1495 UNIVERSITY OF ABERDEEN

JC2002 Java Programming

Lecture 33: Regular expressions

Regular expressions (regex)

- A *regular expression* (*regex*) is a string that describes a search pattern for matching characters in other strings
  - Such expressions are useful for validating input and ensuring that data is in a particular format
- Regular expressions can be used to perform all types of text search and text replace operations
- A large and complex regular expression is used for example to validate the syntax of a program
  - If the program code does not match the regular expression, the compiler knows that there is a syntax error in the code

Regular expression characters

- A regex consists of *literal characters* and *metacharacters*
  - Literal characters are regular characters with a literal meaning: for example, character `'b'` is a literal character matching with character `'b'`
  - Metacharacters are characters that have special meaning in regex: for example, metacharacter `'.'` (dot) matches with any character
  - Some metacharacters are preceded by the *escape sequence* (backslash) `'\'`: for example, metacharacter `'\d'` matches with any digit
  - Backslash is also used to distinguish literal characters from metacharacters: for example, `'*'` is a metacharacter, and `'\*'` is a literal character matching with character `'*'` (asterisk)

Some common metacharacters

Metacharacter	Description
.	Matches any character (except newline)
^	Matches the starting position within the string
$	Matches the ending position of the string
*	Matches the preceding element zero or more times
?	Matches the preceding element zero or one time
+	Matches the preceding element one or more times
|	Matches any of the patterns separated by ‘|’

Examples of using metacharacters

Regex	Example matches and non-matches
`bo.`	Matches “box” and “boy” but not “but” or “bo”
`^cat`	Matches “cat” but not “a cat”
`hat$`	Matches “hat” and “chat” but not “hatch”
`c*at`	Matches “at” and “cat” and “ccat” but not “chat”
`c?at`	Matches “at” and “cat” but not “ccat”
`c?at`	Matches “cat” and “ccat” but not “at”
`cat|dog`	Matches “cat” and “dog” but not “cow”

Some common character classes

- The *character class* is the most basic regex concept after literal match
  - Character classes are defined by metacharacters that match with specific types of characters, like digits or whitespaces
- Some commonly used examples of character classes:

Character	Matches	Character	Matches
`\d`	Any digit	`\D`	Any non-digit
`\w`	Any word character	`\W`	Any non-word character
`\s`	Any whitespace character	`\S`	Any non-whitespace character
`\b`	Word boundaries		

Using brackets in regex

- Brackets `[ ]` are used to match any single character that is contained within the brackets
  - For example, `[abc]` matches `'a'`, `'b'`, and `'c'` but not `'d'`
- Within brackets, metacharacter `'^'` is used to match a character that is NOT contained within the brackets
  - For example, `[^ab]` matches `'c'` and `'z'` but not `'a'` or `'b'`
- Within brackets, `'-'` is used to define to match a range of characters
  - For example, `[a-d]` matches `'a'`, `'b'`, `'c'`, and `'d'` but not `'e'`

Quantifiers

- Regex *quantifiers* are used to specify length of a sequence to match

Quantifier	Description
`n{x}`	Matches any string that contains a sequence of *x* times character ‘n’ (*x* is a number)
`n{x,y}`	Matches any string that contains a sequence of at least *x* but no more than *y* times character ‘n’
`n{x,}`	Matches any string that contains a sequence of at least *x* times character ‘n’

String methods for regex operations

- Class `String` provides several methods for performing regex operations
  - Method **`matches()`** takes a `String` object containing a regex as input argument and returns `true` only if the *whole* string matches the regex
  - Method **`split()`** uses regex expression as input to find delimiters for tokenising the string
  - Method **`replaceAll()`** uses the regex input argument to find matching substrings and replaces them with the replacement argument
  - Method **`replaceFirst()`** is similar to `replaceAll()`, but replaces only the first matching substring
  - Note that `String` method `replace()` does not support regex!

Regex example using String methods (1)

```
1  import java.util.Scanner;
2  public class StringRegexExample {
3    public static void main(String[] args){
4      Scanner scanner = new Scanner(System.in);
5      System.out.println("Enter 'stop' when you want to finish!");
6      do {
7        System.out.print("Enter the email: ");
8        Sinput = scanner.nextLine();
9        if(input.matches("[a-z]+@[a-z]+\\.[a-z]{2,3}")) {
10         System.out.println("Your email is valid!");
11       }
12       else if(input.matches("stop|Stop|STOP")) {
13         break;
14       }
15       else {
16         System.out.println("Your email is not valid!");
17       }
18     } while(true);
19   }
20 }
```

Regex example using String methods (2)

```
1  import java.util.Scanner;
2  public class StringRegexE
3    public static void main
4      Scanner scanner = new
5      System.out.println("E
6      do {
7        System.out.print("Enter the email: ");
8        Sinput = scanner.nextLine();
9        if(input.matches("[a-z]+@[a-z]+\\.[a-z]{2,3}")) {
10         System.out.println("Your email is valid!");
11       }
12       else if(input.matches("stop|Stop|STOP")) {
13         break;
14       }
15       else {
16         System.out.println("Your email is not valid!");
17       }
18     } while(true);
19   }
20 }
```

For simplicity, we assume that the email format is `username@domain.xxx`, and only lowercase letters are allowed in the username and URL

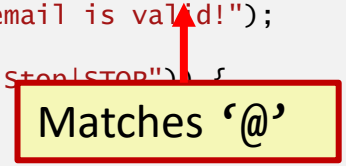
Regex example using String methods (3)

```
1  import java.util.Scanner;
2  public class StringRegexExample {
3    public static void main(String[] args){
4      Scanner scanner = new Scanner(System.in);
5      System.out.println("Enter 'stop' when you want to finish!");
6      do {
7        System.out.print("Enter the email: ");
8        Sinput = scanner.nextLine();
9        if(input.matches("[a-z]+@[a-z]+\\.[a-z]{2,3}")) {
10         System.out.println("Your email is valid!");
11       }
12       else if(input.matches("stop|Stop|STOP")) {
13         break;
14       }
15       else {
16         System.out.printl
17       }
18     } while(true);
19   }
20 }
```

Matches a sequence of one or more lowercase letters

Regex example using String methods (4)

```
1  import java.util.Scanner;
2  public class StringRegexExample {
3    public static void main(String[] args){
4      Scanner scanner = new Scanner(System.in);
5      System.out.println("Enter 'stop' when you want to finish!");
6      do {
7        System.out.print("Enter the email: ");
8        Sinput = scanner.nextLine();
9        if(input.matches("[a-z]+@[a-z]+\\.[a-z]{2,3}")) {
10         System.out.println("Your email is valid!");
11       }
12       else if(input.matches("stop|Stop|STOP")) {
13         break;
14       }
15       else {
16         System.out.println("Your email is not valid!");
17       }
18     } while(true);
19   }
20 }
```



Regex example using String methods (5)

```
1  import java.util.Scanner;
2  public class StringRegexExample {
3    public static void main(String[] args){
4      Scanner scanner = new Scanner(System.in);
5      System.out.println("Enter 'stop' when you want to finish!");
6      do {
7        System.out.print("Enter the email: ");
8        Sinput = scanner.nextLine();
9        if(input.matches("[a-z]+@[a-z]+\\.[a-z]{2,3}")) {
10         System.out.println("Your email is valid!");
11       }
12       else if(input.matches("stop|Stop|STOP")) {
13         break;
14       }
15       else {
16         System.out.println("Your em
17       }
18     } while(true);
19   }
20 }
```

Matches a sequence of one or more lowercase letters

Regex example using String methods (6)

```
1  import java.util.Scanner;
2  public class StringRegexExample {
3    public static void main(String[] args){
4      Scanner scanner = new Scanner(System.in);
5      System.out.println("Enter 'stop' when you want to finish!");
6      do {
7        System.out.print("Enter the email: ");
8        Sinput = scanner.nextLine();
9        if(input.matches("[a-z]+@[a-z]+\\.[a-z]{2,3}")) {
10         System.out.println("Your email is valid!");
11       }
12       else if(input.matches("stop|Stop|STOP")) {
13         break;
14       }
15       else {
16         System.out.printl
17       }
18     } while(true);
19   }
20 }
```

Matches '.': note that in a Java string, regex character '\.' must be written as '\\.', because Java compiler assumes backslash as an escape character before regex compiler!

Regex example using String methods (7)

```
1  import java.util.Scanner;
2  public class StringRegexExample {
3    public static void main(String[] args){
4      Scanner scanner = new Scanner(System.in);
5      System.out.println("Enter 'stop' when you want to finish!");
6      do {
7        System.out.print("Enter the email: ");
8        Sinput = scanner.nextLine();
9        if(input.matches("[a-z]+@[a-z]+\\.[a-z]{2,3}")) {
10         System.out.println("Your email is valid!");
11       }
12       else if(input.matches("stop|Stop|STOP")) {
13         break;
14       }
15       else {
16         System.out.println("Your email is not va
17       }
18     } while(true);
19   }
20 }
```

Matches a sequence of two to three lowercase letters

Regex example using String methods (8)

```
1  import java.util.Scanner;
2  public class StringRegexExample {
3    public static void main(String[] args){
4      Scanner scanner = new Scanner(System.in);
5      System.out.println("Enter 'stop' when you want to finish!");
6      do {
7        System.out.print("Enter the email: ");
8        Sinput = scanner.nextLine();
9        if(input.matches("[a-z]+@[a-z]+\\.[a-z]{2,3}")) {
10         System.out.println("Your email is valid!");
11       }
12         else if(input.matches("stop|Stop|STOP")) {
13         break;
14       }
15       else {
16         System.out.println("Your email is not valid!");
17       }
18     } while(true);
19   }
20 }
```

Accepts different ways to write “stop”

Regex example using String methods (9)

```
1  import java.util.Scanner;
2  public class StringRegexExample {
3    public static void main(String[] args){
4      Scanner scanner = new Scanner(System.in);
5      System.out.println("Enter 'stop' when you want to finish!");
6      do {
7        System.out.print("Enter your email: ");
8        Sinput = scanner.nextLine();
9        if(input.matches("[a-z]+@[a-z]+\\.[a-z]{2,3}")) {
10         System.out.println("Your email is valid!");
11       }
12       else if(input.matches("stop|Stop|STOP")) {
13         break;
14       }
15       else {
16         System.out.println("Your email
17       }
18     } while(true);
19   }
20 }
```

```
$ java StringRegexExample
Enter 'stop' when you want to finish!
Enter the email: teacher@school.edu
Your email is valid!
Enter the email: james.smith@company.com
Your email is not valid!
Enter the email: STOP
$
```

Classes Pattern and Matcher

- Java does not have any built-in regex class, but we can import **`java.util.regex`** package to work with regular expressions using the following classes:
  - Class **`Pattern`**: defines a pattern (to be used in a search)
  - Class **`Matcher`**: used to search for the pattern
  - Class **`PatternSyntaxException`**: defines an exception thrown when there is a syntax error in a regex string

Using Pattern and Matcher

- A Pattern object is created by static method **`Pattern.compile()`**
  - The first argument is a regex string specifying the pattern to be searched
  - The second argument (optional) specifies flags instructing how the search is performed, for example flag **`Pattern.CASE_INSENSITIVE`** instructs ignoring the case
- Method **`matcher()`** of the `Pattern` object is used to search the pattern in the string given as input argument; the method returns a `Matcher` object with information about the result
- Method **`find()`** of the `Matcher` object returns `true` if the pattern was found, and `false` if it was not found

Pattern and Matcher example (1)

```
1  import java.util.regex.Matcher;
2  import java.util.regex.Pattern;
3
4  public class PatternMatcherExample {
5    public static void main(String[] args){
6      Pattern pattern = Pattern.compile("[0-3]\\d/[0-1]\\d/\\d\\d\\d\\d");
7      String text = "John Smith was born on 14/05/1973.\n" +
8                    "His wife Jane was born on 09/12/1976.\n" +
9                    "They had a son, born on 31/10/1997 " +
10                   "and a daughter, born on 01/02/2001.";
11     Matcher matcher = pattern.matcher(text);
12     while(matcher.find()) {
13       System.out.println("Date found: " + matcher.group());
14     }
15   }
16 }
```

Pattern and Matcher example (2)

```
1  import java.util.regex.Matcher;
2  import java.util.regex.Pattern;
3
4  public class PatternMatcherExample {
5    public static void main(String[] args){
6      Pattern pattern = Pattern.compile("[0-3]\\d/[0-1]\\d/\\d\\d\\d\\d");
7      String text = "John Smith was born on 14/05/1973.\n" +
8                    "His wife Jane was born on 09/12/1976.\n" +
9                    "They had a son, born on 31/10/1997." +
10                   "and a daughter, born on 01/
11     Matcher matcher = pattern.matcher(text);
12     while(matcher.find()) {
13       System.out.println("Date found: " + matc
14     }
15   }
16 }
```

Compile pattern matcher for dates in format dd/mm/yyyy; (note that this regex validates the dates only weakly)

Pattern and Matcher example (3)

```
1  import java.util.regex.Matcher;
2  import java.util.regex.Pattern;
3
4  public class PatternMatcherExample {
5    public static void main(String[] args){
6      Pattern pattern = Pattern.compile("[0-3]\\d/[0-1]\\d/\\d\\d\\d\\d");
7      String text = "John Smith was born on 14/05/1973.\n" +
8                    "His wife Jane was born on 09/12/1976.\n" +
9                    "They had a son, born on 31/10/1997 " +
10                   "and a daughter, born on 01/02/2001.";
11     Matcher matcher = pattern.matcher(text);
12     while(matcher.find()) {
13       System.out.println("Date found: " + matcher.group());
14     }
15   }
16 }
```

Try to find the specified patterns in the text

Pattern and Matcher example (4)

```
1  import java.util.regex.Matcher;
2  import java.util.regex.Pattern;
3
4  public class PatternMatcherExample {
5    public static void main(String[] args){
6      Pattern pattern = Pattern.compile("[0-3]\\d/[0-1]\\d/\\d\\d\\d\\d");
7      String text = "John Smith was born on 14/05/1973.\n" +
8                    "His wife Jane was born on 09/12/1976.\n" +
9                    "They had a son, born on 31/10/1997 " +
10                   "and a daughter, born on 01/02/2001.";
11     Matcher matcher = pattern.matcher(text);
12      while(matcher.find()) {
13        System.out.println("Date found: " + matcher.group());
14      }
15   }
16 }
```

Loop through all the matching substrings found in the input string

Pattern and Matcher example (5)

```
1  import java.util.regex.Matcher;
2  import java.util.regex.Pattern;
3
4  public class PatternMatcherExample {
5    public static void main(String[] args){
6      Pattern pattern = Pattern.compile("[0-3]\\d/[0-1]\\d/\\d\\d\\d\\d");
7      String text = "John Smith was born on 14/05/1973.\n" +
8                    "His wife Jane was born on 09/12/1976.\n" +
9                    "They had a son, born on 31/10/1997 " +
10                   "and a daughter, born on 01/02/2001.";
11     Matcher matcher = pattern.matcher(text);
12     while(matcher.find()) {
13       System.out.println("Date found: " + matcher.group());
14     }
15   }
16 }
```

```
$ java PatternMatcherExample
Date found: 14/05/1973
Date found: 09/12/1976
Date found: 31/10/1997
Date found: 01/02/2001
$
```

Questions, comments?