PERTEMUAN II

ARRAY, PERCABANGAN, DAN PERULANGAN

TUJUAN PRAKTIKUM

1. Praktikan mengetahui array, percabangan, dan perulangan pada Java.
2. Praktikan mengetahui bentuk umum dari array, percabangan, dan perulangan dalam Java.
3. Praktikan dapat membuat program sederhana dengan menggunakan bahasa pemrograman Java, untuk array, percabangan, dan perulangan.

2.1 Array

2.1.1 Deklarasi Dan Penciptaan *Array*

Seperti yang telah dijelaskan secara singkat mengenai *array* pada pertemuan sebelumnya adalah kelompok variabel dengan tipe sejenis dan dinyatakan dengan nama yang sama. Dengan kata lain, *array* merupakan tipe khusus yang menyatukan sekelompok variabel dengan tipe yang sama. *Array* di Java dideklarasikan dengan kurung siku: […].

Sintaks umum deklarasi *array*:

type var-name[];

Atau

type[] var-name;

type berperan dalam mendeklarasikan tipe basis dari *array*. Tipe basis menentukan tipe data bagi masing-masing elemen yang membentuk *array*. Dengan demikian, tipe basis untuk *array* menentukan tipe data yang dimuat oleh *array*. Berikut contoh-contoh deklarasi *array* di Java:

- *int numbers[];*
- *char[] letters;*
- *long grid[][];*

Pada Java tidak perlu menspesifikasikan ukuran *array* kosong saat mendeklarasikan *array*. Kita harus selalu menyatakan ukuran *array* secara eksplisit saat melakukan

operasi penciptaan menggunakan operator **new()** atau dengan mendaftarkan itemitem untuk *array* pada saat penciptaan.

Berikut contoh pendeklarasian *array* dengan menspesifikasikan ukuran spesifiknya menggunakan operator **new()** dan dengan memberikan daftar item yang termasuk di dalam *array*:

char alphabet[] = new() char [26];

int primes = {7, 11, 13};

2.1.2 *Array* 1 Dimensi

Array 1 dimensi pada dasarnya senarai (deretan) variabel bertipe serupa. Kita lebih dahulu menciptakan variabel *array* dari tipe yang dikehendaki.

int monthDays[];

Meskipun deklarasi telah menetapkan bahwa **monthDays** adalah variabel *array*, namun sesungguhnya tidak ada *array* yang diciptakan pada saat itu. Nilai dari **monthDays** adalah nilai **null** yang merepresentasikan *array* tanpa nilai dan bukan bernilai nol.

Bentuk umum **new()** untuk *array* 1 dimensi adalah sebagai berikut:

array-var = new() type[size]

Dalam hal ini, *type* menspesifikasikan tipe data yang dialokasikan, *size* menspesifikasikan jumlah dari elemen *array*, dan *array-var* adalah variabel *array* yang dirangkai ke *array*. Dengan demikian, untuk menggunakan **new()**, kita harus menspesifikasikan tipe dan jumlah elemen untuk dialokasikan. Elemen-elemen di *array* yang dialokasikan oleh **new()** akan secara otomatis dinisialisasi ke nol.

Perhatikan contoh berikut:

monthDays = new() int[12];

Setelah pernyataan diatas dieksekusi, **monthDays** akan mengacu ke *array* berisi 12 bilangan **int**, kedua belas angka tersebut dinisialisasi dengan nol.

Ingat:

Proses untuk dapat memperoleh *array* ialah 2 langkah, yaitu:

1. Deklarasi variabel dari tipe *array* yang dikehendaki.

2. Alokasi memori yang akan menyimpan *array* menggunakan operator **new()** dan memberikannya ke variable *array*.

Setelah adanya pengalokasian *array*, kita dapat mengakses elemen spesifik di *array* dengan menspesifikasikan indeks, yaitu dengan menspesifikasikannya di pasangan kurung siku. Semua *array* dimulai dengan indeks **nol**. Berikut contoh untuk memberikan nilai 29 ke elemen kedua dari **monthDays**, digunakan perintah:

monthDays[1] = 29;

Berikut ini untuk menampilkan nilai yang disimpan pada indeks 3, yaitu:

System.out.println(monthDays[3]);

Berikut ini salah satu contoh penciptaan untuk *array* yang berisi jumlah hari maksimum di masing-masing bulan:

```
class Bulan
{
      public static void main(String[] args)
      {
      // langkah 1: deklarasi variabel array
            int monthDays[];
      // langkah 2: penciptaan array bilangan int dengan 12 elemen
            monthDays = new int[12];
            monthDays[0] = 31;
            monthDays[1]  =  29;
            monthDays[2]  =  31;
            monthDays[3]  =  30;
            monthDays[4]  =  31;
            monthDays[5]  =  30;
            monthDays[6]  =  31;
            monthDays[7]  =  31;
            monthDays[8]  =  30;
            monthDays[9]  =  31;
            monthDays[10]  =  30;
            monthDays[11] = 31;
            System.out.println("Agustus   mempunyai   "+monthDays[7]+"
hari");
      }
}
```

Hasil (*output*) dari contoh listing program diatas adalah **Agustus mempunyai 31 hari**. Simbol "**//**" (dua garis miring) digunakan untuk suatu komentar (informasi singkat) program, dimana komentar tersebut tidak dieksekusi.

2.1.3 *Array* Multi Dimensi

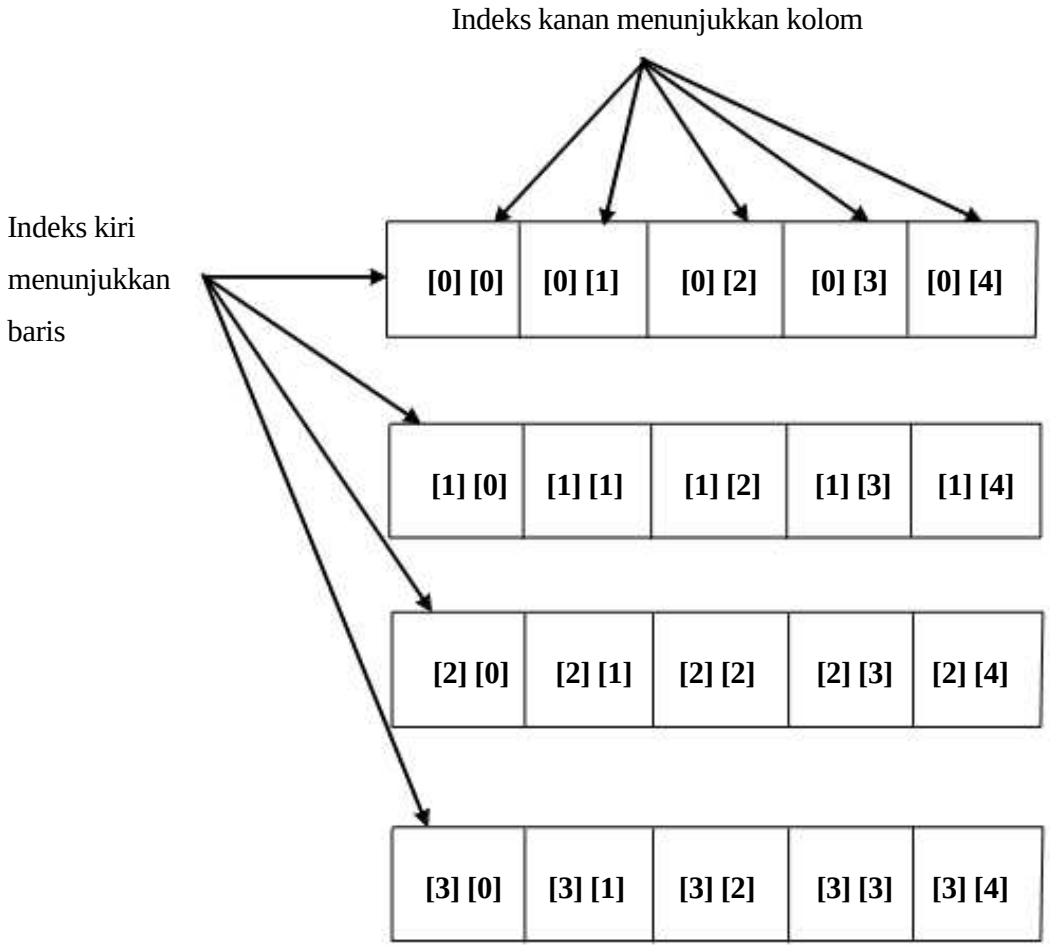
Di Java, *array* multidimensi sesungguhnya *array* dari *array*. Untuk mendeklarasikan variabel *array*, menspesifikasikan masing-masing indeks menggunakan himpunan kurung siku yang lain.

Contoh:

Array 2 dimensi bertipe **int** bernama **intArr** dideklarasikan dan diciptakan dengan perintah sebagai berikut:

int int2DArr[][] = new() int [4][5];

Perintah ini mendeklarasikan dan mengalokasikan *array* 4 kali 5 dan memberikan ke variabel **int2DArr**. Secara internal matriks ini diimplementasikan sebagai *array* dari *array* bertipe **int**. Secara konseptual *array* ini digambarkan sebagai berikut:



int two D [] [] = new int (4)(5);

Gambar 2.1 Array 2 Dimensi 4 x 5

Berikut contoh listing program yang memberikan angka ke masing-masing elemen di *array* dari kiri ke kanan, puncak ke bawah, kemudian menampilkan masing-masing elemen:

```
class Multidimensi
{
   public static void main(String[] args){
      //Langkah 1: deklarasi variabel array dan penciptaan array
      int int2DArr[][] = new int[4][5];
      int k=0;
      for (int i=0;i<4;i++){
            for (int j=0;j<5;j++){
                  int2DArr[i][j] = k++;
            }
      }
      for (int i=0;i<4;i++){
            for (int j=0;j<5;j++){
                  System.out.print(int2DArr[i][j] + " ");
            }
            System.out.println();
      }
   }
}
```

Hasil (*output*) dari contoh listing program diatas:

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```

Ketika adanya pengalokasian *array* multidimensi, kita hanya perlu menspesifikasikan memori untuk dimensi pertama (paling kiri). Kita dapat mengalokasikan dimensi sisanya secara terpisah.

Contoh:

Kita mengalokasikan dimensi pertama **int2DArrA** saat dideklarasikan, kemudian dapat mengalokasikan dimensi kedua secara manual sebagai berikut:

```
// Deklarasi dan alokasi dimensi pertama
int int2DArrA = new() int[4][];

// Alokasi dimensi kedua secara manual
int2DarrA[0] = new() int[5];
int2DarrA[1] = new() int[5];
int2DarrA[2] = new() int[5];
int2DarrA[3] = new() int[5];
```

Kelebihan pengalokasian masing-masing dimensi secara manual ialah kita bisa tidak mengalokasikan jumlah elemen yang sama untuk masing-masing dimensi. Karena *array* multidimensi sesungguhnya *array* dari *array*, maka panjang masing-masing *array* dapat dikendalikan/ ditentukan.

Contoh:

```
class ArrayUnequallnt2D{
        public static void main(String[] args){
                // Langkah 1: deklarasi variabel array dan penciptaan array
                int int2DUnequalArr[][] = new int[4][];
                int2DUnequalArr[0]  =  new  int[1];
                int2DUnequalArr[1]  =  new  int[2];
                int2DUnequalArr[2]  =  new  int[3];
                int2DUnequalArr[3]  =  new  int[4];
                int k=0;
                for (int i=0;i<4;i++){
                        for (int j=0;j<i+1;j++){
                                int2DUnequalArr[i][j] = k++;
                        }
                }
                for (int i=0;i<4;i++){
                        for (int j=0;j<i+1;j++){
                                System.out.print(int2DUnequalArr[i][j] + " ");
                        }
                        System.out.println();
                }
        }
}
```

Hasil (*output*) dari contoh listing program diatas:

```
0
1 2
3 4 5
6 7 8 9
```

Inisialisasi

Kita dapat melakukan inisialisasi *array* multidimensi, yaitu kita cukup mengapit masing inisialisasi dimensi dengan satu pasangan kurung kurawal.

Contoh:

```
class Inisialisasi{
      public static void main(String[] args){
            // Deklarasi dan inisialisasi
            double  aDoubleMat[][]  =
            {
```

```
                  {0*0,  1*0,  2*0,  3*0},
                  {0*1,  1*1,  2*1,  3*1},
                  {0*2, 1*2, 2*2, 3*2},
                  {0*3,  1*3,  2*3,  3*3}
            };
            for (int i=0;i<4;i++){
                  for (int j=0;j<4;j++){
                        System.out.print(aDoubleMat[i][j]+ " ");
                  }
                  System.out.println();
            }
      }
}
```

Hasil (*output*) dari contoh listing program diatas:

```
0.0 0.0 0.0 0.0
0.0 1.0 2.0 3.0
0.0 2.0 4.0 6.0
0.0 3.0 6.0 9.0
```

4.2 Percabangan

Sampai sekarang, semua contoh program yang diberikan sepenuhnya *linier*. Pernyataan-pernyataan dijalankan secara berurutan, satu setelah yang lainnya, sampai semuanya selesai dijalankan. Program yang lebih menarik akan membutuhkan persyaratan tertentu untuk menjalankan suatu pernyataan, dan melewati pernyataan lainnya. Java menyediakan sejumlah mekanisme untuk menghasilkan pengaturan bagian program mana yang dijalankan, berupa pernyataan-pernyataan percabangan. Terdapat beberapa jenis percabangan di java, yaitu: *if, if-else*, dan *switch*.

2.2.1 Percabangan *if*

Percabangan *if* di Java menyatakan pernyataan akan dieksekusi jika memenuhi syarat atau kondisi tertentu. Sintaks percabangan ini adalah sebagai berikut:

ifStatement

Gambar 2.2 Diagram Sintaks Percabangan if

Contoh:

```
public class If
{
	public static void main(String[] args){
		int anInt = 0;
		if (anInt==0){
			System.out.println("Variabel anInt bernilai nol");
		}
	}
}
```

Hasil (*output*) dari contoh listing program diatas: **`Variabel anInt bernilai nol`**.

Contoh program *if* diatas dapat diringkas sebagai berikut:

```
public class Ifringkas
{
	public static void main(String[] args){
		int anInt = 0;
		if (anInt==0)
			System.out.println("Variabel anInt bernilai nol");
	}
}
```

Hasil (*output*) dari contoh listing program diatas: **`Variabel anInt bernilai nol`**.

2.2.2 Percabangan *if-else*

Percabangan ini untuk memilih salah satu dari 2 kemungkinan kemunculan. Dengan kata lain, bentuk *if-else* menyebabkan eksekusi dijlankan melalui sekumpulan **boolean**, sehingga hanya bagian tertentu program yang dijalankan. Berikut bentuk umum pernyataan **if-else**:

```
if (condition or Boolean expression)
	statement1
else
	statement2
```

Klausa *else* bersifat pilihan (*optional*). Setiap *statement* dapat berupa kumpulan pernyataan yang dibatasi dengan kurung kurawal. *Boolean expression* atau

pernyataan *boolean* dapat berupa sembarang pernyataan yang menghasilkan besaran *boolean*. Jika *condition* atau *Boolean expression* dievaluasi bernilai **true**, maka *statement1* dieksekusi, sedangkan jika *condition* atau *Boolean expression* dievaluasi bernilai **false**, maka *statement2* yang dieksekusi.

Contoh 1:

```
public class Ifelse1
{
public static void main(String[] args){
      int anInt = 0;
      if (anInt==0)
      System.out.println("Variabel  anInt  bernilai  nol");
      else
      System.out.println("Variabel  anInt  tidak  bernilai  nol");
      }
}
```

Hasil (*output*) dari contoh listing program diatas: **`Variabel anInt bernilai nol`**.

Contoh 2:

```
public class Ifelse2
{
      public static void main(String[] args){
            int anInt = 1;
            if (anInt==0)
                  System.out.println("Variabel anInt bernilai nol");
            else
                  System.out.println("Variabel  anInt  tidak  bernilai
nol");
      }
}
```

Hasil (*output*) dari contoh listing program diatas:

`Variabel anInt tidak bernilai nol`.

Bahasa Java menyediakan beragam kendali percabangan tidak lokal, yaitu: *break*, *return*, dan *continue*.

Break

Java tidak memiliki pernyataan **goto**. Penggunaan **goto** di bahasa pemrograman lain adalah cara untuk mencabang secara sembarang, yang membuat program sulit untuk dimengerti dan mengurangi optimasi *compiler* tertentu. Namun,

ada beberapa keadaan dimana **goto** berguna dan bentuk yang sah untuk pengaturan program.

Pernyataan **break** pada Java dirancang untuk mengatasi semua kasus tersebut. Istilah **break** mengacu pada proses memecahkan blok program. Proses tersebut memerintahkan *runtime* untuk menjalankan program di belakang blok tertentu. Untuk dapat ditunjuk, suatu blok diberi nama, dan Java memiliki bentuk label untuk menyatakan nama suatu blok.

Contoh:

```
class Break
{
      public static void main(String[] args)
      {
            boolean t =true;
a:    {
b:          {
c:                {

            System.out.println("Sebelum break");
            if (t)
                  break b;
                  System.out.println("Ini tidak akan dieksekusi");
            }
            System.out.println("Ini tidak akan dieksekusi");
            }
            System.out.println("Ini adalah setelah b");
            }
      }
}
```

Hasil (*output*) dari contoh listing program diatas:

```
Sebelum break
Ini adalah setelah b
```

Return

Java menggunakan bentuk sub-rutin yang disebut *method* untuk mengimplementasikan antarmuka procedural ke kelas objek. Setiap saat dalam *method* dapat digunakan pernyataan **return** yang menyebabkan eksekusi mencabang kembali ke pemanggil *method*.

Contoh:

```
class Return1
{
      public static void main(String[] args)
      {
            boolean t = true;
            System.out.println("Before the return");
            if (t)
                  return;
```

```
            System.out.println("This won't execute");
      }
}
```

Hasil (*output*) dari contoh listing program diatas:

```
Before the return
```

Continue

Seringkali kita ingin keluar lebih cepat dari perulangan. Kita mungkin juga ingin meneruskan perulangan, tetapi harus menghentikan sisa proses pada program untuk iterasi yang bersangkutan. Ini dilakukan dengan **goto** yang memintas program, tetapi masih di dalam perulangan. Pernyataan *continue* di Java melakukan persis seperti itu. Berikut **contoh** program penggunaan *continue* yang menyebabkan 2 bilangan dicetak dalam setiap baris:

```
class Continue1
{
      public static void main(String[] args)
      {
            for (int i = 0; i<10; i++) {
            System.out.print(i + " "); if
            (i% 2 == 0)
                  continue;
            System.out.println("");
            }
      }
}
```

Hasil (*output*) dari contoh listing program diatas:

```
0 1
2 3
4 5
6 7
8 9
```

Program *continue1* diatas menggunakan **System.out.print** untuk mencetak tanpa pindah baris. Kemudian digunakan operator **mod**, % untuk memeriksa apakah indeksnya genap, atau dapat dibagi 2. Jika genap, perulangan dilanjutkan tanpa mencetak perpindahan baris.

2.2.3 Percabangan *switch*

Pernyataan *switch* memberikan suatu cara untuk mengirim bagian program berdasarkan nilai suatu variabel atau pernyataan tunggal. Percabangan *switch* dimaksudkan untuk menangani banyak kemungkinan kemunculan. Berikut ini bentuk umum percabangan *switch*:

```
switch (expression) {
case valeue1:
      break;
case value2;
      break;
case valueN;
      break;
default;
}
```

Expression dapat menghasilkan suatu tipe sederhana, dan setiap *value* yang disebutkan pada pernyataan **case** harus berupa tipe yang cocok. Pernyataan *switch* bekerja dengan cara seperti ini: nilai **expression** dibandingkan dengan setiap nilai pada pernyataan **case**. Jika ada yang cocok, maka urutan program yang ada di belakang penyataan **case** akan dijalankan. Jika tidak ada yang cocok, maka pernyataan **default** yang dijalankan. Pernyataan **default** merupakan pilihan juga. Jika tidak ada yang cocok dan tidak ada **default**, tidak ada yang dikerjakan.

Kata kunci **break** sering digunakan dalam pernyataan *switch* tanpa label. Di dalam pernyataan *switch*, **break** tanpa label menyebabkan eksekusi percabangan langsung menuju akhir pernyataan *switch*. Jika kita tidak menuliskan **break**, maka eksekusi akan dilanjutkan ke dalam **case** selanjutnya.

Pernyataan **break** adalah *optional* atau pilihan karena terdapat keperluan untuk mengeksekusi beberapa **case** sekaligus jika pernyataan **case** diatasnya di eksekusi. Pernyataan *switch* berbeda dengan pernyataan *if*. Dimana *switch* hanya dapat menguji kesamaan, sedangkan pernyataan *if* dapat melakukan evaluasi sembarang tipe ekspresi **boolean**. Dengan demikian *switch* terlihat seperti hanya mencocokkan diantara nilai-nilai ekspresi dan konstanta-konstanta **case**.

Contoh 1:

```
public class Switch1
{
      public  static  void  main(String[]  args)
      {
            int a;
            a = 5;
            switch (a) {
```

```
                case  0:  System.out.println("Nol");
                case  1:  System.out.println("Satu");
                case 2: System.out.println("Dua");
                case  3:  System.out.println("Tiga");
                case  4:  System.out.println("Empat");
                case 5: System.out.println("Lima");
                case 6: System.out.println("Enam");
                case 7: System.out.println("Tujuh");
                case 8: System.out.println("Delapan");
                case 9: System.out.println("Sembilan");
                default:  System.out.println("bukan  karakter  digit");
            }
      }
}
```

Hasil (*output*) dari contoh listing program diatas:

```
Lima
Enam
Tujuh
Delapan
Sembilan
bukan karakter digit
```

Contoh 2:

```
public class Switchbreak2
{
      public  static  void  main(String[]  args)
      {
            int a;
            a = 5;
            switch (a) {
                case  0:  System.out.println("Nol");
                break;
                case  1:  System.out.println("Satu");
                break;
                case  2:  System.out.println("Dua");
                break;
                case  3:  System.out.println("Tiga");
                break;
                case  4:  System.out.println("Empat");
                break;
                case  5:  System.out.println("Lima");
                break;
                case  6:  System.out.println("Enam");
                break;
                case  7:  System.out.println("Tujuh");
                break;
                case  8:  System.out.println("Delapan");
                break;
                case  9:  System.out.println("Sembilan");
                break;
                default: System.out.println("bukan karakter digit");
            }
      }
}
```

Hasil (*output*) dari contoh listing program diatas:

```
Lima
```

2.3 Perulangan

Terdapat beberapa jenis perulangan pada Java, yaitu

- for
- while
- do-while

Pernyataan-pernyataan diatas menciptakan *loop*. *Loop* secara berulang mengeksekusi sebarisan instruksi yang sama sampai kondisi akhir ditemui. Dengan kata lain, *looping* atau *loop* artinya mengulangi eksekusi blok program tertentu sampai tercapai kondisi untuk menghentikannya (terminasi). Setiap perulangan memiliki 4 bagian: inisialisasi (*initialization*), badan program (*body*)/ *statement*, iterasi (*iteration*), dan *termination.*

2.3.1 Perulangan *for*

Perulangan *for* menyediakan sarana mengulang kode dalam jumlah yang tertentu. Pengulangan ini terstruktur untuk mengulangi kode sampai tercapai batas tertentu. Berikut bentuk dasar perulangan *for*:

for(InitializationExpression; LoopCondition; StepExpression)

statement

- **InitializationExpression**, digunakan untuk inisialisasi variabel kendali perulangan.
- **LoopCondition**, membandingkan variabel kendali perulangan dengan suatu nilai batas.
- **StepExpression**, menspesifkasikan cara variabel kendali dimodifikasi sebelum iterasi berikutnya dari perulangan.

Contoh:

```
public class For1
{
      public static void main(String[] args)
      {
            int i;
            for (i = 1;i<11;i++)
            System.out.println(i);
      }
```

```
}
```

Hasil (*output*) dari contoh listing program diatas:

```
1
2
3
4
5
6
7
8
9
10
```

2.3.1.1 Deklarasi variabel Kendali *loop* Di Dalam *for-loop*

Sering variabel yang digunakan untuk mengendalikan *for-loop* hanya diperlukan untuk tujuan *loop* dan tidak digunakan di selain keperluan itu. Dalam kasus seperti ini, sebaiknya kita mendeklarasikan variabel di dalam bagian inisialisasi **for**.

Contoh:

```
public class For2
{
      public  static  void  main(String[]  args)
      {
            for (int i = 1;i<11;i++)
            System.out.println(i);
       }
}
```

Hasil (*output*) dari contoh listing program diatas:

```
1
2
3
4
5
6
7
8
9
10
```

2.3.1.2 Penggunaan Koma Di *for-loop*

Sering kita memberikan lebih dari satu pernyataan di bagian inisialisasi dan iterasi di *for-loop*.

Contoh:

```
public class Forkoma1{
      public static void main(String[] args){
            for(int i=1,j=4; i<j; i++,j--){
```

```
            System.out.println("i = " + i);
            System.out.println("j = " + j);
            }
      }
}
```

Hasil (*output*) dari contoh listing program diatas:

```
i = 1
j = 4
i = 2
j = 3
```

Loop diatas dikendalikan interaksi 2 variabel. Karena *loop* dikendalikan 2 variabel, maka lebih baik kedua variabel tersebut dimasukkan ke pernyataan **for**, dibanding j ditangani secara manual terpisah. Java mengijinkan 1 variabel atau lebih berada di pernyataan *loop*, bahkan Java mengijinkan banyak pernyataan berada di bagian inisialisasi dan iterasi dari pernyataan **for**. Masing-masing pernyataan itu harus dipisahkan menggunakan tanda koma.

2.3.1.3 Pernyataan *for-loop* Bersarang

Java memungkinkan *loop* yang disarangkan di *loop* yang lain. Satu *loop* berada di dalam *loop* yang lainnya.

Contoh:

```
public class Loopbersarang1{
      public static void main(String[] args){
            for(int i=0;i<10;i++){
                        for(int j=i;j<10;j++){
            System.out.print("*");}
            System.out.println();
            }
      }
}
```

Hasil (*output*) dari contoh listing program diatas:

```
         *

      **
      *

   **
   *

**
*
```

2.3.2 Perulangan *while*

Pengulangan *while* mempunyai kondisi yang menngendalikan pernyataan perulangan. Berikut bentuk dasar perulangan *while*:

[initialization;]

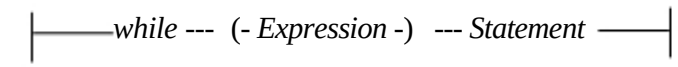
while (Pernyataan *boolean*/ LoopCondition/ *Termination)* {

Statement;

[iteration;]

}

WhileStatement



Gambar 2.3 Diagram Sintaks Perulangan *while*

Jika **LoopCondition/** pernyataan *boolean* dievaluasi **true**, maka *statement* dieksekusi dan proses terus berlanjut berulang kali. Penting diketahui bahwa **LoopCondition** muncul sebelum badan dari pernyataan. Ini berarti jika **LoopCondition** sejak semula dievaluasi **false**, maka *statement* tidak pernah dieksekusi. Hal ini merupakan perbedaan penting antara perulangan *while* dengan perulangan *do-while*. Berikut ini perulangan dengan *while* yang menghitung mundur dari **n**, menuliskan tepat sepuluh "track".

Contoh:

```
class While1{
      public static void main(String[] args){
            int n = 10;
            while (--n >= 0)
            System.out.println("track " + (n + 1));
      }
}
```

Hasil (*output*) dari contoh listing program diatas:

```
track 10
track  9
track  7
track  6
track  5
track  4
track  3
track  2
track 1
```

2.3.3 Perulangan *do-while*

Perulangan *do-while* serupa dengan perulangan *while*, hanya saja pemeriksaan kondisi yang dilakukan adalah setelah pernyataan. Kadang-kadang diinginkan untuk mengeksekusi badan program dengan perulangan *while* sekurang-kurangnya sekali, bahkan jika pernyataan *boolean*-nya langsung menghasilkan *false*. Akibarnya, kita harus memeriksa pernyataan *boolean*/ *LoopCondition* di akhir perulangan, bukan di awal seperti pada perulangan *while*. Bentuk umum atau bentuk dasar perulangan *do-while*:

[initialization;]

do {

Statement;

[iteration;]

*} while(*Pernyataan *boolen/ LoopCondition/ Termination);*

DoStatement

Contoh:

```
class DoWhile{
      public static void main(String[] args){
            int n = 10;
            do
            System.out.println("track  "  +  n);
            while (--n > 0);
      }
}
```

Hasil (*output*) dari contoh listing program diatas:

```
track  10
track   9
track   7
track   6
track   5
track   4
track   3
track   2
track  1
```

Pada contoh program *do-while* diatas, kita menggunakan pernyataan (--n > 0) untuk memeriksa kondisi perulangan *do-while*. Pernyataan **--n** dijalankan, lalu **n** dikurangi

1, sehingga menghasilkan nilai baru **n**. Kemudian **n** dibandingkan dengan nol. Jika masih lebih besar dari nol, perulangan diteruskan, jika tidak maka dihentikan.

STEP BY STEP

1. Jelaskan tentang array dan sebutkan jenis-jenis array!
2. Tuliskan bentuk umum dari array!
3. Sebutkan sintaks percabangan, dan perulangan pada Java!
4. Tuliskan bentuk umum sintaks percabangan, dan perulangan masing-masing!

MATERI PRAKTIKUM

Buat program dengan aturan sebgai berikut:

a. Menampilkan kata ”Daftar Pilihan” dilayar.

b. Menampilakan menu :

1. Simpan data nilai
2. perhitungan faktorial
3. keluar

Pilih salah satu[1..3]:<input angka>

c. Jika memilih angka 1 maka akan dilakukan proses berikut:

- Menyimpan nilai sebanyak 5 buah di dalam variabel array dan juga berarti sebanyak 5 kali user menginputkan nilai melalui keyboard.
  ”Masukkan nilai=<input angka>” lakukan perulangan sebanyak 5 kali.
- Nilai yang diinputkan dan disimpan dalam variabel array akan ditampilkan sebagai berikut:
  ”Elemen ke-0=<nilai yang tersimpan dalam variabel array0>”
  ”Elemen ke-1=<nilai yang tersimpan dalam variabel array1>”
  ”Elemen ke-2=<nilai yang tersimpan dalam variabel array2>”
  ”Elemen ke-3=<nilai yang tersimpan dalam variabel array3>”
  ”Elemen ke-4=<nilai yang tersimpan dalam variabel array4>”

d. Jika memilih angka 2 maka akan dilakukan proses berikut:

- Menghitung faktorial integer dari 1-10, deklarasikan batas nilai dan deklarasikan nilai awal faktorial yaitu 1.
- Lakukan proses perulangan bersarang(nested loop) dan lakukan proses perhitungan hasil = nilai faktorial * hasil
- Jika telah perulangan telah berhenti maka akan menghasilkan output berikut:

  0! Adalah 1

  1! Adalah 1

  ...

  ...<sampai>

  ...

  9! Adalah 362880

  10! Adalah 3628800

LAPORAN AKHIR

Buatlah kesimpulan mengenai sintaks array, percabangan dan perulangan bahasa pemrograman JAVA yang telah dipraktekkan beserta dengan program-program yang telah dibuat berikut tampilan hasilnya.