# CE0825a: Object Oriented Programming II
# 9: Regular Expressions, Databases

James A Sutherland

Abertay University

Monday, 7th March 2016

## Regular Expressions

- Invented 1956
- Stephen Cole Kleene (pronounced Clay-knee)
- Widely used now: PostgreSQL, MySQL, Apache, Java, JavaScript, PHP, . . .
- Also Linux/Unix command line tools: awk, vi, egrep

## Regular Expressions

- Invented 1956
- Stephen Cole Kleene (pronounced Clay-knee)
- Widely used now: PostgreSQL, MySQL, Apache, Java, JavaScript, PHP, . . .
- Also Linux/Unix command line tools: awk, vi, egrep

## Regular Expressions

- Invented 1956
- Stephen Cole Kleene (pronounced Clay-knee)
- Widely used now: PostgreSQL, MySQL, Apache, Java, JavaScript, PHP, . . .
- Also Linux/Unix command line tools: awk, vi, egrep

## Regular Expressions

- Invented 1956
- Stephen Cole Kleene (pronounced Clay-knee)
- Widely used now: PostgreSQL, MySQL, Apache, Java, JavaScript, PHP, . . .
- Also Linux/Unix command line tools: awk, vi, egrep

## Perl

- Larry Wall, NASA sysadmin at JPL in 1987
- Perl: officially not an acronym ...
- ...or Practical Extraction and Reporting Language
- ...or the Pathologically Eclectic Rubbish Lister!
- “Perl Compatible Regexes”

## Perl

- Larry Wall, NASA sysadmin at JPL in 1987
- Perl: officially not an acronym . . .
- . . . or Practical Extraction and Reporting Language
- . . . or the Pathologically Eclectic Rubbish Lister!
- “Perl Compatible Regexes”

## Perl

- Larry Wall, NASA sysadmin at JPL in 1987
- Perl: officially not an acronym . . .
- . . . or Practical Extraction and Reporting Language
- . . . or the Pathologically Eclectic Rubbish Lister!
- “Perl Compatible Regexes”

## Perl

- Larry Wall, NASA sysadmin at JPL in 1987
- Perl: officially not an acronym . . .
- . . . or Practical Extraction and Reporting Language
- . . . or the Pathologically Eclectic Rubbish Lister!
- “Perl Compatible Regexes”

## Perl

- Larry Wall, NASA sysadmin at JPL in 1987
- Perl: officially not an acronym . . .
- . . . or Practical Extraction and Reporting Language
- . . . or the Pathologically Eclectic Rubbish Lister!
- "Perl Compatible Regexes"

## Delimiters

- Minor variations in each language
- Widely used so plenty of examples
- Commonly shown wrapped in //
- Paired for replacement: s/old/new/
- Modifiers on the end: s/old/new/gi
- In Java, usually a String: “like this”

## Delimiters

- Minor variations in each language
- Widely used so plenty of examples
- Commonly shown wrapped in //
- Paired for replacement: s/old/new/
- Modifiers on the end: s/old/new/gi
- In Java, usually a String: “like this”

## Delimiters

- Minor variations in each language
- Widely used so plenty of examples
- Commonly shown wrapped in //
- Paired for replacement: s/old/new/
- Modifiers on the end: s/old/new/gi
- In Java, usually a String: “like this”

## Delimiters

- Minor variations in each language
- Widely used so plenty of examples
- Commonly shown wrapped in //
- Paired for replacement: s/old/new/
- Modifiers on the end: s/old/new/gi
- In Java, usually a String: “like this”

## Delimiters

- Minor variations in each language
- Widely used so plenty of examples
- Commonly shown wrapped in //
- Paired for replacement: s/old/new/
- Modifiers on the end: s/old/new/gi
- In Java, usually a String: “like this”

## Delimiters

- Minor variations in each language
- Widely used so plenty of examples
- Commonly shown wrapped in //
- Paired for replacement: s/old/new/
- Modifiers on the end: s/old/new/gi
- In Java, usually a String: “like this”

## Basic Language

- String of characters: ‘grey’
- Vertical bar for or: ‘grey|gray’
- Grouping brackets: ‘gr(a|e)y’
- ? for optional: ‘colou?r’
- * for zero or more: ‘gre*n’
- + for one or more: ‘Kha+n’
- {n}: exactly n: ‘gre{2}n’
- {n,}: n or more: ‘Kha{3,}n’
- {n,m}: between n and m: ‘Kha{3,7}n’

## Basic Language

- String of characters: 'grey'
- Vertical bar for or: 'grey|gray'
- Grouping brackets: 'gr(a|e)y'
- ? for optional: 'colou?r'
- * for zero or more: 'gre*n'
- + for one or more: 'Kha+n'
- {n}: exactly n: 'gre{2}n'
- {n,}: n or more: 'Kha{3,}n'
- {n,m}: between n and m: 'Kha{3,7}n'

## Basic Language

- String of characters: ‘grey’
- Vertical bar for or: ‘grey|gray’
- Grouping brackets: ‘gr(a|e)y’
- ? for optional: ‘colou?r’
- * for zero or more: ‘gre*n’
- + for one or more: ‘Kha+n’
- {n}: exactly n: ‘gre{2}n’
- {n,}: n or more: ‘Kha{3,}n’
- {n,m}: between n and m: ‘Kha{3,7}n’

## Basic Language

- String of characters: ‘grey’
- Vertical bar for or: ‘grey|gray’
- Grouping brackets: ‘gr(a|e)y’
- ? for optional: ‘colou?r’
- * for zero or more: ‘gre*n’
- + for one or more: ‘Kha+n’
- {n}: exactly n: ‘gre{2}n’
- {n,}: n or more: ‘Kha{3,}n’
- {n,m}: between n and m: ‘Kha{3,7}n’

## Basic Language

- String of characters: 'grey'
- Vertical bar for or: 'grey|gray'
- Grouping brackets: 'gr(a|e)y'
- ? for optional: 'colou?r'
- * for zero or more: 'gre*n'
- + for one or more: 'Kha+n'
- {n}: exactly n: 'gre{2}n'
- {n,}: n or more: 'Kha{3,}n'
- {n,m}: between n and m: 'Kha{3,7}n'

## Basic Language

- String of characters: ‘grey’
- Vertical bar for or: ‘grey|gray’
- Grouping brackets: ‘gr(a|e)y’
- ? for optional: ‘colou?r’
- * for zero or more: ‘gre*n’
- + for one or more: ‘Kha+n’
- {n}: exactly n: ‘gre{2}n’
- {n,}: n or more: ‘Kha{3,}n’
- {n,m}: between n and m: ‘Kha{3,7}n’

## Basic Language

- String of characters: 'grey'
- Vertical bar for or: 'grey|gray'
- Grouping brackets: 'gr(a|e)y'
- ? for optional: 'colou?r'
- * for zero or more: 'gre*n'
- + for one or more: 'Kha+n'
- {n}: exactly n: 'gre{2}n'
- {n,}: n or more: 'Kha{3,}n'
- {n,m}: between n and m: 'Kha{3,7}n'

## Basic Language

- String of characters: ‘grey’
- Vertical bar for or: ‘grey|gray’
- Grouping brackets: ‘gr(a|e)y’
- ? for optional: ‘colou?r’
- * for zero or more: ‘gre*n’
- + for one or more: ‘Kha+n’
- {n}: exactly n: ‘gre{2}n’
- {n,}: n or more: ‘Kha{3,}n’
- {n,m}: between n and m: ‘Kha{3,7}n’

## Basic Language

- String of characters: ‘grey’
- Vertical bar for or: ‘grey|gray’
- Grouping brackets: ‘gr(a|e)y’
- ? for optional: ‘colou?r’
- * for zero or more: ‘gre*n’
- + for one or more: ‘Kha+n’
- {n}: exactly n: ‘gre{2}n’
- {n,}: n or more: ‘Kha{3,}n’
- {n,m}: between n and m: ‘Kha{3,7}n’

## Character Groups

- .: Any single character
- [*abc*]: One of a list of characters
- [^abc]: Anything except that list
- ^: Start of a line or this string
- $: End of a line or this string
- (): Grouping (see previous)

## Character Groups

- .: Any single character
- [*abc*]: One of a list of characters
- [^abc]: Anything except that list
- ^: Start of a line or this string
- $: End of a line or this string
- (): Grouping (see previous)

## Character Groups

- .: Any single character
- [*abc*]: One of a list of characters
- [^abc]: Anything except that list
- ^: Start of a line or this string
- $: End of a line or this string
- (): Grouping (see previous)

## Character Groups

- .: Any single character
- [*abc*]: One of a list of characters
- [^abc]: Anything except that list
- ^: Start of a line or this string
- $: End of a line or this string
- (): Grouping (see previous)

## Character Groups

- .: Any single character
- [*abc*]: One of a list of characters
- [^abc]: Anything except that list
- ^: Start of a line or this string
- $: End of a line or this string
- (): Grouping (see previous)

## Character Groups

- .: Any single character
- [*abc*]: One of a list of characters
- [^abc]: Anything except that list
- ^: Start of a line or this string
- $: End of a line or this string
- (): Grouping (see previous)

## Character Classes

- \d: Any digit [0 – 9]
- \D: Any non-digit [^0-9]
- \s: Whitespace [ \t\n\x0b\r\f]
- \S: Any non-whitespace [^\s]
- \w: Any word character [a-zA-Z_0-9]
- \W: Any non-word character
- \b: Word boundary
- \\: Backslash - \escapes next character
- Remember \has meaning in Java Strings too, so double them all up! \\\\to match \.

## Character Classes

- \d: Any digit [0 – 9]
- \D: Any non-digit [^0-9]
- \s: Whitespace [ \t\n\x0b\r\f]
- \S: Any non-whitespace [^\s]
- \w: Any word character [a-zA-Z_0-9]
- \W: Any non-word character
- \b: Word boundary
- \\: Backslash - \escapes next character
- Remember \has meaning in Java Strings too, so double them all up! \\\\to match \.

## Character Classes

- \d: Any digit [0 – 9]
- \D: Any non-digit [^0-9]
- \s: Whitespace [ \t\n\x0b\r\f]
- \S: Any non-whitespace [^\s]
- \w: Any word character [a-zA-Z_0-9]
- \W: Any non-word character
- \b: Word boundary
- \\: Backslash - \escapes next character
- Remember \has meaning in Java Strings too, so double them all up! \\\\to match \.

## Character Classes

- \d: Any digit [0 – 9]
- \D: Any non-digit [^0-9]
- \s: Whitespace [ \t\n\x0b\r\f]
- \S: Any non-whitespace [^\s]
- \w: Any word character [a-zA-Z_0-9]
- \W: Any non-word character
- \b: Word boundary
- \\: Backslash - \escapes next character
- Remember \has meaning in Java Strings too, so double them all up! \\\\to match \.

## Character Classes

- \d: Any digit [0 – 9]
- \D: Any non-digit [^0-9]
- \s: Whitespace [ \t\n\x0b\r\f]
- \S: Any non-whitespace [^\s]
- \w: Any word character [a-zA-Z_0-9]
- \W: Any non-word character
- \b: Word boundary
- \\: Backslash - \escapes next character
- Remember \has meaning in Java Strings too, so double them all up! \\\\to match \.

## Character Classes

- \d: Any digit [0 – 9]
- \D: Any non-digit [^0-9]
- \s: Whitespace [ \t\n\x0b\r\f]
- \S: Any non-whitespace [^\s]
- \w: Any word character [a-zA-Z_0-9]
- \W: Any non-word character
- \b: Word boundary
- \\: Backslash - \escapes next character
- Remember \has meaning in Java Strings too, so double them all up! \\\\to match \.

## Character Classes

- \d: Any digit [0 – 9]
- \D: Any non-digit [^0-9]
- \s: Whitespace [ \t\n\x0b\r\f]
- \S: Any non-whitespace [^\s]
- \w: Any word character [a-zA-Z_0-9]
- \W: Any non-word character
- \b: Word boundary
- \\: Backslash - \escapes next character
- Remember \has meaning in Java Strings too, so double them all up! \\\\to match \.

## Character Classes

- \d: Any digit [0 – 9]
- \D: Any non-digit [^0-9]
- \s: Whitespace [ \t\n\x0b\r\f]
- \S: Any non-whitespace [^\s]
- \w: Any word character [a-zA-Z_0-9]
- \W: Any non-word character
- \b: Word boundary
- \\: Backslash - \escapes next character
- Remember \has meaning in Java Strings too, so double them all up! \\\\to match \.

## Character Classes

- \d: Any digit [0 – 9]
- \D: Any non-digit [^0-9]
- \s: Whitespace [ \t\n\x0b\r\f]
- \S: Any non-whitespace [^\s]
- \w: Any word character [a-zA-Z_0-9]
- \W: Any non-word character
- \b: Word boundary
- \\: Backslash - \escapes next character
- Remember \has meaning in Java Strings too, so double them all up! \\\\to match \.

## Java Regex Methods

- .match(“regex”) Return true if this string matches “regex”
- .split(“regex”) Chop the String into an array of Strings at each regex
- .replaceFirst(“regex”,“replace”)
- .replaceAll(“regex”,“replace”)

## Java Regex Methods

- .match("regex") Return true if this string matches "regex"
- .split("regex") Chop the String into an array of Strings at each regex
- .replaceFirst("regex","replace")
- .replaceAll("regex","replace")

## Java Regex Methods

- .match(“regex”) Return true if this string matches “regex”
- .split(“regex”) Chop the String into an array of Strings at each regex
- .replaceFirst(“regex”,“replace”)
- .replaceAll(“regex”,“replace”)

## Java Regex Methods

- .match("regex") Return true if this string matches "regex"
- .split("regex") Chop the String into an array of Strings at each regex
- .replaceFirst("regex","replace")
- .replaceAll("regex","replace")

## Java Regex Backreferences

- Wrapping part of the regex in brackets ‘captures’ it
- Refer to each capture group with \1 (or 2 etc)
- Either in matching, or replacement
- So, ([0-9]{4})\1 would match 12341234 but not 12345678

## Java Regex Backreferences

- Wrapping part of the regex in brackets ‘captures’ it
- Refer to each capture group with \1 (or 2 etc)
- Either in matching, or replacement
- So, ([0-9]{4})\1 would match 12341234 but not 12345678

## Java Regex Backreferences

- Wrapping part of the regex in brackets ‘captures’ it
- Refer to each capture group with \1 (or 2 etc)
- Either in matching, or replacement
- So, ([0-9]{4})\1 would match 12341234 but not 12345678

## Java Regex Backreferences

- Wrapping part of the regex in brackets ‘captures’ it
- Refer to each capture group with \1 (or 2 etc)
- Either in matching, or replacement
- So, ([0-9]{4})\1 would match 12341234 but not 12345678

## Mode Prefixes

At start of string:

- (?i) Case insensitive
- (?s) Single line mode, so . matches line breaks
- (?m) Multi-line mode, so ^and $ match each line within a string

## Mode Prefixes

At start of string:

- (?i) Case insensitive
- (?s) Single line mode, so . matches line breaks
- (?m) Multi-line mode, so ^and $ match each line within a string

## Mode Prefixes

At start of string:

- (?i) Case insensitive
- (?s) Single line mode, so . matches line breaks
- (?m) Multi-line mode, so ^and $ match each line within a string

## Regular Expressions: Summary

- Simple multi-platform way to match and manipulate strings
- Present in almost every language you use
- Slight variations: //i versus (?i)

## Regular Expressions: Summary

- Simple multi-platform way to match and manipulate strings
- Present in almost every language you use
- Slight variations: //i versus (?i)

## Regular Expressions: Summary

- Simple multi-platform way to match and manipulate strings
- Present in almost every language you use
- Slight variations: //i versus (?i)

## Java SQL

- JDBC – Java DataBase Connectivity
- MySQL Connector
  `http://dev.mysql.com/downloads/connector/j/`

## Java SQL

- JDBC – Java DataBase Connectivity
- MySQL Connector
  `http://dev.mysql.com/downloads/connector/j/`

## Java SQL Example

```
Connection con;
try {
        String url =
            "jdbc:mysql://lochnagar.abertay.ac.uk"
                +":3306/sql12345678";
        con = DriverManager.getConnection(url,
            "sql12345678", "a6b5c4d3");
        Statement st=con.createStatement();
        ResultSet rs=st.executeQuery("SELECT
            VERSION()");
        while(rs.next()) {
            System.out.println(rs.getString(1)); }
} catch (SQLException e) {
        System.err.println(e);
}
```

## Lab Task Week 9

1. Write and test a regular expression to validate staff and student IDs from Java
2. Retrieve some table data from lochnagar with SQL in Java

## Lab Task Week 9

1. Write and test a regular expression to validate staff and student IDs from Java
2. Retrieve some table data from lochnagar with SQL in Java