

Bài 19

Xử lý chuỗi

Module: ADVANCED PROGRAMMING WITH JAVA

- Trình bày được chuỗi trong Java
- Sử dụng được các phương thức xử lý chuỗi có sẵn trong Java
- Sử dụng được lớpStringBuilder
- Trình bày được Regular Expression và các ứng dụng của Regular Expression
- Trình bày được các Character Class trong Java
- Trình bày được các Quantifier trong Java
- Sử dụng Regular Expression để validate chuỗi, tìm kiếm trong chuỗi, crawl dữ liệu text

Chuỗi

Chuỗi trong Java

Các phương thức có sẵn của lớp String

- Chuỗi là một tập hợp các ký tự
- Hằng chuỗi bao gồm một hoặc nhiều ký tự nằm trong dấu nháy đơn "" hoặc kép ""
- Ví dụ:
 - "Welcome to Java"
 - "B"
 - "123"
- Cách hằng chuỗi được sử dụng đến gán cho các biến chuỗi

Khai báo chuỗi

- Sử dụng kiểu dữ liệu String để khai báo một chuỗi trong Java.
- Ví dụ:
 - Khai báo biến message có kiểu chuỗi, nhận vào giá trị Welcometo Java:
String message = "Welcome to Java";
 - String là một kiểu dữ liệu tham chiếu. Trong ví dụ trên, message là một biến tham chiếu, tham chiếu tới một đối tượng chuỗi có giá trị là Welcome to Java.
- String là một lớp được định nghĩa sẵn trong java, thuộc gói java.lang.

Các phương thức của lớp String

- Chuỗi là một tập hợp các ký tự

<i>Method</i>	<i>Description</i>
<code>length()</code>	Returns the number of characters in this string.
<code>charAt(index)</code>	Returns the character at the specified index from this string.
<code>concat(s1)</code>	Returns a new string that concatenates this string with string s1.
<code>toUpperCase()</code>	Returns a new string with all letters in uppercase.
<code>toLowerCase()</code>	Returns a new string with all letters in lowercase
<code>trim()</code>	Returns a new string with whitespace characters trimmed on both sides.

Trả về độ dài chuỗi

- Phương thức `length()` trả về số ký tự trong chuỗi.
- Ví dụ

```
String message = "Welcome to Java";  
System.out.println("The length of " + message + " is "  
    + message.length());
```

The length of Welcome to Java is 15

- Indices
message
- | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| W | e | l | c | o | m | e | | t | o | | J | a | v | a |
- message.charAt(0) message.length() is 15 message.charAt(14)

Nối chuỗi

- Sử dụng phương thức `concat()` để nối 2 chuỗi.
- Ví dụ: nối chuỗi `s1` với `s2` lưu vào chuỗi `s3`

```
String s3 = s1.concat(s2);
```

- Sử dụng toán tử `+` để nối chuỗi

```
String s3 = s1 + s2;
```

```
String myString = message + " and " + "HTML";
```

- Một số ví dụ

```
// Three strings are concatenated
```

```
String message = "Welcome " + "to " + "Java";
```

```
// String Chapter is concatenated with number 2
```

```
String s = "Chapter" + 2; // s becomes Chapter2
```

```
// String Supplement is concatenated with character B
```

```
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB
```

Chuyển đổi chuỗi

- Phương thức `toLowerCase()` chuyển toàn bộ ký tự trong chuỗi hiện có thành ký tự chữ thường.
- Phương thức `toUpperCase()` chuyển toàn bộ ký tự trong chuỗi hiện có thành ký tự chữ hoa.

`"Welcome".toLowerCase()` returns a new string `welcome`.

`"Welcome".toUpperCase()` returns a new string `WELCOME`.

- Phương thức `trim()` loại bỏ ký tự trắng trong chuỗi như `'\t', '\f', '\r', '\n'`

`"\t Good Night \n".trim()` returns a new string `Good Night`.

So sánh chuỗi

- Các phương thức so sánh chuỗi

<i>Method</i>	<i>Description</i>
<code>equals(s1)</code>	Returns true if this string is equal to string s1.
<code>equalsIgnoreCase(s1)</code>	Returns true if this string is equal to string s1; it is case insensitive.
<code>compareTo(s1)</code>	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than s1.
<code>compareToIgnoreCase(s1)</code>	Same as <code>compareTo</code> except that the comparison is case insensitive.
<code>startsWith(prefix)</code>	Returns true if this string starts with the specified prefix.
<code>endsWith(suffix)</code>	Returns true if this string ends with the specified suffix.
<code>contains(s1)</code>	Returns true if s1 is a substring in this string.

So sánh chuỗi

```
import java.util.Scanner;

public class OrderTwoCities {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter two cities
        System.out.print("Enter the first city: ");
        String city1 = input.nextLine();
        System.out.print("Enter the second city: ");
        String city2 = input.nextLine();

        if (city1.compareTo(city2) < 0)
            System.out.println("The cities in alphabetical order are " +
                               city1 + " " + city2);
        else
            System.out.println("The cities in alphabetical order are " +
                               city2 + " " + city1);
    }
}
```

Enter the first city: New York

Enter the second city: Boston

The cities in alphabetical order are Boston New York

Chuyển đổi giữa chuỗi và số

- Sử dụng phương thức

```
int intValue = Integer.parseInt(intString);
```

```
double doubleValue = Double.parseDouble(doubleString);
```


Trắc nghiệm?

- Cho 3 chuỗi s1, s2, s3 có giá trị lần lượt như sau

```
String s1 = "Welcome to Java";  
String s2 = "Programming is fun";  
String s3 = "Welcome to Java";
```

- Kết quả các biểu thức sau trả về như thế nào?

- | | |
|-------------------------|-----------------------------|
| (a) s1 == s2 | (l) s1.lastIndexOf("o", 15) |
| (b) s2 == s3 | (m) s1.length() |
| (c) s1.equals(s2) | (n) s1.substring(5) |
| (d) s1.equals(s3) | (o) s1.substring(5, 11) |
| (e) s1.compareTo(s2) | (p) s1.startsWith("We1") |
| (f) s2.compareTo(s3) | (q) s1.endsWith("Java") |
| (g) s2.compareTo(s2) | (r) s1.toLowerCase() |
| (h) s1.charAt(0) | (s) s1.toUpperCase() |
| (i) s1.indexOf('j') | (t) s1.concat(s2) |
| (j) s1.indexOf("to") | (u) s1.contains(s2) |
| (k) s1.lastIndexOf('a') | (v) "\t We1 \t".trim() |

Lớp StringBuilder

StringBuilder

Các phương thức có sẵn của lớp StringBuilder

StringBuilder

- Lớp StringBuilder dùng mô tả các dữ liệu dạng chuỗi có thể sửa đổi linh động.
- Sử dụng StringBuilder để thao tác chuỗi có ưu điểm tiết kiệm bộ nhớ và tăng tốc khi chương trình có nhiều thao tác xử lý với chuỗi.
- StringBuilder thuộc gói java.lang

Khởi tạo chuỗi với **StringBuilder**

- Khác với String, **StringBuilder** có cách thức khởi tạo riêng thông qua các hàm tạo:
 - **StringBuilder()**: hàm tạo mặc định sẽ khởi tạo một mảng 16 ký tự.
 - **StringBuilder(int capacity)**: hàm tạo dùng để khởi tạo số ký tự ban đầu là capacity
 - **StringBuider(String str)**: hàm tạo dùng khởi tạo một mảng ký tự lưu trữ chuỗi str.

java.lang.StringBuilder

```
+StringBuilder()  
+StringBuilder(capacity: int)  
+StringBuilder(s: String)
```

Constructs an empty string builder with capacity 16.
Constructs a string builder with the specified capacity.
Constructs a string builder with the specified string.

Các phương thức thường dùng của StringBuilder



- Các phương thức thường dùng:
 - **append()**: đính thêm các ký tự hoặc chuỗi vào cho StringBuilder
 - **insert()**: chèn thêm ký tự hoặc chuỗi vào vị trí xác định trong StringBuilder
 - **delete()**: xóa ký tự hoặc chuỗi trong StringBuilder
 - **reverse()**: đảo ngược chuỗi trong StringBuilder.
 - **toString()**: chuyển StringBuilder thành String.

java.lang.StringBuilder

```
+append(data: char[]): StringBuilder
+append(data: char[], offset: int, len: int):
  StringBuilder
+append(v: aPrimitiveType): StringBuilder

+append(s: String): StringBuilder
+delete(startIndex: int, endIndex: int):
  StringBuilder
+deleteCharAt(index: int): StringBuilder
+insert(index: int, data: char[], offset: int,
  len: int): StringBuilder
+insert(offset: int, data: char[]):
  StringBuilder
+insert(offset: int, b: aPrimitiveType):
  StringBuilder
+insert(offset: int, s: String): StringBuilder
+replace(startIndex: int, endIndex: int, s:
  String): StringBuilder
+reverse(): StringBuilder
+setCharAt(index: int, ch: char): void
```

Appends a char array into this string builder.
Appends a subarray in data into this string builder.

Appends a primitive type value as a string to this builder.
Appends a string to this string builder.
Deletes characters from startIndex to endIndex-1.

Deletes a character at the specified index.
Inserts a subarray of the data in the array into the builder at the specified index.
Inserts data into this builder at the position offset.

Inserts a value converted to a string into this builder.

Inserts a string into this builder at the position offset.
Replaces the characters in this builder from startIndex to endIndex-1 with the specified string.
Reverses the characters in the builder.
Sets a new character at the specified index in this builder.

java.lang.StringBuilder

```
+toString(): String
+capacity(): int
+charAt(index: int): char
+length(): int
+setLength(newLength: int): void
+substring(startIndex: int): String
+substring(startIndex: int, endIndex: int):
  String
+trimToSize(): void
```

Returns a string object from the string builder.
Returns the capacity of this string builder.
Returns the character at the specified index.
Returns the number of characters in this builder.
Sets a new length in this builder.
Returns a substring starting at startIndex.
Returns a substring from startIndex to endIndex-1.
Reduces the storage size used for the string builder.

Ví dụ sử dụng StringBuilder

- Đoạn mã sau tạo ra chuỗi: "Welcome to Java"

```
StringBuilder stringBuilder = new StringBuilder();  
stringBuilder.append("Welcome");  
stringBuilder.append(' ');  
stringBuilder.append("to");  
stringBuilder.append(' ');  
stringBuilder.append("Java");
```

Welcome to Java.

- Chèn chuỗi "HTML and" vào vị trí thứ 11

```
stringBuilder.insert(11, "HTML and ");
```

Welcome to HTML and Java.

- Sử dụng các phương thức delete(), deleteCharAt(), reverse(), replace(), setCharAt()

```
stringBuilder.delete(8, 11) changes the builder to Welcome Java.  
stringBuilder.deleteCharAt(8) changes the builder to Welcome o Java.  
stringBuilder.reverse() changes the builder to avaJ ot emocleW.  
stringBuilder.replace(11, 15, "HTML") changes the builder to Welcome to HTML.  
stringBuilder.setCharAt(0, 'w') sets the builder to welcome to Java.
```

Biểu thức chính quy (Regular Expression)

Các phương thức có sẵn của lớp StringBuilder

Biểu thức chính quy

- Regular Expression viết tắt là Regex là một chuỗi mẫu được sử dụng để quy định dạng thức của các chuỗi. Nếu một chuỗi nào đó phù hợp với mẫu dạng thức thì chuỗi đó được gọi là so khớp.
- Ví dụ:
 - $[0-9]\{3,7\}$ là biểu thức chính quy để so khớp các chuỗi từ 3 đến 7 ký tự số.
 - $[0-9]$ đại diện cho 1 ký tự số
 - $\{3,7\}$ đại diện cho số lần xuất hiện (ít nhất 3, nhiều nhất 7)

Regular Expression trong Java

- Java cung cấp gói **java.util.regex** cho pattern so khớp với các Regular Expression.
- Gói **java.util.regex** chủ yếu chứa 3 lớp sau:
 - Lớp **Pattern**: *được sử dụng để xác định một khuôn mẫu cho các biểu thức chính quy. Lớp này không có constructor, sử dụng phương thức static compile(String) để tạo đối tượng.*
 - Lớp **Matcher**: *được sử dụng để thực hiện các hoạt động so khớp trên một chuỗi ký tự. Lớp này không có constructor, sử dụng phương thức matcher(String) của đối tượng Pattern để tạo đối tượng.*
 - **PatternSyntaxException**: xảy ra khi có lỗi cú pháp trong mẫu Regular Expression.

Sử dụng Regular Expression trong Java



- Phương thức `matches()` thuộc lớp `Matcher` trong Java xác định có hay không chuỗi này so khớp với regular expression đã cho.
- Cú pháp: `public boolean matches(String regex)`
- Trong đó:
 - **regex** – Regular expression từ đó chuỗi này được so khớp.
 - Trả về `true` nếu và chỉ nếu chuỗi này so khớp với regular expression đã cung cấp.

Sử dụng Regular Expression trong Java



- Các cách sử dụng phương thức matches()
- Cách 1:

```
Pattern p = Pattern.compile(".s");  
Matcher m = p.matcher("as");  
boolean b = m.matches();
```

- Cách 2:

```
boolean b2=Pattern.compile(".s").matcher("as").matches();
```

- Cách 3:

```
boolean b3 = Pattern.matches(".s", "as");
```

Quy tắc viết biểu thức chính quy

- Regex gồm hằng ký tự và các ký tự đặc biệt.

<i>Regular Expression</i>	<i>Matches</i>	<i>Example</i>
x	a specified character x	Java matches Java
.	any single character	Java matches J..a
(ab cd)	ab or cd	ten matches t(en im)
[abc]	a, b, or c	Java matches Ja[uvwx]a
[^abc]	any character except a, b, or c	Java matches Ja[^ars]a
[a-z]	a through z	Java matches [A-M]av[a-d]
[^a-z]	any character except a through z	Java matches Jav[^b-d]
[a-e[m-p]]	a through e or m through p	Java matches [A-G[I-M]]av[a-d]
[a-e&&[c-p]]	intersection of a-e with c-p	Java matches [A-P&&[I-M]]av[a-d]

Quy tắc viết biểu thức chính quy

- Regex gồm hằng ký tự và các ký tự đặc biệt.

<code>\d</code>	a digit, same as <code>[0-9]</code>	<code>Java2</code> matches <code>"Java[\d]"</code>
<code>\D</code>	a non-digit	<code>\$Java</code> matches <code>"[\D][\D]ava"</code>
<code>\w</code>	a word character	<code>Java1</code> matches <code>"[\w]ava[\w]"</code>
<code>\W</code>	a non-word character	<code>\$Java</code> matches <code>"[\W][\w]ava"</code>
<code>\s</code>	a whitespace character	<code>"Java 2"</code> matches <code>"Java\s2"</code>
<code>\S</code>	a non-whitespace char	<code>Java</code> matches <code>"[\S]ava"</code>

Quy tắc viết biểu thức chính quy



- Regex gồm hằng ký tự và các ký tự đặc biệt.

p^*	zero or more occurrences of pattern p	<code>aaaabb</code> matches " <code>a*bb</code> " <code>ababab</code> matches " <code>(ab)*</code> "
p^+	one or more occurrences of pattern p	<code>a</code> matches " <code>a+b*</code> " <code>able</code> matches " <code>(ab)+.*</code> "
$p?$	zero or one occurrence of pattern p	<code>Java</code> matches " <code>J?Java</code> " <code>Java</code> matches " <code>J?ava</code> "
$p\{n\}$	exactly n occurrences of pattern p	<code>Java</code> matches " <code>Ja{1}.*</code> " <code>Java</code> does not match " <code>.{2}</code> "
$p\{n,\}$	at least n occurrences of pattern p	<code>aaaa</code> matches " <code>a{1,}</code> " <code>a</code> does not match " <code>a{2,}</code> "
$p\{n,m\}$	between n and m occurrences (inclusive)	<code>aaaa</code> matches " <code>a{1,9}</code> " <code>abb</code> does not match " <code>a{2,9}bb</code> "

Ví dụ về sử dụng regular expression

- Ví dụ 1: Tạo pattern để kiểm tra giá trị đưa vào phải theo mẫu:
XXX-XX-XXXX.

Với x là một số.

```
[\\d]{3}-[\\d]{2}-[\\d]{4}
```

```
"111-22-3333".matches("[\\d]{3}-[\\d]{2}-[\\d]{4}") returns true.  
"11-22-3333".matches("[\\d]{3}-[\\d]{2}-[\\d]{4}") returns false.
```

Ví dụ về sử dụng regular expression

- Ví dụ 2: Tạo pattern để kiểm tra giá trị một số đưa vào phải là số chẵn.

```
[\\d]*[02468]
```

```
"123".matches("[\\d]*[02468]") returns false.
```

```
"122".matches("[\\d]*[02468]") returns true.
```


Ví dụ về sử dụng regular expression

- Ví dụ 3: Tạo pattern để kiểm tra giá trị đưa vào phải theo mẫu:
(xxx) xxx-xxxx.
Với x là một số.

```
[\\d]*[02468]
```

```
"123".matches("[\\d]*[02468]") returns false.
```

```
"122".matches("[\\d]*[02468]") returns true.
```

Ví dụ về sử dụng regular expression

- Ví dụ 4: Tạo pattern để kiểm tra giá trị đưa vào phải là một chuỗi gồm 25 ký tự và ký tự đầu tiên phải viết hoa.

`[A-Z][a-zA-Z]{1,24}`

`"Smith".matches("[A-Z][a-zA-Z]{1,24}")` returns **true**.

`"Jones123".matches("[A-Z][a-zA-Z]{1,24}")` returns **false**.

Tổng kết

Hướng dẫn

- Hướng dẫn làm bài thực hành và bài tập
- Chuẩn bị bài tiếp: OOAD SODLID