Strings and Regular Expressions

Object Oriented Programming
2024 First Semester
Shin-chi Tadaki (Saga University)

1. `String` class
2. Creation and concatenation
3. Regular expressions

Today's sample programs

- `https://github.com/oop-mc-saga/StringsAndRegularExpressions`

String class

- The `String` class stores a sequence of characters
- Major methods
  - `concat()`: concatenates the specified string at the end and returns a new string
  - `indexOf()`: returns the index of the specified sub-string
  - `replace()`: replaces the sub-strings into the new one and returns a new string
  - `substring()`: returns the sub-string by specifying indexes

Notes on `String` class

- `String` objects are *immutable* (it can not be changed).
- Comparing two `String` objects
  - `==`: equals as an objects
  - `equals()`: stores the same string as data

Example 1.1: Comparing strings

```
1  public static void main(String[] args) {
2      String a = "abc";
3      String b = a;
4      String c = "ab";
5      String d = c;
6      c = c + "c";
7      if (a == b) {
8          System.out.println("a is the same object of b");
9      }
10     if (a == c) {
11         System.out.println("a is the same object of c");
12     }
13     if (c != d) {
14         System.out.println("d is not the same object of c");
15     }
16     if (a.equals(b)) {
17         System.out.println("a stores the same string of b");
18     }
19     if (a.equals(c)) {
20         System.out.println("a stores the same string of c");
21     }
22 }
```

example/StringTest.java

Searching in `String` class

- `char charAt(int index)`
  - returns a character at the specified `index`
- `int indexOf(String str)`
  - returns the index where the specified `str` first appears
- `int indexOf(String str, int from)`
  - returns the index where the specified `str` first appears after the position `from`
- `String substring(int begin, int end)`
  - returns the substring by specifying the begin and end indexes

`toString()` method

- `Object.toString()`
  - Converts the instance into a string.
- You can define how to convert the object into a string by overriding the `toString()` method.

StringBuilder class

- The `StringBuilder` class supports to concatenate objects into a string
- `append(Object o)`: appends the object at the end of the string using `o.toString()`
- `delete(int start, int end)`: deletes the substring by specifying the range
- `insert(int offset, Object o)`: inserts the object `o` at the position `offset`
- `toString()`: converts the content of `StringBuilder` instance into a `String` instance

Example 2.1: StringBuilder

```java
public static <T> String list2String(List<T> list) {
    StringBuilder sb = new StringBuilder();
    sb.append("[");
    list.stream().forEachOrdered(
            p -> sb.append(p).append(",")
    );
    int k = sb.lastIndexOf(",");
    sb.deleteCharAt(k).append("]");
    return sb.toString();
}
```

example/BuilderExample.java

StringJoiner class

- Joining objects with a separator
- You can also specify the prefix and suffix.

```
1  public static <T> String list2String2(List<T> list) {
2      StringJoiner sj = new StringJoiner(",", "(", ")");
3      list.stream().forEachOrdered(t -> sj.add(t.toString()));
4      return sj.toString();
5  }
```

example/BuilderExample.java

Regular expressions

- Regular expressions are string patterns with repetitions of characters or strings
- special characters of positions
  - `^`: the beginning of the string
    - `^Java`: Strings starting `Java`
  - `$`: the end of the string
    - `Java$`: Strings end with `Java`

Parts of syntaxes for regular expressions

- `X?` : `X` appears 0 or 1 time
- `X+` : `X` repeats more than once
- `X*` : `X` repeats more than 0 times
- `X{n}`: `X` repeats n times
- `X{n,}`: `X` repeats more than n times
- `[abc]`: `a`, `b`, or `c`
- `\s`: whitespace characters (space, tab, etc)
- `\S` : non-whitespace characters
- `\d` : digit `[0-9]`
- `\D` : non-digit

Splitting strings using regular expressions

- Lines using various delimiters such as space, tab, comma, colon, etc.
- Regular expressions give a simple way to split strings
- `String ss[] = s.split("\\s|,|:");`
  - Splits the string `s` by space, comma, or colon
  - `\\s`: whitespace characters such as space, tab, etc
- `x|y`: `x` or `y`

Example 3.1: Splitting strings

```
1  public static void main(String[] args) {
2      String input[] = {
3          "a,b,c,d,e,f",
4          "a b c d e f",
5          "a\tb\tc\td\te\tf",
6          "a:b:c:d:e:f"
7      };
8      for (String s : input) {
9          String ss[] = s.split("\\s|,|:");
10         for (String e : ss) {
11             System.out.print(e + " ");
12         }
13         System.out.println();
14     }
15 }
```

regexExample/SplitExample.java

Find strings using regular expressions

- Define regular expression
  - `Pattern p = Pattern.compile(String regex);`
- Generate `matcher`
  - `Matcher m = p.matcher(input);`
- Search matched strings
  - `boolean m.find()`: find the next subsequence matched
  - `int m.start()`: the start index of the previous match
  - `String m.group()`: the input subsequence of the previous match.

Example 3.2: Finding patterns

```
1  public static void main(String[] args) {
2      String input = "00101110010011";
3      //Define regular expression
4      Pattern p = Pattern.compile("101+");
5      Matcher m = p.matcher(input);
6      int c = 0;//starting position of matching
7      while (m.find(c)) {//matching by starting position
8          c = m.start();//actual matched position
9          String s = m.group();
10         System.out.println("matches " + s + " at " + c);
11         c++;//next matching
12     }
13 }
```

regexExample/RegexExample.java

Numbering matched patterns

- Parentheses `()` are used to define group patterns
- For expressing characters `(` and `)`, use `\(` and `\)`
- Matched patterns `((A)(B(C)))` are numbered as
  1. `((A)(B(C)))`
  2. `(A)`
  3. `(B(C))`
  4. `(C)`

Example 3.3: Matching patterns

```java
public static void main(String[] args) {
    String dates[] = {"20100401", "20110530", "20101109",
        "19991010", "19890321", "Aug5,2019", "2010Sep9"};
    Pattern p = Pattern.compile("((\\d{4})(\\d\\d)(\\d\\d))");
    for (String d : dates) {
        Matcher m = p.matcher(d);
        while (m.find()) {
            int n = m.groupCount();//number of matched positions
            StringJoiner sj = new StringJoiner("/");
            String str = m.group(1);//the whole matched string
            for(int i=2;i<=n;i++){//append matched substrings
                sj.add(m.group(i));
            }
            System.out.println(str + " -> " + sj.toString());
        }
    }
}
```

regexExample/RegexExample2.java

Replacing strings using regular expressions

- Simple replacements
  - `m.replaceFirst()`
  - `m.replaceAll()`
- reusing matched string
  - matched strings are numbered as `$n`

Example 3.4: Replacing strings

```
1  public static void main(String[] args) {
2      String input = "001011101001101";
3      //Define regular expression
4      Pattern p = Pattern.compile("101+");
5      Matcher m = p.matcher(input);
6      //Simple replacement
7      System.out.println(m.replaceFirst("121"));
8      System.out.println(m.replaceAll("121"));
9      //Using matched string
10     System.out.println(m.replaceAll("_$0_"));
11     //Using matched position
12     p = Pattern.compile("(10)(1+)");
13     m = p.matcher(input);
14     System.out.println(m.replaceAll("12$2"));
15 }
```

regexExample/ReplaceExample.java

Exercise

Understand the role of variable `k` in `exercise/Simple.java`.