Pertemuan III

STRUKTUR KONTROL DAN ARRAY

3.1. Struktur Kontrol

3.1.1. Percabangan

Percabangan adalah pernyataan dari Java yang mengijinkan user untuk memilih dan mengeksekusi blok kode spesifik sesuai kondisi yang telah ditentukan dan mengabaikan blok kode yang lain.

Pernyataan-pernyataan yang dapat digunakan untuk struktur kontrol percabangan antara lain :

- *if*
- *if - else*
- *switch*

3.1.1.1. Percabangan dengan *if*

Pernyataan *if* digunakan untuk menentukan sebuah pernyataan (atau blok kode) akan dieksekusi jika dan hanya jika persyaratan bernilai benar (true).

Bentuk dari pernyataan *if* adalah :

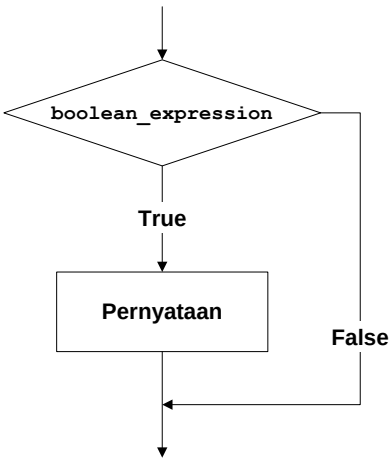
```
if(boolean_expression) statement;
```

atau :

```
if (boolean_expression){
            statement1;
            statement2;
            . . .
}
```

dimana, *boolean_expression* adalah sebuah penyataan logika yang bernilai benar (*true*) atau salah (*false*), atau variabel bertipe *boolean*.

Pernyataan *if* dapat digambarkan dalam *flowchart* sebagai berikut :



Gambar 3.1. *Flowchart* pernyataan *if*

Penggunaan pernyataan *if* dalam program dapat dilihat pada contoh program dibawah ini:

Pernyataanif1.java

```
public class Pernyataanif1{
    public static void main(String args[]){
        Byte grade = 95;
        if ( grade > 80 ) System.out.println("Selamat Anda Lulus!");
    }
}
```

atau :

Pernyataanif2.java

```
public class Pernyataanif2{
    public static void main(String args[]){
        Byte grade = 95;
        if ( grade > 80 ){
            System.out.println("Selamat Anda Lulus");
            System.out.println("Dengan nilai "+grade+"!");
        }
    }
}
```

3.1.1.2. Percabangan dengan *if - else*

Pernyataan *if – else* digunakan untuk menentukan dua kondisi alur program, jika persyaratan bernilai benar (true) maka sebuah pernyataan (atau blok kode) akan dieksekusi. Sedangkan jika persyaratan bernilai salah (false) maka sebuah pernyataan (atau blok kode) lain yang akan dieksekusi.

Bentuk dari pernyataan *if - else* adalah :

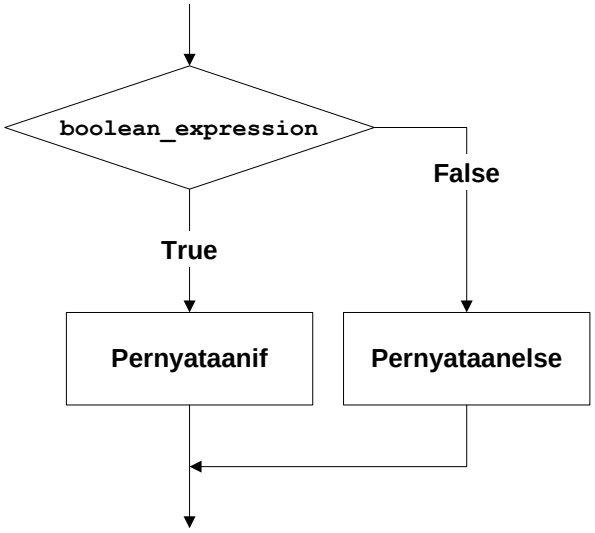
```
if (boolean_expression)
    statementif;
else
    statementelse;
```

atau :

```
if (boolean_expression){
    statementif1;
    statementif2;
    . . .
}
else){
    statementelse1;
    statementelse2;
    . . .
}
```

dimana, *boolean_expression* adalah sebuah penyataan logika yang bernilai benar (*true*) atau salah (*false*), atau variabel bertipe *boolean*.

Pernyataan *if - else* dapat digambarkan dalam *flowchart* sebagai berikut :



Gambar 3.2. *Flowchart* pernyataan *if-else*

Penggunaan pernyataan *if - else* dalam program dapat dilihat pada contoh program dibawah ini:

Pernyataanifelse1.java

```
 import java.io.*;

 public class Pernyataanifelse1{
    public static void main(String args[]){
       BufferedReader dataIn = new BufferedReader(new
InputStreamReader(System.in));
       String gradestring = "";
       Byte grade = 0;

       System.out.print("Ketik nilai Anda : ");
       try{
          gradestring = dataIn.readLine();
       }
       catch( IOException e ){
          System.out.println("Ada kesalahan !");
       }

       grade = new Byte (gradestring);

       if ( grade >= 80 ) System.out.println("Selamat Anda Lulus!");
       else System.out.println("Maaf Anda Belum Lulus!");
    }
 }
```

Atau

Pernyataanifelse2.java

```
 import java.io.*;

 public class Pernyataanifelse2{
    public static void main(String args[]){
       BufferedReader dataIn = new BufferedReader(new
InputStreamReader(System.in));
```

```
      String gradestring = "";
      Byte grade = 0;

      System.out.print("Ketik nilai Anda : ");
      try{
         gradestring = dataIn.readLine();
      }
      catch( IOException e ){
         System.out.println("Ada kesalahan !");
      }

      grade = new Byte (gradestring);

      if ( grade >= 80 ) {
         System.out.println("Selamat Anda Lulus!");
         System.out.println("Karena nilai Anda "+grade+"!");
      }
      else {
         System.out.println("Maaf Anda Belum Lulus!");
         System.out.println("Karena nilai Anda "+grade+"!");
      }
   }
}
```

Karena input yang kita ketik berupa data bertipe string, maka harus dikonversi ke bentuk byte menggunakan pernyataan :

```
grade = new Byte (gradestring);
```

Pernyataan pada bagian kondisi *else* dari blok *if-else* dapat menjadi struktur *if-else* yang lain. Kondisi struktur seperti ini mengijinkan kita untuk membuat seleksi dengan persyaratan yang lebih kompleks.

Bentuk pernyataan *if-else if* :

```
if ( boolean_expression1 )
    statement1;
else if ( boolean_expression2 )
    statement2;
else
    statement3;
```

Kita dapat memiliki banyak blok *else-if* sesudah pernyataan *if*. Blok *else* bersifat opsional dan dapat dihilangkan. Pada contoh yang ditampilkan di atas, jika *boolean_expression1* bernilai benar (*true*), maka program akan mengeksekusi *statement1* dan melewati pernyataan yang lain. Jika *boolean_expression1* bernilai salah (*false*) dan *boolean_expression2* bernilai benar (*true*), maka program akan mengeksekusi *statement2* dan mengabaikan *statement* yang lain.

Penggunaan pernyataan *if* - *else* dalam program dapat dilihat pada contoh program dibawah ini :

Pernyataanifelseif.java

```
import java.io.*;

public class Pernyataanifelseif{
   public static void main(String args[]){
```

```
      BufferedReader dataIn = new BufferedReader(new
InputStreamReader(System.in));
      String gradestring = "";
      byte grade = 0;

      System.out.print("Ketik nilai Anda : ");
      try{
        gradestring = dataIn.readLine();
      }
      catch( IOException e ){
        System.out.println("Ada kesalahan !");
      }

      //Konversi nilai string ke Byte
      grade = new Byte (gradestring);

      if ( grade == 100 ) {
        System.out.println("Selamat Anda Lulus dengan Kategori
Sempurna!");
        System.out.println("Karena nilai Anda "+grade+"!");
      }
      else if ( grade >= 95 ) {
        System.out.println("Selamat Anda Lulus dengan Kategori Sangat
Memuaskan!");
        System.out.println("Karena nilai Anda "+grade+"!");
      }
      else if ( grade >= 90 ) {
        System.out.println("Selamat Anda Lulus dengan Kategori
Memuaskan!");
        System.out.println("Karena nilai Anda "+grade+"!");
      }
      else if ( grade >= 80 ) {
        System.out.println("Selamat Anda Lulus dengan Kategori Baik!");
        System.out.println("Karena nilai Anda "+grade+"!");
      }
      else {
        System.out.println("Maaf Anda Belum Lulus!");
        System.out.println("Karena nilai Anda "+grade+"!");
      }
    }
  }
```

3.1.1.3. Percabangan dengan *switch*

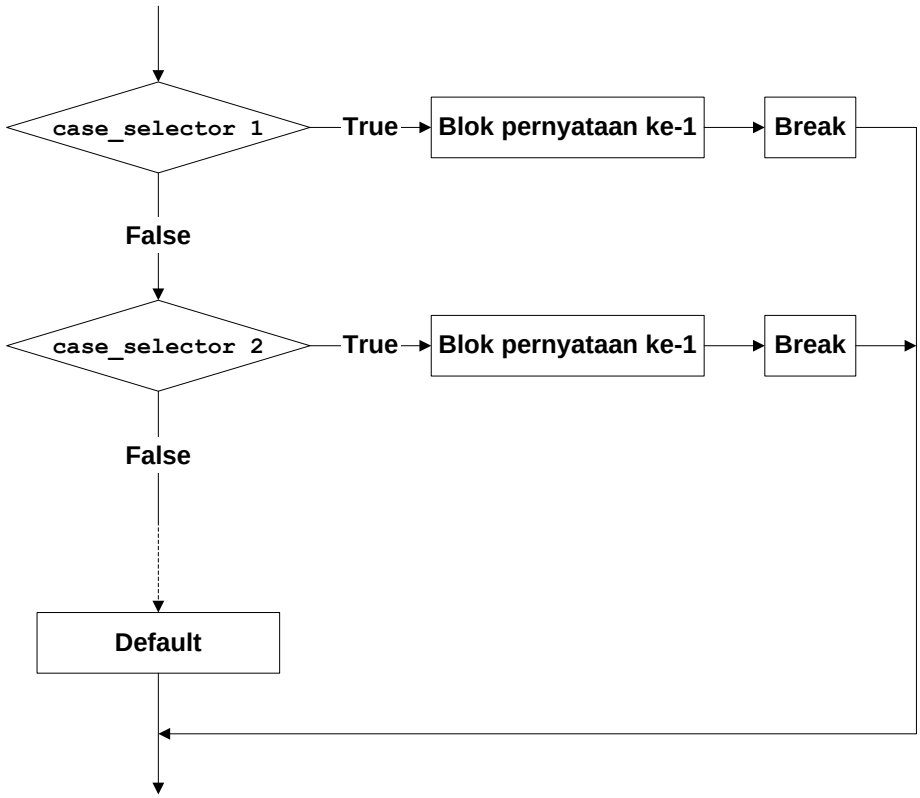
Cara lain untuk membuat cabang adalah dengan menggunakan kata kunci *switch*. *Switch* digunakan untuk menangani percabangan yang memiliki banyak kondisi.

Bentuk statement *switch* adalah :

```
switch (switch_expression){
  case case_selector1:
    statement1; //
    statement2; //blok pernyataan ke-1
    . . . //
    break;
  case case_selector2:
    statement1; //
    statement2; //blok pernyataan ke-2
    . . . //
    break;
  . . .
```

```
   default:
     statement1; //
     statement2; //block n
     . . . //
     break;
}
```

Pernyataan *switch* dapat digambarkan dalam *flowchart* sebagai berikut :



Gambar 3.3. *Flowchart* pernyataan *switch*

switch_expression adalah ekspresi yang menghasilkan nilai *integral* (untuk java SE 7 sudah dapat menggunakan nilai *String*). *case_selector1*, *case_selector2* dan seterusnya adalah konstanta unik dari nilai *switch_expression*.

Ketika pernyataan *switch* ditemukan pada potongan kode program, java pertama kali akan memeriksa *switch_expression*, dan menuju ke *case* yang akan menyamakan nilai yang dimiliki oleh *switch_expression* dengan nilai *case_selector*. Jika ditemukan program akan mengeksekusi pernyataan dari kode setelah *case* yang ditemukan sampai menemui pernyataan *break*, selanjutnya akan mengabaikan pernyataan yang lainnya hingga akhir dari struktur pernyataan *switch*.

Jika tidak ditemui *case* yang cocok, maka program akan mengeksekusi blok *default* (kalau ada), karena bagian blok *default* bersifat opsional. Sebuah pernyataan *switch* bisa jadi tidak memiliki blok *default*.

Tidak seperti pada pernyataan ***if***, beberapa pernyataan pada struktur pernyataan *switch* akan dieksekusi tanpa memerlukan tanda kurung kurawal (**{}**).

Ketika sebuah *case* pada pernyataan *switch* menemui kecocokan, semua pernyataan setelah case tersebut akan dieksekusi. Untuk menghindari program mengeksekusi pernyataan pada *case* berikutnya, kita menggunakan pernyataan *break* sebagai pernyataan akhir pada setiap blok *case*.

Program if - else bertingkat sebelumnya dapat kita buat menggunakan pernyataan switch sebagai berikut :

Pernyataanswitch.java

```
 import java.io.*;

 public class Pernyataanswitch{
    public static void main(String args[]){
       BufferedReader dataIn = new BufferedReader(new
InputStreamReader(System.in));
       String angkastring = "";
       byte angka = 0;

       System.out.print("Ketik angka 0..9 : ");
       try{
          angkastring = dataIn.readLine();
       }
       catch( IOException e ){
          System.out.println("Ada kesalahan !");
       }

       //Konversi nilai string ke Byte
       angka = new Byte (angkastring);

       switch (angka){
          case 0: System.out.println("Angka yang diketik adalah nol");
                  break;
          case 1: System.out.println("Angka yang diketik adalah satu");
                  break;
          case 2: System.out.println("Angka yang diketik adalah dua");
                  break;
          case 3: System.out.println("Angka yang diketik adalah tiga");
                  break;
          case 4: System.out.println("Angka yang diketik adalah empat");
                  break;
          case 5: System.out.println("Angka yang diketik adalah lima");
                  break;
          case 6: System.out.println("Angka yang diketik adalah enam");
                  break;
          case 7: System.out.println("Angka yang diketik adalah tujuh");
                  break;
          case 8: System.out.println("Angka yang diketik adalah delapan");
                  break;
          case 9: System.out.println("Angka yang diketik adalah
sembilan");
                  break;
          default: System.out.println("Angka yang diketik tidak sesuai");

       }
    }
 }
```

3.1.2. Perulangan

Struktur kontrol perulangan adalah berupa pernyataan dari Java yang menyebabkan eksekusi terhadap blok kode program dilakukan berulang-ulang sesuai dengan kondisi tertentu. Ada tiga macam struktur kontrol pengulangan, yaitu *for*, *while*, dan *do - while*.

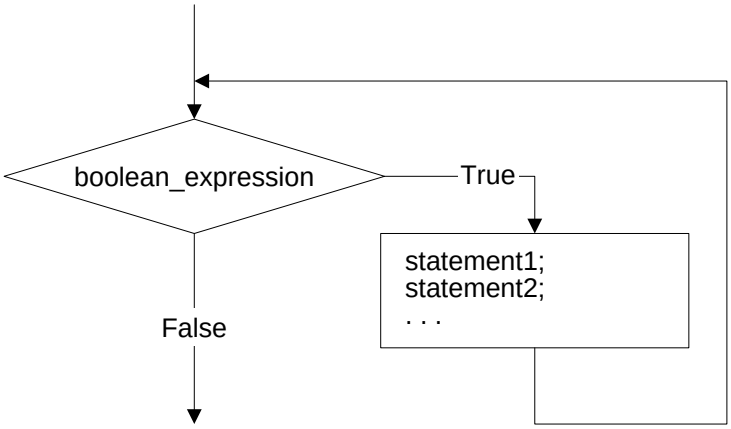
3.1.2.1. Menggunakan *while*

Pernyataan pengulangan *while* adalah pernyataan atau blok pernyataan yang diulang-ulang sampai mencapai kondisi yang tidak cocok.

Bentuk pernyataan while :

```
while (boolean_expression){
    statement1;
    statement2;
    . . .
}
```

Alurnya dapat dilihat pada flowchart di bawah ini :



Gambar 3.4. *Flowchart* pernyataan *while*

Pernyataan di dalam pengulangan *while* akan dieksekusi berulang-ulang selama kondisi *boolean_expression* bernilai benar (*true*).

Contoh, pada kode dibawah ini :

```
int i = 4;
while  ( i > 0 ){
    System.out.print(i);
    i--;
}
```

Contoh diatas akan mencetak angka 4321 pada layar. Perlu dicatat jika bagian i--; dihilangkan, akan menghasilkan pengulangan yang terus menerus (*infinite loop*). Sehingga, ketika menggunakan *while loop* atau bentuk pengulangan yang lain, pastikan untuk memberikan pernyataan yang membuat pengulangan berhenti pada suatu kondisi.

3.1.2.2. Menggunakan *do - while*

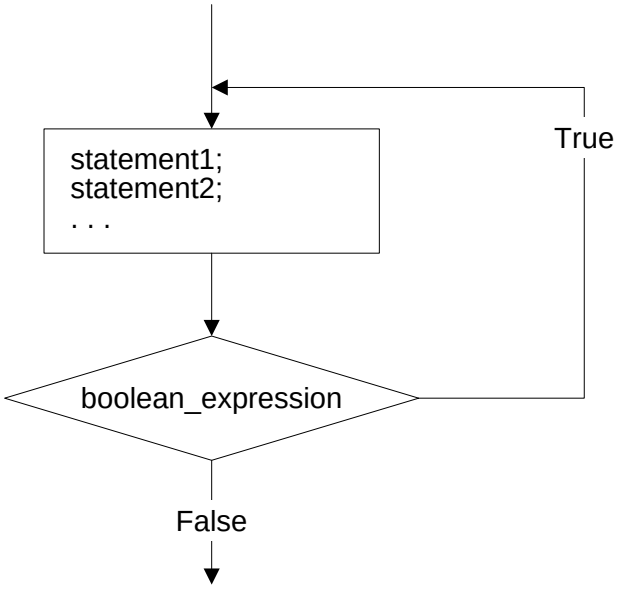
Pengulangan *do-while* mirip dengan pengulangan *while*. Pernyataan di dalam pengulangan *do-while* akan dieksekusi beberapa kali selama kondisi bernilai benar(*true*).

Perbedaan antara *while* dan *do-while* adalah dimana pernyataan di dalam pengulangan *do-while* akan dieksekusi sedikitnya satu kali, sedangkan pada perngulangan *while* ada kemungkinan tidak dieksekusi.

Bentuk pernyataan *do-while* :

```
do{
     statement1;
     statement2;
     . . .
}while( boolean_expression );
```

Alurnya dapat dilihat pada flowchart di bawah ini :



Gambar 3.5. *Flowchart* pernyataan *do - while*

Pernyataan di dalam *do-while loop* akan dieksekusi pertama kali, dan akan dievaluasi kondisi dari *boolean_expression*. Jika nilai pada *boolean_expression* tersebut bernilai *true*, pernyataan di dalam *do-while loop* akan dieksekusi lagi.

Berikut ini contoh pengulangan *do-while* :

```
int x = 0;
do
{
    System.out.print(x);
    x++;
}while (x<10);
```

Contoh diatas akan memberikan output 0123456789 pada layar.

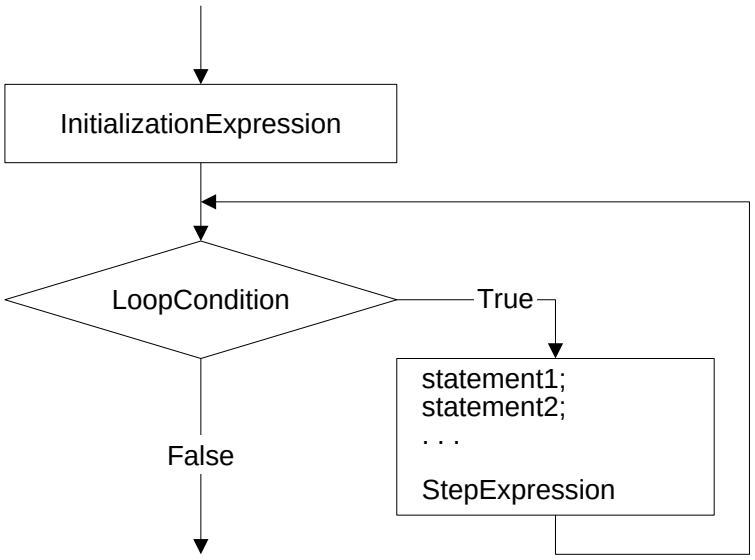
3.1.2.3. Menggunakan *for*

Pernyataan pengulangan *for* memiliki kondisi hampir mirip seperti struktur pengulangan sebelumnya yaitu melakukan pengulangan untuk mengeksekusi kode yang sama selama kondisi/jumlah tertentu.

Bentuk dari perulangan for :

```
for (InitializationExpression; LoopCondition; StepExpression){
    statement1;
    statement2;
    . . .
}
```

Alurnya dapat dilihat pada flowchart di bawah ini :



Gambar 3.6. *Flowchart* pernyataan *for*

dimana, *InitializationExpression* adalah inisialisasi dari variabel loop. *LoopCondition digunakan untuk* membandingkan variabel pengulangan pada nilai batas tertentu. *StepExpression* untuk melakukan update pada variabel loop.

Berikut ini adalah contoh dari for loop :

```
int i;
for( i = 0; i < 10; i++ ){
    System.out.print(i);
}
```

Pada contoh ini, pernyataan i=0 merupakan inisialisasi dari variabel. Selanjutnya, kondisi i<10 diperiksa. Jika kondisi bernilai true, pernyataan di dalam pengulangan *for* dieksekusi. Kemudian, ekspresi i++ dieksekusi, lalu akan kembali pada bagian pemeriksaan terhadap kondisi i<10 lagi. Kondisi ini akan dilakukan berulang-ulang sampai mencapai nilai yang salah (false).

3.2. Array

3.2.1. Pengenalan Array

Pada pertemuan sebelumnya kita telah membahas cara mendeklarasikan berbagai macam variabel dengan menggunakan berbagai tipe data. Dalam mendeklarasikan variabel, kita sering menggunakan sebuah tipe data beserta nama variabel atau *identifier* yang unik. Jika kita ingin menggunakan variabel tersebut, kita akan memanggil dengan nama variabel atau *identifier*-nya.

Sebagai contoh, kita memiliki tiga variabel dengan tipe data int yang memiliki *identifier* berbeda untuk tiap variabel.

```
int Angka1;
int Angka2;
int Angka3;

Angka1 = 1;
Angka2 = 2;
Angka3 = 3;
```

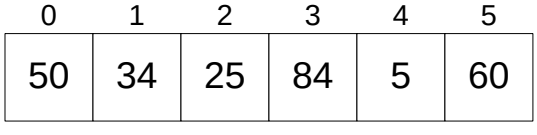
Seperti yang dapat kita lihat pada contoh diatas, kode tersebut akan sia-sia karena harus menginisialisasi dan menggunakan setiap variabel padahal sebenarnya variabel-variabel tersebut digunakan untuk tujuan yang sama. Apalagi jika data dalam variabel tersebut merupakan data yang berurutan, maka kita harus mengakses satu persatu nama variabelnya.

Pada bahasa pemrograman Java maupun di bahasa pemrograman yang lain, terdapat sebuah kemampuan untuk menggunakan satu variabel yang dapat menyimpan beberapa data dan memanipulasinya dengan lebih efektif. Tipe variabel inilah yang disebut sebagai array.

Array adalah suatu tipe yang dibentuk dari tipe data primitif untuk menyimpan sejumlah item yang bertipe sama. Array merupakan konsep yang penting dalam pemrograman, karena array memungkinkan untuk menyimpan data maupun referensi obyek dalam jumlah banyak dan terindeks. Array menggunakan indeks integer untuk menentukan urutan elemen-elemennya, dimana elemen pertamanya dimulai dari indeks 0, elemen kedua memiliki indeks 1, dan seterusnya.

Sebuah array akan menyimpan beberapa item data yang memiliki tipe data sama didalam sebuah blok memori yang berdekatan yang kemudian dibagai menjadi beberapa ruang. Array adalah sebuah variabel/sebuah lokasi tertentu yang memiliki satu nama sebagai *identifier*, namun *identifier* ini dapat menyimpan lebih dari sebuah nilai.

Penyimpanan data dalam array dapat digambarkan seperti pada gambar 3.7 dibawah ini :



Gambar 3.7. Blok penyimpanan dalam array bertipe integer

3.2.2. Mendeklarasikan Array

Array harus dideklarasikan seperti layaknya sebuah variabel. Pada saat mendeklarasikan array, kita harus membuat sebuah daftar dari tipe data, yang diikuti oleh sepasang tanda kurung siku "[]", lalu diikuti oleh nama *identifier*-nya. Sebagai contoh :

```
int[] ages;
```

atau kita dapat menempatkan sepasang tanda kurung siku [] sesudah nama *identifier*. Sebagai contoh :

```
int ages[];
```

Setelah mendeklarasikan array, kita harus membuat array dan menentukan berapa panjangnya dengan sebuah konstruktor. Proses ini di Java disebut sebagai *instantiation* (istilah dalam Java yang berarti membuat). Untuk meng-*instantiate* sebuah obyek, kita membutuhkan sebuah konstruktor *new*. Sebagai catatan bahwa ukuran dari array tidak dapat diubah setelah anda menginisialisasinya. Sebagai contoh :

```
//deklarasi
int ages[];
//instantiate obyek
ages = new int[100];
```

atau bisa juga ditulis dengan :

```
//deklarasi dan instantiate obyek
int ages[] = new int[100];
```

Pada contoh diatas, pendeklarasian tersebut akan memberitahukan kepada compiler Java, bahwa identifier *ages* akan digunakan sebagai nama array yang berisi data bertipe *integer*, dan dilanjutkan dengan membuat atau meng-*instantiate* sebuah array baru yang terdiri dari 100 elemen.

Selain menggunakan sebuah pernyataan *new* untuk meng-instantiate array, Anda juga dapat mendeklarasikan, membangun, kemudian memberikan sebuah nilai pada array sekaligus dalam sebuah pernyataan. Sebagai contoh,

```
//Membuat sebuah array yang terdiri dari penginisialisasian
//4variabel double bagi value {100,90,80,75}
double []grades = {100, 90, 80, 75};
```

3.2.3. Mengakses Elemen Array

Untuk mengakses sebuah elemen dalam array, atau mengakses sebagian dari array, kita harus menggunakan sebuah angka atau yang disebut sebagai indeks atau *subscript*.

Pada saat memasukkan nilai ke dalam array, sebuah nomor indeks atau *subscript* telah diberikan kepada tiap anggota array, sehingga program dan programmer dapat mengakses setiap nilai pada array apabila dibutuhkan. Nilai indeks selalu dalam tipe integer, dimulai dari angka nol dan dilanjutkan ke angka berikutnya sampai akhir array. Sebagai catatan bahwa indeks di dalam array dimulai dari 0 sampai dengan ukuran array dikurangi 1.

Sebagai contoh, pada array yang kita deklarasikan tadi, kita memberi nilai atau membaca nilai array sebagai berikut :

```
 //memberikan nilai 10 kepada elemen array pertama
ages[0] = 10;
//memberikan nilai 30 kepada elemen array indeks ke-90
ages[90] = 10;
//mencetak elemen array yang terakhir
System.out.print(ages[99]);
//mencetak elemen array indeks ke-90
System.out.print(ages[90]);
```

Perlu diperhatikan bahwa sekali array dideklarasikan dan dikonstruksi, nilai yang disimpan dalam setiap anggota array akan diinisialisasi sebagai nol. Oleh karena itu, apabila kita menggunakan tipe data seperti String, array tidak akan diinisalisasi menjadi string kosong “”. Untuk itu Anda tetap harus membuat String array secara eksplisit.

Berikut ini adalah contoh kode untuk mencetak seluruh elemen didalam array. Dalam contoh ini digunakanlah pernyataan *for loop*, sehingga kode kita menjadi lebih pendek.

```
public class ArraySample{
   public static void main( String[] args ){
      int[] ages = new int[100];
      for( int i=0; i<100; i++ ){
         System.out.print( ages[i] );
      }
   }
}
```

3.2.4. Panjang Array

Untuk mengetahui berapa banyak elemen didalam sebuah array, kita dapat menggunakan atribut *length* dari array. Atribut ini akan mengembalikan ukuran dari array itu sendiri. Sebagai contoh :

```
arrayName.length
```

Pada contoh sebelumnya, kita dapat menuliskannya kembali seperti berikut ini :

```
public class ArraySample{
   public static void main( String[] args ){
      int[] ages = new int[100];
      for( int i=0; i<ages.length; i++ ){
         System.out.print( ages[i] );
      }
   }
}
```

3.2.5. Array Multidimensi

Array multidimensi diimplementasikan sebagai array yang terletak di dalam array. Array multidimensi dideklarasikan dengan menambahkan jumlah tanda kurung setelah nama array.

Sebagai contoh array multidimensi dapat dilihat pada kode program dibawah ini :

```
 // Elemen 512 x 128 dari integer array
 int[][] twoD = new int[512][128];
 // karakter array 8 x 16 x 24
 char[][][] threeD = new char[8][16][24];
 // String array 4 baris x 2 kolom
 String[][] dogs = {{"terry","brown"},{"Kristin","white"},{"toby",
"gray"},{"fido","black"}};
```

Untuk mengakses sebuah elemen didalam array multidimensi, sama saja dengan mengakses array satu dimensi. Misalnya saja, untuk mengakses elemen pertama dari baris pertama didalam array dogs, kita akan menulis :

```
 System.out.print( dogs[0][0] );
```

Kode diatas akan mencetak String “terry” di layar.

3.2.6. Latihan

Modifikasi program dibawah ini agar dapat menampilkan nilai maksimal, nilai minimal dan mengurutkan data dalam array.

```
 import java.io.BufferedReader;
 import java.io.InputStreamReader;
 import java.io.IOException;

 public class LatihanArray{
   public static void main( String[] args ){
     BufferedReader dataIn = new BufferedReader(new
InputStreamReader(System.in));
     String BacaInput = "";
     Byte JmlData = 0;
     Byte DataArray[];

     System.out.print("Jumlah data :");

     try{
       BacaInput = dataIn.readLine();
     }
     catch( IOException e ){
       System.out.println("Ada kesalahan !");
     }

     JmlData = new Byte (BacaInput);
     DataArray = new Byte[JmlData];

     //Membaca data dari keyboard
     System.out.println();
     for (Byte i = 0;i<JmlData;i++){
       System.out.print("DataArray["+i+"] = ");

       try{
         BacaInput = dataIn.readLine();
       }
       catch( IOException e ){
         System.out.println("Ada kesalahan !");
       }
       DataArray[i] = new Byte (BacaInput);
     }
```

```
      //Menampilkan data dari array
      System.out.println();
      for (Byte i = 0;i<JmlData;i++){
        System.out.println("DataArray["+i+"] = "+DataArray[i]);
      }
   }
}
```

Referensi:

1. Hariyanto, Bambang, (2007), *Esensi-esensiBahasaPemrograman Java*, Edisi 2, Informatika Bandung, November 2007.
2. Tim Pengembang JENI, JENI 1-6, Depdiknas, 2007
3. Utomo, EkoPriyo, (2009), *PanduanMudahMengenalBahasa Java*, YramaWidya, Juni 2009.