**semanTIK**, Vol.2, No.1, Jan-Jun 2016, pp. 203-210
ISSN : 2502-8928 (Online)


# DETEKSI KEMIRIPAN *SOURCE CODE* PADA BAHASA PEMROGRAMAN *JAVA* MENGGUNAKAN METODE ANALISIS LEKSIKAL

**Holis Hermansyah[*1], Bambang Pramono[2], Subardin[3]**
[*1,2,3]Jurusan Teknik Informatika, Fakultas Teknik, Universitas Halu Oleo, Kendari
e-mail: [*1]**holishermansyah@gmail.com**, [2] bambangpramono09@gmail.com,
[3]mail.bardin@gmail.com

**Abstrak**

Laboratorium Pemrograman Jurusan Teknik Infomatika Fakultas Teknik Universitas Halu Oleo sebagai tempat pembelajaran bagi para mahasiswa Informatika yang mengikuti kelas pemrograman selalu memberikan praktikum sebagai salah satu media pengukur tingkat pemahaman mahasiswa dalam hal membuat program. Banyaknya *source code* yang harus diperiksa oleh Dosen atau Assisten laboratorium mengakibatkan sulitnya melakukan pemeriksaan serta sulitnya mengukur *kredibilitas* masing-masing tugas milik mahasiswa

Metode deteksi kemiripan kode menggunakan analisis leksikal dapat digunakan untuk merubah kode menjadi *token* unik. *Token* ini digunakan untuk mengidentifikasi struktur program yang diperiksa. Sebelum proses pencocokan *token* menggunakan *array*, terlebih dahulu *token* yang terbentuk dari kedua *source code* yang diperiksa di sorting atau diurutkan berdasarkan abjad. Setelah itu dilakukan proses pencocokan string menggunakan *array* per posisi *array*. Semakin besar kemiripan alur dan kode program maka semakin besar kemungkinan kode tersebut merupakan *plagiat*.

Nilai presentase kemiripan kode program yang dihasilkan dari proses pencocokan *token* ini yang nantinya digunakan sebagai acuan penentuan *plagiarisme* dengan ketentuan jika nilai presentase lebih besar atau sama dengan 70 % maka *source* dianggap *plagiat*, namun jika lebih kecil dari 70% tidak dianggap sebagai *plagiat*.

**Kata kunci—** *Source Code, Analisis leksikal,Java,Token,Plagiarism.*

***Abstract***

*Programming Laboratory of Informatics Engineering Department of Engineering Faculty, Halu Oleo University is a learning place for college students majoring in informatics to follow programming class that become one of the standard to determine the student's level of understanding in creating a program.*

*Similarity detection method using Lexical analysis can be used to convert the programming code into an unique token. This token later used to identify the structure of a program that is being examined. After that, the token that already formed from two different source codes will be sorted alphabetically before entering the string matching process by using arrays. The string matching process will do the matching based on the position of each array. The more the similarities of the structure and the program codes, the more likely the code represented as plagiarism.*

*The percentage value resulting from the token matching process later will be used as reference to determine the percentage of plagiarism in the program. If the similarities percentage greater than 70%, the examined source codes can be considered as plagiarism, but if it less than 70% the examined source codes will be not considered as plagiarism.*

***Keywords****— Source Code, Lexical Analysis, Java, Token, Plagiarism.*

## 1. PENDAHULUAN

Laboratorium Rekayasa Perangkat Lunak Jurusan Teknik Infomatika Fakultas Teknik Universitas Halu Oleo sebagai tempat pembelajaran bagi para mahasiswa informatika yang mengikuti kelas pemrograman selalu memberikan praktikum sebagai salah satu media pengukur tingkat pemahaman mahasiswa dalam hal membuat program. Pemrograman adalah proses menulis, menguji dan memperbaiki, dan memelihara kode yang membangun sebuah program komputer. Kode ini ditulis dalam berbagai bahasa pemrograman.

Bahasa pemrograman merupakan alat yang sangat penting bagi *programmer* untuk mengimplementasikan algoritma. Tiap bahasa pemrograman memiliki kelebihan dan kekurangan tersendiri, dan *programmer* memiliki referensi tersendiri dalam memilih suatu bahasa pemrograman. Beberapa faktor penting seseorang dalam memilih bahasa pemrograman adalah *syntax*, *editor*, dokumentasi, performa, *library*, fleksibilitas, komunitas dan popularitas.

Banyaknya *source code* yang harus diperiksa oleh Dosen atau Assisten Laboratorium mengakibatkan sulitnya melakukan pemeriksaan serta sulitnya mengukur kredibilitas masing-masing tugas milik mahasiswa. Kode program terperiksa yang memiliki tingkat *similarity* (kemiripan) yang cukup tinggi antar kode dapat dijadikan acuan adanya tindakan-tindakan kecurangan seperti melakukan tindakan *plagiat* kode terhadap tugas mahasiswa lain [1].

Metode deteksi kemiripan kode menggunakan Analisis Leksikal dapat digunakan untuk merubah kode menjadi *node* ataupun *token* unik masing-masing kode terperiksa. Semakin besar kemiripan maka semakin besar kemungkinan kode tersebut merupakan hasil *plagiat* [2]. Pada judul yang diajukan, sistem akan melakukan analisis leksikal pada program yang diperiksa dalam hal ini bahasa pemrograman *Java*. Analisis ini meliputi pengubahan *source code* kedalam bentuk *token-token* yang memiliki nama atau ID yang unik, penghapusan komentar,serta penghapusan *white space*. Kemudian dilakukan pencocokan *source code* yang telah menjadi *token* yang memiliki kesamaan pada program pembanding berdasarkan struktur yang terbentuk. Jika ditemukan ID *token* dan *token* yang sama pada posisi yang sama maka program akan mencatatnya dan kemudian menghitung tingkaat kemiripan *source code* tersebut dan memberikan nilai Presentase (*Similarity*).

## 2. METODE PENELITIAN

### 2.1. Plagiarisme

*Plagiarisme* merupakan tindakan kriminal yang sering terjadi dalam dunia akademis. *Plagiarisme* itu sendiri berasal dari kata latin "*Plagiarus*" yang berarti penculik dan "*Plagiare*" yang berarti mencuri. Jadi, secara sederhana *plagiat* berarti mengambil ide, kata-kata, dan kalimat seseorang dan memposisikannya sebagai hasil karyanya sendiri atau mengunakan ide, kata-kata, dan kalimat tanpa mencantumkan sumber dimana seorang penulis mengutipnya [3].

### 2.2. Analisis Leksikal

Analisis leksikal merupakan fungsi analisis dalam *compiler* yang bertugas mendekomposisi program sumber menjadi bagian-bagian kecil (*token*). Analisis leksikal atau *scanner* bertugas mengidentifikasi semua besaran pembangun bahasa (leksikal) yang ada pada *source code*. *Scanner* menerima masukan *source code* berupa serangkaian karakter kemudian memilah-milahnya menjadi bagian-bagian kecil yang mempunyai satu arti yang disebut *token*, seperti *konstanta*, nama variabel, *keyword*, *operator* [4].

### 2.3. Analisa Kebutuhan Sistem

1. Kebutuhan Data Masukan

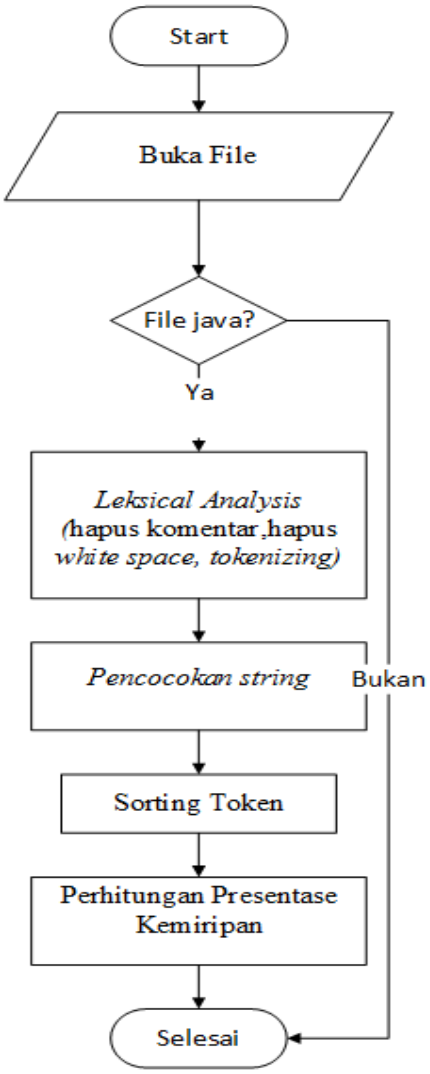
Untuk data masukan yang dibutuhkan dari sistem ini adalah adanya *file class Java* yang akan dihitung *presentase* kemiripannya minimal dua buah *file class*, selanjutnya sistem akan menggunakan metode analisis leksikal dan pencocokan string untuk menghasilkan *output*.

2. Kebutuhan Data Keluaran

Data keluaran dari sistem yang telah di proses untuk kemudian ditampilkan kepada pengguna sistem yaitu presentase nilai kemiripan dan kesimpulan dari hasil presentase apakah *file* tersebut merupakan *plagiat* atau bukan.

2.4. Arsitektur

*Flowchart* dari proses sistem yang dilakukan secara keseluruhan dapat ditunjukkan oleh Gambar 1.




Gambar 1 *Flowchart* program

2.5. Ilustrasi Metode Analisis Leksikal Terhadap Perencanaan Aplikasi

a. Analisis Leksikal

Tahap ini merupakan tahap awal sebelum dilakukan perhitungan kemiripan kedua *source code* yang akan diperiksa. Setelah *source code* dimasukan maka sistem otomatis akan mentransformasikan *source code* tersebut kedalam bentuk *token* berdasarkan ID masing-masing tipe.

b. Sorting

Proses ini merupakan proses mengurutkan *token-token* yang terbentuk dari kedua *source code* yang diperiksa menurut abjad. Proses ini dilakukan dengan tujuan supaya jika terdapat *source code* yang sama tetapi terdapat fungsi atau prosedur yang sama pada posisi yang berbeda tetap akan terdeteksi.

c. Perhitungan Nilai Kemiripan

Setelah kedua *source code* berbentuk *token* dan *token-token* telah terurut, maka proses selanjutnya adalah perhitungan nilai kemiripan kedua *source* menggunakan *array*. Panjang *array* sesuai dengan panjang *token* yang terbentuk.

Metode perhitungan kemiripan ini didasarkan atas posisi dan isi dari *token* tersebut. Jika pada posisi *array* ke-0 di cocokan dengan posisi *array* ke-0 pada *source* kedua sama, baik ID *token* dan isi dari *token* maka program akan mencatat sebagi satu kemiripan. Kemdian program akan melanjutkan pada posisi *array* selanjutnya sampai posisi terakhir pada *array*.

Jika panjang *array* pada kedua *source* berbeda dikarenakan banyaknya *token* yang terbentuk pada kedua *source* berbeda, maka pencocokan diberhentikan pada nilai *array* yang terpendek dan sisanya dianggap sebagai ketidaksamaan.

Menentukan nilai kemiripan menggunakan Persamaan (1).

$$similarity = \left(\frac{\sum kesamaan}{\sum elemen\_objek}\right) x\ 100\ \% \quad (1)$$

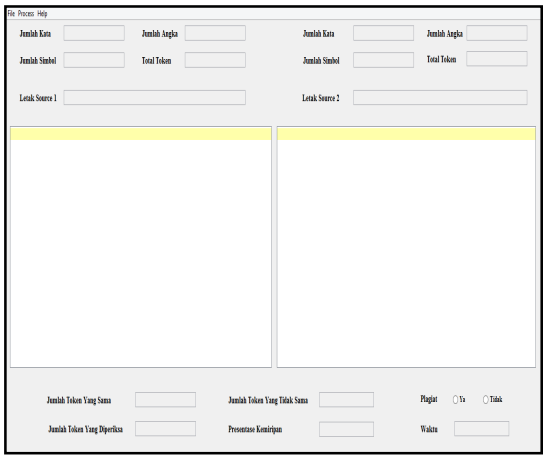
## 3. HASIL DAN PEMBAHASAN

3.1 Perancangan Interface

Perancangan antar muka (*interface*) merupakan perancangan untuk melihat desain awal dari sebuah sistem. Gambar 2 menunjukan perancangan sistem *interface* dari peracangan aplikasi pendeteksi kemiripan *source code*.

Gambar 2 merupakan rancangan desain dari halaman utama aplikasi yang terdiri dari tiga meu, yaitu: *file*, process dan help. Pada menu *File* terdiri submenu *Open Source* 1 dan *Open Source* 2 yang berfungsi untuk menginsert *file* dan *exit*. Pada menu Process terdiri submenu *Scanning* yang berfungsi untuk mencocokan kedua *file Source code*

yang akan dibandingkan, *Show* Analisis untuk menampilkan hasil analisis, *Show Similar* untuk menampilkan *token-token* yang mempunyai kemiripan, *Show Not Similar* untuk menampilkan *token-token* yang tidak sama dan *Reset* untuk me-*refresh* halaman utama ke default. Menu *Help* merupakan batuan tentang bagaimana menjalankan aplikasi. Terdiri dari submenu *Help* dan *About*.

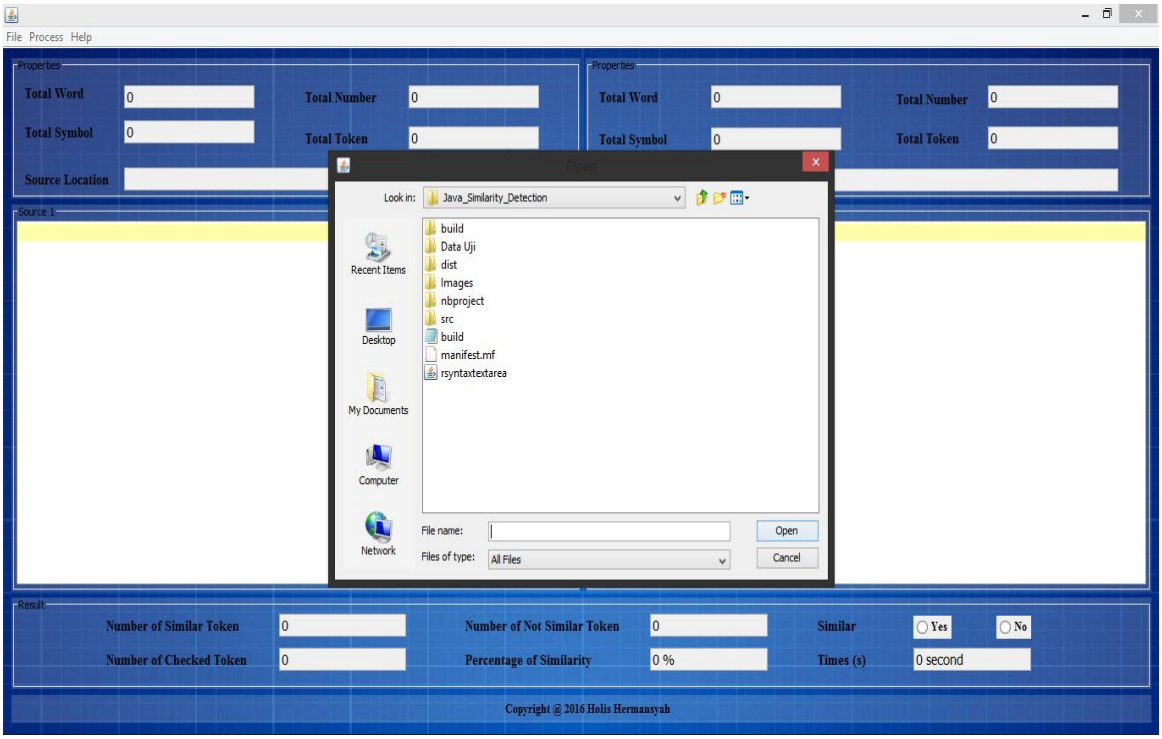


Gambar 2 Desain *interface* aplikasi

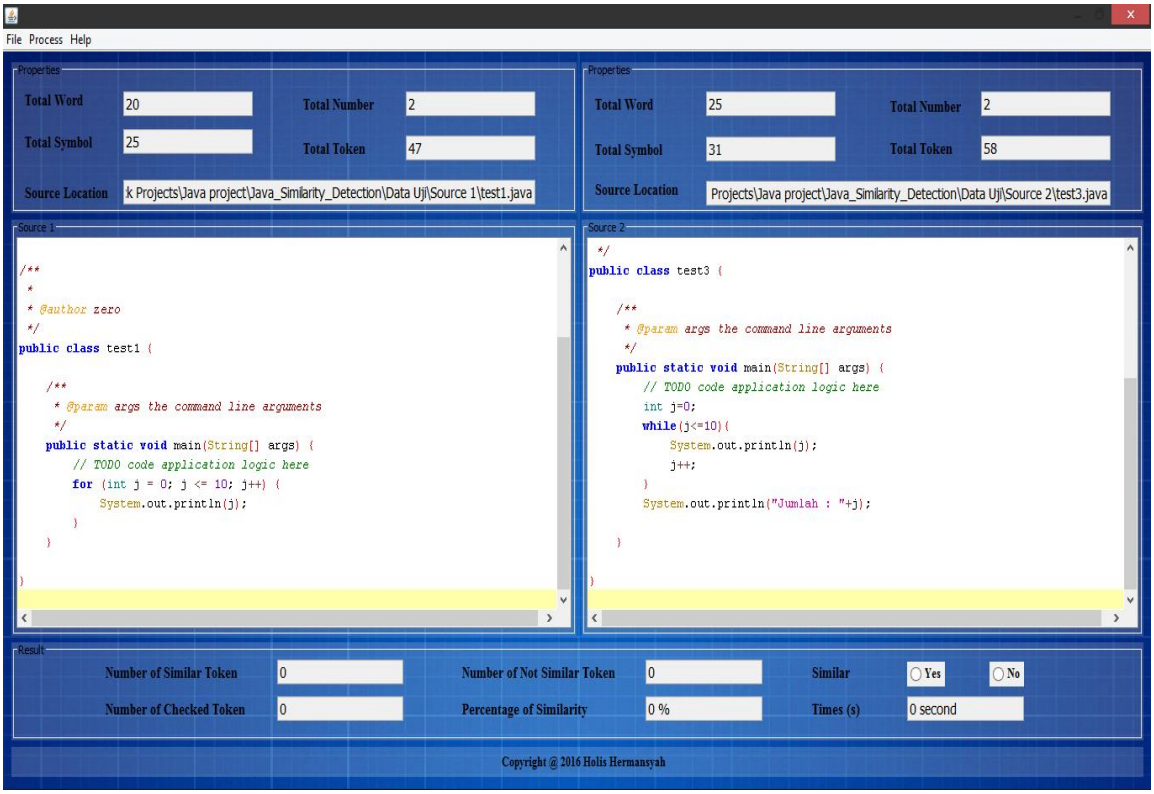
3.2 Pengujian Sistem

Pengujian ini dilakukan untuk memeriksa apakah semua menu dan submenu yang ada pada sistem dapat berfungsi dengan baik. Pengujian ini dilakukan dengan memasukkan *source code*, *scanning*, menampilkan hasil analisis, menampilkan *token* yang mirip dan yang tidak mirip pada *source code* yang diperiksa serta menampilkan menu *Help* dan *About*.

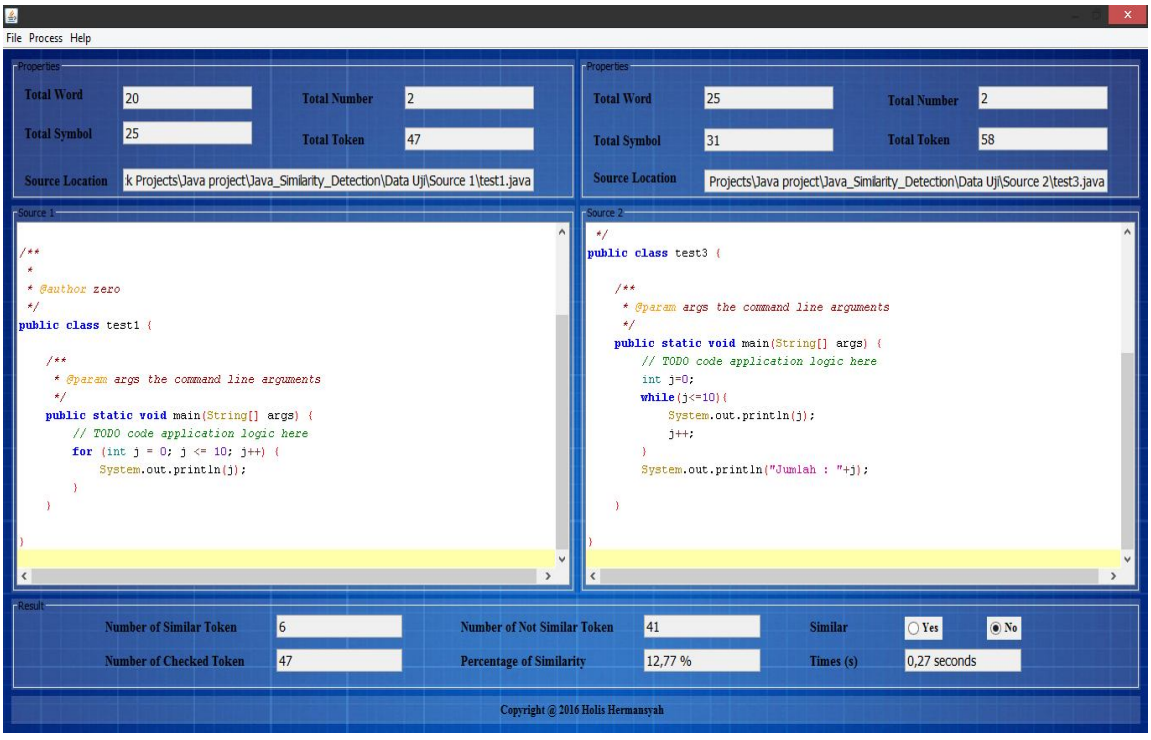
1. Memasukan dua buah *source code* yang akan diperiksa ke dalam *text editor* (Gambar 3)
2. Setelah *source code* ter-*input* maka pada menu utama akan tampil *properties* masing-masing *source code* yang berupa jumlah kata, jumlah simbol, jumlah angka dan total *token* yang terbentuk (Gambar 4).
3. Langkah selanjutnya yaitu melakukan proses *scanning* dengan menekan menu *process* dan memilih submenu *scanning*. Setelah sistem selesai men-*scan* kedua *source* maka akan tampil jumlah *token* yang sama, jumlah *token* yang tidak sama, jumlah *token* yang diperiksa, presentase kemiripan, keterangan *plagiat* atau bukan dalam bentuk *radio button* dan waktu yang diperlukan pada saat melakukan proses *scanning* (Gambar 5).
4. Menampilkan hasil analsis secara keseluruhan (Gambar 6).



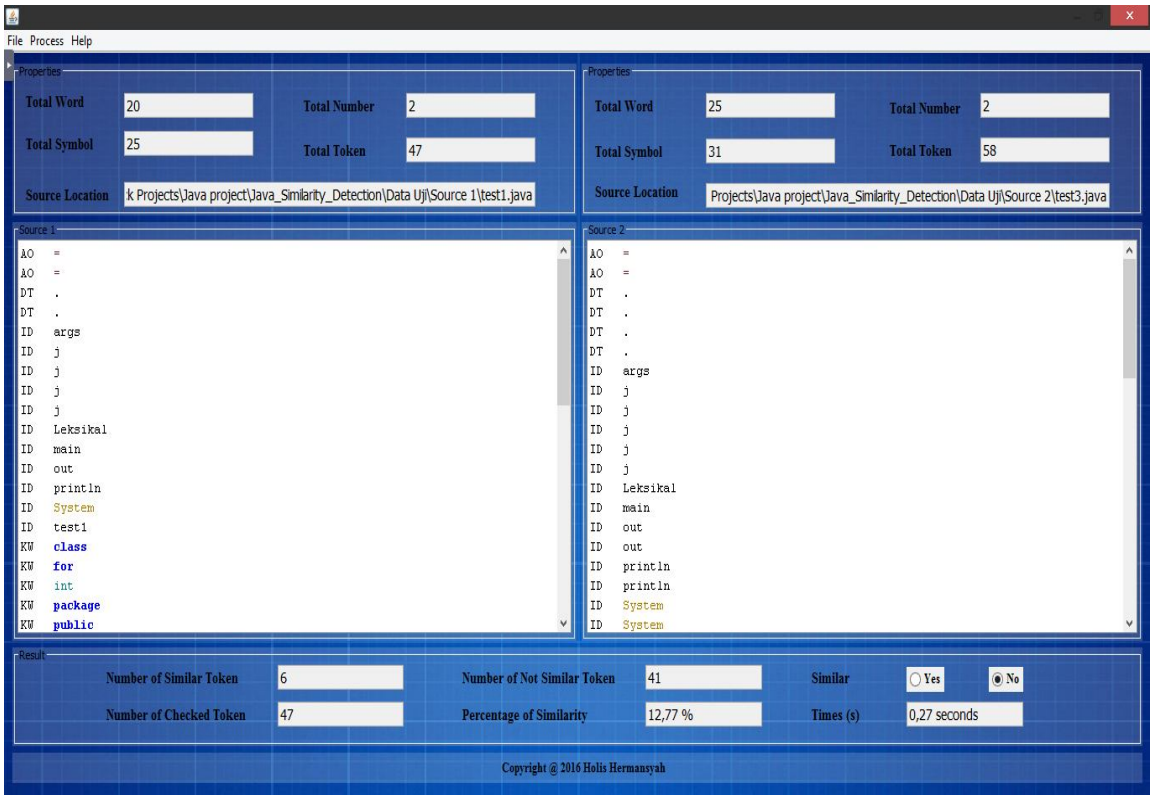

Gambar 3 Tampilan dialog *open source*




Gambar 4 Tampilan *source code* yang telah di-*input*




Gambar 5 Tampilan hasil *scanning* kemiripan




Gambar 6 Menampilkan hasil analisis keseluruhan

### 3.3 Hasil Analisis

Pada pengujian sistem yang dilakukan, sistem memeriksa dua *source code* yang masing-masing berjumlah 10 *file class*. Daftar *file class* yang di uji ditunjukkan oleh Tabel 1 dan Tabel 2.

Tabel 1 Daftar nama *Class Source* 1

| NO | *SOURCE* 1 | | | | |
|---|---|---|---|---|---|
| | NAMA *CLASS* | ∑ KATA | ∑ SIMBOL | ∑ ANGKA | TOTAL *TOKEN* |
| 1 | BilPrima.*Java* | 42 | 48 | 2 | 92 |
| 2 | CekNilai.*Java* | 94 | 120 | 5 | 219 |
| 3 | HitungLingkaran.*Java* | 47 | 47 | 2 | 96 |
| 4 | test1.*Java* | 20 | 25 | 2 | 47 |
| 5 | test3.*Java* | 25 | 31 | 2 | 58 |
| 6 | test5.*Java* | 28 | 36 | 4 | 68 |
| 7 | NilaiUjian.*Java* | 75 | 104 | 10 | 189 |
| 8 | scanner.*Java* | 33 | 33 | 0 | 66 |
| 9 | MembuatPola XD i*Java*.*Java* | 59 | 88 | 3 | 154 |
| 10 | Main.*Java* | 100 | 123 | 1 | 224 |

Tabel 2 Daftar Nama *Class Source* 2

| NO | *SOURCE* 2 | | | | |
|---|---|---|---|---|---|
| | NAMA *CLASS* | ∑ KATA | ∑ SIMBOL | ∑ ANGKA | TOTAL *TOKEN* |
| 1 | prima.*Java* | 46 | 49 | 2 | 97 |
| 2 | nilai.*Java* | 94 | 120 | 5 | 219 |
| 3 | Lingkaran.*Java* | 49 | 50 | 2 | 101 |
| 4 | test2.*Java* | 20 | 25 | 2 | 47 |
| 5 | test4.*Java* | 25 | 31 | 2 | 58 |
| 6 | test6.*Java* | 29 | 36 | 4 | 69 |
| 7 | nilaiujian.*Java* | 75 | 104 | 10 | 189 |
| 8 | inputNama.*Java* | 33 | 33 | 0 | 66 |
| 9 | NewMain2.*Java* | 60 | 87 | 3 | 154 |
| 10 | Main.*Java* | 100 | 123 | 1 | 224 |

Untuk hasil analisa pengujian sistem terhadap kinerja algoritma ditunjukkan oleh Tebel 3.

Tabel 3 Hasil Analisa Pengujian Sistem

| NO | ∑*TOKEN* YANG SAMA | ∑*TOKEN* YANG DIPERIKSA | KEMIRIPAN (%) | WAKTU (S) |
|---|---|---|---|---|
| 1 | 23 | 92 | 25 | 0,31 |
| 2 | 16 | 219 | 7,31 | 0,55 |
| 3 | 26 | 96 | 27,08 | 0,23 |
| 4 | 41 | 47 | 87,23 | 0,11 |
| 5 | 52 | 58 | 89,66 | 0,11 |
| 6 | 22 | 68 | 32,35 | 0,16 |
| 7 | 169 | 189 | 89,42 | 0,5 |
| 8 | 60 | 66 | 90,91 | 0,14 |
| 9 | 152 | 154 | 98,7 | 0,31 |
| 10 | 224 | 224 | 100 | 0,44 |

Tabel 3 menunjukkan bahwa terdapat 10 hasil pengujian, dimana pengujian nomor 1 merupakan hasil pengujian antara *source* nomor 1 pada Tabel 1 dan *source* nomor 1 pada Tabel 2. Begitu juga dengan hasil pengujian pada nomor dua merupakan hasil pengujian antara *source* nomor 2 pada Tabel 1 dengan *source* nomor 2 pada Tabel 2, begitu juga dengan hasil pengujian nomor 3 sampai 10 diuji berdasarkan urutan nomor Tabel 1 dan Tabel 2.

## 4. KESIMPULAN

Berdasarkan uraian dan hasil analisa yang telah dilakukan selama pengembangan Aplikasi Deteksi Kemiripan *Source Code* Pada Bahasa Pemrograman *Java* Menggunakan Metode Analisis Leksikal ini, dapat diambil kesimpulan yaitu :

1. Pada dasarnya proses deteksi kemiripan *source code* ini dilakukan dengan mendeteksi struktur atau alur serta *source code* dari program tersebut dengan metode analisis leksikal.
2. Hasil dari aplikasi ini berupa nilai presentase kemiripan yang dapat digunakan sebagai acuan penentuan adanya tindakan *plagiarisme.*

## 5. SARAN

Dalam pembuatan Aplikasi Deteksi Kemiripan *Source Code* Pada Bahasa Pemrograman *Java* Menggunakan Metode Analisis Leksikal ini masih banyak terdapat kekurangan dan jauh dari kata sempurna. Untuk itu masih perlu dilakukan sebuah penyempurnaan. Berikut beberapa saran untuk pengembangan lebih lanjut dari aplikasi ini :

1. Aplikasi ini mendeteksi kemiripan kode program hanya pada bahasa *Java*, untuk kedepannya diharapkan dapat mendeteksi lebih dari satu bahasa pemrograman.
2. Jumlah *file* yang diperiksa pada aplikasi ini berjumlah dua buah *source code*, untuk lebih menghemat waktu pemeriksaan kedepannya dapat dikembangkan dengan memeriksaa *file* lebih dari satu.
3. *File* yang diperiksa pada aplikasi ini adalah *file class* dari program yang akan diperiksa, untuk pengembangan dapat ditingkatkan dengan memeriksa *project* dari apliksi tersebut.

## DAFTAR PUSTAKA

[1] Redya, F., 2006, Penerapan Algoritma Boyer-Moore Untuk Pengecekan Plagiatisme Source Code, *Jurnal*, Laboratorium Ilmu dan Rekayasa Komputasi, Institut Teknologi Bandung.

[2] Budhy, E., 2014, Deteksi Similarity Source Code Menggunakan Metode Deteksi Abstract Syntax Tree, *Seminar Nasional Sains Dan Teknologi 2014.* Fakultas Tenik, Universitas Muhammadiyah Jakarta.

[3] Sastroasmoro, S, 2007, Beberapa Catatan Tentang *Plagiarisme*, *Majalah Kedokteran Indonesia*, Volum: 57, Nomor: 8, Agustus 2007.

[4] Henryw, 2014. Teknik Kompilasi, *http://henryw.blog.binusian.org/teknik-kompilasi.html*. Diakses tanggal 29 Februari 2016.