Лекция 8

Strings, Characters and Regular Expressions

OBJECTIVES

In this lecture you will learn:

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

Outline

8.1 Introduction

String and character processing

- Class `java.lang.String`
- Class `java.lang.StringBuilder`
- Class `java.lang.Character`
- Class `java.util.StringTokenizer`

Regular expressions

- Valid input
- Package `java.util.regex`
- Classes `Matcher` and `Pattern`

8.2 Fundamentals of Characters and Strings

Characters

- "Building blocks" of Java source programs
- Character literals
- Unicode character set

String

- Series of characters treated as single unit
- May include letters, digits, special characters.
- Object of class `String`
- String literals

Performance Tip

Java treats all string literals with the same contents as a single `String` object that has many references to it. This conserves memory.

8.3 Class `String`

Class `String`

- Represent strings in Java

8.3.1 `String` Constructors

Fig. 8.1 demonstrates four constructors

- **No-argument constructor**
- **One-argument constructor**
  - **A `String` object**
- **One-argument constructor**
  - **A `char` array**
- **Three-argument constructor**
  - **A `char` array**
  - **An integer specifies the starting position**
  - **An integer specifies the number of characters to access**

Software Engineering Observation

It is 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.

Outline

```
1  // Fig. 8.1: StringConstructors.java
2  // String class constructors.
3
4  public class StringConstructors
5  {
6     public static void main( String args[] )
7     {
8        char charArray[] = { 'b', 'i', 'r', 't', 'h', ' ', 'd', 'a', 'y' };
9        String s = new String( "hello" );
10
11       // use String constructors
12       String s1 = new String();
13       String s2 = new String( s );
14       String s3 = new String( charArray );
15       String s4 = new String( charArray, 6, 3 );
16
17       System.out.printf(
18          "s1 = %s\ns2 = %s\ns3 = %s\ns4 = %s\n",
19          s1, s2, s3, s4 ); // display strings
20    } // end main
21 } // end class StringConstructors
```

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

StringConstructors.java

Line 12

Line 13

Program output

No-argument constructor creates an empty string

One-argument constructor creates a `String` that contains the same sequence of characters as the `String` argument

One-argument constructor creates a `String` that contains a copy of the characters in the array argument

Three-argument constructor creates a `String` that contains a copy of partial characters in the array argument

Outline

```
1  // Special case
2  // String class constructors.
3
4  public class StringConstructors
5  {
6     public static void main( String args[] )
7     {
8        String s1 = "smart";
9        String s2 = "sm" + "art";
10       String s3 = " boy";
11       if (s1 == s2)    //equal strings
12           System.out.println("s1 equals s2");
13       else
14           System.out.println("s1 is not equal to s2");
15       if ((s2 + s3) == "smart boy") //different  strings
16           System.out.println("s2 + s3 equals \"smart boy\"");
17       else
18          System.out.println("s2 + s3 equals \"smart boy\"");
19
20    } // end main
21 } // end class StringConstructors
```

Expression is evaluated at compile time

Compares one and the same string

Expression is evaluated at runtime

Compares two different strings

```
s1 equals s2
s2 + s3 is not equal to "smart boy"
```

Common Programming Error

Attempting to access a character that is 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`*.

8.3.2 `String` Methods `length`, `charAt` and `getChars`

Method `length`

- Determine the length of a `String`
  - 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`

Outline

StringMiscellaneous.java

(1 of 2)

Line 15

```
1  // Fig. 8.2: StringMiscellaneous.java
2  // This application demonstrates the length, charAt and getChars
3  // methods of the String class.
4
5  public class StringMiscellaneous
6  {
7     public static void main( String args[] )
8     {
9        String s1 = "hello there";
10       char charArray[] = new char[ 5 ];
11
12       System.out.printf( "s1: %s", s1 );
13
14       // test length method
15       System.out.printf( "\nLength of s1: %d", s1.length() );
16
17       // loop through characters in s1 with charAt and display reversed
18       System.out.print( "\nThe string reversed is: " );
19
20       for ( int count = s1.length() - 1; count >= 0; count-- )
21          System.out.printf( "%s ", s1.charAt( count ) );
22
```

Determine number of characters in `String s1`

Display the characters of the string `s1` in reverse order

Outline

```
23       // copy characters from string into charArray
24       s1.getChars( 0, 5, charArray, 0 );
25       System.out.print( "\nThe character array is: " );
26
27       for ( char character : charArray )
28          System.out.print( character );
29
30       System.out.println();
31    } // end main
32 } // end class StringMiscellaneous
```

Copy (some of) `s1`'s characters to `charArray`

```
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
```

`StringMiscellaneous.java`

(2 of 2)

Line 24

Program output

8.3.3 Comparing Strings

Comparing `String` objects

- Method `equals`
- Method `equalsIgnoreCase`
- Method `compareTo`
- Method `regionMatches`

Outline

StringCompare.java

(1 of 3)

Line 17

Line 23

```
1  // Fig. 8.3: StringCompare.java
2  // String methods equals, equalsIgnoreCase, compareTo and regionMatches.
3
4  public class StringCompare
5  {
6     public static void main( String args[] )
7     {
8        String s1 = new String( "hello" ); // s1 is a copy of "hello"
9        String s2 = "goodbye";
10       String s3 = "Happy Birthday";
11       String s4 = "happy birthday";
12
13       System.out.printf(
14          "s1 = %s\ns2 = %s\ns3 = %s\ns4 = %s\n\n", s1, s2, s3, s4 );
15
16       // test for equality
17       if ( s1.equals( "hello" ) ) // true
18          System.out.println( "s1 equals \"hello\"" );
19       else
20          System.out.println( "s1 does not equal \"hello\"" );
21
22       // test for equality with ==
23       if ( s1 == "hello" )  // false; they are not the same object
24          System.out.println( "s1 is the same object as \"hello\"" );
25       else
26          System.out.println( "s1 is not the same object as \"hello\"" );
27
```

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

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

Outline

StringCompare.java

(2 of 3)

```
28    // test for equality (ignore case)
29    if ( s3.equalsIgnoreCase( s4 ) )  // true
30       System.out.printf( "%s equals %s with
31    else
32       System.out.println( "s3 does not equal s4" );
33
34    // test compareTo
35    System.out.printf(
36       "\ns1.compareTo( s2 ) is %d", s1.compareTo( s2 ) );
37    System.out.printf(
38       "\ns2.compareTo( s1 ) is %d", s2.compareTo( s1 ) );
39    System.out.printf(
40       "\ns1.compareTo( s1 ) is %d", s1.compareTo( s1 ) );
41    System.out.printf(
42       "\ns3.compareTo( s4 ) is %d", s3.compareTo( s4 ) );
43    System.out.printf(
44       "\ns4.compareTo( s3 ) is %d\n\n", s4.compareTo( s3 ) );
45
46    // test regionMatches (case sensitive)
47    if ( s3.regionMatches( 0, s4, 0, 5 ) )
48       System.out.println( "First 5 characters of s3 and s4
49    else
50       System.out.println(
51          "First 5 characters of s3 and s4 do not match" );
52
```

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

Line 29

Method `compareTo` compares `String` objects

Line 47

Method `regionMatches` compares portions of two `String` objects for equality

Outline

```
53       // test regionMatches (ignore case)
54       if ( s3.regionMatches( true, 0, s4, 0, 5 ) )
55          System.out.println( "First 5 characters of s3 and s4 match" );
56       else
57          System.out.println(
58             "First 5 characters of s3 and s4 do not match" );
59    } // end main
60 } // end class StringCompare
```

```
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
```

StringCompare.java

(3 of 3)

Program output

Common Programming Error

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`.

Outline

StringStartEnd.java

(1 of 2)

Lines 13 and 22

```
1  // Fig. 8.4: StringStartEnd.java
2  // String methods startsWith and endsWith.
3
4  public class StringStartEnd
5  {
6     public static void main( String args[] )
7     {
8        String strings[] = { "started", "starting", "ended", "ending" };
9
10       // test method startsWith
11       for ( String string : strings )
12       {
13          if ( string.startsWith( "st" ) )
14             System.out.printf( "\"%s\" starts with \"st\"\n", string );
15       } // end for
16
17       System.out.println();
18
19       // test method startsWith starting from position 2 of string
20       for ( String string : strings )
21       {
22          if ( string.startsWith( "art", 2 ) )
23             System.out.printf(
24                "\"%s\" starts with \"art\" at position 2\n", string );
25       } // end for
26
27       System.out.println();
28
```

Method `startsWith` determines if `String` starts with specified characters

```
29       // test method endsWith
30       for ( String string : strings )
31       {
32          if ( string.endsWith( "ed" ) )
33             System.out.printf( "\"%s\" ends with \"ed\"\n", string );
34       } // end for
35    } // end main
36 } // end class StringStartEnd
```

Method `endsWith` determines if `String` ends with specified characters

```
"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"
```

Outline

`StringStartEnd.java`

(2 of 2)

Line 32

Program output

8.3.4 Locating Characters and Substrings in Strings

Search for characters in `String`

- **Method `indexOf`**
- **Method `lastIndexOf`**

Outline

StringIndexMethods.java

(1 of 3)

```
1  // Fig. 8.5: StringIndexMethods.java
2  // String searching methods indexOf and lastIndexOf.
3
4  public class StringIndexMethods
5  {
6     public static void main( String args[] )
7     {
8        String letters = "abcdefghijklmabcdefghijklm";
9
10       // test indexOf to locate a character in a string
11       System.out.printf(
12          "'c' is located at index %d\n", letters.indexOf( 'c' ) );
13       System.out.printf(
14          "'a' is located at index %d\n", letters.indexOf( 'a', 1 ) );
15       System.out.printf(
16          "'$' is located at index %d\n\n", letters.indexOf( '$' ) );
17
18       // test lastIndexOf to find a character in a string
19       System.out.printf( "Last 'c' is located at in
20          letters.lastIndexOf( 'c' ) );
21       System.out.printf( "Last 'a' is located at in
22          letters.lastIndexOf( 'a', 25 ) );
23       System.out.printf( "Last '$' is located at index %d\n\n",
24          letters.lastIndexOf( '$' ) );
25
```

Method `indexOf` finds first occurrence of character in `String`

Method `lastIndexOf` finds last occurrence of character in `String`

Outline

```
26       // test indexOf to locate a substring in a string
27       System.out.printf( "\"def\" is located at index %d\n",
28          letters.indexOf( "def" ) );
29       System.out.printf( "\"def\" is located at index %d\n",
30          letters.indexOf( "def", 7 ) );
31       System.out.printf( "\"hello\" is located at index %d\n\n",
32          letters.indexOf( "hello" ) );
33
34       // test lastIndexOf to find a substring in a string
35       System.out.printf( "Last \"def\" is located at index %d\n",
36          letters.lastIndexOf( "def" ) );
37       System.out.printf( "Last \"def\" is located at index %d\n",
38          letters.lastIndexOf( "def", 25 ) );
39       System.out.printf( "Last \"hello\" is located at index %d\n",
40          letters.lastIndexOf( "hello" ) );
41    } // end main
42 } // end class StringIndexMethods
```

`StringIndexMethods.java`

Methods `indexOf` and `lastIndexOf` can also find occurrences of substrings

Lines 28, 30, 32, 36, 38 and 40

```
'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
```

Outline

`StringIndexMethods.java`

(3 of 3)

Program output

8.3.5 Extracting Substrings from Strings

- Create `String`s from other `String`s
  - Method `substring`

Outline

```
1  // Fig. 8.6: SubString.java
2  // String class substring methods.
3
4  public class SubString
5  {
6     public static void main( String args[] )
7     {
8        String letters = "abcdefghijklmabcdefghijklm";
9
10       // test substring methods
11       System.out.printf( "Substring from index 20 to end is \"%s\"\n",
12          letters.substring( 20 ) );
13       System.out.printf( "%s \"%s\"\n",
14          "Substring from index 3 up to, but not including 6 is",
15          letters.substring( 3, 6 ) );
16    } // end main
17 } // end class SubString
```

Beginning at index 20, extract characters from `String letters`

Extract characters from index **3** to **6** from `String letters`

SubString.java

Line 12

Line 15

Program output

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

8.3.6 Concatenating Strings

Method `concat`

- **Concatenate two `String` objects**

Outline

```
1  // Fig. 8.7: StringConcatenation.java
2  // String concat method.
3
4  public class StringConcatenation
5  {
6     public static void main( String args[] )
7     {
8        String s1 = new String( "Happy " );
9        String s2 = new String( "Birthday" );
10
11       System.out.printf( "s1 = %s\ns2 = %s\n\n",s1, s2 );
12       System.out.printf(
13          "Result of s1.concat( s2 ) = %s\n", s1.concat( s2 ) );
14       System.out.printf( "s1 after concatenation = %s\n", s1 );
15    } // end main
16 } // end class StringConcatenation
```

```
s1 = Happy
s2 = Birthday

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

Concatenate `String s2` to `String s1`

However, `String s1` is not modified by method `concat`

StringConcatenation.java

Line 13

Line 14

8.3.7 Miscellaneous `String` Methods

Miscellaneous `String` methods

- **Return modified copies of `String`**
- **Return character array**

Outline

```
1  // Fig. 8.8: StringMiscellaneous2.java
2  // String methods replace, toLowerCase, toUpperCase, trim and toCharArray.
3
4  public class StringMiscellaneous2
5  {
6     public static void main( String args[] )
7     {
8        String s1 = new String( "hello" );
9        String s2 = new String( "GOODBYE" );
10       String s3 = new String( "   spaces   " );
11
12       System.out.printf( "s1 = %s\ns2 = %s\ns3 = %s\n\n", s1, s2, s3 );
13
14       // test method replace
15       System.out.printf(
16          "Replace 'l' with 'L' in s1: %s\n\n", s1.replace( 'l', 'L' ) );
17
18       // test toLowerCase and toUpperCase
19       System.out.printf( "s1.toUpperCase() = %s\n", s1.toUpperCase() );
20       System.out.printf( "s2.toLowerCase() = %s\n\n", s2.toLowerCase() );
21
22       // test trim method
23       System.out.printf( "s3 after trim = \"%s\"\n\n", s3.trim() );
24
```

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

`StringMiscellaneous2.java`

(1 of 2)

Line 16

Line 20

Line 23

Outline

```
25       // test toCharArray method
26       char charArray[] = s1.toCharArray();
27       System.out.print( "s1 as a character array = " );
28
29       for ( char character : charArray )
30          System.out.print( character );
31
32       System.out.println();
33    } // end main
34 } // end class StringMiscellaneous2
```

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

`StringMiscellaneous2.java`

(2 of 2)

Line 26

Program output

```
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
```

8.3.8 `String` Method `valueOf`

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

Outline

StringValueOf.java

(1 of 2)

Line 12

Line 13

```
1  // Fig. 8.9: StringValueOf.java
2  // String valueOf methods.
3
4  public class StringValueOf
5  {
6     public static void main( String args[] )
7     {
8        char charArray[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
9        boolean booleanValue = true;
10       char characterValue = 'Z';
11       int integerValue = 7;
12       long longValue = 10_000_000_000L; // L suffix indicates long
13       float floatValue = 2.5f; // f indicates that 2.5 is a float
14       double doubleValue = 33.333; // no suffix, double is default
15       Object objectRef   = "hello"; // assign string to an Object reference
16
```

Use literal value `2.5f` as the initial values of `float` variable

Use literal value `10_000_000_000L` as the initial values of `long` variable

Outline

```
17       System.out.printf(
18          "char array = %s\n", String.valueOf( charArray ) );
19       System.out.printf( "part of char array = %s\n",
20          String.valueOf( charArray, 3, 3 ) );
21       System.out.printf(
22          "boolean = %s\n", String.valueOf( booleanValue ) );
23       System.out.printf(
24          "char = %s\n", String.valueOf( characterValue ) );
25       System.out.printf( "int = %s\n", String.valueOf( integerValue ) );
26       System.out.printf( "long = %s\n", String.valueOf( longValue ) );
27       System.out.printf( "float = %s\n", String.valueOf( floatValue ) );
28       System.out.printf(
29          "double = %s\n", String.valueOf( doubleValue ) );
30       System.out.printf( "Object = %s", String.valueOf( objectRef ) );
31    } // end main
32 } // end class StringValueOf
```

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

(2 of 2)

Lines 18-29

Program output

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

8.4 Class `StringBuilder`

Class `StringBuilder`

- 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

Performance Tip

Java can perform certain optimizations involving `String` objects (such as sharing one `String` object among multiple references) because it knows these objects will not change. `String`s (not `StringBuilder`s) should be used if the data will not change.

Performance Tip

In programs that frequently perform string concatenation, or other string modifications, it is more efficient to implement the modifications with class StringBuilder.

8.4.1 `StringBuilder` Constructors

Four `StringBuilder` constructors

- **No-argument constructor**
  - **Creates `StringBuilder` with no characters**
  - **Capacity of 16 characters**
- **One-argument constructor**
  - **`int` argument**
    - **Specifies the initial capacity**
- **One-argument constructor**
  - **`String` argument**
  - **Creates `StringBuilder` containing the characters in the `String` argument**

Outline

```
1  // Fig. 8.10: StringBuilderConstructors.java
2  // StringBuilder constructors.
3
4  public class StringBuilderConstructors
5  {
6     public static void main( String args[] )
7     {
8        StringBuilder buffer1 = new StringBuilder();
9        StringBuilder buffer2 = new StringBuilder( 10 );
10       StringBuilder buffer3 = new StringBuilder( "hello" );
11
12       System.out.printf( "buffer1 = \"%s\"\n", buffer1.toString() );
13       System.out.printf( "buffer2 = \"%s\"\n", buffer2.toString() );
14       System.out.printf( "buffer3 = \"%s\"\n", buffer3.toString() );
15    } // end main
16 } // end class StringBuilderConstructors
```

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

No-argument constructor creates empty **`StringBuilder`** with capacity of **`16`** characters

One-argument constructor creates empty **`StringBuilder`** with capacity of specified (**`10`**) characters

One-argument constructor creates **`StringBuilder`** with **`String`** "**`hello`**" and default capacity of **`16`** characters. If the number of the character increases from its current capacity, it increases the capacity by `(oldcapacity*2)+2`

Method **`toString`** returns **`String`** representation of **`StringBuilder`**

`StringBuilderConstructors.java`

`Line 8`

`Line 9`

`Line 10`

`Lines 12-14`

`Program output`

8.4.2 StringBuilder Methods length, capacity, setLength and ensureCapacity

- Method `length`
  - Return `StringBuilder` length
- Method `capacity`
  - Return `StringBuilder` capacity
- Method `setLength`
  - Increase or decrease `StringBuilder` length
- Method `ensureCapacity`
  - Set `StringBuilder` capacity
  - Guarantee that `StringBuilder` has minimum capacity

Outline

```
1  // Fig. 8.11: StringBuilderCapLen.java
2  // StringBuilder length, setLength, capacity and ensureCapacity methods.
3
4  public class StringBuilderCapLen
5  {
6     public static void main( String args[] )
7     {
8        StringBuilder buffer = new StringBuilder( "Hello, how are you?" );
9
10       System.out.printf( "buffer = %s\nlength = %d\ncapacity = %d\n\n",
11          buffer.toString(), buffer.length(), buffer.capacity() );
12
13       buffer.ensureCapacity( 75 );
14       System.out.printf( "New capacity = %d\n\n", buffer.capacity() );
15
16       buffer.setLength( 10 );
17       System.out.printf( "New length = %d\nbuf = %s\n",
18          buffer.length(), buffer.toString() );
19    } // end main
20 } // end class StringBuilderCapLen
```

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

New capacity = 75

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

Method `length` returns `StringBuilder` length

Method `capacity` returns `StringBuilder` capacity

Use method `ensureCapacity` to set capacity to 75

Use method `setLength` to set length to 10

Only 10 characters from `StringBuilder` are printed

`StringBuilderCapLen.java`

`Line 11`
`Line 11`
`Line 13`
`Line 16`
`Program output`

Performance Tip

Dynamically increasing the capacity of a StringBuilder can take a relatively long time. Executing a large number of these operations can degrade the performance of an application. If a StringBuilder is going to increase greatly in size, possibly multiple times, setting its capacity high at the beginning will increase performance.

8.4.3 StringBuilder Methods charAt, setCharAt, getChars and reverse

Manipulating StringBuilder characters

- Method charAt
  - Return StringBuilder character at specified index
- Method setCharAt
  - Set StringBuilder character at specified index
- Method getChars
  - Return character array from StringBuilder
- Method reverse
  - Reverse StringBuilder contents

Outline

StringBuilderChars.java

(1 of 2)

```
1  // Fig. 8.12: StringBuilderChars.java
2  // StringBuilder methods charAt, setCharAt, getChars and reverse.
3
4  public class StringBuilderChars
5  {
6     public static void main( String args[] )
7     {
8        StringBuilder buffer = new StringBuilder( "hello there" );
9
10       System.out.printf( "buffer = %s\n", buffer.toString() );
11       System.out.printf( "Character at 0: %s\nCharacter at 4: %s\n\n",
12          buffer.charAt( 0 ), buffer.charAt( 4 ) );
13
14       char charArray[] = new char[ buffer.length() ];
15       buffer.getChars( 0, buffer.length(), charArray, 0
16       System.out.print( "The characters are: " );
17
18       for ( char character : charArray )
19          System.out.print( character );
20
```

Line 12

Line 15

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

Return character array from `StringBuilder`

```
21       buffer.setCharAt( 0, 'H' );
22       buffer.setCharAt( 6, 'T' );
23       System.out.printf( "\n\nbuf = %s", buffer.toString() );
24
25       buffer.reverse();
26       System.out.printf( "\n\nbuf = %s\n", buffer.toString() );
27    } // end main
28 } // 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
```

Replace characters at indices 0 and 6 with 'H' and 'T,' respectively

Reverse characters in **StringBuilder**

Outline

StringBuilderChars.java

(2 of 2)

Lines 21 and 22

Line 25

Program output

Common Programming Error

Attempting to access a character that is outside the bounds of a StringBuilder (i.e., with an index less than 0 or greater than or equal to the StringBuilder's length) results in a StringIndexOutOfBoundsException.

8.4.4 `StringBuilder` append Methods

Method `append`

- **Allow data values to be added to `StringBuilder`**

Outline

StringBuilderAppend.java

(1 of 3)

```
1  // Fig. 8.13: StringBuilderAppend.java
2  // StringBuilder append methods.
3
4  public class StringBuilderAppend
5  {
6     public static void main( String args[] )
7     {
8        Object objectRef = "hello";
9        String string = "goodbye";
10       char charArray[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
11       boolean booleanValue = true;
12       char characterValue = 'Z';
13       int integerValue = 7;
14       long longValue = 10000000000L;
15       float floatValue = 2.5f; // f suffix indicates 2.5 is a float
16       double doubleValue = 33.333;
17
18       StringBuilder lastBuffer = new StringBuilder( "last StringBuilder" );
19       StringBuilder buffer = new StringBuilder();
20
```

```
21    buffer.append( objectRef );
22    buffer.append( "\n" ); // each of these contains new line
23    buffer.append( string );
24    buffer.append( "\n" );
25    buffer.append( charArray );
26    buffer.append( "\n" );
27    buffer.append( charArray, 0, 3 );
28    buffer.append( "\n" );
29    buffer.append( booleanValue );
30    buffer.append( "\n" );
31    buffer.append( characterValue );
32    buffer.append( "\n" );
33    buffer.append( integerValue );
34    buffer.append( "\n" );
35    buffer.append( longValue );
36    buffer.append( "\n" );
37    buffer.append( floatValue );
38    buffer.append( "\n" );
39    buffer.append( doubleValue );
40    buffer.append( "\n" );
41    buffer.append( lastBuffer );
42
```

Append `String` "`hello`" to `StringBuilder`

Append a new line character to `StringBuilder`

Append `String` "`goodbye`"

Append "`a b c d e f`"

Append "`a b c`"

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

Append `StringBuilder`

Outline

`StringBuilderAppend.java`
(2 of 3)
Line 21
Line 23
Line 25
Line 27
Lines 29, 31, 33, 35, 37 and 39
Line 41

```
43       System.out.printf( "buffer contains %s\n", buffer.toString() );
44    } // end main
45 } // end StringBuilderAppend
```

```
buffer contains
hello
goodbye
abcdef
abc
true
Z
7
10000000000
2.5
33.333
last StringBuilder
```

Outline

StringBuilderAppend.java

(3 of 3)

Program output

8.4.5 `StringBuilder` Insertion and Deletion Methods

Method `insert`

- Allow data-type values to be inserted into `StringBuilder`

Methods `delete` and `deleteCharAt`

- Allow characters to be removed from `StringBuilder`

Outline

StringBuilderInsert.java

(1 of 3)

```
1  // Fig. 8.14: StringBuilderInsert.java
2  // StringBuilder methods insert, delete and deleteCharAt.
3
4  public class StringBuilderInsert
5  {
6     public static void main( String args[] )
7     {
8        Object objectRef = "hello";
9        String string = "goodbye";
10       char charArray[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
11       boolean booleanValue = true;
12       char characterValue = 'K';
13       int integerValue = 7;
14       long longValue = 10000000;
15       float floatValue = 2.5f; // f suffix indicates that 2.5 is a float
16       double doubleValue = 33.333;
17
18       StringBuilder buffer = new StringBuilder();
19
```

Outline

```
20      buffer.insert( 0, objectRef );
21      buffer.insert( 0, "  " ); // each of these contains two spaces
22      buffer.insert( 0, string );
23      buffer.insert( 0, "  " );
24      buffer.insert( 0, charArray );
25      buffer.insert( 0, "  " );
26      buffer.insert( 0, charArray, 3, 3 );
27      buffer.insert( 0, "  " );
28      buffer.insert( 0, booleanValue );
29      buffer.insert( 0, "  " );
30      buffer.insert( 0, characterValue );
31      buffer.insert( 0, "  " );
32      buffer.insert( 0, integerValue );
33      buffer.insert( 0, "  " );
34      buffer.insert( 0, longValue );
35      buffer.insert( 0, "  " );
36      buffer.insert( 0, floatValue );
37      buffer.insert( 0, "  " );
38      buffer.insert( 0, doubleValue );
39
```

Use method `insert` to insert data in beginning of `StringBuilder`

`StringBuilderInsert.java`

(2 of 3)

Lines 20-38

```
40       System.out.printf(
41          "buffer after inserts:\n%s\n\n", buffer.toString() );
42
43       buffer.deleteCharAt( 10 ); // delete 5 in 2.5
44       buffer.delete( 2, 6 ); // delete .333 in 33.333
45
46       System.out.printf(
47          "buffer after deletes:\n%s\n", buffer.toString() );
48    } // end main
49 } // end class StringBuilderInsert
```

```
buffer after inserts:
33.333  2.5  10000000  7  K  true  def  abcdef  goodbye  hello

buffer after deletes:
33  2.  10000000  7  K  true  def  abcdef  goodbye  hello
```

Use method `deleteCharAt` to remove character from index 10 in `StringBuilder`

Remove characters from indices 2 through 5 (inclusive)

8.5 Class Character

Treat primitive variables as objects

- **Type wrapper classes**
  - `Boolean`
  - `Character`
  - `Double`
  - `Float`
  - `Byte`
  - `Short`
  - `Integer`
  - `Long`
- **We examine class `Character`**

Outline

StaticCharMethods.java

(1 of 3)

```
1  // Fig. 8.15: StaticCharMethods.java
2  // Static Character testing methods and case conversion methods.
3  import java.util.Scanner;
4
5  public class StaticCharMethods
6  {
7     public static void main( String args[] )
8     {
9        Scanner scanner = new Scanner( System.in ); // create scanner
10       System.out.println( "Enter a character and press Enter" );
11       String input = scanner.next();
12       char c = input.charAt( 0 ); // get input character
13
```

Obtain first character of the input string

12

```
14       // display character info
15       System.out.printf( "is defined: %b\n", Character.isDefined( c ) );
16       System.out.printf( "is digit: %b\n", Character.isDigit( c ) );
17       System.out.printf( "is first character in a Java identifier: %b\n",
18          Character.isJavaIdentifierStart( c ) );
19       System.out.printf( "is part of a Java identifier: %b\n",
20          Character.isJavaIdentifierPart( c ) );
21       System.out.printf( "is letter: %b\n", Character.isLetter( c ) );
22       System.out.printf(
23          "is letter or digit: %b\n", Character.isLetterOrDigit( c ) );
24       System.out.printf(
25          "is lower case: %b\n", Character.isLowerCase( c ) );
26       System.out.printf(
27          "is upper case: %b\n", Character.isUpperCase( c ) );
28       System.out.printf(
29          "to upper case: %s\n", Character.toUpperCase( c ) );
30       System.out.printf(
31          "to lower case: %s\n", Character.toLowerCase( c ) );
32    } // end main
33 } // end class StaticCharMethods
```

Determine whether `c` is defined Unicode character set

Determine whether `c` is defined Unicode digit

Determine whether `c` can be used as first character in identifier

Determine whether `c` can be used as identifier character

Determine whether `c` is a letter

Determine whether `c` is letter or digit

Determine whether `c` is lowercase and uppercase, respectively

Convert `c` to its uppercase and lowercase, respectively

StaticCharMethods.java

Outline

StaticCharMethods.java

(3 of 3)

Program output

```
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
```

```
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
```

```
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: $
```

Outline

StaticCharMethods2.java

(1 of 2)

Line 28

```
1  // Fig. 8.16: StaticCharMethods2.java
2  // Static Character conversion methods.
3  import java.util.Scanner;
4
5  public class StaticCharMethods2
6  {
7     // create StaticCharMethods2 object execute application
8     public static void main( String args[] )
9     {
10       Scanner scanner = new Scanner( System.in );
11
12       // get radix
13       System.out.println( "Please enter a radix:" );
14       int radix = scanner.nextInt();
15
16       // get user choice
17       System.out.printf( "Please choose one:\n1 -- %s\n2 -- %s\n",
18          "Convert digit to character", "Convert character to digit" );
19       int choice = scanner.nextInt();
20
21       // process request
22       switch ( choice )
23       {
24          case 1: // convert digit to character
25             System.out.println( "Enter a digit:" );
26             int digit = scanner.nextInt();
27             System.out.printf( "Convert digit to character: %s\n",
28                Character.forDigit( digit, radix ) );
29             break;
30
```

Use method `forDigit` to convert `int digit` to number-system character specified by `int radix`

```
31          case 2: // convert character to digit
32             System.out.println( "Enter a character:" );
33             char character = scanner.next().charAt( 0 );
34             System.out.printf( "Convert character to digit: %s\n",
35                Character.digit( character, radix ) );
36             break;
37       } // end switch
38    } // end main
39 } // end class StaticCharMethods2
```

Use method `digit` to convert `char c` to number-system integer specified by `int radix`

```
Please enter a radix:
16
Please choose one:
1 -- Convert digit to character
2 -- Convert character to digit
2
Enter a character:
A
Convert character to digit: 10
```

```
Please enter a radix:
16
Please choose one:
1 -- Convert digit to character
2 -- Convert character to digit
1
Enter a digit:
13
Convert digit to character: d
```

Outline

`StaticCharMethods2.java`

(2 of 2)

Line 35

Program output

Outline

OtherCharMethods.java

Program output

```
1  // Fig. 30.17: OtherCharMethods.java
2  // Non-static Character methods.
3
4  public class OtherCharMethods
5  {
6     public static void main( String args[] )
7     {
8        Character c1 = 'A';
9        Character c2 = 'a';
10
11       System.out.printf(
12          "c1 = %s\nc2 = %s\n\n", c1.charValue(), c2.toString() );
13
14       if ( c1.equals( c2 ) )
15          System.out.println( "c1 and c2 are equal\n" );
16       else
17          System.out.println( "c1 and c2 are not equal\n" );
18    } // end main
19 } // end class OtherCharMethods
```

```
c1 = A
c2 = a

c1 and c2 are not equal
```

Assign two character literals to two `Character` objects. Auto-boxing occurs.

Method `toString` returns a string representation of the `Character` object

Obtain the `char` value stored in `Character` object `c1`

Use method `equals` to determine whether `c1` has the same contents as `c2`

8.6 Class `StringTokenizer`

Tokenizer

- Partition `String` into individual substrings
- Use delimiter
  - Typically whitespace characters (space, tab, newline, etc)
- Java offers `java.util.StringTokenizer`

Outline

TokenTest.java

Program output

```
1  // Fig. 8.18: TokenTest.java
2  // StringTokenizer class.
3  import java.util.Scanner;
4  import java.util.StringTokenizer;
5
6  public class TokenTest
7  {
8     // execute application
9     public static void main( String args[] )
10    {
11       // get sentence
12       Scanner scanner = new Scanner( System.in );
13       System.out.println( "Enter a sentence and press Enter" );
14       String sentence = scanner.nextLine();
15
16       // process user sentence
17       StringTokenizer tokens = new StringTokenizer( sentence );
18       System.out.printf( "Number of elements: %d\nThe tokens are:\n",
19          tokens.countTokens() );
20
21       while ( tokens.hasMoreTokens() )
22          System.out.println( tokens.nextToken() );
23    } // end main
24 } // end class TokenTest
```

```
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
```

Use `StringTokenizer` to parse `String` using default delimiter " `\n\t\r`"

Count number of tokens

Display next token as long as tokens exist

8.7 Regular Expressions, Class `Pattern` and Class `Matcher`

Regular expression

- Sequence of characters and symbols
- Useful for validating input and ensuring data format
  - E.g., ZIP code
- Facilitate the construction of a compiler

Regular-expression operations in String

- Method `matches`
  - Matches the contents of a `String` to regular expression
  - Returns a `boolean` indicating whether the match succeeded

8.7 Regular Expressions, Class `Pattern` and Class `Matcher` (Cont.)

Predefine character classes

- **Escape sequence that represents a group of character**
- **Digit**
  - **Numeric character**
- **Word character**
  - **Any letter, digit, underscore**
- **Whitespace character**
  - **Space, tab, carriage return, newline, form feed**

Character	Matches	Character	Matches
`\d`	any digit	`\D`	any non-digit
`\w`	any word character	`\W`	any non-word character
`\s`	any whitespace	`\S`	any non-whitespace
`\p{P}`	matches any punctuation character as period, comma, hyphen, dash, parentheses, apostrophe, ellipsis, colon, semicolon		

Fig. 8.19 | Predefined character classes.

8.7 Regular Expressions, Class `Pattern` and Class `Matcher` (Cont.)

Other patterns

- Square brackets (`[]`)
  - Match characters that do not have a predefined character class
  - E.g., [aeiou] matches a single character that is a vowel
- Dash (-)
  - Ranges of characters
  - E.g., [A-Z] matches a single uppercase letter
- ^
  - Not include the indicated characters
  - E.g., [^Z] matches any character other than Z

8.7 Regular Expressions, Class `Pattern` and Class `Matcher` (Cont.)

Quantifiers

- **Plus (+)**
  - **Match one or more occurrences**
  - **E.g., A+**
    - **Matches AAA but not empty string**
- **Asterisk (*)**
  - **Match zero or more occurrences**
  - **E.g., A***
    - **Matches both AAA and empty string**
- **Others in Fig. 8.22**

Quantifier	Matches
*	Matches zero or more occurrences of the pattern.
+	Matches one or more occurrences of the pattern.
?	Matches zero or one occurrences of the pattern.
{n}	Matches exactly n occurrences.
{n,}	Matches at least n occurrences.
{n,m}	Matches between n and m (inclusive) occurrences.

Fig. 8.22 | Quantifiers used in regular expressions.

Outline

ValidateInput.java

(1 of 2)

```
1  // Fig. 8.20: ValidateInput.java
2  // Validate user information using regular expressions.
3
4  public class ValidateInput
5  {
6     // validate first name
7     public static boolean validateFirstName( String firstName )
8     {
9        return firstName.matches( "[A-Z][a-zA-Z]*" );
10    } // end method validateFirstName
11
12    // validate last name
13    public static boolean validateLastName( String lastName )
14    {
15       return lastName.matches( "[a-zA-z]+([ '-][a-zA-Z]+)*" );
16    } // end method validateLastName
17
18    // validate address
19    public static boolean validateAddress( String address )
20    {
21       return address.matches(
22          "\\d+\\s+([a-zA-Z]+|[a-zA-Z]+\\s[a-zA-Z]+)" );
23    } // end method validateAddress
24
25    // validate city
26    public static boolean validateCity( String city )
27    {
28       return city.matches( "([a-zA-Z]+|[a-zA-Z]+\\s[a-zA-Z]+)" );
29    } // end method validateCity
30
```

Method `matches` returns `true` if the `String` matches the regular expression

Outline

ValidateInput.java

```
31    // validate state
32    public static boolean validateState( String state )
33    {
34       return state.matches( "([a-zA-Z]+|[a-zA-Z]+\\s[a-zA-Z]+)" ) ;
35    } // end method validateState
36
37    // validate zip
38    public static boolean validateZip( String zip )
39    {
40       return zip.matches( "\\d{5}" );
41    } // end method validateZip
42
43    // validate phone
44    public static boolean validatePhone( String phone )
45    {
46       return phone.matches( "[1-9]\\d{2}-[1-9]\\d{2}-\\d{4}" );
47    } // end method validatePhone
48 } // end class ValidateInput
```

Method `matches` returns `true` if the `String` matches the regular expression

Outline

Validate.java

(1 of 3)

```
1  // Fig. 8.21: Validate.java
2  // Validate user information using regular expressions.
3  import java.util.Scanner;
4
5  public class Validate
6  {
7     public static void main( String[] args )
8     {
9        // get user input
10       Scanner scanner = new Scanner( System.in );
11       System.out.println( "Please enter first name:" );
12       String firstName = scanner.nextLine();
13       System.out.println( "Please enter last name:" );
14       String lastName = scanner.nextLine();
15       System.out.println( "Please enter address:" );
16       String address = scanner.nextLine();
17       System.out.println( "Please enter city:" );
18       String city = scanner.nextLine();
19       System.out.println( "Please enter state:" );
20       String state = scanner.nextLine();
21       System.out.println( "Please enter zip:" );
22       String zip = scanner.nextLine();
23       System.out.println( "Please enter phone:" );
24       String phone = scanner.nextLine();
25
```

Outline

Validate.java

(2 of 3)

```
26       // validate user input and display error message
27       System.out.println( "\nValidate Result:" );
28
29       if ( !ValidateInput.validateFirstName( firstName ) )
30          System.out.println( "Invalid first name" );
31       else if ( !ValidateInput.validateLastName( lastName ) )
32          System.out.println( "Invalid last name" );
33       else if ( !ValidateInput.validateAddress( address ) )
34          System.out.println( "Invalid address" );
35       else if ( !ValidateInput.validateCity( city ) )
36          System.out.println( "Invalid city" );
37       else if ( !ValidateInput.validateState( state ) )
38          System.out.println( "Invalid state" );
39       else if ( !ValidateInput.validateZip( zip ) )
40          System.out.println( "Invalid zip code" );
41       else if ( !ValidateInput.validatePhone( phone ) )
42          System.out.println( "Invalid phone number" );
43       else
44          System.out.println( "Valid input.  Thank you." );
45    } // end main
46 } // end class Validate
```

Outline

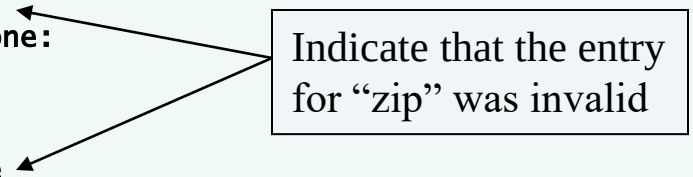
Validate.java

(3 of 3)

Program output

```
Please enter first name:
Jane
Please enter last name:
Doe
Please enter address:
123 Some Street
Please enter city:
Some City
Please enter state:
SS
Please enter zip:
123
Please enter phone:
123-456-7890

Validate Result:
Invalid zip code
```




```
Please enter first name:
Jane
Please enter last name:
Doe
Please enter address:
123 Some Street
Please enter city:
Some City
Please enter state:
SS
Please enter zip:
12345
Please enter phone:
123-456-7890

Validate Result:
Valid input.  Thank you.
```

8.7 Regular Expressions, Class `Pattern` and Class `Matcher` (Cont.)

Replacing substrings and splitting strings

- String method `replaceAll`
  - Replace text in a string with new text
- String method `replaceFirst`
  - Replace the first occurrence of a pattern match
- String method `split`
  - Divides string into several substrings

Outline

```
1  // Fig. 8.23: RegexSubstitution.java
2  // Using methods replaceFirst, replaceAll and split.
3
4  public class RegexSubstitution
5  {
6     public static void main( String args[] )
7     {
8        String firstString = "This sentence ends in 5 stars ?!,.*****";
9        String secondString = "1, 2, 3, 4, 5, 6, 7, 8";
10
11       System.out.printf( "Original String 1: %s\n", firstString );
14       // replace all puntuation marks with empty string
15       firstString = firstString. replaceAll("(?!')\\p{P}", "")
14       // replace '*' with '^'
15       firstString = firstString.replaceAll( "\\*", "^" );
16       System.out.printf( "^ substituted for *: %s\n", firstString );
17
18       // replace 'stars' with 'carets'
19       firstString = firstString.replaceAll( "stars", "carets" );
20
21       System.out.printf(
22          "\"carets\" substituted for \"stars\": %s\n", firstString );
23
24       // replace words with 'word'
25       System.out.printf( "Every word replaced by \"word\": %s\n\n",
26          firstString.replaceAll( "\\w+", "word" ) );
27
28       System.out.printf( "Original String 2: %s\n", secondString );
29
```

Replace all punctuation marks like ?!,. * :( ) in `firstString` with empty string

Replace every instance of "*" in `firstString` with "^"

Replace every instance of "`stars`" in `firstString` with "`carets`"

Replace every word in `firstString` with "`word`"

`RegexSubstitution.java`

Outline

```
30      // replace first three digits with 'digit'
31      for ( int i = 0; i < 3; i++ )
32         secondString = secondString.replaceFirst( "\\d", "digit" );
33
34      System.out.printf(
35         "First 3 digits replaced by \"digit\" : %s\n", s
36      String output = "String split at commas: [";
37
38      String[] results = secondString.split( ",\\s*" ); // split on commas
39
40      for ( String string : results )
41         output += "\"" + string + "\", "; // output results
42
43      // remove the extra comma and add a bracket
44      output = output.substring( 0, output.length() - 2 ) + "]";
45      System.out.println( output );
46   } // end main
47 } // end class RegexSubstitution
```

`replaceFirst` replaces a single occurrence of the regular expression

split returns array of substrings between matches of the regular expression

```
Original String 1: This sentence ends in 5 stars *****
^ substituted for *: This sentence ends in 5 stars ^^^^^
"carets" substituted for "stars": This sentence ends in 5 carets ^^^^^
Every word replaced by "word": word word word word word word ^^^^^

Original String 2: 1, 2, 3, 4, 5, 6, 7, 8
First 3 digits replaced by "digit" : digit, digit, digit, 4, 5, 6, 7, 8
String split at commas: ["digit", "digit", "digit", "4", "5", "6", "7", "8"]
```

RegexSubstitution.java

(2 of 2)

Line 32

Line 38

8.7 Regular Expressions, Class `Pattern` and Class `Matcher` (Cont.)

Class `Pattern`

- Represents a regular expression

Class `Match`

- Contains a regular-expression pattern and a `CharSequence`
- Interface `CharSequence`
  - Allows read access to a sequence of characters
  - `String` and `StringBuilder` implement `CharSequence`

Common Programming Error

A regular expression can be tested against an object of any class that implements interface `CharSequence`, but the regular expression must be a String. Attempting to create a regular expression as a StringBuilder is an error.

Outline

RegexMatches.java

Program output

```
1  // Fig. 8.24: RegexMatches.java
2  // Demonstrating Classes Pattern and Matcher.
3  import java.util.regex.Matcher;
4  import java.util.regex.Pattern;
5
6  public class RegexMatches
7  {
8     public static void main( String args[] )
9     {
10       // create regular expression
11       Pattern expression =
12          Pattern.compile( "J.*\\d[0-35-9]-\\d\\d-\\d\\d" );
13
14       String string1 = "Jane's Birthday is 05-12-75\n" +
15          "Dave's Birthday is 11-04-68\n" +
16          "John's Birthday is 04-28-73\n" +
17          "Joe's Birthday is 12-17-77";
18
19       // match regular expression to string and print matches
20       Matcher matcher = expression.matcher( string1 );
21
22       while ( matcher.find() )
23          System.out.println( matcher.group() );
24    } // end main
25 } // end class RegexMatches
```

```
Jane's Birthday is 05-12-75
Joe's Birthday is 12-17-77
```

`compile` creates a `Pattern` object for regular expression

`matcher` creates the `Matcher` object for the compiled regular expression and the matching sequence

`find` gets the first substring that matches the regular expression

`group` returns the string from the search object that matches the search pattern

Common Programming Error

Method `matches` (from class `String`, `Pattern` or `Matcher`) will return `true` only if the entire search object matches the regular expression. Methods `find` and `lookingAt` (from class `Matcher`) will return `true` if a portion of the search object matches the regular expression.

```
1  /* JavaFX FXML Application
2   * ValidateController.java
3  */
4  import javafx.fxml.FXML;
5  import javafx.scene.control.Alert;
6  import javafx.scene.control.Button;
7  import javafx.scene.control.TextField;
8
9  public class ValidateController {
10     private Alert mBox; // MessageBox
11     @FXML
12     private TextField txtCity;
13     @FXML
14     private TextField txtZip;
15     @FXML
16     private TextField txtState;
17     @FXML
18     private TextField txtPhone;
19     @FXML
20     private TextField txtFirstName;
21     @FXML
22     private TextField txtlLastName;
23     @FXML
24     private TextField txtAddress;
25     @FXML
26     void btnValidateOnAction(ActionEvent event) {
27         validate();
28     }
```

Резюме

JavaFX приложение за проверка на валидност на въведените данни като ще използваме метод `matches()` на `String` обектите

1. Дефинираме компонентите на графичния интерфейс

```
29      // setup the MessageBox properties
30      private void showMessage(String contentText) {
31          mBox.setContentText(contentText);
32          mBox.setHeaderText(null);
33          mBox.showAndWait();
34      }
35      // handles btnValidate action event
36      private void validate() {
37          // ensure that no textboxes are empty
38          if (txtlLastName.getText().equals( ")
39                  || txtFirstName.getText().equals( ")
40                  || txtAddress.getText().equals( ")
41                  || txtCity.getText().equals( ")
42                  || txtState.getText().equals( ")
43                  || txtZip.getText().equals( ")
44                  || txtPhone.getText().equals( )) // end condition
45          {
46              showMesaage("Please fill all fields");
47          } // if first name format invalid show message
48          else if (!txtFirstName.getText().matches("[A-Z][a-zA-Z]*")) {
49              showMessage("Invalid first name");
50          } // if last name format invalid show message
51          else if (!txtlLastName.getText().matches("[A-Z][a-zA-Z]*")) {
52              showMessage("Invalid last name");
53          } // if address format invalid show message
```

showMessage() инициализира свойствата на **Alert** обекта

matches() връща **true** ако **String** обекта съвпада с регулярния израз

2. Дефинираме метода за обработка на събитието **Action** за бутона **btnValidate**, който ще се активира при натискането му

Резюме

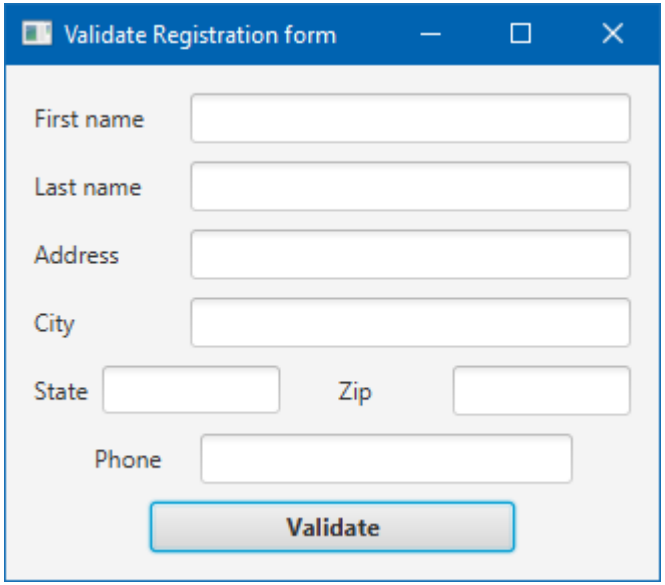
```
54          else if (!txtAddress.getText().matches(
55                  "\\d+\\s+([a-zA-Z]+|[a-zA-Z]+\\s[a-zA-Z]+)")) {
56              showMessage("Invalid address");
57          } // if city format invalid show message
58          else if (!txtCity.getText().matches(
59                  "([a-zA-Z]+|[a-zA-Z]+\\s[a-zA-Z]+)")) {
60              showMessage("Invalid city");
61          } // if state format invalid show message
62          else if (!txtState.getText().matches(
63                  "([a-zA-Z]+|[a-zA-Z]+\\s[a-zA-Z]+)")) {
64              showMessage("Invalid state");
65          } // if zip code format invalid show message
66          else if (!txtZip.getText().matches("\\d{5}")) {
67              showMessage("Invalid zip code");
68          } // if phone number format invalid show message
69          else if (!txtPhone.getText().matches(
70                  "[1-9]\\d{2}-[1-9]\\d{2}-\\d{4}")) {
71              showMessage("Invalid phone number");
72          } else // information is valid, signal user
73          {
74              showMessage("Thank you");
75          }
76      } // end method validateDate
77
```

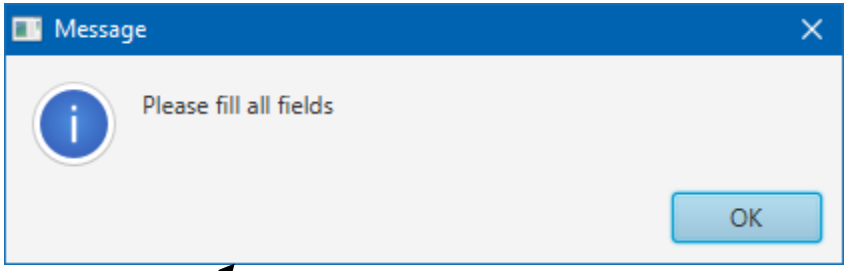
matches() връща **true** ако **String** обекта съвпада с регулярния израз

```
78      @FXML
79      void initialize() {
80          // create an instance of Alert dialog MessageBox
81          mBox = new Alert(Alert.AlertType.INFORMATION);
82      }
83  }
```

Резюме

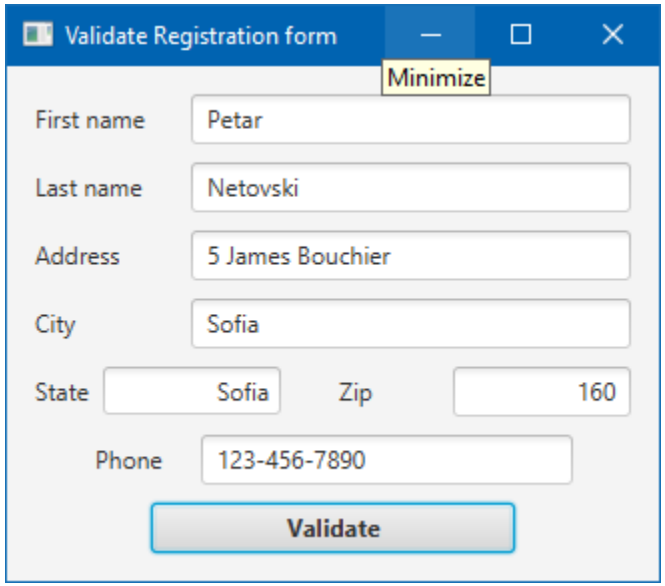
Метод initialize() създава обект от диалогов прозорец

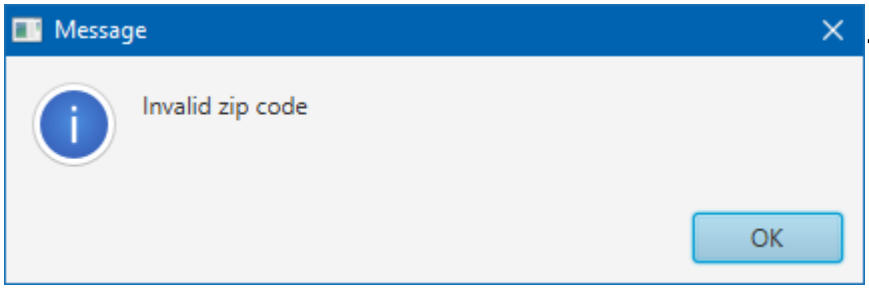




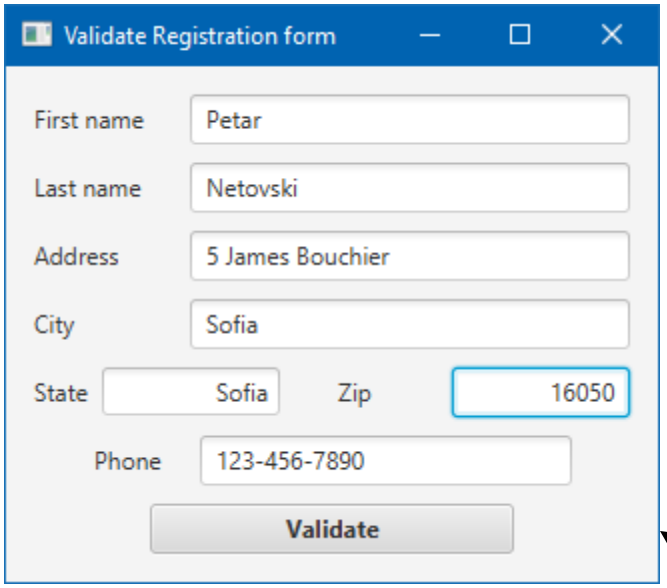
Съобщение за грешка- има **текстови полета, които са оставени празни**

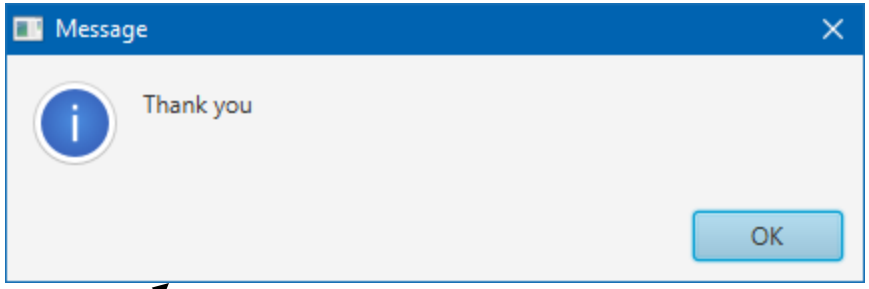
Резюме





Грешни данни за “**Zip**” **TextBox** очаква **точно 5 цифри**





Данните за всички **TextBox** полета са въведени правилно

Задачи

Задача 1.

Редактирайте JavaFX FXML приложението, дадено на предходните слайдове като добавите етикет и текстово поле във формата за въвеждане на `IP` адрес и дефинирайте подходящ регулярен израз за проверка на неговата валидност

Задача 2.

Напишете JavaFX FXML приложение, което да преобразува от стойности от *Fahrenheit* temperature в *Celsius* еквивалетни стойност като използва графичен потребителски за вход и изход на данните . Чрез регулярни изрази да се допуска въвеждане единствено на числа в плаваща запетая, а също и се проверява за липсващи данни

Преобразуването от *Fahrenheit* в *Celsius* става по формулата

$$Celsius = 5.0 / 9.0 * (Fahrenheit - 32)$$

Задачи

Задача 3.

Да се напише JavaFX FXML приложение, при което в текстов прозорец се въвежда серия от цели числа, разделени със запетая, следвана от един или повече празни символи. При натискане на бутон въведените числа да се запишат в масив, който да се сортира по метода на мехурчето и резултат от сортирането да се изведе подходящо форматиран в графичния прозорец.

Да се използват регулярни изрази за проверка на формата на въвежданите данни- липсващи разделители при въвеждане на входните данни или въвеждане на данни различни от цели числа като се блокира изпълнението с извеждане на текст със съобщение за грешка.

Използвайте метод `split()` на `String` обект.

Задача 4.

Задача 3 да се реши като се използва `StringTokenizer` .