BAB 9

Bekerja dengan Java Class Library

9.1 Tujuan

Pada bab ini, kita akan mengantarkan beberapa konsep dasar dari Pemrograman berorientasi obyek (*Object Oriented Porgramming*). Selanjutnya, kita akan membahas konsep dari class dan obyek, serta penggunaan class dan anggota-anggotanya termasuk perbandingan, konversi dan pengubahan obyek. Untuk saat ini, kita memfokuskan dalam penggunaan class yang telah dijabarkan dalam Java Class library, lalu akan kita lanjutkan tentang pembuatan class anda sendiri.

Pada akhir pembahasan, diharapkan pembaca dapat :

1. Menjelaskan mengenai Pemrograman berorientasi Obyek dan beberapa konsepnya
2. Perbedaan antara class dan obyek
3. Pebedaan antara variabel/method yang diturunkan dan variable/method class (static)
4. Menjelaskan mengenai method, serta cara pemanggilan dan pemberian parameter ke dalam method
5. Mengidentifikasi beberapa jangkauan dari sebuah variabel
6. Mengubah tipe data dan obyek primitif
7. Membandingkan obyek dan menjabarkan class dari obyek.

9.2 Pengenalan Pemrograman Berorientasi Obyek

OOP berputar pada konsep dari obyek yang merupakan elemen dasar dari program Anda. Ketika kita membandingkan dengan dunia nyata, kita dapat menemukan beberapa obyek disekitar kita seperti mobil, singa, manusia dan seterusnya. Obyek ini dikarakterisasi oleh atribut dan tingkah lakunya.

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

Obyek	***Atribut***	***Tingkah Laku***
Mobil	Tipe dari transmisi manufaktur Warna	Berbelok Mengerem Mempercepat
Singa	Berat Warna Lapar atau tidak lapar Jinak atau liar	roaring Tidur Berburu

Table 1: Example of Real-life Objects

Dengan deskripsi ini, obyek pada dunia nyata dapat secara mudah asumsikan sebagai obyek perangkat lunak menggunakan atribut sebagai data dan tingkah laku sebagai method. Data dan method dapat digunakan dalam pemrograman game atau perangkat lunak interaktif untuk membuat simulasi obyek pada dunia nyata. Contohnya adalah perangkat lunak obyek mobil dalam permainan balap mobil atau perangkat lunak obyek singa dalam sebuah perangkat lunak pendidikan interaktif pada kebun binatang untuk anak anak.

9.3 Class dan Object

9.3.1 Perbedaan Class dan Object

Pada dunia perangkat lunak, sebuah obyek adalah sebuah komponen perangkat lunak yang stukturnya mirip dengan obyek pada dunia nyata. Setiap obyek dibangun dari sekumpulan data (atribut) yang disebut variabel untuk menjabarkan karakteristik khusus dari obyek, dan juga terdiri dari sekumpulan method yang menjabarkan tingkah laku dari obyek. Bisa dikatakan bahwa obyek adalah sebuah perangkat lunak yang berisi sekumpulan variabel dan method yg berhubungan. Variabel dan method dalam obyek Java secara formal diketahui sebagai variabel instance dan method instance. Hal ini dilakukan untuk membedakan dari variabel class dan method class, dimana akan dibahas kemudian.

Class adalah sturktur dasar dari OOP. Class terdiri dari dua tipe dari anggota dimana disebut dengan field (attribut/properti) dan method. Field merupakan tipe data yang didefinisikan oleh class, sementara method merupakan operasi. Sebuah obyek adalah sebuah instance (keturunan) dari class.

Untuk dapat membedakanantara class dan obyek, mari kita mendiskusikan beberapa contoh berikut ini. Kita memiliki sebuah class mobil dimana dapat digunakan untuk medefinisikan beberapa obyek mobil. Pada tabel dibawah, mobil A dan mobil B adalah obyek dari class mobil. Class memiliki field nomor, plat, warna, manufaktur dan kecepatan yang diisi dengan nilai pada obyek mobil A dan mobil B. Mobil juga dapat berakselerasi, berbelok dan melakukan rem.

Class mobil		**Obyek mobil A**	**Obyek Mobil B**
Variabel Intsance	Nomor Plat	ABC 111	XYZ 123
	Warna	Biru	Merah
	Manufaktur	Mitsubishi	Toyota
	Kecepatan	50 km/h	100 km/h
Method *Instance*	Method Akselerasi		
	Method Belok		
	Method Rem		

Table 2: Contoh class car dan object-object nya

Ketika diinisialisi, setiap obyek mendapat satu set variabel yang baru. Bagaimanapun, implementasi dari method dibagi diantara objek pada class yang sama.

Class menyediakan keuntungan dari *reusability*. Programmer perangkat lunak dapat menggunakan sebuah kelas beberapa kali untuk membuat banyak objek.

9.3.2 Instansiasi Class

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

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

Ini juga sama dengan,

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

Gambar 1: Instanstiasi Class

9.3.3 Variabel Class dan Variabel Method

Selain dari variabel instance, kita juga memungkinkan untuk mendefinisikan variabel dari class, yang nantinya variabel ini dimiliki oleh class. Ini berarti variabel ini dapat memiliki nilai yang sama untuk semua objek pada class 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 method main(). Di dalam Java, kita dapat mendefinisikan banyak method yang akan kita panggil dari method yang berbeda.

Sebuah method adalah bagian-bagian kode yang dapat dipanggil oleh program utama atau dari method lainnya untuk menjalankan fungsi yang spesifik.

Berikut adalah karakteristik dari method :

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

Sekarang mengapa kita butuh untuk membuat banyak method? Mengapa kita tidak menuliskan semua kode pada sebuah method? Hal ini karena penyelesaian masalah yang sangat efektif adalah memecah masalah-masalah tersebut menjadi beberapa bagian. Kita juga dapat melakukan hal ini di Java dengan membuat method untuk mengatasi bagian tertentu dari masalah. Sebuah permasalahan dapat dipecah-pecah menjadi beberapa bagian kecil. Hal ini sangat baik sekali untuk membuat program yang sangat besar.

9.4.2 Memanggil Instance dan memberikan Variabel dari Method

Sekarang, untuk mengilustrasikan bagaimana memanggil method, mari kita menggunakan class string sebagai contoh. Anda dapat menggunakan the dokumentasi dari Java API untuk melihat semua method yang tersedia dalam class string. Selanjutnya, kita akan membuat method, kita sendiri. Tapi untuk saat ini, mari terlebih dahulu kita gunakan method yang sudah disediakan oleh Java.

Untuk memanggil sebuah instance method, kita dapat menuliskan :

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

mari kita mengambil dua contoh method yang ditemukan dalam class String.

Deklarasi method	**Definisi**
public char charAt(int index)	Mengambil karakter pada indeks tertentu.
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 Pemberian Variabel Dalam Method

Pada contoh kita sebelumnya , kita sudah pernah mecoba melewati variabel pada method. Walaupun kita belumdapat membedakan antara perbedaan tipe variabel yang diberikan (*passing*) ke method dalam Java. Ada dua tipe data variabel *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 membuat sebuah salinan dari nilai variable yang dikirimkan ke method. Walaupun demikian, method tidak dapat secara langsung memodifikasi nilai variabel pengirimnya meskipun parameter salinannya sudah dimodifikasi nilainya di dalam method.

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;
    }
}
```

Pass i as parameter which is copied to j

Pada contoh diatas, kita memanggil method tes dan melewatkan nilai variabel i sebagai parameter. Nilai pada i disalinkan ke variable j pada method. Pada kondisi ini variabel j adalah merupakan variabel pengganti pada method tes, jika nilai j berubah maka nilai variabel i yang terletak pada main tidak akan ikut berubah walaupun awalnya variabel j merupakan salinan dari variabel i.

Secara default, semua tipe data primitif ketika dilewatkan pada sebuah method adalah *pass-by-value*.

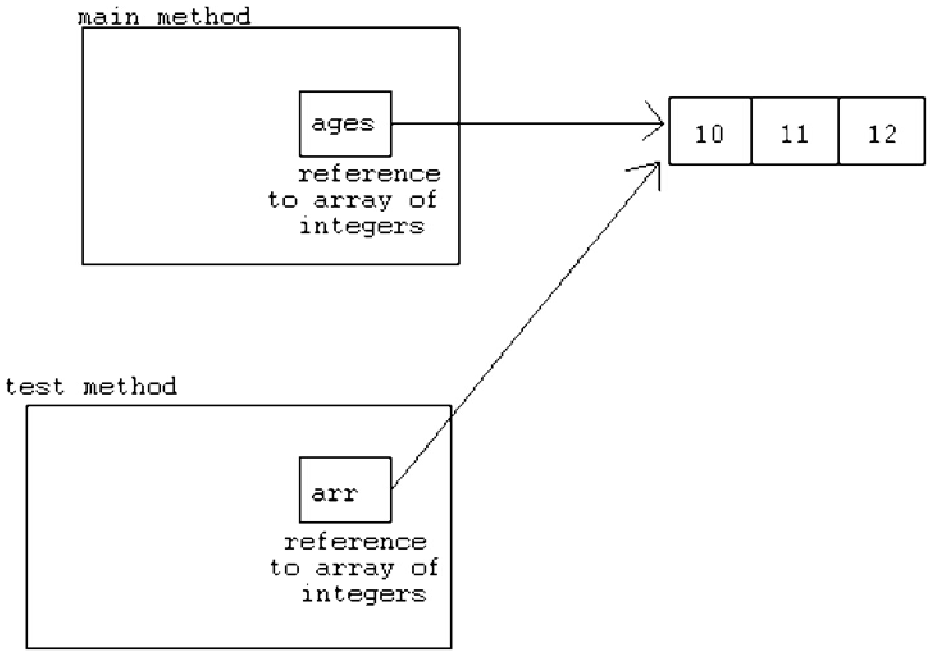
9.4.3.2 *Pass-by-reference*

Ketika sebuah pass-by-reference terjadi, alamat memori dari nilai pada sebuah variabel dilewatkan pada saat pemanggilan method. Hal ini berarti bahwa method menyalin alamat memori dari variabel yang dilewatkan pada method. Ini tidak seperti pada pass-by-value, method dapat memodifikasi variabel asli dengan menggunakan alamat memori tersebut, meskipun berbeda nama variabel yang digunakan dalam method dengan variabel aslinya, kedua variabel ini menunjukkan lokasi dari data yang 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 :

Kesalahan konsep mengenai pass-by-reference pada Java adalah pada saat membuat method untuk menukar nilai variabel menggunakan pengalamatan java. Perlu diingat bahwa java memanipulasi obyek-obyek dengan cara 'by reference', akan tetapi java mengirimkan alamat obyek ke dalam method dengan cara 'by value'. Untuk itu, anda tidak dapat menuliskan method penukaran standar ke dalam obyek.

9.4.4 Memanggil Method Static

Method Static adalah method yang dapat dipakai tanpa harus menginisialisasi suatu class (maksudnya tanpa menggunakan variabel terlebih dahulu). Method static hanya dimiliki oleh class dan tidak dapat digunakan oleh instance (atau objek) dari suatu class. Method static dibedakan dari method yang dapat instance 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

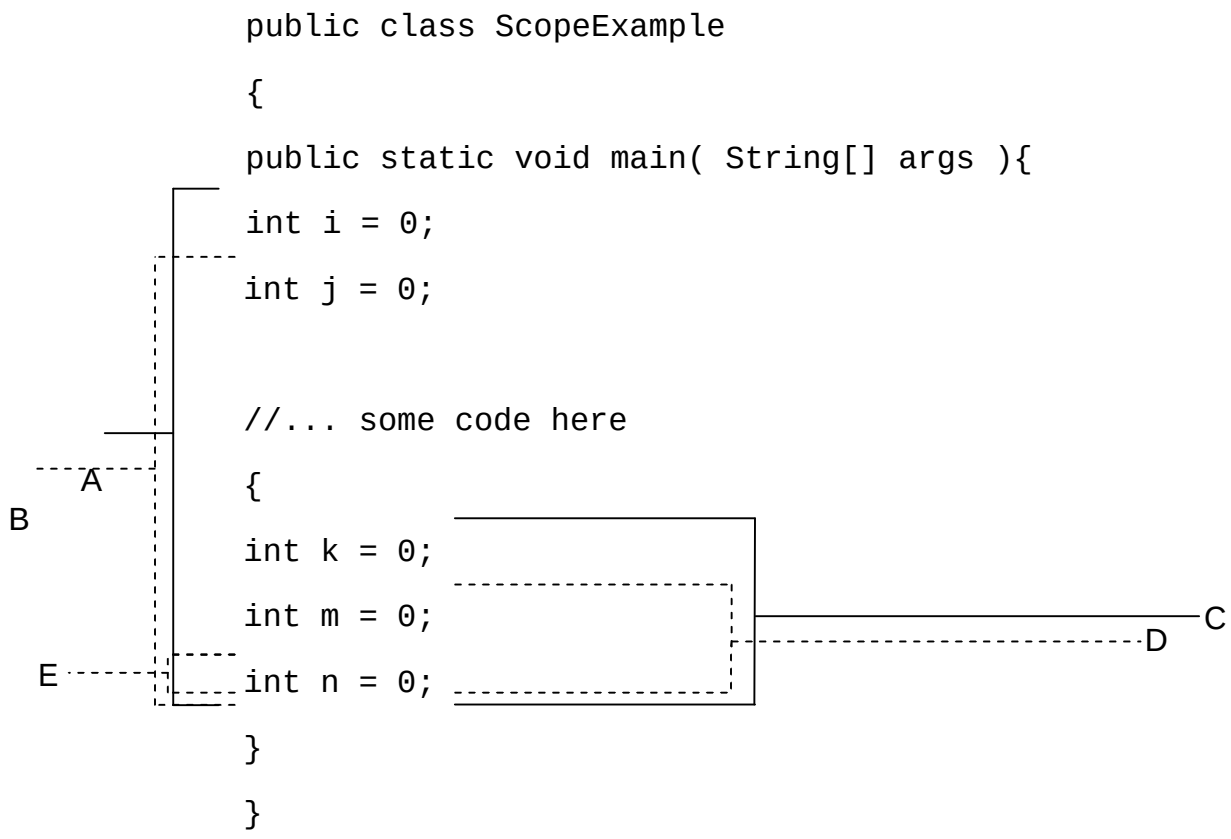
Selain nama dan tipe data yang dimiliki oleh variabel, suatu variablejuga mempunyai jangkauan. Jangkauan ini menentukan kemampuan program dalam mengakses variable. Jangkauan ini juga menentukan siklus hidup dari suatu variable atau berapa lama variable itu berada dalam memori. Jangkauan ini ditentukan oleh letak pendeklarasian variabel di dalam program.

Untuk memudahkan anda mengenai jangkauan variabel, kita bedakan variabel yang terletak di dalam kurawal {.....}. Blok kode yang terdapat di luar kurung kurawal bisa disebut juga dengan blok luar, dan blok kode yang terletak di dalam kurung kurawal disebut dengan blok dalam.

Jika kamu mendeklarasikan variabel di blok luar, variabel akan dapat dipakai oleh blok bagian dalam. Lain halnya jika kamu mendeklarasikan variabel di blok dalam, kamu tidak bisa harapkan blok terluar untuk menggunakan variabel tersebut.

Jadi dapat disimpulkan, suatu jangkauan variable dapat terletak di dalam blok dimana variabel tersebut sudah di deklarasikan, dimulai dari tempat dimana variable itu di dklarasikan dan di blok-blok bagian dalam.

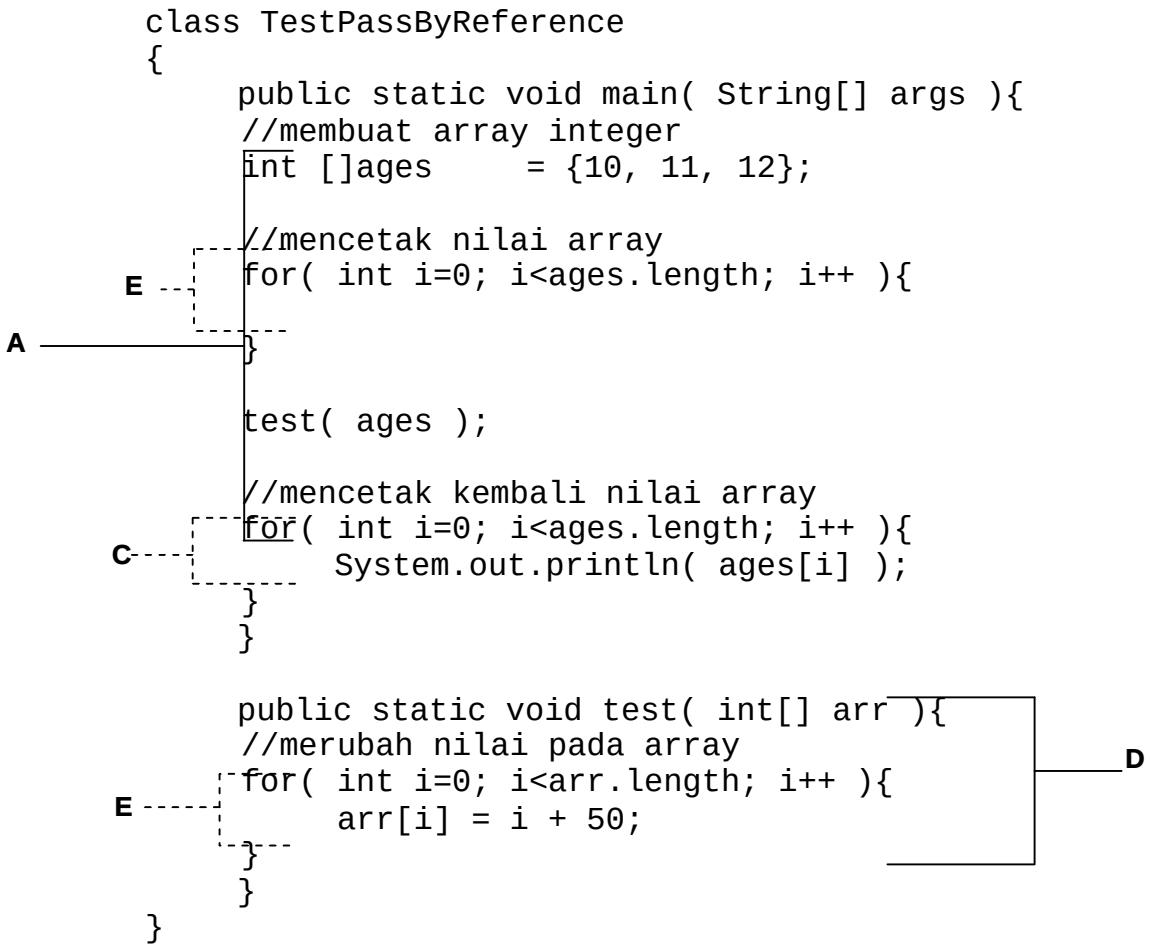
Sebagai Contoh, perhatikan potongan kode berikut:



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.

Contoh lain, jika terdapat dua method, yaitu main dan test,



Pada method main Jangkauan variabel adalah,

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

Pada method test, Jangkauan variabel adalah,

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

Pada saat variabel di deklarasikan, hanya boleh terdapat sebuah variabel dengan identifier/nama dapat di deklarasikan di dalam sebuah jangkauan. Hal ini maksudnya jika kamu mempunyai deklarasi berikut,

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

Compiler akan menghasilkan error karena *compiler* mendeteksi nama yang lain dari variabel di satu blok, namun compiler tidak akan menghasilkan error jika menumukan dua variabel dengan nama yang sama jika kedua variabel tersebut 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.out.print (baris 2) pertama dipanggil, dia mencetak nilai dari variabel test pertama (nilai 0) pada saat variabel terlihat pada jangkauan. Pernyataan System.out.print yang kedua (baris6), nilai 20 dicetak mengambil nilai variabel test terdekat (baris 6)pada jangkauan blok kode tersebut.

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 mengkonversi tipe data primitif ke obyek dan sebaliknya. Kemudian, pada akhirnya kita akan belajar bagaimana membandingkan sebuah obyek.

9.5.1 Casting Tipe data Primitif

Casting antara tipe primitif dapat mengijinkan Anda untuk mengkonversikan sebuah nilai dari sebuah tipe data tertentu kepada tipe primitif yang lain. Hal ini biasanya terjadi diantara tipe data angka.

Ada sebuah tipe data primitif yang tidak dapat kita casting, yaitu tipe data boolean.

Sebagai contoh dari *typecasting* adalah pada saat Anda menyimpan sebuah variabel dengan tipe data 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, walaupun variabel yang dituju (double) memiliki ukuran 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 nilai integer 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'. Kode numerik yang merepresentasikan huruf kapital A adalah 65, berdasarkan set karakter 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 mengkonversi 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 konversi
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 Obyek

Instances dari class-class juga dapat di ubah 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 menjelaskan mengenai inheritance pada bab selanjutnya.

Sejalan dengan konversi nilai primitif ke tipe yang lebih besar ukurannya, beberapa object mungkin tidak membutuhkan untuk ubah secara explisit. Faktanya karena semua subclass terdiri atas informasi yang sama seperti class induknya.

Sebagai contoh, jika terdapat method yang memiliki dua argumen, satu tipe *object* dan tipe *window* yang lain. Anda dapat melewatkan instance dari class apapun untuk argumen *object* karena semua class java adalah subclass dari *object*. Untuk argumen *window*, anda dapat melewatkannya subclassnya, seperti *dialog*, *FileDialog*, dan *frame*. Hal ini dapat dituliskan dalam program dan bukan hanya dalam memanggil method. Jika anda mempunyai variabel yang didefinisikan sebagai class *window*, anda dapat memberikan obyek dari class tersebut atau dari subclassnya untuk dijadikan variabel tanpa konversi.

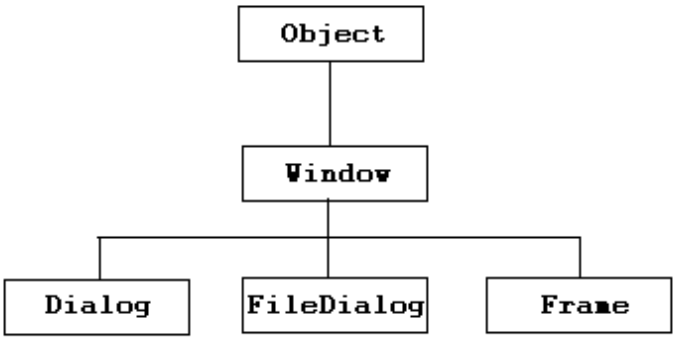


Figure 2: Contoh Hierarchy Class

Untuk kebalikannya pun bisa dilakukan, Anda dapat menggunakan superclass ketika sebuah subclass dibentuk. Akan tetapi error akan ditemukan k**arena subclass kemungkinan memiliki lebih banyak tingkah laku daripada superclassnya, maka mungkin akan terjadi hilangnya kendali dari superclass untuk method tertentu.** Superclass *object* mungkin tidak memiliki semua kemungkinan tingkah laku yang dapat dilakukan subclass *object*. Sebagai contoh jika anda memiliki operasi yang memanggil method dalam obyek dari class integer, mengunakan object dari class Number tidak akan didapatkan method lebih banyak dari yang dispesifikasikan dalam integer. Error akan terjadi jika Anda mencoba untuk memanggil method yang tidak dimiliki oleh obyek tujuan.

Untuk mengunakan obyek-obyek superclass dimana obyek-obyek subclass ditentukan terlebih dahulu, anda harus mengkonversi mereka secara eksplisit. Anda tidak akan kehilangan informasi sebelumnya setelah proses konversi, tapi anda mendapatkan semua method dan

variabel yang didefinisikan oleh subclass. Untuk konversi sebuah obyek ke class yang lain, Anda dapat menggunakan operasi yang sama seperti konversi untuk tipe-tipe primitif : Untuk memilih,

(classname)object

dimana,

classname, adalah nama dari class tujuan.

object, adalah sesuatu yang mengarah pada obyek sumber.

- **Catatan:** casting akan membuat pengalamatan ke obyek yang lama dari classname yang dituliskan; obyek yang lama akan dapat tetap dipakai seperti sebelumnya.

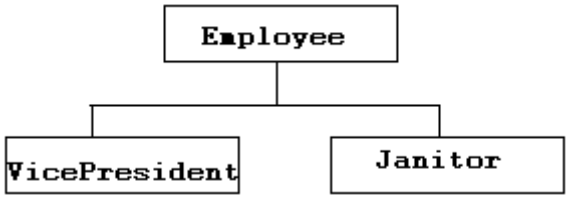


Figure 3: Class Hierarchy untuk superclass Employee

Contoh berikut konversi sebuah instance dari class VicePresident ke sabuah instance dari class Employee; VicePresident adalah sebuah subclass dari Employee dengan lebih banyak informasi, disini didefinisikan bahwa VicePresident memiliki akses menuju *executive washroom*,

```
Employee emp = new Employee();
VicePresident veep = new VicePresident();
emp = veep; // tidak adah konversi 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 konversi dari sebuah obyek ke sebuah tipe data primitif, atau sebaliknya. Tipe-tipe data primitif dan obyek adalah sesuatu yang sangat berbeda dalam Java, dan Anda tidak bisa secara langsung konversi diantara keduanya.

Sebagai sebuah alternatif, package **java.lang** yang terdiri atas class-class yang berhubungan untuk setiap tipe data primitif 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 besar (Short -> sort, Double -> double dan sebagainya). Selain itu terdapat dua class yang memiliki nama berbeda dari tipe data primitifnya yaitu: Character digunakan untuk variabel char dan Integer untuk variabel int. **(Disebut dengan Wrapper Classes)**

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

Menggunakan class-class yang berhubungan untuk setiap tipe primitif, anda dapat membuat sebuah obyek yang memiliki nilai yang sama.

Contoh :

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

//Pernyataan berikut meng-konversi sebuah obyek Integer ke
//tipe data primitif int nya. Hasilnya adalah sebuah int
//dengan nilai 7801

int newCount = dataCount.intValue();

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

- **PERHATIAN:** class Void tidak mewakili apapun dalam Java, jadi class ini tidak akan digunakan ketika melakukan konversi antara nilai primitif dan obyek. Kata void digunakan dalam definisi method untuk mengindikasikan bahwa method tidak memiliki sebuah nilai yang dihasilkan.

9.5.3 Membandingkan dua Obyek

Dalam diskusi kita sebelumnya, kita mempelajari tentang operator untuk membandingkan nilai — sama dengan, tidak sama dengan, lebih kecil daripada, dan sebagainya. Operator ini hanya bekerja pada tipe primitif, bukan pada obyek. Jika Anda berusaha untuk menggunakan nilai selain tipe data primitif sebagai operand, compiler Java akan menghasilkan error.

Salah satu contoh untuk aturan ini adalah operator untuk persamaan : == (sama dengan) dan != (tidak samadengan), ketika operator ini dimasukkan ke sebuah obyek, operator ini tidak akan melakukan apa yang sebenarnya diinginkan. Penggunaan operator ini akan memeriksa kesamaan antara satu obyek ke obyek lain, bukan nilainya.

Untuk membandingkan instance dari sebuah, 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. Namun, jika Anda menggunakan operator == untuk membandingkan obyek ini, kedua obyek tersebut akan

menghasilkan hasil yang tidak sama. Walaupun isinya ternyata sama akan tetapi pada kenyataannya mereka bukan merupakan obyek yang sama.

Untuk melihat jika dua object String memiliki nilai yang sama, sebuah method dari class yaitu equals() digunakan. Method ini akan memeriksa setiap karakter dalam *string* dan mengembalikan nilai true jika dua obyek 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));
  }
 }
```

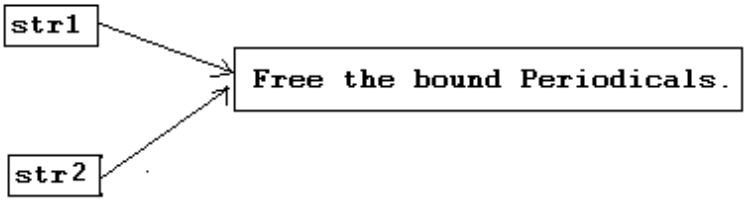
Output program ini adalah sebagai berikut ,

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 dua baris kode dari program sebagai berikut.

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

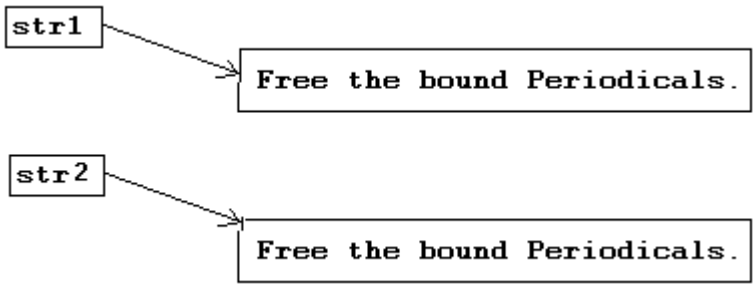


Gambar 4: Keduanya mengarah ke object yang sama

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

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

Pada bagian yang kedua dari program ini, anda membuat obyek String baru dengan nilai yang sama dari str1 dan memberikkanya ke str2. Sekarang Anda memiliki dua obyek string yang berbeda yaitu str1 dan str2, keduanya memilki nilai yang sama. Ketika dilakukan pengujian untuk melihat jika mereka obyek yang sama dengan menggunakan operator ==, akan dihasilkan nilai: false — kedua obyek bukan obyek yang sama dalam memory. ketika dilakukan pengujian menggunakan method equals() maka akan dihasilkan : true – kedua obyek tersebut memiliki nilai yang sama.



Gambar 5: Alamat sekarang mengarah pada object yang berbeda

- **Catatan:** Mengapa kita tidak dapat hanya menggunakan literal yang lain ketika ingin mengubah str2, selain menggunakan new? String literals sudah dioptimasi dalam Java; jika Anda membuat sebuah string menggunakan literal dan kemudian menggunakan literal yang lain dengan character yang sama, Java cukup memberikan Anda obyek String yang pertama kembali. Jadi kedua String tadi adalah obyek yang sama; Anda harus menghidari langkah anda untuk membuat dua obyek terpisah.

9.5.5 Menentukan Class dari sebuah Object

Jika kita ingin mengetahui class dari sebuah obyek dapat dilakukan dengan cara :

1. **Method getClass()** mengembalikan sebuah obyek Class (dimana Class itu sendiri merupakan sebuah class) yang memilki sebuah method getName(). Selanjutnya getName() akan mengembalikan sebuah string yang mewakili nama class.

   Sebagai contoh,

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

2. **operator InstanceOf**

   instanceOf memiliki dua operand: obyek pada sebelah kiri dan nama class pada sebelah kanan. Pernyataan ini mengembalikan nilai true atau false tergantung pada benar/salah obyek adalah sebuah instance dari penamaan class atau beberapa 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 method yang mengenali jika character yang dipakai adalah sebuah digit. Sebagai contoh, jika Saya input '3', dia akan mengembalikan nilai true.