Method Pada Java - Java Method - Java NetBeans

By Wisnu Nunu 20:23 1 comment

Pada postingan sebelumnya saya sudah membahas berbagai macam method pada java tapi saya belum membahas apa itu method ? apa saja jenis-jenis method ?, mari kita bahas lebih dalam lagi.
Method pada Java adalah kumpulan pernyataan yang dikelompokkan bersama-sama untuk melakukan operasi, misalnya ketika kita memanggil System.out.println() method maka sistem akan mengeksekusi satu atau lebih pernyataan untuk menampilkan pesan pada konsol.
Pada postingan ini kita akan belajar membuat method kita sendiri dengan atau tanpa nilai kembali, memanggil method dengan atau tanpa parameter, dan menerapkan method abstrak dalam rancangan program.

Membuat Method

Ingatlah contoh berikut yang menjelaskan syntax dari method:

```
public static int methodName(int a, int b) {
  // body
}
```

Catatan :

- **public static :** modifier.
- **int :** tipe data.
- **methodName :** nama method/metode.
- **a, b :** parameter formal.
- **int a, int b :** daftar parameter.

Definisi method terdiri dari header method dan body method. Hal yang sama seperti ditunjukkan berikut ini:

```
modifier returnType nameOfMethod (Parameter List) {
 // method body
}
```

Catatan :

- **modifier :** Mendefinisikan jenis akses method tentu saja digunakan secara opsional sesuai dengan kondisi yang dibutuhkan.
- **returnType :** Method yang mungkin dapat mengembalikan nilai.
- **nameOfMethod :** Nama method yang terdiri dari nama method dan daftar parameter.
- **Parameter List :** Daftar parameter, terdiri dari jenis, order(perintah), dan jumlah parameter method, digunakan secara opsional, bisa berisi nol parameter.
- **method body :** Method body mendefinisikan apa yang method lakukan dengan pernyataan.

Contoh :

Berikut adalah source code dari method max(), method ini membutuhkan dua parameter yaitu num1 dan num2 dan mengembalikan maksimal dua num :

```
public static int minFunction(int n1, int n2) {
   int min;
   if (n1 > n2)
      min = n2;
   else
      min = n1;

   return min;
}
```

Memanggil Method:

Untuk menggunakan method kita harus memanggilnya, terdapat dua cara dalam memanggil method yaitu method yang mengembalikan nilai atau yang tidak mengembalikan apa-apa.
Proses memanggil method sangat sederhana. Ketika kita memanggil method kontrol program akan ditransfer ke method tersebut. Method yang disebut ini kemudian kembali pada kontrol pemanggil dalam dua kondisi, ketika :

- Statement (pernyataan) kembali dijalankan.
- Mencapai akhir method.

Method kembali dianggap sebagai panggilan untuh sebuah pernyataan.

```
System.out.println("programmergalaulagi.blogspot.com");
```

Method nilai kembali dapat dilihat pada contoh berikut.

```
int result = sum(6, 9);
```

Contoh :

Berikut adalah contoh untuk menunjukkan bagaimana menentukan method dan cara memanggilnya :

```
public class Test{

   public static void main(String[] args) {
      int a = 11;
      int b = 6;
      int c = minFunction(a, b);
      System.out.println("nilai terkecil = " + c);
   }

   /** mengembalikan nilai terkecil antara dua nomor*/
   public static int minFunction(int n1, int n2) {
      int min;
      if (n1 > n2)
         min = n2;
      else
         min = n1;

      return min;
   }
}
```

Berikut adalah hasil dari kode diatas.

```
nilai terkecil = 6
```

Void Keyword

Void keyword memungkinkan kita untuk membuat method yang tidak mengembalikan nilai. Dalam contoh berikut kita akan mempertimbangkan void method methodRankPoints, method ini merupakan void method yang tidak mengembalikan nilai apapun. Cara memanggil void method haurs menjadi statement yaitu methodRankPoints(255,7); Ini adalah pernyataan Java yang berakhir dengan titik koma seperti contoh berikut:

Contoh :

```
public class TestVoid{

   public static void main(String[] args) {
      methodRankPoints(255.7);
   }

   public static void methodRankPoints(double points) {
      if (points >= 202.5) {
         System.out.println("Rangking:A1");
      }
      else if (points >= 122.4) {
         System.out.println("Rangking:A2");
      }
      else {
         System.out.println("Rangking:A3");
      }
   }
}
```

Berikut adalah hasil dari kode diatas:

```
Rangking:A1
```

Passing Parameter oleh Nilai

Saat bekerja dibawah calling proses, argumen dilewati. Ini harus dalam urutan yang sama seperti parameter masing-masing dalam spesifikasi method.
Passing Parameter oleh nilai berarti memanggil method untuk parameter. Melalui ini nilai argumen akan dilewatkan ke parameter.

Contoh :

Program berikut menunjukkan contoh passing parameter dengan nilai. Nilai-nilai argumen tetap sama bahkan setelah pemanggilan method.

```
public class SwappingTest{

   public static void main(String[] args) {
      int a = 30;
      int b = 45;

      System.out.println("sebelum swapping, a = " +
                         a + " and b = " + b);

      swapFunction(a, b);
      System.out.println("\n**sekarang, sebelum dan setelah swapping nilai akan sama**:");
      System.out.println("setelah swapping, a = " +
```

```
                        a + " dan b adalah " + b);
    }

    public static void swapFunction(int a, int b) {

        System.out.println("sebelum swapping(didalam), a = " + a
                           + " b = " + b);
        // Swap n1 with n2
        int c = a;
        a = b;
        b = c;

        System.out.println("setelah swapping(didalam), a = " + a
                           + " b = " + b);
    }
}
```

Berikut adalah hasil dari kode diatas.

```
sebelum swapping, a = 30 dan b = 45
sebelum swapping(didalam), a = 30 b = 45
setelah swapping(didalam), a = 45 b = 30

**sekarang, sebelum dan setelah swapping akan sama **:
After swapping, a = 30 and b is 45
```

Method Overloading

Ketika kelas memiliki dua atau lebih method dengan nama yang sama tetapi parameter yang berbeda, yang dikenal sebagai metode overloading. Hal ini berbeda dari override. Dalam override metode memiliki nama method yang sama, jenis, jumlah parameter dll

Mari kita pertimbangkan contoh yang ditampilkan sebelumnya untuk menemukan nomor minimal tipe integer. Katakanlah jika kita ingin mencari jumlah minimum tipe double. Maka konsep Overloading akan diperkenalkan untuk membuat dua atau lebih metode dengan nama yang sama tetapi parameter yang berbeda.

Berikut contoh yang akan menjelaskan:

```
public class OverloadingTest{

  public static void main(String[] args) {
    int a = 11;
    int b = 6;
    double c = 7.3;
    double d = 9.4;
    int result1 = minFunction(a, b);
    // nama fungsi yang sama name dengan parameters yang berbeda
    double result2 = minFunction(c, d);
    System.out.println("nilai terkecil = " + result1);
    System.out.println("nilai terkecil = " + result2);
  }

 // integer
  public static int minFunction(int n1, int n2) {
    int min;
    if (n1 > n2)
      min = n2;
```

```
    else
      min = n1;

    return min;
  }
  // double
  public static double minFunction(double n1, double n2) {
    double min;
    if (n1 > n2)
      min = n2;
    else
      min = n1;

    return min;
  }
}
```

Berikut adalah hasil dari kode diatas.

```
nilai terkecil = 6
nilai terkecil = 7.3
```

Overloading method membuat program mudah dibaca. Di sini, dua metode diberikan nama yang sama tetapi dengan parameter yang berbeda. Menghasilkan jumlah minimum dari integer dan tipe double.

Menggunakan Argumen Command-Line

Kadang-kadang kita ingin menyampaikan informasi ke dalam program ketika kita menjalankannya. Hal ini dicapai dengan melewati argumen baris perintah untuk main().
Sebuah baris perintah argumen adalah informasi yang langsung termasuk nama program pada baris perintah ketika dijalankan. Untuk mengakses argumen baris perintah dalam program Java cukup mudah .Mereka disimpan sebagai string dalam array String dilewatkan ke main().

Contoh:

Berikut adalah program akan menampilkan semua argumen baris perintah yang disebutkan diatas.

```
public class CommandLine {

   public static void main(String args[]){
      for(int i=0; i<args.length; i++){
         System.out.println("args[" + i + "]: " +
                                          args[i]);
      }
   }
}
```

Program yang dilaksanakan seperti yang ditunjukkan dibawah ini:

```
$java CommandLine ini adalah command line 200 -100
```

Berikut adalah hasil dari kode diatas.

Konstruktor

Sebuah konstruktor menginisialisasi sebuah objek ketika dibuat. Ini memiliki nama yang sama dengan class dan syntax mirip dengan method. Namun, konstruktor tidak memiliki tipe kembali eksplisit.

Biasanya, Anda akan menggunakan konstruktor untuk memberikan nilai awal untuk variabel instance didefinisikan oleh class, atau untuk melakukan prosedur startup lain yang diperlukan untuk membuat objek sepenuhnya terbentuk.

Semua calss memiliki konstruktor, apakah kita menentukan satu atau tidak, karena Java secara otomatis menyediakan konstruktor default yang menginisialisasi semua variabel anggota ke nol. Namun, setelah kita mendefinisikan konstruktor kitasendiri, constructor default tidak lagi digunakan.

Contoh :

Berikut adalah contoh sederhana yang menggunakan kontruktor tanpa parameter.

```
// A simple constructor.
class MyClass {
  int x;

  // Following is the constructor
  MyClass() {
    x = 10;
  }
}
```

Kita akan memanggil konstruktor untuk menginisialisasi objek sebagai berikut.

```
public class ConsDemo {

   public static void main(String args[]) {
      MyClass t1 = new MyClass();
      MyClass t2 = new MyClass();
      System.out.println(t1.x + " " + t2.x);
   }
}
```

Paling sering, kita memerlukan constructor yang menerima satu atau lebih parameter. Parameter ditambahkan ke konstruktor dengan cara yang sama bahwa mereka ditambahkan ke sebuah method, hanya menyatakan mereka dalam tanda kurung setelah nama konstruktor.

Contoh :

Berikut adalah contoh sederhana yang menggunakan konstruktor dengan parameter.

```
// constructor sederhana.
class MyClass {
   int x;

   // berikut adalah constructor
   MyClass(int i ) {
      x = i;
   }
}
```

Kita akan memanggil konstruktor untuk menginisialisaikan objek sebagai berikut.

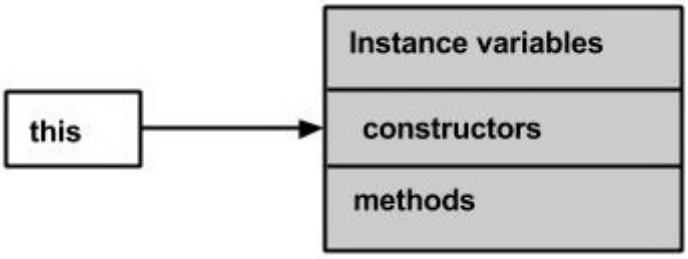
```
public class KonstruktorTest{

   public static void main(String args[]) {
      MyClass t1 = new MyClass( 10 );
      MyClass t2 = new MyClass( 20 );
      System.out.println(t1.x + " " + t2.x);
   }
}
```

Berikut adalah hasil dari kode diatas.

```
10 20
```

This keyword

This adalah keyword pada Java yang digunakan sebagai referensi ke objek yang dimaksud dari class saat ini , dengan method contoh atau konstruktor. Menggunakan this kita dapat merujuk anggota class seperti konstruktor, variabel, dan method.
This keyword hanya digunakan dalam method, instance atau konstruktor.



Sumber : http://www.tutorialspoint.com/java/java_methods.htm
Secara umum this keyword digunakan untuk:

- Membedakan variabel instances dengan variabel lokal jike mereka memiliki nama yang sama, dalam konstruktor atau method.

```
class Murid{

   int umur;
   Murid(int umur){
   this.umur=umur;
   }

}
```

- Memanggil satu jenis konstruktor (parameterized konstruktor atau default) dari lainnya pada class, ini dikenal sebagai eksplisit konstruktor invocation.

```
class Murid{

   int umur
   Murid(){
   this(20);
   }

   Murid(int umur){
   this.umur=umur;
   }

}
```

Contoh :

Berikut adalah contoh penggunaan this keyword untuk mengakses anggota dari sebuah class.

```
public class This_Test {

   //Instance variable num
   int num=10;

   This_Test(){
      System.out.println("Berikut adalah contoh program keyword this ");
   }

   This_Test(int num){
      //Invoking default konstruktor
      this();

      //Menyamakan local variable num keinstance variable num
      this.num=num;
   }

   public void greet(){
      System.out.println("Selamat datang di programmergalaulagi.blogspot.com");
   }

   public void print(){
      //Local variable num
      int num=20;

      //Mencetak instance variable
      System.out.println("nilai dari local variable num adalah : "+num);

      //Mencetak local variable
      System.out.println("nilai dari instance variable num adalah : "+this.num);

      //Memanggil greet method dari sebuah class
      this.greet();
   }

   public static void main(String[] args){
      //Class
      This_Test obj1=new This_Test();

      //Memanggil print method
      obj1.print();

      //nilai baru dari num variable melalui parametrized constructor
      This_Test obj2=new This_Test(30);

      //Memanggil print method lagi
      obj2.print();
   }

}
```

Berikut adalah hasil dari kode diatas:

```
Berikut adalah contoh program keyword this
nilai dari local variable num adalah : 20
niliai dari instance variable num adalah : 10
```

```
Selamat datang di programmergalaulagi.blogspot.com
Berikut adalah contoh program keyword this
nilai dari local variable num adalah : 20
nilai dari instance variable num adalah : 30
Selamat datang di programmergalalulagi.blogspot.com
```

Variable Arguments (var-args)

JDK 1.5 memungkinkan kita untuk menggunakan sejumlah variabel argumen dari jenis yang sama untuk sebuah metode. Parameter dalam metode ini dinyatakan sebagai berikut:

```
typeName... parameterName
```

Dalam metode deklarasi, Anda menentukan jenis diikuti oleh elipsis (...) Hanya satu parameter variabel-panjang dapat ditentukan dalam metode, dan parameter ini harus menjadi parameter terakhir. Parameter biasa harus mendahuluinya.

Contoh :

```
public class VarargsTest {

   public static void main(String args[]) {
      // Memanggil method dengan variable args
   printMax(34, 3, 3, 2, 56.5);
      printMax(new double[]{1, 2, 3});
   }

   public static void printMax( double... numbers) {
   if (numbers.length == 0) {
      System.out.println("Tidak ada argumen dilewati");
      return;
   }

   double result = numbers[0];

   for (int i = 1; i <  numbers.length; i++)
      if (numbers[i] >  result)
      result = numbers[i];
      System.out.println("Nilai max adalah " + result);
   }
}
```

Berikut adalah hasil dari kode diatas.

```
Nilai max adalah 56.5
Nilai max adalah 3.0
```

finalize() Method

Hal ini dimungkinkan untuk menentukan metode yang akan dipanggil sebelum kehancuran obyek oleh pengumpul sampah. Metode ini disebut finalize(), dan dapat digunakan untuk memastikan bahwa obyek berakhir bersih.
Misalnya, kita mungkin menggunakan finalize() memastikan bahwa file terbuka yang dimiliki oleh objek yang ditutup.
Untuk menambahkan finalizer untuk class, kita hanya menentukan metode finalize(). Java runtime memanggil metode yang setiap kali itu adalah untuk mendaur ulang sebuah objek dari kelas itu.

Di dalam metode finalize(), kita akan menentukan tindakan-tindakan yang harus dilakukan sebelum suatu benda hancur. Berikut adalah penulisan method finalize() yang biasa digunakan.

```
protected void finalize( )
{
   // finalization code disini
}
```

Di sini, kata kunci yang dilindungi adalah specifier yang mencegah akses untuk finalize() dengan kode didefinisikan di luar kelasnya.

Ini berarti bahwa kita tidak bisa tahu kapan atau bahkan finalize() akan dieksekusi. Misalnya, jika program kita berakhir sebelum pengumpulan sampah terjadi, finalize() tidak akan dieksekusi.

+Java Programming Tutorials +NetBeans