", "starting", "ended", "ending" };

      // test method startsWith
      for ( String string : strings )
      {
         if ( string.startsWith( "st" ) )
            System.out.printf( "\"%s\" starts with \"st\"\n", string );
      } // end for

      System.out.println();

```

Fig. 16.4 | `String` methods `startsWith` and `endsWith`. (Part 1 of 3.)

```
      // test method startsWith starting from position 2 of string
      for ( String string : strings )
      {
         if ( string.startsWith( "art", 2 ) )
            System.out.printf(
               "\"%s\" starts with \"art\" at position 2\n", string );
      } // end for

      System.out.println();

      // test method endsWith
      for ( String string : strings )
      {
         if ( string.endsWith( "ed" ) )
            System.out.printf( "\"%s\" ends with \"ed\"\n", string );
      } // end for
   } // end main
} // end class StringStartEnd
```

Fig. 16.4 | `String` methods `startsWith` and `endsWith`. (Part 2 of 3.)

```
"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"
```

Fig. 16.4 | `String` methods `startsWith` and `endsWith`. (Part 3 of 3.)

16.3.4 Locating Characters and Substrings in Strings

- Figure 16.5 demonstrates the many versions of `String` methods `indexOf` and `lastIndexOf` that search for a specified character or substring in a `String`.

- **indexOf**(String str, int fromIndex)
  Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

- **lastIndexOf** (int ch, int fromIndex)
  Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.

```java
// Fig. 16.5: StringIndexMethods.java
// String searching methods indexOf and lastIndexOf.

public class StringIndexMethods
{
   public static void main( String[] args )
   {
      String letters = "abcdefghijklmabcdefghijklm";

      // test indexOf to locate a character in a string
      System.out.printf(
         "'c' is located at index %d\n", letters.indexOf( 'c' ) );
      System.out.printf(
         "'a' is located at index %d\n", letters.indexOf( 'a', 1 ) );
      System.out.printf(
         "'$' is located at index %d\n\n", letters.indexOf( '$' ) );

      // test lastIndexOf to find a character in a string
      System.out.printf( "Last 'c' is located at index %d\n",
         letters.lastIndexOf( 'c' ) );
      System.out.printf( "Last 'a' is located at index %d\n",
         letters.lastIndexOf( 'a', 25 ) );
      System.out.printf( "Last '$' is located at index %d\n\n",
         letters.lastIndexOf( '$' ) );
```

Fig. 16.5 | String-searching methods `indexOf` and `lastIndexOf`. (Part 1 of 3.)

```java

      // test indexOf to locate a substring in a string
      System.out.printf( "\"def\" is located at index %d\n",
         letters.indexOf( "def" ) );
      System.out.printf( "\"def\" is located at index %d\n",
         letters.indexOf( "def", 7 ) );
      System.out.printf( "\"hello\" is located at index %d\n\n",
         letters.indexOf( "hello" ) );

      // test lastIndexOf to find a substring in a string
      System.out.printf( "Last \"def\" is located at index %d\n",
         letters.lastIndexOf( "def" ) );
      System.out.printf( "Last \"def\" is located at index %d\n",
         letters.lastIndexOf( "def", 25 ) );
      System.out.printf( "Last \"hello\" is located at index %d\n",
         letters.lastIndexOf( "hello" ) );
   } // end main
} // end class StringIndexMethods
```

Fig. 16.5 | String-searching methods `indexOf` and `lastIndexOf`. (Part 2 of 3.)

```
'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
```

Fig. 16.5 | String-searching methods `indexOf` and `lastIndexOf`. (Part 3 of 3.)

16.3.5 Extracting Substrings from Strings

- Class `String` provides two `substring` methods to enable a new `String` object to be created by copying part of an existing `String` object. Each method returns a new `String` object.
- The version that takes one integer argument specifies the starting index in the original `String` from which characters are to be copied.
- The version that takes two integer arguments receives the starting index from which to copy characters in the original `String` and the index one beyond the last character to copy.

```java
// Fig. 16.6: SubString.java
// String class substring methods.

public class SubString
{
   public static void main( String[] args )
   {
      String letters = "abcdefghijklmabcdefghijklm";

      // test substring methods
      System.out.printf( "Substring from index 20 to end is \"%s\"\n",
         letters.substring( 20 ) );
      System.out.printf( "%s \"%s\"\n",
         "Substring from index 3 up to, but not including 6 is",
         letters.substring( 3, 6 ) );
   } // end main
} // end class SubString
```

```
Substring from index 20 to end is "hijklm"
Substring from index 3 up to, but not including 6 is "def"
```

Fig. 16.6 | `String` class `substring` methods.

16.3.6 Concatenating Strings

- `String` method `concat` concatenates two `String` objects and returns a new `String` object containing the characters from both original `String`s.
- The original `String`s to which `s1` and `s2` refer are not modified.

```java
// Fig. 16.7: StringConcatenation.java
// String method concat.

public class StringConcatenation
{
   public static void main( String[] args )
   {
      String s1 = "Happy ";
      String s2 = "Birthday";

      System.out.printf( "s1 = %s\ns2 = %s\n\n",s1, s2 );
      System.out.printf(
         "Result of s1.concat( s2 ) = %s\n", s1.concat( s2 ) );
      System.out.printf( "s1 after concatenation = %s\n", s1 );
   } // end main
} // end class StringConcatenation
```

```
s1 = Happy
s2 = Birthday

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

Fig. 16.7 | String method concat.

16.3.7 Miscellaneous String Methods

- Method replace return a new String object in which every occurrence of the first char argument is replaced with the second.
  - An overloaded version enables you to replace substrings rather than individual characters.
- Method toUpperCase generates a new String with uppercase letters.
- Method toLowerCase returns a new String object with lowercase letters.
- Method trim generates a new String object that removes all whitespace characters that appear at the beginning or end of the String on which trim operates.
- Method toCharArray creates a new character array containing a copy of the characters in the String.

```java
// Fig. 16.8: StringMiscellaneous2.java
// String methods replace, toLowerCase, toUpperCase, trim and toCharArray.

public class StringMiscellaneous2
{
   public static void main( String[] args )
   {
      String s1 = "hello";
      String s2 = "GOODBYE";
      String s3 = "   spaces   ";

      System.out.printf( "s1 = %s\ns2 = %s\ns3 = %s\n\n", s1, s2, s3 );

      // test method replace
      System.out.printf(
         "Replace 'l' with 'L' in s1: %s\n\n", s1.replace( 'l', 'L' ) );

      // test toLowerCase and toUpperCase
      System.out.printf( "s1.toUpperCase() = %s\n", s1.toUpperCase() );
      System.out.printf( "s2.toLowerCase() = %s\n\n", s2.toLowerCase() );

```

Fig. 16.8 | `String` methods `replace`, `toLowerCase`, `toUpperCase`, `trim` and `toCharArray`. (Part 1 of 3.)

```java
      // test trim method
      System.out.printf( "s3 after trim = \"%s\"\n\n", s3.trim() );

      // test toCharArray method
      char[] charArray = s1.toCharArray();
      System.out.print( "s1 as a character array = " );

      for ( char character : charArray )
         System.out.print( character );

      System.out.println();
   } // end main
} // end class StringMiscellaneous2
```

Fig. 16.8 | `String` methods `replace`, `toLowerCase`, `toUpperCase`, `trim` and `toCharArray`. (Part 2 of 3.)

```
s1 = hello
s2 = GOODBYE
s3 =    spaces

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

s1.toUpperCase() = HELLO
s2.toLowerCase() = goodbye

s3 after trim = "spaces"

s1 as a character array = hello
```

Fig. 16.8 | `String` methods `replace`, `toLowerCase`, `toUpperCase`, `trim` and `toCharArray`. (Part 3 of 3.)

16.3.8 String Method valueOf

- Class `String` provides `static valueOf` methods that take an argument of any type and convert it to a `String` object.
- Class `StringBuilder` is used to create and manipulate dynamic string information.
- Every `StringBuilder` is capable of storing a number of characters specified by its capacity.
- If the capacity of a `StringBuilder` is exceeded, the capacity expands to accommodate the additional characters.

```java
// Fig. 16.9: StringValueOf.java
// String valueOf methods.

public class StringValueOf
{
   public static void main( String[] args )
   {
      char[] charArray = { 'a', 'b', 'c', 'd', 'e', 'f' };
      boolean booleanValue = true;
      char characterValue = 'Z';
      int integerValue = 7;
      long longValue = 10000000000L; // L suffix indicates long
      float floatValue = 2.5f; // f indicates that 2.5 is a float
      double doubleValue = 33.333; // no suffix, double is default
      Object objectRef = "hello"; // assign string to an Object reference

      System.out.printf(
         "char array = %s\n", String.valueOf( charArray ) );
      System.out.printf( "part of char array = %s\n",
         String.valueOf( charArray, 3, 3 ) );
      System.out.printf(
         "boolean = %s\n", String.valueOf( booleanValue ) );
      System.out.printf(
         "char = %s\n", String.valueOf( characterValue ) );
```

Fig. 16.9 | `String valueOf` methods. (Part 1 of 2.)

```java
      System.out.printf( "int = %s\n", String.valueOf( integerValue ) );
      System.out.printf( "long = %s\n", String.valueOf( longValue ) );
      System.out.printf( "float = %s\n", String.valueOf( floatValue ) );
      System.out.printf(
         "double = %s\n", String.valueOf( doubleValue ) );
      System.out.printf( "Object = %s", String.valueOf( objectRef ) );
   } // end main
} // end class StringValueOf
```

```
char array = abcdef
part of char array = def
boolean = true
char = Z
int = 7
long = 10000000000
float = 2.5
double = 33.333
Object = hello
```

Fig. 16.9 | String valueOf methods. (Part 2 of 2.)

16.5 Class Character

- Eight type-wrapper classes that enable primitive-type values to be treated as objects:
  - `Boolean`, `Character`, `Double`, `Float`, `Byte`, `Short`, `Integer` and `Long`
- Most `Character` methods are `static` methods designed for convenience in processing individual `char` values.

16.5 Class Character (cont.)

- Method `isDefined` determines whether a character is defined in the Unicode character set.
- Method `isDigit` determines whether a character is a defined Unicode digit.
- Method `isJavaIdentifierStart` determines whether a character can be the first character of an identifier in Java—that is, a letter, an underscore (_) or a dollar sign ($).
- Method `isJavaIdentifierPart` determine whether a character can be used in an identifier in Java—that is, a digit, a letter, an underscore (_) or a dollar sign ($).

```java
// Fig. 16.15: StaticCharMethods.java
// Character static methods for testing characters and converting case.
import java.util.Scanner;

public class StaticCharMethods
{
   public static void main( String[] args )
   {
      Scanner scanner = new Scanner( System.in ); // create scanner
      System.out.println( "Enter a character and press Enter" );
      String input = scanner.next();
      char c = input.charAt( 0 ); // get input character

      // display character info
      System.out.printf( "is defined: %b\n", Character.isDefined( c ) );
      System.out.printf( "is digit: %b\n", Character.isDigit( c ) );
      System.out.printf( "is first character in a Java identifier: %b\n",
         Character.isJavaIdentifierStart( c ) );
      System.out.printf( "is part of a Java identifier: %b\n",
         Character.isJavaIdentifierPart( c ) );
      System.out.printf( "is letter: %b\n", Character.isLetter( c ) );
```

Fig. 16.15 | `Character` `static` methods for testing characters and converting case. (Part 1 of 5.)

```java
      System.out.printf(
         "is letter or digit: %b\n", Character.isLetterOrDigit( c ) );
      System.out.printf(
         "is lower case: %b\n", Character.isLowerCase( c ) );
      System.out.printf(
         "is upper case: %b\n", Character.isUpperCase( c ) );
      System.out.printf(
         "to upper case: %s\n", Character.toUpperCase( c ) );
      System.out.printf(
         "to lower case: %s\n", Character.toLowerCase( c ) );
   } // end main
} // end class StaticCharMethods
```

Fig. 16.15 | `Character` `static` methods for testing characters and converting case. (Part 2 of 5.)

```
Enter a character and press Enter
A
is defined: true
is digit: false
is first character in a Java identifier: true
is part of a Java identifier: true
is letter: true
is letter or digit: true
is lower case: false
is upper case: true
to upper case: A
to lower case: a
```

Fig. 16.15 | `Character` `static` methods for testing characters and converting case. (Part 3 of 5.)

```
Enter a character and press Enter
8
is defined: true
is digit: true
is first character in a Java identifier: false
is part of a Java identifier: true
is letter: false
is letter or digit: true
is lower case: false
is upper case: false
to upper case: 8
to lower case: 8
```

Fig. 16.15 | `Character` `static` methods for testing characters and converting case. (Part 4 of 5.)

```
Enter a character and press Enter
$
is defined: true
is digit: false
is first character in a Java identifier: true
is part of a Java identifier: true
is letter: false
is letter or digit: false
is lower case: false
is upper case: false
to upper case: $
to lower case: $
```

Fig. 16.15 | `Character` `static` methods for testing characters and converting case. (Part 5 of 5.)

16.5 Class Character (cont.)

- Method `isLetter` determines whether a character is a letter.
- Method `isLetterOrDigit` determines whether a character is a letter or a digit.
- Method `isLowerCase` determines whether a character is a lowercase letter.
- Method `isUpperCase` determines whether a character is an uppercase letter.
- Method `toUpperCase` converts a character to its uppercase equivalent.
- Method `toLowerCase` converts a character to its lowercase equivalent.

16.5 Class Character (cont.)

- Java automatically converts `char` literals into `Character` objects when they are assigned to `Character` variables
  - Process known as autoboxing.
- Method `charValue` returns the `char` value stored in the object.
- Method `toString` returns the `String` representation of the `char` value stored in the object.
- Method `equals` determines if two `Character`s have the same contents.

```java
// Fig. 16.17: OtherCharMethods.java
// Character class non-static methods.
public class OtherCharMethods
{
   public static void main( String[] args )
   {
      Character c1 = 'A';
      Character c2 = 'a';

      System.out.printf(
         "c1 = %s\nc2 = %s\n\n", c1.charValue(), c2.toString() );

      if ( c1.equals( c2 ) )
         System.out.println( "c1 and c2 are equal\n" );
      else
         System.out.println( "c1 and c2 are not equal\n" );
   } // end main
} // end class OtherCharMethods
```

```
c1 = A
c2 = a

c1 and c2 are not equal
```

Fig. 16.17 | `Character` class non-`static` methods.

16.6 Tokenizing `String`s

- When you read a sentence, your mind breaks it into tokens—individual words and punctuation marks that convey meaning.
- Compilers also perform tokenization.
- `String` method `split` breaks a `String` into its component tokens and returns an array of `String`s.
- Tokens are separated by delimiters
  - Typically white-space characters such as space, tab, newline and carriage return.
  - Other characters can also be used as delimiters to separate tokens.

```java
// Fig. 16.18: TokenTest.java
// StringTokenizer object used to tokenize strings.
import java.util.Scanner;
import java.util.StringTokenizer;

public class TokenTest
{
   // execute application
   public static void main( String[] args )
   {
      // get sentence
      Scanner scanner = new Scanner( System.in );
      System.out.println( "Enter a sentence and press Enter" );
      String sentence = scanner.nextLine();

      // process user sentence
      String[] tokens = sentence.split( " " );
      System.out.printf( "Number of elements: %d\nThe tokens are:\n",
         tokens.length );

      for ( String token : tokens )
         System.out.println( token );
   } // end main
} // end class TokenTest
```

Fig. 16.18 | StringTokenizer object used to tokenize strings. (Part 1 of 2.)

```
Enter a sentence and press Enter
This is a sentence with seven tokens
Number of elements: 7
The tokens are:
This
is
a
sentence
with
seven
tokens
```

Fig. 16.18 | `StringTokenizer` object used to tokenize strings. (Part 2 of 2.)

Lab Session

- ***Savings Account Class.*** Create class SavingsAccount.
- Use a static variable annualInterestRate to store the annual interest rate for all account holders. Each object of the class contains a private instance variable savingsBalance indicating the amount the saver currently has on deposit.
- Provide method calculateMonthlyInterest to calculate the monthly interest by multiplying the savingsBalance by annualInterestRate divided by 12 - this interest should be added to savings-Balance.
- Provide a static method modifyInterestRate that sets the annualInterestRate to a new value.
- Write a program to test class SavingsAccount.
  - Instantiate two savingsAccount objects, saver1 and saver2, with balances of $2000.00 and $3000.00, respectively.
  - Set annualInterestRate to 4%, then calculate the monthly interest for each of 12 months and print the new balances for both savers.
  - Next, set the annualInterestRate to 5%, calculate the next month's interest and print the new balances for both savers.

Lab Session

- Create a class Rectangle with attributes length and width, each of which defaults to 1.
- Provide methods that calculate the rectangle’s perimeter and area.
- It has set and get methods for both length and width.
- The set methods should verify that length and width are each floating-point numbers larger than 0.0 and less than 20.0.
- Write a program to test class Rectangle.

End of Class

- Readings part II
  - Chapter 16