Penerapan *Pattern Matching* pada Fitur *Renaming Refactor* di IDE *NetBeans*

Rezan Achmad / 13508104
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if18104@students.if.itb.ac.id, rezanachmad@yahoo.co.id

***Abstrak*. Sekarang ini telah banyak perangkat lunak yang dibuat untuk membantu para pengembang untuk membangut perangkat lunak lain agar pembuatan lebih cepat dan mudah. Perangkat lunak ini biasa disebut IDE (*Integrated Development System*). IDE menyediakan beberapa fitur yang mempermudah dalam *coding* / mengembangkan aplikasi. Salah satu fiturnya yaitu *renaming refactor*. Ketika seorang pengembang ini mengubah nama variabel yang telah dia deklarasikan, *renaming refactor* adalah fitur yang tepat untuk digunakan. Dengan renaming refactor, pengembang tersebut tidak perlu mengubah variabel yang ingin diganti secara manual satu persatu tetapi secara otomatis dengn menggunanakn renaming refactor. *Renaming refactor* menggunakan prinsip *pattern matching* untuk mencari nama variabel yang akan diubah. Ada beberapa jenis *pattern matching* diantaranya yaitu *brute force* dan *Boooyer-More*.**

Kata Kunci : *Booyer-Moore, IDE ,pattern matching, renaming refactor.*

I. PENDAHULUAN

Saat sekarang ini sudah banyak perangkat lunak yang mempermudah para pengembangan (*developer*) untuk membuat aplikasi. Perangkat-perangkat lunak ini sangat diperlukan karena semakin banyak elemen kehidupan yang membutuhkan perangkat lunak yang kompleks. Tidak sekedar kompleks, para konsumen / client banyak menuntut aplikasi diselesaikan dalam jangka watu yang singkat. Oleh karena itu, beberapa orang membuat suatu perangkat lunak yang berfungsi untuk membantu membuat perangkat lunak lainnya.

Sangat banyak jenis perangkat lunak yang telah diciptakan untuk mempemudah membuat perangkat lunak. Fitur – fitur yang disediakan pun beragaman, tergantung kompleksitas perangakt lunak tersebut. Ada yang hanya sekedar memberi warna-warna pada kode, memberi *hint* saat mengetik variabel, tipe data, fungsi / prosedur ataupun fungsi, menyediakan fitur *drag and drop*, generate UML ke fungsi-fungsi serta kelas-kelas dan sebaliknya, menyediakan fitur sub-version, *rename* fungsi, kelas atau variabel dan masih banyak lagi.

Perangkat-perangkat lunak tersebut banyak yang bisa diunduh secara gratis di dunia maya tetapi ada pula yang berbayar. Beberapa merek perangkat lunak yang disediakan secara gratis yaitu Notepad++, Geany, NetBeans, Eclipse dan lain-lain. Sedangkan yang berbayar yaitu Visual Studio dari Microsoft, C++ Builder dari Embacaredo, Adobe Dreamwave dari Adobe dan masih banyak lagi.

Netbeans, Eclipse, Visual Studio dan sejenisnya biasa disebut IDE (*Intregated Develompent Environment*). IDE menyediakan "lingkungan" bagi pengembang untuk mengembangkan atau membangun perangkat lunak. "Lingkungan" tersebut berisi fitur-fitur tertentu sehingga mempermudah pengembangan membuat perangkat lunak yang sebelumnya telah disebutkan. Selain itu, IDE tidak hanya memfasilitasi satu bahasa tetapi banyak bahasa. Sebagai contoh, NetBeans memfasilitasi berbagai bahasa seperti Java, C++, C, PHP, JavaScript, JSP dan masih banyak lagi.

Salah satu fitur yang cukup menarik pada IDE yaitu *refactor* pada IDE NetBeans. *Refactor* memiliki banyak bagian-bagian fitur salah satunya *rename* variabel. *Rename variabel* memungkinkan pengembang untuk mengubah suatu variabel. Keunggulan dari *refactor* ini yaitu dapat me-*rename* semua *variabel* yang dimaksud di dalam *source code*. Fitur ini mirip dengan fitur replace pada perangkat lunak *Microfost Word*.

Mekanisme penggantian nama variabel tersebut salah satunya memakai algoritma *pattern matching*. Algoritma ini digunakan untuk mencari string yang terdapat suatu *source code*. Setelah ditemukan, *string* tersebut akan ditimpa dengan *string* yang baru.

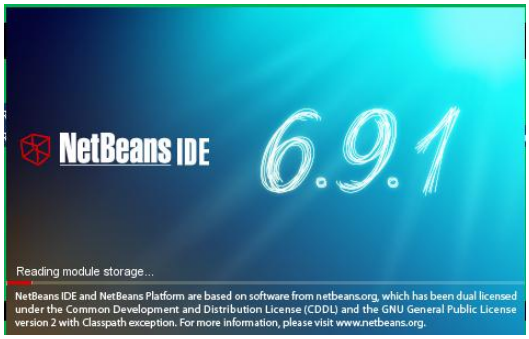
II. FITUR *REFACTOR* PADA *NETBEANS*

Sebelum membahas lebih lanjut fitur refacor pada NetBeans, terlebih dahulu akan dijelaskan sekilas tentang NetBeans.

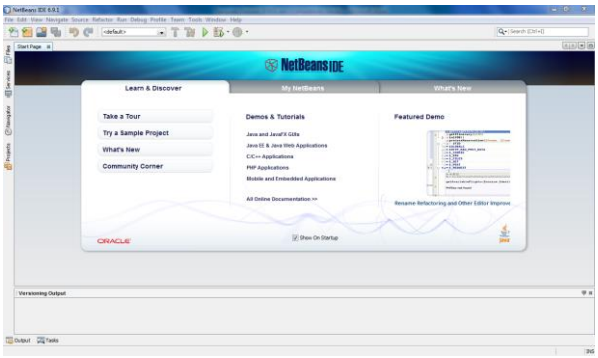
Tentang *NetBeans*

NetBeans merupakan salah satu IDE yang banyak digunakan oleh pengembang perangkat lunak. Netbenas merupakan IDE yang gratis sehingga bisa digunakan

oleh siapa saja. *NetBeans* dibuat oleh Sun Microsystem yang sekarang telah dibeli Oracle. *NetBeans* awalnya dikhususkan untuk menunjang pemakaian bahasa Java namun sekarang telah bisa menunjang beberapa bahasa yang sering digunakan untuk membangun perangkat lunak.



Gambar 1 Tampilan Loading NetBeans 6.9.1



Gambar 2 Halaman Awal NetBeans 6.9.1

Refactor pada NetBeans

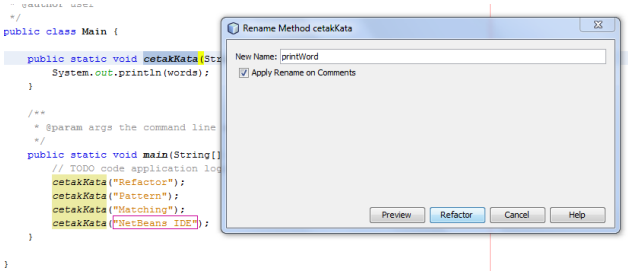
IDE NetBeans memiliki banyak fitur salah satunya yaitu refactor. Seperti yang disebutkan di halaman ini http://wiki.netbeans.org/Refactoring#Refactoring_Simplified, *refactor* adalah teknik untuk mengubah struktur kode tanpa mengubah perilaku dari kode tersebut. Yang termasuk struktur kode seperti nama variabel, fungsi / prosedur serta enkapsulasi atribut.

Refactor pun punya beberapa macam sub-fitur. Sub-sub fitur yang dimaksud yaitu :

1. *Rename*
   Fitur ini bisa digunakan untuk mengubah nama variabel atau fungsi / prosedur.
2. *Encapsulate Field*
   Fitur ini digunakan untuk membuat operasi *get()* atau *set()* terhadapt atribut kelas.
3. *Introduce Methode*
   Fitur ini digunakan untuk menambah fungsi/prodesur baru. Sebenarnya penambahan fungsi / prosedur bisa dilakukan secara manual namun, dengan menggunakan fitur ini, header fungsi akan lebih rapi terlihat. Selain itu, dengan fitur ini, pengembang bisa membuat sekumpulan baris kode dibungkus oleh sebuah fungsi / prosedur.

Pada makalah ini, fokus pembahasan hanya untuk *Rename Factoring*. Fitur ini cukup membantu pengembang untuk mengubah suatu nama variabel atau operasi. Bayangkan jika terdapat suatu operasi yang sering dipanggil dalam kode program. Ketika pengembangn ingin mengubah nama fungsi tersebut, betapa repotnya si pengembang tersbut mencari fungsi yang dimaksud kemudian mengganti nama fungsinya. Sebenarnya bisa menggunakan fitur replace tetapi replace tidak melakukan pencarian secara exact mathcing. Oleh karena itu *renaming refactor* diperlukan untuk mengatasi masalah tersebut.

Di bawah ini merupakan contoh penggunaan *renaming refactor* pada *NetBeans*.



Gambar 3 Sebelum Rename Refactor

Misalkan dibuat operasi *cetakKata* yang berfungsi untuk menampilkan tulisan di layar *command prompt*. Suatu saat pengembang ingin mengubah nama operasi tersebut menjadi *printWord*. Cara mengubah nama operasi ersebut yaitu :

1. Letakkan kursor pada nana fungsi
2. Tekan *CTRL-R* atau klik kanan pilih *Refactor* → *Rename*
3. Akan muncul jendela untuk memasukkan nama operasi yang baru
4. Klik OK.

```
public class Main {

    public static void printWord(String words) {
        System.out.println(words);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        printWord("Refactor");
        printWord("Pattern");
        printWord("Matching");
        printWord("NetBeans IDE");
    }

}
```

Gambar 4 Setelah Rename Refactor

III. *PATTERN MATCHING*

Pattern Matching adalah suatu algoritma yang berfungsi untuk mencari suatu *pattern* yang terdapat pada suatu teks. Misalnya terdapat teks T : "Halo selamat siang. Halo Dunia" dan pattern P: "siang". Yang dilakukan *Pattern Matching* yaitu bagaimana mencari *pattern* P didalam teks T.

Ada berbagai macam jenis algoritma yang bisa dipakai untuk menemukan *pattern* P didalam T. Beberapa algoritma yang dipakai pada pattern matching yaitu *brute force*, *Knut Morris Prat* (KMP) dan *Booyer-Moore*. Pada makalah ini yang akan dijelaskan hanya algoritma *Bruteforce* dan *Booyer-Moore*.

Bruteforce

Algoritma ini adalah algoritma yang paling mendasar dalam pemograman. Ciri khas dari bruteforce yaitu mencoba semua kemungkinan yang ada. Pada *pattern matching* ilustrasi algoritma bruteforce sebagai berikut.

Text :

A	R	I	O	R	I	T	H	M

Pattern :

R	I	T	H

Proses Perbandingan :

A	R	I	O	R	I	T	H	M

1								
R	I	T	H					

A	R	I	O	R	I	T	H	M

	2	3	4					
	R	I	T	H				

A	R	I	O	R	I	T	H	M
		5						
		R	I	T	H			

A	R	I	O	R	I	T	H	M

			6					
			R	I	T	H		

A	R	I	O	R	I	T	H	M

				7	8	9	10	
				R	I	T	H	

Proses algoritma bruteforce untuk perbandingan string bisa dilihat diatas. Algoritma ini membandingkan semua karakter yang terdapat pada *pattern* dan teks. Jumlah perbandingan yang butuhkan oleh algoritma bruteforce untuk mencari *pattern* "RITH" didalam teks "ARIORITHM" yaitu 10 perbandingan.

Algoritma *bruteforce* sering kali disebut algoritma yang tidak mangkus karena boros dalam menggunakan sumber daya. Sebagi contoh pada proses perbandingan ke 3. Jika ingin menghemat sumber daya, sebaiknya perbandingan dimulai pada karakter ke-5 teks karena sub-string "RIO" sudah jelas tidak terdapat pada pattern RITHM.

Pseudocode bruteforce adalah sebagai berikut :

```
integer brute(text : string, pattern : string)
  n       : integer
  m       : integer
  j       : integer
  i       : integer

  n       ← text.length()
  m       ← pattern.length()

  for (i ← 0, i <= (n-m), i ← i + 1)
    j ← 0

    while j < m and text[i+j] = pattern[j]
      j ← j + 1

    if (j = m)
      → i

  → -1
```

Boyer-Moore

Booyer-Moore adalah algoritma bruteforce yang telah diperbaiki sehingga tidak perlu membandingkan karakter-karakter tertentu.

Terdapat dua teknik yang terdapat pada algoritma *Booyer-Moore* yaitu :

1. *Teknik Looking-Glass*
   Mencari *pattern* pada teks dengan cara membandingkan karakter mulai dari belakang *pattern*.
2. Teknik *Character-Jump*.
   Ketika terjadi ketikdacocokan antara karakter P[j] dan karakter T[i], terdapat tiga kemungkinan cara yang dilakukan.

T

…	X	A	…

i

P

…	B	A	…

j

a. Jika di dalam P terdapat "X", geser P hingga "X" yang paling akhir pada P sejajar dengan "X" pada T.

T

		X	A

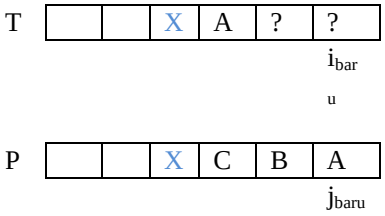
i

P

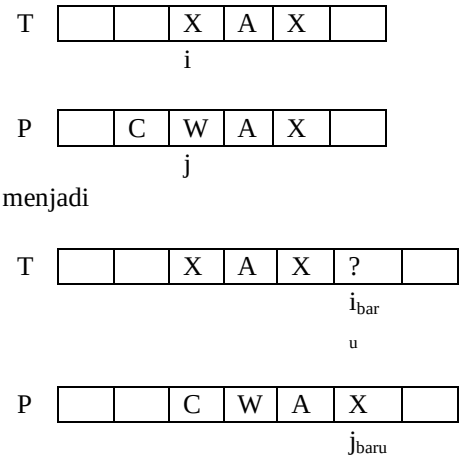
X	C	B	A

j

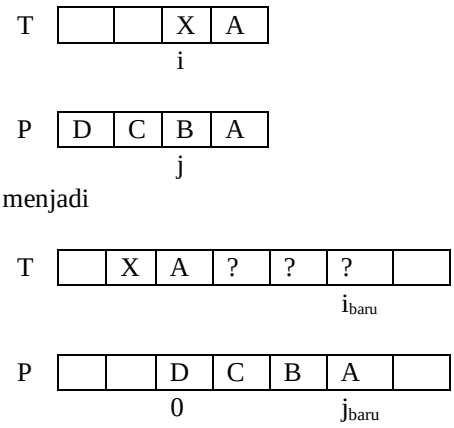
menjadi



b. Jika di dalam P terdapat "X", tetapi tidak memungkinkan untuk menggeser P ke kanan sehingga "X" pada P sejajar dengn "X" pada T, geser P satu karakter ke kanan.



c. Jika kasus a dan b tidak memenuhi, geser P hingga P[0] sejajar dengan T[i+1]



Untuk lebih jelasnya, lihat proses perbandingan karakter dengan menggunakan *Booyer Moore* di bawah ini.

Text :

A	R	I	O	R	I	T	H	M

Pattern :

R	I	T	H

Proses Perbandingan :

A	R	I	O	R	I	T	H	M

1

R	I	T	H					

A	R	I	O	R	I	T	H	M

5 4 3 2

				R	I	T	H	

Dengan menggunakan *Booyer-Moore* dibutuhkan lima perbandingan untuk menemukan *pattern* P didalam teks T. Hal ini membuktikan bahwa Booyer-Moore lebih mangkus dibandingkan dengan algoritma *bruteforce* yang membutuhkan sepuluh perbandingan untuk mendapatkan *pattern*.

IV. PENERAPAN *PATTERN MATCHING* PADA *RENAME REFACTOR*

Untuk melakukan *rename refactor* dibutuhkan sebuah algoritma *pattern matching* untuk menemukan nama fungsi / variabel yang akan diubah namanya. Pada pembahasan ini, algoritma *pattern matching* yang digunakan adalah algoritma *Booyer-Moore*.

Berikut pseudo code algritma *Booyer-Moore* :

```
integer BMMatch(text : string, pattern :
string pattern)
  last  : array of integer
  n     : integer
  m     : integer
  i     : integer
  j     : integer
  lo    : integer

  last  ← buildLast(pattern)
  n     ← text.length()
  m     ← pattern.length()
  i     ← m - 1

  if i > n-1
    → -1

  j ← m-1
  do
    if pattern[j] = text[i]
      if j = 0
        → i
      else
        i ← i - 1
        j ← j - 1
    else
      lo ← last[i]
      i ← i + m - min(j, lo + 1)
      j ← m - 1
  while i <= n-1

  → -1


array of integer buildLast(string pattern)

  last : array of integer [0..128]

  for (i ← 0, i < 128, i ← i +1)
```

```
    last[i] ← -1

  for (i ← 0, i < pattern.length(), i ← i
  + 1)
    last[pattern[i]] ← i

  → last
```

Pada algoritma diatas terdapat fungsi *buildLast*. Kegunanan fungsi ini yaitu untuk mengetahui kapan kemunculan terakhir dari suatu karakter di dalam pattern. Fungsi ini akan digunakan oleh *Booyer-Moore* untuk melakukan lompatan.

Algoritma *pseudo code* diatas hanya diperuntukan untuk pencocokan yang tidak *exact matching* serta hanya menampilkan kecocokan pertama saja. Sekarang algoritma tersebut akan diperbaiki agar bisa dipakai pada *rename refactor*.

```
integer BMMatch(text : string, pattern :
string pattern)
  last    : array of integer
  result  : array of integer
  n       : integer
  m       : integer
  i       : integer
  j       : integer
  lo      : integer

  last  ← buildLast(pattern)
  n     ← text.length()
  m     ← pattern.length()
  i     ← m - 1

  if i > n-1
    → result

  j ← m-1
  do
    if pattern[j] = text[i]
      if j = 0
        result.push(i)
        i ← i + m
        j ← j + m
      else
        i ← i - 1
        j ← j - 1
    else
      lo ← last[i]
      i ← i + m - min(j, lo + 1)
      j ← m - 1
  while i <= n-1

→ result
```

Tulisan merah merupakan kode tambahan agar ditemukan beberapa posisi. Posisi – posisi tersebut ditampung dalam sebuah *array of integer* dan akan dikembalikan di akhir fungsi.

Agar bisa dilakukan exact matching, sebelum memakai fungsi ini terlebih dahulu isi pattern diubah. Misalnya akan dicari fungsi *cetakKata*. *Pattern cetakKata* diubah menjadi "*cetakKata(*" (tidak termasuk tanda petik) dengan asumsi semua fungsi dalam kode program ditulis dengan gaya seperti itu.

Setelah posisi-posisi kemunculan ditemukan, langkah berikutnya yaitu mengganti *pattern-pattern* yang lama dengan yang baru. *Pseudo code* untuk melakukan pergantian (*replace*) adalah sebagai berikut.

```
string  replace(positions  :  array  of
integer, oldPattern : string, newPattern :
string, text : string)

  inc     : integer
  p       : integer
  oldL    : integer
  newL    : integer
  length  : integer
  result  : string

  inc     ← 0
  oldL    ← oldPattern.length()
  newL    ← newPattern.length()
  length  ← text.length()
  result  ← text

  foreach positions as p
    result ← result.substr(0, p + inc) +
    newPattern + result.substr(p + inc +
    oldL, length + inc)
    inc ← inc + newL - oldL

    → result
```

Sebelum memakai fungsi *replace* harus dipastikan bahwa *oldPattern* bernilai "*cetakKata(*" dan *newPattern* bernilai "*printWord(*".

Fungsi diatas mengganti setiap kemunculan "*cetakKata(*" dengan "*printWord(*". Pada fungsi replace terdapat variabel *inc*. Variabel ini digunakan menyesuaikan *p* dengan isi *text* yang baru.

V. KESIMPULAN

Renaming refactor merupakan salah satu fitur pada IDE yang biasa digunakan oleh pengembang untuk mengubah nama suatu variabel, fungsi atau prosedur. *Renaming refactor* ini menggunakan prinsip pattern matching yaitu mencari kecocokan pattern pada suatu teks. Setelah posisi-posisi pattern didapatkan, dilakukan renaming untuk mengganti pattern yang lama dengan patter yang baru. Algoritma pattern matching yang cukup mangkus yaitu *Booyer-Moore*. *Booyer-Moore* menggunkan prinsip *Looking-Glass* dan *Character Jump* sehingga mempercepat pencarian.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. *Dikat Kuliah IF3051 Strategi Algoritma*. ITB, 2009.

[2] http://www.informatika.org/~rinaldi/Stmik/2009-2010/PatternMatching.ppt

[3] http://wiki.netbeans.org/Refactoring#Refactoring_Simplified

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.

Bandung, 9 Desember 2010

Rezan Achmad / 13508104