- 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

- `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.

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

- 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*** |

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

| *Predefined Character* | *Character Class* | *Negated Character* | *Negated Class* |
|---|---|---|---|
| *\d (digit)* | *[0-9]* | *\D* | *[^0-9]* |
| *\w (word char)* | *[a-zA-Z0-9_]* | *\W* | *[^a-zA-Z0-9_]* |
| *\s (white space)* | *[ \r\t\n\f\0XB]* | *\S* | *[^ \r\t\n\f\0XB]* |

- 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 |
|---|---|---|
| \\d\\d | *Find any two digits. ** | *Jo told me **20** ways to San Jose in 15 minutes.* |
| \\sin\\s | *Find "in" surrounded by two spaces and then the next three characters.* | *Jo told me 20 ways to San Jose **in** 15 minutes.* |
| \\Sin\\S | *Find "in" surrounded by two non space characters and then the next three characters.* | *Jo told me 20 ways to San Jose in 15 **minu**tes.* |

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

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

- 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.

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

- 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.

| Anchor | Description |
|---|---|
| ^ | *Matches the beginning of a line* |
| $ | *Matches the end of a line* |
| \b | *Matches the start or the end of a word* |
| \B | *Does not match the start or the end of a word* |

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

- 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.

| String | Pattern |
|---|---|
| *Match 3 Parts* | *(george).(washington)@(example.com)* |
| *Group Numbers* | *(1).(2)@(3)* |
| *Pattern* | *(\\S+?)\\.(\\S+?)@(\\S+)* |

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

replaceAll
method