Panduan Penanganan *Error* Aplikasi

Rolly Maulana Awangga
Github : github.com/awangga
M. Innal Kariem
Github : github.com/karieminnal
Rayhan Prastya
Github : github.com/rayprastya

Informatics Research Center
Applied Bachelor Program of Informatics Engineering

Bandung
2019

‘Jika Kamu tidak dapat menahan lelahnya belajar,
Maka kamu harus sanggup menahan perihnya Kebodohan.’
Imam Syafi’i

Acknowledgements

Puji dan syukur kami panjatkan hadirat Allah S.W.T atas rahmat dan karunia-Nya kami dapat menyelesaikan panduan penanganan *error* aplikasi ini. Dan tidak lupa juga kami ucapkan kepada rekan dan para dosen yang namanya tidak dapat kami sebutkan satu per satu, yang telah membantu kami dalam proses pengerjaan panduan penanganan *error* aplikasi ini, diharapkan panduan ini dapat berguna bagi para pembaca dan juga menjadi acuan baik itu dalam pemahaman tentang berbagai macam jenis *error* atau proses penyelesaian suatu *error*.

Abstract

Panduan Penanganan *Error* Aplikasi ini dibuat dengan tujuan memberikan pemahaman mendalam tentang *error* kepada para sivitas akademika. Dimulai dari pengenalan berbagai macam *error* hingga cara penyelesaiannya, Panduan ini akan menjabarkan tentang pengenalan berbagai macam *error*, standar penulisan sebuah program, hingga cara penyelesaiannya. Dengan demikian diharapkan semua sivitas akademika dapat memahami berbagai jenis *error* yang terdapat pada suatu program, dan dapat mengatasi *error* yang terdapat pada suatu program.

Contents

Chapter 1

Standar Perlengkapan

Error terjadi pada suatu program akibat ketidaksesuaian penyusunan suatu program dengan standar yang sudah ditetepkan. Dalam kasus penangan *error*, beberapa bahasa pemrograman memiliki *IDE* yang merupakan singkatan dari *Integrated Development Environment* yang dapat melakukan pengecekan *error* secara *realtime*. Disini kita membutuhkan *IDE* dari suatu bahasa pemrograman yang memiliki *variable explorer* . *variable explorer* berfungsi menampilkan konten-konten apasaja yang ada di baris *coding*-an kita, yang bertujuan untuk mempermudah kita untuk menyusun *code* program. Jadi perlengkapan yang kita harus persiapkan adalah :

1. Bahasa pemorgraman
2. *IDE* dari suatu bahasa pemrograman yang memiliki *variable explorer*.

1.1 Jenis *Error*

Error atau bisa disebut dengan kesalahan pada program memiliki berbagai jenis tipe, diantaranya :

1. *Syntax errors*

   Syntax errors adalah *error* yang diakibatkan oleh kesalahan dalam penulisan bahasa program yang tidak dapat dimengerti oleh *compiler* , contohnya adalah sebagai berikut.

```
#include <stdio.h>

int main(void) {
    printf("Hello World!\n";
```

```
    return 0;
}
```

Terdapat *syntax error* pada program ini, *syntax error* disini diakibatkan kurangnya tanda ")" pada akhir baris *code* tersebut. Kurangnya " " , ")" , "]" , "}" pada akhir atau awal baris *code* juga dapat menyebabkan terjadinya *syntax error*.

2. *Semantic errors*

   Semantic errors terjadi akibat tidak tepatnya varibel dengan *statement* yang sudah dibuat, ketidakjelasan logika pada program yang dibuat akan menimbulkan *semantic error* contohnya adalah sebagai berikut.

```
public static void main(String[] args) {
String NPM;
NPM = "Rayhan";
System.out.println(NPM);
}
```

   Program ini akan menghasilkan *semantic error*,*compiler* tetap dapat menjalan kan program tersebut,namun output yang dikeluarkan berupa String. jika output yang diinginkan berupa NPM yang bertipe data INT, maka tipe data yang diinputkan harus sesuai dengan output yang diinginkan.

Chapter 2

Standar Penulisan Program

Dalam pengerjaan suatu program, hendaknya baris *code* yang kita buat sesuai standar dari penulisan program, karena setiap bahasa pemrograman memiliki aturan penulisannya sendiri-sendiri, contohnya pada bahasa C, bahasa C bersifat *case sensitifity* dimana huruf kapital dan huruf kecil memiliki arti yang berbeda,contohnya pada gambar berikut.

```
#include <stdio.h>

int main(void) {
    printf("Hello World!\n");
    Return 0;
}
```

Pada baris *code* berikut terdapat parameter *return* yang seharusnya diketik dengan huruf kecil, karena pada awalan *parameter return* diketik menggunakan huruf kapital, sehingga baris *code* tersebut menghasilkan *error*.

Sehingga pada dasarnya, sebaiknya saat mengerjakan suatu program hendaknya mengikuti standar penulisan dari bahasa program tersebut dan terstruktur, agar memudahkan proses pengerjaan suatu program dan terlihat lebih rapi.

2.1 Standar Penulisan Nama Variabel

Setiap pemberian nama pada variabel hendaknya sesuai dengan isi/*context* dari *code* yang sedang dibuat, untuk mempermudah proses pengembangan sebuah program, contoh penamaan variabel yang baik adalah sebagai berikut.

```
public static void main(String[] args) {
int NPM;
NPM = 1184069;
System.out.println(NPM);
}
```

Didalam program tersebut terdapat sebuah variabel yang diberi nama NPM (Nomor Pokok Mahasiswa), variabel tersebut diberi nama NPM karena *output* yang diharapkan nantinya adalah pencetakan Nomor Pokok Mahasiswa, jika variabel tersebut diberikan nama yang asal-asalan, hal ini akan mempersulit developer itu sendiri saat hendak menggunakan variabel itu kembali karena nama yang diberikan pada variabel itu asal-asalan.

2.2 Standar Penamaan Fungsi

Sama hal-nya dengan pemberian nama pada sebuah variabel, penamaan fungsi pada suatu program hendaknya disesuaikan dengan *output*/isi dari proses fungsi tersebut, karena pemberian nama fungsi yang tidak sesuai dengan *output* atau isi yang diharapkan hanya akan mempersulit developer dalam proses pengembangan sebuah aplikasi, contoh penamaan fungsi yang baik adalah sebagai berikut.

```
def getURL(self,PROYEK):
if PROYEK == '2':
active_url = "https://cobahayo.herokuapp.com/"
else:
active_url = "https://proyek3d4tiv2.herokuapp.com/"
return active_url
```

Output dari program diatas ditunjukkan untuk mahasiswa yang hendak mengikuti proyek 2, yang nantinya jika mahasiswa menginput angka 2 (Proyek 2), maka program akan mengaktifkan *url* `"https://cobahayo.herokuapp.com/"` , jika fungsi tersebut diberikan nama yang asal-asalan, hal ini akan mempersulit developer dalam melakukan pengembangan pada aplikasi.

2.3 Standar Pembuatan Fungsi

Fungsi berfungsi untuk menjalankan suatu tugas yang ditulis dalam 1 blok *code* yang akan dieksekusi apabila fungsi tersebut dipanggil pada program utama. Pembuatan suatu fungsi sebaiknya dibedakan sesuai dengan pekerjaan yang dilakukan oleh fungsi tersebut, tidak dianjurkan untuk mencampurkan berbagai macam pekerjaan pada 1 fungsi, hal ini dapat menimbulkan *error*, sehingga sebaiknya fungsi dibuat untuk 1 pekerjaan saja. Tidak membuat 2 buah fungsi dengan jenis pekerjaan yang sama karena 1 fungsi saja cukup karena fungsi tersebut dapat digunakan secara terus menerus tanpa harus membuat fungsi baru lagi dengan cara kerja yang sama. Isi dari fungsi harus jelas, karena pembuatan fungsi yang tidak memiliki kejelasan akan cara kerjanya dapat menimbulkan *error*, berikut contoh pembuatan fungsi yang baik.

```
<?php
$db = mysqli_connect("localhost","root","","registrasi");

function query($query){
global $db;
$result = mysqli_query($db,$query);
$rows = [];
while( $row = mysqli_fetch_assoc($result)) {
$rows[] = $row;
}
return $rows;
}

function ubah($data){
global $db;

$id = $data["id"];
$Nama= htmlspecialchars($data["nama"]);
$NPM= htmlspecialchars($data["npm"]);
$Email= htmlspecialchars($data["email"]);
$Jurusan= htmlspecialchars($data["jurusan"]);
$Dosen= htmlspecialchars($data["dosen"]);
$laporanLama= htmlspecialchars($data["laporanLama"]);
```

```
if ($_FILES ['laporan']['error'] === 4) {
$Laporan = $laporanLama;
} else {
$Laporan = upload();
}
$query = "UPDATE mahasiswa SET
npm = '$NPM',
nama = '$Nama',
email = '$Email',
jurusan = '$Jurusan',
dosen = '$Dosen',
laporan = '$Laporan'
WHERE id = $id
";
mysqli_query($db,$query);

return mysqli_affected_rows($db);
}

?>
```

2.4 Pembuatan Program Utama

Program utama adalah bagian dari program yang memiliki fungsi untuk memanggil *class* dan *method* yang dibutuhkan untuk menjalankan suatu program suatu program, dalam standar penulisan program sebaiknya program utama ditempatkan di tempat yang berbeda dari penempatan fungsi, *class*, dan *method*. Jika program utama ditempatkan pada satu tempat yang sama dengan fungsi, *class*, dan *method* program akan terlihat tidak tertata rapi, dan bahkan dapat menyebabkan terjadinnya *error*. Contoh penempatan program utama yang baik adalah sebagai berikut.

```
public static double factorial(double d) {
// mengurutkan elemen
if (d<=1)
{
return 1;
```

```
} else {
return d * factorial(d-1);
}
}

public static void main(String[] args) {


System.out.println(factorial(3));

}

}
```

Pembuatan sebuah fungsi, *class*, dan *method* dapat dilakukan diluar program utama, sehingga program utama tinggal memanggil fungsi, *class*, dan *method* yang dibutuhkan untuk menjalankan sebuah program.

2.5 Pemberian Komentar

Pemberian komentar berperan penting dalam proses pengembangan sebuah program apalagi jika bekerja dalam suatu tim, pemberian komentar dapat membantu untuk memperjelas bagian fungsi yang masih ambigu, atau komentar dapat digunakan sebagai media komunikasi antar developer dalalm proses pengembangan suatu program.

```
//------------ faktorial-----------------------

public static double factorial(double d) {
// mengurutkan elemen
if (d<=1)
{
return 1;
} else {
return d * factorial(d-1);
}
}
```

```
//----------- running program----------------------------

public static void main(String[] args) {


System.out.println(factorial(3));

}

}
```

Sebuah komentar dapat ditandai dengan "simbol-simbol" berikut.

1. "//—-"
2. "//"
3. "/* */"
4. "<!-- -->"

Chapter 3

Langkah-Langkah Penanganan *Error*

Dalam suatu program terdapat kesalahan *internal* maupun *external* kesalahan *internal* contoh kesalahannya adalah *code* program yang sedang dibuat mengalami kesalahan *syntax* yang membuat *code* program tersebut menjadi *error* dan menjadikan program tersebut tidak bisa dijalankan dengan baik. selanjutnnya untuk kesalahan *external* bisa muncul karena ada kesalahan dalam *database*, fungsi API (antarmuka pemrograman aplikasi), perbedaan versi interpreter dan *compiler* dll. berikut adalah langkah-langkah untuk menangani error tersebut.

3.1 kesalahan *internal*

1. *Syntax Error*

   Syntax error merupakan jenis kesalahan yang terjadi akibat perintah yang diketikan tidak sesuai dengan aturan *code* bahasa pemrograman yang sedang digunakan. Karena setiap bahasa pemrograman memiliki aturan pengkodean yang harus dipatuhi. Contohnya pada bahasa pemrograman C, setiap perintah harus diakhiri dengan tanda titik koma (;), jika tidak diakhiri dengan titik koma(;) maka program akan menampilkan pesan *syntax error* saat dijalankan. Maka dari itu untuk menghindari kesalahan dalam penulisan program kita harus mengetahui tata cara penulisan program atau aturan dalam bahasa pemrograman tersebut agar tidak terjadi kesalahan *syntax*.

2. *Passing Variable Error Passing variable error* merupakan jenis *error* yang terjadi akibat kesalahan dalam menginput *variable*. berikut contoh *Passing variable error*.

```
public static void main(String[] args) {
String NPM;
NPM = "Rayhan";
System.out.println(NPM);
}
```

Program ini merupakan contoh *Passing variable error*. karena tipe data yang dinputkan salah, seharusnya tipe data yang diinputkan berupa *integer* (angka) bukan *String*.

3. *Run-time Error*

   Run-time Error merupakan jenis kesalahan yang terjadi karena ketika *code* program melakukan sesuatu yang tidak memungkinkan. Contohnya dalam bahasa C++, Kebocoran memori Program-program secara terus menerus menggunakan *RAM (random access memory* dan mencegah lokasi memori dari digunakan untuk tugas-tugas lain setelah pekerjaan selesai. Menjalankan loop tak terbatas atau tidak membatalkan memori yang terpakai dapat menyebabkan *run-time error*. untuk mengatasinya kita tidak boleh salah dalam mengalokasikan memori atau program hendak mengakses file namun file tersebut tidak ditemukan.

4. *Logical Error*

   Logical Error kesalahan yang satu ini merupakan jenis *error* yang paling susah terdeteksi karena terjadinya bukan karena kesalahan penulisan *syntax* atau kesalahan proses *run-time*, namun kesalahan dari sisi programmer, dalam hal ini algoritma yang digunakan. Karena logikanya salah, tentunya *output* yang dihasilkan juga akan salah. Untuk mendeteksi dimana letak kesalahannya, bukanlah hal yang mudah. Terkadang kita harus mengurutkan algoritma yang digunakan baris per baris *line-by-line*.

3.2 kesalahan *external*

1. *Database*

   Database adalah kumpulan informasi atau data yang dinormalisasi agar tidak terjadi redudansi yang disimpan dalam media elektronik. fungsi *database* untuk menyimpan data yang nanti akan digunakan kembali. dalam *database* juga tidak luput dari kesalahan, kesalahan yang terdapat pada database bisa sangat

berakibat fatal dalam sebuah aplikasi karena *database* menyimpan semua data-data yang akan digunakan atau data yang di*input*kan oleh *user*. kesalahan dalam merancang *database* dapat dicegah dengan beberapa langkah sebagai berikut.

(a) Penyalahgunaan tipe data

Sebelum merancang database apapun, kita harus mengetahui apa saja yang dibutuhkan. Misalnya, database yang anda pakai dapat menawarkan jenis *INTERGER*, tapi sebaiknya menggunakan *TINYINT* untuk menyimpannya. Kolom tanggal dan waktu, *floating point* dan angka desimal. Beberapa *database* bahkan mendukung *array* Jadi, apapun database yang dipilih jangan salah untuk mendefinisikan tipe data yang tepat untuk setiap kolom. Hal ini dapat menghemat banyak *cost* dan *space* pada penyimpanan anda.

(b) Menggunakan Tipe Data Selain *integer/uuid* pada *Primary Key*

Kesalahan fatal lain yang sering dilakukan dalam membuat *database* adalah menggunakan tipe data selain *integer/uuid* pada *primary Key*. Hal itu memang bisa dilakukan, tetapi itu bukanlah *best practice* atau bukan cara yang terbaik.

(c) Mengabaikan *Timezone*

Mengelola zona waktu pada *field DATE* dan *DATE TIME* dapat menjadi masalah serius dalam sistem. Sistem harus selalu menyajikan tanggal dan waktu yang tepat kepada *user*, terutama di zona waktu mereka sendiri. Misalnya, waktu kadaluawarsanya suatu penawaran khusus (fitur yang paling penting di setiap toko online yang ada) harus dipahami oleh semua *user* dengan cara yang sama. Jika kita hanya mengatakan "promosi berakhir pada tanggal 25 November", mereka akan menganggap promosinya akan berakhir pada tengah malam 25 November di zona waktu mereka sendiri. Berhati-hatilah, para *user* harus melihat tanggal promosi di zona waktu mereka sendiri. Dalam system yang memiliki *multi-timezone*, *field* tipe *DATE* harusnya itu tidak digunakan. *Field* Ini harus selalu menjadi tipe *TIMESTAMP*.

2. Versi interpreter dan *compiler*

Interpreter adalah perangkat lunak yang mampu mengeksekusi *code* program atau sebuah perintah yang lalu menterjemahkannya ke dalam bahasa mesin,

sehingga mesin melakukan instruksi yang diminta oleh *programmer* tersebut. sedangkan *Compiler* sendiri adalah program sistem yang digunakan sebagai alat bantu dalam pemrogaman.Perangkat lunak yang melakukan proses penterjemahan *code* (yang dibuat programmer) ke dalam bahasa mesin. Hasil dari terjemahan ini adalah bahasa mesin. Pada beberapa *compiler*, *output* berupa bahasa mesin dilaksanakan dengan sebuah proses *assembler* yang berbeda-beda. ketidak samaan versi dari interpreter dan *compiler* atau perbedaan versi yang menjadikan tidak *comaptible* dengan sebuah perintah atau *code* program yang menjadikan *code* program tersebut tidak bisa dieksekusi oleh bahasa mesin, cara untuk menanganinya adalah menyesuaikan versi iterpreter dan *compiler*agar textitcomaptible dengan *code* program.

Chapter 4

Contoh Error

Disini kami akan memberikan contoh dari beberapa error aplikasi

1. Error Syntax

   Program Bahasa C dibawah ini akan error pada baris ke-4 karena pada baris sebelumnya (baris ke-3) statement belum ditutup menggunakan titik koma (;)

   ```
   void main() {
   int a=2, b=0;
   printf("%i", a)
   printf("%i", b);
   }
   ```

2. Error Runtime

   contoh yang banyak terjadi error pada aplikasi adalah error runtime karena pembagian suatu bilangan dengan nol. Lihat contoh program bahasa c berikut ini Secara syntax tentu tidak terdapat error, namun jika dirunning, operasi pembagian pada baris ke-5 akan menyebabkan error “division by zero”.

   ```
   void main() {
   int a=2, b=0;
   printf("%i", a)
   printf("%i", b);
   printf("%i", a/b);
   }
   ```

3. Logical Error

   contoh error yang termasuk dalam jenis Logical Error adalah saat kita membuat sebuah program yang menghasilkan nilai B dari suatu lingkaran yang A diinput oleh user. Jika user menginputkan nilai 7, tentu program seharusnya akan menampilkan nilai 154. Namun jika ternyata program tersebut tidak menghasilkan hasil sesuai yang diharapkan, inilah yang disebut sebagai logical error.

```
void main() {
int A;
float B;
printf("Masukkan : ");
scanf("%i", &A);
luas = 2 * 3.14 * A * A;
printf("%f", B);
}
```

4. *Database*

   Contoh kesalahan dalam *Database* sebagai berikut:

   (a) Semua kesalahan internal (ORA-600)

   (b) blok kesalahan korupsi (ORA-1578)

   (c) kesalahan deadlock (ORA-60)

   (d) *Error message string*

   Pesan kesalahan biasanya berisi informasi diagnostik tentang penyebab kesalahan. Banyak pesan kesalahan memiliki variabel substitusi di mana informasi, seperti nama objek yang menghasilkan kesalahan, dimasukkan.

   (e) *Severity*

   Tingkat keparahan kesalahan mengindikasikan seberapa serius kesalahan tersebut. Kesalahan yang memiliki tingkat keparahan yang rendah, seperti 1 atau 2, adalah pesan informasi atau peringatan tingkat rendah. Kesalahan yang memiliki tingkat keparahan tinggi menunjukkan masalah yang harus diatasi secepat mungkin dan sesegera mungkin.

   (f) *Procedure name*

   Procedure name adalah sebuah nama prosedur yang tersimpan atau pemicu di mana kesalahan telah terjadi.

(g) *Line number*

Menunjukkan sebuah pernyataan yang berada dalam sebuah *batch*, prosedur tersimpan, pemicu, atau fungsi yang menghasilkan sebuah kesalahan.

5. Versi interpreter atau *compiler*

error dalam bahasa pemrograman disini kita akan menjalankan program "hello world" dan pada saat dijalankan terdapat peringatan sebagai berikut :

```
java.lang.UnsupportedClassVersionError: "hello world" :
 Unsupported major.minor version 51.0
    at java.lang.ClassLoader.defineClass1(Native Method)
    at java.lang.ClassLoader.defineClassCond(Unknown Source)
       ....................................
```

Untuk memperbaiki masalah yang ada diatas, kita harus mencoba menjalankan kode Java dengan versi Java JRE yang lebih baru atau menentukan parameter target ke *compiler* Java untuk memerintahkan *compiler* membuat kode program yang kompatibel dengan versi Java sebelumnya.

Chapter 5

Kegiatan Mingguan

5.1 Form Penilaian Mingguan

Dalam rangka penanganan *error* setiap minggunya diukur dengan penilaian. Setiap penilaian memiliki bobot dan karakteristik. Masing *error* dilaporkan dalam Issues di github, yang kemudian **semua proses** penyelesaiannya dilakukan dengan commit sesuai dengan nomor issues nya. Issues bukan hanya judul *error*nya saja, tetapi bisa juga judulnya fitur yang dikembangkan yang pastinya selama proses pengembangan pasti *error*. Karena tidak wajar proses pengembangan akan terbebas dari *error*. Tabel 5.1, adalah standar penilaian mingguan yang dilaporkan kepada pembimbimng nilainya.

No	Parameter	Bobot	Nilai
1	Setiap Commit memiliki pesan yang mudah dibaca dan dimengerti, berisi apa yang dilakukan dan ditambahkan dengan nomor *Issues*	5 poin per commit yang ada di dalam issues	
2	Terdapat *Error* dalam kurun waktu 1 minggu selama bimbingan yang sudah dibuatkan pada *Issues* beserta penjelasan *error* tersebut	**10** point untuk issues kode error yang baru dan terselesaikan,**5** point untuk issues kode error yang baru dan belum terselesaikan, **1** point untuk issues kode *error yang sama*	
3	Aplikasi berbasiskan kelas dan atau method dan atau fungsi dimana setiap kelas,method,fungsi hanya memiliki 1 algoritma kegiatan saja	10 poin,hanya berlaku 1 kali	

4	Memiliki Program utama yang terpisah dari file fungsi, *class*, dan *method*	5 poin ,hanya berlaku 1 kali	
5	Semua penggunaan nama variable di dalam program menggambarkan isi dari variabel	5 poin,hanya berlaku 1 kali	
6	Tutorial pengembangan aplikasi serta penanganan errornya dituliskan dalam laporan berformat latex beserta tabel nilainya dalam 1 chapter telah di compile dan bebas error latex	5 poin,hanya berlaku 1 kali	
Penilaian			
7	Nilai akhir	Akumulasi dari point-point diatas	

Table 5.1: Tabel Penilaian

5.2 Contoh Kegiatan Mingguan

Contoh commit yang standar bisa dilihat pada listing 5.1. Penulisan pesan commit jelas dan mudah dimengerti dan berhubungan dengan file kodok.py serta dikaitkan dengan issues nomor 2.

```
1 git add kodok.py
2 git commit -m "perubahan pada perintah print dengan menambahkan tanda
    kurung karena beda versi python yang sebelumnya dari versi 2.6 #2"
```

Listing 5.1: Contoh commit standar

Bibliography