Bab 2

Penyeleksian Kondisi

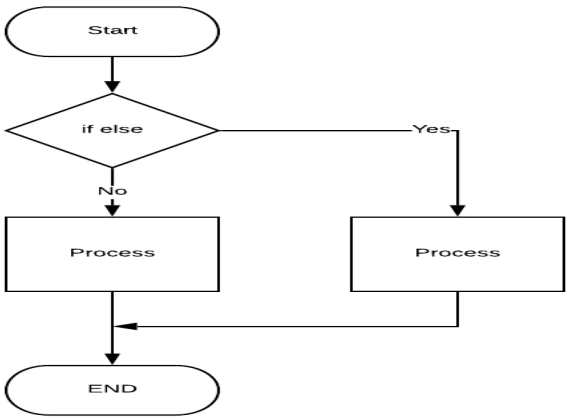
Tujuan

1. Memahami konsep pengambilan keputusan pemrograman komputer
2. Mampu melakukan *Control Flow* program menggunakan penyeleksian kondisi
3. Mampu menggunakan *Try Catch* untuk *Error Handling*

2.1 IF ELSE

If-Else Statement adalah *Control Flow* paling sederhana di dalam pemrograman, logika dasarnya adalah jika kondisi pada *if* terpenuhi, maka perintah di dalam *if* akan dijalankan, jika *if* tersebut memiliki *else*, maka perintah *else* akan dijalankan ketika kondisi *if* tidak terpenuhi. Bentuk lain dari *if-else* adalah *if-else if-else*, pada kasus ini, *else if* akan selalu di cek terlebih dahulu sebelum *else*. Berikut beberapa hal yang harus diketahui tentang *if-else statement*:

- *if* dapat dijalankan tanpa adanya *else if* atau pun *else*, tetapi tidak sebaliknya
- *else* harus terletak setelah semua *else if* dan semua *else if* harus terletak setelah *if*
- Hanya terdapat satu *if* dan *else* dalam satu blok *if-else if-else*
- Ketika salah satu kondisi *else if* terpenuhi, maka kondisi *else if* yang lain tidak akan dijalankan



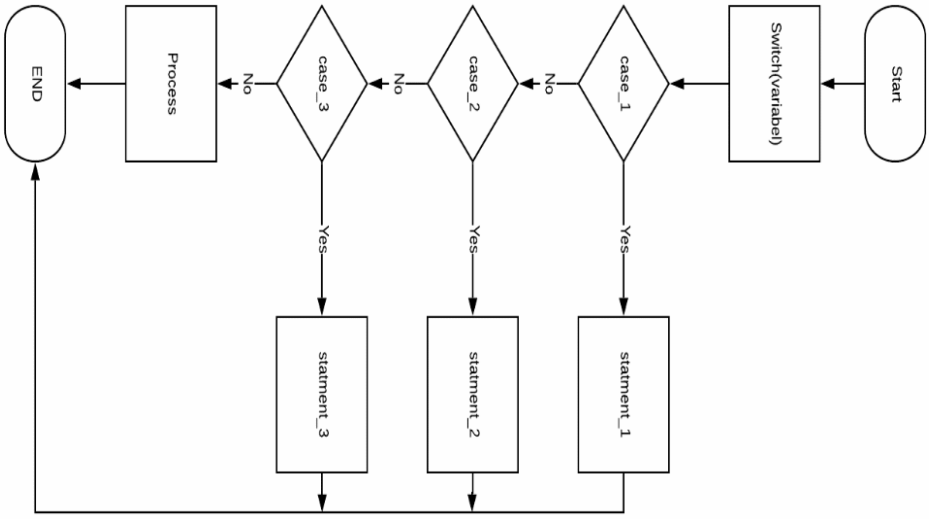
Berikut contoh penggunaan *if-else if-else statement*

```
public static void main (String[] args) {
    int score = 80;
    char grade;
    if (score >= 80) {
         grade = ‘A’;
    } else if (score >=75) {
         grade = ‘B’;
    } else if (score >= 65) {
         grade = ‘C’;
    } else {
         grade = ‘E’;
    }
    System.out.println(“Your grade is “ + grade);
}
```

Drawing 10: If-Else If-Else untuk menentukan nilai grade berdasarkan score

2.2 SWITCH CASE

Switch Case Statement adalah bentuk penyeleksian kondisi dengan satu variabel uji. Pada dasarnya fungsi *switch case* mirip dengan *if else*, perbedaannya terletak pada cara menguji variabel uji tersebut, di dalam *switch case*, *switch* berfungsi untuk menyatakan variabel yang akan diuji, kemudian *case* yang tersedia akan menjadi pembanding terhadap variabel dalam *switch* tadi, jika salah satu *case* terpenuhi, pake perintah dalam *case* tersebut akan dijalankan hingga terdapat *break* atau *return statement*.



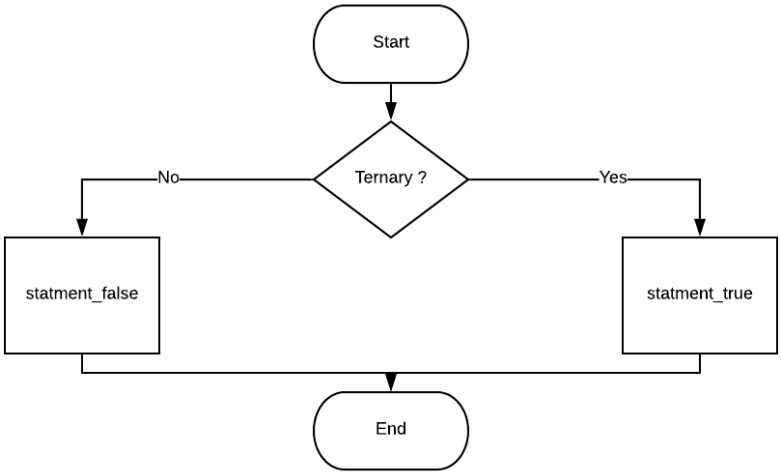
```
public static void main (String[] args) {
    int month = 9;
    switch (month) {
          case 12:
               System.out.println(“December”);
               break;
          case 9:
               System.out.println(“September”);
               break;
          default:
               break;
    }
}
```

Drawing 11: Switch case untuk mencetak nama bulan berdasarkan month

Jika semua *case* tidak terpenuhi, maka perintah di dalam *default* akan dijalankan, dalam hal ini default berfungsi seperti *else* pada *If Else Statement.*

2.3 TERNARY OPERATOR

Untuk mempersingkat penulisan *If Else Statement*, Java menyediakan *Ternary Operator (? :).* Pada *ternary operator*, variabel uji terdapat di sebelah kiri tanda tanya (?), jika variabel uji tersebut bernilai *true* maka nilai sebelah kiri titik dua (:) yang akan digunakan, jika *false*, maka nilai sebelah kanan titik dua yang akan digunakan. Operasi *ternary* tidak bisa berdiri sendiri, operasi ternary harus terletak di dalam variabel atau perintah lainnya.



```
public static void main (String[] args) {
    int score = 75;
    char grade = score >= 80 ? ‘A’ : ‘B’;
    System.out.println(“Your Grade is “ + grade);
}
```

Drawing 12: Menentukan nilai grade menggunakan ternary operator

2.4 TRY CATCH

Dalam alur sebuah program, terdapat sebuah kejadian yang disebut *exception* yang mengacaukan alur dari program. *Exception* dapat mengakibatkan alur program terhambat atau dalam kasus terburuknya terjadi terminasi / penghentian secara paksa.

Try Catch Statement berfungsi khusus untuk mengatasi *exception, Try Catch* Statement terdiri dari blok *try*, *catch*, dan *finally*. Pada dasarnya terdapat 3 jenis *Exception* dalam java yaitu :

a) *Checked Exception*, atau *Compile Time Exception* yang jika tidak diatasi, maka program tidak akan bisa di-*compile*, *exception* yang termasuk di dalamnya adalah *NoSuchFieldException* yang terjadi karena terdapat pemanggilan variabel yang tidak ada.

b) *Unchecked Exception*, atau *Runtime Exception* adalah kesalahan yang terjadi selama program berjalan dan umumnya terjadi karena kesalahan *logic* dalam penulisan kode, beberapa *exception* yang termasuk di dalamnya adalah:

Exception	Penyebab
ArithmeticException	Melakukan kesalahan aritmatika, seperti pembagian dengan nol.
InputMismatchException	Input data yang tidak sesuai
NullPointerException	Kesalahan penggunaan dari referensi null

Table 7: Beberapa Class Unchecked Exception

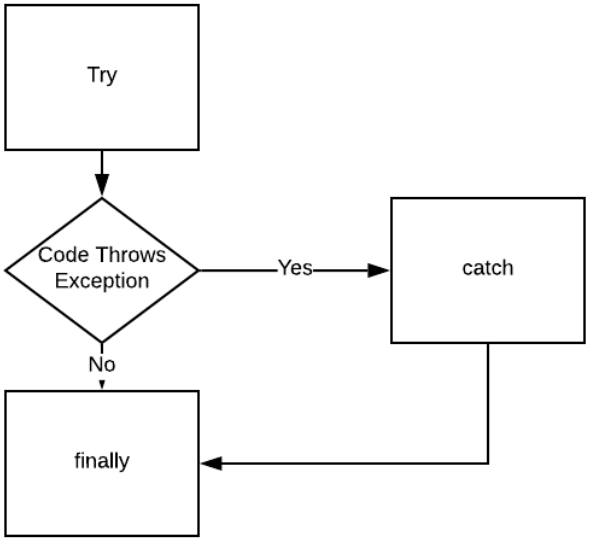
c) *Error*, Berbeda dengan *exception* sebelumnya, error merupakan kesalahan yang tidak dapat ditangani dengan *try catch*, seperti *StackOverflow* yang terjadi ketika JVM kehabisan memori.

Terdapat beberapa cara untuk mengatasi *Exception* selain dengan blok *Try Catch*, seperti melempar *exception* pada *method* dengan keyword *throws* atau membuat objek *throwable* dengan keyword *throw*, namun pada bab ini yang akan

dipelajari adalah mengatasi *Exception* dengan *Try Catch* dan membuat objek *throwable* dengan keyword *throw*.

2.4.1 Menangani *Exception* dengan *Try Catch Statement*

Untuk mengatasi *exception* dengan cara ini, blok perintah yang mungkin menghasilkan *exception* disimpan di dalam blok *try*, lalu dalam blok *catch* ditentukan *exception* apa saja yang akan ditangani dan perintah penanganannya. Satu blok *Try* dapat memuat lebih dari satu blok *catch* tapi tidak sebaliknya.




```
public static void main (String[] args) {
    Scanner in = new Scanner(System.in);
    try {
         int i = in.nextInt();
         int result = 10 / 0;
    } catch (ArithmeticException ae) {
         System.out.println(ae.getMessage());
    } catch (InputMismatchException ime) {
         System.out.println(ime.toString);
    } finally {
         in.close();
         System.out.println(“end”);
    }
}
```

Drawing 13: Membuat Exception Ketika Nilai i Negatif

Program di atas adalah *try catch* dengan 2 blok *catch*, blok *catch* yang dijalankan bergantung pada *exception* apa yang terjadi, sedangkan blok *finally* adalah blok yang akan selalu dijalankan sekalipun tidak terjadi *exception*, sama dengan *try*, blok *finally* hanya boleh terdapat satu dalam satu blok *try catch*.

2.4.2 Menangani *Exception* dengan objek *Throwable*

Penanganan *exception* dengan cara ini lebih ditujukan jika kita ingin mendefinisikan sendiri *exception* sesuai kebutuhan tanpa menunggu terjadinya *exception*. Berikut cara menangani *exception* dengan membuat objek *Throwable*.

```
public static void main (String[] args) {
    Scanner in = new Scanner(System.in);
    try {
         int I = in.nextInt();
         if (I < 0) {
              throw new NumberFormatException();
         }
    } catch (NumberFormatException nfe) {
         System.out.println(nfe.getMessage());
    } finally {
         in.close();
    }
}
```

Drawing 14: Exception Terjadi Ketika Nilai I Negatif

Program di atas akan membuat *exception* jika nilai inputan negatif, dengan cara ini kita bisa membuat pesan error sendiri ketika membuat objek *throwable* dengan menaruh pesan yang diinginkan dalam *constructor throwable*.