# BAB III

# METODOLOGI DAN PERANCANGAN

## 3.1 Metode Penelitian

Metode yang digunakan dalam penelitian ini adalah sebagai berikut.

1. Studi Pustaka

   Melakukan studi terhadap laporan atau jurnal penelitian yang telah ditulis oleh peneliti lain, artikel-artikel yang berkaitan dengan plagiarisme, dan algoritma maupun teknik preprocessing yang dapat digunakan untuk membangun aplikasi pendeteksi plagiarisme.

2. Analisis Masalah

   Melakukan observasi data-data dari hasil penelitian sebelumnya untuk menetapkan batasan-batasan dan kebutuhan dalam membangun aplikasi pendeteksi plagiarisme.

3. Perancangan

   Melakukan perancangan algoritma dan teknik preprocessing yang akan diterapkan dalam aplikasi pendeteksi plagiarisme dalam kode program bahasa Java sesuai dengan tujuan, batasan, dan kebutuhan yang sudah ditetapkan.

4. Implementasi

   Mengimplementasikan algoritma dan teknik yang sudah dirancang sebelumnya dalam membangun aplikasi pendeteksi plagiarisme pada kode program bahasa Java.

5. Pengujian

   Melakukan pengujian terhadap aplikasi yang sudah dibuat dengan melakukan perbandingan terhadap kode program bahasa Java, dan menganalisis hasil dari

aplikasi tersebut dengan menghitung waktu eksekusi dan persentase ketepatan yang dihasilkan.

6. Penulisan Laporan

   Melakukan penulisan laporan secara bertahap mulai dari tahap studi pustaka sampai dengan pengujian aplikasi.

## 3.2 Analisis Masalah

Salah satu hal yang menyebabkan semakin banyaknya tindak plagiarisme adalah mudahnya pengaksesan dan pertukaran informasi dengan adanya internet, *e-mail, flashdisk*, maupun media lainnya. Ditambah dengan kehadiran komputer jinjing maupun *gadget*, siapapun dapat mengakses berbagai informasi kapanpun dari manapun.

Disamping maraknya plagiarisme, perbandingan antara panjang kode yang diperiksa dengan jumlah pengajar yang memeriksa kode-kode tersebut juga menjadi kendala dalam menindaki plagiarisme. Misalkan saja seorang pengajar mengampu suatu matakuliah pemrograman dengan kapasitas kelas 30 mahasiswa selama 14 pertemuan. Setiap pertemuan masing-masing mahasiswa menghasilkan 100 baris kode, maka setiap pertemuan seorang dosen harus memeriksa 3000 baris kode. Dan untuk membandingkan setiap pasang kode secara spesifik, pengajar tersebut harus melakukan pemeriksaan terhadap 200 baris kode sebanyak 435 kali untuk setiap pertemuan. Sehingga untuk melakukan perbandingan spesifik terhadap setiap pasang kode, pengajar tersebut harus memeriksa 87.000 baris kode untuk setiap pertemuan, dan 1.218.000 baris kode selama satu semester. Bagaimana bila setiap mahasiswa menghasilkan 1000 baris kode selama 2 atau 3

pertemuan dan pengajar tersebut mengajar pada 3 kelas yang masing-masing berisi 40 orang mahasiswa?

Untuk mengatasi masalah tersebut, sudah dilakukan sejumlah penelitian. Diantaranya adalah penelitian yang dilakukan oleh Andreas Arfianto pada tahun 2011, Regina Natalia pada tahun 2011, Ahmad Gull Liaqat dan Aijaz Ahmad pada tahun 2011.

Penelitian yang dilakukan oleh Andreas Arfianto dan Regina Natalia memproses kode dengan bahasa C yang merupakan bahasa prosedural dan tidak dapat diterapkan pada bahasa yang berbasis objek. Penelitian yang dilakukan oleh Ahmad Gull Liaqat dan Aijaz Ahmad memproses kode bahasa Java dengan algoritma *Levenshtein Distance*. Berdasarkan penelitian yang dilakukan oleh Regina Natalia, algoritma *Levenshtein Distance* dibuktikan kurang cocok untuk digunakan untuk melakukan perbandingan *string*. Proses *preprocessing* yang digunakan pada penelitian milik Ahmad Gull Liaqat dan Aijaz Ahmad juga hanya memungkinkan pemeriksaan untuk perubahan pada komentar dan nama variabel, dan tidak mencangkup elemen-elemen lain seperti perubahan struktur *switch* menjadi *if-else*, atau struktur *if-else* menjadi *conditional ternary* pada kode Java.

Selain itu, aplikasi yang dibangun pada penelitian-penelitian tersebut merupakan aplikasi berbasis *desktop*, sehingga hanya dapat digunakan oleh orang yang memiliki aplikasi tersebut.

### 3.3 Pemecahan Masalah

Berdasarkan masalah yang ditemukan, ditawarkan sebuah solusi berupa aplikasi dengan kemampuan sebagai berikut.

1. Menghitung bobot kecenderungan plagiarisme pada sepasang proyek kode Java pada saat yang bersamaan berdasarkan baris, dan kemudian menampilkan hasilnya dalam bentuk presentase kecenderungan plagiarisme yang terjadi.
2. Memodifikasi teknik *preprocessing* yang telah ada pada aplikasi yang telah dibuat sebelumnya sehingga dapat memiliki batasan yang lebih luas.
3. Menghasilkan presentase kecenderungan plagiarisme pada sejumlah proyek kode Java dengan algoritma *Forward Fast Search*.
4. Merupakan aplikasi yang berbasis pada *website*.

### 3.4 Rancangan Masukkan dan Keluaran Sistem

Rancangan masukkan dari aplikasi ini adalah sebagai berikut.

1. Dua *file* atau proyek kode program Java dengan ekstensi .java untuk *file* dan .tar atau .zip untuk proyek.
2. Sebuah *file* .zip atau .tar yang berisi seluruh *file* atau proyek kode program Java.

Rancangan keluaran dari aplikasi ini adalah sebagai berikut.

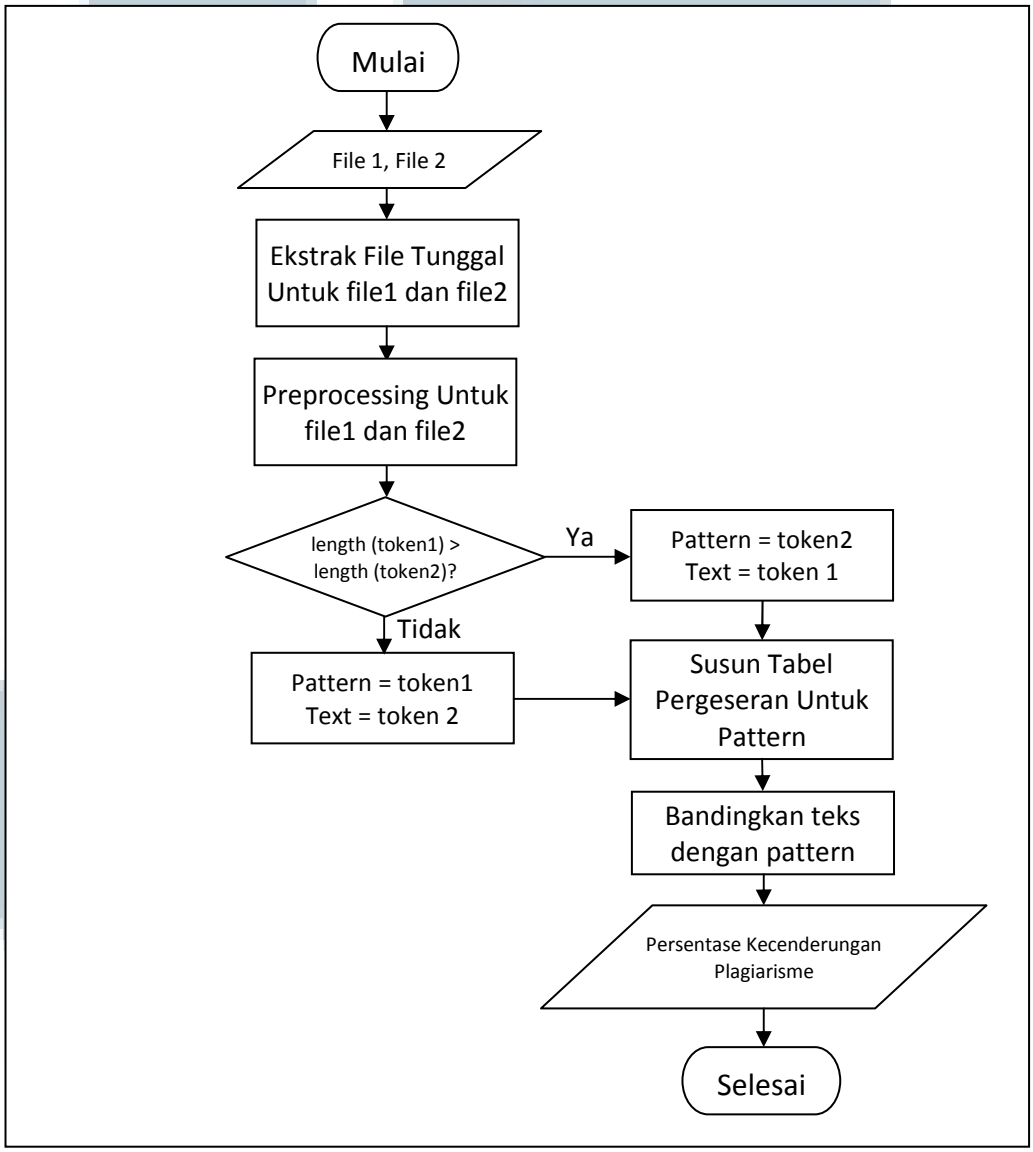
1. Sebuah halaman HTML yang memuat presentase kecenderungan dua proyek kode Java dan informasi mengenai kedua *file* tersebut yang meliputi jumlah *class*, banyak *line* yang dibandingkan pada kedua *file*, dan jumlah *line* yang serupa.
2. Sebuah halaman HTML yang memuat tabel presentase kecenderungan seluruh proyek kode Java yang diperiksa dan informasi mengenai seluruh *file* tersebut yang meliputi jumlah *file* yang diperiksa, jumlah *class* pada masing-masing proyek, banyak *line* yang dibandingkan pada masing-masing *file*, dan jumlah *line* yang serupa pada masing-masing perbandingan.

## 3.5 Perancangan Sistem

Secara garis besar, rancangan sistem dibagi menjadi dua proses utama, yaitu proses perbandingan dua kode program Java dan proses lebih dari dua kode Java.

Proses perbandingan dua kode program Java menerima dua *file* dengan ekstensi .java, .zip, atau .tar yang nantinya akan diekstrak dan menjalani tahap *preprocessing* untuk menghasilkan *token* dan dibandingkan. Proses ini menghasilkan sebuah nilai presentase kecenderungan plagiarisme.

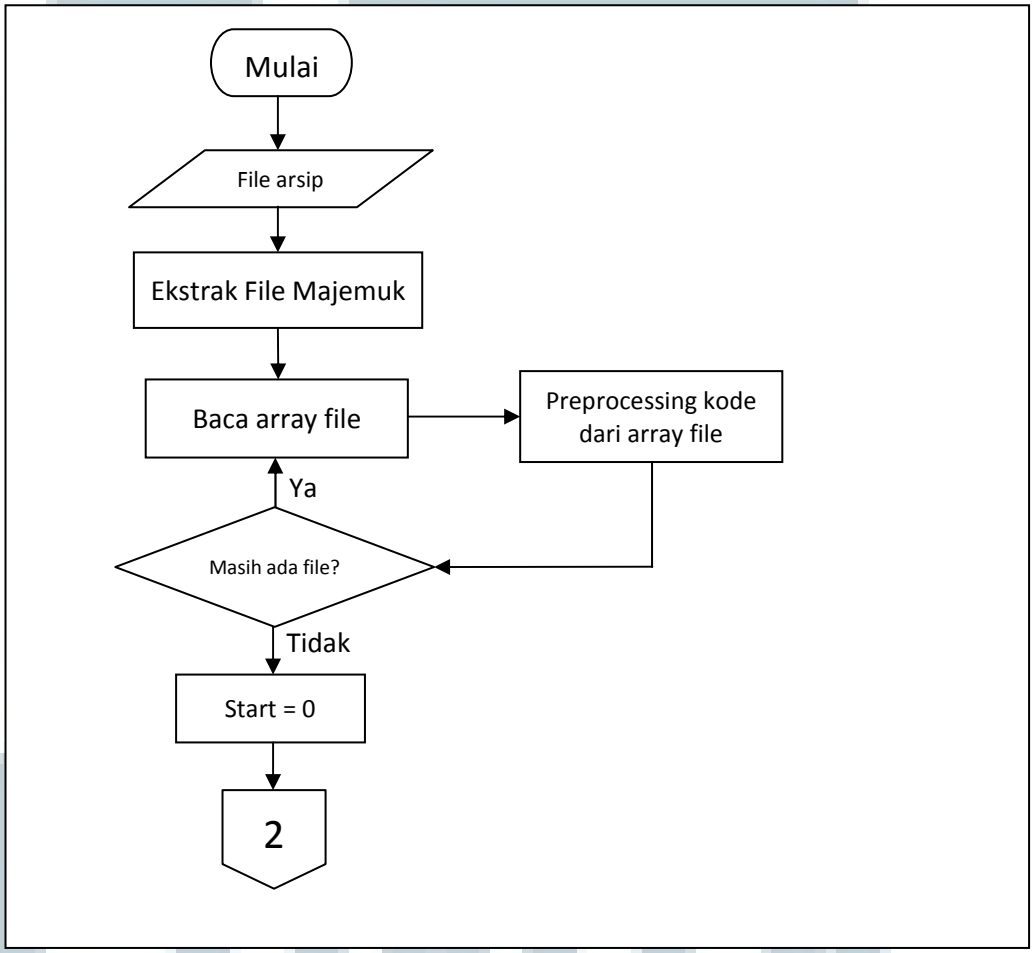
Gambar 3.1 adalah *flowchart* dari proses perbandingan dua kode program Java.



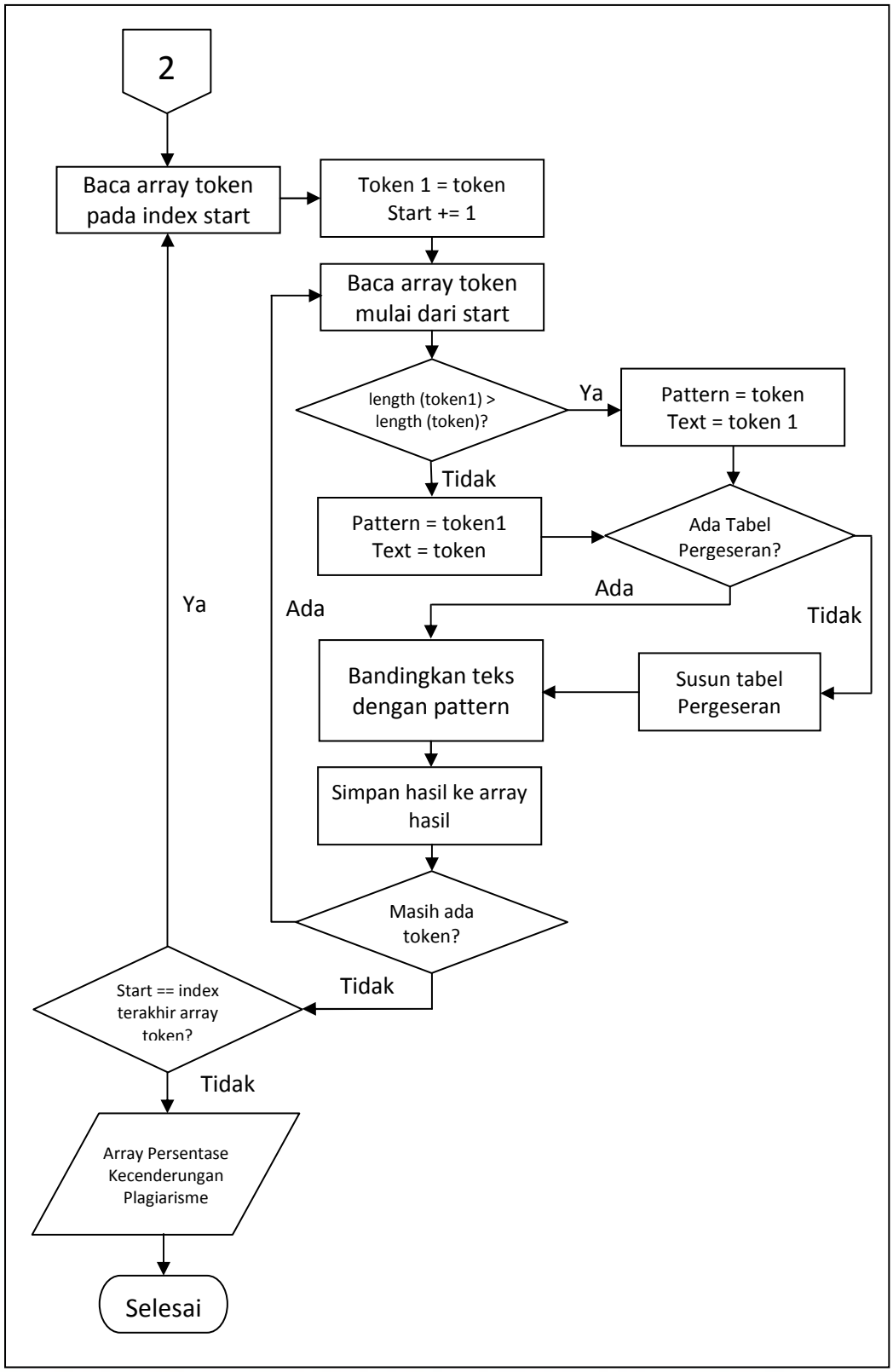

Gambar 3.1 *Flowchart* Proses Perbandingan Dua Proyek Kode Java

Proses proses perbandingan lebih dari dua kode Java menerima sebuah *file* dengan ekstensi .zip, atau .tar yang berisi seluruh proyek kode yang akan dibandingkan. *File* arsip yang digunakan pada proses ini harus memiliki lebih dari satu folder utama. Seluruh kode akan diproses untuk menghasilkan sebuah tabel yang berisi nilai presentase kecenderungan plagiarisme antar proyek.

Gambar 3.2 adalah *flowchart* dari proses perbandingan lebih dari dua kode Java.




Gambar 3.2 *Flowchart* Proses Perbandingan Lebih dari Dua Proyek Kode Java




Gambar 3.2 *Flowchart* Proses Perbandingan Lebih dari Dua Proyek Kode Java (Lanjutan)

Terdapat tahapan-tahapan proses yang harus dilalui pada kedua rancangan proses utama tersebut. Tahapan perancangan dari sistem yang dibuat yaitu:

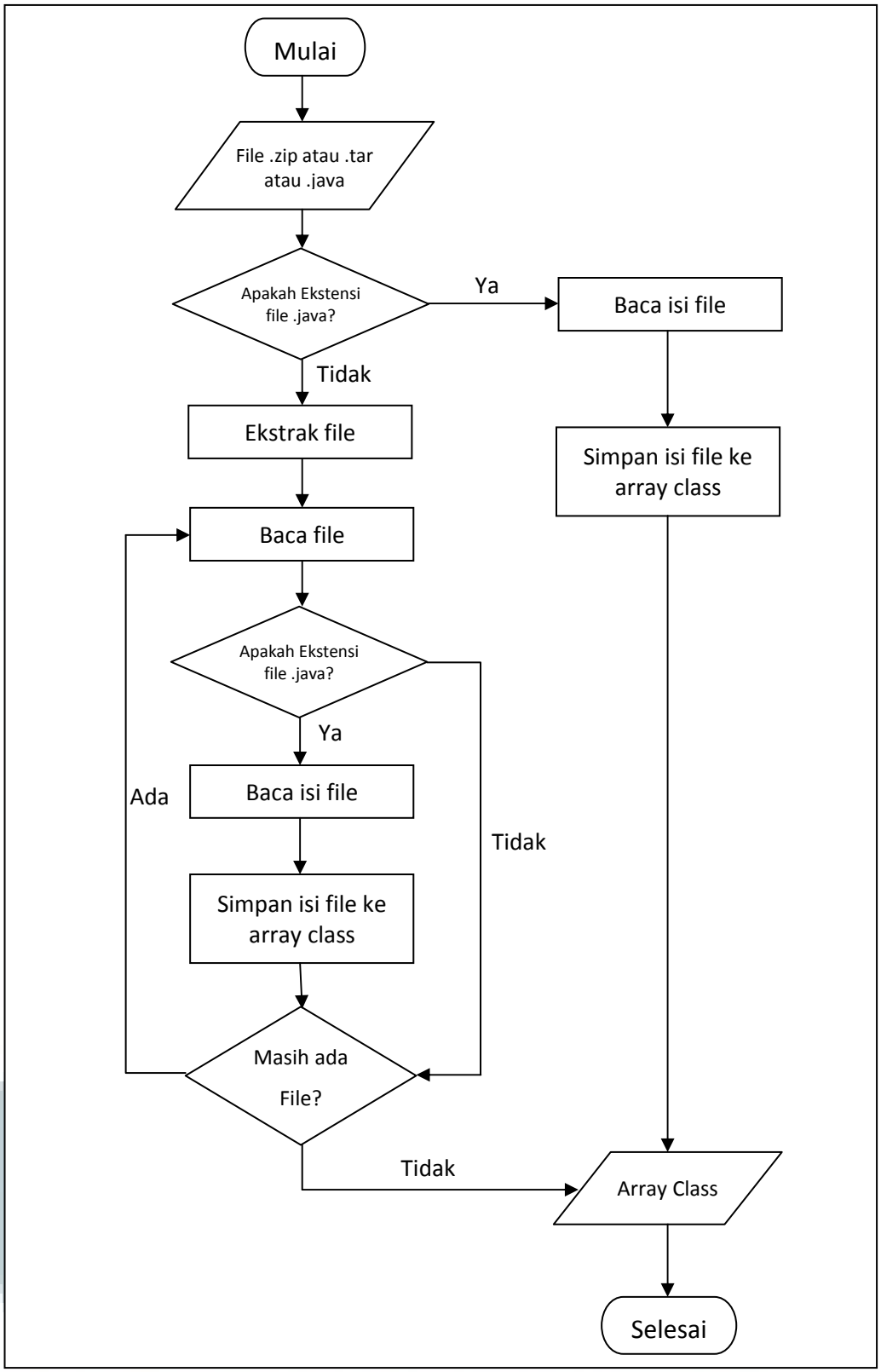
1. Tahap ekstrasi kode program dari *file* kode atau *file* arsip yang berisi sebuah proyek kode Java dan mengelompokkan class-class berdasarkan proyek kode tersebut.
2. Tahap ekstrasi kode program dari *file* kode atau *file* arsip yang berisi sejumlah proyek kode Java dan mengelompokkan class-class berdasarkan proyek kode tersebut.
3. Tahap *preprocessing* sebagai untuk menyeragamkan format kode sebelum dilakukan pembandingan kode program.
4. Tahap perhitungan tabel *Bad Character* dan tabel *Forward Good Suffix* untuk setiap *pattern* pada seluruh kode yang bertindak sebagai pembanding.
5. Tahap perhitungan persentase kecenderungan plagiarisme dengan algoritma *Forward Fast Search*.

### 3.5.1 Proses Sistem

#### A. Tahap Ekstraksi Kode Program dari File Arsip Proyek Kode Java

Tahapan proses ini bertujuan untuk membaca isi *file* Java atau *file* arsip yang berisi satu proyek kode Java dan menyimpan isi dari masing-masing *file* yang dibaca ke dalam *array* untuk diproses lebih lanjut. Proses ini menerima *file* Java dengan ekstensi .java atau *file* arsip dengan ekstensi .zip atau .tar dan menghasilkan sebuah *array* yang berisi isi *file* dari seluruh *file* Java yang ditemukan.

Gambar 3.3 adalah *flowchart* dari proses ekstraksi kode program dari file arsip proyek kode java.

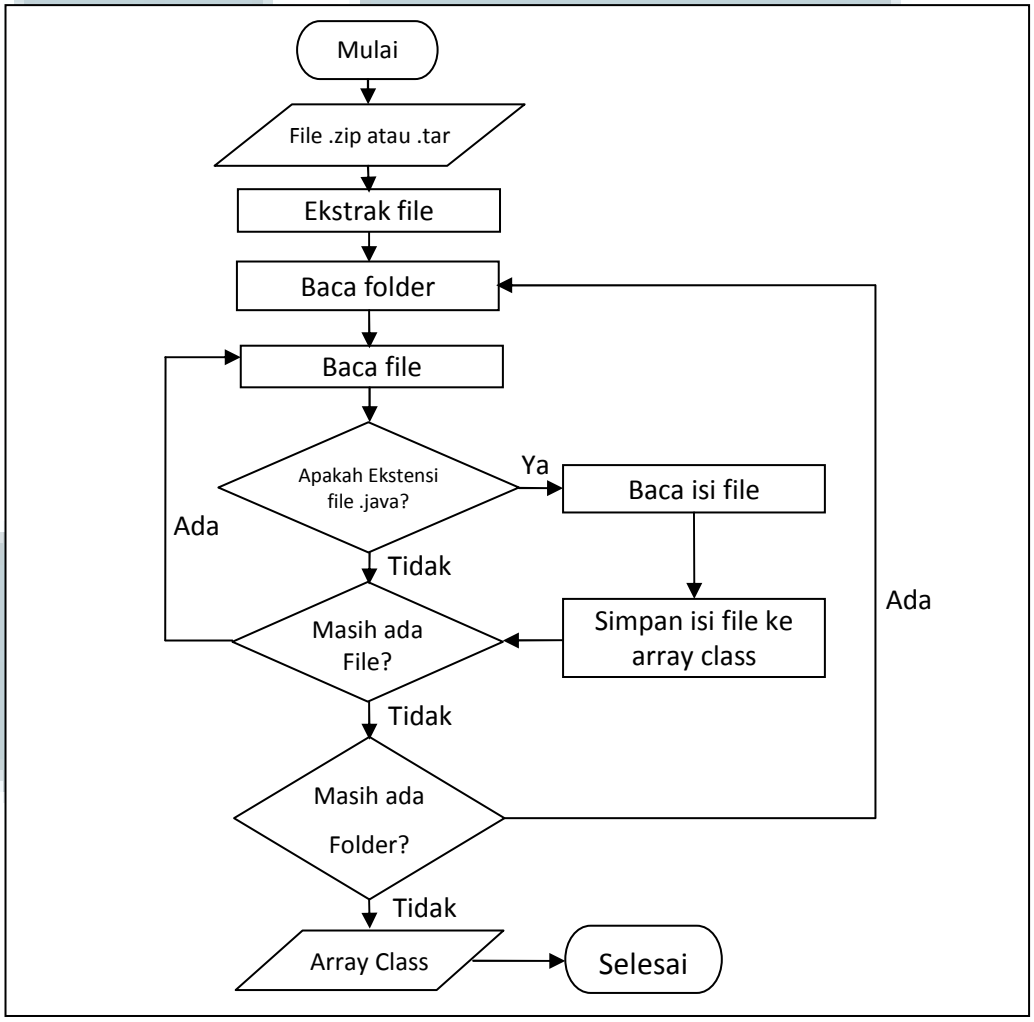



Gambar 3.3 *Flowchart* Ekstraksi Kode Program Tunggal

## B. Tahap Ekstraksi Kode Program dari File Arsip Kumpulan Proyek Kode Java

Tahapan proses ini bertujuan untuk membaca isi *file* arsip yang berisi sejumlah proyek kode Java dan menyimpan seluruh kode dari masing-masing proyek ke dalam *array* untuk diproses lebih lanjut. Proses ini menerima *file* arsip dengan ekstensi .java dan menghasilkan sebuah *array* dua dimensi yang berisi isi *file* dari seluruh *file* Java yang ditemukan dan dipisahkan berdasarkan proyek kode masing-masing.

Gambar 3.4 adalah *flowchart* dari proses ekstraksi kode program dari file arsip kumpulan proyek kode java.

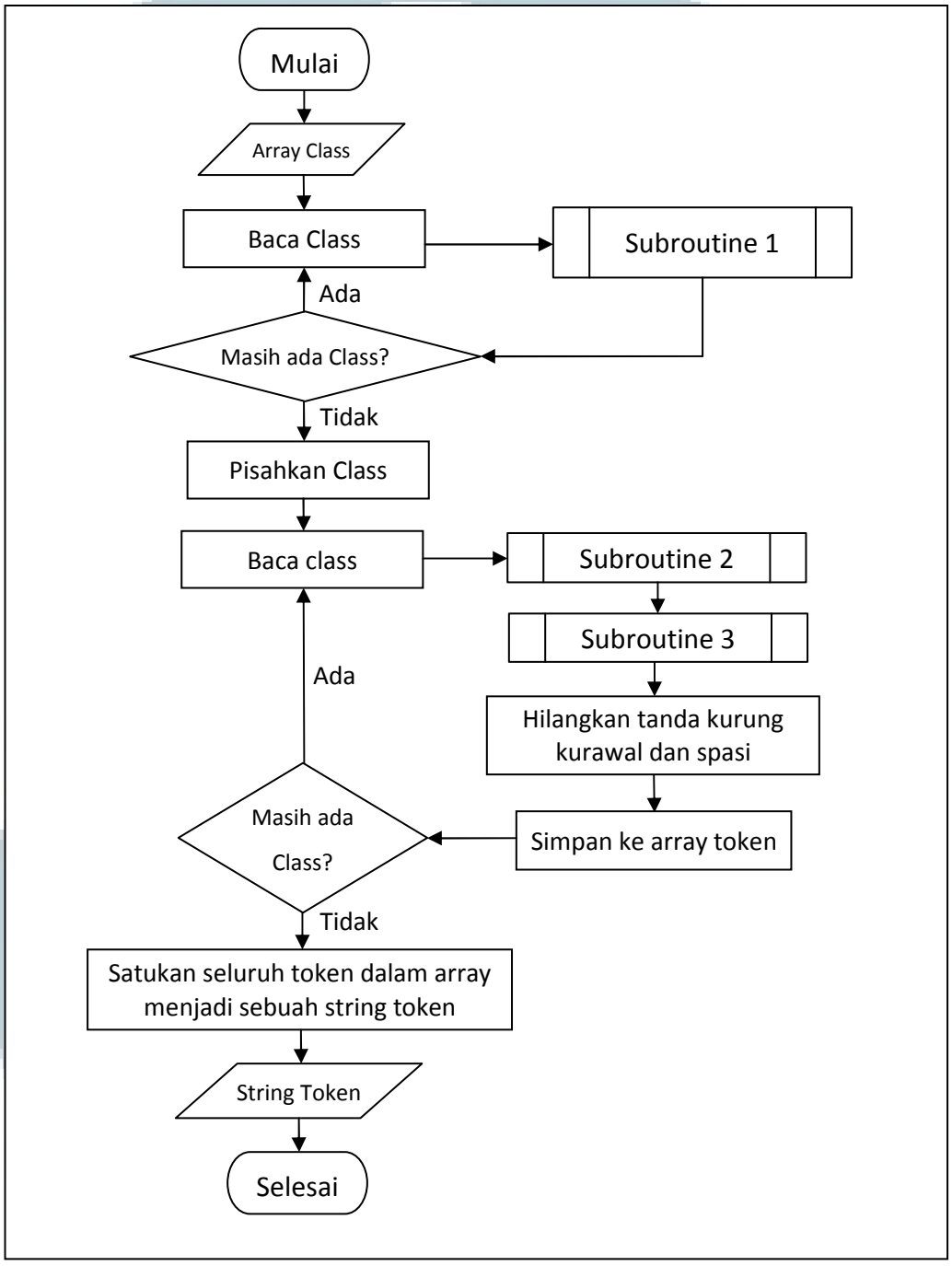



Gambar 3.4 *Flowchart* Ekstraksi Kode Program Majemuk

## C. Tahap Preprocessing

Tahap *preprocessing* bertujuan untuk menyeragamkan kode program sebelum dilakukan proses perbandingan untuk meminimalkan perbedaan di antara dua kode. Tahap *preprocessing* sendiri dibagi menjadi tiga *subroutine* lainnya.

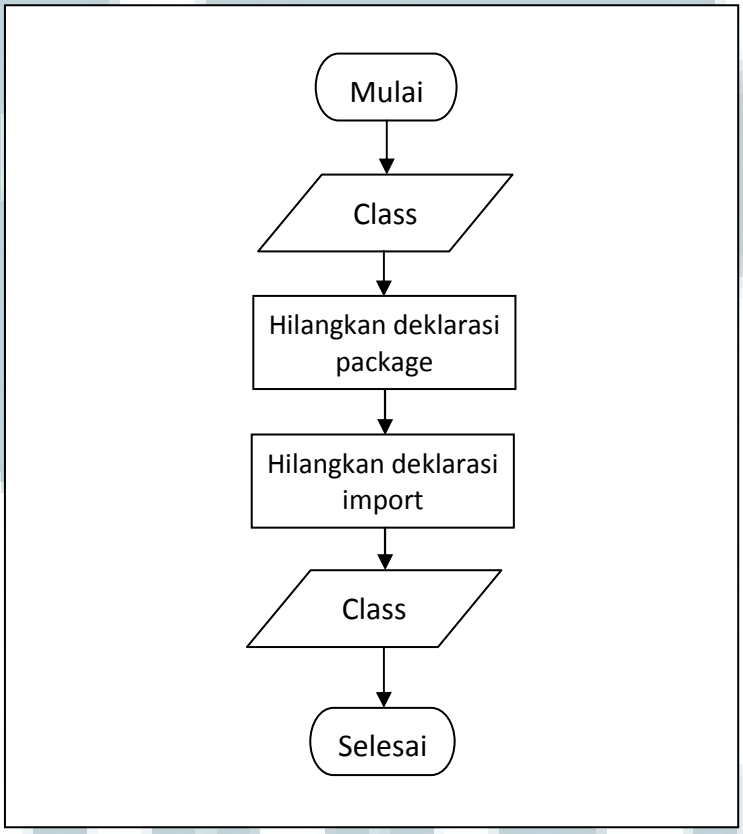
Gambar 3.5 adalah *flowchart* utama dari proses *preprocessing*.




Gambar 3.5 *Flowchart* Proses *Preprocessing* Utama

Proses *Subroutine1* bertujuan untuk menghilangkan deklarasi *package* dan *import* yang hanya dilakukan satu kali pada setiap *file* sebelum seluruh *class* pada *array* dipisahkan. Tujuan pemisahan *class* adalah agar setiap *index* pada *array* hanya menyimpan satu buah class meski ada *file* Java yang mendeklarasikan lebih dari satu *class*.

Gambar 3.6 adalah *flowchart* untuk proses *subroutine1* pada proses *preprocessing*.



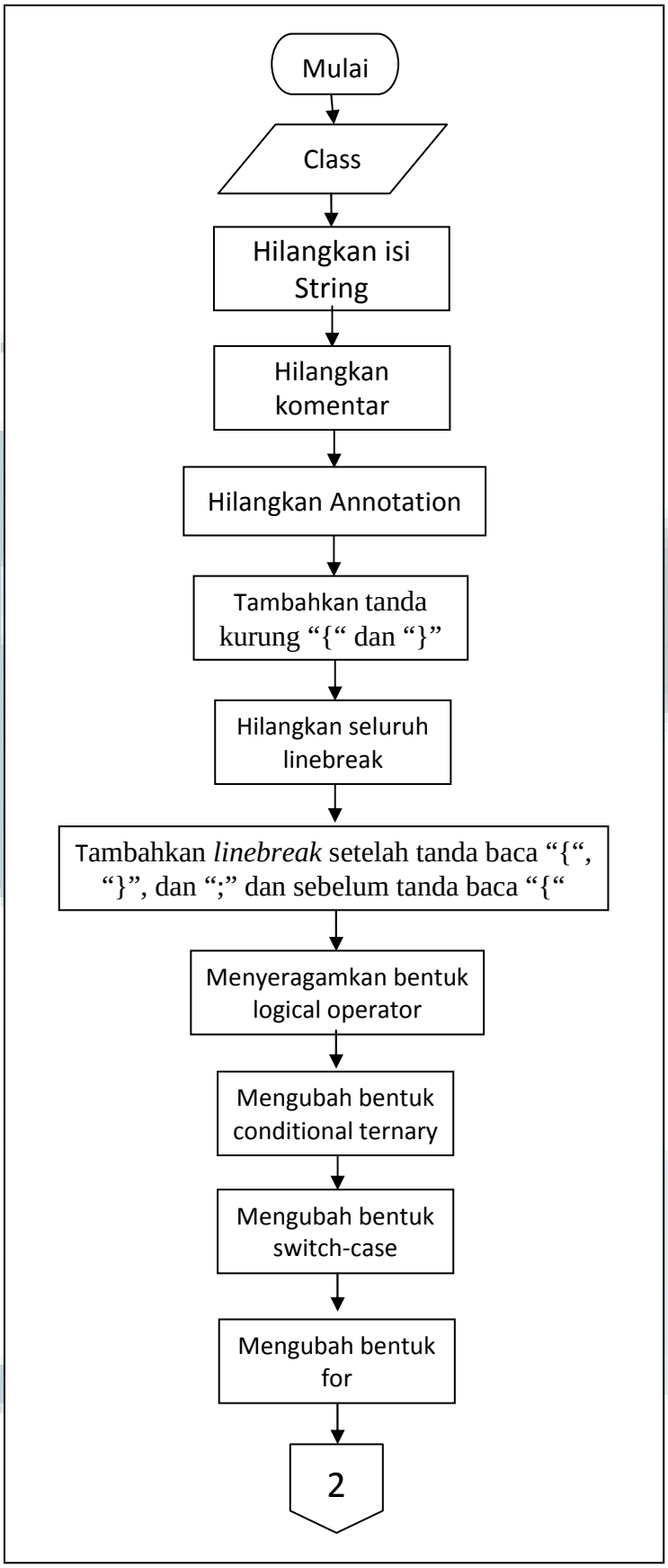

Gambar 3.6 *Flowchart Subroutine1* Proses *Preprocessing*

Proses *Subroutine2* bertujuan untuk menyeragamkan format kode sebelum dilakukan penyeragaman elemen kode dengan menggunakan *kode*. Tahap-tahap dari proses *Subroutine2* meliputi:

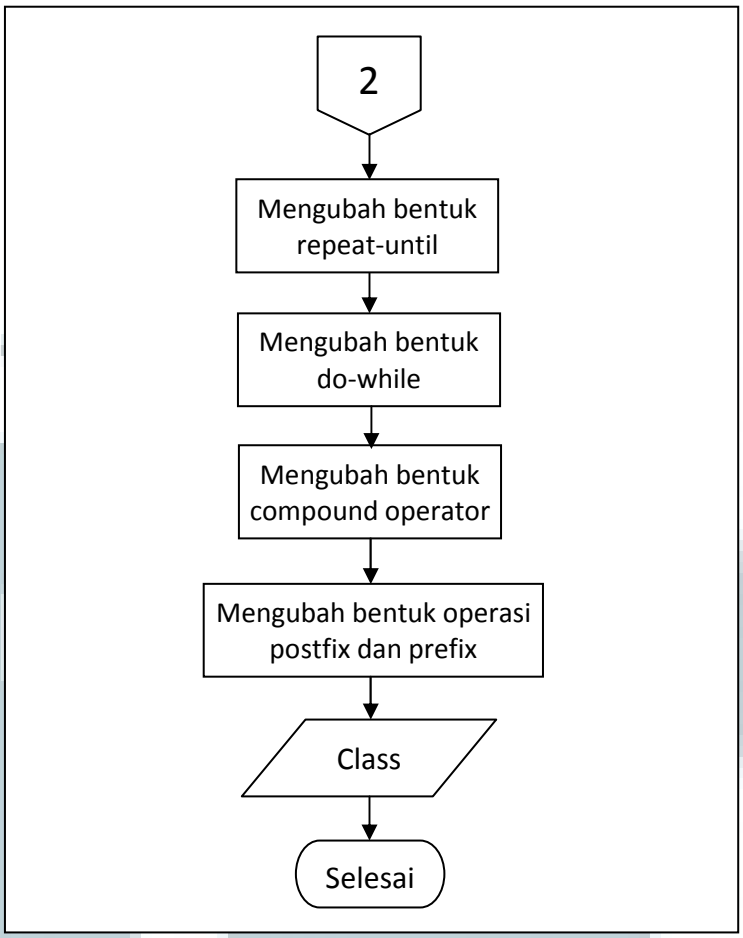
1. Menghilangkan isi dari *string*, yaitu seluruh karakter yang berada dalam tanda baca kutip dua.

2. Menghilangkan komentar, yaitu seluruh karakter dalam satu line setelah karakter “//” atau “#”, atau diapit oleh karakter “/*” dan “*/”.
3. Menghilangkan *annotations* pada kode Java.
4. Menambahkan tanda kurung “{“ dan “}” pada bentuk *if*, *else*, *for*, *while*, *do-while*, *reperat-until*, dan *try-catch* satu baris.
5. Menghilangkan seluruh *linebreak*.
6. Menambahkan *linebreak* setelah tanda baca “{“, “}”, dan “;” dan sebelum tanda baca “{“.
7. Mengganti *operator* “<=”, “<”, “&&”, “||”, dan “!=” masing-masing menjadi ">=", ">", "&", "|", dan "==", dan menghilangkan *operator* “!”.
8. Mengubah bentuk *conditional ternary* menjadi bentuk *if-else*.
9. Mengubah bentuk *switch-case* menjadi *if-else*.
10. Mengubah bentuk *for* menjadi bentuk *while* dan *foreach*.
11. Mengubah bentuk *repeat-until* menjadi *do-while*.
12. Mengubah bentuk *do-while* menjadi *while*.
13. Mengubah bentuk *compound operator* (var1 op= var2) menjadi bentuk operasi biasa (var1 = var1 op var2).
14. Mengubah bentuk *prefix* dan *postfix* baik *increment* maupun *decrement* menjadi bentuk operasi biasa.

Gambar 3.7 adalah *flowchart* untuk proses *subroutine2* pada proses *preprocessing*.



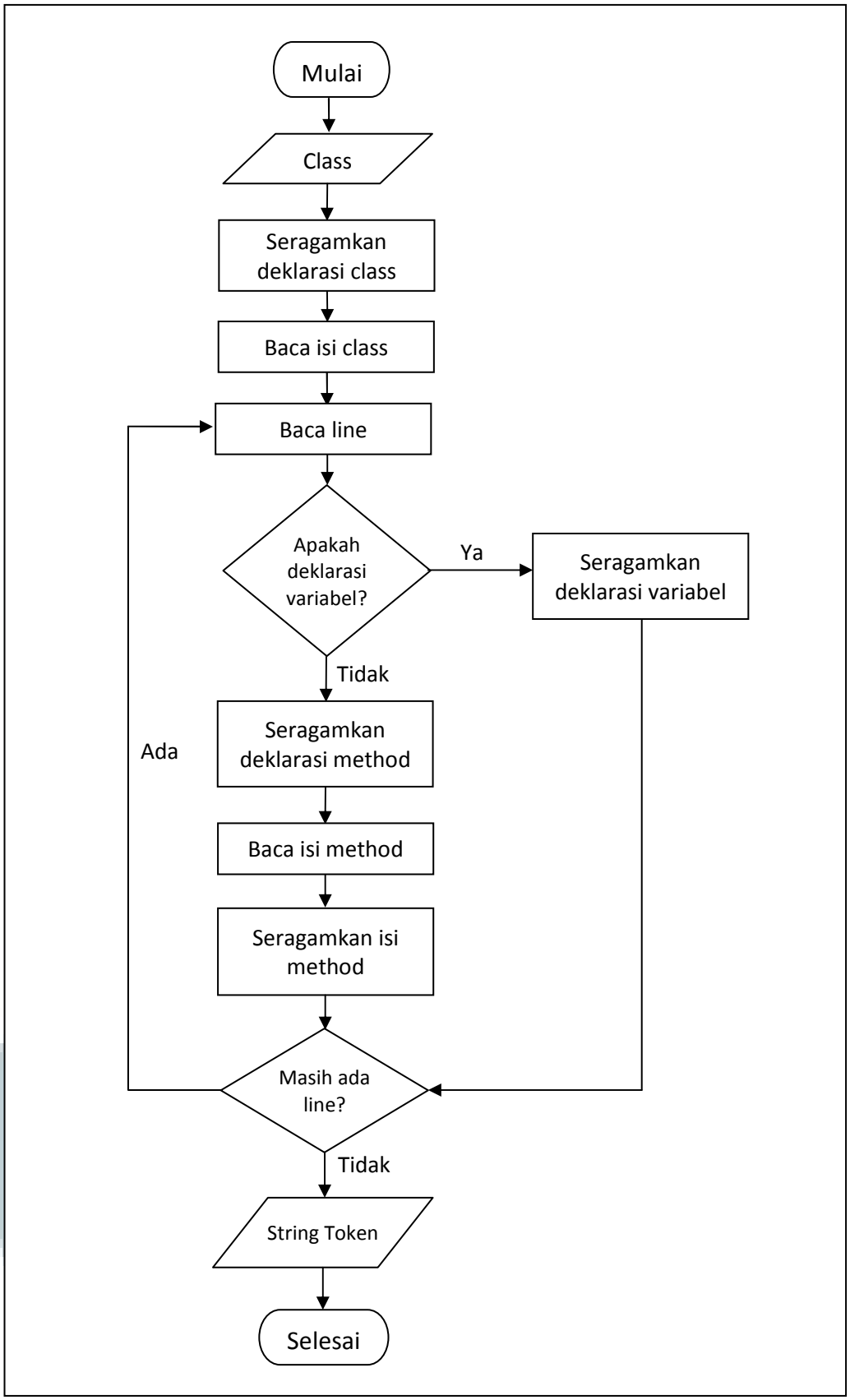

Gambar 3.7 *Flowchart Subroutine2* Proses *Preprocessing*




Gambar 3.7 *Flowchart Subroutine2* Proses *Preprocessing* (Lanjutan)

Proses *Subroutine3* bertujuan untuk menyeragamkan seluruh kode dengan mengganti seluruh elemen kode dengan *token*. *Token* yang dihasilkan dari proses ini nantinya akan dikembalikan ke proses *preprocessing* utama untuk dihilangkan seluruh spasi dan beberapa tanda bacanya.

Gambar 3.8 adalah *flowchart* untuk proses *subroutine3* pada proses *preprocessing*.

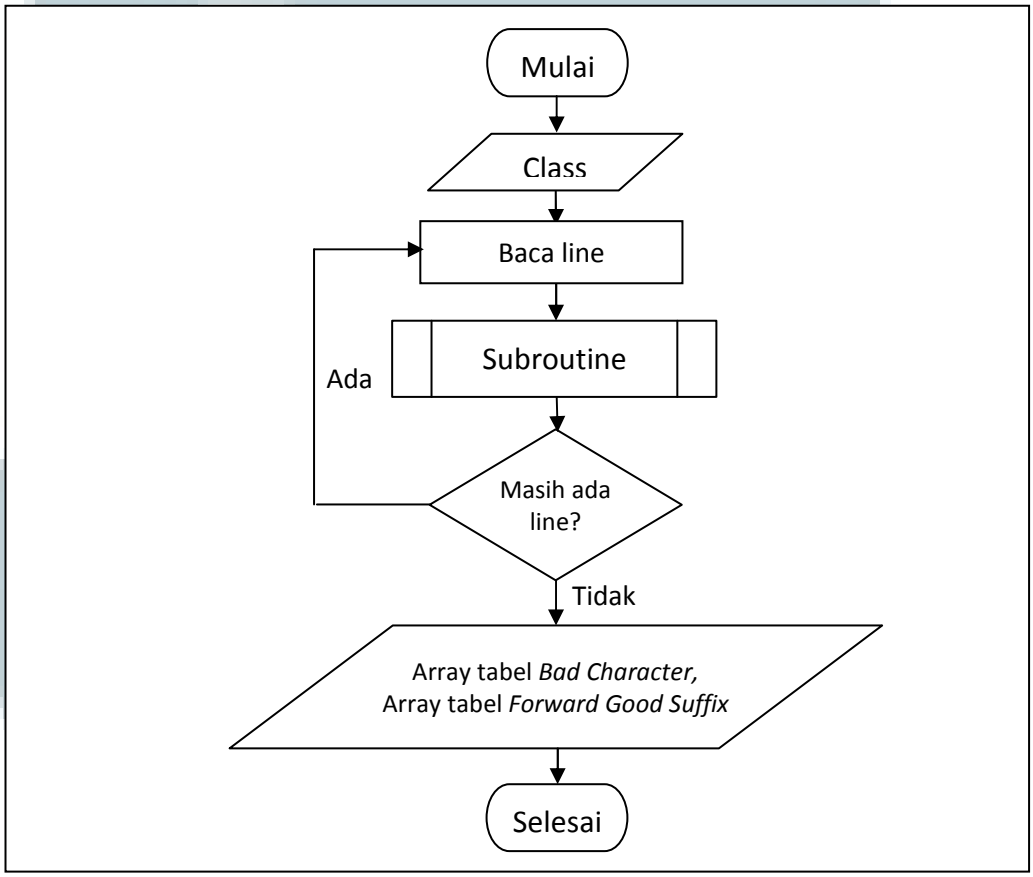



Gambar 3.8 *Flowchart Subroutine3* Proses *Preprocessing*

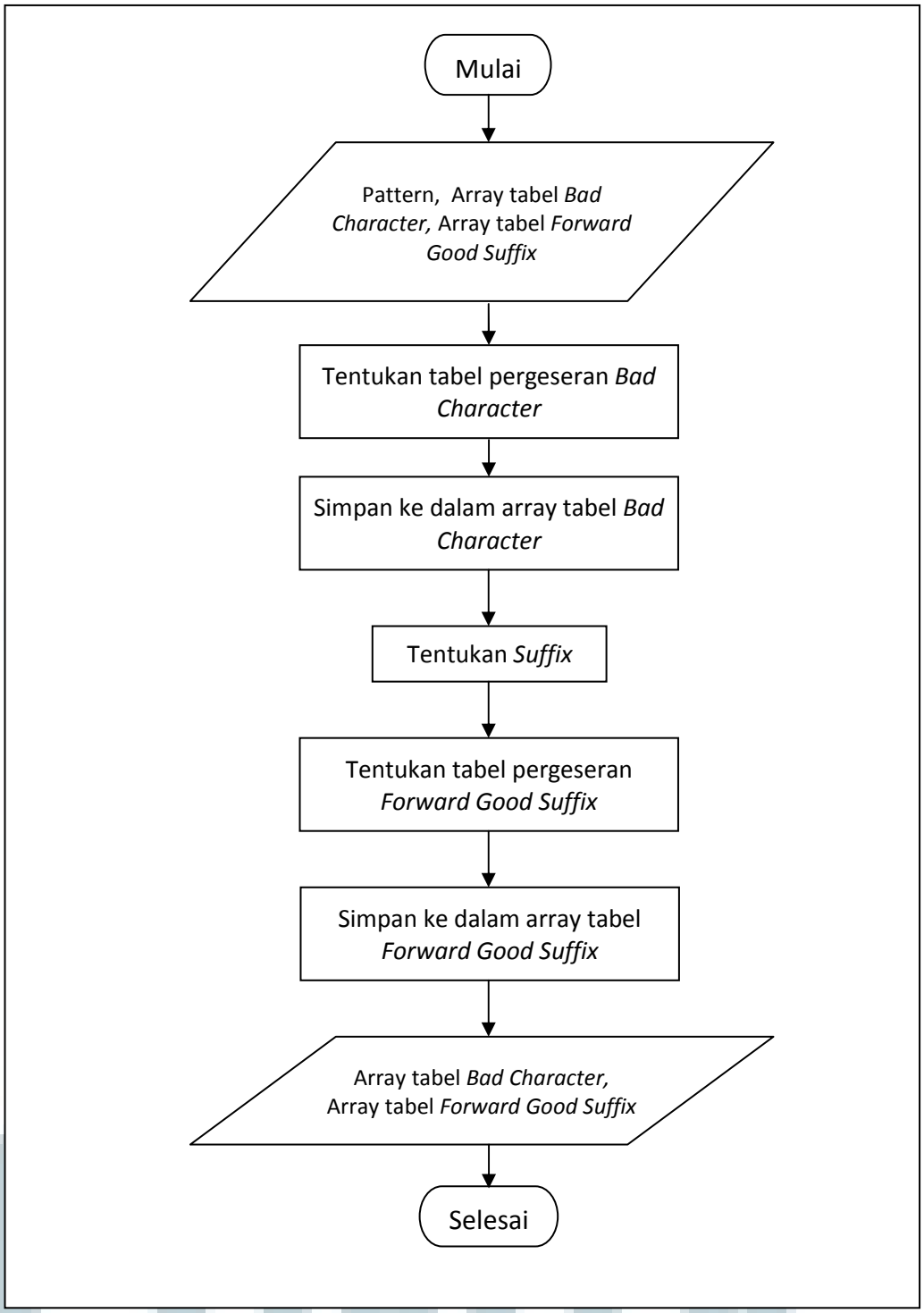
## D. Tahap Penyusunan Tabel Bad Character dan Tabel Forward Good Suffix

Tahapan proses ini bertujuan untuk membangun tabel *Bad Character* dan tabel *Forward Good Suffix* untuk menentukan jumlah pergeseran karakter pada saat perbandingan dilakukan dengan algoritma *Forward Fast Search*. Proses ini menerima sebuah *string token*, dimana setiap *line* pada *token* tersebut akan menjadi sebuah *pattern* yang nantinya akan dicocokkan dengan *text*, dan menghasilkan dua buah *array* untuk tabel *Bad Character* dan tabel *Forward Good Suffix*.

Gambar 3.9 dan gambar 3.10 adalah *flowchart* dari proses penyusunan tabel *bad character* dan tabel *forward good suffix.*




Gambar 3.9 *Flowchart* Penyusunan Tabel Pergeseran

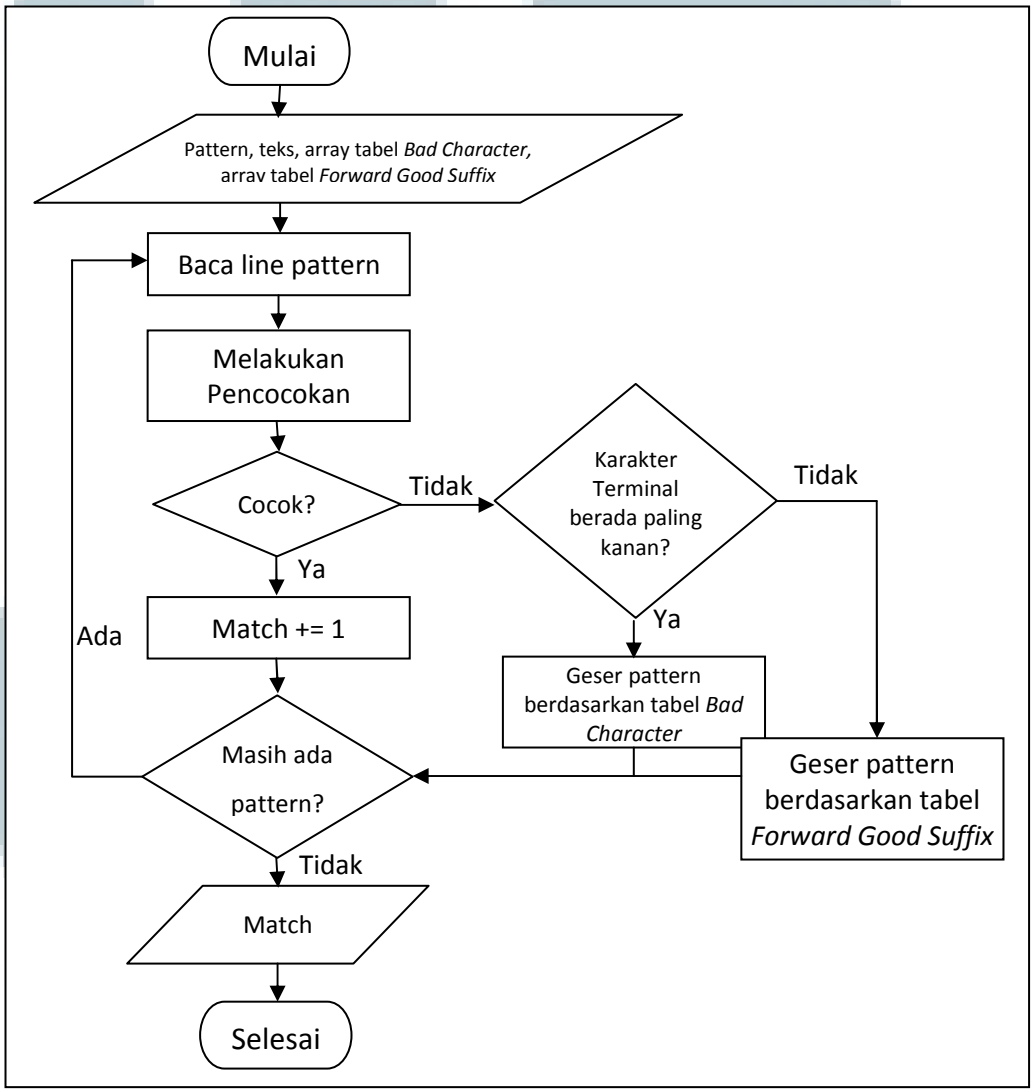



Gambar 3.10 *Flowchart Subroutine* Proses Penyusunan Tabel Pergeseran

## E. Tahap Perhitungan Persentase Kecenderungan Plagiarisme Dengan Algoritma Forward Fast Search

Tahapan proses ini bertujuan untuk membandingkan dua *token* dan menghitung persentase kecenderungan plagiarisme *pattern* terhadap *text*. Proses ini menerima dua *token*, dan tabel *Bad Character* dan *Forward Good Suffix* dari *token* yang berperan sebagai *pattern*. *Token* yang berperan sebagai *pattern* dipilih dari *token* yang lebih pendek.

Gambar 3.11 adalah *flowchart* dari proses perhitungan persentase kecenderungan plagiarisme dengan algoritma *forward fast search*.

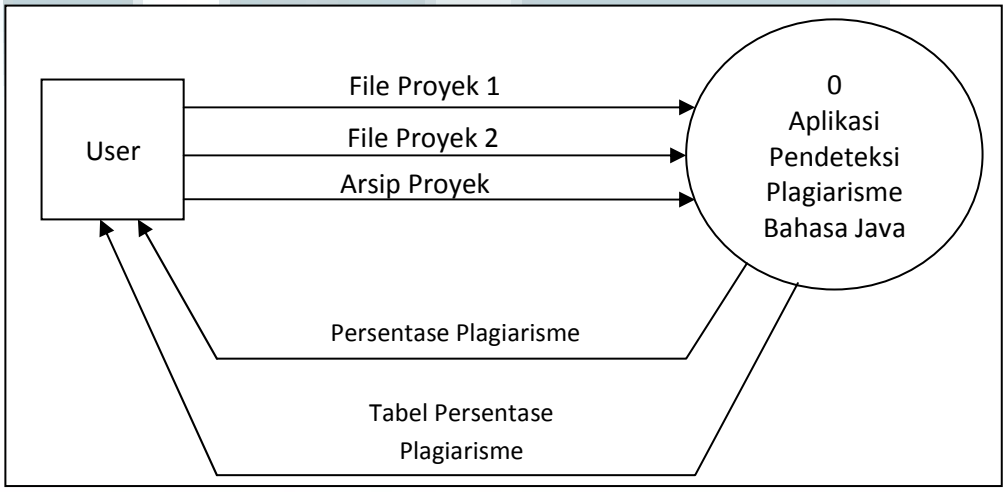



Gambar 3.11 *Flowchart* Perhitungan Persentase Kecenderungan Plagiarisme

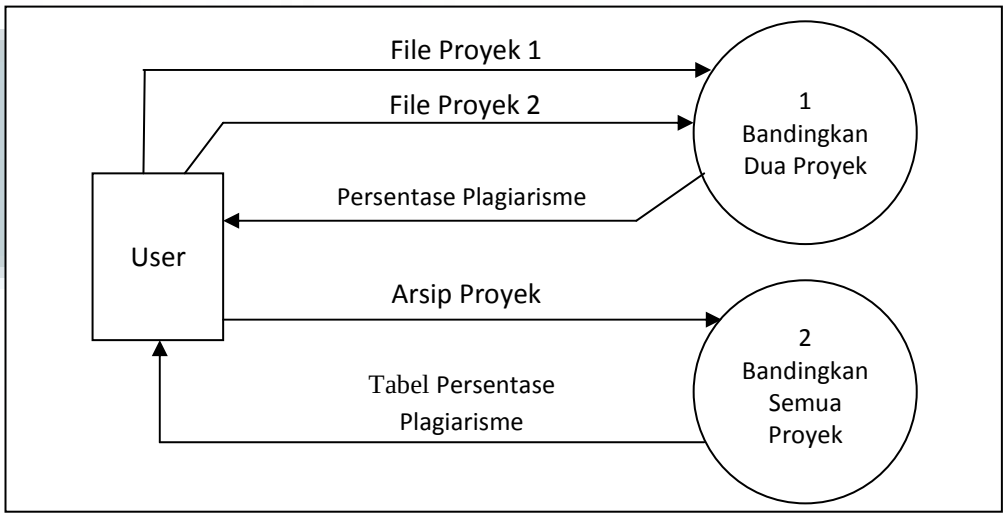
Hasil dari proses ini akan digunakan untuk mencari persentase kemiripan dengan menggunakan rumus statistika sederhana seperti yang sudah dibahas pada Bab II. Kode dengan persentase kemiripan yang terlalu tinggi ketika dibandingkan dengan persentase kemiripan rata-rata dapat dicurigai telah melakukan tindak plagiarisme dan butuh diperiksa lebih lanjut.

### 3.5.2 Data Flow Diagram

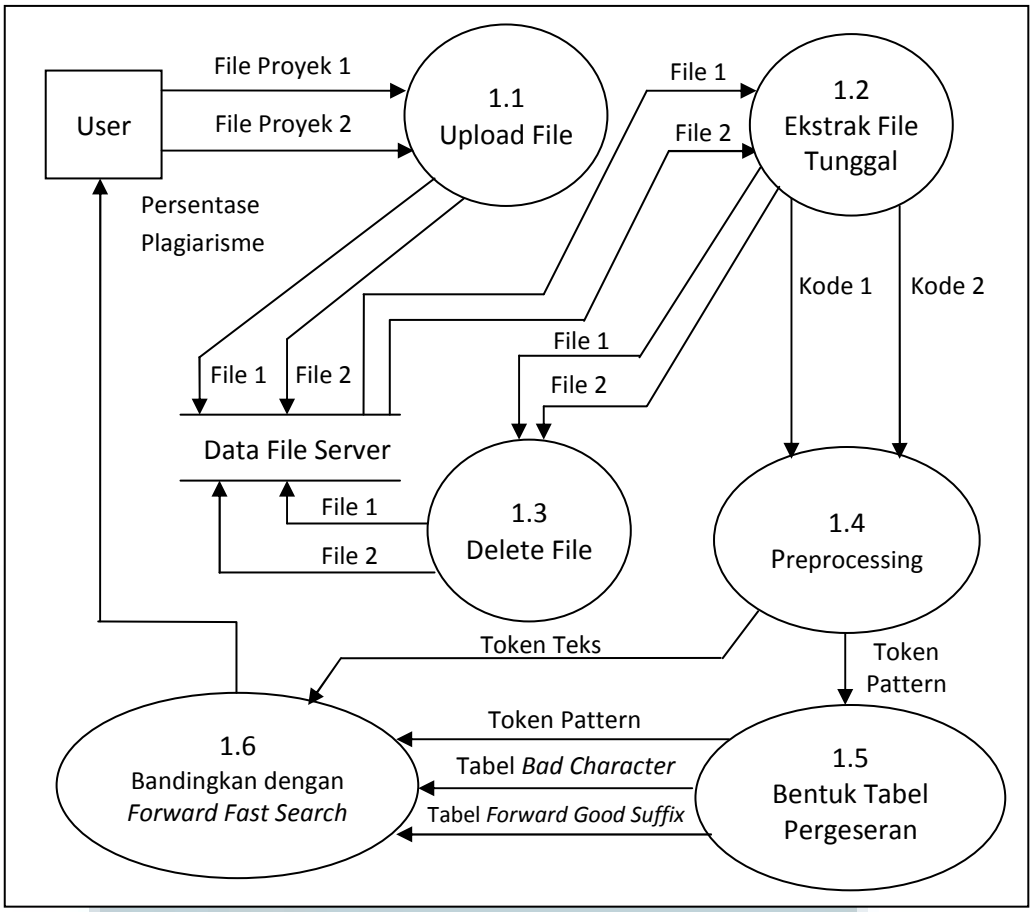
Berdasarkan data *input* dari *user* dan *output* dari sistem, dapat digambarkan *Data Flow Diagram* seperti yang terlihat pada gambar 3.12, gambar 3.13, gambar 3.14, dan gambar 3.15.



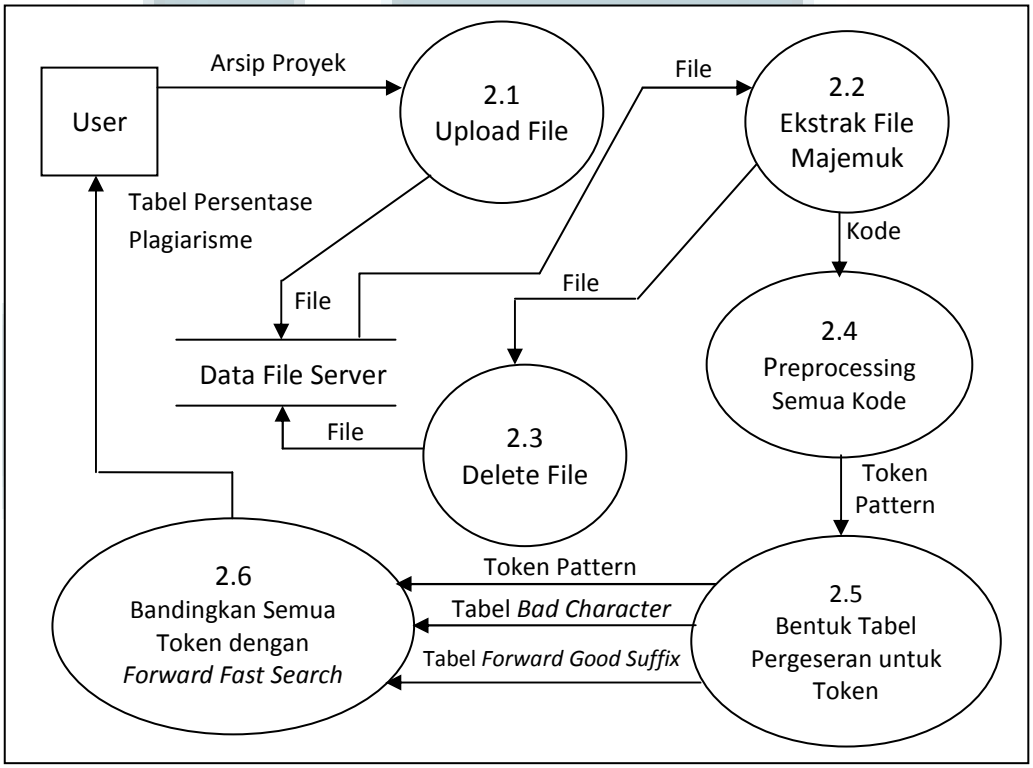

Gambar 3.12 *Context Diagram*




Gambar 3.13 *Data Flow Diagram Level* 1




Gambar 3.14 *Data Flow Diagram Level* 2 "Bandingkan Dua Proyek"

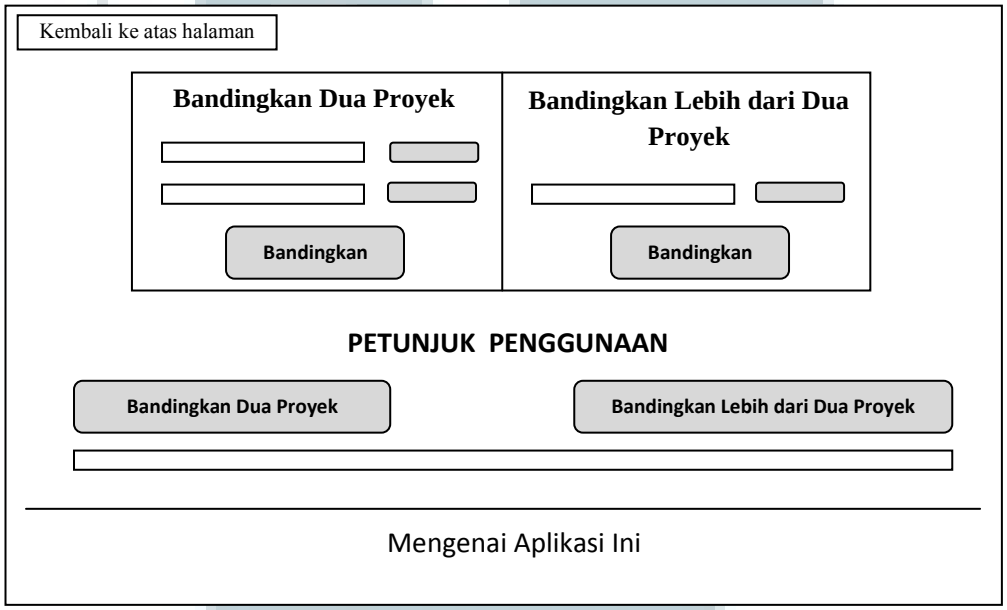



Gambar 3.15 *Data Flow Diagram Level* 2 "Bandingkan Semua Proyek"

### 3.5.3 Tampilan Antarmuka

Desain tampilan antarmuka untuk aplikasi ini dibagi menjadi 5, yaitu halaman utama, halaman proses untuk perbandingan dua proyek, halaman proses untuk perbandingan lebih dari dua proyek, halaman hasil untuk perbandingan dua proyek, dan halaman hasil untuk perbandingan lebih dari dua proyek.

Gambar 3.16 adalah desain tampilan antarmuka untuk halaman utama.



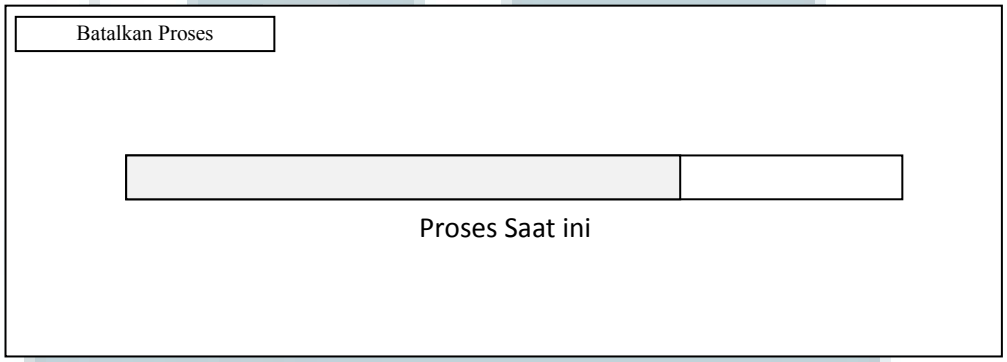

Gambar 3.16 Desain Antarmuka Halaman Utama

Pada bagian atas terdapat dua bagian untuk melakukan *upload file* sesuai dengan proses yang ingin dilakukan. Ketika tombol "Bandingkan" ditekan, maka proses *upload file* akan dilakukan, lalu halaman akan diarahkan ke halaman proses yang bersangkutan.

Pada bagian "Petunjuk Penggunaan" terdapat dua tombol untuk menampilkan petunjuk penggunaan pada kotak di bawahnya sesuai dengan tombol yang ditekan. Pada bagian paling bawah pada halaman utama terdapat keterangan mengenai aplikasi ini.

Pada bagian kiri atas, terdapat tombol untuk kembali ke bagian atas halaman. Tombol ini digunakan untuk mempermudah *user* mencapai bagian atas halaman tanpa perlu melakukan *scroll* secara manual ketika petunjuk penggunaan ditampilkan.

Gambar 3.17 adalah desain tampilan antarmuka untuk halaman proses perbandingan dua proyek.




Gambar 3.17 Desain Antarmuka Proses Perbandingan Dua Proyek

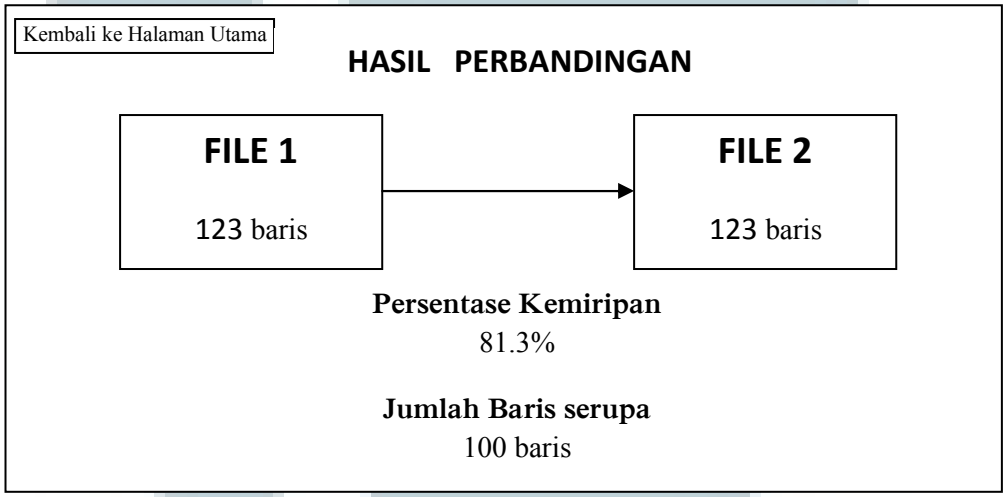
Pada halaman ini hanya terdapat sebuah *progress bar* yang menunjukkan seberapa jauh proses sudah berlangsung dan sebuah teks yang memberitahu proses yang sedang dilakukan pada bagian tengah, dan sebuah tombol untuk membatalkan proses pada bagian kiri atas.

Gambar 3.18 adalah desain tampilan antarmuka untuk halaman proses perbandingan lebih dari dua proyek.


Gambar 3.18 Desain Antarmuka Proses Perbandingan Lebih dari Dua Proyek

Pada bagian tengah halaman ini terdapat dua buah *progress bar*. *Progress bar* pertama menunjukkan berapa file yang sudah terproses pada proses yang sedang dilakukan, sementara *progress bar* kedua menunjukkan seberapa jauh proses sudah berlangsung dan proses yang sedang dilakukan. Selain itu, pada bagian kiri atas terdapat sebuah tombol untuk membatalkan proses.

Gambar 3.19 adalah desain tampilan antarmuka untuk halaman hasil perbandingan dua proyek.



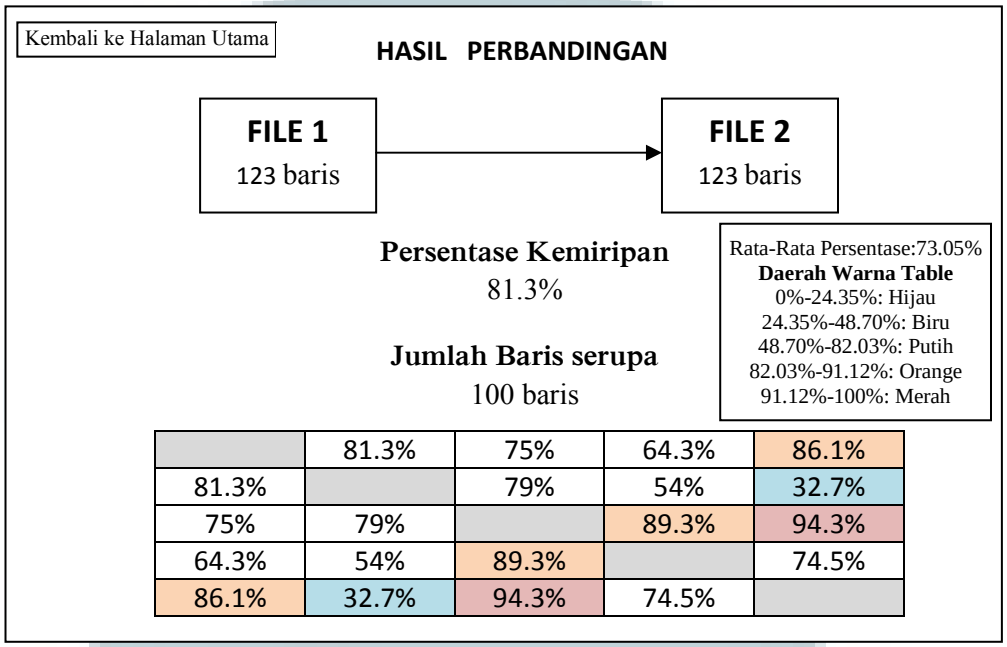

Gambar 3.19 Desain Tampilan Antarmuka Hasil Perbandingan Dua Proyek

Halaman ini menampilkan hasil dari perbandingan dua proyek kode program Java pada bagian tengah, dan sebuah tombol untuk kembali ke halaman utama pada bagian kiri atas. “File 1” adalah nama dari *file* yang berperan sebagai *pattern*, sedangkan “File 2” adalah nama dari *file* yang berperan sebagai teks. Di bawah nama *file* terdapat keterangan jumlah baris *token* dari kedua *file* tersebut yang dibandingkan. Selain nama dan jumlah baris dari kedua *file*, ditampilkan juga persentase kemiripan atau persentase kecenderungan plagiarisme dari perbandingan kedua kode tersebut, dan jumlah line yang serupa.

Ketika salah satu file yang dibandingkan memiliki *syntax error*, maka nama dari *file* yang bersangkutan akan ditampilkan dengan warna merah, jumlah baris

dari file tersebut akan bertuliskan “Error” dengan warna merah, dan persentase kemiripan serta jumlah line serupa akan memiliki nilai “N/A”.

Gambar 3.20 adalah desain tampilan antarmuka untuk halaman hasil perbandingan lebih dari dua proyek.




Gambar 3.20 Desain Antarmuka Hasil Perbandingan Lebih dari Dua Proyek

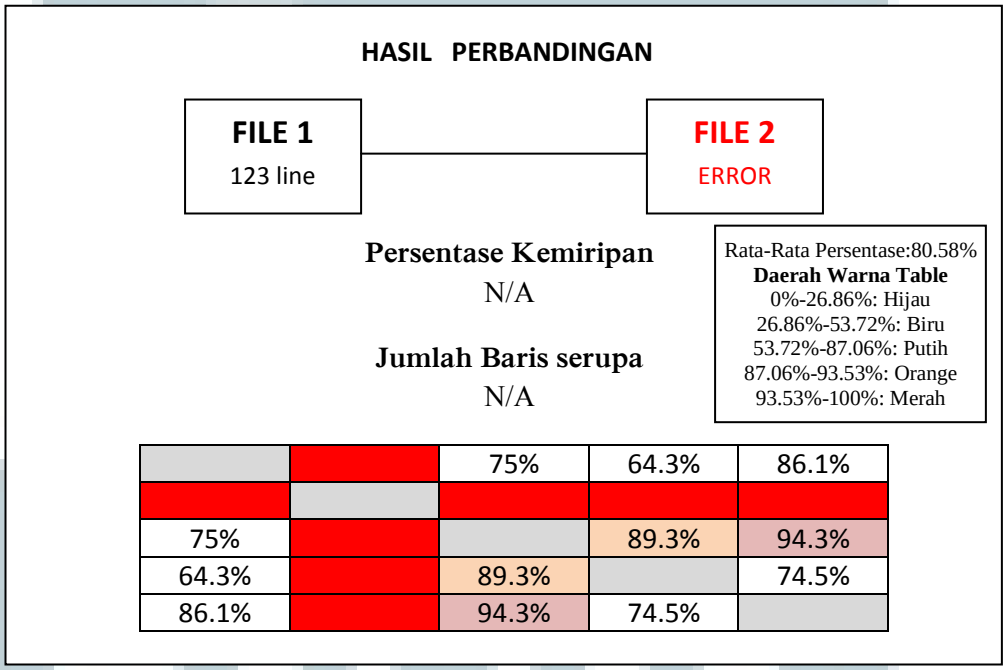
Bagian kiri atas dari halaman ini menampilkan tombol untuk kembali ke halaman utama. Batas atas tengah dari halaman ini menampilkan data perbandingan yang akan berubah ketika *user* mengarahkan *pointer mouse* ke salah satu *cell* dari tabel yang ingin dilihat datanya. Secara *default* data yang ditampilkan adalah data hasil perbandingan antara *file* yang berada di *index* pertama dan *index* kedua. Bagian kanan tengah menampilkan informasi nilai rata-rata dari seluruh hasil perbandingan, dan petunjuk mengenai warna pada tabel.

Bagian bawah dari halaman ini menampilkan tabel yang berisi nama *file* dan persentase kemiripan dari kedua *file* tersebut. Warna latar belakang pada sebuah *cell* akan berbeda-beda sesuai dengan nilai persentase yang tercantum. Urutan warna latar belakang dari nilai yang paling rendah adalah hijau (di bawah

0.3*rata-rata), biru (sampai dengan 0.6*rata-rata), putih (sampai dengan rata-rata+0.3*(100-rata-rata)), orange (sampai dengan rata-rata+0.6*(100-rata-rata)), dan merah (sampai dengan 100%).

Ketika ada file yang memiliki *syntax error*, nama dari *file* tersebut akan ditampilkan dengan warna merah beserta dengan jumlah baris yang bertuliskan "Error" berwarna merah, dan juga persentase kemiripan dan jumlah baris serupa akan memiliki nilai "N/A". Pada tabel, bagian yang mencantumkan perbandingan dengan *file* yang memiliki *syntax error* tersebut akan berwarna merah tanpa ada persentase kemiripan yang tertulis.

Gambar 3.21 adalah desain tampilan antarmuka hasil perbandingan dengan *file* yang memiliki *syntax error*.




Gambar 3.21 Desain Antarmuka Hasil Perbandingan Lebih dari Dua Proyek Dengan Kode yang Memiliki *Syntax Error*