TUGAS AKHIR - KI141502

Rancang Bangun Aplikasi Konversi Diagram Alir Menjadi Problem Analysis Diagram untuk Pembangkitan Kode Sumber

Risyanggi Azmi Faizin
NRP 5111200113

Dosen Pembimbing
Dwi Sunaryono, S.Kom., M.Kom.
Rizky Januar Akbar, S.Kom., M.Eng.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2016

[Halaman ini sengaja dikosongkan]

FINAL PROJECT - KI141502

Software Design and Implementation of Flowchart Conversion Into Problem Analysis Diagram for Source Code Generation

Risyanggi Azmi Faizin
NRP 5111200113

Advisor
Dwi Sunaryono, S.Kom., M.Kom.
Rizky Januar Akbar, S.Kom., M.Eng.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2016

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

Rancang Bangun Aplikasi Konversi Diagram Alir menjadi Problem Analysis Diagram untuk Pembangkitan Kode Sumber

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Algoritma dan Pemrograman
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
Risyanggi Azmi Faizin
NRP : 5112 100 113

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Dwi Sunaryono, S.Kom., M.Kom. .....
NIP: 197205281997021001 (pembimbing 1)

Rizky Januar Akbar, S.Kom., M.Eng. .....
NIP: 198701032014041001 (pembimbing 2)

SURABAYA
JUNI 2016

[Halaman ini sengaja dikosongkan]

Rancang Bangun Aplikasi Konversi Diagram Alir Menjadi Problem Analysis Diagram untuk Pembangkitan Kode Sumber

Nama Mahasiswa : Risyanggi Azmi Faizin
NRP : 5111200113
Jurusan : Teknik Informatika FTIf-ITS
Dosen Pembimbing 1 : Dwi Sunaryono, S.Kom., M.Kom.
Dosen Pembimbing 2 : Rizky Januar Akbar, S.Kom, M.Eng.

ABSTRAK

Salah satu langkah dalam merancang sebuah program adalah dengan membuat diagram alir. Diagram alir memudahkan analis dan programmer memecah masalah menjadi bagian yang lebih kecil untuk memudahkan analisis dan evaluasi lebih lanjut. Namun merealisasikan program dari diagram alir tidak mudah dan butuh waktu yang tidak sedikit.

Diagram alir memiliki simbol yang dapat direalisasikan ke kode sumber. Namun untuk mengkonversi diagram alir menjadi kode sumber, dibutuhkan sebuah bentuk diagram yang dapat menjadi perantara diagram alir dan kode sumber, yaitu problem analysis diagram yang bisa disingkat PAD. PAD lebih menggambarkan alur dan hirarki program karena struktur dasar dari kode sumber sudah diwakilkan oleh simbol dasar pada PAD.

Untuk mengimplementasi metode konversi diagram alir menjadi kode sumber menggunakan perantara PAD, maka dibuatlah aplikasi konversi diagram alir menjadi kode sumber. Aplikasi tersebut memiliki fitur membuat diagram alir, memvalidasi diagram alir, dan mengkonversi diagram alir menjadi kode sumber.

Kata kunci**: **Diagram Alir, Flowchart, Problem Analysis Diagram, Source Code, Kode Sumber.

[Halaman ini sengaja dikosongkan]

Software Design and Implementation of Flowchart Conversion Into Problem Analysis Diagram for Source Code Generation

Student Name	: Risyanggi Azmi Faizin
Student ID	: 5111200113
Major	: Teknik Informatika FTIf-ITS
Advisor 1	: Dwi Sunaryono, S.Kom., M.Kom.
Advisor 2	: Rizky Januar Akbar, S.Kom, M.Eng.

ABSTRACT

One of possible method in building software is making a flowchart. Flowchart helps analysts and programmers break problems into smaller pieces for easier further analysis and evaluation. But converting flowchart into software is not easy and need amount of time.

Flowchart has symbols that can be realized in source code. To convert flowchart into source code, it needs another form of a diagram which can connect flowchart and source code, that is problem analysis diagram, also as known as PAD. PAD can describe order and hierarchy of program better because basic structure of source code are represented by basic symbols of PAD.

In purpose to implement the conversion of flowchart into source code using the PAD intermediary method, then a flowchart into source code converter application is proposed. The application has features to create flowcharts, validate flowcharts, and convert flowcharts into source codes.

Keywords**: **Flowchart, Problem Analysis Diagram, Source Code

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan tugas akhir yang berjudul "**Rancang Bangun Aplikasi Konversi Diagram Alir Menjadi Problem Analysis Diagram untuk Pembangkitan Kode Sumber**".

Harapan dari penulis semoga apa yang tertulis di dalam buku tugas akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini, serta dapat memberikan kontribusi yang nyata.

Dalam pelaksanaan dan pembuatan tugas akhir ini tentunya sangat banyak bantuan yang penulis terima dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Allah SWT dan Nabi Muhammad SAW.
2. Bapak, Ibuk, dan keluarga yang selalu memberikan dukungan, baik moral maupun material, secara penuh untuk menyelesaikan tugas akhir ini.
3. Bapak Dwi Sunaryono selaku dosen pembimbing pertama yang telah bersedia meluangkan waktu untuk memberikan petunjuk selama proses pengerjaan tugas akhir ini.
4. Bapak Rizky Januar Akbar selaku dosen pembimbing kedua yang telah bersedia meluangkan waktu untuk memberikan petunjuk selama proses pengerjaan tugas akhir ini. Tanpa bimbingan rutin setiap minggu mungkin tugas akhir ini akan semakin sulit selesai.
5. Bapak, Ibu dosen Jurusan Teknik Informatika ITS yang telah banyak memberikan ilmu dan bimbingan yang tak ternilai harganya bagi penulis.
6. Eric Ivander Jeadi sebagai teman dekat yang memberikan ide, mendengarkan isi curatan hati penulis, dan membantu menyelesaikan tugas akhir ini.
7. Teman-teman penghuni laboraturium DTK yang selalu memberi dorongan dan inspirasi kepada penulis.

8. Teman-teman TC 2012 yang tidak dapat disebutkan satu per satu bantuannya kepada penulis.
9. Serta pihak-pihak lain yang tidak dapat disebutkan disini yang telah banyak membantu penulis dalam penyusunan tugas akhir ini.

Penulis telah berusaha sebaik-baiknya dalam menyusun tugas akhir ini, namun penulis mohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya.

Surabaya, 13 Juni 2015
Penulis

Risyanggi Azmi Faizin

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar tugas akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan tugas akhir, dan sistematika penulisan.

1.1. Latar Belakang

Dewasa ini, teknologi informasi berkembang dengan sangat pesat. Hampir seluruh aspek kehidupan menggunakan produk teknologi informasi untuk mempermudah kegiatannya. Salah satu produk teknologi informasi yang umum digunakan adalah aplikasi.

Salah satu langkah dalam merancang sebuah aplikasi adalah dengan membuat diagram alir. Diagram alir memudahkan analis dan programmer memecah masalah menjadi bagian yang lebih kecil untuk memudahkan analisis dan evaluasi lebih lanjut. Namun merealisasikan aplikasi dari diagram alir tidak mudah dan butuh waktu yang tidak sedikit.

Diagram alir memiliki simbol yang dapat direalisasikan ke kode sumber. Simbol process yang menjelaskan proses dapat diubah ke dalam bentuk *statement* pada kode sumber. Simbol decision yang menjelaskan percabangan alur dapat diubah ke dalam bentuk *loop* atau *if* pada kode sumber. Dari karakteristik tersebut, penulis menawarkan sebuah solusi membuat aplikasi yang dapat melakukan konversi diagram alir menjadi kode sumber sehingga dapat meringankan pekerjaan programmer.

Untuk mengkonversi diagram alir menjadi kode sumber, dibutuhkan sebuah bentuk diagram yang dapat menjadi perantara diagram alir dan kode sumber, yaitu *problem analysis diagram* yang bisa disingkat PAD. PAD lebih menggambarkan alur dan hirarki program karena bentuk dasar dari kode sumber sudah diwakilkan oleh simbol dasar pada PAD [1].

1.2. Rumusan Permasalahan

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana membuat aplikasi untuk merancang diagram alir?
2. Bagaimana memeriksa keabsahan diagram alir?
3. Bagaimana mengubah diagram alir menjadi Problem Analysis Diagram?
4. Bagaimana mengubah Problem Analysis Diagram menjadi kode sumber?

1.3. Batasan Permasalahan

Beberapa batasan masalah yang terdapat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Implementasi tugas akhir ini menggunakan bahasa pemrograman Java.
2. Kode sumber keluaran aplikasi ini dalam bahasa pemrograman C dengan tipe data primitif.
3. Aplikasi hanya dapat mengolah satu rangkaian diagram alir dan menghasilkan satu fungsi main.
4. Keluaran sumber kode hasil konversi perlu ditambahkan deklarasi variabel-variabel yang ada di diagram alir.
5. Pustaka kode sumber yang bukan standar *input output* harus ditambahkan secara manual.

1.4. Tujuan

Tujuan dari pembuatan tugas akhir ini adalah sebagai berikut:

1. Membuat aplikasi untuk merancang diagram alir.
2. Melakukan pembangkitan kode sumber dari diagram alir.
3. Mengetahui bentuk PAD dari diagram alir.
4. Mengetahui keabsahan diagram alir.

1.5. Manfaat

Manfaat dari hasil pembuatan tugas akhir antara lain:

1. Membantu pekerjaan programmer untuk merealisasikan diagram alur menjadi kode sumber.
2. Membantu pembelajaran pemrograman untuk pemula.

1.6. Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan tugas akhir adalah sebagai berikut:

1.6.1. Penyusunan Proposal Tugas Akhir

Proposal tugas akhir ini berisi mengenai rencana pengembangan aplikasi konversi diagram alir menjadi kode sumber. Pendahuluan ini terdiri atas hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah untuk tugas akhir, tujuan dari pembuatan tugas akhir, dan manfaat dari hasil pembuatan tugas akhir. Selain itu dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan tugas akhir. Subbab metodologi berisi penjelasan mengenai tahapan penyusunan tugas akhir mulai dari penyusunan proposal hingga penyusunan buku tugas akhir. Terdapat pula subbab jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.

1.6.2. Studi literatur

Pada studi literatur ini, akan dipelajari sejumlah referensi yang diperlukan dalam pembuatan aplikasi yaitu mengenai diagram alir dan *problem analysis diagram*, algoritma untuk mengubah diagram alir menjadi kode sumber, tools yang digunakan, dan pustaka yang dibutuhkan.

1.6.3. Analisis dan Perancangan Sistem

Pada tahap ini dilakukan analisis dan pendefinisian kebutuhan sistem untuk masalah yang sedang dihadapi. Selanjutnya, dilakukan perancangan sistem dengan beberapa tahap sebagai berikut:

a. Perancangan proses aplikasi.
b. Perancangan antarmuka sistem.
c. Perancangan diagram kelas.
d. Perancangan algoritma mengubah diagram alir menjadi kode sumber.

1.6.4. Implementasi

Pada tahap ini dilakukan pembuatan elemen perangkat lunak. Sistem yang dibuat berpedoman pada rancangan yang telah dibuat pada proses perancangan dan analisis sistem. Pada proses implementasi algoritma mengubah diagram alir menjadi kode sumber terdapat tiga sub algoritma:

1. Pemberian kode untuk setiap elemen diagram alir.
2. Mengubah diagram alir yang sudah diberi kode menjadi *problem analysis diagram*.
3. Mengubah *problem analysis diagram* menjadi kode sumber

1.6.5. Pengujian dan Evaluasi

Pada tahap ini dilakukan pengujian aplikasi kepada pengguna secara langsung. Pengujian dan evaluasi perangkat dilakukan untuk mengevaluasi hasil analisis program. Tahapan-tahapan dari pengujian adalah sebagai berikut:

1. Pengujian fitur-fitur yang ada.
2. Pengujian aplikasi terhadap beberapa soal pemrograman berbasis algoritma.

1.6.6. Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan pendokumentasian dan pelaporan dari seluruh konsep, dasar teori, implementasi, proses yang telah dilakukan, dan hasil-hasil yang telah didapatkan selama pengerjaan tugas akhir.

1.7. Sistematika Penulisan

Buku tugas akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan tugas akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku tugas akhir terdiri atas beberapa bagian seperti berikut ini.

Bab I Pendahuluan
Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan tugas akhir, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penyusunan tugas akhir.

Bab II Tinjauan Pustaka
Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan mendasari pembuatan tugas akhir ini.

Bab III Analisis dan Perancangan Sistem
Bab ini membahas mengenai perancangan perangkat lunak. Perancangan perangkat lunak meliputi perancangan data, arsitektur, proses dan perancangan antarmuka pada aplikasi.

Bab IV Implementasi
Bab ini berisi implementasi dari perancangan perangkat lunak.

Bab V Pengujian dan Evaluasi
Bab ini membahas pengujiandengan metode pengujian subjektif untuk mengetahui penilaian aspek kegunaan (*usability*) dari perangkat lunak dan pengujian hasil

analisis kakas.

Bab VI Kesimpulan dan Saran

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan. Bab ini membahas saran-saran untuk pengembangan sistem lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan tugas akhir.

Lampiran

Merupakan bab tambahan yang berisi daftar istilah yang penting pada aplikasi ini.

BAB II
TINJAUAN PUSTAKA

2.1. Diagram Alir

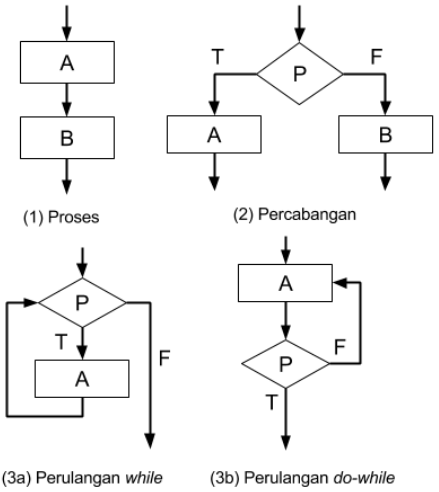
Diagram alir, atau lebih dikenal dengan sebutan *flowchart*, adalah diagram yang merepresentasikan definisi, analisis, ataupun metode dari solusi sebuah masalah. Simbol yang digunakan dalam flowchart harus mewakili operasi, data, alir, dan lain-lain. Tabel 2.1 berisi simbol-simbol diagram alir [2].

Tabel 2.1 Daftar Simbol Diagram Alir

Simbol	Nama	Deskripsi
	Flow Line	Panah yang mengarah dari simbol ke simbol lain, menggambarkan alir.
	On-Page Connector	Connector disimbolkan dengan lingkaran, yang di dalam lingkaran tersebut ada sebuah angka atau huruf yang mewakili identifikasi connector. Ada dua tipe dari connector ini. Connector dari flow line atau connector menuju flow line. Alir yang menuju ke connector akan dilanjutkan dari connector yang menuju flow line. Connector dari flow line harus berjumlah satu, sedangkan flow menuju flow line boleh berjumlah lebih dari satu.
	Terminal	Terminal berbentuk persegi dengan ujung kanan dan kiri berbentuk setengah lingkaran. Biasanya isinya hanya tulisan “start” atau “end”, menunjukkan awal atau akhir dari flowchart.

Tabel 2.2 Daftar Simbol Diagram Alir (lanjutan)

Simbol	Nama	Deskripsi
	Decision	Decision berbentuk belah ketupat, yang membagi alir menjadi dua pilihan. Tulisan dalam decision biasanya pertanyaan yang hasilnya benar atau salah. Panah yang berasal dari decision harus bertuliskan “yes” atau “no”.
	Input / Output	Jajar genjang ini mewakili input atau output data. Misalnya tampilkan nilai atau dapatkan nilai dari user.
	Predefined Process	Permulaan sub program atau proses menjalankan sub program.
	Process	Menjalankan proses perhitungan, pengolahan data, dll.



Gambar 2.1 Struktur Dasar Diagram Alir

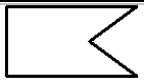
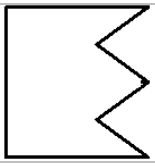
Kombinasi dari elemen diagram alir dapat dibagi lagi menjadi tiga macam struktur dasar: proses, percabangan, dan perulangan yang dapat dilihat pada Gambar 2.1. Dari ketiga struktur tersebut, diagram alir dapat membuat berbagai macam algoritma yang sederhana maupun rumit.

2.2. Problem Analysis Diagram

Problem Analysis Diagram (PAD) adalah diagram yang digunakan untuk menggambarkan proses algoritma atau alur ketika membuat sebuah aplikasi. PAD adalah diagram evolusi dari diagram alir yang bersifat dua dimensi karena menggambarkan proses dari atas ke bawah dan kiri ke kanan. Pada dasarnya, diagram alir dan PAD dapat diubah ke dan dari satu sama lain. Tapi PAD lebih menggambarkan alur dan hirarki program dibandingkan diagram alir sehingga PAD digunakan untuk mempermudah proses mengubah diagram alir menjadi kode sumber [1].

Tabel 2.2 berisi simbol-simbol yang digunakan dalam PAD [3].

Tabel 2.3 Daftar Simbol Problem Analysis Diagram

Simbol	Nama	Deskripsi
	Terminal	Menunjukkan awal atau akhir dari sebuah PAD
	Process	Menjalankan proses
	Selection	Pilihan ya atau tidak. Jika ya maka proses berjalan ke atas, jika tidak maka berjalan ke bawah
	Multi selection	Pilihan yang lebih dari beberapa kondisi

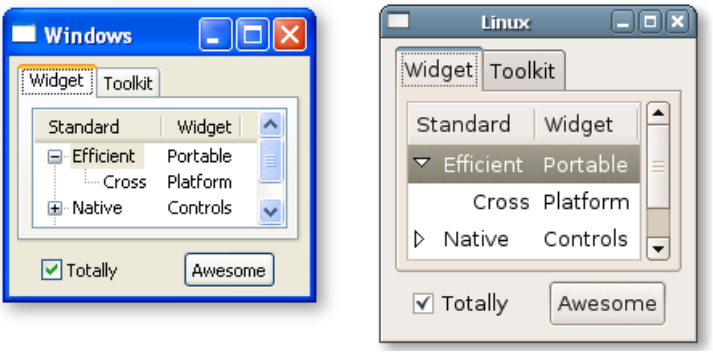
Tabel 2.4 Daftar Simbol Problem Analysis Diagram (lanjutan)

Simbol	Nama	Deskripsi
	Pre-decision loop	Perulangan while
	Post-decision loop	Perulangan do-while
	Subprogram	Permulaan sub program atau proses menjalankan sub program.
	Line	Menghubungkan antar proses

2.3. Java SWT

Standard Widget Tookit atau yang biasa disingkat SWT adalah pustaka perangkat lunak berbasis GUI (*Graphical User Interface*) untuk Java. Perbedaan dengan pustaka GUI yang lain adalah SWT berjalan lebih cepat, menggunakan *memory* dengan efektif, dan tampilan antarmuka yang menyesuaikan sistem operasi dimana aplikasi itu dijalankan [4].

Gambar 2.2 menunjukkan tampilan antarmuka pada Windows XP dan Linux. Gambar kiri adalah antarmuka di Windows XP dan gambar kanan adalah antarmuka di Linux.



Gambar 2.2 Contoh Tampilan Antarmuka Aplikasi

2.4. Another Tool for Language Recognition

Another Tool for Language Recognition (ANTLR) adalah tool untuk membaca, pengolahan, melaksanakan, atau menerjemahkan teks atau file biner. Ini banyak digunakan untuk membangun bahasa pemrograman, aplikasi, atau kerangka kerja. Dari pustaka bahasa, ANTLR dapat membangun *parse tree* [5].

Pustaka ANTLR digunakan untuk mengecek keabsahan sintaks pada kode sumber dengan menggunakan *grammar* dari bahasa kode sumber yang akan dihasilkan. Tugas akhir ini menghasilkan kode sumber berbahasa C, maka *grammar* yang dibutuhkan juga berbahasa C.

[Halaman ini sengaja dikosongkan]

BAB III
ANALISIS DAN PERANCANGAN SISTEM

Bab ini menjelaskan tentang analisis dan perancangan aplikasi konversi diagram alir menjadi kode sumber. Pembahasan yang akan dilakukan meliputi analisis fitur yang dibutuhkan dan perancangan perangkat lunak.

3.1. Analisis Perangkat Lunak

Subbab ini menjelaskan tentang hasil analisis kebutuhan perangkat lunak serta arsitektur aplikasi konversi diagram alir menjadi kode sumber.

3.1.1. Deskripsi Umum Perangkat Lunak

Pada tugas akhir ini dilakukan pengembangan aplikasi konversi diagram alir menjadi kode sumber berbasis desktop. Pengguna berinteraksi dengan aplikasi menggunakan keyboard dan mouse. Aplikasi ini akan mengkonversi diagram alir yang dibuat oleh user menjadi kode sumber berbahasa C dengan pendekatan *Problem Analysis Diagram*.

3.1.2. Kebutuhan Fungsional

Pada aplikasi ini, terdapat beberapa kebutuhan fungsional yang mendukung untuk jalannya aplikasi. Fungsi yang terdapat dalam aplikasi ini adalah sebagai berikut:

a. Membuat diagram alir
   Pengguna dapat membuat diagram alir menggunakan tool-tool yang disediakan pada aplikasi.
b. Memvalidasi diagram alir
   Pengguna dapat mengetahui keabsahan diagram alir.
c. Menyimpan dan membuka diagram alir
   Pengguna dapat menyimpan diagram alir yang sudah dibuat ke dalam file dan membuka kembali file tersebut.

d. Mengkonversi diagram alir menjadi kode sumber
   Aplikasi ini mengubah diagram alir ke kode sumber
e. Melihat *problem analysis diagram*
   Pengguna dapat melihat bentuk PAD dari diagram alir yang dibuat.

3.1.3. Identifikasi Pengguna

Dalam aplikasi ini hanya memiliki satu aktor, yaitu orang yang menggunakan aplikasi konversi diagram alir menjadi kode sumber.

3.2. Perancangan Perangkat Lunak

Subbab ini membahas bagaimana rancangan dari aplikasi tugas akhir ini. Meliputi: Model Kasus Penggunaan, Definisi Aktor, Definisi Kasus Penggunaan, Arsitektur Umum Sistem, Rancangan Antarmuka Aplikasi, dan Rancangan Proses Aplikasi.

3.2.1. Model Kasus Penggunaan

Berdasarkan analisis spesifikasi kebutuhan fungsional dan analisis aktor dari sistem dibuat kasus penggunaan sistem. Kasus-kasus penggunaan dalam sistem ini akan dijelaskan secara rinci pada subbab ini. Kasus penggunaan digambarkan dalam sebuah tabel kasus penggunaan Tabel 3.1.

Tabel 3.1 Tabel Kasus penggunaan

Kode Kasus Penggunaan	Nama
UC-0001	Membuat diagram alir
UC-0002	Memvalidasi diagram alir
UC-0003	Menyimpan file diagram alir

UC-0004	Membuka file diagram alir
UC-0005	Mengkonversi diagram alir menjadi kode sumber
UC-0006	Melihat *problem analysis diagram*

3.2.2. Definisi Kasus Penggunaan

Detail mengenai kasus penggunaan tersebut dapat dilihat pada subbab berikut ini.

3.2.2.1. Membuat Diagram Alir

Spesifikasi kasus penggunaan membuat diagram alir dapat dilihat pada Tabel 3.2.

Tabel 3.2 Spesifikasi Kasus Membuat Diagram Alir

Nama	Membuat Diagram Alir
Kode	UC-0001
Deskripsi	Pengguna dapat membuat diagram alir menggunakan tool-tool yang disediakan pada aplikasi.
Aktor	Pengguna
Kondisi Awal	-
Aliran Kejadian Normal	1. Pengguna pilih menu “file”, lalu pilih “new” 2. Pengguna membuat diagram menggunakan tool
Kondisi Akhir	Terdapat diagram yang dibuat oleh pengguna pada aplikasi

3.2.2.2. Memvalidasi Diagram Alir

Spesifikasi kasus penggunaan memvalidasi diagram alir dapat dilihat pada Tabel 3.3.

Tabel 3.3 Spesifikasi Kasus Memvalidasi Diagram Alir

Nama	Memvalidasi Diagram Alir
Kode	UC-0002
Deskripsi	Pengguna mengecek keabsahan diagram alir
Aktor	Pengguna
Kondisi Awal	Sudah ada diagram alir yang sedang terbuka
Aliran Kejadian Normal	1. Pengguna pilih menu “diagram”, lalu pilih “validate”
Kondisi Akhir	1. Muncul status keberhasilan validasi di bagian bawah 2. List validasi terisi dengan diagram yang salah

3.2.2.3. Menyimpan file diagram alir

Spesifikasi kasus penggunaan menyimpan file diagram alir dapat dilihat pada Tabel 3.4.

Tabel 3.4 Spesifikasi Kasus Menyimpan File Diagram Alir

Nama	Menyimpan File Diagram Alir
Kode	UC-0003
Deskripsi	Pengguna dapat menyimpan diagram alir yang sudah dibuat ke file.

Aktor	Pengguna
Kondisi Awal	Sudah ada diagram alir yang sedang terbuka
Aliran Kejadian Normal	1. Pengguna pilih menu “file”, lalu pilih “save” 2. Muncul dialog simpan file 3. Pengguna memilih tempat menyimpan file 4. Pengguna menekan tombol “save” 5. Diagram alir akan tersimpan ke file yang dipilih pengguna
Aliran Kejadian Alternatif	1A. Pengguna menekan tombol “cancel” 1. Diagram alir tidak tersimpan ke file
Kondisi Akhir	Dialog simpan file tertutup

3.2.2.4. Membuka file diagram alir

Spesifikasi kasus penggunaan membuka file diagram alir dapat dilihat pada Tabel 3.5.

Tabel 3.5 Spesifikasi Kasus Membuka File Diagram Alir

Nama	Membuka File Diagram Alir
Kode	UC-0004
Deskripsi	Pengguna dapat membuka kembali diagram alir yang sudah disimpan ke file.
Aktor	Pengguna
Kondisi Awal	-

Aliran Kejadian Normal	1. Pengguna pilih menu “file”, lalu pilih “open” 2. Muncul dialog buka file 3. Pengguna memilih file diagram alir 4. Pengguna menekan tombol “open” 5. Diagram alir dari file akan terbuka
Aliran Kejadian Alternatif	1A. Pengguna menekan tombol “cancel”
Kondisi Akhir	Dialog buka file tertutup

3.2.2.5. Mengkonversi Diagram Alir Menjadi Kode Sumber

Spesifikasi kasus penggunaan mengkonversi diagram alir menjadi kode sumber dapat dilihat pada Tabel 3.6.

Tabel 3.6 Spesifikasi Kasus Mengkonversi Diagram Alir Menjadi Kode Sumber

Nama	Mengkonversi Diagram Alir Menjadi Kode Sumber
Kode	UC-0005
Deskripsi	Pengguna dapat mengubah diagram alir yang sebelumnya dibuat menjadi kode sumber.
Aktor	Pengguna
Kondisi Awal	Sudah ada diagram alir yang sedang terbuka dan sudah melakukan validasi
Aliran Kejadian Normal	1. Pengguna pilih menu “diagram”, lalu pilih “generate code” 2. Muncul dialog simpan file 3. Pengguna memilih tempat menyimpan file kode sumber

	4. Pengguna menekan tombol “save” 5. Kode sumber akan tersimpan ke file yang dipilih pengguna
Aliran Kejadian Alternatif	1A. Pengguna menekan tombol “cancel” 1. File diagram alir tidak terbuka
Kondisi Akhir	-

3.2.2.6. Melihat Problem Analysis Diagram

Spesifikasi kasus penggunaan mengkonversi diagram alir menjadi kode sumber dapat dilihat pada Tabel 3.7.

Tabel 3.7 Spesifikasi Kasus Melihat Problem Analysis Diagram

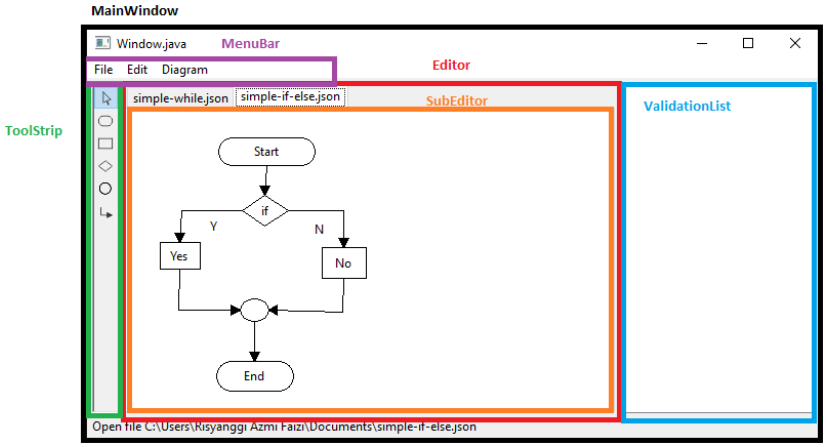
Nama	Melihat Problem Analysis Diagram
Kode	UC-0006
Deskripsi	Pengguna dapat melihat PAD hasil konversi dari diagram alir yang sebelumnya dibuat.
Aktor	Pengguna
Kondisi Awal	Sudah ada diagram alir yang sedang terbuka dan sudah melakukan validasi
Aliran Kejadian Normal	1. Pengguna pilih menu “diagram”, lalu pilih “show PAD” 2. Muncul halaman baru berisi *problem analysis diagram*
Kondisi Akhir	-

3.2.3. Arsitektur Umum Sistem

Arsitektur sistem pada aplikasi konversi diagram alir menjadi kode sumber didukung oleh beberapa perangkat yaitu komputer, *mouse* dan *keyboard*.

3.2.4. Rancangan Antarmuka Aplikasi

Rancangan antarmuka aplikasi diperlukan untuk memberikan gambaran umum kepada pengguna bagaimana sistem yang ada dalam aplikasi ini berinteraksi dengan pengguna. Selain itu, rancangan ini juga memberikan gambaran bagi pengguna apakah tampilan yang sudah disediakan oleh aplikasi mudah untuk dipahami dan digunakan, sehingga akan muncul kesan *user experience* yang baik dan mudah.



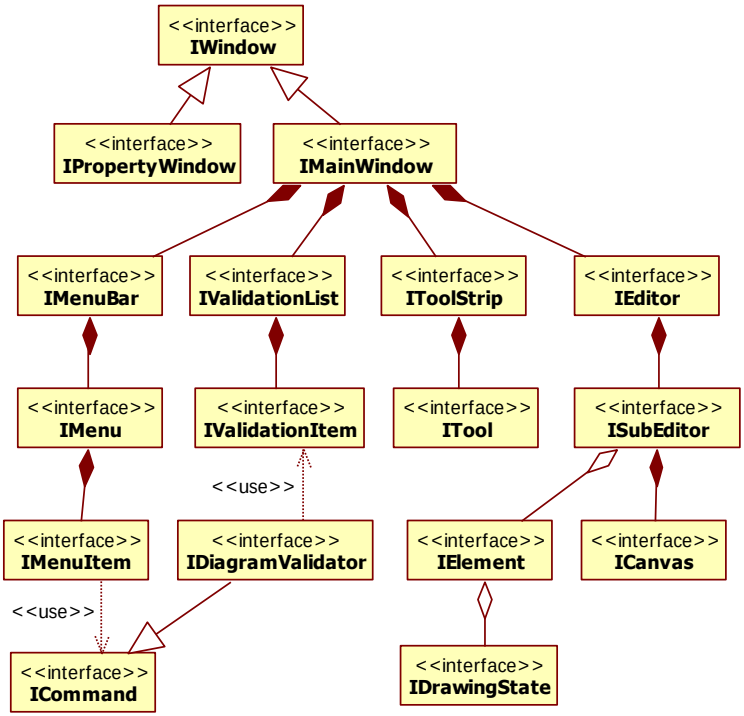
Gambar 3.1 Tampilan Antarmuka Aplikasi

Rancangan antarmuka terlihat pada Gambar 3.1. Elemen antarmuka jika dikaitkan dengan diagram kelas pada Gambar 3.2:

a. Main Window
   Tampilan utama aplikasi, kelas IMainWindow
b. Menu Bar

Berisi pilihan menu dalam aplikasi, kelas IMenuBar

c. Editor
   Berisi tab untuk membuat diagram, kelas IEditor
d. Sub Editor
   Tab yang berisi canvas untuk membuat diagram, kelas ISubEditor
e. Tool Strip
   Tool untuk membuat diagram, kelas IToolStrip
f. Validation List
   List validasi yang gagal setelah pengguna melakukan validasi, kelas IValidationList



Gambar 3.2 Diagram kelas rancangan antarmuka

3.2.5. Rancangan Diagram Kelas Aplikasi

Rancangan diagram kelas berisi kelas *interface* yang belum diimplementasi beserta hubungan dengan kelas *interface* lain pembentuk aplikasi. Diagram kelas dapat dilihat pada Gambar 3.2. Subbab di bawah ini akan menjelaskan detail setiap kelas rancangan.

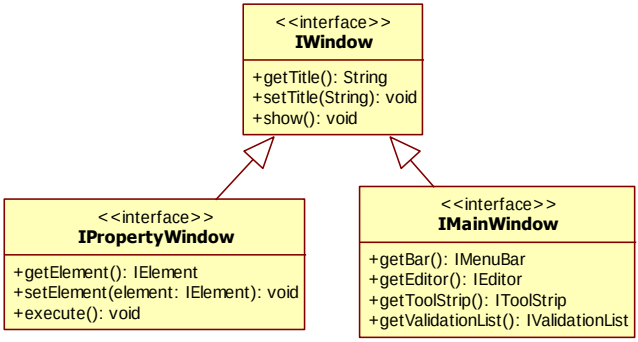
3.2.5.1. Kelas IWindow

Kelas IWindow adalah kelas umum berupa jendela tampilan secara umum. Kelas ini dapat dapat diturunkan menjadi jendela yang lebih khusus. Detail kelas ini ada pada Gambar 3.3.

Fungsi show() pada kelas IWindow berfungsi untuk menampilkan jendela antarmuka pada tiap kelas kongkrit dari kelas IWindow.

3.2.5.2. Kelas IPropertyWindow

Kelas IPropertyWindow adalah kelas berupa jendela tampilan untuk memanipulasi detail sebuah elemen diagram. Detail kelas ini ada pada Gambar 3.3. Fungsi getElement() dan setElement adalah fungsi untuk memanipulasi elemen yang akan dimanipulasi. Fungsi execute() adalah fungsi untuk mengaplikasikan perubahan pada elemen.



Gambar 3.3 Kelas IWindow dan Turunannya

3.2.5.3. Kelas IMainWindow

Kelas IMainWindow adalah kelas berupa jendela tampilan utama pada aplikasi konversi diagram alir menjadi kode sumber. Gambar 3.2 menunjukkan kelas IMainWindow mempunyai empat komponen yaitu IMenuBar, IEditor, IToolBar dan IValidationList. Detail kelas ini ada pada Gambar 3.3. Fungsi-fungsi yang ada pada kelas IMainWindow berfungsi untuk mendapatkan komponen dalam kelas tersebut.

3.2.5.4. Kelas IMenuBar

Kelas IMenuBar adalah kelas berupa menu kumpulan menu di bagian atas jendela utama yaitu kelas IMenu. Detail kelas ini ada pada Gambar 3.4.

3.2.5.5. Kelas IMenu

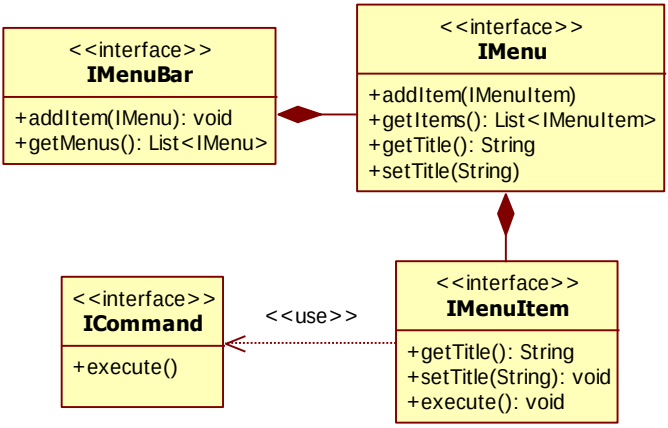
Kelas IMenu adalah kelas berupa tiap buah menu utama yang terlihat di jendela utama. Kelas ini akan menampilkan list sub menu yaitu kelas IMenuItem jika menu diklik. Detail kelas ini ada pada Gambar 3.4. Fungsi addItem() berfungsi untuk menambahkan sub menu.

3.2.5.6. Kelas IMenuItem

Kelas IMenuItem adalah isi dari kelas menu dan akan menjalankan fungsi khusus yang terdapat pada kelas ICommand jika diklik. Detail kelas ini ada pada Gambar 3.4. Fungsi execute() berfungsi untuk menjalankan fungsi khusus sesuai dengan kelas konkrit dari kelas IMenuItem.

3.2.5.7. Kelas ICommand

Kelas ICommand memiliki fungsi execute yaitu fungsi khusus yang akan dijalankan oleh kelas yang memanggilnya. Detail kelas ini ada pada Gambar 3.4.



Gambar 3.4 Kelas IMenu

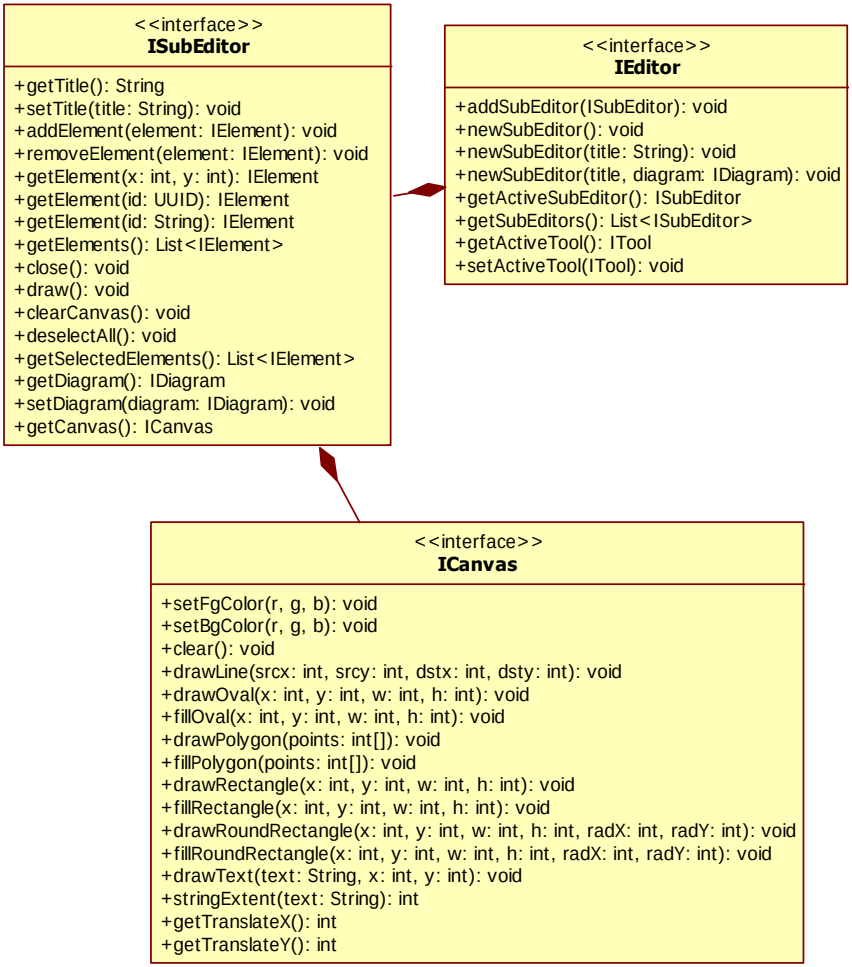
3.2.5.8. Kelas IEditor

Kelas IEditor adalah kelas berupa komponen untuk membuat sebuah diagram. Kelas ini berisi kumpulan sub-editor. Detail kelas ini ada pada Gambar 3.5. Fungsi addSubEditor berfungsi untuk menambahkan sub-editor. Fungsi newSubEditor() berfungsi untuk membuat sub-editor, lalu ditambahkan secara otomatis. Fungsi getActiveSubEditor() berfungsi untuk mendapatkan sub-editor yang sedang aktif. Fungsi getActiveTool() berfungsi untuk mendapatkan tool yang sedang aktif.

3.2.5.9. Kelas ISubEditor

Kelas ISubEditor adalah kelas berupa tab yang berisi kanvas untuk menggambar diagram. Kelas ini menyimpan tipe dari diagram yang ada di dalam kanvas dan elemen-elemen yang akan

digambarkan di kanvas. Detail kelas ini ada pada Gambar 3.5. Selain memiliki fungsi *setter* dan *getter*, kelas ini memiliki fungsi pendukung editor. Fungsi addElement() berfungsi untuk menambahkan elemen ke editor. Fungsi removeElement() berfungsi untuk menghapus elemen dari editor.



Gambar 3.5 Kelas IEditor

Fungsi getElement() berfungsi untuk mendapatkan elemen berdasarkan titik koordinat atau ID dari elemen. Fungsi close() berfungsi untuk menghapus sub-editor. Fungsi draw() berfungsi untuk menggambarkan semua elemen ke kanvas. Fungsi clearCanvas() berfungsi untuk menghapus semua gambar di kanvas. Fungsi deselectAll() berfungsi untuk mengeset semua status elemen menjadi normal. Fungsi getSelectedElements() berfungsi untuk mendapatkan list dari elemen yang statusnya terpilih.

3.2.5.10. Kelas ICanvas

Kelas ICanvas adalah kelas berupa kanvas untuk menggambar diagram. Detail kelas ini ada pada Gambar 3.5. Kelas ini memiliki fungsi untuk menggambar. Fungsi setFgColor() dan setBgColor() berfungsi untuk mengeset warna yang akan digambarkan ke kanvas. Fungsi clear() berfungi untuk mereset kanvas menjadi kosong. Fungsi-fungsi yang berawalan draw berfungsi untuk menggambar elemen sesuai dengan nama fungsi dengan warna yang sebelumnya tersimpan oleh setFgColor(). Fungsi-fungsi yang berawalan fill berfungsi untuk mengisi warna yang sebelumnya tersimpan oleh setBgColor() dalam bentuk elemen dua dimensi sesuai dengan nama fungsi. Fungsi stringExtent() berfungsi untuk mendapatkan panjang dari sebuat text secara vertical dan horizontal dalam satuan pixel. Fungsi getTranslateX() dan getTranslateY() berfungsi untuk mendapatkan jumlah pergeseran koordinat x dan y pada kanvas.

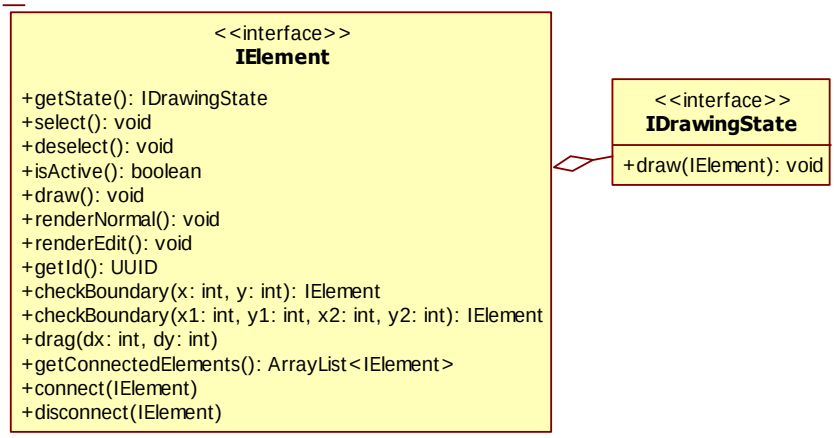
3.2.5.11. Kelas IElement

Kelas IElement adalah *interface* berupa elemen yang dapat digambarkan di kanvas. Setiap elemen yang berhubungan dengan elemen lain disimpan di dalam list elemen yang berhubungan. Elemen memiliki dua macam jenis status yaitu terpilih dan tidak terpilih. Kelas elemen dapat diturunkan menjadi bentuk-bentuk

yang dapat digambarkan ke kanvas, seperti garis, bentuk dua dimensi, dan *control point*. Detail kelas ini ada pada Gambar 3.6. Fungsi getState() berfungsi untuk mendapatkan status dari elemen. Fungsi select() berfungsi untuk mengubah status elemen menjadi terpilih. Fungsi deselect() berfungsi untuk mengubah status elemen menjadi normal. Fungsi draw() berfungsi untuk menggambar elemen di kanvas sesuai dengan status elemen tersebut.

3.2.5.12. Kelas IDrawingState

Kelas IWindow adalah kelas status dari sebuah elemen. Kelas ini memiliki fungsi draw element untuk menggambar elemen di kanvas sesuai dengan state elemen tersebut. Objek konkrit dari kelas ini adalah NormalState dan EditState. Detail kelas ini ada pada Gambar 3.6.



Gambar 3.6 Kelas IElement

3.2.5.13. Kelas IToolStrip

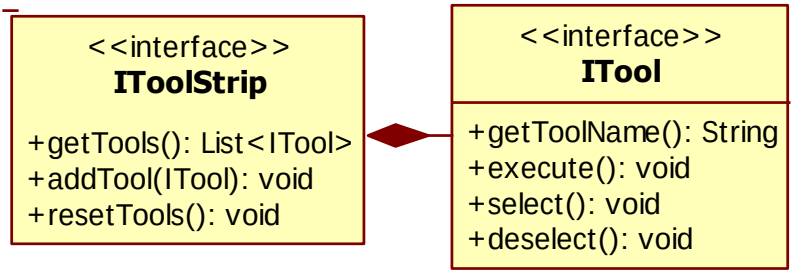
Kelas IToolStrip adalah kelas berisi kumpulan tool untuk menggambar diagram. Detail kelas ini ada pada Gambar 3.7.

3.2.5.14. Kelas ITool

Kelas ITool adalah tool untuk menggambar diagram. Pengguna dapat memilih tool yang sedang aktif. Setiap tool memiliki fungsi yang berbeda jika berinteraksi dengan kanvas menggunakan mouse dan keyboard. Detail kelas ini ada pada Gambar 3.7. Fungsi execute() akan dijalankan pada saat tool diklik.

3.2.5.15. Kelas IValidationList

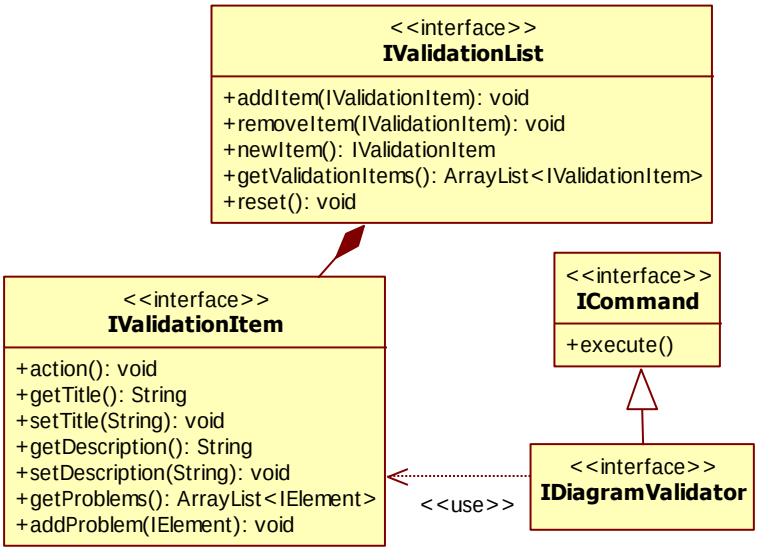
Kelas IValidationList adalah kelas berupa list berisi kesalahan pada saat validasi. Detail kelas ada pada Gambar 3.8. Fungsi-fungsi dari kelas IValidationList berfungsi untuk memanipulasi isi dari list yaitu IValidationItem.



Gambar 3.7 Kelas ITool

3.2.5.16. Kelas IValidationItem

Kelas IValidationItem adalah satu baris dari list kesalahan pada saat validasi. Detail kelas ada pada Gambar 3.8. Kelas IValidationItem memiliki atribut *title* dan *problems*. *Title* adalah text yang muncul terlihat di komponen list IValidationList. *Problems* adalah sekumpulan elemen diagram yang salah. Pada saat kelas ini diklik, maka elemen-elemen yang ada di dalam *problems* akan terpilih.



Gambar 3.8 Kelas IValidation

3.2.5.17. Kelas IDiagramValidator

Kelas IDiagramValidator yang meng-*extend* kelas ICommand berisi fungsi "execute" untuk memvalidasi diagram. Detail kelas ada pada Gambar 3.8.

3.2.6. Rancangan Algoritma Konversi Diagram Alir menjadi Kode Sumber

Proses konversi diagram alir menjadi kode sumber yang akan digunakan dalam aplikasi konveri diagram alir menjadi kode sumber adalah diagram alir dikonversi ke bentuk PAD terlebih dahulu, lalu PAD dikonversi menjadi kode sumber karena PAD lebih menggambarkan hirarki dari sebuah kode sumber. Gambar 3.9 menjelaskan fase konversi diagram alir menjadi PAD. Kode

yang berwarna ungu direpresentasikan oleh simbol pada diagram alir dan PAD yang berada dalam kotak bergaris putus-putus.

Salah satu perbedaan dari diagram alir dan PAD adalah percabangan pada diagram alir dapat menjadi selection, post-decision loop, atau pre-decision loop. Agar model diagram alir tidak ambigu dan lebih jelas, maka aplikasi konversi diagram alir menjadi kode sumber menggunakan sebuah simbol tambahan yang digambarkan saat aliran keluar dari struktur percabangan atau perulangan berbentuk "on page connector" seperti pada Tabel 2.1 yaitu "convergence" [1]. Contoh potongan diagram alir yang menggunakan "convergence" ada pada Gambar 3.10. Untuk menghindari ambigu, bentuk belah ketupat yang ada pada diagram alir dinamakan "*judgment*", sedangkan struktur diagram alir yang menjadi percabangan (*if-else*) adalah "percabangan".

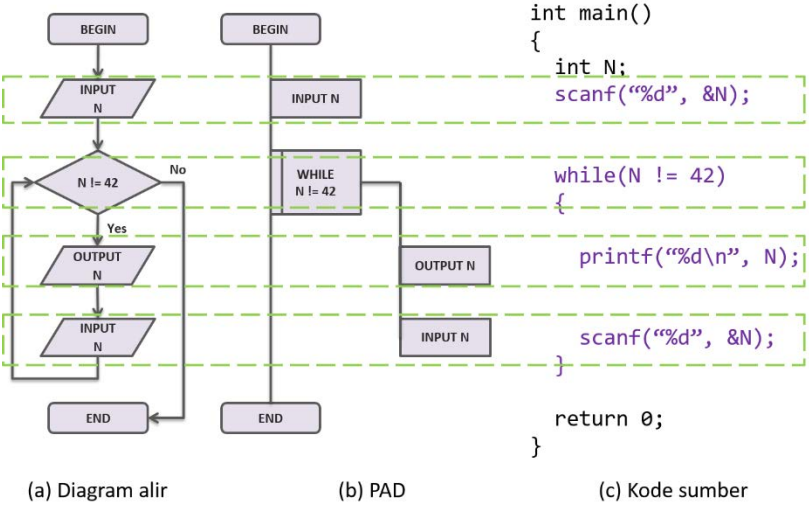
3.2.6.1. Identifikasi dan Pengkodean Elemen Diagram Alir

Langkah untuk mengkonversi diagram alir menjadi PAD adalah mengidentifikasi semua *judgment* pada diagram alir menjadi salah satu dari percabangan atau perulangan (*while* atau *do-while*).

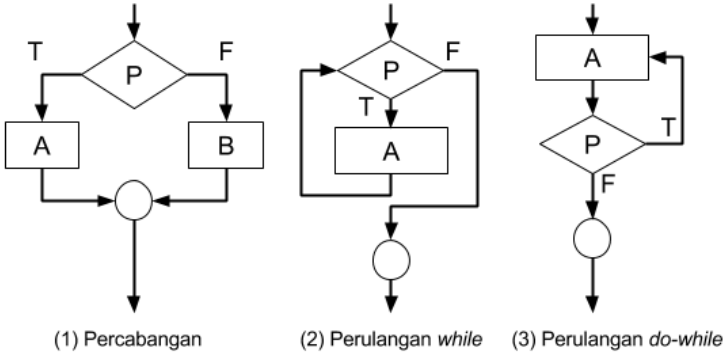
Dari Gambar 3.10 dapat dilihat bahwa pada perulangan, ada elemen yang dilalui lebih dari satu kali sedangkan pada percabangan tidak. Untuk mengidentifikasi *while* atau *do-while*, cek apakah perulangan itu berasal dari judgment atau menuju judgment. Jika berasal dari judgment, maka perulangan tersebut adalah perulangan *do-while*. Jika perulangan berasal dari proses dan menuju judgment, maka perulangan itu adalah perulangan *while*. Jika tidak keduanya, maka diagram alir tersebut tidak valid [1].

Selanjutnya, diperlukan sebuah strategi pengkodean setiap elemen pada diagram alir untuk menentukan urutan atau kedalaman aliran. Kode yang digunakan adalah berbentuk `(x,y)` dimana x menunjukkan nomor cabang saat masuk ke dalam percabangan sedangkan y menunjukkan urutan kode dalam setiap

x. Kode tersebut akan bertambah panjang setiap masuk ke dalam percabangan atau perulangan. Panjang dari kode menunjukkan level kedalaman kode. Kode dari "convergence" selalu sama dengan kode dari elemen "judgment"-nya. Nilai x dan y dimulai dari 0.



Gambar 3.9 Konversi diagram alir menjadi PAD dan kode sumber



Gambar 3.10 Potongan Diagram Alir Menggunakan "Convergence"

Berikut adalah kode yang terbentuk Gambar 3.10. Pada gambar percabangan, kode dari P adalah `(0,1)`, kode dari A adalah `(0,1)(0,0)`, dan kode dari B adalah `(0,1)(1,0)`. Pada gambar perulangan *while* dan perulangan *do-while*, kode dari P adalah `(0,1)` sedangkan kode dari A adalah `(0,1)(0,0)`.

Untuk mengidentifikasi setiap *judgment* dan pemberian kode pada setiap elemen, maka dilakukan fungsi rekursif *depth-first search* untuk mengunjungi dan menandai setiap elemen dari diagram alir. Jika rekursif menemukan sebuah elemen *judgment*, maka kunjungi dahulu anak dari elemen *judgment* tersebut.

Jika *judgment* tersebut adalah perulangan *while*, maka aliran rekursif akan mengunjungi *judgment* tersebut dua kali jika *judgment* tersebut adalah perulangan *while*.

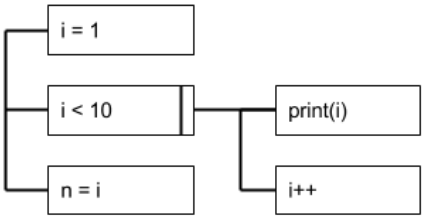
Jika *judgment* tersebut adalah perulangan *do-while*, maka aliran rekursif akan mengunjungi anak dari *judgment* tersebut dua kali, lalu perlu dilakukan pengkodean ulang pada semua elemen yang ada di dalam perulangan *do-while* tersebut.

Jika semua anak dari *judgment* sudah dilalui dan *judgment* belum teridentifikasi, maka tipe *judgment* tersebut adalah percabangan.

3.2.6.2. Konversi Diagram Alir Menjadi PAD

Setelah semua elemen diagram alir diberi kode dan semua *judgment* teridentifikasi menjadi percabangan atau perulangan, selanjutnya adalah melakukan fungsi rekursif *depth-first search* untuk memetakan setiap elemen diagram alir menjadi PAD. Siapkan kelas-kelas elemen PAD untuk menyimpan PAD.

Elemen PAD secara umum ada dua macam, yaitu *container* dan *block*. *Container* adalah elemen PAD yang memiliki beberapa *block* sedangkan *block* adalah sebuah elemen PAD seperti proses, selection, loop, dsb. Pada Gambar 3.11, garis tegak lurus adalah *container*, sedangkan segi empat di sebelah kanan *container* adalah *block*.



Gambar 3.11 Contoh PAD

Fungsi rekursif dimulai dari kode elemen diagram alir yang paling awal, yaitu `(0,0)`. Lalu cari elemen diagram alir yang mempunyai kode tersebut.

Jika elemen diagram alir tersebut adalah proses, maka masukkan elemen tersebut ke PAD dan proses kode setelah elemen sebelumnya di mana nilai y adalah 1 yaitu `(0,1)`.

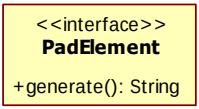
Jika elemen diagram alir tersebut adalah perulangan baik *while* maupun *do-while*, maka masukkan elemen tersebut ke PAD dan proses kode anak dari kode `(0,0)` yaitu `(0,0)(0,0)`. Setelah anak dari kode `(0,0)` selesai dipetakan ke PAD, lanjut untuk memproses kode `(0,1)`. Proses rekursi berhenti ketika tidak ada elemen yang mempunyai kode yang akan dicari.

3.2.6.3. Konversi PAD Menjadi Kode Sumber

PAD adalah diagram yang menggambarkan kode sumber dengan sangat baik. Dari PAD yang sudah dibuat sebelumnya, dibutuhkan fungsi rekursif *depth-first search* untuk mengubah semua elemen PAD menjadi kode sumber.

Setiap elemen PAD memiliki template khusus untuk menghasilkan kode sumber, ditunjukkan pada Gambar 3.12 yaitu fungsi generate().

Dimulai dari elemen *container* paling awal, ubah setiap elemen yang ada di dalamnya. Jika menemui proses, maka langsung masukkan ke kode sumber.



Gambar 3.12 Kelas elemen PAD

Jika menemukan percabangan atau perulangan, maka buat kode sumber yang sesuai dengan percabangan (*if-else*) atau perulangan (*while* atau *do-while*). Lalu masuk ke *container* di dalam elemen percabangan atau perulangan tersebut.

BAB IV
IMPLEMENTASI

Bab ini akan menjelaskan tentang implementasi tugas akhir berdasarkan rancangan perangkat lunak. Proses implementasi mengacu pada rancangan perangkat yang telah dilakukan sebelumnya, namun juga dimungkinkan terjadinya perubahan-perubahan jika dirasa perlu. Implementasi dilakukan dalam bahasa Java.

4.1. Lingkungan Implementasi

Subbab ini menjelaskan tentang lingkungan implementasi perangkat lunak yang dibangun. Lingkungan selama proses implementasi aplikasi konversi diagram alir menjadi kode sumber ada pada Tabel 4.1.

Tabel 4.1 Spesifikasi Lingkungan Implementasi

Perangkat Keras	- Prosesor Intel(R) Core(TM) i7-3240 CPU @ 3.40GHz - Memori 4 GB
Perangkat Lunak	- Sistem Operasi Microsoft Windows 10 64-bit - Eclipse Mars 4.5.0 - JDK 1.8.0_91 - Java Eclipse SWT 4.5

4.2. Implementasi Antar muka

Antarmuka pada perancangan yang dijelaskan pada bab 3.2.4 akan diimplementasi menggunakan pustaka Java SWT. Setiap kelas antarmuka akan mengimplementasi *interface* perancangan antarmuka pada bab 3.2.5 dan meng-*extend* kelas dari Java SWT. Tabel 4.2 menjelaskan pemetaan kelas antarmuka yang mengimplementasi, interface perancangan, dan kelas *parent*.

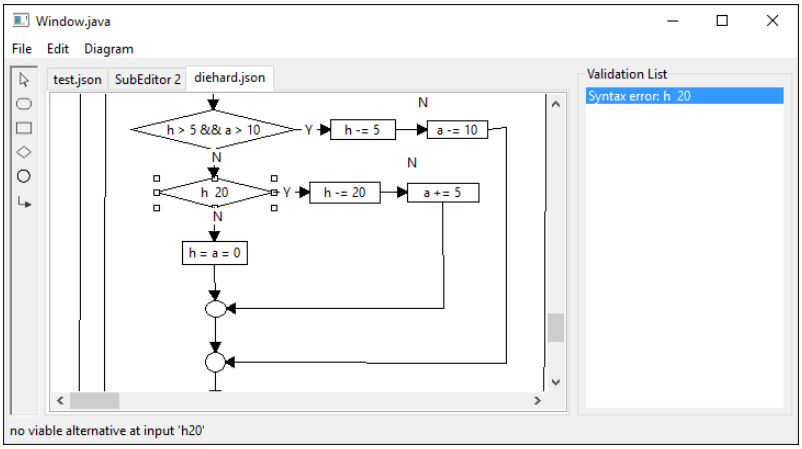
Tabel 4.2 Implementasi Kelas Antarmuka

Implementasi	Interface	Extend
AWindow	IWindow	org.eclipse.swt.widgets.Shell
MainWindow	IMainWindow	AWindow
MenuBar	IMenuBar	org.eclipse.swt.widgets.Menu
AMenu	IMenu	org.eclipse.swt.widgets.Menu
FileMenu		AMenu
EditMenu		AMenu
DiagramMenu		AMenu
AMenuItem	IMenuItem	org.eclipse.swt.widgets.MenuItem
ValidationList	IValidationList	org.eclipse.swt.widgets.List
Editor	IEditor	org.eclipse.swt.widgets.TabFolder
SubEditor	ISubEditor	org.eclipse.swt.widgets.TabItem
Canvas	ICanvas	org.eclipse.swt.widgets.Canvas
ToolStrip	IToolStrip	org.eclipse.swt.widgets.ToolBar
ATool	ITool, org.eclipse.swt .events .MouseListener	org.eclipse.swt.widgets.ToolItem
PointerTool		ATool
TerminatorTool		ATool
ProcessTool		ATool
JudgmentTool		ATool
ConvergenceTool		ATool
PolylineTool		ATool
APropertyWindow	IPropertyWindow	AWindow

Kelas Canvas dan ATool akan mengimplementasi interface MouseListener dari Java SWT untuk dapat melakukan interaksi dengan mouse. Beberapa fungsi dari mouse listener adalah mouseDown, mouseMove, mouseUp, dan mouseDoubleClick.

4.3. Implementasi Kelas Elemen dan Diagram

Subbab ini akan menjelaskan tentang implementasi dari kelas-kelas elemen yang dapat digambarkan pada editor diagram alir. Elemen-elemen tersebut bias berupa elemen diagram atau elemen yang dapat membantu memvisualisasi diagram.



Gambar 4.1 Antarmuka Utama

4.3.1. Diagram Kelas Elemen

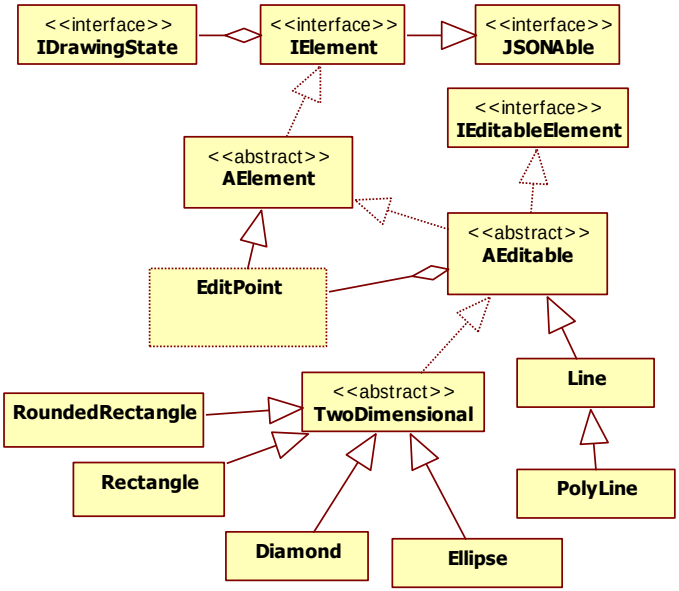
Diagram kelas pada Gambar 4.2 menggambarkan kelas-kelas elemen yang dapat digambar ke kanvas dalam SubEditor. Kelas AElement adalah bentuk abstrak dari kelas IElement. Semua elemen gambar pasti turunan dari kelas AElement.

Salah satu turunan kelas AElement adalah AEditable, yaitu elemen yang dapat diperbesar dan diperkecil ukurannya menggunakan *edit point*. *Edit point* adalah gambar berbentuk persegi yang berada di delapan ujung sebuah elemen yang statusnya terseleksi seperti elemen *judgment* h 20 pada Gambar 4.1.

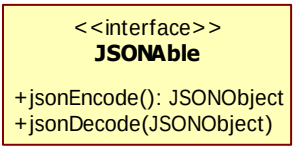
Kelas AEditable dapat diturunkan menjadi dua kelas, yaitu kelas Line berupa garis, dan kelas abstrak TwoDimensional berupa elemen dua dimensi. Kelas TwoDimensional akan diturunkan menjadi elemen-elemen diagram.

Setiap kelas konkrit dari AElement akan merealisasi fungsi dari kelas JSONAble untuk dapat disimpan dan dibuka ke dalam file. Kelas JSONAble pada Gambar 4.3 memiliki dua fungsi yaitu jsonEncode() untuk mendapatkan objek JSON berisi atribut dari objek elemen yang memanggil dan jsonDecode() untuk

memasukkan atribut pada JSON ke dalam objek elemen. Setiap elemen memiliki id unik bertipe UUID yang pasti tersimpan dalam objek JSON ketika elemen akan disimpan.



Gambar 4.2 Diagram Kelas Elemen

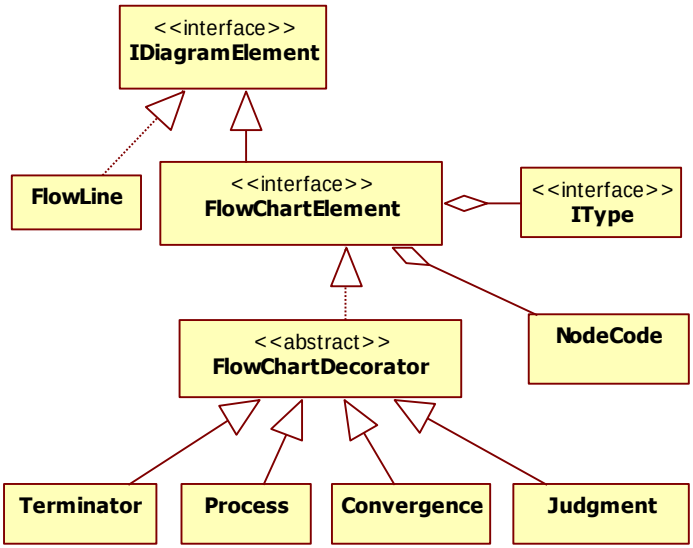


Gambar 4.3 Kelas JSONAble

4.3.2. Diagram Kelas Diagram Alir

Diagram kelas pada Gambar 4.4 menggambarkan kelas-kelas yang digunakan untuk menyimpan elemen diagram alir. Subbab selanjutnya akan menjelaskan detail setiap kelas. Kelas

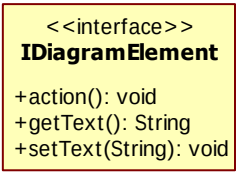
konkrit dari elemen diagram alir menggunakan kelas FlowChartDecorator.



Gambar 4.4 Diagram Kelas Flowchart

4.3.2.1. Kelas IDiagramElement

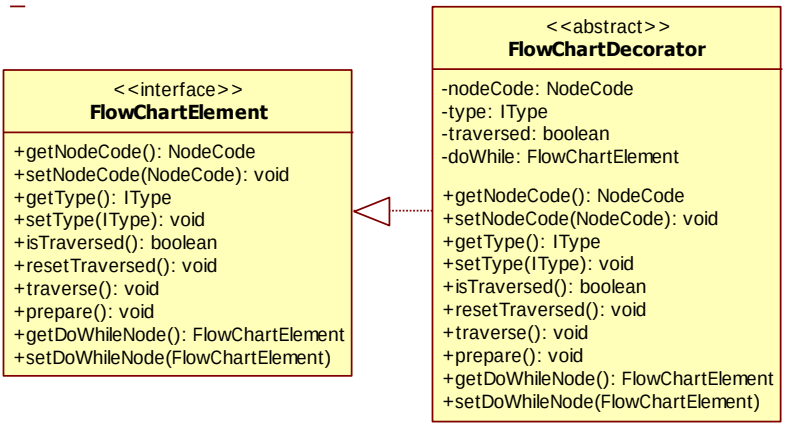
Kelas ini adalah *interface* umum untuk elemen diagram. Elemen diagram akan menjalankan fungsi action() saat elemen tersebut menerima *double-click*. Setiap elemen diagram memiliki text yang menjelaskan maksud dari elemen tersebut sesuai dengan jenis elemen diagram. Detail kelas ini ada pada Gambar 4.5.



Gambar 4.5 Kelas IDiagramElement

4.3.2.2. Kelas FlowChartElement

Kelas ini adalah bentuk umum dari elemen diagram alir. Kelas FlowChartElement memiliki property-properti yang membantu tahap konversi diagram alir menjadi PAD yaitu nodeCode untuk pengkodean elemen, type untuk tipe dari elemen diagram alir, traversed untuk menandakan elemen sudah dilewati dalam rekursi, dan doWhile untuk menyimpan elemen perulangan *do-while*. Detail kelas ini ada pada Gambar 4.6.



Gambar 4.6 Kelas FlowChartElement

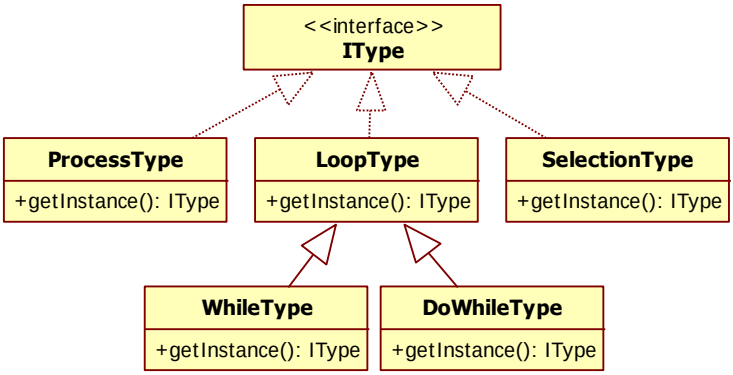
4.3.2.3. Kelas IType

Kelas IType adalah tipe elemen diagram alir yang sudah teridentifikasi oleh proses identifikasi dan pengkodean diagram alir. Detail kelas ada pada Gambar 4.7.

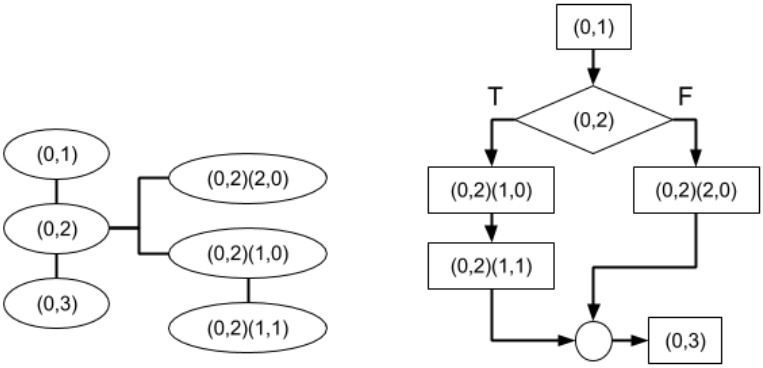
4.3.2.4. Kelas NodeCode

Kelas NodeCode adalah kelas untuk menyimpan kode elemen diagram alir untuk keperluan konversi ke PAD. NodeCode menyimpan elemen yang mempunyai kode ini. NodeCode juga

menyimpan NodeCode anak, NodeCode ayah dan NodeCode saudara. Kode berbentuk (x,y), dimana x bernilai 0 untuk saudara, 1 atau 2 untuk anak, sedangkan y merepresentasikan urutan dalam satu turunan. Detail kelas ini ada pada Gambar 4.9.



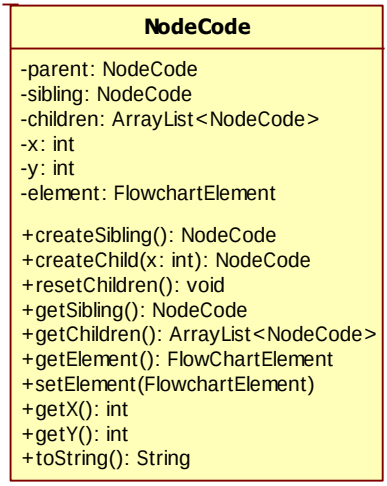
Gambar 4.7 Kelas IType



Gambar 4.8 NodeCode dan Diagram Alir yang Sepadan

Contoh bentuk penyimpanan NodeCode ada pada Gambar 4.8. Garis melintang menggambarkan kode atas mempunyai saudara kode di bawahnya. Garis mendatar menggambarkan kode sebelah kiri mempunyai anak kode di sebelah kanannya. Kode `(0,1)` memiliki satu saudara yaitu kode `(0,2)`. Kode `(0,2)`

memiliki dua anak, yaitu `(0,2)(1,0)` dan `(0,2)(2,0)`. Jika x bernilai 1 pada anak pertama, berarti anak tersebut berasal dari *flowline* yang bernilai benar, sedangkan jika x bernilai 2 pada anak pertama, berarti anak tersebut berasal dari flowline yang bernilai salah.



Gambar 4.9 Kelas NodeCode

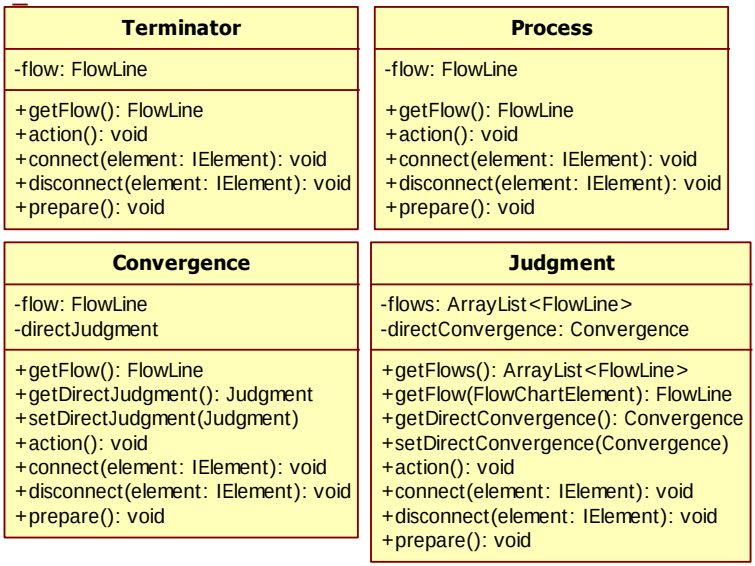
4.3.2.5. Kelas Elemen Diagram Alir

Kelas-kelas elemen diagram alir adalah kelas utama yang menyimpan elemen diagram alir. Setiap kelas memiliki atribut, fungsi, dan bentuk yang berbeda. Detail kelas elemen diagram alir ada pada Gambar 4.10. Kelas Terminator adalah kelas untuk menggambar elemen terminator *start* dan *end*. Kelas Proses, kelas Convergence, dan kelas Judgment adalah kelas yang menggambar elemen proses, *convergence*, dan *judgment*.

4.3.3. Diagram Kelas PAD

Kelas-kelas ini adalah kelas yang digunakan untuk menyimpan elemen PAD. Diagram kelas ada pada Gambar 4.11.

Setiap kelas dari PAD memiliki memiliki fungsi generate, yaitu untuk mengkonversi PAD menjadi kode sumber. Fungsi generate akan direalisasikan oleh kelas konkrit dari kelas PadElement.



Gambar 4.10 Kelas Elemen Diagram Alir

Elemen PAD secara umum dibagi menjadi dua, yaitu BlockSingle dan BlockContainer. Contoh pada Gambar 3.11, garis melintang adalah elemen BlockContainer, sedangkan kotak yang berisi potongan kode sumber adalah elemen BlockSingle.

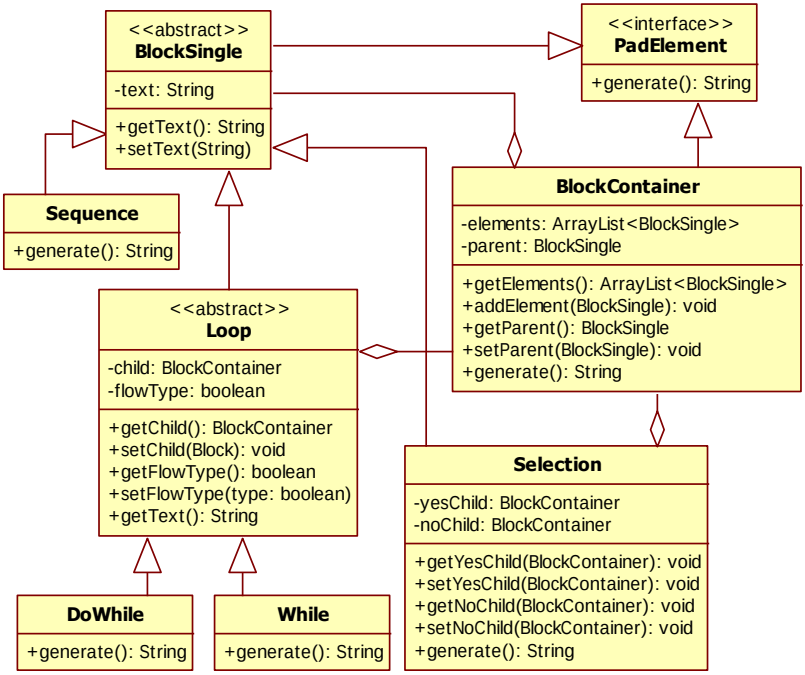
BlockContainer akan memanggil fungsi generate semua elemen BlockSingle anaknya pada saat fungsi generate dari BlockContainer dipanggil.

Kelas BlockSingle dapat diturunkan menjadi tiga jenis kelas, yaitu kelas Sequence, kelas Selection, dan kelas Loop.

Kelas Sequence berasal dari kelas Proses pada elemen diagram alir yang hanya berisi satu baris kode sumber.

Kelas Selection berasal dari kelas Judgment pada elemen diagram alir, berisi dua anak yaitu yesChild dan noChild. Kelas Selection mewakili percabangan *if-else* pada kode sumber.

Kelas Loop berasal dari kelas Judgment pada elemen diagram alir, memiliki satu anak dan jenis aliran dari anak tersebut, apakah *true* atau *false*. Kelas loop dapat diturunkan menjadi kelas While dan DoWhile. Kelas While dan kelas DoWhile mewakili perulangan *while* dan *do-while* pada kode sumber.



Gambar 4.11 Diagram Kelas PAD

4.4. Implementasi Validasi Diagram Alir

Diagram kelas untuk memvalidasi diagram alir ada pada Gambar 4.12. Pada saat pengguna memilih menu validasi diagram

alir, kelas-kelas konkrit dari IDiagramValidator akan dipanggil. Masing-masing kelas mengurusi ranah keabsahan diagram alir yang berbeda sebagai berikut:

1. TerminatorValidator
   a. Tidak ada elemen start atau end
   b. Elemen start atau end lebih dari satu
2. FlowValidator
   a. *Judgment* yang tidak memiliki pasangan *convergence*
   b. *Convergence* yang tidak memiliki pasangan *judgment*
   c. Aliran yang tidak sesuai
   d. Perulangan tak terhingga
3. JudgmentValidator
   a. *Judgment* tidak memiliki anak berjumlah dua
   b. Aliran *judgment* belum terdefinisi, atau memiliki aliran yang sama
4. ElementsConnectionValidator
   a. Ada elemen yang tidak terkoneksi
5. SyntaxValidator
   a. Sintaks tidak sesuai

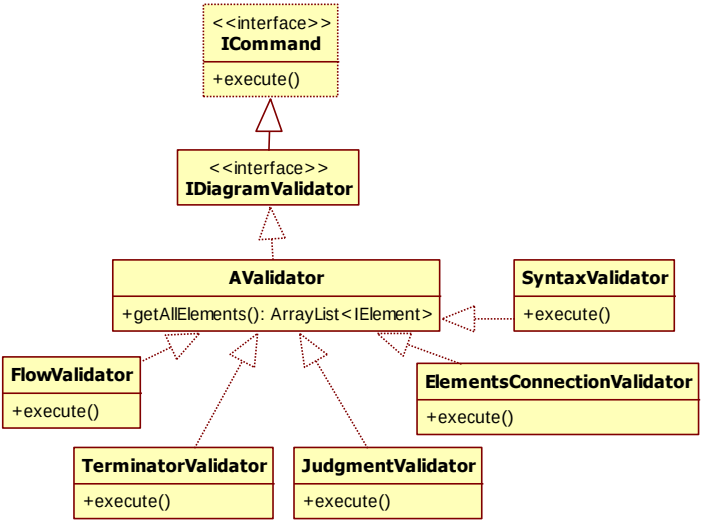
4.5. Implementasi Algoritma Konversi Diagram Alir Menjadi Kode Sumber

Subbab ini akan menjelaskan implementasi algoritma mengubah diagram alir ke kode sumber dengan pendekatan PAD. Untuk melakukan konversi diagram alir menjadi kode sumber, terdapat tiga implementasi algoritma yang akan dijelaskan pada subbab di bawah ini.

4.5.1. Identifikasi dan Pengkodean Elemen Diagram Alir

Di bab rancangan untuk Identifikasi dan Pengkodean Elemen Diagram Alir, disebutkan bahwa ada dua proses rekursif yaitu untuk melakukan identifikasi dan pengkodean dan untuk

pengkodean ulang saat masuk ke dalam perulangan *do-while*. Namun pada implementasi, ada perubahan yaitu tidak perlu menggunakan fungsi pengkodean ulang. Untuk melakukan pengkodean ulang hanya perlu masuk ke dalam rekursif lagi dengan menghitung jumlah perulangan *do-while* yang sedang aktif.



Gambar 4.12 Diagram kelas validator diagram alir

Dari rancangan identifikasi dan pengkodean elemen diagram alir serta analisa lebih lanjut, dibuatlah strategi identifikasi dan pengkodean elemen diagram berupa *pseudo-code* yang ada pada Kode Sumber 4.1.

Baris 1 sampai 8 adalah persiapan untuk memanggil fungsi rekursif pada baris 10 sampai selesai. Basis rekursif pada baris 11 dan 12 menunjukkan fungsi akan keluar ketika bertemu dengan elemen terminator *end*. Fungsi utama terpisah berdasarkan jenis elemen yang sedang diproses. Baris 17 sampai 36 jika tipe elemen adalah proses, baris 38 sampai 80 jika tipe elemen adalah *judgment*, dan baris 82 sampai 111 jika tipe elemen adalah convergence.

```
/* Begin code flowchartElement, send father, his son, and new code */
father = terminator start;
son = father.getFlow().getSon();
newCode = new NodeCode(); // (0,0)
set nodeCode of father with newCode
stackOfJudgment = new Stack<>();
doWhileStack = new Stack<>();
codeAlgorithm(father, son, father.getNodeCode().createSibling());

function codeAlgorithm(FlowChartElement father, FlowChartElement currElem, NodeCode currCode)
  if (currElem is Terminator)
    return; // exit point of recursion

  if (currElem.getDoWhileNode() != doWhileStack.peek)
    reset traversed of currElem // masih butuh recode

  if (currElem is Process)
    if (currElem isntTraversed)
      traverse currElem
      set nodeCode of currElem with currCode
      son = currElem.getFlow().getSon();

      codeOfSon = currCode.createSibling();
      codeAlgorithm(currElem, son, codeOfSon);

      if (father is Judgment && father type not defined or DoWhile)
        set type of father with DoWhile
        thisCode = node code of currElem
        set nodeCode of father with thisCode
        thisCode.clearChildren();

        doWhileStack.push(father);
        father.setDoWhileNode(doWhileStack);
```

```
34          childCode = create child of thisCode;
35          codeAlgorithm(father, currElem, childCode);
36          doWhileStack.pop;
37
38    if (currElem is Judgment)
39      if (currElem isntTraversed)
40        traverse currElem
41        set nodeCode of currElem with currCode
42
43        if (currElem.getDirectConvergence() == null)
44          stackOfJudgment.push(currElem);
45
46        convergenceson = null
47        for each (sons of currElem as son)
48          if (son is Convergence)
49            convergenceSon = son;
50            continue;
51
52          sonCode = create child of currCode;
53          codeAlgorithm(currElem, son, sonCode);
54
55        if (convergenceson != null)
56          codeAlgorithm(currElem, convergenceson, null);
57
58        if (currElem.getType == null)
59          set type of father with Selection
60
61        convergence = directConvergence of currElem
62        sonCode = create sibling from node code of
  convergence
63        sonNode = son of convergence
64        codeAlgorithm(convergence, sonNode, sonCode);
65
66      else  // been traversed.
67        if (currElem.getType == null)
68          set type of father with While
69        else  // and been recognized
```

```
70          if (father is Judgment && father.getType == null)
71            set type of father with DoWhile
72            thisCode = node code of currElem
73            set nodeCode of father with thisCode
74            thisCode.clearChildren();
75
76            doWhileStack.push(father);
77            father.setDoWhileNode(doWhileStack);
78            childCode = create child of thisCode;
79            codeAlgorithm(father, currElem, childCode);
80            doWhileStack.pop;
81
82    if (currElem is Convergence)
83      if (currElem isntTraversed)
84        traverse currElem
85
86        if (currElem doesnt have directJudgment)
87          judgment = stackOfJudgment.pop;
88          set direct judgment of currElem with judgment
89          set direct convergence of judgment with currElem
90      set nodeCode of currElem with node code of currElem
    directJudgment
91
92
93  /* Begining to recode do-while child */
94            doWhileStack.push(father);
95            father.setDoWhileNode(doWhileStack);
96            childCode = create child of thisCode;
97            codeAlgorithm(father, currElem, childCode);
98            doWhileStack.pop;
99
100   if (currElem is Convergence)
101 /* match a judgment node and a convergence node */
102     if (currElem isntTraversed)
103 /* Convergence hasn't been traversed */
104       traverse currElem
```

```
105
106         if (currElem doesnt have directJudgment)
107  /* Connect convergence with judgment */
108           judgment = stackOfJudgment.pop;
109           set direct judgment of currElem with judgment
110           set direct convergence of judgment with currElem
111       set nodeCode of currElem with node code of currElem
     directJudgment
```

Kode Sumber 4.1 Algoritma Identifikasi dan Pengkodean Elemen Diagram Alir

4.5.2. Konversi Diagram Alir Menjadi PAD

Diagram Alir yang sudah diproses dan diberi kode akan diproses untuk dikonversi menjadi PAD. *Pseudo-code* untuk poses konversi diagram alir menjadi PAD ada pada Kode Sumber 4.2.

Baris 1 sampai 3 menjelaskan parameter yang dimasukkan ke proses rekursi adalah kode dari elemen diagram alir, dan elemen PAD BlockContainer. Pada saat pertama fungsi dipanggil, parameter pertama adalah kode awal yang dibuat pada saat proses identifikasi dan pengkodean elemen diagram alir, sedangkan parameter kedua adalah sebuah BlockContainer baru yang diberi nama fatherBlock.

Inti dari proses konversi diagram alir menjadi PAD adalah melakukan proses rekursi untuk mengunjungi semua kode yang telah dibuat. Dari kode tersebut dapat tersambung ke elemen yang memiliki kode itu dan dapat tersambung ke kode saudara dan kode anaknya.

Perlakuan fungsi akan berbeda pada setiap jenis elemen. Baris 12 sampai 15 jika tipe elemen adalah proses. Baris 17 sampai 26 jika tipe elemen adalah *selection*. Baris 28 sampai 40 jika tipe elemen adalah perulangan *while* atau *do-while*.

```
1  firstCode = first code from codeAlgorithm;
2  fatherBlock = new BlockContainer;
3  convertToPAD(firstCode, fatherBlock);
```

```
4
5   function convertToPAD(currCode, BlockContainer
    fatherBlock)
6     if (currCode == null)
7       return;
8     currElem = currCode.getElement();
9     if (currElem == null)
10      return;
11
12    if (currElem type is "Process")
13      element = new Sequence;
14      element.setText(currElem.getText());
15      add element to fatherBlock;
16
17    if (currElem type is "Selection")
18      element = new Selection;
19      element.setText(currElem.getText());
20
21      foreach currCode children thats not convergence
22        nextFlow = child.getElement();
23        subBlock = new BlockContainer;
24        set subBlock as yes or no child of element;
25        convertToPAD(nextFlow.getNodeCode(), subBlock);
26      add element to fatherBlock;
27
28    if (currElem type is "DoWhile" OR "While")
29      element = new While or DoWhile;
30      element.setText(currElem.getText());
31      subContainer = new BlockContainer;
32
33      childCode = the only child of currCode;
34      set subContainer as element child
35      add element to fatherBlock;
36      convertToPAD(childCode, subContainer);
37
```

```
38    /* Go to next code from the same layer */
39    currCode = currCode next sibling;
40    convertToPAD(currCode, fatherBlock);
```

Kode Sumber 4.2 Algoritma Konversi Diagram Alir Menjadi PAD

4.5.3. Konversi PAD Menjadi Kode Sumber

Proses konversi PAD menjadi kode sumber adalah memanggil fungsi generate pada fatherBlock, yaitu BlockContainer pertama yang dibuat pada saat proses konversi diagram alir menjadi PAD. Keluaran dari generate adalah kode sumber hasil konversi dari PAD.

Kode Sumber 4.3 adalah kode fungsi generate() pada Kelas BlockContainer, yang akan memanggil fungsi generate dari anak-anak objek BlockContainer. Kode Sumber 4.4, Kode Sumber 4.5, Kode Sumber 4.6, dan Kode Sumber 4.7 adalah fungsi generate dari elemen PAD yang menghasilkan kode sumber berdasarkan masing-masing jenis elemen PAD.

Kode sumber hasil konversi harus dimasukkan ke kode sumber *template*, untuk menyesuaikan standar kode sumber berbahasa C. Kode sumber *template* ditunjukkan oleh Kode Sumber 4.8.

```
public String generate() {
    String ans = "";
    for (BlockSingle element : elements) {
        ans += element.generate();
    }
    return ans;
}
```

Kode Sumber 4.3 Fungsi generate() pada Kelas BlockContainer

```
public String generate() {
    return getText() + ";\n";
}
```

Kode Sumber 4.4 Fungsi generate() pada Kelas Sequence

```
public String generate() {
    String ans = "if (" + getText() + ") {\n";
    if (getYesChild() != null) {
        ans += getYesChild().generate();
    }
    ans += "} else {\n";
    if (getNoChild() != null) {
       ans += getNoChild().generate();
    }
    ans += createTabIndent() + "}\n";
    return ans;
}
```

Kode Sumber 4.5 Fungsi generate() pada Kelas Selection

```
public String generate() {
    String ans = "while (" + getText() + ") {\n";
    ans += getChild().generate();
    ans += "}\n";
    return ans;
}
```

Kode Sumber 4.6 Fungsi generate() pada Kelas While

```
public String generate() {
    String ans = "do {\n";
    ans += getChild().generate();
    ans += "} while (" + getText() + ");\n";
    return ans;
}
```

Kode Sumber 4.7 Fungsi generate() pada Kelas DoWhile

```
#include <stdio.h>

void main() {
    /* <<kode sumber hasil koversi>> */
}
```

Kode Sumber 4.8 *Template* untuk Kode Sumber Hasil Konversi

[Halaman ini sengaja dikosongkan]

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas pengujian dan evaluasi pada aplikasi yang dikembangkan. Pengujian yang dilakukan adalah pengujian terhadap kebutuhan fungsionalitas sistem dan kegunaan sistem. Pengujian fungsionalitas mengacu pada kasus penggunaan pada bab tiga. Pengujian kegunaan program dilakukan dengan mengetahui tanggapan dari pengguna terhadap sistem. Hasil evaluasi menjabarkan tentang rangkuman hasil pengujian pada bagian akhir bab ini.

5.1. Lingkungan Pengujian

Lingkungan pengujian sistem pada pengerjaan tugas akhir ini dilakukan pada lingkungan dan alat kakas seperti yang tertera pada Tabel 5.1.

Tabel 5.1 Lingkungan Pengujian Sistem

Perangkat Keras	- Prosesor Intel(R) Core(TM) i7-3240 CPU @ 3.40GHz - Memori 4 GB
Perangkat Lunak	- Sistem Operasi Microsoft Windows 10 64-bit - Eclipse Mars 4.5.0 - JDK 1.8.0_91 - Java Eclipse SWT 4.5

5.2. Pengujian Aplikasi

Pengujian aplikasi dilakukan untuk mengetahui kesesuaian keluaran dari tiap tahap atau langkah penggunaan fitur terhadap skenario yang dipersiapkan. Berikut ini penjabaran skenario dan hasil uji coba yang dilakukan terhadap perangkat lunak yang dibangun.

5.2.1. Skenario Pengujian Fungsionalitas

Pada subbab ini dijelaskan beberapa skenario uji coba perangkat lunak secara mandiri berdasarkan metode kotak hitam sebagai dasar tolok ukur keberhasilan. Pengujian fungsionalitas yang terdapat pada aplikasi dijabarkan sebagai berikut:

a) Membuat diagram alir.
b) Memvalidasi diagram alir.
c) Menyimpan dan membuka file diagram alir.
d) Mengkonversi diagram alir menjadi kode sumber.
e) Melihat *problem analysis diagram*.

Daftar uji coba tersebut merupakan pengujian yang dilakukan untuk menguji setiap kasus penggunaan pada perangkat lunak yang dibangun. Berdasarkan daftar pengujian yang telah disebutkan dibuat beberapa skenario yang dilakukan pada setiap pengujian tersebut. Penjelasan mengenai cara dan hasil pengujian fungsionalitas perangkat lunak dibahas pada subbab hasil uji coba.

5.2.2. Hasil Pengujian Fungsionalitas Aplikasi

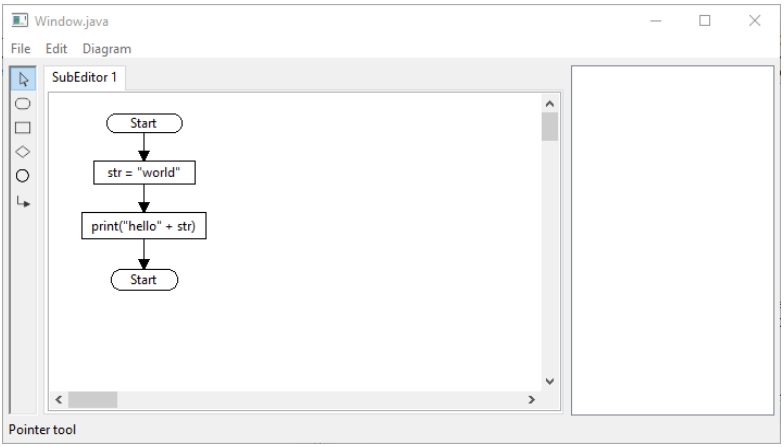
Pada subbab ini dijelaskan secara detail mengenai skenario yang dilakukan dan hasil yang didapatkan dari pengujian fungsionalitas perangkat lunak yang dibangun. Penjelasan disajikan dengan menampilkan kondisi awal, masukan, keluaran, hasil yang dicapai, dan kondisi akhir. Berikut ini merupakan penjabaran skenario dan hasil pengujian yang dicapai pada tiap-tiap fungsionalitas perangkat lunak.

5.2.2.1. Uji Coba Membuat Diagram Alir

Uji coba membuat diagram alir berfungsi untuk mengetahui keberhasilan aplikasi dalam hal menggambarkan bentuk-bentuk diagram alir yang digambarkan oleh pengguna menggnakan tool-tool yang ada. Hasil dari pengujian dapat dilihat pada Tabel 5.2. Hasil uji coba dapat dilihat pada Gambar 5.1.

Tabel 5.2 Uji Coba Membuat Diagram Alir

ID	UJ- 01
Nama	Uji Coba Membuat Diagram Alir
Tujuan Uji Coba	Pengguna dapat membuat diagram alir menggunakan tool-tool yang disediakan pada aplikasi.
Kondisi awal	-
Skenario	Pengguna membuat diagram alir sederhana
Keluaran yang diharapkan	Diagram alir terbuat
Hasil uji coba	Berhasil
Kondisi akhir	Terdapat diagram yang dibuat oleh pengguna pada aplikasi



Gambar 5.1 Hasil Uji Coba Membuat Diagram Alir

5.2.2.2. Uji Coba Memvalidasi Diagram Alir

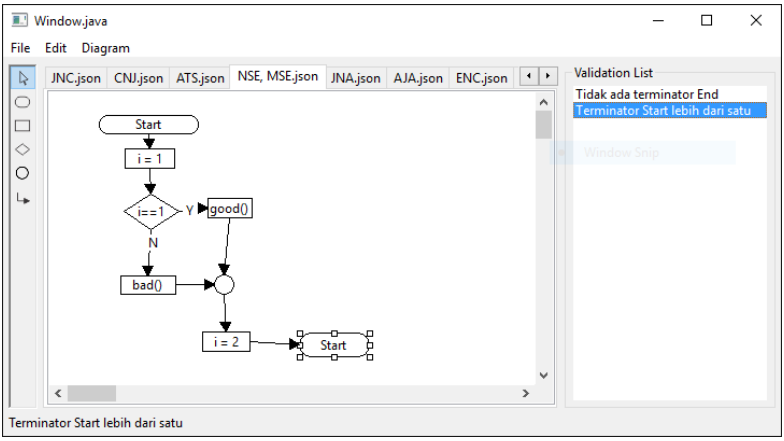
Uji coba memvalidasi diagram alir berfungsi untuk mengetahui keberhasilan aplikasi dalam memvalidasi diagram alir dan mengecek kesalahan-kesalahan pengguna dalam membuat diagram alir. Hasil dari pengujian dapat dilihat pada Tabel 5.3.

Pengujian dilakukan terhadap beberapa jenis kesalahan pada diagram alir dengan hasil sebagai berikut:

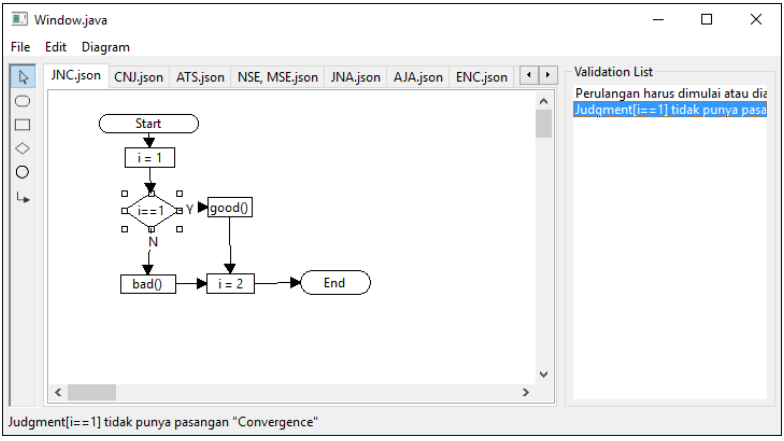
a. Tidak ada elemen start atau end dan elemen start atau end lebih dari satu ada pada Gambar 5.2
b. Judgment yang tidak memiliki pasangan convergence ada pada Gambar 5.3
c. Convergence yang tidak memiliki pasangan judgment ada pada Gambar 5.4
d. Aliran yang tidak sesuai ada pada Gambar 5.5
e. Judgment tidak memiliki anak berjumlah dua ada pada Gambar 5.6
f. Aliran judgment belum terdefinisi, atau memiliki aliran yang sama ada pada Gambar 5.7
g. Ada elemen yang tidak terkoneksi ada pada Gambar 5.8
h. Sintaks tidak sesuai ada pada Gambar 5.9

Tabel 5.3 Uji Coba Memvalidasi Diagram Alir

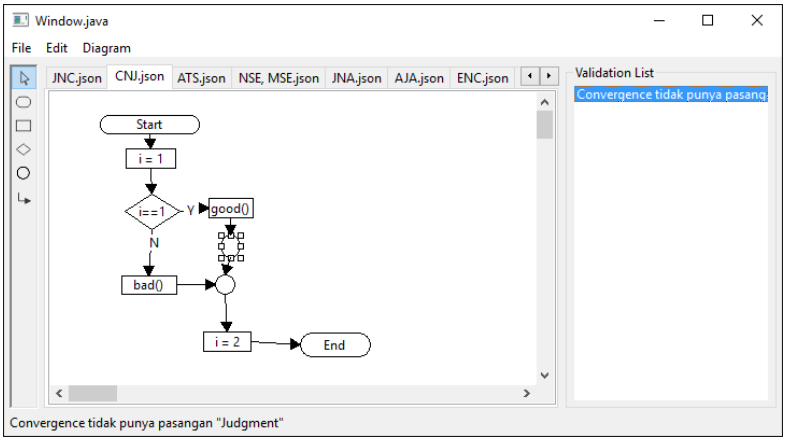
ID	UJ- 02
Nama	Uji Coba Memvalidasi Diagram Alir
Tujuan Uji Coba	Pengguna dapat memastikan keabsahan diagram alir agar dapat diproses lebih lanjut
Kondisi awal	Sudah ada diagram alir yang sedang terbuka
Skenario	Pengguna pilih menu “diagram”, lalu pilih “validate”
Keluaran yang diharapkan	Pengguna mengetahui status keabsahan diagram alir yang dibuat
Hasil uji coba	Berhasil
Kondisi akhir	Muncul status keberhasilan validasi diagram



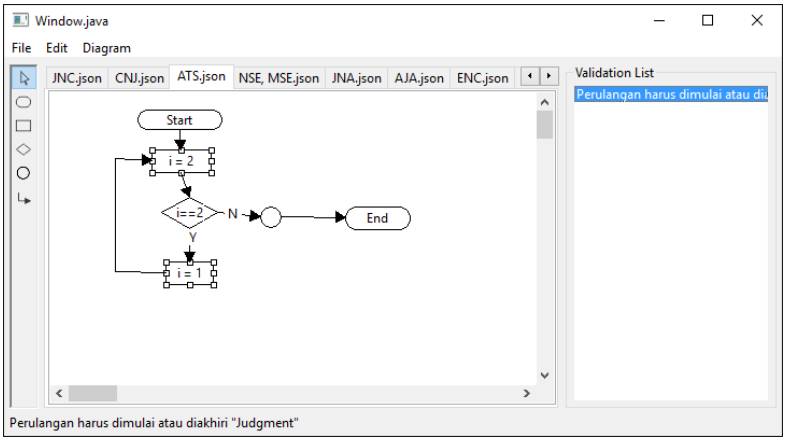
Gambar 5.2 Uji coba kesalahan elemen terminator



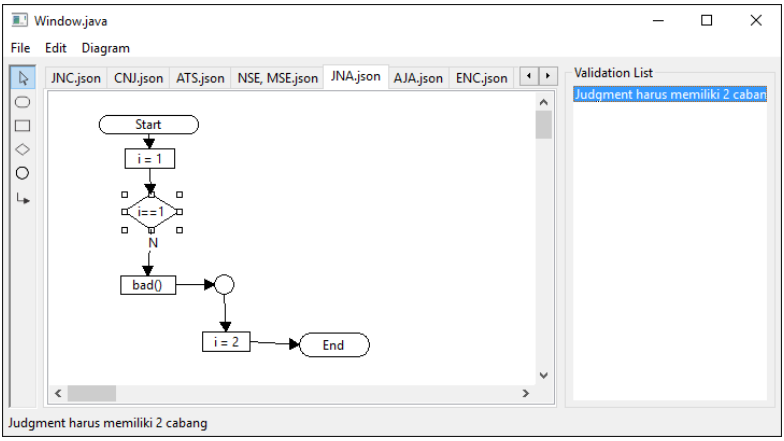
Gambar 5.3 Uji coba *judgment* yang tidak memiliki pasangan *convergence*



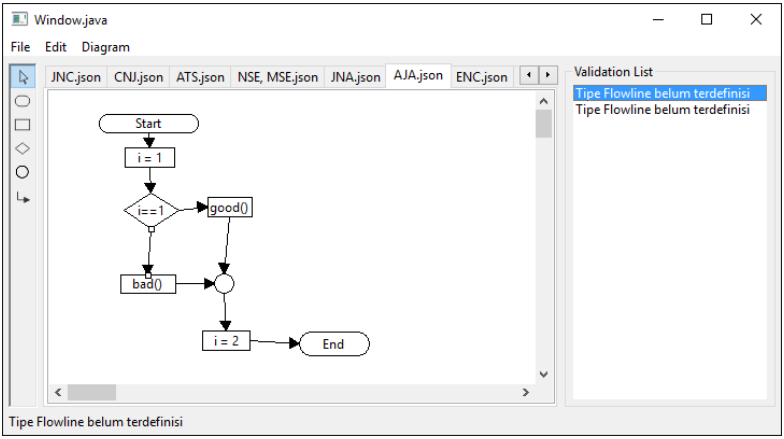
Gambar 5.4 Uji coba *convergence* yang tidak memiliki pasangan *judgment*



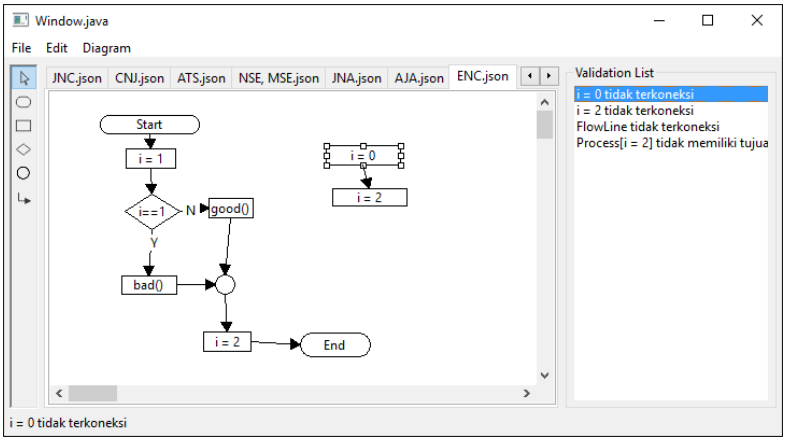
Gambar 5.5 Uji coba aliran yang tidak sesuai



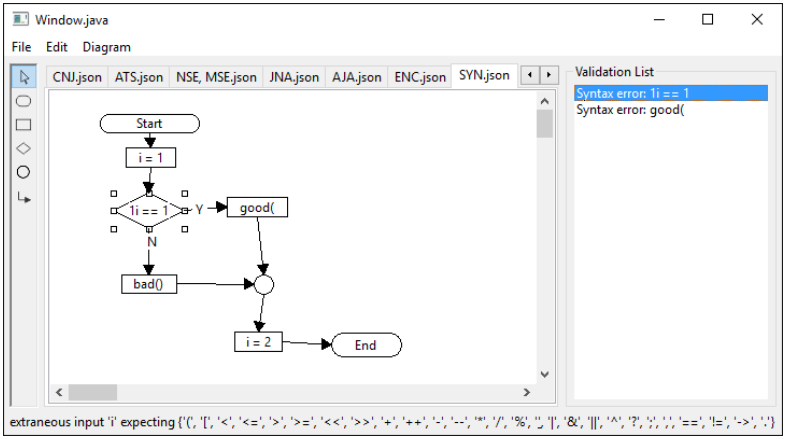
Gambar 5.6 Uji coba judgment tidak memiliki anak berjumlah dua



Gambar 5.7 Uji coba aliran judgment belum terdefinisi



Gambar 5.8 Uji coba elemen tidak terkoneksi



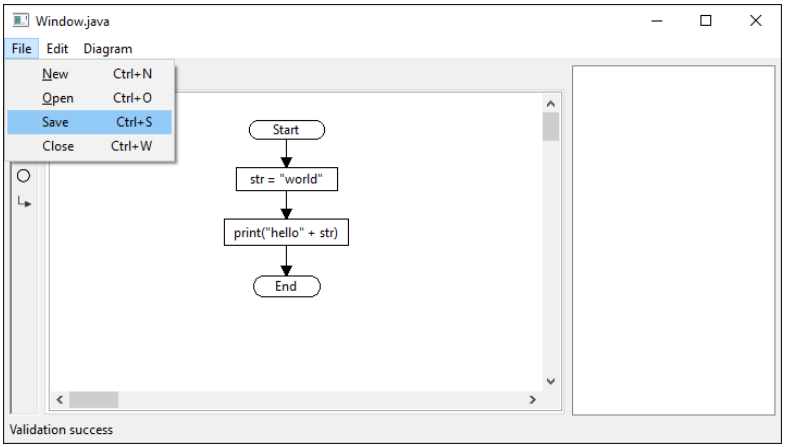
Gambar 5.9 Uji coba kesalahan sintaks

5.2.2.3. Uji Coba Menyimpan dan Membuka File Diagram Alir

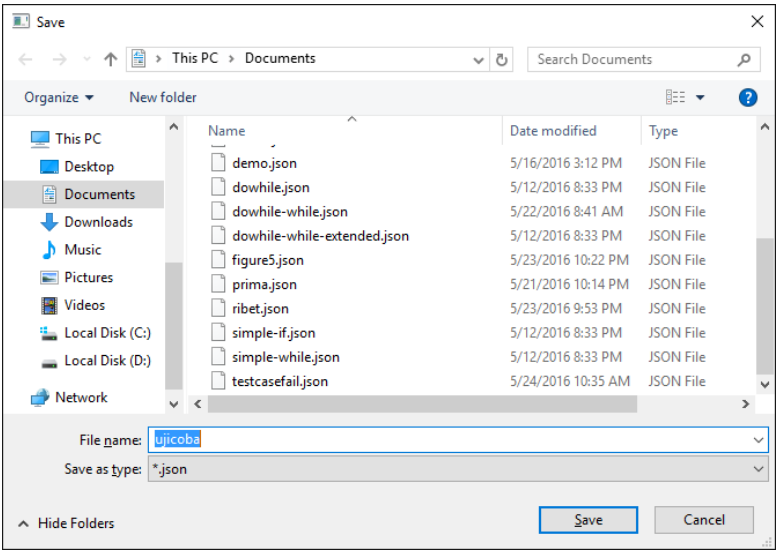
Uji coba menyimpan dan membuka file diagram alir berfungsi untuk mengetahui keberhasilan aplikasi dalam menyimpan diagram alir ke dalam file dan membukanya kembali. Hasil dari pengujian dapat dilihat pada Tabel 5.4. Gambar 5.10 menunjukkan saat pengguna memilih “save” dan “open”. Gambar 5.11 dan Gambar 5.12 menunjukkan jendela menyimpan dan membuka file diagram alir. Hasil uji coba dapat dilihat pada Gambar 5.13.

Tabel 5.4 Uji Coba Menyimpan dan Membuka File Diagram Alir

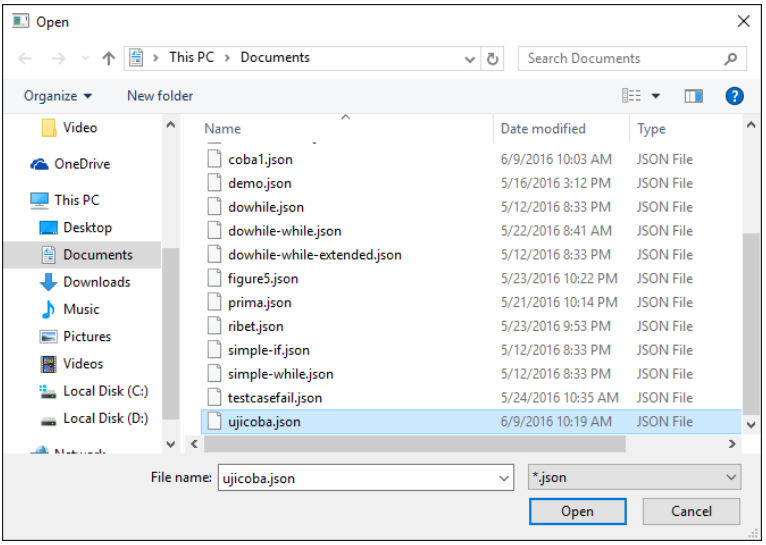
ID	UJ- 03
Nama	Uji Coba Menyimpan dan Membuka File Diagram Alir
Tujuan Uji Coba	Pengguna dapat menyimpan dan membuka file diagram alir
Kondisi awal	Sudah ada diagram alir yang terbuka
Skenario	Pengguna pilih menu “file”, lalu “save”. Pilih file tempat menyimpan. Lalu pengguna pilih menu “file”, lalu “open”. Pilih file yang sebelumnya disimpan.
Keluaran yang diharapkan	Diagram alir yang muncul setelah dibuka kembali sama dengan diagram alir yang disimpan
Hasil uji coba	Berhasil
Kondisi akhir	Terdapat diagram alir yang sebelumnya dibuka.



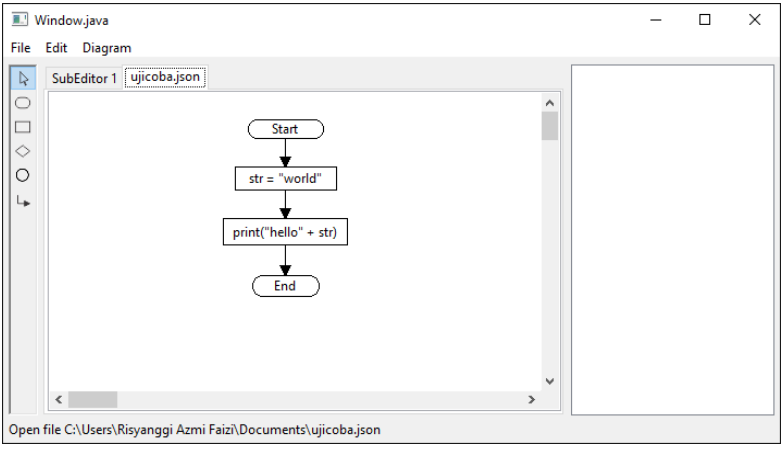
Gambar 5.10 Pilih Menu "Save" dan "Open"



Gambar 5.11 Uji Coba Menyimpan File Diagram Alir



Gambar 5.12 Uji Coba Membuka File Diagram Alir



Gambar 5.13 Hasil Uji Coba Membuka File Diagram Alir

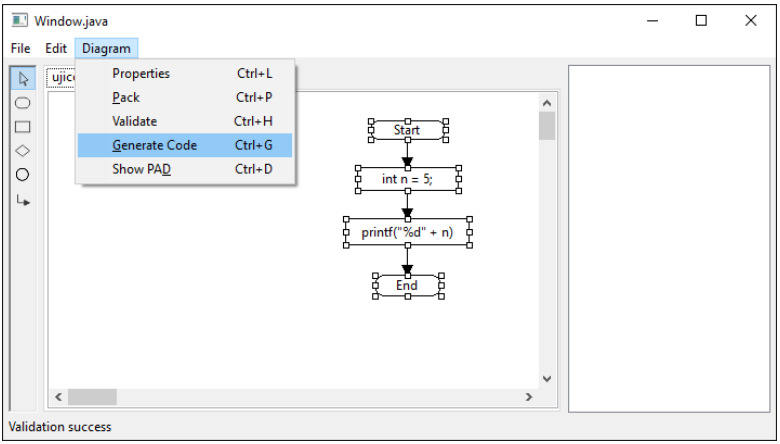
5.2.2.4. Uji Coba Mengkonversi Diagram Alir Menjadi Kode Sumber

Uji coba mengkonversi diagram alir menjadi kode sumber berfungsi untuk mengetahui keberhasilan aplikasi dalam mengkonversi diagram alir yang dibuat oleh pengguna menjadi kode sumber.

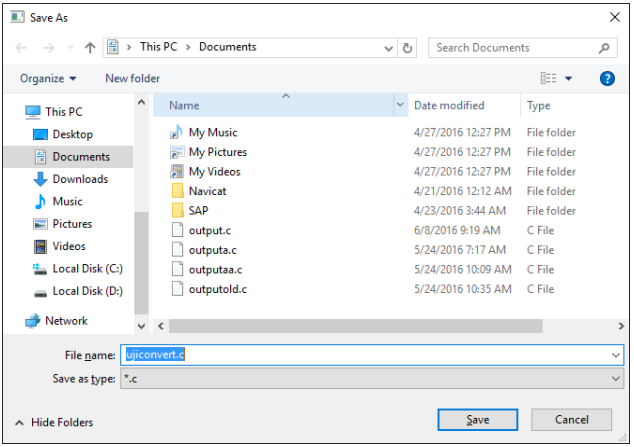
Gambar 5.14 menunjukkan saat pengguna memilih menu "generate code". Gambar 5.15 menunjukkan jendela penyimpanan file kode sumber hasil konversi. Kode Sumber 5.1 adalah kode sumber hasil konversi dari diagram alir.

Tabel 5.5 Uji Coba Mengkonversi Diagram Alir Menjadi Kode Sumber

ID	UJ- 04
Nama	Uji Coba Mengkonversi Diagram Alir Menjadi Kode Sumber
Tujuan Uji Coba	Pengguna dapat membuat diagram alir menggunakan tool-tool yang disediakan pada aplikasi.
Kondisi awal	Sudah ada diagram alir yang terbuka.
Skenario	Pengguna pilih menu "diagram", lalu pilih "generate code". Pengguna menyimpan kode sumber ke file, lalu meyocokkan kode sumber dengan diagram alir yang telah dibuat.
Keluaran yang diharapkan	Diagram alir dan kode sumber cocok secara semantik.
Hasil uji coba	Berhasil
Kondisi akhir	Terdapat kode sumber dalam sebuah file.



Gambar 5.14 Pilih Menu "Generate Code"



Gambar 5.15 Simpan Hasil Uji Coba Konversi

```
#include <stdio.h>

void main() {
    int n = 5;;
    printf("%d" + n);
}
```

Kode Sumber 5.1 Hasil Uji Coba Konversi Diagram Alir

5.2.2.5. Uji Coba Melihat PAD

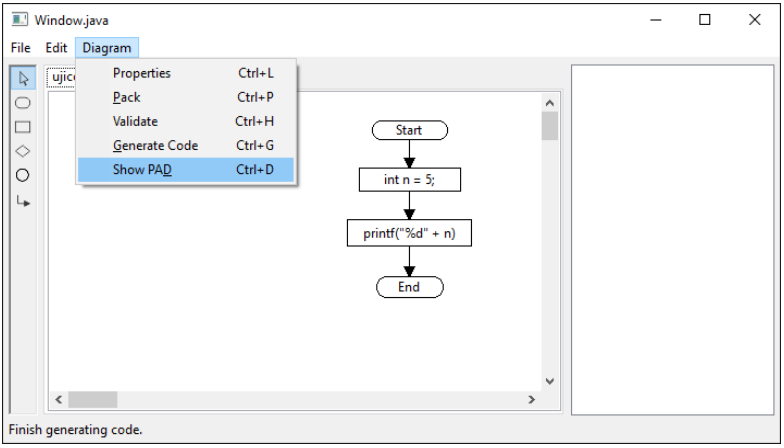
Uji coba melihat PAD berfungsi untuk mengetahui keberhasilan aplikasi dalam menampilkan PAD hasil konversi dari diagram alir. Hasil dari pengujian dapat dilihat pada Tabel 5.6. Kode Sumber 5.2 adalah visualisasi PAD dalam bentuk text. Nama elemen adalah yang ada di dalam kurung siku. Text tepat setelah kurung tutup kurung siku adalah isi dari elemen. Indentasi menunjukkan kedalaman PAD.

```
[Start]
  [BlockContainer]
    [Sequence] int n = 5;
    [Sequence] printf("%d" + n)
[End]
```

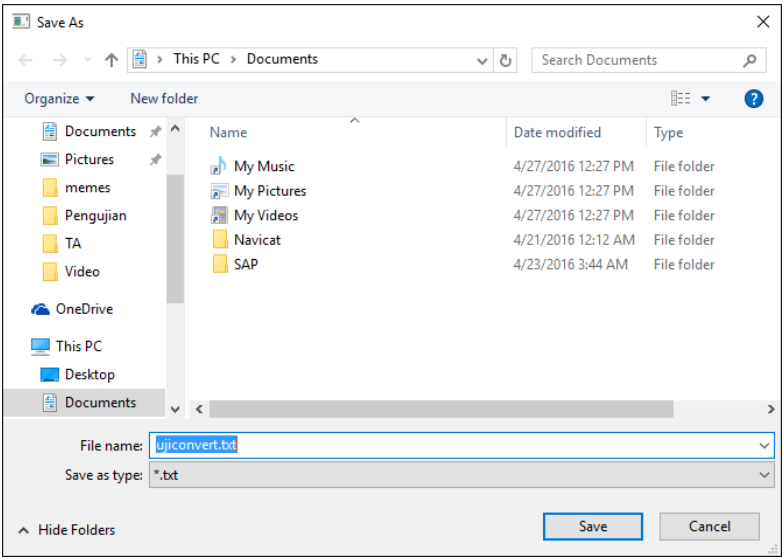
Kode Sumber 5.2 Visualisasi PAD dalam Bentuk Kode

Tabel 5.6 Uji Coba Melihat PAD

ID	UJ- 05
Nama	Uji Coba Membuat Diagram Alir
Tujuan Uji Coba	Pengguna dapat membuat diagram alir menggunakan tool-tool yang disediakan pada aplikasi.
Kondisi awal	-
Skenario	Pengguna membuat diagram alir sederhana
Keluaran yang diharapkan	Diagram alir terbuat
Hasil uji coba	Berhasil
Kondisi akhir	Terdapat diagram yang dibuat oleh pengguna pada aplikasi



Gambar 5.16 Pilih Menu "Show PAD"



Gambar 5.17 Menyimpan PAD

5.2.3. Skenario Pengujian Terhadap Soal Pemrograman

Pada subbab ini dijelaskan beberapa skenario uji coba menyelesaikan soal pemrograman dengan aplikasi konversi diagram alir menjadi kode sumber. Pengujian melibatkan orang-orang yang paham konsep diagram alir dan kosep dasar pemrograman dengan pengujian soal pemrograman sedarhana.

Penguji akan mengerjakan beberapa soal pemrograman dengan membuat diagram alir lalu mengkonversi diagram alir tersebut menjadi kode sumber. Tabel 5.7 menunjukkan soal yang digunakan pada tahap pengujian soal pemrograman.

Setelah penguji melakukan pengujian terhadap soal pemrograman, penguji akan diwawancarai dengan beberapa pertanyaan untuk mengetahui hal-hal sebagai berikut:

1. Kesesuaian fungsionalitas dengan kebutuhan pengguna
2. Korelasi diagram alir dengan kode sumber
3. Keabsahan validasi diagram alir
4. Masukan untuk aplikasi di masa mendatang

Tabel 5.7 Soal Pengujian

Soal 1: Faktorial Buatlah program untuk menghitung faktorial dari angka n yang diinputkan. 1 <= n < 30. Contoh input: 5 Contoh Output: 120
Soal 2: Kabisat Buatlah program untuk mengetahui apakah sebuah tahun n adalah tahun kabisat. Jika n habis dibagi 400, maka n tahun kabisat. Jika n tidak habis dibagi 400 dan habis dibagi 100, maka n bukan tahun kabisat. Jika n tidak habis dibagi 400 dan tidak habis dibagi 100 dan habis dibagi 4, maka n tahun kabisat.

Jika n tidak habis dibagi 400, 100, dan 4, maka n bukan tahun kabisat. Jika tahun kabisat, maka tampilkan "Kabisat" tanpa tanda petik. Jika bukan tahun kabisat, maka tampilkan "Bukan Kabisat" tanpa tanda petik. Contoh input: 2015 Contoh output: Bukan Kabisat

5.2.4. Hasil Pengujian Terhadap Soal Pemrograman

Hasil dari skenario pengujian terhadap soal pemrograman dijelaskan di Tabel 5.8. Hasil skenario pengujian menunjukkan bahwa ketiga penguji dapat menyelesaikan kedua soal yang diberikan menggunakan aplikasi konversi diagram alir menjadi kode sumber. Tabel 5.9, Tabel 5.10, dan Tabel 5.11 menunjukkan hasil wawancara terhadap penguji.

Tabel 5.8 Hasil Skenario Pengujian Soal Pemrograman

Nama	Soal 1	Soal 2
Djuned Fernando Djusdek	Sukses	Sukses
M Arief Ridwan	Sukses	Sukses
Hafiz Nuzal Djufri	Sukses	Sukses

Tabel 5.9 Hasil Wawancara Terhadap Penguji ke-1

Nama : M Arief Ridwan
NRP : 5112100097
Email : aridwann@gmail.com

No	Pertanyaan
1	Menurut anda, apakah aplikasi sudah dapat berjalan dengan baik?
	Aplikasi sudah berjalan dengan bagus.
2	Apakah fungsi yang dibutuhkan untuk membuat diagram alir sudah ada?

	Ya, fungsi yang dibutuhkan sudah ada.
3	Apakah korelasi diagram alir dan kode sumber yang dihasilkan sudah sesuai?
	Sesuai dengan yang dituliskan pada aplikasi.
4	Apakah aplikasi dapat memvalidasi diagram alir?
	Ya, aplikasi dapat memvalidasi diagram alir
5	Apa saran anda untuk aplikasi ini?
	UI perlu diperbaiki. Tidak semua orang paham convergence.

Tabel 5.10 Hasil Wawancara Terhadap Penguji ke-2

Nama : Djuned Fernando Djusdek
NRP : 5112100071
Email : santen_suru@gmail.com

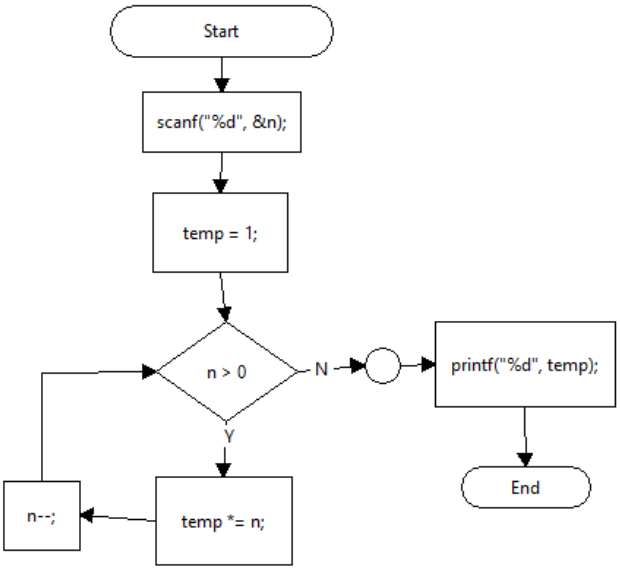
No	Pertanyaan
1	Menurut anda, apakah aplikasi sudah dapat berjalan dengan baik?
	Aplikasi sudah berjalan dengan sangat baik.
2	Apakah fungsi yang dibutuhkan untuk membuat diagram alir sudah ada?
	Ya, fungsi yang dibutuhkan sudah ada.
3	Apakah korelasi diagram alir dan kode sumber yang dihasilkan sudah sesuai?
	Sudah sesuai dengan batasan-batasan.
4	Apakah aplikasi dapat memvalidasi diagram alir?
	Aplikasi sudah dapat menvalidasi diagram alir.
5	Apa saran anda untuk aplikasi ini?
	UI perlu diperbaiki. Seharusnya variabel dapat didefine di awal.

Tabel 5.11 Hasil Wawancara Terhadap Penguji ke-3

Nama : Hafiz Nuzal Djufri
NRP : 5112100170
Email : hafiz.nual@gmail.com

No	Pertanyaan
1	Menurut anda, apakah aplikasi sudah dapat berjalan dengan baik?
	Aplikasi sudah berjalan dengan sangat baik.
2	Apakah fungsi yang dibutuhkan untuk membuat diagram alir sudah ada?
	Ya, fungsi yang dibutuhkan sudah ada.
3	Apakah korelasi diagram alir dan kode sumber yang dihasilkan sudah sesuai?
	Sudah.
4	Apakah aplikasi dapat memvalidasi diagram alir?
	Aplikasi sudah dapat menvalidasi diagram alir.
5	Apa saran anda untuk aplikasi ini?
	Butuh beberapa perbaikan pada proses pembuatan elemen diagram. Jadikan tool pointer sebagai default setelah menggunakan tool yang lain

Diagram alir, PAD hasil konversi dari diagram alir, dan kode sumber hasil konversi dari PAD berturut turut ada pada Gambar 5.18, Gambar 5.19, Gambar 5.20, Gambar 5.21, Gambar 5.22, Gambar 5.23, Kode Sumber 5.3, Kode Sumber 5.4, Kode Sumber 5.5, Kode Sumber 5.6, Kode Sumber 5.7, Kode Sumber 5.8, Kode Sumber 5.9, Kode Sumber 5.10, Kode Sumber 5.11, Kode Sumber 5.12, Kode Sumber 5.13, dan Kode Sumber 5.14.



Gambar 5.18 Diagram Alir dari Penguji ke-1 Soal ke-1

```
[Start]
  [BlockContainer]
    [Sequence] scanf("%d", &n);
    [Sequence] temp = 1;
    [While] n > 0
      Child:
      [BlockContainer]
        [Sequence] temp *= n;
        [Sequence] n--;
    [Sequence] printf("%d", temp);
[End]
```

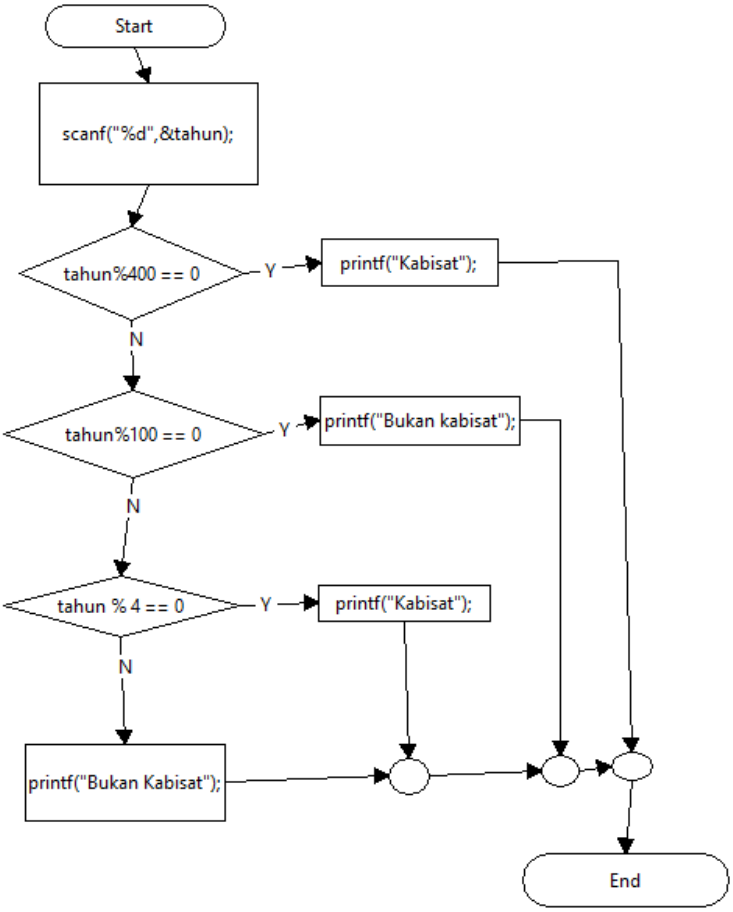
Kode Sumber 5.3 PAD dari Penguji ke-1 Soal ke-1

```
#include <stdio.h>

void main() {
    scanf("%d", &n);
    temp = 1;
```

```
    while (n > 0) {
        temp *= n;
        n--;
    }
    printf("%d", temp);
}
```

Kode Sumber 5.4 Kode Sumber dari Penguji ke-1 Soal ke-1



Gambar 5.19 Diagram Alir dari Penguji ke-1 Soal ke-2

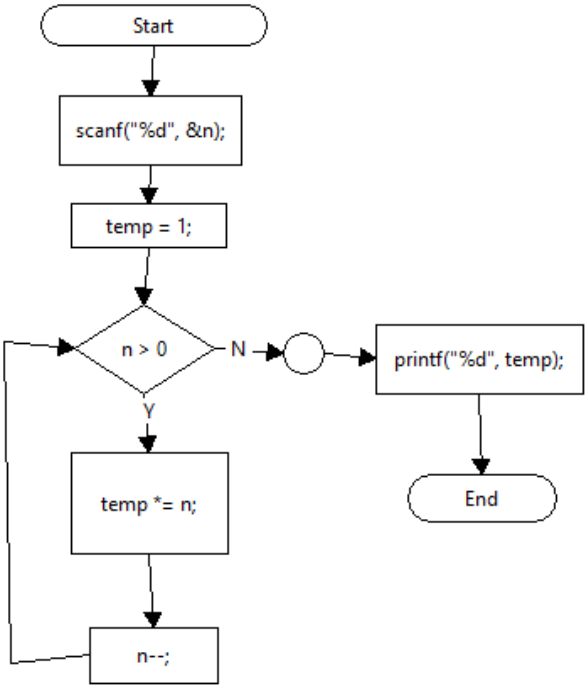
```
[Start]
  [BlockContainer]
    [Sequence] scanf("%d",&tahun);
    [Selection] tahun%400 == 0
      Child Yes:
      [BlockContainer]
        [Sequence] printf("Kabisat");
      Child No:
      [BlockContainer]
        [Selection] tahun%100 == 0
          Child Yes:
          [BlockContainer]
            [Sequence] printf("Bukan kabisat");
          Child No:
          [BlockContainer]
            [Selection] tahun % 4 == 0
              Child Yes:
              [BlockContainer]
                [Sequence] printf("Kabisat");
              Child No:
              [BlockContainer]
                [Sequence] printf("Bukan Kabisat");
[End]
```

Kode Sumber 5.5 PAD dari Penguji ke-1 Soal ke-2

```c
#include <stdio.h>

void main() {
    scanf("%d",&tahun);
    if (tahun%400 == 0) {
        printf("Kabisat");
    } else {
        if (tahun%100 == 0) {
            printf("Bukan kabisat");
        } else {
            if (tahun % 4 == 0) {
                printf("Kabisat");
            } else {
                printf("Bukan Kabisat");
            }
        }
    }
}
```

Kode Sumber 5.6 Kode Sumber dari Penguji ke-1 Soal ke-2



Gambar 5.20 Diagram Alir dari Penguji ke-2 Soal ke-1

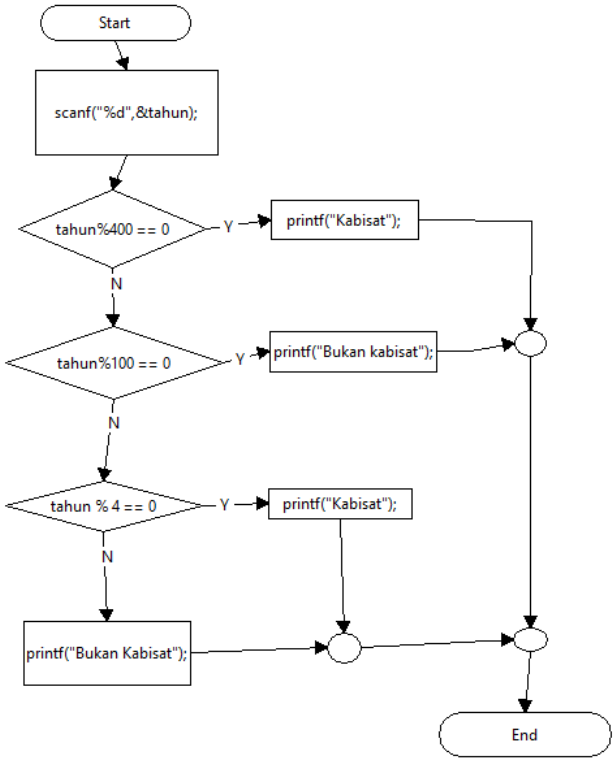
```
[Start]
  [BlockContainer]
    [Sequence] scanf("%d", &n);
    [Sequence] temp = 1;
    [While] n > 0
      Child:
      [BlockContainer]
        [Sequence] temp *= n;
        [Sequence] n--;
    [Sequence] printf("%d", temp);
[End]
```

Kode Sumber 5.7 PAD dari Penguji ke-2 Soal ke-1

```c
#include <stdio.h>

void main() {
    scanf("%d", &n);
    temp = 1;
    while (n > 0) {
        temp *= n;
        n--;
    }
    printf("%d", temp);
}
```

Kode Sumber 5.8 Kode Sumber dari Penguji ke-2 Soal ke-1



Gambar 5.21 Diagram Alir dari Penguji ke-2 Soal ke-2

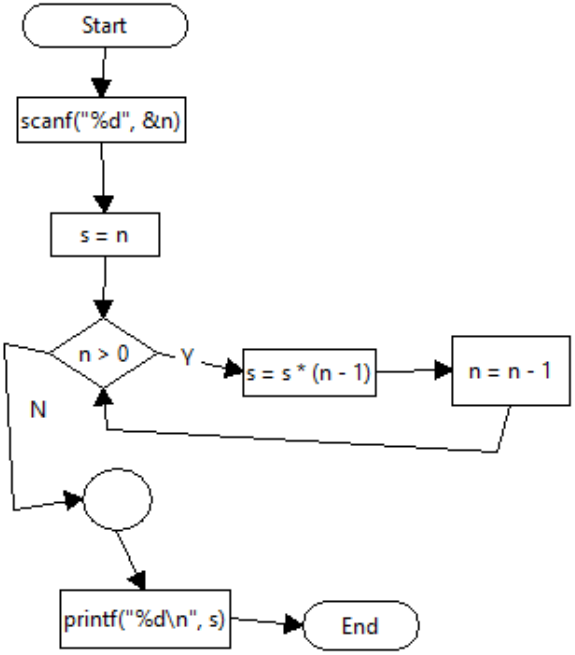
```
[Start]
  [BlockContainer]
    [Sequence] scanf("%d",&tahun);
    [Selection] tahun%400 == 0
      Child Yes:
      [BlockContainer]
        [Sequence] printf("Kabisat");
      Child No:
      [BlockContainer]
        [Selection] tahun%100 == 0
          Child Yes:
          [BlockContainer]
            [Sequence] printf("Bukan kabisat");
          Child No:
          [BlockContainer]
            [Selection] tahun % 4 == 0
              Child Yes:
              [BlockContainer]
                [Sequence] printf("Kabisat");
              Child No:
              [BlockContainer]
                [Sequence] printf("Bukan Kabisat");
[End]
```

Kode Sumber 5.9 PAD dari Penguji ke-2 Soal ke-2

```c
#include <stdio.h>

void main() {
    scanf("%d",&tahun);
    if (tahun%400 == 0) {
        printf("Kabisat");
    } else {
        if (tahun%100 == 0) {
            printf("Bukan kabisat");
        } else {
            if (tahun % 4 == 0) {
                printf("Kabisat");
            } else {
                printf("Bukan Kabisat");
            }
        }
    }
}
```

Kode Sumber 5.10 Kode Sumber dari Penguji ke-2 Soal ke-2



Gambar 5.22 Diagram Alir dari Penguji ke-3 Soal ke-1

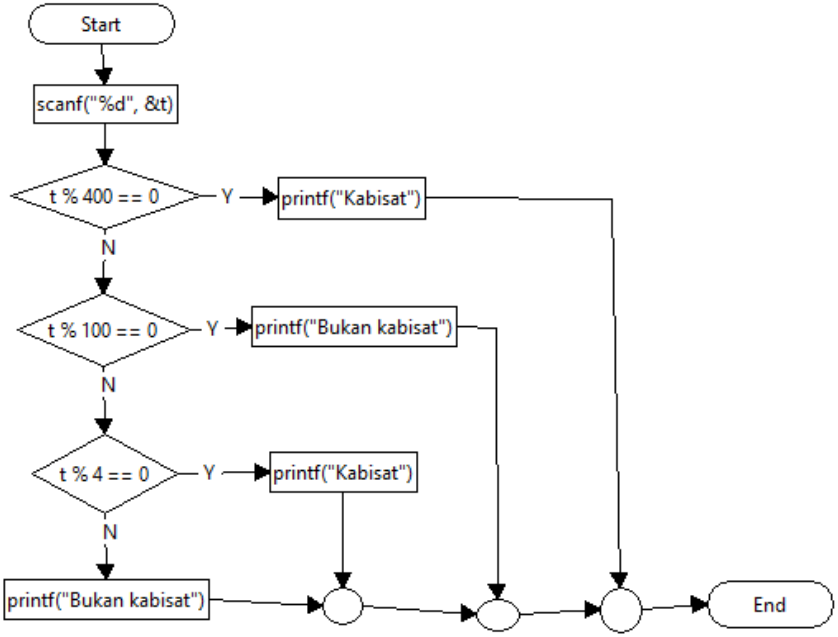
```
[Start]
  [BlockContainer]
    [Sequence] scanf("%d", &n)
    [Sequence] s = n
    [While] n > 0
      Child:
      [BlockContainer]
        [Sequence] s = s * (n - 1)
        [Sequence] n = n - 1
    [Sequence] printf("%d\n", s)
[End]
```

Kode Sumber 5.11 PAD dari Penguji ke-3 Soal ke-1

```c
#include <stdio.h>

void main() {
    scanf("%d", &n);
    s = n;
    while (n > 0) {
        s = s * (n - 1);
        n = n - 1;
    }
    printf("%d\n", s);
}
```

Kode Sumber 5.12 Kode Sumber dari Penguji ke-3 Soal ke-1



Gambar 5.23 Diagram Alir dari Penguji ke-3 Soal ke-2

```
[Start]
  [BlockContainer]
    [Sequence] scanf("%d", &t)
    [Selection] t % 400 == 0
      Child Yes:
      [BlockContainer]
        [Sequence] printf("Kabisat")
      Child No:
      [BlockContainer]
        [Selection] t % 100 == 0
          Child Yes:
          [BlockContainer]
            [Sequence] printf("Bukan kabisat")
          Child No:
          [BlockContainer]
            [Selection] t % 4 == 0
              Child Yes:
              [BlockContainer]
                [Sequence] printf("Kabisat")
              Child No:
              [BlockContainer]
                [Sequence] printf("Bukan kabisat")
[End]
```

Kode Sumber 5.13 PAD dari Penguji ke-3 Soal ke-2

```c
#include <stdio.h>

void main() {
    scanf("%d", &t);
    if (t % 400 == 0) {
        printf("Kabisat");
    } else {
        if (t % 100 == 0) {
            printf("Bukan kabisat");
        } else {
            if (t % 4 == 0) {
                printf("Kabisat");
            } else {
                printf("Bukan kabisat");
            }
        }
    }
}
```

Kode Sumber 5.14 Kode Sumber dari Penguji ke-3 Soal ke-2

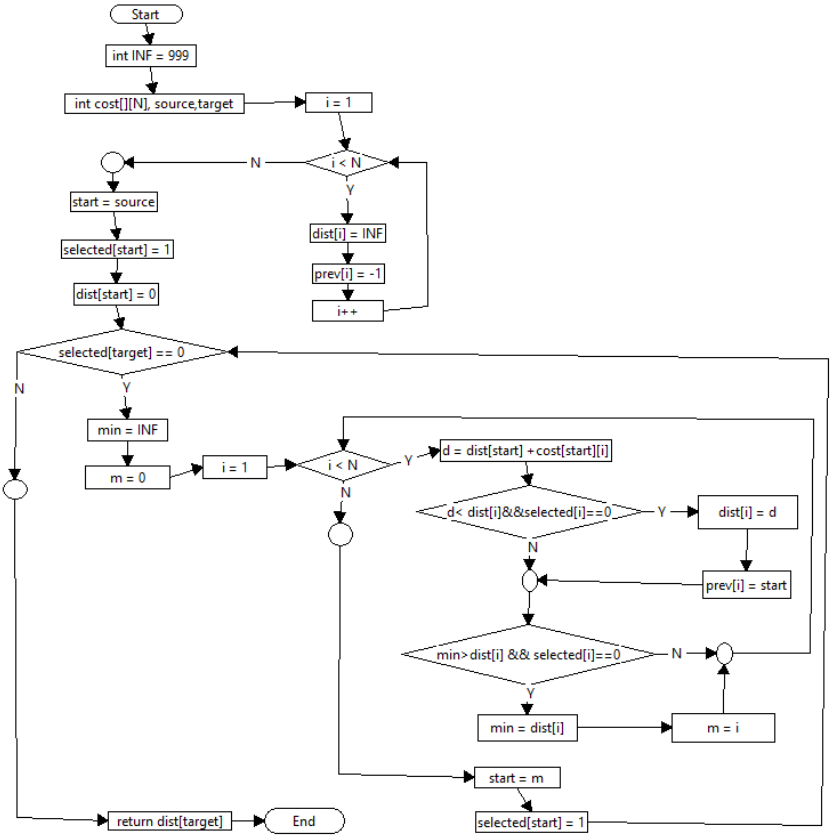
5.2.5. Skenario dan Hasil Pengujian Terhadap Soal Kompleks

Pengujian pada bab ini adalah mengkonversi diagram alir algoritma dijkstra menjadi kode sumber menggunakan aplikasi konversi diagram alir menjadi kode sumber. Persoalan dijkstra adalah mencari jarak terpendek antara dua titik seperti pada Gambar 5.24. Jika nilai jarak w(u,v) adalah INF, berarti titik u tidak bisa langsung menuju ke tiik v.

w(u,v)	a	b	c	d	d	f
a	0	4	2	INF	INF	INF
b	4	0	1	5	INF	INF
c	2	1	0	8	10	INF
d	INF	5	8	0	2	6
e	INF	INF	10	2	0	3
f	INF	INF	INF	6	3	0

Gambar 5.24 Pemetaan titik dan jarak untuk dijkstra

Diagram alir untuk menyelesaikan algoritma dijkstra ada pada Gambar 5.25. Konversi diagram alir dijkstra menghasilkan kode sumber berbahasa C pada Kode Sumber 5.15. Kode sumber yang dihasilkan ada dalam fungsi main, sedangkan seharusnya kode sumber yang dihasilkan ada dalam fungsi khusus dijkstra dengan parameter.



Gambar 5.25 Diagram alir algoritma dijkstra

```
#include <stdio.h>

void main() {
    int INF = 999;
    int cost[][N], source,target;
    i = 1;
    while (i < N) {
        dist[i] = INF;
        prev[i] = -1;
        i++;
```

```
    }
    start = source;
    selected[start] = 1;
    dist[start] = 0;
    while (selected[target] == 0) {
        min = INF;
        m = 0;
        i = 1;
        while (i < N) {
            d = dist[start] +cost[start][i];
            if (d< dist[i]&&selected[i]==0) {
                dist[i] = d;
                prev[i] = start;
            }
            if (min>dist[i] && selected[i]==0) {
                min = dist[i];
                m = i;
            }
        }
        start = m;
        selected[start] = 1;
    }
    return dist[target];
}
```

Kode Sumber 5.15 Algoritma dijkstra hasil konversi

5.3. Evaluasi

Dari hasil pengujian, diambil beberapa informasi yang penting. Aplikasi konversi diagram alir sudah mencukupi fungsional untuk membuat diagram alir, namun ada beberapa hal yang perlu diperhatikan.

Pada saat membuat konektor antar elemen diagram alir, garis tidak dapat rapi secara otomatis. Diperlukan ketelitian untuk membuat diagram alir menjadi rapi. Untuk menggeser elemen, aplikasi tidak menunjukkan perubahan setiap pergeseran perangkat mouse sebelum pengguna melepas drag pada mouse.

Sebaiknya jadikan tool pointer sebagai default setelah menggunakan tool yang lain.

Elemen “convergence” adalah elemen yang asing, bukan merupakan elemen standar yang dipakai pada diagram alir meskipun dalam aplikasi ini dibutuhkan elemen tersebut. Mungkin untuk selanjutnya elemen “convergence” dapat dihilangkan, namun aplikasi tetap dapat menentukan titik akhir dari sebuah elemen “judgment”.

Aplikasi konversi diagram alir sudah dapat memvalidasi diagram alir, termasuk kesalahan sintaks dan ketidaksesuaian elemen “judgment” dan “convergence” dalam diagram alir.

Aplikasi konversi diagram alir sudah dapat mengkonversi diagram alir menjadi kode sumber dengan baik. Namun aplikasi ini tidak dapat mendeteksi adanya variabel-variabel yang belum dideklarasi. Pengguna harus mengubah kode sumber hasil konversi untuk menambahkan deklarasi variabel. Indentasi pada kode sumber sudah cukup baik.

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan tugas akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap tugas akhir ini di masa yang akan datang.

6.1. Kesimpulan

Dari hasil selama proses perancangan, implementasi, serta pengujian dapat diambil kesimpulan sebagai berikut:

1. Aplikasi telah berhasil membuat diagram alir meskipun diagram alir yang terbuat sulit untuk dirapikan.
2. Aplikasi telah berhasil memvalidasi keabsahan diagram alir.
3. Aplikasi telah berhasil menampilkan *problem analysis diagram* hasil konversi dari diagram alir.
4. Aplikasi telah berhasil mengkonversi diagram alir menjadi kode sumber sesuai batasan.

6.2. Saran

Berikut saran-saran untuk pengembangan dan perbaikan sistem di masa yang akan datang. Diantaranya adalah sebagai berikut:

1. Menyempurnakan tool-tool untuk membuat diagram alir, terutama mengganti tool ke pointer tool setelah menggunakan tool yang lain.
2. Memperbaiki dan memperbagus tampilan.
3. Menyediakan fungsi untuk merapikan hasil diagram alir.
4. Elemen "convergence" dapat dihilangkan, namun aplikasi tetap dapat menentukan titik akhir dari sebuah elemen "judgment".

5. Aplikasi otomatis menambahkan deklarasi variabel pada hasil konversi dari diagram alir berdasarkan variabel yang disebutkan di diagram alir.
6. Perlu ditambahkan fitur *undo* dan *redo*.

DAFTAR PUSTAKA

[1] X.-H. Wu, M.-C. Qu, Z.-Q. Liu and J.-Z. Li, "Automatic Conversion of Structured Flowcharts into Problem Analysis Diagram for Generation of Codes," *Journal of Software,* vol. 7, no. 5, p. 1109, 2012.

[2] European Computer Manufacturers Association, Standard ECMA-4 Flow Charts, Geneva: ECMA, 1966.

[3] "About Problem Analysis Diagram," Kumamoto University, [Online]. Available: http://www2.ee.knct.ac.jp/el/E2/L210/algorism/pad1.html.

[4] M. Scarpino, S. Holder, S. Ng and L. Mihalkovic, SWT/JFace in Action, Greenwich: Manning Publications Co., 2005.

[5] T. Parr, "ANTLR," 2014. [Online]. Available: http://www.antlr.org/. [Accessed 12 December 2015].

[6] DonnyVerdianNet, "donnyverdian.net," 3 6 2013. [Online]. Available: http://donnyverdian.net/mengobati-phobia-ketinggian-begini-caranya/. [Accessed 3 7 2015].

[7] T. Kuhn and O. Thomann, "Abstract Syntax Tree," 20 November 2006. [Online]. Available: http://www.eclipse.org/articles/Article-JavaCodeManipulation_AST/index.html. [Accessed 11 December 2015].

[8] W.-J. Gong, M.-C. Qu, X.-H. Wu and P.-J. Ma, "The Verification of Structure Identification Algorithm and Error Detection Strategies for Structured Flowchart," *Elsevier,* p. 880, 2012.

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Penulis, Risyanggi Azmi Faizin, lahir di kota Surabaya pada tanggal 29 Juli 1993. Penulis dibesarkan di kota Sidoarjo, Jawa Timur.

Penulis menempuh pendidikan formal di SD Al-Hikmah Surabaya (1999-2005), SMP Al-Hikmah Surabaya (2005-2008), SMAN 15 Surabaya (2008-2011). Pada tahun 2012, penulis melanjutkan pendidikan S1 jurusan Teknik Informatika Fakultas Teknologi Informasi di Institut Teknologi Sepuluh Nopember Surabaya, Jawa Timur.

Di jurusan Teknik Informatika, penulis mengambil bidang minat Algoritma dan Pemrograman atau biasa disingkat menjadi Alpro dan memiliki ketertarikan di bidang desain *web*, perancangan perangkat lunak, *big data*, dan Pemodelan 3D. Penulis aktif sebagai vokalis di Paduan Suara Mahasiswa ITS. Penulis dapat dihubungi melalui alamat surel (*email*) risyanggi@gmail.com.