# BAB 9

# Bekerja dengan Java Class Library

## 9.1 Tujuan

Pada sesi ini, kita akan mengantarkan beberapa konsep dasar dari Object-Oriented objcts, dan Programming (OOP). Selanjutnya kita akan membahas konsep dari classes dan bagaimana menggunakan class dan anggotanya. Perubahan dan pemilihan object juga akan dibahas. Sekarang, kita akan focus dalam menggunakan class yang telah dijabarkan dalam Java Class library, kita akan membahas nanti tentang bagaimana membikin class anda sendiri.

Pada akhir pelajaran, siswa seharusnya dapat :
1. menjelaskan OOP dan beberapa konsepnya
2. perbedaan antara class dan object
3. pebedaan antara instance variables/method dan class (static) variable/method
4. menjelaskan method apa dan bagaimana memanggil method parameter
5. mengidentifikasi beberapa jangkauan dari sebuah variable
6. memilih tipe data primitive dan object
7. membandingkan objects dan menjabarkan class dari objects.

## 9.2 Pengenalan Pemrograman Berorientasi Object

OOP berputar pada konsep dari object sebagai dasar element dari program anda. Ketika kita membandingkan dengan dunia nyata, kita dapat menemukan beberapa objek disekitar kita, seperti mobil, singa, manusia dan seterusnya. Object ini dikarakterisasi oleh sifat / attributnya dan tingkah lakunya.

Contohnya, objek sebuah mobil mempunyai sifat tipe transmisi, warna dan manufaktur. Mempunyai kelakuan berbelok, mengerem dan berakselerasi. Dengan cara yang sama pula kita dapat mendefinisikan perbedaan sifat dan tingkah laku dari singa. Coba perhatikan table dibawah ini sebagai contoh perbandingan :

| ***Object*** | ***Properties*** | ***Behavior*** |
|---|---|---|
| Car | type of transmission<br>manufacturer<br>color | turning<br>braking<br>accelerating |
| Lion | Weight<br>Color<br>hungry or not hungry<br>tamed or wild | roaring<br>sleeping<br>hunting |

*Table 1: Example of Real-life Objects*

Dengan deskripsi ini, objek pada dunia nyata dapat secara mudah dimodelisasi sebagai objek software menggunakan sifat sebagai data dan tingkah laku sebagai method. Data disini dan method dapat digunakan dalam pemrograman game atausoftware interaktif untuk menstimulasi objek dunia nyata. Contohnya adalah sebagai software objek mobil dalam permainan balap mobil atau software objek singadalam sebuah software pendidikan interaktif pada kebun binatang untuk anak anak.

# 9.3 Class dan Object

## *9.3.1 Perbedaan Class dan Object*

Pada dunia software, sebuah objek adalah sebuah komponen software yang stukturnya mirip dengan objek pada dunia nyata. Setiap objek dibuat dari satu set data (sifat) dimana variable menjabarkan esensial karakter dari objek, dan juga terdiri dari satu set dari methode (tingkah laku) yang menjabarkan bagaimana tingkah laku dari objek. Jadi objek adalah sebuah berkas software dari variable dan method yg berhubungan. Variable dan methods dalam objek Java scara formal diketahui sebagai instance variable dan instance methods untuk membedakannya dari variable klas dan method klas, dimana akan dibahas kemudian.

Klas adalah sturktur dasa dari OOP. Dia terdiri dari dua tipe dari anggota dimana disebut dengan field (attribut/properti) dan method. Field mespesifikasi tipe data yang didefinisikan oleh class, sementara methode spesifikasi dari operasi. Sebuah objek adalah sebuah instance pada class.

Untuk dapat membedakanantara class dan obect, mari kita mendiskusikan beberapa contoh. Apa yang kita miliki disini adalah sebuah class mobil dimana dapat digunakan untuk medefinisikan beberapa object mobil. Pada table dibawah, mobil A dan mobil B adalah objek dari kelas mobil. Kelas memiliki field plat nomer, warna, manufaktur, dan kecepatan yang diisi dengan nilai korespondendi pada objek mobil A dan mobil B. mobil juga dapat berakselerasi, berbelok dan mengerem.

| Car Class | | Object Car A | Object Car B |
|---|---|---|---|
| *Instance Variables* | Plate Number | ABC 111 | XYZ 123 |
| | Color | Blue | Red |
| | Manufacturer | Mitsubishi | Toyota |
| | Current Speed | 50 km/h | 100 km/h |
| *Instance Methods* | Accelerate Method | | |
| | Turn Method | | |
| | Brake Method | | |

*Table 2: Contoh class car dan object-object nya*

Ketika diinisialisi, tiap objek mendapat satu set baru dari state variable. Bagaimanapun, implementasi dari method dibagi diantara objek pada kelas yang sama.

Kelas menyediakan keuntungan dari reusability. Software programmers dapat digunakan dari sebuah kelas lagi dan lagi untuk membuat beberapa objek.

### *9.3.2 Instansiasi Class*

Untuk membuat sebuah objek atau sebuah instance pada sebuah kelas. Kita menggunakan operator baru. Sebagai contoh, jika anda ingin membuat instance dari kelas string, kita menggunakan kode berikut :

```
String  str2 = new String(“Hello world!”);
```

or also equivalent to,

```
String str2 = "Hello";
```

*Figure 1: Classs Instantiation*

### *9.3.3 Variabel Class dan Method*

Sebagai tambahan pada contoh variable, hal ini juga memungkinkan untuk mendefinisikan variable kelas, dimana variable milik dari seluruh kelas. Ini berarti bahwa memiliki nilai yang sama untuk semua objek pada kelas yang sama. Mereka juga disebut static member variables.

## 9.4 Method

### *9.4.1 Apakah Method itu dan mengapa menggunakan Method?*

Pada contoh yang telah kita diskusikan sebelumnya, kita hanya memiliki satu method, dan itu adalah main() method. Didalam Java, kita dapat mendefinisikan beberapa method yang akan kita panggil dari method yang berbeda.

Sebuah method adalah bagian terpisah dari kode yang akan dipanggil oleh program utama dan beberapa method lainnya untuk menunjukkan beberapa fungsi spesifik.

Berikut adalah karakteristik dari method :

1. dapat mengembalikan satu atau tidak ada nilai
2. dia mungkin dapat diterima sebagai beberapa parameter yang dibutuhkan atau tidak ada arameter sama sekali. Parameter juga disebut sebagai fungsi argument
3. setelah method telah selesai dieksekusi, dia akan kembali pada method yang memanggilnya.

Sekarang mengapa kita butuh untuk membuat method? Mengapa kita tidak meletakkan semua kode pada sebuah method yang sangat besar? Pemecahan masalah disini alah dekomposisi. Kita juga dapat melakukan ini di Java dengan mmbuat method untuk mengatasi bagian spesifik dari masalah. Mengambil sebuah permasalahan dan memecahkannya menjadi bagian kecil, bagian dapat diatur adalah penting untuk menulis program yang besar.

### *9.4.2 Memanggil Instance dari Method dan Passing Variabel*

Sekarang kita ilustrasikan bagaimana memanggil method, mari kita menggunakan kelas string sebagai contoh. Anda dapat menggunakan the Java API documentation untuk melihat semua method dalam kelas string yang tersedia. Selanjutnya, kita akan membuat method kita sendiri. Tapi sekarang mari kita menggunakan apa yang tersedia.

Untuk memanggil sebuah instance method, kita menuliskan :

```
nameOfObject.nameOfMethod( parameters );
```

mari kita mengambil dua contoh yang ditemukan dalam kelas String.

| **Method declaration** | **Definition** |
|---|---|
| **public char charAt(int index)** | Mengambil karakter pada index. |
| **public boolean equalsIgnoreCase (String anotherString)** | Membandingkan antar String, tidak case sensitive. |

*Table 3: Method dari Class String*

Menggunakan method :

```
String    str1 = "Hello";
char      x = str2.charAt(0); //will return the
character H
               //simpan pada variabel x

String    str2 = "hello";

//return boolean
boolean  result = str1.equalsIgnoreCase( str1 );
```

## *9.4.3 Passing Variabel Dalam Method*

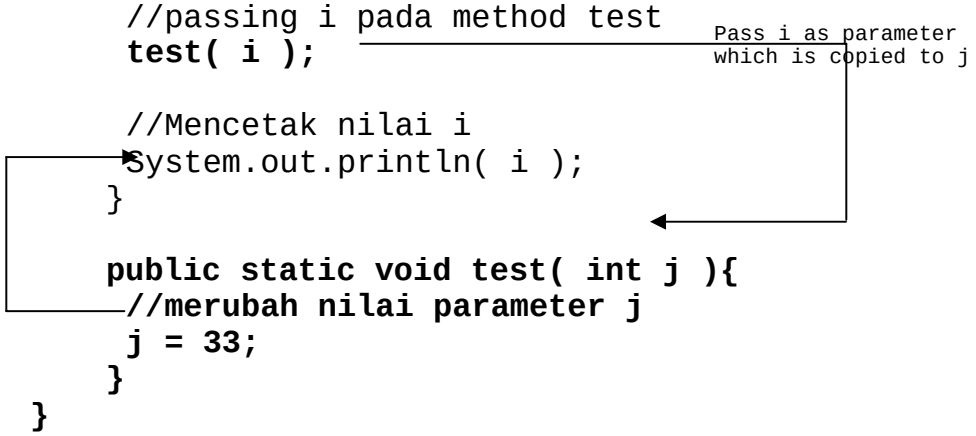
Pada contoh kita, kita telah mecoba melewati variable pada method. Bagaimanapun juga kita tidak dapat membedakan antara perbedaan tipe variabel passing dalam Java. Ada dua tipe data passing pada method, yang pertama adalah pass-by-value dan yang kedua adalah pass-by-reference.

### 9.4.3.1 Pass-by-Value

Ketika pass-by-values terjadi, method menggunakan sebuah copy pada nilai pada variable yang dilewatkan pada method. method tidak dapat secara langsung dimodifikasi secara argument langsung meskipun jika dimodifikasi parameternya selama perhitungan berlangsung.

Contoh :

```
public class TestPassByValue
{
    public static void main( String[] args ){
     int i = 10;
     //mencetak nilai i
     System.out.println( i );

     //memanggil method test
     //passing i pada method test
     test( i );

     //Mencetak nilai i
     System.out.println( i );
    }

    public static void test( int j ){
     //merubah nilai parameter j
     j = 33;
    }
}
```




Pada contoh diatas yang telah diberikan, kita memanggil method tes dan melewatkan nilai i sebagai parameter. Nilai pada i dikopikan pada variable pada method j. sejak j adalah variable pengganti pada method tes, dia tidak akan berdampak pada nilai variable jika i pada main semenjak memiliki perbedaan kopy pada variable.

Secara default, semua tipe data primitive ketika dilewatkan pada sebuah method adalah pass-by-values

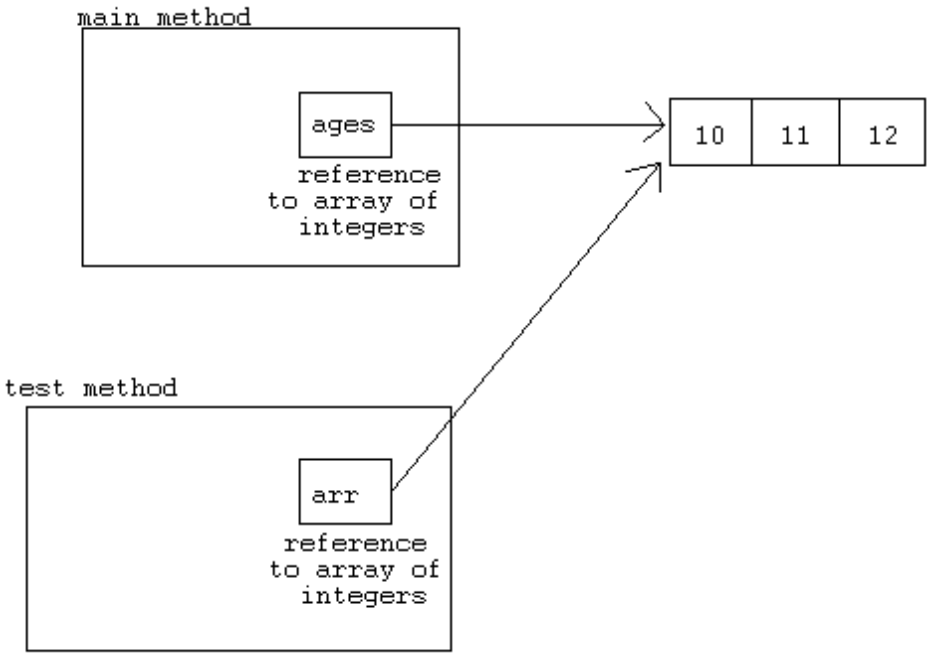
### 9.4.3.2 Pass-by-reference

Ketika sebuah pass-by-reference terjadi, referensi pada sebuah objek dilewatkan dengan cara memanggil method. Hal ini berarti bahwa method mengkopi referensi pada variable yang dilewatkan pada method. Bagaimanapun juga, tidak seperti apda pass-by-value, method dapat membuat objek actual yang menerangkan pointing to, since, meskipun berbeda keterangan yang digunakan dalam method, lokasi dari data yang mereka tunjukkan adalah sama.

contoh :

```
class TestPassByReference
{
     public static void main( String[] args ){
          //membuat array integer
          int []ages     = {10, 11, 12};

          //mencetak nilai array
          for( int i=0; i<ages.length; i++ ){
               System.out.println( ages[i] );
          }

          test( ages );

          for( int i=0; i<ages.length; i++ ){
               System.out.println( ages[i] );
          }
     }

     public static void test( int[] arr ){
          //merubah nilai array
          for( int i=0; i<arr.length; i++ ){
               arr[i] = i + 50;
          }
     }
}
```

Pass ages as parameter which is copied to variable arr




*Gambar 2 : Contoh Pass By Reference*

***Petunjuk Penulisan Program :***

*Keadaan yang salah tentang nilai oleh referensi di java adalah ketika membuat method swap menggunakan referensi Java, mencatat tentang manipulasi object Java 'by reference' tetapi nilai objeck dari referensi dari method 'by value,'" adalah hasil, anda tidak dapat menulis standart swap method ke swap objek.*

### *9.4.4 Memanggil Method Static*

method Static adalah cara yang dapat dipakai tanpa inisialisasi suatu class (maksudnya tanpa menggunakan kata kunci yang baru ), method static mempunyai class yang lengkap dan contoh yang tidak pasti (atau objek) dari suatu class. method static dibedakan dari contoh method di dalam suatu class oleh kata kunci static.

Untuk memanggil method static, ketik,

```
Classname.staticMethodName(params);
```

Contoh dari static method yang digunakan :

```
//mencetak data pada layar
System.out.println(“Hello world”);

//convert string menjadi integer
int i = Integer.parseInt(“10”);


String hexEquivalent = Integer.toHexString( 10 );
```

### *9.4.5 Lingkup Variabel*

Sebagai tambahan dari suatu variable nama dan tipe data, suatu variable mempunyai jangkauan, Jangkauan menentukan dimana program dapat mengakses variable, jangkauan juga menentukan kehidupan dari suatu variable atau berapa lama variable itu berada dalam memory. Jangkauan ditentukan oleh dimana deklarasi variable di tempatkan di dalam program.

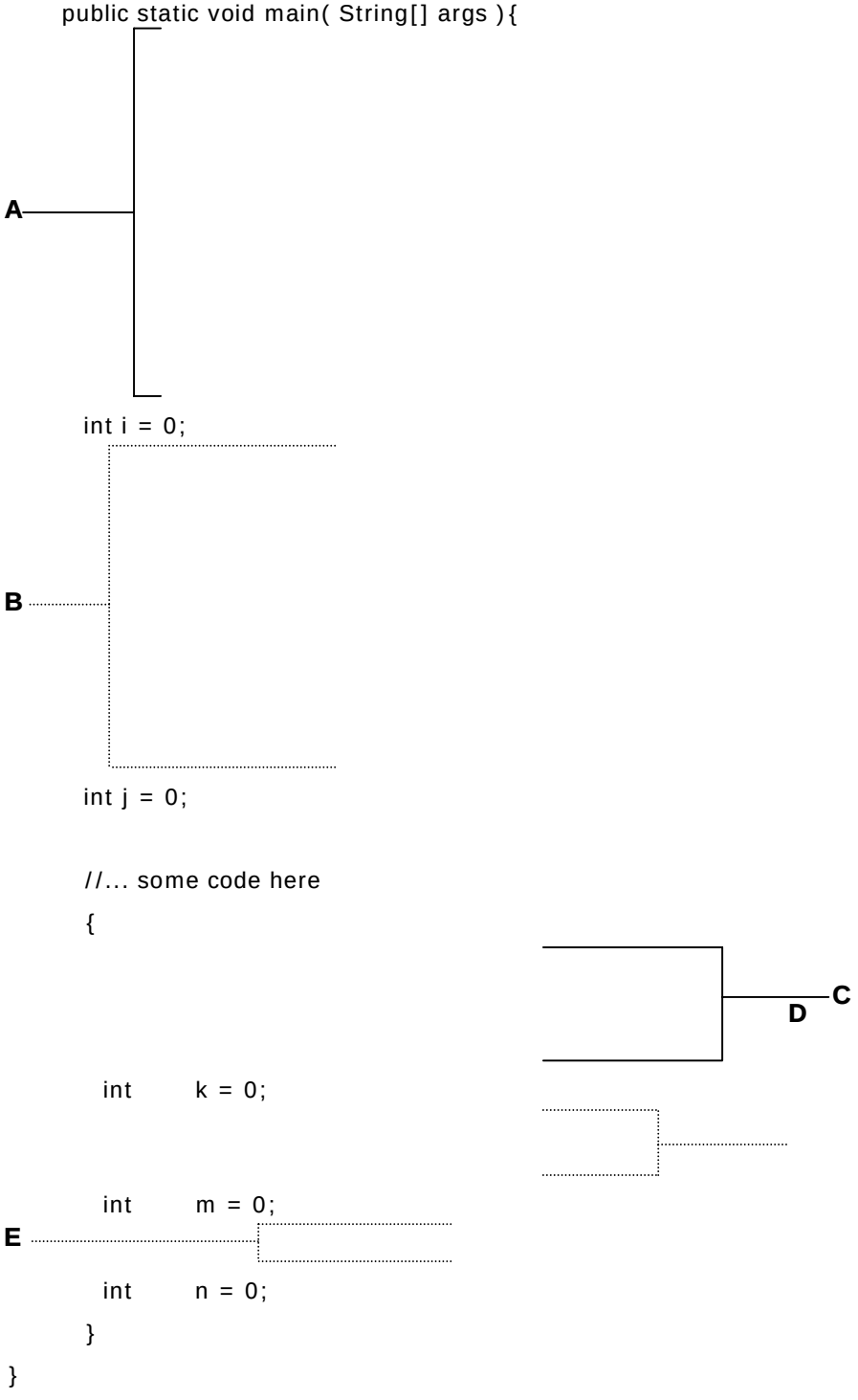
Untuk menyederhanakannya, coba berpikir tentang jangkauan apapun antara kurung kurawal {.....}, diluar kurung kurawal disebut dengan blok terluar, dan didalam kurung kurawal disebut dengan blok terdalam.

Jika kamu mendeklarasikan variable di blok luar. Mereka akan terlihat (yaitu, dapat dipakai) Oleh blok bagian dalam, bagaimana pun, jika kamu mendeklarasikan variable di blok dalam, kamu tidak bisa harapkan blok terluar untuk melihat itu.

Suatu jangkauan variable di dalam blok dimana jika sudah di deklarasi, dimulai dari titik dimana variable itu di dklarasikan, dan di blokbagian dalam.

Contoh, yang diberi code snippet,

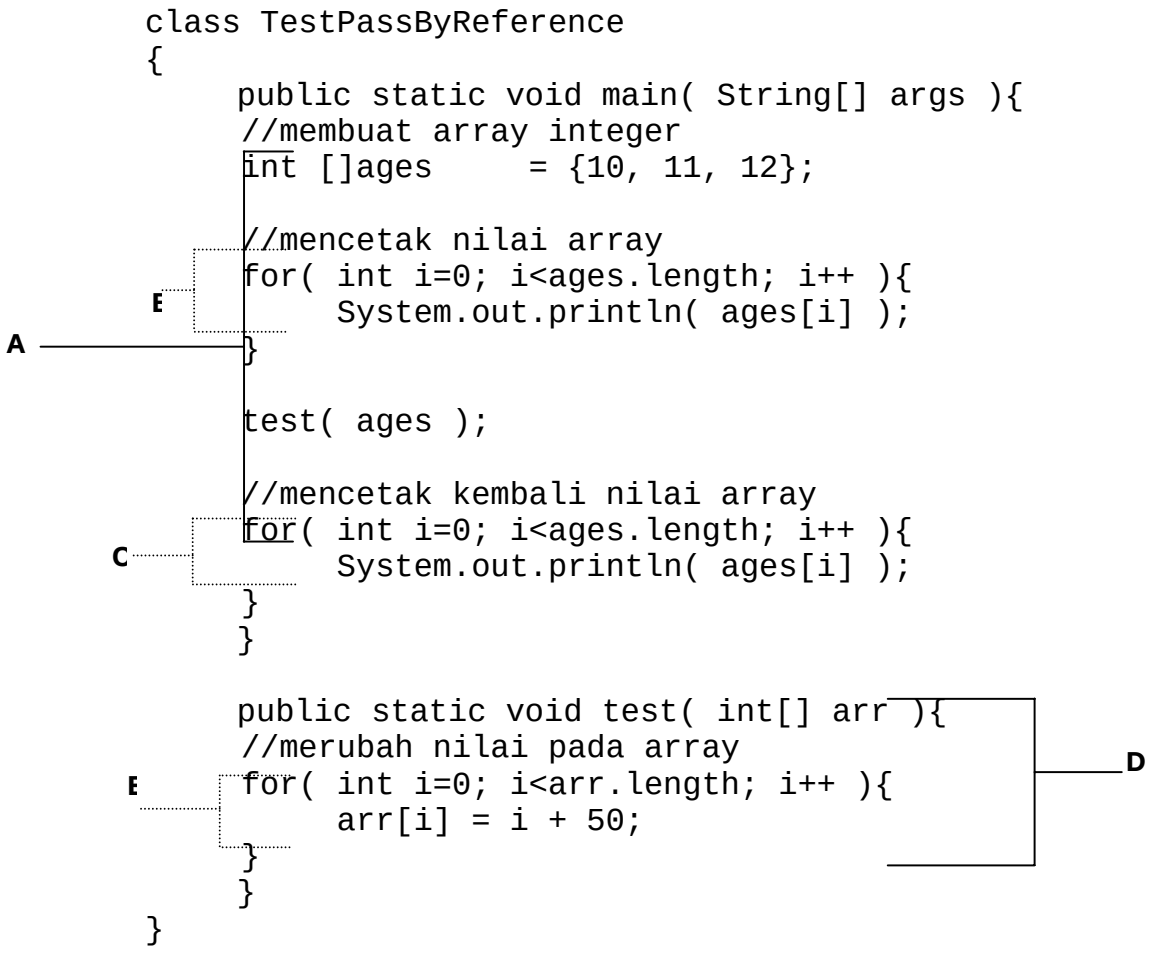
```
public class ScopeExample
{
    public static void main( String[] args ){
        int i = 0;
        int j = 0;

        //... some code here
        {
            int     k = 0;

            int     m = 0;
            int     n = 0;
        }
}
```



Kode yang kita miliki disini mempunyai lima jangkauan yang ditandai oleh baris dan keterangan yang mewakili jangkauan itu, dengan variable i,j,k,m dan n, dan 5 jangkauan A,B,C,D dan E, kita mempunyai beberapa jangkauan variable berikut:

Jangkauan variable i adalah A.
Jangkauan variable j adalah B.
Jangkauan variable k adalah C.
Jangkauan variable m adalah D.
Jangkauan variable n adalah E.

Sekarang, memberi kedua method utama dan menguji di contoh kita sebelumnya,




```
class TestPassByReference
{
     public static void main( String[] args ){
     //membuat array integer
     int []ages     = {10, 11, 12};

     //mencetak nilai array
     for( int i=0; i<ages.length; i++ ){
          System.out.println( ages[i] );
     }

     test( ages );

     //mencetak kembali nilai array
     for( int i=0; i<ages.length; i++ ){
          System.out.println( ages[i] );
     }
     }

     public static void test( int[] arr ){
     //merubah nilai pada array
     for( int i=0; i<arr.length; i++ ){
          arr[i] = i + 50;
     }
     }
}
```

Pada method pertama, Jangkauan variables adalah,

```
ages[ ]   - scope A
i in B    - scope B
i in C    – scope C
```

Pada method ujian, Jangkauan variables adalah,

```
arr[ ]    - scope D
i in E    - scope E
```

manakala variable di deklarasikan,hanya satu variable yang di identifikasi atau nama dapat di identifikasi di jangkauan, maksudnya jika kamu mempunyai deklarasi berikut,

```
{
     int test = 10;
     int test = 20;
}
```

Compilermu akan menghasilkan error karena kamu perlu mempunyai nama yang lain dari variable di satu blok, bagaimanapun, kamu dapat mempunyai dua variable dengan nama yang sama, jika mereka tidak dideklarasikan pada blok yang sama, Contoh

```
int test = 0;
System.out.print( test );
//..some code here
{
     int test = 20;
     System.out.print( test );
}
```

Manakala system pertama out.print itu memanggil, dia mencetak nilai dari variable ujian pertama sejak terdapat pada variable jangkauan itu. Yang kedua, system.out print, nilai 20 dicetak sejak tertutup ujian jangkauan variable itu.

***Petunjuk Penulisan program :***

*Hindari pemberian nama yang sama kepada variabel supaya Anda tidak kebingungan.*

# 9.5 Casting, Converting dan Comparing Objects

Pada bagian ini, kita akan belajar bagaimana mengunakan typecasting.Typecasting atau casting adalah proses konversi data dari tipe data tertentu ke tipe data yang lain. Kita juga akan belajar bagaimana meng-konversi tipe data primitive ke object dan sebaliknya. Kemudian, pada akhirnya kita akan belajar bagaimana membandingkan sebuah object.

### *9.5.1 Casting Tipe Primitiv*

Casting antara tipe primitve mendukung Anda untuk mengkonversikan sebuah value dari sebuah tipe data tertentu kepada tipe primitive yang lain. Hal ini biasanya terjadi diantara tipe data numerik.

Ada sebuah tipe data primitive yang tetap tidak dapat kita casting, dan dia adalah tipe data boolean.

Sebagai contoh dari typecasting adalah pada saat Anda menyimpan sebuah integer kepada sebuah variabel dengan tipe data double. Sebagai contoh:

```
int   numInt = 10;
double  numDouble = numInt; //implicit cast
```

Pada contoh ini dapat kita lihat bahwa, walaupu variabel yang dituju (double) memiliki nilai yang lebih besar daripada nilai yang akan kita tempatkan didalamnya, data tersebut secara implisit dapat kita casting ke tipe data double.

Contoh yang lain adalah apabila kita ingin untuk melakukan typecasting sebuah int ke char atau sebaliknya. Sebuah karakter akan dapat digunakan sebagai int karena setiap karakter memiliki sebuah nilai numerik yang merepresentasikan posisinya dalam satu set karakter. Jika sebuah variable memiliki nilai 65, maka cast (char) i akan menghasilkan nilai 'A'. Numerik kode yang merepresentasikan kapital A adalah 65, berdasarkan karakter set ASCII, dan Java telah mengadopsi bagian ini untuk mendukung karakter.

```
char valChar = 'A';
int  valInt = valChar;
System.out.print( valInt ); //casting explisit: keluaran 65
```

Ketika kita men-convert data yang bertipe besar ke tipe data yang lebih kecil, kita harus menggunakan **explicit cast**. Explicit casts mengikuti bentuk sebagai berikut :

```
(dataType)value
```

dimana,

*dataType*, adalah nama dari tipe data yang Anda convert
*value,* adalah pernyataan yang dihasilkan pada nilai dari the source type.

Sebagai contoh,

```
double        valDouble = 10.12;
int     valInt = (int)valDouble; //men-convert valDouble ke tipe int

double x = 10.2;
int     y = 2;

int     result = (int)(x/y); //hasil typecast operasi ke int
```

### *9.5.2 Casting Objects*

Instances dari class-class juga dapat di pilih ke instance-instance dari class-class yang lain, dengan **satu batasan**: **class-class sumber dan tujuan harus terhubung dengan mekanisme inheritance; satu class harus menjadi sebuah subclass terhadap class yang lain.** kita akan akan menjelaskan mengenai inheritance pada kesempatan selanjutnya.

Sejalan dengan pemilihan nilai primitive untuk tipe yang lebih besar, beberapa object mungkin tidak membutuhkan untuk dipilih secara explisit. Faktanya, karena sebuah semua subclass terdiri atas informasi yang sama, Anda dapat menggunakan instance dari subclass diamanpun sebuah superclass diharapkan berada.

Sebagai contoh, mempertimbangkan methode yang memiliki dua argument, satu tipe object dan tipe window yang lain. Anda dapat melewatkan instance dari beberapa class untuk argument object karena semua class java adalah subclass dari object. Untuk argument window, anda dapat melewatkannya kedalam subclassnya, seperti dialog, FileDialog, dan frame. Ini benar dimanapun dalam program, bukan hanya dalam memanggil methode. Jika anda mempunyai variabel yang didefinisikan sebagai window class, anda dapat memberikan object dari kelas tersebut atau dari subclassnya untuk variabelnya tanpa pemilihan.

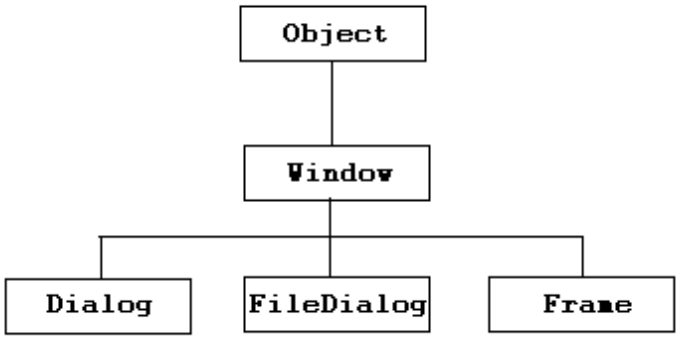



*Figure 2: Contoh Hierarchy Class*

Ini dibenarkan dalam kasus yang berkebalikan, dan Anda dapat menggunakan superclass ketika sebuah subclass dibentuk. Ada yang didapatkan dalam kasus ini, bagaimanapun: **Karena subclass terdiri dari lebih banyak kemungkinan aksi daripada superclassnya, terdapat kehilangan dalam keseimbangan keterlibatan.** Object superclass itu mungkin tidak memiliki semua kemungkinan aksi yang diperlukan untuk aksi pada tempat dari object subclass berada. Sebagai contoh jika anda memiliki operasi yang memanggil methode dalam object dari class integer, mengunakan object dari class Number tidak akan terdiri dari benyak methode yang dispesifikasikan dalam integer. error terjadi jika Anda mencoba untuk memanggil methode yang tidak memiliki object tujuan.

Untuk mengunakan object-object superclass dimana object-object subclass diharapkan, anda harus memilih mereka secara eksplisit. Anda tidak akan kehilangan beberapa informasi dalam pemilihan, tapi anda memperoleh keuntungan dari semua method dan

variabel yang mendefinisikan subclass. Untuk memilih sebuah object ke class yang lain, Anda menggunakan operasi yang sama sebagaimana untuk tipe-tipe primitive :

Untuk memilih,

(classname)object

dimana,
*classname,* adalah nama dari class tujuan.
*object,* adalah sesuatu yang mengarah pada sumber object.

- **Catatan:** pemilihan ini membuat referensi ke object yang lama dari tipe namaclass; object yang lama melanjutkan aksi seperti yang telah ada sebelumnya.

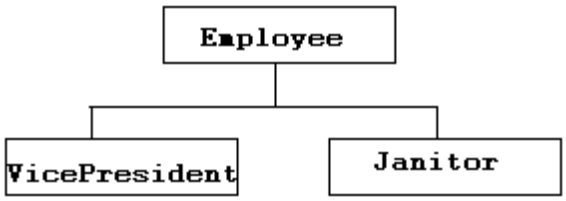



*Figure 3: Class Hierarchy untuk superclass Employee*

Contoh berikut memilih sebuah instance dari class VicePresident ke sabuah instance dari class Employee; VicePresident adalah sebuah dari Employee dengan lebih banyak information, dimana disini mendefinisikan bahwa VicePresident memilihi executive washroom privileges,

```
Employee emp = new Employee();
VicePresident veep = new VicePresident();
emp = veep; // tidak adah pemilihan yang diperlukan untuk penggunaan
yang cenderung naik
veep = (VicePresident)emp; // Harus memilih dengan pemilihan secara
eksplisit
```

### *9.5.3 Convert Tipe Primitive ke Object Dan Sebaliknya*

Satu hal yang tidak dapat Anda lakukan pada beberapa keadaan yaitu pemilihan dari sebuah objectke sebuah tipe data primitive, atau vice versa. Tipe-tipe primitive dan objectadalah sesuatu yang sangat berbeda dalam Java, dan Anda tidak bisa secara langsung memilih diantara dua atau saling menukar diantara keduanya.

Sebagai sebuah alternatif, package **java.lang** yang terdiri atas class-class yang sesuai untuk setiap tipe data primitivenya yaitu : Float, Boolean, Byte, dan sebagainya. Kebanyakan dari class-class ini memiliki nama yang sama seperti tipe datanya, kecuali jika nama classnya diawali dengan huruf capital(Short -> sort, Double -> double dan sebagainya). Juga dua class memiliki nama yang berbeda dari tipe data yang sesuai : Character digunakan untuk variabel char dan Integer untuk variabel int. **(Disebut dengan Wrapper Classes)**

Java merepresentasikan type data dan versi classnya dengan sangat berbeda, dan sebuah program tidak akan berhasil tercompile jika Anda menggunakan hanya satu ketika yang lain juga diperlukan.

Menggunakan class-class yang sesuai untuk setiap tipe primitive, anda dapat membuat sebuah object yang memiliki nilai yang sama.

**Contoh :**

```
//Pernyataan berikut membentuk sebuah instance bertype Integer
// class dengan nilai integer 7801 (primitive -> Object)
Integer dataCount = new Integer(7801);

//Pernyataan berikut meng-converts sebuah object Integer ke
//tipe data primitive int nya. Hasilnya adalah sebuah int //dengan nilai 7801

int newCount = dataCount.intValue();

// Anda perlu suatu translasi biasa pada program
// yang meng-convert sebuah String ke sebuah tipe numeric, //seperti suatu int
// Object->primitive
String pennsylvania = "65000";
int penn = Integer.parseInt(pennsylvania);
```

- **PERHATIAN:** class Void tidak mewakili sesuatu dalam Java, jadi disini tidak ada alasan menggunakannya ketika melakukan translasi antara nilai primitive dan object. Ini adalah penjelasan mengenai kata kunci void, dimana digunakan dalam definisi method untuk mengindikasikan bahwa methode tidak memiliki sebuah nilai kembalian.

### *9.5.3 Comparing Objects*

Dalam diskusi kita sebelumnya, kita mempelajari tentang operator untuk membandingkan nilai —sama, tidak sama, lebih kecil daripada, dan sebagainya. Operator ini yang paling banyak bekerja hanya pada tipe primitive, bukan pada object. Jika Anda berusaha untuk menggunakan nilai lain sebagai operands, Compiler Java akan menghasilkan error.

Pengecualian untuk aturan ini adalah operator untuk persamaan : == (sama) dan != (tidak). Ketika dinampilkan ke object, operator ini tidak akan melakukan apa yang sebenarnya anda inginkan. Malahan mengecheck jika satu object memilki nilai yang sama seperti object lain, mereka mengenali jika kedua sisi dari operator menunjuk object yang sama.

Untuk membandingkan instances dari sebuah class dan memiliki hasil yang berarti, Anda harus mengimplementasikan method khusus dalam class Anda dan memanggil method tersebut. Sebuah contoh yang baik untuk ini adalah class String.

Sangat mungkin memiliki dua object String yang memiliki nilai yang sama. Jika Anda menggunakan operator == untuk membandingkan object ini, bagaimanapun, kita akan

mempertimbangkan nilai yang tidak sama. Walaupun isinya sesuai mereka bukan merupakan object yang sama.

Untuk melihat jika dua object String memiliki nilai yang sesuai, sebuah method dari class yang disebuat dengan equals() digunakan. Method menguji setiap character dalam string dan mengembalikan nilai true jika dua string memiliki nilai yang sama.

Kode berikut mengilustrasikan hal tersebut,

```
class EqualsTest {
  public static void main(String[] arguments) {
    String str1, str2;
    str1 = "Free the bound periodicals.";
    str2 = str1;

    System.out.println("String1: " + str1);
    System.out.println("String2: " + str2);
    System.out.println("Same object? " + (str1 == str2));

    str2 = new String(str1);

    System.out.println("String1: " + str1);
    System.out.println("String2: " + str2);
    System.out.println("Same object? " + (str1 == str2));
    System.out.println("Same value? " + str1.equals(str2));
  }
 }
```

Output program ini adalah sebagai berikut ,

OUTPUT:

```
String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? true
String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? false
Same value? True
```

Sekarang mari mendiskusikan tentang kode.

```
String str1, str2;
str1 = "Free the bound periodicals.";
```

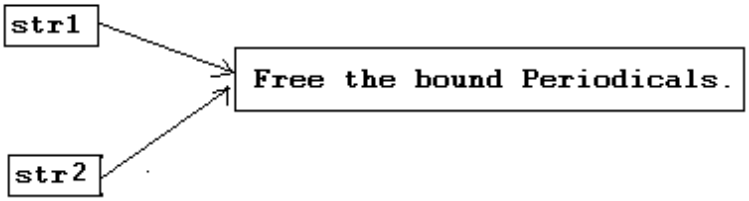



*Figure 4: Keduanya mengarah ke object yang sama*

Bagian pertama dari program ini mendeklarasikan dua variabel (str1 dan str2), memberikan literal "Free the bound periodicals." untuk str1, dan kemudian memberi nilai tersebut untuk str2. Seperti yang Anda pelajari sebelumnya, str1 dan str2 sekarang menunjuk ke object yang sama, dan uji kesamaan membuktikan hal tersebut.

```
str2 = new String(str1);
```

Padabagian yang kedua dari program ini, anda membuat object String baru dengan nilai yang sama sebagai str1 dan memberi str2 ke object baru String tersebut. Sekarang Anda memiliki dua object string yang berbeda yaitu str1 dan str2, keduanya memilki nilai yang sama.Test mereka untuk melihat jika meeka object yang sama dengan menggunakan operator == mengembalikan nilai yang diinginkan : false—mereka buka object yang sama dalam memory. Test mereka menggunakan method equals() juga mengembalikan jawaban yang diinginkan: true—mereka memiliki niali yang sama.

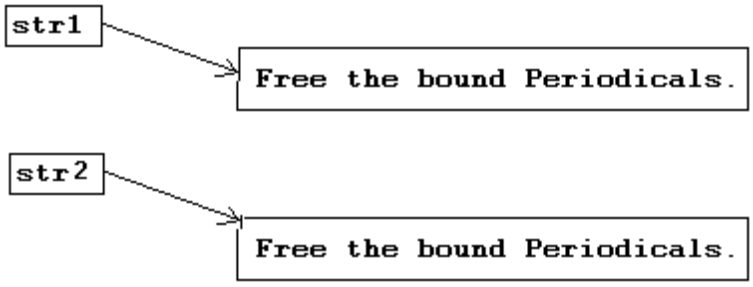



*Figure 5: Sekarang mengarah pada object yang berbeda*

- **Catatan:** Mengapa Anda tidak dapat hanya menggunakan literal yang lain ketika Anda mengubah str2, lebih dari menggunakan new? String literals diandalkan dalam Java; jika Anda membuat sebuah string menggunakan literal dan kemudian menggunakan literal yang lain dengan character yang sama, Java cukup mengetahui untuk memberikan Anda object String yang pertama kembali. kedua String adalah object yang sama; Anda harus menghidari langkah anda untuk membuat dua object terpisah.

### *9.5.5 Menentukan Class dari sebuah Object*

Ingin menemukan apakah sebuah class object itu? Disini langkah untuk melakukannya untuk sebuah object yang diberikan sebagai kunci variabel :

1. **Method getClass()** mengembalikan sebuah object Class (dimana Class itu sendiri merupakan sebuah class) yang memilki sebuah method yang disebut getName(). Pada bagiannya, getName() mengembalikan sebuah string yang mewakili nama class.

   Sebagai contoh,

   ```
   String name = key.getClass().getName();
   ```

**2. operator InstanceOf**

instanceOf memiliki dua operands: suatu mengarahke sebuah object pada sebelah kiri dan nama class pada sebelah kanan. pernyataan mengembalikan nilai true atau false tergantung pada apakah object adalah sebuah instance dari penamaan class atau beberapa dari subclass milik class tersebut.

Sebagai contoh,

```
boolean ex1 = "Texas" instanceof String; // true
Object pt = new Point(10, 10);
boolean ex2 = pt instanceof String; // false
```

# 9.6 Latihan

## *9.6.1 Mendefinisikan Istilah*

Dengan kata-kata Anda sendiri, definisikan istilah-istilah berikut ini :

1. Class
2. Object
3. Instantiate
4. Instance Variable
5. Instance Method
6. Class Variables atau static member variables
7. Constructor

## *9.6.2 Java Scavanger Hunt*

Pipoy adalah suatu anggota baru dalam bahasa pemrograman Java. Dia hanya memperdengarkan bahwa telah ada APIs siap pakai dalam Java yang salah satunya dapat digunakan dalam program mereka, dan ia ingin sekali untuk mengusahakan mereka keluar. Masalahnya adalah, Pipoy tidak memiliki copy dari dokumentasi Java, dan dia juga tidak memiliki acces internet, jadi tidak ada jalan untuknya untuk menunjukkan Java APIs.

Tugas Anda adalah untuk membantu Pipoy memperhatikan APIs (Application Programming Interface). Anda harus menyebutkan class dimana seharusnya method berada, deklarasi method dan penggunaan contoh yang dinyatakan method.
Sebagai contoh, jika Pipoy ingin untuk mengetahui method yang mengknversisebuah String ke integer,jawaban Anda seharusnya menjadi:

**Class:** Integer
**Method Declaration:** public static int parseInt( String value )
**Sample Usage:**

```
String      strValue = "100";
int     value = Integer.parseInt( strValue );
```

yakinkan bahwa snippet dari kode yang Anda tulis dalam contoh Anda menggunakan compiles dan memberi output jawaban yang benar, jadi tidak membingungkan Pipoy. **(Hint: Semua methods adalah dalam java.lang package).** Dalam kasus dimana Anda dapat menemukan lebih banyak methods yang dapat menyelesaikan tugas, berikan hanya satu.

**Sekarang mari memulai pencarian!**

1. Perhatikan sebuah method yang diuji jika String pasti diakhiri suffix yang pasti. Sebagai contoh, jika diberikan string "Hello", Method harus mengembalikan nilai true suffix yang diberikan adalah "lo", dan false jika suffix yang diberikan adalah "alp".
2. Perhatikan untuk method yang mengenali character yang mewakili sebuah digit yang spesifik dalam radix khusus. Sebagai contoh, jika input digit adalah 15, dan the radix adalah 16, method akan mengembalikan Character F, sejak F adalah representasi hexadecimal untuk angka 15 (berbasis 10).
3. Perhatikan untuk method yang mengakhiri running Java Virtual Machine yang sedang berjalan
4. Perhatikan untuk method yang memperoleh lantai dari sebuah nilai double. Sebagai contoh, jika Saya input a 3.13, method harus mengembalikan nilai 3.
5. Perhatikan untuk method yang mengenali jika character yang dipakai adalah sebuah digit. Sebagai contoh, jika Saya input '3', dia akan mengembalikan nilai true.