

Bab 6. Method

OBJEKTIF :

1. Mahasiswa mampu memahami mengenai materi Method pada Java.
2. Mahasiswa mampu memahami mengenai penggunaan Method pada program Java.
3. Mahasiswa mampu mensimulasikan penggunaan Method pada program Java untuk kejadian di dunia nyata.

6.1 Method

Method adalah kumpulan statement-statement yang diberikan nama. Sejauh ini kita telah melihat method dalam dua bentuk:

- Kita menuliskan method bernama `main` dalam setiap program yang kita tulis.
- Kita menggunakan method-method yang tersedia dalam Java seperti `System.out.println` dan `Math.pow`.

Dalam bab ini, kita akan mempelajari bagaimana menuliskan method kita sendiri, selain method `main`, yang dapat dieksekusi seperti kita mengeksekusi method-method yang sudah tersedia dalam Java.

Kenapa Kita Memerlukan Method?

Method digunakan untuk memecah sebuah program besar menjadi beberapa bagian kecil sehingga program dapat lebih mudah dikelola. Gambar berikut mengilustrasikan pemecahan suatu program menjadi beberapa bagian yang lebih kecil:

Program ini mempunyai satu method yang panjang dan kompleks yang berisi semua statement yang diperlukan untuk menyelesaikan suatu persoalan.

```
public class PersoalanBesar
{
    public static void main(String[] args)
    {
        statement;
        statement;
        statement;
        statement;
        statement;
        statement;
        statement;
        statement;
        statement;
        statement;
        statement;
        statement;
        statement;
        statement;
        statement;
        statement;
    }
}
```

Dalam program ini, persoalan dibagi menjadi persoalan-persoalan kecil yang setiap persoalannya diselesaikan oleh method terpisah.

```
public class PersoalanTerbagi
{
    public static void main(String[] args)
    {
        statement;
        statement;
        statement;
    }

    public static void method1()
    {
        statement;
        statement;
        statement;
    }

    public static void method2()
    {
        statement;
        statement;
        statement;
    }

    public static void method3()
    {
        statement;
        statement;
        statement;
    }
}
```

Alasan lain untuk menuliskan method adalah mengurangi pengulangan penulisan kode-kode program. Jika suatu tugas tertentu dilakukan dalam beberapa tempat dalam sebuah program, sebuah method dapat ditulis sekali untuk melakukan tugas tersebut, lalu dieksekusi setiap kali dibutuhkan. Keuntungan penggunaan method ini disebut dengan *code-reuse* atau penggunaan ulang kode. Ini karena kita hanya perlu menulis kode-kode method satu kali dan setelahnya kita dapat menggunakannya setiap kali kita membutuhkannya dalam bagian program kita yang lain.

Method `void` dan Method non-void

Method secara umum dapat dibedakan menjadi dua: method `void` dan method non-void. Method `void` adalah method yang melakukan sebuah tugas namun tidak menghasilkan sebuah nilai. Kata `void` dalam Bahasa Indonesia berarti kosong. Kata kosong mengacu ke tidak adanya nilai yang dihasilkan. Method non-void adalah method yang melakukan sebuah tugas dan menghasilkan sebuah nilai.

Salah satu contoh dari method `void` adalah method `System.out.println`. Method `System.out.println` melakukan tugas mencetak pesan ke console dan berakhir ketika tugas selesai dilakukan. Berikut adalah contoh penggunaan method `System.out.println`:

```
int number = 9;
System.out.println(number);
```

Statement pada baris 1 kode di atas mendeklarasikan dan menginisialisasi variabel `number` dengan nilai 9. Statement pada baris 2 memanggil method `System.out.println` dengan memberikan variabel `number` sebagai argument. Method `System.out.println` melakukan tugasnya dengan mencetak nilai dari variabel `number` ke console lalu berhenti.

Sedangkan method non-void adalah method yang menghasilkan sebuah nilai dan mengembalikan hasil nilai tersebut ke pemanggil method. Salah satu contoh dari method yang mengembalikan nilai adalah method `Math.pow` yang digunakan untuk mengangkat angka. Sebagai contoh, perhatikan kode berikut:

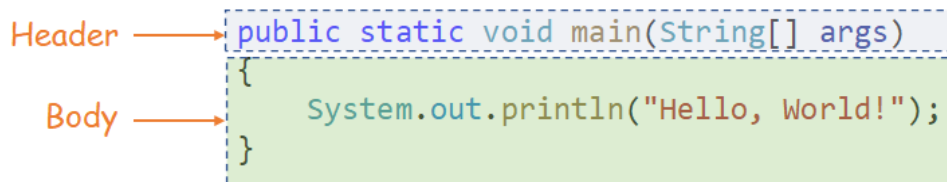
```
int hasil;
hasil = Math.pow(2, 3);
System.out.println(hasil);
```

Statement pada baris 1 mendeklarasikan variabel `hasil`. Statement pada baris 2 memanggil method `Math.pow` dengan memberikan argument 2 dan 3. Statement ini menyebabkan method `Math.pow` dieksekusi dan melakukan tugasnya mengkalkulasi 2^3 lalu nilai hasil kalkulasi dikembalikan ke pemanggil method dan ditugaskan ke variabel `hasil`. Ketika kita memanggil method non-void umumnya kita memerlukan hasil dari pemanggilan method untuk diolah lebih lanjut. Oleh karena ini, pemanggilan method non void umumnya ditulis sebagai bagian dari statement. Statement berikutnya, statement pada baris 3, mencetak nilai yang disimpan variabel `hasil` ke console.

Mendefinisikan Method

Untuk membuat sebuah method kita harus menuliskan **definisi** method. Definisi method terdiri dari dua bagian: **header** dan **body**. Header dari method, yang dituliskan pada awal dari definisi method terdiri dari hal-hal penting dari method tersebut, termasuk nama method. Body dari method terdiri dari kumpulan statement-statement yang dieksekusi ketika method tersebut dipanggil. Statement-statement pada body method dituliskan di dalam kurung kurawal. Method

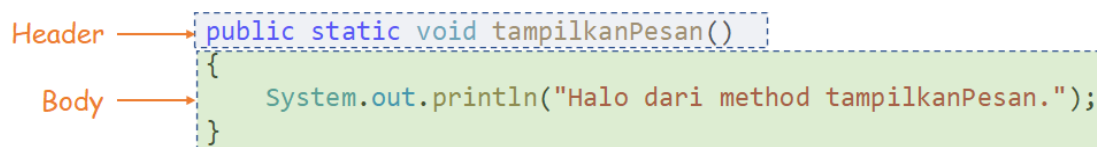
`main` yang kita tuliskan ketika kita membuat sebuah program adalah contoh dari method `void`. Gambar berikut menjelaskan bagian-bagian dari definisi suatu method `main`:



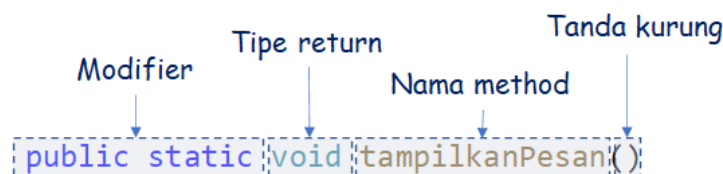
Seperti yang telah kita ketahui, semua program Java yang komplit harus mempunyai method `main`. Program Java dapat juga mempunyai method lain selain method `main`. Berikut adalah contoh penulisan definisi method sederhana yang mencetak sebuah pesan:

```
public static void tampilkanPesan()
{
    System.out.println("Halo dari method tampilkanPesan.");
}
```

Gambar berikut menunjukkan bagian header dan body dari method di atas:



Gambar berikut menjelaskan bagian-bagian dari header method di atas:



Header dari method mengandung bagian-bagian berikut:

- **Modifier** - keyword `public` dan `static` adalah modifier dari method. Kita akan membahas apa yang dimaksud dengan modifier ini nanti saat pembahasan class. Penjelasan singkat dari modifier `public static`: kata `public` berarti method ini tersedia secara public (umum) untuk kode di luar class dan kata `static` berarti method ini adalah bagian dari class, bukan bagian dari object tertentu. Untuk saat ini kita akan selalu menggunakan `public static` untuk semua method kita.
- **Tipe return** - adalah tipe data dari nilai yang dikembalikan oleh method. Pada contoh, kita menuliskan tipe return `void`. Ini berarti method ini adalah method void yang tidak mengembalikan sebuah nilai. Kita akan melihat nanti pada method yang mengembalikan nilai, tipe return ini dapat berupa berbagai tipe-tipe data, seperti `int`, `double`, `String`, dan sebagainya.
- **Nama method** - Kita harus memberikan setiap method dengan nama. Aturan pemberian nama method sama seperti aturan pemberian nama variabel. Umumnya secara konvensi, kita menggunakan *camelcase* yaitu dengan menggunakan huruf besar untuk mengawali kata kedua dan selanjutnya untuk nama method dengan lebih dari satu suku kata.
- **Tanda kurung** - Nama method selalu diikuti dengan tanda kurung. Pada contoh method `tampilkanPesan`, kita mengosongkan tanda kurung ini. Nanti kita akan melihat bahwa method dapat menerima argument-argument. Pada method yang menerima argument-argument kita menuliskan satu atau lebih deklarasi variabel di dalam tanda kurung.

Memanggil Method

Statement-statement di dalam body method dieksekusi ketika method tersebut dipanggil. Method `main` dipanggil secara otomatis oleh JVM ketika program mulai dijalankan. Untuk method-method lainnya, kita harus menuliskan statement yang melakukan pemanggilan method. Berikut adalah statement pemanggilan method `tampilkanPesan`:

```
tampilkanPesan();
```

Untuk memanggil method, kita menuliskan namanya dengan diikuti tanda kurung buka dan tanda kurung tutup.

Program berikut mendemonstrasikan penulisan definisi method `cetakHalo` pada program dan pemanggilan method ini pada method `main`:

Program (MethodSederhana.java)

```
/*
    Program ini mendefinisikan dan memanggil sebuah method sederhana
*/
public class MethodSederhana
{
    public static void main(String[] args)
    {
        System.out.println("Halo dari method main.");
        tampilkanPesan();
        System.out.println("Kembali ke method main.");
    }

    /*
        Method tampilkanPesan mencetak teks sapaan
    */
    public static void tampilkanPesan()
    {
        System.out.println("Halo dari method tampilkanPesan.");
    }
}
```

Output Program (MethodSederhana.java)

```
Halo dari method main.
Halo dari method tampilkanPesan.
Kembali ke method main.
```

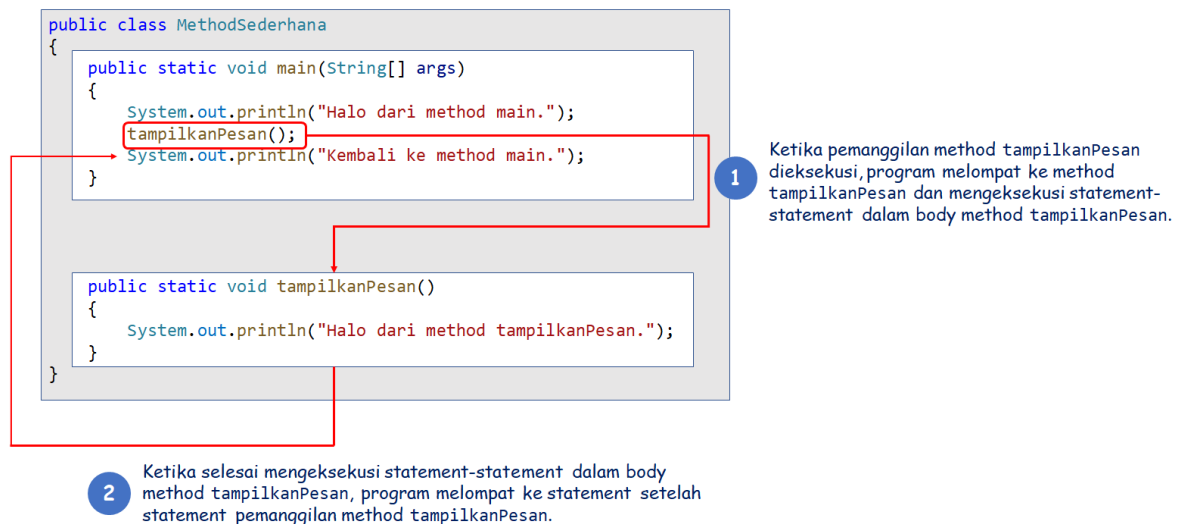
Ketika sebuah method dipanggil, program melompat ke method tersebut dan mengeksekusi statement-statement dalam body method tersebut. Contoh program di atas mempunyai alur program seperti berikut:

- Ketika program dijalankan, program masuk ke method `main` dan memulai mengeksekusi statement pertama dari method ini
- Program mengeksekusi statement pertama dari method `main`, pada baris 8, yang menyebabkan sebuah pesan "Halo dari method main." dicetak ke layar.
- Program mengesekusi statement pada baris 9 yang merupakan statement pemanggilan method `tampilkanPesan`. Pemanggilan method ini menyebabkan program melompat ke

method `tampilkanPesan` dan mengeksekusi statement-statement dalam body method `tampilkanPesan`. Ini menyebabkan program mencetak Halo dari method `tampilkanPesan`.". Ketika program selesai mengeksekusi semua statement dalam body method `tampilkanPesan`, program melompat kembali ke method `main` ke baris 10.

- Program melanjutkan eksekusi statement pada baris 10 yang menyebabkan program mencetak "Kembali ke method main." ke layar.
- Setelah selesai mengeksekusi statement baris 10, karena tidak ada lagi statement yang tersisa dalam body method `main`, maka program selesai dan berhenti.

Gambar berikut mengilustrasikan alur program contoh di atas saat mengeksekusi statement pemanggilan method `tampilkanPesan`:



6.2 Memberikan Argument ke Method

Method dapat menerima nilai-nilai untuk diproses. Nilai-nilai yang kita berikan ke method ini disebut sebagai argument. Kita sebelumnya telah melihat penggunaan argument dalam pemanggilan method. Misalkan, statement:

```
System.out.println("Selamat Datang!");
```

memanggil method `System.out.println` dan memberikan string `"Selamat Datang!"` sebagai argument. Pada penulisan pemanggilan method dengan memberikan argument kita menuliskan nilai argument yang kita berikan ke method tersebut di dalam tanda kurung.

Berikut adalah contoh penulisan definisi method yang menerima argument:

```
public static void cetakNilai(int num)
{
    System.out.println("Nilai yang diberikan adalah " + num);
}
```

Perhatikan pada header definisi method di atas, di dalam tanda kurung kita menuliskan deklarasi variabel berikut: `int num`. Ini adalah deklarasi variabel bernama `num` yang bertipe integer. Variabel `num` yang dideklarasikan pada header dari method `cetakNilai` ini disebut sebagai **variabel parameter** atau sering disingkat sebagai **parameter** saja. Parameter ini berguna untuk menyimpan nilai argument yang diberikan saat pemanggilan method.

Statement berikut adalah pemanggilan method `cetakNilai` dengan memberikan nilai 5 sebagai argument:

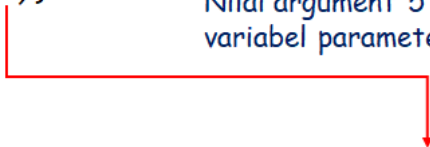
```
cetakNilai(5);
```

Ketika pemanggilan method ini dieksekusi, nilai argument yang dituliskan dalam tanda kurung disalin ke variabel parameter `num`, lalu statement dalam body method `cetakNilai` dieksekusi. Gambar berikut mengilustrasikan ini.

```
cetakNilai(5);
```

Nilai argument 5 disalin ke variabel parameter num.

```
public static void cetakNilai(int num)
{
    System.out.println("Nilai yang diberikan adalah " + num);
}
```



Di dalam method `cetakNilai`, variabel `num` akan menyimpan nilai argument yang diberikan. Jika kita memberikan nilai 5 sebagai argument, maka variabel `num` akan menyimpan nilai 5. Sehingga ketika statement dalam body dieksekusi, pesan berikut akan ditampilkan:

```
Nilai yang diberikan adalah 5
```

Kita juga dapat memberikan isi dari variabel dan nilai dari ekspresi sebagai argument. Sebagai contoh, statement-statement berikut memanggil method `cetakNilai` dengan berbagai macam argument:

```
cetakNilai(x);
cetakNilai(x * 4);
```

Pada statement pertama kita memberikan nilai yang disimpan dalam variabel `x` sebagai argument ke method `cetakNilai`. Pada statement kedua, kita memberikan hasil evaluasi dari ekspresi `x * 4` sebagai argument ke method `cetakNilai`.

Program berikut mendemonstrasikan pemanggilan-pemanggilan method ini.

Program (MemberikanArg.java)

```
/*
    Program ini mendemonstrasikan sebuah method dengan parameter.
*/
public class MemberikanArg
{
    public static void main(String[] args)
    {
        int x = 10;

        System.out.println("Saya memberikan nilai ke method cetakNilai.");
        cetakNilai(5);      // Memberikan argument 5 ke cetakNilai
        cetakNilai(x);      // Memberikan argument 10 ke cetakNilai
        cetakNilai(x * 4);  // Memberikan argument 40 ke cetakNilai
    }
}
```

```

    /*
        Method cetakNilai menampilkan nilai dari
        parameter integernya.
    */
    public static void cetakNilai(int num)
    {
        System.out.println("Nilai yang diberikan adalah " + num);
    }
}

```

Output Program (MemberikanArg.java)

```

Saya memberikan nilai ke method cetakNilai.
Nilai yang diberikan adalah 5
Nilai yang diberikan adalah 10
Nilai yang diberikan adalah 40

```

Hal yang perlu diperhatikan ketika memberikan sebuah variabel sebagai argument adalah hanya tuliskan nama variabel dalam tanda kurung dari pemanggilan method. Jangan menuliskan tipe data dari variabel argument dalam pemanggilan method. Sebagai contoh, statement berikut akan menyebabkan error:

```
cetakNilai(int x);           // ERROR
```

Pemanggilan method yang benar seharusnya seperti berikut:

```
cetakNilai(x);
```

Memberikan Lebih dari Satu Argument

Seringkali method membutuhkan lebih dari satu argument untuk mengerjakan tugasnya. Sebagai contoh, method berikut menerima dua argument:

```

public static void tampilkanJumlah(double num1, double num2)
{
    double sum;    // Untuk menyimpan jumlah

    sum = num1 + num2;
    System.out.println("Jumlahnya adalah " + sum);
}

```

Perhatikan bahwa dua variabel parameter, `num1` dan `num2`, dideklarasikan di dalam tanda kurung dalam header method dan deklarasi-deklarasi tersebut dipisahkan oleh koma. Berikut adalah contoh sebuah statement yang memanggil method di atas:

```
tampilkanJumlah(5, 10);
```

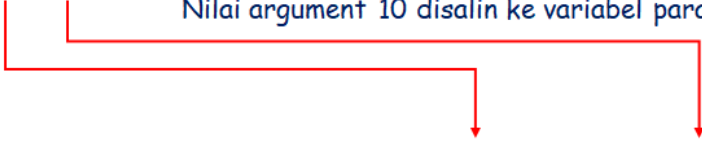
Statement di atas memberikan argument 5 dan 10 ke method `tampilkanNilai`. Kedua nilai yang diberikan ke variabel parameter sesuai dengan urutan yang tertulis dalam pemanggilan method. Dengan kata lain, argument pertama diberikan ke variabel parameter pertama, argument kedua diberikan ke variabel parameter kedua, dan seterusnya. Sehingga, statement di atas memberikan

nilai 5 ke parameter `num1` dan memberikan nilai 10 ke parameter `num2`. Gambar berikut mengilustrasikan pemberian nilai pada pemanggilan method tersebut:

Nilai argument yang diberikan ke method sesuai dengan urutan penulisan variabel parameter pada header dari method.

```
tampilkanJumlah(5, 10);
```

Nilai argument 5 disalin ke variabel parameter num1.
Nilai argument 10 disalin ke variabel parameter num2.



```
public static void tampilkanJumlah(double num1, double num2)
{
    double sum;

    sum = num1 + num2;
    System.out.println("Jumlahnya adalah " + sum);
}
```

Argumen Tipe Primitif Diberikan sebagai Nilainya

Dalam Java, semua argument yang bertipe data primitif diberikan ke method sebagai nilai. Ini berarti hanya salinan dari nilai argument yang diberikan ke variabel parameter. Variabel-variabel parameter dari method adalah variabel terpisah dan berbeda dengan variabel-variabel yang dituliskan di dalam tanda kurung pada pemanggilan method. Jika sebuah method mengubah nilai variabel parameter, perubahan ini tidak mempengaruhi nilai dari variabel argument. Sebagai contoh, perhatikan program berikut:

Program (MemberikanNilai.java)

```
/*
    Program ini mendemonstrasikan bahwa hanya salinan dari nilai argument
    yang diberikan ke method.
*/
public class MemberikanNilai
{
    public static void main(String[] args)
    {
        int angka = 99;    // variabel angka mulai dengan nilai 99

        // Tampilkan nilai dalam angka
        System.out.println("angka menyimpan nilai " + angka);

        // Panggil ubahSaya, memberikan nilai dalam angka
        // sebagai argument.
        ubahSaya(angka);

        // Tampilkan nilai dalam angka kembali.
        System.out.println("angka menyimpan nilai " + angka);
    }

    /*
        Method ubahSaya menerima sebuah argument dan lalu
        mengubah nilai dari parameter.
    */
}
```



```

public static void ubahSaya(int nilaiKu)
{
    System.out.println("Saya mengubah nilai parameter.");

    // Ubah nilai parameter nilaiKu ke 0.
    nilaiKu = 0;

    // Tampilkan nilai dalam nilaiKu
    System.out.println("Sekarang nilai parameter adalah " + nilaiKu);
}
}

```

Output Program (MemberikanNilai.java)

```

angka menyimpan nilai 99
Saya mengubah nilai parameter.
Sekarang nilai parameter adalah 0
angka menyimpan nilai 99

```

Pada program di atas, meskipun variabel parameter `nilaiKu` diubah nilainya dalam method `ubahSaya`, argument `angka` tidak berubah nilainya. Ini karena variabel `nilaiKu` hanya menyimpan salinan nilai dari variabel `angka`.

Memberikan Argument berupa Object ke Method

Sejauh ini kita telah melihat method-method yang menerima nilai-nilai bertipe data primitif sebagai argument. Kita dapat juga menulis method yang menerima referensi ke object sebagai argument. Sebagai contoh, perhatikan method berikut:

```

public static void tampilkanPanjang(String str)
{
    System.out.println(str + " mempunyai panjang " + str.length() +
        " karakter.");
}

```

Method ini menerima sebuah referensi ke sebuah object `String` sebagai argument-nya, dan menampilkan pesan ke console berapa banyak karakter dalam object `String` tersebut. Kode berikut mencontohkan bagaimana memanggil method di atas:

```

String nama = "Herman";
tampilkanPanjang(nama);

```

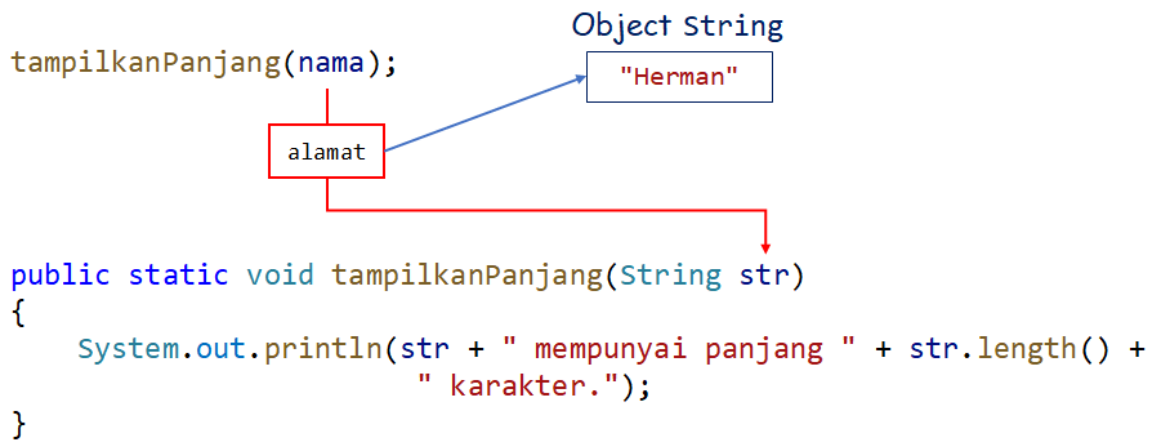
Ketika kode di atas dieksekusi, method `tampilkanPanjang` akan menampilkan pesan berikut:

```

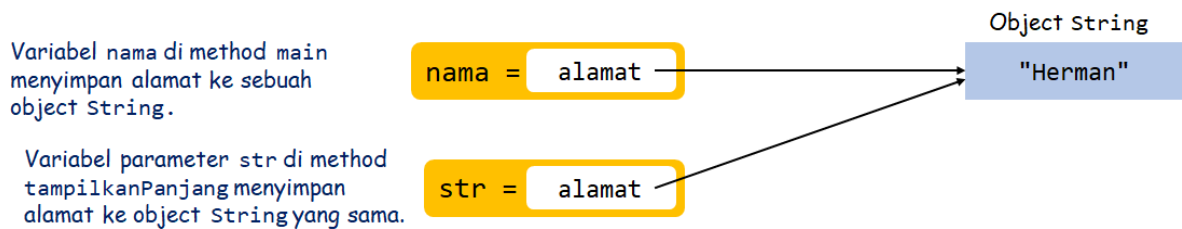
Herman mempunyai panjang 6 karakter.

```

Ketika sebuah object, seperti sebuah `String` diberikan sebagai argument ke suatu method, bukan nilai object tersebut yang diberikan, namun referensi (alamat memori) ke object tersebut yang diberikan. Gambar berikut mengilustrasikan ini:



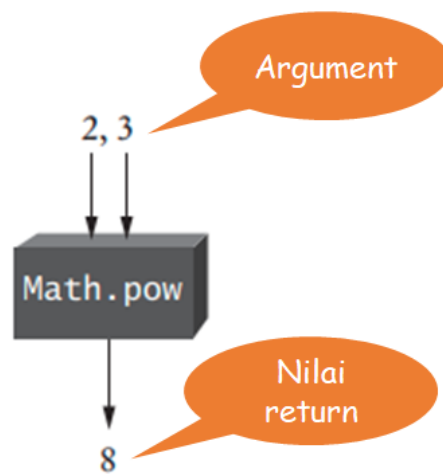
Sehingga saat method `tampilkanPanjang` dieksekusi, variabel `str` dan variabel `nama` mereferensikan object yang sama. Gambar berikut mengilustrasikan ini:



6.3 Nilai Return

Kita telah melihat data dapat diberikan ke sebuah method melalui variabel parameter. Data dapat juga dikembalikan dari method ke statement yang memanggil method tersebut. Method yang mengembalikan sebuah nilai disebut sebagai method non-void.

Salah satu contoh method non-void adalah method `Math.pow` yang telah tersedia dalam Java. Saat kita menggunakan method `Math.pow` ataupun method-method lain yang sudah ditulis oleh programmer lain, kita dapat membayangkan method-method tersebut sebagai “kotak hitam”. Kita tidak perlu mengetahui bagaimana method-method tersebut melakukan proses komputasi. Kita hanya perlu mengetahui spesifikasi dari method tersebut. Misalkan spesifikasi dari method `Math.pow` adalah: Jika kita memberikan argument `xx` dan `yy` yang berupa nilai numerik, maka method `Math.pow` akan memberikan nilai `xyxy`. Ketika kita memanggil method `Math.pow` dengan memberikan dua nilai argument 2 dan 3 seperti berikut: `Math.pow(2, 3)`. Nilai-nilai argument ini dapat kita bayangkan seperti “input” ke method tersebut. Lalu, method tersebut melakukan tugasnya dan memberikan sebuah nilai “output” 8. Nilai “output” dari pemanggilan method ini disebut sebagai **nilai return** (nilai kembali). Gambar berikut mengilustrasikan method `Math.pow` sebagai “kotak hitam”:



Mendefinisikan Method Non-void

Ketika kita menuliskan method non-void yang mengembalikan sebuah nilai, kita harus menentukan tipe data dari nilai return dari method tersebut. Pada method `void` kita menuliskan keyword `void` pada bagian nilai return di header dari method. Pada method non-void kita menggunakan nama tipe data seperti `int`, `double`, `boolean`, atau tipe-tipe data lainnya, pada bagian tipe return di header dari method. Sebagai contoh, method berikut mengembalikan nilai `int`:

```
public static int jumlah(int num1, int num2)
{
    int hasil;

    hasil = num1 + num2;
    return hasil;
}
```

Nama dari method di atas adalah `jumlah` dan tipe return method tersebut adalah `int`. Gambar berikut menjelaskan bagain tipe return dari method ini:

Tipe return
↓
`public static int jumlah(int num1, int num2)`

Selain menentukan tipe return berdasarkan tipe dari nilai return pada header dari method non-void, pada body dari method non-void kita juga harus menuliskan statement `return`. Statement `return` menghentikan eksekusi method dan mengembalikan sebuah nilai ke statement yang memanggil method tersebut. Pada contoh di atas, statement:

```
return hasil;
```

menghentikan eksekusi method `jumlah` dan mengembalikan nilai yang disimpan oleh variabel `hasil`.

Bentuk umum dari syntax statement `return` adalah seperti berikut:

```
return Ekspresi
```

`Ekspresi` adalah nilai yang dikembalikan yang dapat ekspresi apa saja yang mempunyai sebuah nilai seperti variabel, literal, atau ekspresi aritmatika. Pada contoh di atas, kita mengembalikan nilai yang disimpan dalam variabel `hasil`. Namun, kita dapat juga menghilangkan variabel `result` dan mengembalikan ekspresi `num1 + num2`, seperti terlihat pada kode berikut:

```
public static int jumlah(int num1, int num2)
{
    return num1 + num2;
}
```

Memanggil Method Non-void

Program berikut mencontohkan cara memanggil method `jumlah`:

Program (MengembalikanNilai.java)

```
import java.util.Scanner;

/*
    Program ini mendemonstrasikan method non-void
*/
public class MengembalikanNilai
{
    public static void main(String[] args)
    {
        int angka1, angka2, total;

        Scanner keyboard = new Scanner(System.in);

        // Minta angka 1 ke pengguna
        System.out.print("Masukkan angka 1: ");
        angka1 = keyboard.nextInt();

        // Minta angka 2 ke pengguna
        System.out.print("Masukkan angka 2: ");
        angka2 = keyboard.nextInt();

        // Panggil method jumlah untuk mendapat jumlah angka1
        // dan angka2
        total = jumlah(angka1, angka2);

        // Tampilkan hasil
        System.out.println("Jumlah " + angka1 + " dan " + angka2 +
            " adalah " + total);
    }

    /*
        Method jumlah mengembalikan jumlah dari dua parameternya.
    */
    public static int jumlah(int num1, int num2)
    {
        int hasil;

        hasil = num1 + num2;
        return hasil;
    }
}
```

```
}
```

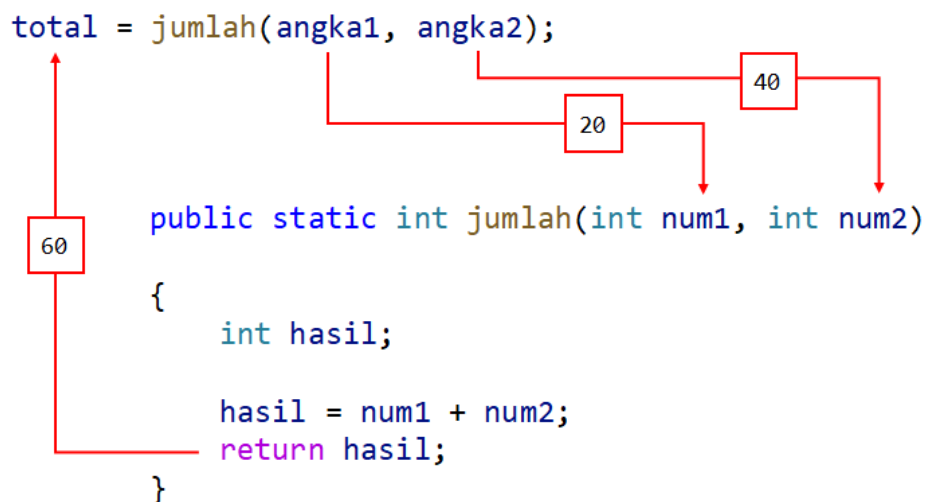
Output Program (MengembalikanNilai.java)

```
Masukkan angka 1: 20
Masukkan angka 2: 40
Jumlah 20 dan 40 adalah 60
```

Dalam program `MengembalikanNilai.java` di atas, statement pada baris 14:

```
total = jumlah(angka1, angka2);
```

adalah statement yang memanggil method `jumlah` dengan memberikan argument `angka1` dan `angka2` dan menugaskan nilai kembali dari method `jumlah` ke variabel `total`. Pada contoh output, pengguna memasukkan nilai 20 ke angka 1 dan nilai 40 ke angka 2, sehingga pemanggilan method ini mengembalikan nilai 60. Gambar berikut mengilustrasikan bagaimana argument diberikan ke method dan bagaimana nilai dikembalikan dari method:



Ketika kita memanggil method non-void, umumnya kita ingin melakukan hal lebih lanjut terhadap nilai return yang dikembalikan. Pada contoh program di atas, kita menugaskan hasilnya ke sebuah variabel. Penugasan nilai kembali ke sebuah variabel seperti contoh tersebut adalah hal yang umumnya dilakukan terhadap nilai return. Namun, hal tersebut bukan satu-satunya hal yang dapat kita lakukan terhadap nilai return, kita dapat melakukan hal-hal lain. Sebagai contoh, kode berikut menunjukkan sebuah ekspresi matematika yang menggunakan pemanggilan method `jumlah`:

```
int x = 10, y = 15;
double rerata;
rerata = jumlah(x, y) / 2.0;
```

Pada statement terakhir dari kode di atas, method `jumlah` dipanggil dengan argument `x` dan `y`. Pemanggilan method ini mengembalikan nilai return, 25, yang kemudian dibagi dengan 2.0. Hasil pembagian, 12.5, kemudian ditugaskan ke variabel `rerata`.

Contoh lain dari penggunaan nilai return dari pemanggilan method adalah menggunakannya sebagai argument ke pemanggilan method lain. Kode berikut mencontohkannya:

```
int x = 10, y = 15;
System.out.println("Jumlahnya adalah " + jumlah(x, y));
```

Statement terakhir, memberikan nilai return dari pemanggilan method `jumlah(x, y)` sebagai argument ke pemanggilan method `System.out.println`. Sehingga, pesan "Jumlahnya adalah 25" akan ditampilkan ke layar.

Mengembalikan Nilai `boolean`

Seringkali kita membutuhkan sebuah method yang menguji sebuah argument dan mengembalikan nilai `true` atau `false` yang mengindikasikan apakah suatu kondisi terpenuhi atau tidak. Method seperti ini mengembalikan sebuah nilai `boolean`. Sebagai contoh, method berikut menerima sebuah argument dan mengembalikan `true` jika argument tersebut berada dalam rentang 1 sampai dengan 100, dan mengembalikan `false` jika tidak:

```
public static boolean isValid(int number)
{
    boolean status;

    if (number >= 1 && number <= 100)
    {
        status = true;
    }
    else
    {
        status = false;
    }

    return status;
}
```

Kode berikut mencontohkan statement `if-else` yang memanggil method di atas:

```
int value = 20;
if (isValid(value))
{
    System.out.println("Nilai di dalam rentang 1 s.d 100.");
}
else
{
    System.out.println("Nilai di luar rentang 1 s.d 100.");
}
```

Ketika kode di atas dieksekusi, pesan "Nilai di dalam rentang 1 s.d 100." akan ditampilkan.

Mengembalikan Referensi ke sebuah Object

Method dapat juga mengembalikan sebuah referensi ke tipe non primitif, seperti object `String`. Program berikut mendemonstrasikan ini:

Program (MengembalikanString.java)

```
/*
    Program ini mendemonstrasikan sebuah method yang mengembalikan
    referensi ke object String.
*/
public class MengembalikanString
{
    public static void main(String[] args)
    {
        String namaCustomer;

        namaCustomer = namaLengkap("Budi", "Susilo");
        System.out.println(namaCustomer);
    }

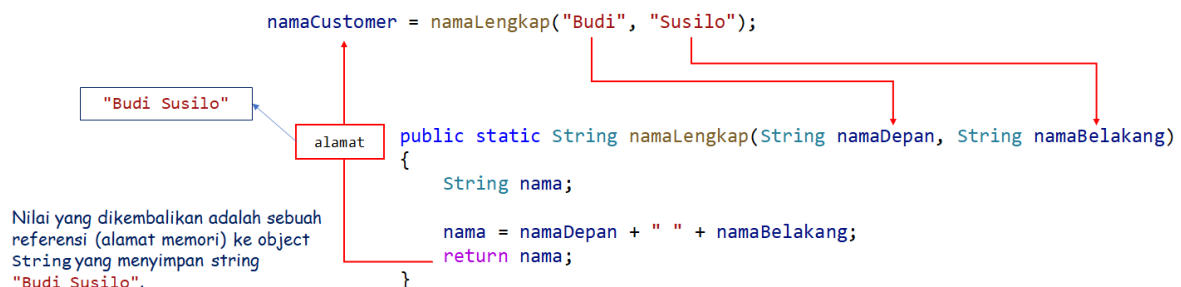
    /*
        Method namaLengkap menerima dua argument String
        berisi nama depan dan nama belakang. Method ini mengkonkatenasi
        keduanya menjadi sebuah object String.
    */
    public static String namaLengkap(String namaDepan, String namaBelakang)
    {
        String nama;

        nama = namaDepan + " " + namaBelakang;
        return nama;
    }
}
```

Output Program (MengembalikanString.java)

Budi Susilo

Baris 11 pada program di atas memanggil method `namaLengkap` dengan memberikan argument pertama `"Budi"` dan argument kedua `"Susilo"`. Pemanggilan method ini mengembalikan sebuah referensi ke sebuah object `String` yang berisi `"Budi Susilo"`. Referensi tersebut kemudian ditugaskan ke variabel `namaCustomer`. Proses ini diilustrasikan pada gambar berikut:



6.4 Lingkup Variabel

Lingkup (*scope*) variabel adalah bagian di dalam program dimana variabel dapat diakses atau digunakan. Sebagai contoh, lingkup dari variabel parameter dari sebuah method adalah keseluruhan body dari method tersebut. Ini berarti kita dapat menggunakan variabel parameter di dalam body method variabel parameter tersebut didefinisikan namun kita tidak dapat menggunakan variabel parameter tersebut pada method lain. Perhatikan potongan program berikut:

```
public static int keliling(int sisi)
{
    return 4 * sisi;
}

public static int luas()
{
    return sisi * sisi;    // ERROR
}
```

Statement baris 8 pada potongan program di atas akan menghasilkan error. Ini karena, statement tersebut yang berada di dalam method `luas` mencoba mengakses variabel parameter `sisi` dari method `keliling`.

Kita dapat memperbaiki potongan program di atas menjadi seperti berikut:

```
public static int keliling(int sisi)
{
    return 4 * sisi;
}

public static int luas(int sisi)
{
    return sisi * sisi;
}
```

Pada method `keliling` dan method `luas`, kita menggunakan variabel parameter dengan nama yang sama yaitu `sisi`. Ini tidak menghasilkan error, karena lingkup dari variabel parameter adalah body dari method yang mendeklarasikannya. Sehingga, meskipun memiliki nama yang sama, variabel parameter `sisi` pada method `keliling` adalah variabel yang berbeda dari variabel parameter `sisi` pada method `luas`.

Variabel Lokal

Variabel-variabel yang dideklarasikan di dalam sebuah method disebut sebagai variabel lokal dari method tersebut. Lingkup dari variabel lokal adalah di method tempat variabel tersebut dideklarasikan. Ini berarti, variabel lokal hanya dapat diakses di dalam body method tempat variabel tersebut dideklarasikan dan tidak dapat diakses oleh statement-statement di luar method tersebut. Method-method yang berbeda dapat mempunyai variabel-variabel lokal dengan nama yang sama karena method-method tersebut tidak dapat melihat variabel-variabel lokal dari method-method yang lain. Sebagai contoh, program berikut menghasilkan error karena statement dalam sebuah method mencoba mengakses variabel lokal dari method lain:


```

import java.util.Scanner;

public class VariabelLokal
{
    public static void main(String[] args)
    {
        promptPengguna();
        hitungLuas();
    }

    public static void promptPengguna()
    {
        int sisi;
        Scanner keyboard = new Scanner(System.in);

        System.out.print("Masukkan panjang sisi: ");
        sisi = keyboard.nextInt();
    }

    public static int hitungLuas()
    {
        int luas;
        luas = sisi * sisi; // ERROR
        System.out.println("Luas = " + luas);
    }
}

```

Error pada program di atas terjadi karena pada baris 23, statement yang berada di dalam method `hitungLuas` mencoba mengakses variabel `sisi` yang merupakan variabel lokal dari method `promptPengguna`.

Jangka Hidup Variabel Lokal dan Variabel Parameter

Jangka hidup variabel adalah jangka waktu variabel tersebut tersimpan dalam memori. Variabel lokal dan juga variabel parameter mempunyai jangka hidup selama method tersebut dipanggil. Setelah method selesai mengeksekusi semua statement-nya, variabel lokal dan variabel parameter dihapus dari memori.

Misalkan kita mempunyai program seperti berikut:

```

public class Menjumlahkan
{
    public static void main(String[] args)
    {
        int total;

        total = jumlah(15, 43);

        // Tampilkan hasil
        System.out.println("15 + 43 = " + total);
    }

    public static int jumlah(int num1, int num2)
    {

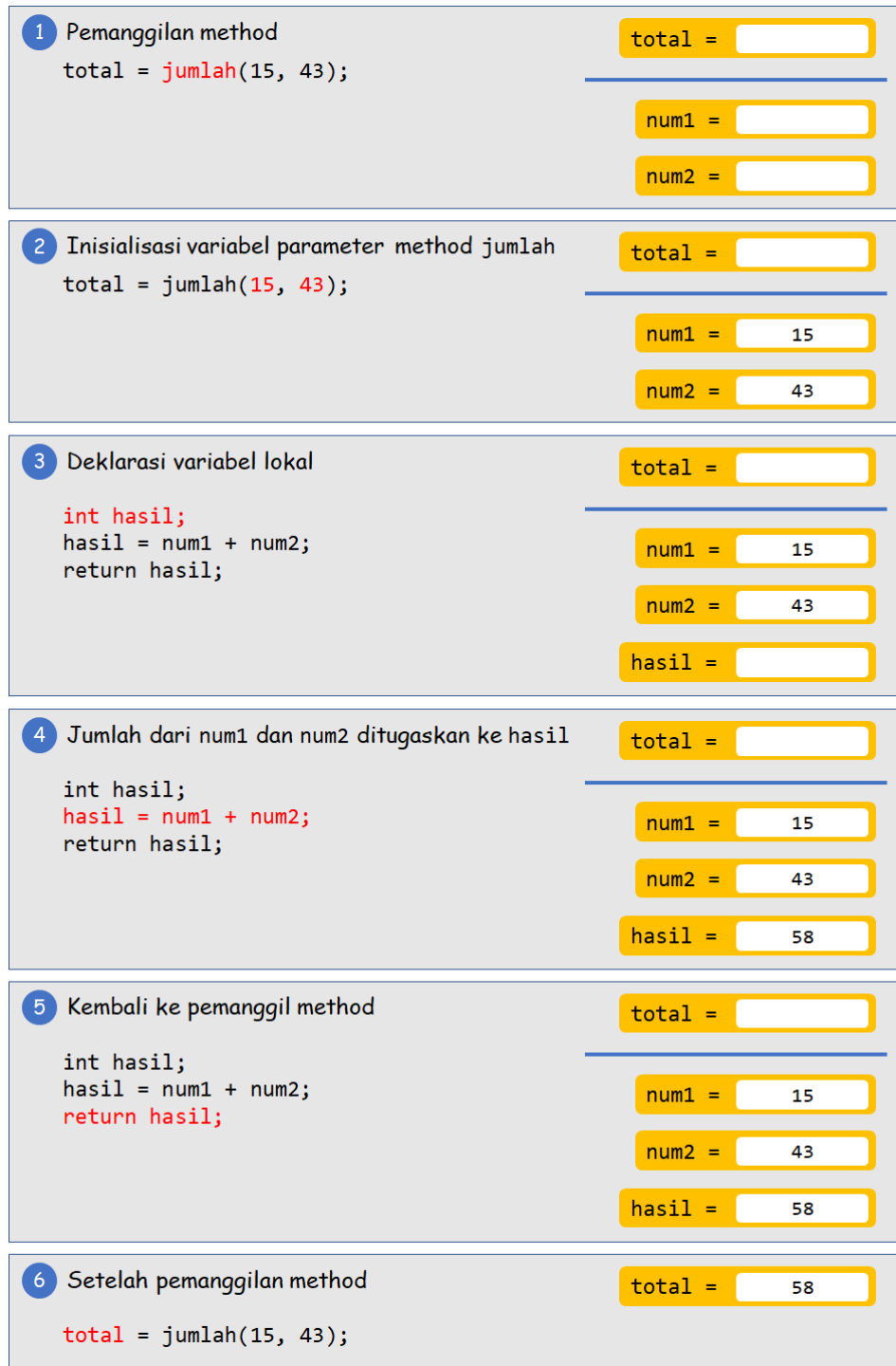
```

```

    int hasil;
    hasil = num1 + num2;
    return hasil;
}
}

```

Jangka hidup dari variabel parameter, `num1` dan `num2`, dan variabel lokal, `hasil`, dari method `jumlah` saat method tersebut dipanggil diilustrasikan oleh gambar berikut:



Penjelasan dari gambar di atas:

1. Saat pemanggilan method `jumlah`, variabel parameter `num1` dan `num2` diciptakan di dalam memori.
2. Nilai-nilai argument ditugaskan ke masing-masing variabel parameter.
3. Variabel lokal `hasil` dari method `jumlah` diciptakan dalam memori.
4. Jumlah dari `num1` dan `num2` ditugaskan ke variabel lokal `hasil`.

5. Statement `return` menghentikan eksekusi method dan memberikan nilai dari variabel lokal `hasil` ke pemanggil.
6. Saat kembali ke pemanggil, semua variabel parameter dan variabel lokal dihapus dalam memori. Nilai yang dikembalikan dari method `jumlah` ditugaskan ke variabel `total`.

6.5 Penggunaan Ulang Method

Ketika menulis sebuah program yang besar, penggunaan method dapat menyingkat waktu pembuatan program. Misalkan kita membuat sebuah program yang mempunyai kode-kode yang identik dituliskan berulang kali. Kita dapat menuliskan sebuah method untuk kode-kode identik tersebut. Sebagai contoh misalkan kita mempunyai program dengan kode-kode berulang seperti berikut:

```
import java.util.Scanner;

/*
    Program berikut mencontohkan penulisan kode berulang
    yang tidak efisien.
*/
public class KodeBerulang
{
    public static void main(String[] args)
    {
        int jam, menit;

        Scanner keyboard = new Scanner(System.in);

        // Meminta pengguna untuk memasukkan jam
        // di rentang yang valid.
        do
        {
            System.out.print("Masukkan nilai antara 0 dan 23: ");
            jam = keyboard.nextInt();
        }
        while (jam < 0 || jam > 23);

        // Meminta pengguna untuk memasukkan menit
        // di rentang yang valid.
        do
        {
            System.out.print("Masukkan nilai antara 0 dan 59: ");
            menit = keyboard.nextInt();
        }
        while (menit < 0 || menit > 59);

        // Tampilkan waktu yang diinput.
        System.out.println("waktu yang diinput:");
        System.out.println(jam + ":" + menit);
    }
}
```

Pada program di atas kita menuliskan dua statement `do-while` pada baris 17 sampai dengan baris 22 dan pada baris 26 sampai dengan 31. Kedua statement `do-while` ini mirip, keduanya membaca angka yang dari pengguna dan memastikan angka yang dimasukkan dalam rentang tertentu. Perbedaan dari kedua statement `do-while` ini adalah pada statement `do-while` pertama angka yang diminta berada dalam rentang 0 sampai dengan 23 sedangkan pada statement `do-while` kedua angka yang diminta berada dalam rentang 0 sampai dengan 59. Kita dapat membuat sebuah method sehingga kita tidak perlu menulis ulang statement `do-while` ini.

Program berikut bekerja sama seperti program di atas, namun pada program ini kita menghilangkan penulisan ulang kode dengan membuat sebuah method yang digunakan untuk meminta angka ke pengguna dalam rentang tertentu:

Program (MethodReusable.java)

```
import java.util.Scanner;

/*
    Program berikut mencontohkan penggunaan method
    untuk menggantikan kode berulang sehingga menyingkat
    waktu penulisan program dan menjadikan program lebih
    mudah dibaca.
*/
public class MethodReusable
{
    public static void main(String[] args)
    {
        int jam, menit;
        jam = bacaIntSampaiDengan(23);
        menit = bacaIntSampaiDengan(59);

        // Tampilkan waktu yang diinput.
        System.out.println("waktu yang diinput:");
        System.out.println(jam + ":" + menit);
    }

    /*
        Prompt pengguna untuk memasukkan angka di bawah angka tertinggi
        yang diberikan. Ulangi prompt jika pengguna memasukkan angka yang tidak
        valid.
        Parameter: tertinggi adalah nilai tertinggi input yang diperbolehkan
        Return: nilai yang diberikan pengguna (antara 0 sampai dengan tertinggi)
    */
    public static int bacaIntSampaiDengan(int tertinggi)
    {
        int input;
        Scanner keyboard = new Scanner(System.in);
        do
        {
            System.out.print("Masukkan nilai antara 0 dan " + tertinggi + ": ");
            input = keyboard.nextInt();
        }
        while (input < 0 || input > tertinggi);

        return input;
    }
}
```

Output Program (MethodReusable.java)

```
Masukkan nilai antara 0 dan 23: 18  
Masukkan nilai antara 0 dan 59: 35  
waktu yang diinput:  
18:35
```

REFERENSI

- [1] Horstmann, Cay S. 2012. *Big Java: Late Objects, 1st Edition*. United States of America: John Wiley & Sons, Inc.
- [2] Gaddis, Tony. 2016. *Starting Out with Java: From Control Structures through Objects (6th Edition)*. Boston: Pearson.