

- ◆ In this session, you will learn to:
 - ◆ Parse strings using the split() method
 - ◆ Parse strings using the StringTokenizer class
 - ◆ Tokenize using the Scanner class
 - ◆ Work with regular expressions
 - ◆ Replace strings using the replaceAll() method

Java

Programming in Java

Using the split() Method

- ◆ `split()` method:
 - ◆ Used to parse a string
 - ◆ Breaks down the string apart when applied with character/s
- ◆ The following embedded Word document shows the utilization of the `split()` method.



split method

- ◆ `StringTokenizer` class:
 - ◆ Used to parse a string
 - ◆ Allows accessing tokens by iteration
- ◆ The embedded Word document demonstrates the utilization of the `StringTokenizer` class.



`StringTokenizer`
class

- ◆ **Scanner class:**
 - ◆ Used to tokenize an input stream or a string
 - ◆ Used to tokenize numbers and convert them into primitive type
- ◆ The embedded Word document shows how to tokenize a string or a stream using the `Scanner` class.



Scanner class

- ◆ Regular expressions:
 - ◆ Used to match strings of text
 - ◆ Provide detailed vocabulary
 - ◆ Used to search and extract or replace strings
- ◆ Java objects used in regular expression:
 - ◆ `Pattern`
 - ◆ `Matcher`
 - ◆ `PatternSyntaxException`
 - ◆ `java.util.regex`

- ◆ The `Pattern` object defines a regular expression.
- ◆ The `Matcher` object specifies the target string to search.
- ◆ The `Pattern` and `Matcher` objects work together.
- ◆ The following embedded Word document shows the search operation using the `Pattern` and `Matcher` classes.



Pattern and
Matcher classes

- ◆ The following table shows the character class patterns to be used in regular expressions.

| <i>Character</i> | <i>Description</i> |
|---------------------|---|
| <code>.</code> | <i>Matches any single character (letter, digit, or special character), except end-of-line markers</i> |
| <code>[abc]</code> | <i>Would match any “a”, “b”, or “c” in that position</i> |
| <code>[^abc]</code> | <i>Would match any character that is not “a”, “b”, or “c” in that position</i> |
| <code>[a-c]</code> | <i>A range of characters (in this case, “a”, “b”, and “c”)</i> |
| <code> </code> | <i>Alteration; essentially an “or” indicator</i> |

Programming in Java

Character Class Patterns (Contd.)

- ◆ Target string: *It was the best of times*
- ◆ The following table shows the character class patterns applied on the preceding target string.

| Pattern | Description | Text Matched |
|------------|--|---------------------------------|
| w.s | Any sequence that starts with a "w" followed by any character followed by "s". | It was the best of times |
| w[abc]s | Any sequence that starts with a "w" followed by "a", "b", or "c" and then "s". | It was the best of times |
| t[^eao]mes | Any sequence that starts with a "t" followed by any character that is not "a", "e" or "o" followed by "mes". | It was the best of times |



Let us see how to apply the character class pattern on a string.

Programming in Java

Predefined Character Classes

- ◆ Character classes are used repeatedly.
- ◆ These classes are turned into predefined character classes.
- ◆ The following table shows the predefined characters and the character classes.

| <i>Predefined Character</i> | <i>Character Class</i> | <i>Negated Character</i> | <i>Negated Class</i> |
|--|----------------------------|--------------------------|-----------------------------|
| <code>\d</code> (<i>digit</i>) | <code>[0-9]</code> | <code>\D</code> | <code>[^0-9]</code> |
| <code>\w</code> (<i>word char</i>) | <code>[a-zA-Z0-9_]</code> | <code>\W</code> | <code>[^a-zA-Z0-9_]</code> |
| <code>\s</code> (<i>white space</i>) | <code>[\r\t\n\f0XB]</code> | <code>\S</code> | <code>[^\r\t\n\f0XB]</code> |

Programming in Java

Predefined Character Classes (Contd.)

- ◆ Target string: *Jo told me 20 ways to San Jose in 15 minutes.*
- ◆ The following table shows the predefined character class patterns on the preceding target string.

| Pattern | Description | Text Matched |
|-----------------------|--|--|
| <code>\\d\\d</code> | Find any two digits. ** | Jo told me 20 ways to San Jose in 15 minutes. |
| <code>\\sin\\s</code> | Find “in” surrounded by two spaces and then the next three characters. | Jo told me 20 ways to San Jose in 15 minutes. |
| <code>\\Sin\\S</code> | Find “in” surrounded by two non space characters and then the next three characters. | Jo told me 20 ways to San Jose in 15 minutes. |



Let us see how to apply the predefined character class patterns on a string.

- ◆ Quantifiers select a range of characters in a string.
- ◆ The following table shows the list of quantifiers that can be used in regular expressions.

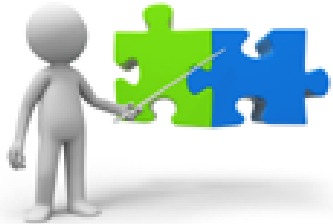
| Quantifier | Description |
|------------|---|
| * | <i>The previous character is repeated zero or more times.</i> |
| + | <i>The previous character is repeated one or more times.</i> |
| ? | <i>The previous character must appear once or not at all.</i> |
| {n} | <i>The previous character appears exactly n times.</i> |
| {m, n} | <i>The previous character appears from m to n times.</i> |
| {m, } | <i>The previous character appears m or more times.</i> |
| (xx) {n} | <i>This group of characters repeats n times.</i> |

Programming in Java

Quantifier: Examples

- ◆ Target string: *Longlonglong ago, in a galaxy far far away*
- ◆ The following table shows implementation of quantifier patterns on the preceding target string.

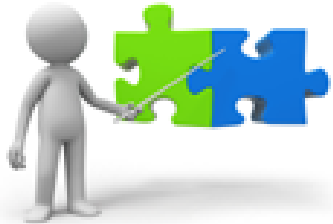
| <i>Pattern</i> | <i>Description</i> | <i>Text Matched</i> |
|------------------|---|--|
| <i>ago.*</i> | <i>Find “ago” and then 0 or all the characters remaining on the line.</i> | <i>Longlonglong ago, in a galaxy far far away</i> |
| <i>gal.{3}</i> | <i>Match “gal” plus the next three characters. This replaces “...” as used in a previous example.</i> | <i>Longlonglong ago, in a galaxy far far away</i> |
| <i>(long){2}</i> | <i>Find “long” repeated twice.</i> | <i>Longlonglong ago, in a galaxy far far away</i> |



Let us see how to apply the quantifier patterns on a string.

- ◆ Greediness principle:
 - ◆ Regular expression grabs as many characters as possible
- ◆ ? operator:
 - ◆ Limits the search to the shortest possible match
- ◆ Target string: *Longlonglong ago, in a galaxy far far away.*
- ◆ The following table shows greediness and the usage of ? operator on the preceding target string.

| <i>Pattern</i> | <i>Description</i> | <i>Text Matched</i> |
|------------------|--|--|
| <i>ago.*far</i> | <i>A regular expression always grabs the most characters possible.</i> | <i>Longlonglong ago, in a galaxy far far away.</i> |
| <i>ago.*?far</i> | <i>The “?” character essentially turns off greediness.</i> | <i>Longlonglong ago, in a galaxy far far away.</i> |



Let us see how the ? operator works on a string search and removes greediness.

- ◆ Boundary characters match different parts of a line using the regular expressions.
- ◆ The following table shows the boundary characters to be matched in a string.

| <i>Anchor</i> | <i>Description</i> |
|-----------------|--|
| <code>^</code> | <i>Matches the beginning of a line</i> |
| <code>\$</code> | <i>Matches the end of a line</i> |
| <code>\b</code> | <i>Matches the start or the end of a word</i> |
| <code>\B</code> | <i>Does not match the start or the end of a word</i> |

Programming in Java

Boundary: Examples

- ◆ Target string: *it was the best of times or it was the worst of times*
- ◆ The following table shows the usage of boundary characters on the preceding target string.

| Pattern | Description | Text Matched |
|-----------------------------|--|---|
| <code>^it.*?times</code> | The sequence starts a line with “it” followed by some characters and ends a line with “times”, and the greediness is off | <i>it was the best of times or it was the worst of times</i> |
| <code>\\sit.*times\$</code> | The sequence that starts with “it” followed by some characters and ends the line with “times” | <i>it was the best of times or it was the worst of times</i> |
| <code>\\bor\\b.{3}</code> | Find “or” surrounded by word boundaries, plus the next three characters | <i>it was the best of times or it was the worst of times</i> |

Programming in Java

Matching and Groups

- ◆ Parentheses are used with regular expressions to identify parts of a string to match.
- ◆ Target string: *george.washington@example.com*
- ◆ The following table shows the use of matching and grouping on the preceding target string.

| <i>String</i> | <i>Pattern</i> |
|----------------------|--|
| <i>Match 3 Parts</i> | <i>(george).(washington)@(example.com)</i> |
| <i>Group Numbers</i> | <i>(1).(2)@(3)</i> |
| <i>Pattern</i> | <i>(\lS+?)\l.(\lS+?)@(\lS+)</i> |



Let us see how to create group pattern using parentheses in a string.

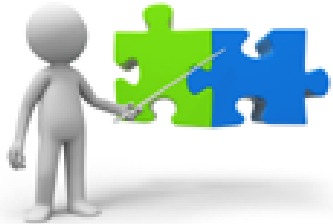
Programming in Java

Using the replaceAll Method

- ◆ Search and replace can be performed by the `replaceAll()` method.
- ◆ The following embedded Word document shows the usage of the `replaceAll()` method.



replaceAll
method



Let us see how to apply the `replaceAll()` method in a string.

Get Ready for the Challenge



- ◆ Which of the following statements is correct regarding regular expressions?
 - ◆ The `Matcher` object defines a regular expression.
 - ◆ Regular expressions is a language for matching strings of text.
 - ◆ A `Scanner` class can be used to tokenize an input stream or a string.
 - ◆ The `Pattern` object specifies the target string to be searched.
 - ◆ Character classes cannot be used repeatedly.

- ◆ Solution:
 - ◆ Regular expressions is a language for matching strings of text.

◆ Fill in the blank:

- ◆ The ____ operator limits the search to the shortest possible match within a string.

◆ Solution:

◆ ?

- ◆ In this session, you learned that:
 - ◆ The `split()` method and the `StringTokenizer` class are used to parse a string.
 - ◆ The `Scanner` class can tokenize strings, streams, and numbers.
 - ◆ Regular expressions is a language for matching strings of text.
 - ◆ The `Pattern` object defines a regular expression.
 - ◆ The `Matcher` object specifies target string to search.
 - ◆ Quantifiers enable to select a range of characters in a string.
 - ◆ The `?` operator limits the search to the shortest possible match.
 - ◆ Search and replace can be performed by the `replaceAll()` method.