Penerapan Regular Expression pada Konversi *Case Style*

Christian Tobing (13519109)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519109@stei.itb.ac.id

***Abstract*— Terdapat beberapa konvensi dalam menulis kode program sesuai dengan bahasa pemrograman maupun eksosistem pemgrogram sendiri. Salah satu konvensi tersebut adalah case style. Penting untuk mengetahui case style yang akan digunakan dalam suatu proyek untuk memudahkan programmer lain yang akan melanjutkan proyek tersebut. Makalah ini akan menjelaskan cara mengubah case style suatu kode program tertentu dari snake case menjadi camel case dan sebaliknya. Metode yang akan digunakan adalah regular expression karena fleksibilitas dan kemudahan dalam melakukan pengelompokan beberapa karakter yang mempunyai karakteristik yang mirip. (*Abstract*)**

Keywords—regular expression, case style, snake case, camel case, string matching, capturing group

I. PENDAULUAN

Dalam dunia pemrograman, salah satu cara yang baik untuk belajar dan mengerti sebuah konsep adalah mencoba untuk mengimplementasikan konsep tersebut secara langsung kedalam sebuah proyek. Bentuk implementasi dari konsep tersebut bisa berupa banyak hal, salah satunya adalah menulis kode program. Kode program yang baik mengikuti standar industri dan *best practice* yang berlaku saat itu. Salah satu *best practice* yang akan dibahas mengenai cara terbaik dalam melakukan penamaan pada variable, nama fungsi, dll.

Setiap bahasa pemgrograman mempunyai ekosistem tersendiri, dan biasanya mempunyai konvensi penamaan yang berbeda pula. Mengerjakan suatu proyek kecil sampai besar sekalipun harus mengikuti standar yang ada. Setiap proyek yang dikerjakan selalu bersinggungan dengan 3 entitas yaitu komputer, programmer itu sendiri dan *maintainer* atau *programmer* lain yang akan memelihara, memeriksa, atau melanjutkan hal tersebut.

Dilihat dari entitas pertama, yaitu komputer. Komputer tidak memedulikan *style* yang digunakan programmer saat menulis kode program. Entitas kedua yaitu programmer sendiri. Mungkin saja *programmer* itu sendiri akan selalu mengerti apa yang ditulis, tapi tidak jarang hal sebaliknya terjadi. Seiring bertambahnya baris kode program, menjadi semakin sulit untuk mengerti kode yang ditulis sebelumnya. Selain karena perbedaan *style* dengan *best practice* yang ada. Sehingga tidak ada padanan khusus dalam melakukan *debugging* pada program yang telah ditulis sebelumnya.

Entitas terakhir adalah sebagai seorang *maintainer*. Pekerjaan seorang *maintainer* dalam sebuah proyek dapat menjadi sulit atau mudah tergantung skill individu dan kecepatan dalam mengerti kode program. Namun tidak peduli pengalaman yang dimiliki oleh seorang *maintainer*, akan sangat sulit untuk mengerti kode program yang tidak mengikuti *naming convention* dan *best practice* yang ada. Hal tersebut dapat diperparah jika untuk sebuah grup yang sama (contoh : penamaan fungsi). Menggunakan style yang jauh berbeda, padahal fungsi yang dibedakan tersebut memang dimaksudkan untuk melakukan hal yang mirip.

Walaupun code editor dewasa ini sudah sangat baik, dan mempunyai fitur lengkap untuk membantu orang mengerti sebuah kode program dengan cepat . Namun, kode yang baik seharusnya dapat mudah dimengerti tanpa menggunakan terlalu banyak bantuan.

Menolak untuk membuat kode program yang baik dan mudah dimengerti merupakan sebuah tindakan yang tidak menghormati orang lain yang akan melanjutkna kode program tersebut. Membuat kode yang terorganisi dengan nama variabel yang berarti dan mudah dimengerti, juga komentar merupakan bentuk kepedulian kita terhadap individu atau kelompok programmer yang akan bekerja Bersama..

II. DASAR TEORI

A. *Regular Expression*

Regular expression (biasanya disingkat menjadi *regex)* merupakan sebuah ekspresi yang berupa string. String tersebut dapat mewakilkan sebuah pola yang membantu untuk mencocokkan, menemukan, dan mengatur sebuah teks. Bisa dibilang regex merupakan salah satu metode yang dapat digunakan untuk melakukan *string matching.* String matching pada menggunakan *regex* merupakan metode yang cukup efektif jika kita ingin memodifikasi atau menemukan kumpulan string yang berbeda namun memiliki karakteristik yang sa,ma. *Regex* menawarkan hal yang berbeda karena dapat melakukan *group capture*. Sehingga kelompok string yang dicari tidak harus selalu *exact match.*

Regex sangat cocok digunakan ketika bekerja dengan teks dan membutuhkan parsing, memodifikasi kelompok string, mencocokkan beberapa string dengan karakteristik sama, melakukan validasi terhadap string masukan, mendeteksi anomali pada suatu teks, dan banyak hal lainnya.

Regex mempunyai beberapa kemampuan untuk melakukan logika kondisional, rekursif, penyimpanan nilai, dll. Implementasi dari regex adalah berupa karakter-karakter khusus yang dapat menentukan kelompok string dengan karakteristik yang diwakilkan. Sintaks *regex* dengan karakter khusus biasa disebut sebagai *pattern.* Dibawah ini merupakan beberapa aturan pada regex.

1. Character Classes

Pattern untuk membedakan jenis jenis karakter

Pattern	Arti
.	Semua karakter kecuali newline
\d	Semua digit (*Arabic numeral*), ekuivalen dengan [0-9]
\D	Semua karakter non-digit(*Arabic numeral*)
\w	Semua karakter alfanumerik, ekuivalen dengan [A-Za-z0-9_]
\W	Semua karakter yang bukan karakter pada kata dari alfabet Latin.
\s	Semua karakter single ‘white space’
\S	Semua karakter selain ‘white space’
\t	Semua tab horizontal
\r	*Carriage return*
\n	Newline
\v	*Vertical tab*
\0	Karakter NULL

Tabel 2.1 Character Classes

2. Quantifiers

Quantifiers mengindikasikan jumlah karakter yang akan dicocokkan.

Pattern	Arti	Contoh
.*	Jumlah kejadian lebih besar sama dengan 0.	\w*
+	Jumlah kejadian lebih besar sama dengan 1.	\w+
?	Jumlah kejadian 0 kali atau 1 kali.	\w?
{n}	Jumlah kejadian n kali	\w{5}
{n,}	Jumlah kejadian paling sedikit n kali	\w{2.}
{n.m}	Jumlah kejadian dari n kali hingga m kali	\w{2.5}
*?	Jumlah kejadian 0 kali atau 1 kali sesedikit mungkin	\w?
+?	Jumlah kejadian lebih besar sama dengan 1 kali sesedikit mungkin	\+?

Tabel 2.2. Tabel Quantifiers

3. Grouping

Mengindikasikan pengelompokan karakter yang mempunyai pattern yang sama.

Karakter	Arti	Contoh
x|y	Mencocokkan x atau y	abc|def
[xyz]	Mencocokkan salah satu antara x, y, atau z	[\wAc]
[x-z]	Mencocokkan salah satu antara x hingga z	[1-9a-zA-Z]
[^xyz]	Mencocokkan karakter apapun kecuali x, y, atau z	[^\wAc]
[^x-z]	Mencocokkan karakter apapun kecuali antara x hingga z	[^1-9a-zA-Z]
(xyz)	Mencocokkan xyz keseluruhan atau tidak sama sekali kemudian mengingatnya	(halo)
(?:xyz)	Mencocokkan xyz keseluruhan atau tidak sama sekali tapi tidak mengingatnya	(?:halo)

Tabel 2.3. Grouping

B. Capturing Group

Capturing group merupakan suatu metode yang memperlakukan beberapa karakter sebagai unit tunggal. Hal tersebut dilakukan dengan menempatkan karakter yang dikelompokkan di dalam tanda kurung.

Misalkan, kode program di bawah ini yang bertugas untuk mendeteksi kata yang terdapat pengulangan menggunakan capturing group regular expression. Jika menggunakan metode string matching lainnya, hal ini tidak mungkin dilakukan, karena jika teks yang akan di cek cukup Panjang, tidak memungkinkan untuk mendapatkan semua *exact match* dari kata yang berulang.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
   public static void Main()
   {
      string pattern = @"(\w+)\s(\1)\W";
      string input = "He said that that was
the the correct answer.";
      foreach (Match match in
Regex.Matches(input, pattern,
RegexOptions.IgnoreCase))
         Console.WriteLine("Duplicate '{0}'
found at positions {1} and {2}.",

match.Groups[1].Value, match.Groups[1].Index,
match.Groups[2].Index);
   }
}
// The example displays the following
output:
//       Duplicate 'that' found at
positions 8 and 13.
//       Duplicate 'the' found at
positions 22 and 26
```

Regex pattern dari contoh di atas adalah : "(\w+)\s(\1)\W"

Penjelasan :

1. (\w+), Cocok dengan 1 atau lebih karakter kata. Ini adalah capturing group yang pertama..

2. \s, Cocok dengan karakter ‘white-space’

3. (\1), Cocok dengan capturing grup yang pertama, ini adalah capturing grup yang kedua

4. \W, Cocok dengan *non-word character*, termasuk ‘white-space’ dan semua pungtuasi.

C. *Case Style*

Saat bekerja dengan computer, programmer selalu dihadapkan dengan penamaan. Hal ini sangat penting karena penamaan yang konsisten dapat membawa proyek menuju kesuksesan. Setiap bahasa pemgrograman mempunyai standar penamaan masing-masing. Standar penamaan tersebut merupakan *case style*, dimana *style* dalam penamaan suatu hal dapat membantu orang yang bekerja dalam grup mengerti maksud dari kode program yang dibuat bersama- sama. Beberapa *case style* sebagai berikut :

1. Camel Case

Case ini mempunyai spesifikasi sebagai berikut : (1) kata pertama diawali dengan *lowercase* (2) Setiap kata dari *subsequent* berikutnya digabung (tanpa whitespace apapun) dan huruf pertamanya adalah kapital, contoh : camelCase

2. Snake Case

Snake case mempunyai spesifikasi sebagai berikut : (1) mengganti seluruh spasi dengan karakter ‘_’ dan setiap kata yang dipisahkan oleh karakter ‘_’ menjadi lowercase . contoh : snake_case_var

3. Kebab Case

Kebab Case mempunyai spesifikasi : Mengganti seluruh spasi dengan karakter ‘-‘ dan membuat semua kata menjadi *lower case*, contoh : kebab-case-var

4. Pascal Case

Pascal Case mempunyai spesifikasi yang mirip dengan Camel Case hanya saja huruf pertama subsequent pertama dari string menjadi huruf kapital, contoh : PascalCase

5. Upper Case Snake Case

Upper Case Snake Case mengganti seluruh spasi pada string dengan karakter ‘_’ dan mengganti semua huruf menjadi upper case, contoh : ‘UPPER_CASE_SNAKE_CASE’

III. PEMBAHASAN

Pattern regex yang akan dijelaskan adalah seputar find and replace *case style* tertentu menjadi *case style* yang lain.

Langkah umum dalam mengganti case adalah sebagai berikut :

- Menentukan regex yang berperan sebagai *capturing group* yang akan mengelompokkan kata dengan *case style* yang ingin diubah
- Meletakkan pattern *regular expression* ke dalam kolom 'find'.
- Meletakkan pattern regular expression *case style* tujuan di kolom replace

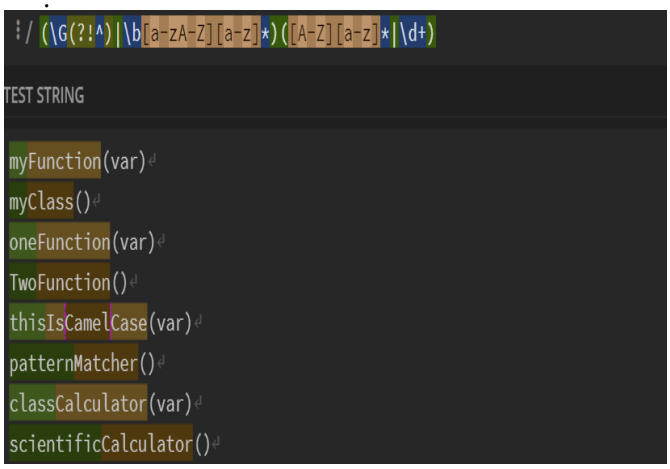
A. *Mengubah Camel Case menjadi Snake Case*

Contoh pertama adalah mengubah kode program yang menggunakan penamaan dengan style camelCase menjadi snake_case style. Berikut dilampirkan proses penggantian *case style* tersebut.

Contoh Camel Case yang akan diubah :

```
myFunction(var)
myClass()
oneFunction(var)
TwoFunction()
thisIsCamelCase(var)
patternMatcher()
classCalculator(var)
scientificCalculator()
```

.



Regex yang digunakan adalah :

(\G(?!^)|\b[a-zA-Z][a-z]*)([A-Z][a-z]*|\d+)

Penjelasan :

Membagi menjadi 2 group capturing sebagai berikut:

Group 1 Capturing :

(\G(?!^)|\b[a-zA-Z][a-z]*)

Keterangan :

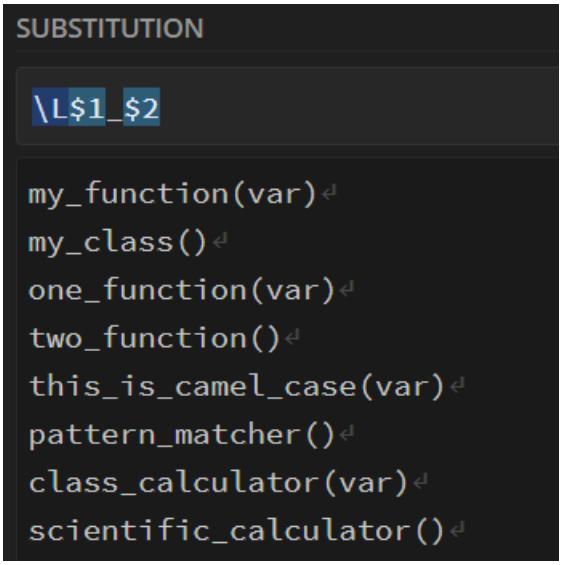
- \b[a-z]+ , bagian ini mengambil awal dari string yang diawali dengan lowercase dan diikuti satu atau lebih *lowercase ASCII* .
- '|', logika 'or' (atau)
- \G(?!^) , posisi akhir dari kecocokan yang sama pada bagian sebelumnya

Group 2 Capturing :

((?:[A-Z]|\d+)[a-z]*)

Keteranngan :

- (?:[A-Z]|\d+), Mencocokkan antara *uppercase* ASCII ([A-Z]) atau 1+ (satu atau lebih) digit (\d+)
- [a-z]* , 0 atau lebih *lowercase* ASCII



Lalu pada bagian substitusi, regex yang digunakan adalah :

\L$1_$2

Keterangan :

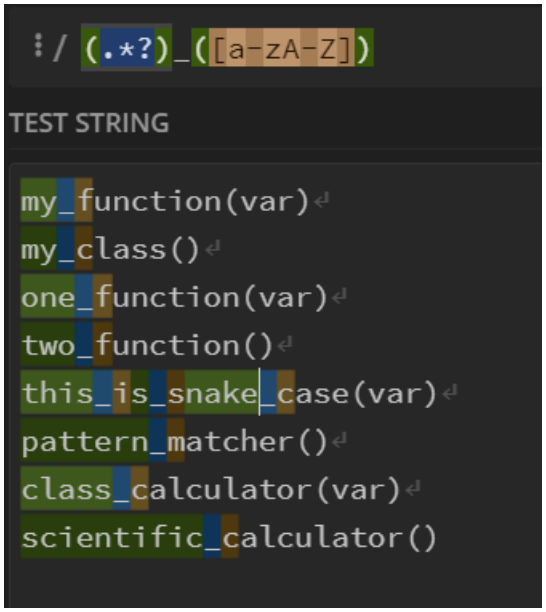
- \L, *case modifier* unutuk mengubah *subsequent* selanjutnya menjadi huruf kecil
- $1 dan $2 menandakan string yang cocok dengan group capturing 1 dan group capturing 2
- '_', Memberikan karakter '_' di antara group 1 dan group 2.

B. *Mengubah Snake Case Menjadi Camel Case*

Contoh kedua adalah mengubah Snake Case mejadi Camel Case. Berikut dilampirkan proses mengubah penamaan tersebut

Daftar kata yang akan diubah :

my_function(var)
my_class()
one_function(var)
two_function()
this_is_snake_case(var)
pattern_matcher()
class_calculator(var)
scientific_calculator()



Regex yang digunakan adalah :
(.*?)_([a-zA-Z])

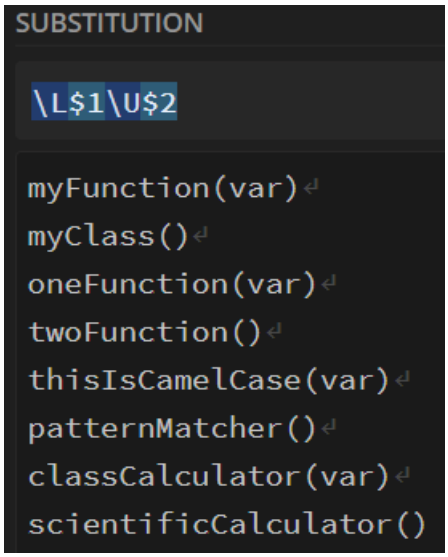
Penjelasan :

Group 1 Capturing : (.*?)

Keterangan :
- '.' , Mencari kecocokan dengan karakter apapun
- '*', Jumlah kecocokan lebih dari sama dengan 0
- '?', *lazy quantifier* berarti mencari kecocokan sesedikit mungkin
- '_', mencocokkan dengan karakter '_'

Group 2 Capturing :

- [a-zA-Z], mencari karakter pertama setelah karakter '_'



Regex yang digunakan adalah :
\L$1\U$2

Keterangan :

- \L, *case modifier* unutuk mengubah *subsequent* selanjutnya menjadi *lower case*
- \U, *case modifier* unutuk mengubah *subsequent* selanjutnya menjadi *uppercase*
- $1 dan $2 menandakan string yang cocok dengan group capturing 1 dan group capturing 2

V. KESIMPULAN

Program untuk konversi *case style* dapat diimplementasikan menggunakan *regular expression*. Dalam prosesnya *regular expression* dapat mengenali pola dari setiap *case style* yang telah dibahas sebelumnya (snake_case dan camelCase) dengan memanfaatkan group capturing. Sehingga setiap grup yang dikenali oleh *regular expression* dapat diproses lagi untuk diubah ke *case style* yang dituju.

Penulis menyadari ada beberapa modifikasi dari *case style* yang tidak dapat dikenali, sehingga diharapkan pengguna dapat menyesuaikan masukan dengan bentuk standar agar dapat dikenali oleh pattern *regex* ada di makalah ini.

REFERENCES

[1] https://www.tutorialspoint.com/javaregex/javaregex_capturing_groups.htm

[2] https://stackoverflow.com/questions/127916/is-programming-style-important-how-importantI.S. Jacobs and C.P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G.T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271-350.

[3] https://www.computerhope.com/jargon/r/regex.htm

[4] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions/Character_Classes

[5] https://docs.microsoft.com/en-us/dotnet/standard/base-types/grouping-constructs-in-regular-expressions

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Jakarta, 11 Mei 2021

Christian Tobing, 13519109