BAB 7
POINTER DALAM JAVA

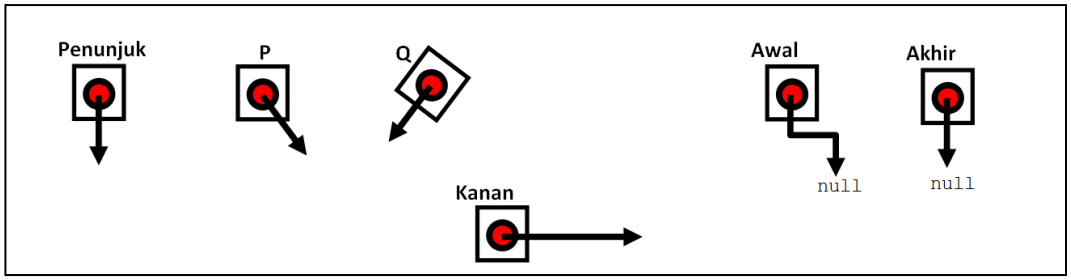
Tujuan :

Mahasiswa dapat mengekploitasi variabel pointer di dalam java

7.1. Pengenalan Pointer

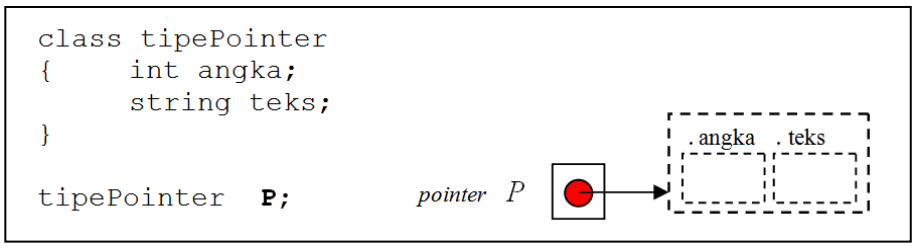
Pointer (Penunjuk) adalah sebuah variabel bertipe khusus yang dapat "menunjuk" (baca: menyimpan alamat dari) sebuah variabel tak bernama/ heap/ obyek. Tipe data dari sebuah pointer bukanlah tipe data primitif sebagaimana int, char, boolean, melainkan tipe data buatan sehingga untuk mendeklarasikannya haruslah menggunakan sebuah ***class*** (Lihat Gambar 7.2). Sebagai referensi anda juga dapat mempelajari kembali bab 2 sub bab 2.2. Struktur Penyimpan yang Terstruktur (Berbasis Record)).

Sebagaimana variabel, Pointer boleh bernama apapun untuk menggambarkan untuk tujuan apa pointer tersebut diciptakan. Gambar 7.1 di bawah ini memberikan gambaran beberapa contoh penamaan pointer (P, Q, Penunjuk, Awal, Akhir, Kanan, dll.)



Gambar 7.1

Selanjutnya, bagaimanakah sebuah pointer diciptakan? Pada Gambar 7.2 berikut ini akan dicontohkan bagaimana pembuatan sebuah pointer bernama **P**.



Gambar 7.2

Pada gambar 7.2 di atas terdapat sebuah perintah `tipePointer P`. Ini artinya diciptakanlah sebuah pointer bernama P yang mampu "menunjuk" (baca: menyimpan alamat dari) sebuah variabel tak bernama (heap) bertipe 'tipePointer'. Di dalam heap tak

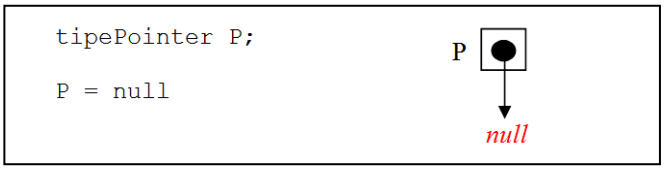
bernama tersebut nantinya akan ada 2 buah variabel anggota yaitu *angka* dan *teks*. Namun saat pointer P tercipta heap yang dimaksud belumlah tercipta (dalam gambar diilustrasikan sebagai kotak persegi dengan garis terputus-putus).

Berdasarkan kemampuannya, terdapat tiga karakteristik dari sebuah pointer yaitu :

a. Pointer dapat menunjuk ke NULL
b. Pointer dapat menunjuk ke HEAP
c. Pointer dapat saling mengcopy (alamat dari) heap yang sedang ditunjuk. Dengan kata lain, pointer dapat dibuat menunjuk ke tempat yang ditunjuk oleh pointer lain.

7.1.1. Pointer menunjuk ke null

Null adalah sebuah tempat kosong di memori yang tidak menyimpan sebuah nilai. Null dapat diumpamakan seperti ground pada sebuah rangkaian elektronika. Sebuah pointer dapat diarahkan untuk menunjuk ke null. Biasanya pointer yang sedang tidak bertugas sebagai penunjuk sebuah heap diarahkan untuk menunjuk ke tempat ini.



Gambar 7.3

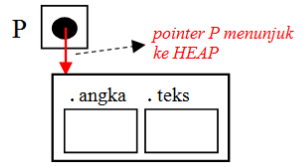
7.1.2. Pointer menunjuk ke sebuah heap baru

Selain menunjuk ke null, pointer juga dapat diarahkan untuk menunjuk ke sebuah heap. Heap adalah sebuah variabel tak bernama yang berisi satu atau lebih variabel lain. Dalam konsep OOP heap dikenal dengan istilah **obyek**. Heap juga sering disebut **Node /Simpul/ List/ Senarai/ Record**. Lebih jauh tentang heap akan dibahas pada subbab 7.2.

Agar pointer dapat menunjuk ke sebuah heap, heap harus diciptakan terlebih dahulu. Perintah untuk menciptakan sebuah heap baru adalah `new()` **.** Setelah sebuah heap tercipta, barulah sebuah pointer dapat diarahkan untuk menunjuk ke heap tersebut. Perintah yang digunakan agar pointer `P` dapat menunjuk ke heap yang baru tersebut adalah `P =`.

```
tipePointer P;

P = new tipePointer();
```



Gambar 7.4

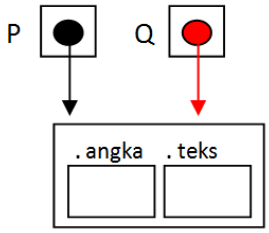
7.1.3. Pointer yang saling mengcopykan alamat heap yang ditunjuk

Pointer dapat saling mengcopy alamat heap yang sedang ditunjuknya. Apabila alamat dari sebuah heap dicopykan dari satu pointer ke pointer yang lain (misalnya dari P dicopy ke Q) maka itu berarti pointer yang menerima copynya (pointer Q) akan diarahkan untuk menunjuk ke heap yang sedang ditunjuk oleh pointer pemberi copy (pointer P).

```
tipePointer P;
P = new tipePointer;

tipePointer Q;

Q = P;  ← Q diarahkan menunjuk ke heap yang ditunjuk oleh P
```



Gambar 7.5

Gambar 7.5 memperlihatkan bahwa setelah `Q = P` dilakukan, maka pointer Q akan menunjuk ke heap yang ditunjuk oleh pointer P. Dengan kata lain pointer P dan pointer Q akan menunjuk ke heap yang sama.

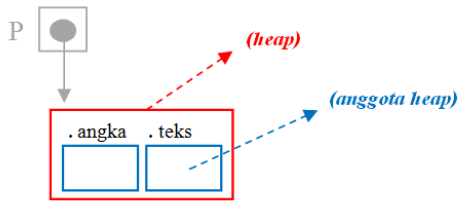
7.2. Pengenalan Heap

Heap, yang disebut juga Obyek/ Simpul/ List/ Node/ Senarai/ Record, adalah sebuah variabel tak bernama yang dialokasikan secara khusus dalam memori yang dapat menyimpan banyak variabel. Agar dapat digunakan, sebuah heap haruslah diciptakan terlebih dahulu. Perintah untuk menciptakan sebuah heap baru adalah **`new()`**. Untuk lebih jelasnya perhatikan Gambar 7.6 berikut.

```
class tipePointer
{     int angka;
      string teks;
}

tipePointer  P;

P = new tipePointer()   ← perintah untuk menciptakan heap
```



Gambar 7.6

Pada program di atas, perintah 'new tipePointer()' digunakan untuk menciptakan sebuah heap. Setelah sebuah heap tercipta, barulah sebuah pointer dapat diarahkan untuk menunjuk ke heap tersebut. Perintah yang digunakan agar pointer P menunjuk ke heap yang baru dibentuk tersebut adalah P =.

Adapun bentuk dari heap yang diciptakan dengan perintah new() identik dengan nama class yang dideklarasikan di atasnya. Pada gambar 7.6 di atas, heap diciptakan dengan class bernama "tipePointer" di mana class tersebut memiliki 2 variabel anggota yaitu angka (bertipe int) dan teks (bertipe String). Oleh karena heap diciptakan atas dasar class "tipePointer" tersebut, maka heap secara otomatis juga memiliki 2 variabel yaitu angka dan teks.

Berdasarkan cara aksesnya, anggota heap yang bernama angka dan teks dapat diakses, baik untuk sekedar ditampilkan ke layar maupun untuk dimanipulasi nilainya. Format yang digunakan untuk penyebutan sebuah anggota heap adalah :

[nama pointer] • *[nama variabel anggota]*

Sebagai contoh perhatikan program di bawah ini.

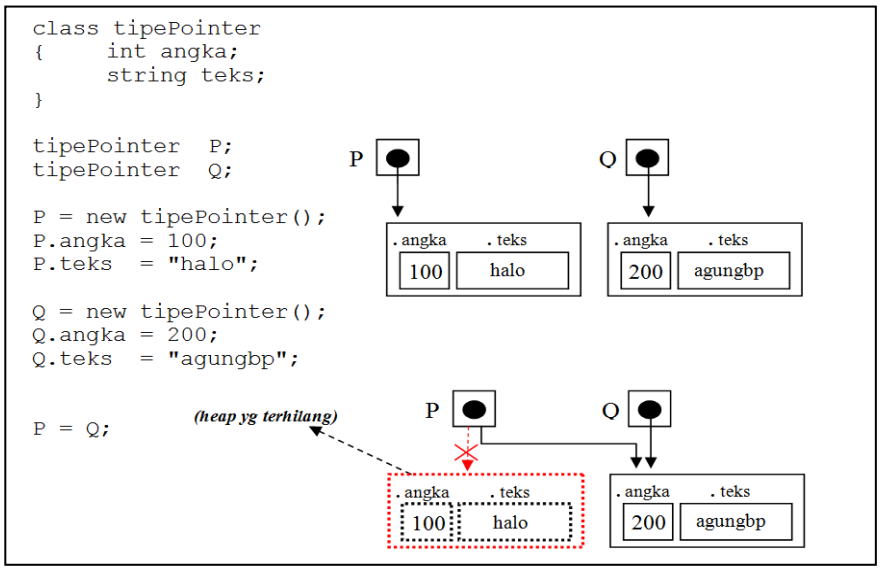
```
P.angka = 20;
P.teks = "agungbp";
System.out.println (P.angka);
System.out.println (P.teks);
```

Dari sisi pengalokasian dan pendealokasiannya, sebuah heap dapat diciptakan dengan menggunakan perintah new. Namun demikian, uniknya Java tidak menyediakan perintah untuk menghapusnya (dealokasi). Hal ini karena compiler java akan melakukan

dealokasi heap yang ada secara otomatis pada saat program/ subprogram selesai dieksekusi. Ini bertujuan untuk mengurangi *human error* apabila progammer lupa melakukan dealokasi heap dari memori yang dapat membuat memori menjadi penuh. Konsep ini tentu agak berbeda dengan bahasa C++ yang menyediakan perintah untuk alokasi maupun dealokasi.

7.3. Heap yang Terhilang

Dalam prakteknya, sebuah heap dapat juga "terhilang" di dalam memori. Hal ini dapat terjadi yaitu jika heap tersebut ditinggalkan oleh pointer yang menunjuknya. Yang dimaksud terhilang di atas adalah sekalipun heap tersebut masih ada di dalam memori, heap tidak dapat diakses lagi karena tidak ada pointer yang menjadi sarana untuk mengaksesnya. Sebagai contoh perhatikan program pada Gambar 7.7. berikut ini.



Gambar 7.7

Pada program di atas pointer `P` menunjuk ke heap yang berisi angka=*100* dan teks=*"halo"*, sedangkan pointer `Q` menunjuk ke heap yang berisi angka=*200* dan teks=*"agungbp"*. Pada kondisi ini apabila kemudian dilakukan perintah `P = Q;` maka `P` yang tadinya menunjuk ke heap yang berisi *100* dan "*halo*" akan berubah arah menunjuk ke heap yang berisi *200* dan "*agungbp*". Hal ini akan mengakibatkan heap yang berisi angka=*100* dan teks="*halo*" terhilang dan tidak dapat diselamatkan (diakses) lagi.

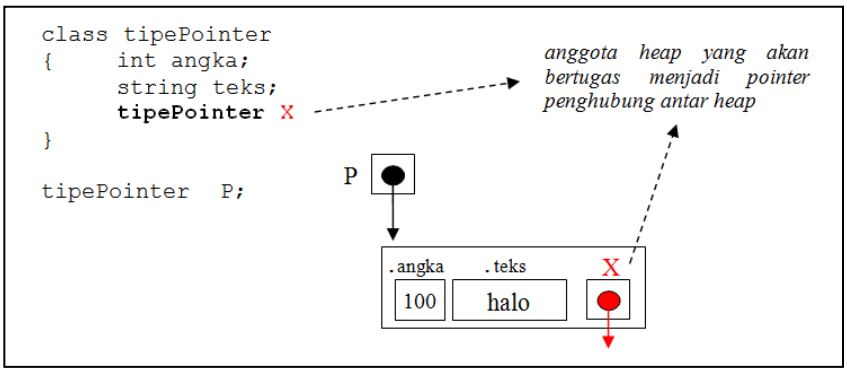
Di sisi lain, sebuah heap dapat ditunjuk oleh lebih dari satu pointer. Pada kasus di atas, setelah penunjukan `P` berpindah dari heap yang berisi angka=*100* dan teks="*halo*" menjadi menunjuk ke heap yang berisi angka=*200* dan teks="*agungbp*", maka heap yang berisi angka=*200* dan teks="*agungbp*" memiliki 2 buah pointer yang menunjuknya yaitu `P` dan `Q`. Dalam kondisi seperti ini heap yang berisi angka=*200* dan teks="*agungbp*" dapat

diakses melalui 2 cara yaitu `P.angka` ataupun `Q.angka`, keduanya akan merujuk pada sebuah nilai yang sama. Demikian juga `P.teks` dan `Q.teks` keduanya juga merujuk pada nilai yang sama.

7.4. Heap Yang Saling Terhubung (Linkedlist / Senarai Berantai)

Dalam rangka memanfaatkan heap sebagai media penyimpan data, sebuah heap dapat juga dibuat untuk saling terhubung dengan heap-heap yang lain yang sejenis.

Untuk dapat melakukan hal ini kita harus terlebih dahulu menambahkan sebuah anggota heap lagi yang berbentuk pointer. Untuk lebih jelasnya perhatikan gambar 7.8 berikut ini.



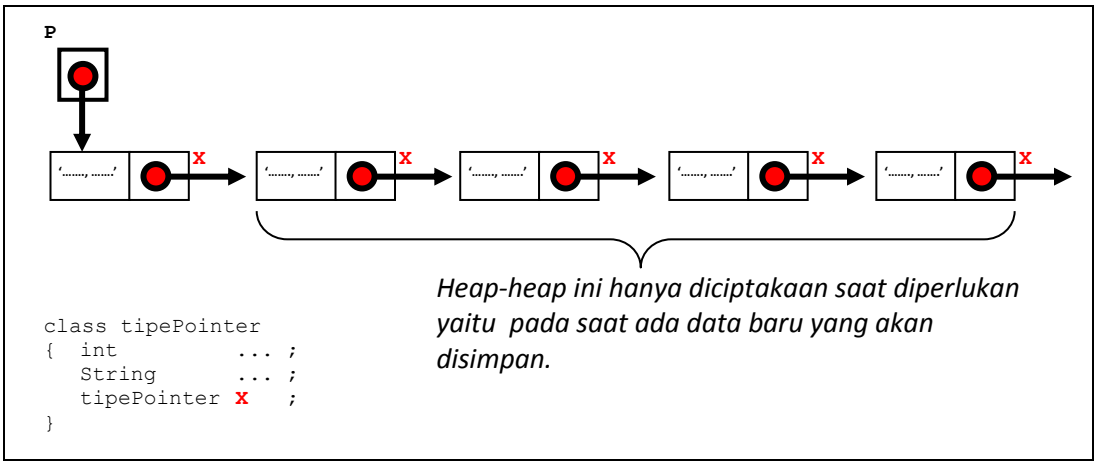
Gambar 7.8

Dari struktur heap di atas terlihat ada satu lagi variabel baru yang ditambahkan sebagai anggota dari class tipePointer, yaitu variabel `X`. Namun jika diperhatikan dengan seksama, uniknya variabel X ini adalah bahwa dia agak berbeda dari saudara-saudaranya variabel angka dan variabel teks yang dideklarasikan menggunakan tipe data primitive int dan String. Variabel X ini bertipe tipePointer (yaitu nama classnya sendiri), sama seperti pointer P yang juga bertipe tipePointer. Dari tipe data ini kita langsung dapat mengetahui bahwa variabel X ini sesungguhnya bukanlah variabel biasa melainkan adalah sebuah pointer yang tentu akan memiliki 3 karakteristik pointer yaitu dapat menunjuk ke null, dapat menunjuk ke heap lain, dan dapat saling mengcopy alamat dari heap yang sedang ditunjuknya.

Karena dideklarasikan menggunakan tipe data "`tipePointer`" sebagaimana juga pointer `P`, maka pointer `X` juga pastilah memiliki kemampuan dan karakteristik yang sama persis dengan pointer `P`. Dan untuk selanjutnya pointer `X` inilah yang nantinya akan dimanfaatkan untuk menjadi pointer penghubung (pointer pengait) antara heap tempat dia berada dengan heap yang lain. Konsep heap terhubung inilah yang nantinya menjadi cikal bakal adanya **Linkedlist (senarai berantai).**

Linked list (list yang terhubung) atau disebut juga *Senarai Berantai,* adalah sebuah konsep pengalokasian memori secara dinamis untuk dapat digunakan untuk menyimpan data (list). Mengapa disebut dinamis? Karena pengalokasian memori ini dapat dilakukan kapanpun saat diperlukan, dan pengalokasiannya cukup mudah yaitu hanya dengan membuat sebuah simpul (*heap)* baru di memori menggunakan perintah `new` setiap ada data yang baru yang akan ditambahkan ke dalam memori.

Berbeda dengan konsep penyimpanan data menggunakan larik/ array dimana kita hanya dapat menyimpan data dalam jumlah tertentu dan tidak dapat diubah lagi (statis), pada *linked list* kita dapat menyimpan data secara lebih dinamis karena pengalokasian simpul di memori baru akan dilakukan pada saat datanya datang.

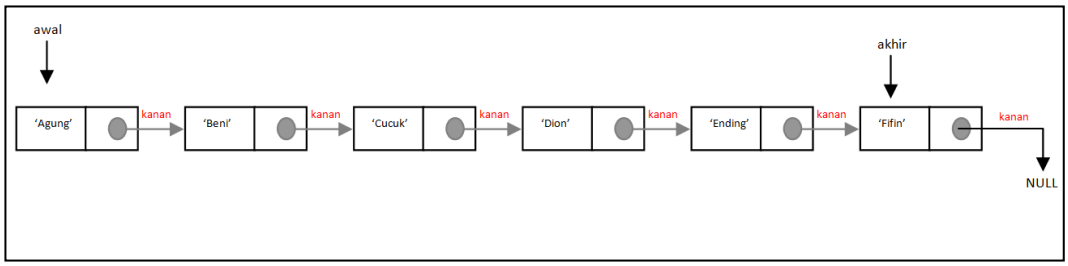


Gambar 7.9

7.5. Pengenalan Single Linked List (Senarai Berantai Tunggal)

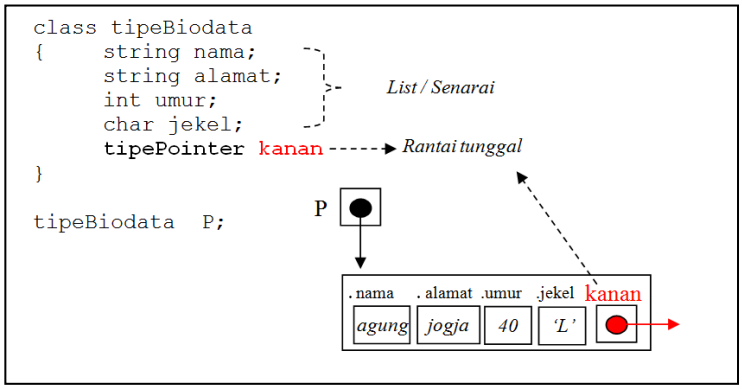
Single Linkedlist adalah kumpulan heap/ obyek/ simpul/ node yang saling terhubung satu sama lain (*linked*) dengan menggunakan **1 (satu) buah pointer pengait di bagian kanan** yang digunakan sebagai penghubung dengan simpul sejenis yang ada di sebelah kanannya. Pada kedua ujung dari single linkedlist ini terdapat 2 buah pointer yang menjadi penunjuknya yaitu `Awal` yang menunjuk heap yang berada pada posisi paling depan, dan `Akhir` yang menunjuk heap yang berada pada posisi paling belakang. Sementara heap paling belakang terkait dengan null.

Kebanyakan orang menyingkat Single Linked List (Senarai Berantai Tunggal) hanya dengan sebutan Linked List saja (Senarai Berantai).



Gambar 7.10

Deklarasi yang digunakan untuk menciptakan heap pada Single Linkedlist ini dapat dilihat pada gambar 7.11 di bawah ini.



Gambar 7.11

Dari struktur heap di atas (di dalam class tipeBiodata) terlihat adanya 2 kelompok variabel anggota. Yang pertama adalah kelompok List/Senarai. Kelompok ini nantinya akan digunakan untuk menyimpan data contohnya berupa biodata (`nama`, `alamat`, `umur` dan `jenis kelamin`). Dan kelompok yang kedua adalah adanya 1 buah pointer penghubung yang bernama `kanan` yang nantinya akan bertugas untuk merangkaikan setiap heap dengan heap yang lain.

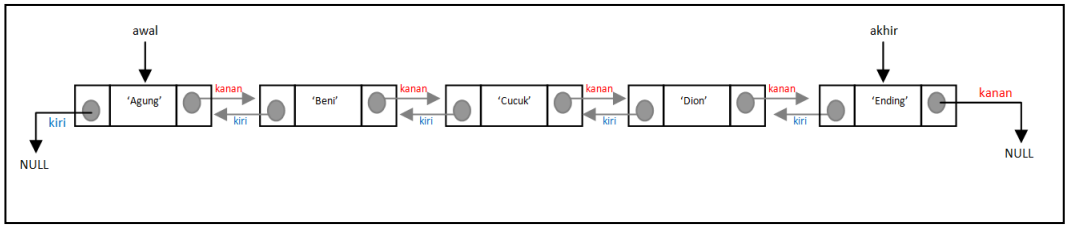
Disebut dengan Single Linkedlist atau Senarai berantai Tunggal karena jumlah dari pointer penghubungnya hanya satu yaitu `kanan`.

Pembahasan lebih lanjut tentang Single Linked List dan operasi-operasi di dalamnya dapat dilihat pada bab 8 dan bab 9.

7.6. Pengenalan Double Linked List (Senarai Berantai Ganda)

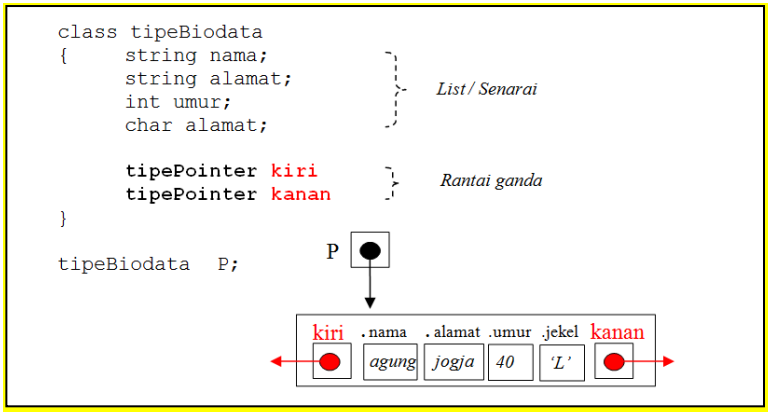
Double Linkedlist adalah kumpulan heap/ obyek/ simpul/ node yang saling terhubung satu sama lain (*linked*) dengan menggunakan **2 (dua) buah pointer pengait di bagian kanan dan kiri** yang digunakan sebagai penghubung dengan simpul sejenis yang

ada di sebelah kanan dan kirinya. Pada kedua ujung dari single linkedlist ini terdapat 2 buah pointer yang menjadi penunjuknya yaitu `Awal` yang menunjuk heap yang berada pada posisi paling depan, dan `Akhir` yang menunjuk heap yang berada pada posisi paling belakang. Sementara pada heap paling belakang dan paling depan masing-masing juga terkait dengan null.



Gambar 7.12

Deklarasi yang digunakan untuk menciptakan heap pada Double Linkedlist ini dapat dilihat pada gambar 7.13 di bawah ini.



Gambar 7.13

Dari struktur heap di atas (di dalam class tipeBiodata) terlihat adanya 2 kelompok variabel anggota. Yang pertama adalah kelompok List/Senarai. Kelompok ini nantinya akan digunakan untuk menyimpan data contohnya berupa biodata (`nama, alamat, umur` dan `jenis kelamin`). Dan kelompok yang kedua adalah adanya 2 buah pointer penghubung yang bernama `kanan` dan `kiri` yang nantinya akan bertugas untuk merangkaikan setiap heap dengan heap yang lain. Pointer `kiri` akan bertugas menghubungkan heap dengan heap tetangga sebelah kiri sementara pointer `kanan` akan bertugas menghubungkan heap dengan tetangga heap sebelah kanan

Disebut dengan Double Linkedlist atau Senarai berantai Ganda karena jumlah dari pointer penghubungnya ada dua buah yaitu `kanan` dan `kiri`.

Pembahasan lebih lanjut tentang Double Linked List dan operasi-operasi di dalamnya dapat dilihat pada bab 10.

7.7. Memanipulasi Anggota Heap Milik Pointer Tetangga

Dalam sebuah deretan heap-heap yang telah saling terhubung satu dengan yang lain, sebuah elemen heap sebetulnya dapat diakses dari pointer lain yang bahkan tidak sedang menunjuk ke heap tersebut. Sebagai contoh perhatikan program di bawah ini.

```
1.    import java.util.Scanner;
2.    class tipePointer
3.    {  String namaKota;
4.       tipePointer kanan;
5.    }
6.    class belajarPointer
7.    {  public static void main(String[] args)
8.       {
9.          tipePointer P;
10.         P = new tipePointer();
11.         P.namaKota  = "Yogyakarta";
12.
13.         tipePointer Q;
14.         Q = new tipePointer();
15.         Q.namaKota  = "Klaten";
16.
17.         tipePointer R;
18.         R = new tipePointer();
19.         R.namaKota  = "Solo";
20.
21.         P.kanan = Q;
22.         Q.kanan = R;
23.         R.kanan = null;
24.         System.out.println("Nilai elemen namaKota pointer P,Q dan R adalah :");
25.         System.out.println("-----------------------------------------------");
26.         System.out.println("Nilai P.namaKota adalah = " + P.namaKota);
27.         System.out.println("Nilai Q.namaKota adalah = " + Q.namaKota);
28.         System.out.println("Nilai R.namaKota adalah = " + R.namaKota);
29.
30.         P.kanan.kanan.namaKota="Surakarta";//mengakses R.namaKota dari P
31.
32.         System.out.println("Nilai elemen namaKota pointer P,Q dan R adalah :");
33.         System.out.println("-----------------------------------------------");
34.         System.out.println("Nilai P.namaKota adalah = " + P.namaKota);
35.         System.out.println("Nilai Q.namaKota adalah = " + Q.namaKota);
36.         System.out.println("Nilai R.namaKota adalah = " + R.namaKota);
37.      }
38.   }
```

Hasil Eksekusi :

```
Nilai elemen namaKota pointer P,Q dan R adalah :
-----------------------------------------------
Nilai P.namaKota adalah = Yogyakarta
Nilai Q.namaKota adalah = Klaten
Nilai R.namaKota adalah = Solo


Nilai elemen namaKota pointer P,Q dan R adalah :
-----------------------------------------------
Nilai P.namaKota adalah = Yogyakarta
Nilai Q.namaKota adalah = Klaten
```

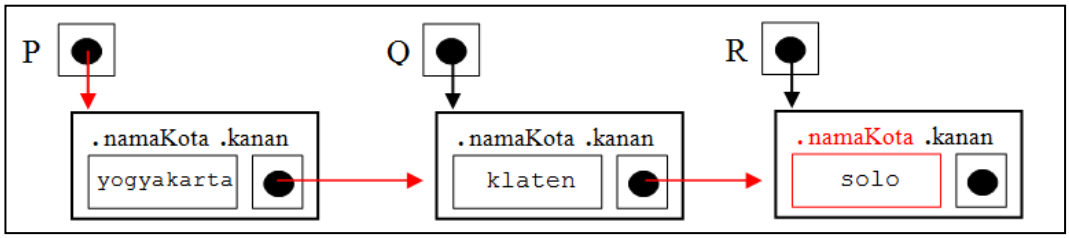
```
Nilai R.namaKota adalah = Surakarta

Press any key to continue . . .
```

Pada program di atas diketahui bahwa terdapat 3 buah pointer yaitu P, Q, dan R. Heap yang ditunjuk oleh pointer P memiliki anggota heap 'namaKota' yang bernilai "Yogyakarta". Heap yang ditunjuk oleh pointer Q memiliki anggota heap 'namaKota' yang bernilai "Klaten", sedangkan Heap yang ditunjuk oleh pointer R memiliki anggota heap 'namaKota' yang bernilai "Solo".

Ketiga heap di atas saling terhubung dengan kondisi P.kanan menunjuk ke heap yang ditunjuk oleh pointer Q, dan Q.kanan menunjuk ke heap yang ditunjuk oleh pointer R.

Ilustrasi untuk hubungan ini dapat dilihat pada Gambar 7.14.



Gambar 7.14

Akibat dari adanya nilai-nilai di atas, jika perintah pada baris 26, 27, dan 28 dieksekusi hasil yang diperoleh adalah "Yogyakarta", "Klaten", dan "Solo".

Namun sekarang perhatikanlah perintah pada baris ke 30. Di sana terlihat adanya upaya untuk mengganti elemen bernilai "*Solo*" yang sedang ditunjuk oleh pointer `R` untuk diubah nilainya menjadi "*Surakarta*". Uniknya, proses penggantian ini dilakukan bukan melalui `R.namaKota = "Surakarta"` , melainkan melalui instruksi `P.kanan.kanan.namaKota = "Surakarta";` Hal ini dimungkinkan terjadi karena baik pointer `R.namaKota` maupun pointer `P.kanan.kanan.namaKota` sama-sama menunjuk ke lokasi yang sama yaitu R.namaKota. dan hal ini bisa terjadi karena masing-masing heap telah saling terhubung.

Akibat adanya instruksi `P.kanan.kanan.namaKota = "Surakarta";` maka tanpa disadari nilai dari elemen R.namaKota pun telah berubah dari yang semula berisi "*Solo*" berubah menjadi "*Surakarta*".

7.8. Latihan

Latihan 1 :

Tuliskan program berikut ini menggunakan textpad.

```
import java.util.Scanner;

class tipePointer
{  int angka;
   string teks;
}

class belajarPointer
{  public static void main(String[] args)
   {
      tipePointer P;
      P = null;
      System.out.println("Nilai P adalah = " + P);
  }
}
```

Running program di atas dan lihat apa hasil yang terjadi? Apakah nilai pointer P dapat dilihat?

Sekarang modifikasilah program di atas menjadi seperti berikut ini.

```
import java.util.Scanner;
class tipePointer
{  int angka;
   string teks;
}

class belajarPointer
{  public static void main(String[] args)
   {
      tipePointer P;
      P = null;
      if (P == null)
          System.out.println("Pointer P menunjuk ke Null");
      else
          System.out.println("Pointer P mengarah ke tempat lain");
  }
}
```

Sekarang runninglah program di atas dan lihat apa hasil yang terjadi? Bagaimana cara pointer P bisa menunjuk ke Null? Instruksi apa yang menyebabkannya pointer P menunjuk ke null? Apakah null dapat dilihat? Jelaskan dalam laporan anda.

Latihan 2 :

Tuliskan program berikut ini.

```
import java.util.Scanner;
class tipePointer
{  int angka;
   string teks;
}

class belajarPointer
```

```
{  public static void main(String[] args)
   {
      tipePointer P;
      P = new tipePointer();
      P.angka = 100;
      P.teks  = "Halo";

      tipePointer Q = new tipePointer();
      Q.angka = 200;
      Q.teks  = "Akakom";

      System.out.println("Nilai elemen P dan Q adalah :");
      System.out.println("-----------------------------");
      System.out.println("Nilai P.angka adalah = " + P.angka);
      System.out.println("Nilai P.teks adalah = " + P.teks);
      System.out.println("Nilai Q.angka adalah = " + Q.angka);
      System.out.println("Nilai Q.teks adalah = " + Q.teks);
  }
}
```

Sekarang running lah program di atas dan lihat apa hasil yang terjadi? Berapa masing-masing nilai elemen angka dan teks, baik yang ditunjuk oleh pointer P maupun pointer Q? Jelaskan dalam laporan anda.

Sekarang modifikasilah program di atas dengan menambahkan pointer R seperti di bawah ini.

```
import java.util.Scanner;
class tipePointer
{  int angka;
   string teks;
}
class belajarPointer
{  public static void main(String[] args)
   {
      tipePointer P;
      P = new tipePointer();
      P.angka = 100;
      P.teks  = "Halo";

      tipePointer Q = new tipePointer();
      Q.angka = 200;
      Q.teks  = "Akakom";

      System.out.println("Nilai elemen P dan Q adalah :");
      System.out.println("-----------------------------");
      System.out.println("Nilai P.angka adalah = " + P.angka);
      System.out.println("Nilai P.teks adalah = " + P.teks);
      System.out.println("Nilai Q.angka adalah = " + Q.angka);
      System.out.println("Nilai Q.teks adalah = " + Q.teks);


      tipePointer R;
      R = P;
      System.out.println("Nilai elemen R adalah :");
      System.out.println("------------------------");
      System.out.println("Nilai R.angka adalah = " + R.angka);
      System.out.println("Nilai R.teks adalah = " + R.teks);

      R = Q;
      System.out.println("Nilai elemen R saat ini adalah :");
      System.out.println("-------------------------------");
      System.out.println("Nilai R.angka adalah = " + R.angka);
      System.out.println("Nilai R.teks adalah = " + R.teks);
  }
}
```

Sekarang running lah program di atas dan lihat apa hasil yang terjadi? Berapa nilai elemen angka dan teks yang ditunjuk oleh pointer R? Bagaimana bisa nilai R dapat bisa menghasilkan nilai yang berbeda pada saat elemen angka dan teks dari R ditampilkan untuk kedua kalinya? Jelaskan dalam laporan anda.

```
import java.util.Scanner;
class tipePointer
{  int angka;
   string teks;
}

class belajarPointer
{  public static void main(String[] args)
   {
      tipePointer P = new tipePointer();
      P.angka = 100;
      P.teks  = "Halo";

      tipePointer Q = new tipePointer();
      Q.angka = 200;
      Q.teks  = "Akakom";

      System.out.println("Nilai P dan Q sebelum pointer dimanipulasi :");
      System.out.println("-------------------------------------------");
      System.out.println("Nilai P.angka adalah = " + P.angka);
      System.out.println("Nilai P.teks adalah = " + P.teks);

      System.out.println("Nilai Q.angka adalah = " + Q.angka);
      System.out.println("Nilai Q.teks adalah = " + Q.teks);

      tipePointer R;
      R = P;
      P = Q;
      Q = R;
      System.out.println("Nilai P dan Q setelah pointer dimanipulasi :");
      System.out.println("-------------------------------------------");
      System.out.println("Nilai P.angka adalah = " + P.angka);
      System.out.println("Nilai P.teks adalah = " + P.teks);

      System.out.println("Nilai Q.angka adalah = " + Q.angka);
      System.out.println("Nilai Q.teks adalah = " + Q.teks);
  }
}
```

Sekarang running lah program di atas dan lihat apa hasil yang terjadi? Berapa nilai masing-masing elemen angka dan teks, baik yang ditunjuk oleh pointer P maupun pointer Q sebelum pointer dimanipulasi maupun setelah dimanipulasi? Bagaimana itu bisa terjadi? Jelaskan dalam laporan anda.

Latihan 3 :

Tuliskan program berikut ini.

```
import java.util.Scanner;
```

```
class tipePointer
{  string namaKota;
   tipePointer kanan;
}

class belajarPointer
{  public static void main(String[] args)
   {
      tipePointer P;
      P = new tipePointer();
      P.namaKota  = "Yogyakarta";

      tipePointer Q;
      Q = new tipePointer();
      Q.namaKota  = "Klaten";

      tipePointer R;
      R = new tipePointer();
      R.namaKota  = "Solo";

      tipePointer S;
      S = new tipePointer();
      S.namaKota  = "Sragen";

      tipePointer T;
      T = new tipePointer();
      T.namaKota  = "Ngawi";

      System.out.println("Nilai P,Q,R,S,T adalah :");
      System.out.println("-----------------------");
      System.out.println("Nilai P.namaKota adalah = " + P.namaKota);
      System.out.println("Nilai Q.namaKota adalah = " + Q.namaKota);
      System.out.println("Nilai R.namaKota adalah = " + R.namaKota);
      System.out.println("Nilai S.namaKota adalah = " + S.namaKota);
      System.out.println("Nilai T.namaKota adalah = " + T.namaKota);

      P.kanan = Q;
      Q.kanan = R;
      R.kanan = S;
      S.kanan = T;
      T.kanan = null;

      System.out.println("Nilai-nilai yang dapat diakses dari pointer P adalah :");
      System.out.println("-----------------------------------------------------");
      System.out.println(P.namaKota);
      System.out.println(P.kanan.namaKota);
      System.out.println(P.kanan.kanan.namaKota);
      System.out.println(P.kanan.kanan.kanan.namaKota);
      System.out.println(P.kanan.kanan.kanan.kanan.namaKota);
   }
}
```

Sekarang running lah program di atas dan lihat apa hasil yang terjadi? Berapa nilai masing-masing elemen namaKota untuk P, Q, R, S, T dengan dilakukannya deretan perintah :

```
System.out.println("Nilai P.namaKota adalah = " + P.namaKota);
System.out.println("Nilai Q.namaKota adalah = " + Q.namaKota);
System.out.println("Nilai R.namaKota adalah = " + R.namaKota);
System.out.println("Nilai S.namaKota adalah = " + S.namaKota);
System.out.println("Nilai T.namaKota adalah = " + T.namaKota);
```

Apa pula hasil yang diperoleh dengan dilakukannya deretan perintah :

```
System.out.println(P.namaKota);
```

```
System.out.println(P.kanan.namaKota);
System.out.println(P.kanan.kanan.namaKota);
System.out.println(P.kanan.kanan.kanan.namaKota);
System.out.println(P.kanan.kanan.kanan.kanan.namaKota);
```

Apa yang dapat anda simpulkan dari percobaan di atas?

Sekarang tambahkan perintah berikut ini di bagian terakhir dari program anda.

```
System.out.println(R.namaKota);
P.kanan.kanan.namaKota = "Surakarta");
System.out.println(R.namaKota);
```

Apa yang terjadi pada elemen R.namaKota ? Bagaimana nilai R.namaKota dapat berubah? Jelaskan dalam laporan anda.

Sekarang modifikasilah juga sehingga menjadi seperti program berikut ini.

```
import java.util.Scanner;

class tipePointer
{  string namaKota;
   tipePointer kanan;
}

class belajarPointer
{  public static void main(String[] args)
   {
      tipePointer P;
      P = new tipePointer();
      P.namaKota  = "Yogyakarta";

      tipePointer Q;
      Q = new tipePointer();
      Q.namaKota  = "Klaten";

      tipePointer R;
      R = new tipePointer();
      R.namaKota  = "Solo";

      tipePointer S;
      S = new tipePointer();
      S.namaKota  = "Sragen";

      tipePointer T;
      T = new tipePointer();
      T.namaKota  = "Ngawi";

      System.out.println("Elemen namaKota untuk pointer P,Q,R,S,T adalah :");
      System.out.println("-----------------------------------------------");
      System.out.println("Nilai P.namaKota adalah = " + P.namaKota);
      System.out.println("Nilai Q.namaKota adalah = " + Q.namaKota);
      System.out.println("Nilai R.namaKota adalah = " + R.namaKota);
      System.out.println("Nilai S.namaKota adalah = " + S.namaKota);
      System.out.println("Nilai T.namaKota adalah = " + T.namaKota);

      P.kanan = Q;
      Q.kanan = R;
      R.kanan = S;
      S.kanan = T;
```

```
      T.kanan = null;

      System.out.println("Elemen namaKota untuk pointer P,Q,R,S,T adalah :");
      System.out.println("-------------------------------------------------");
      System.out.println("Nilai P.namaKota adalah = " + P.namaKota);
      System.out.println("Nilai Q.namaKota adalah = " + P.kanan.namaKota);
      System.out.println("Nilai R.namaKota adalah = " + P.kanan.kanan.namaKota);
      System.out.println("Nilai S.namaKota adalah = " +
P.kanan.kanan.kanan.namaKota);
   System.out.print("Nilai T.namaKota adalah= "+
P.kanan.kanan.kanan.kanan.namaKota);

      tipePointer BANTU;
      BANTU = P;
      while (BANTU!=null)
      {  System.out.println("Nilai BANTU.namaKota adalah = " + BANTU.namaKota);
         BANTU = BANTU.kanan;
      }
  }
}
```

Sekarang running lah program di atas dan lihat apa yang terjadi dengan BANTU.kanan ketika ditampilkan? Berapa kalikah instruksi `System.out.println("Nilai BANTU.namaKota adalah = " + BANTU.namaKota);` dieksekusi oleh compiler? Selalu samakah hasilnya? Bagaimana ini bisa terjadi? Jelaskan dalam laporan anda.

Sekarang modifikasilah juga sehingga dengan menambahkan elemen *"kiri"* menjadi seperti program berikut ini.

```
import java.util.Scanner;

class tipePointer
{  string namaKota;
   tipePointer kiri;
   tipePointer kanan;
}

class belajarPointer
{  public static void main(String[] args)
   {
      tipePointer P;
      P = new tipePointer();
      P.namaKota  = "Yogyakarta";

      tipePointer Q;
      Q = new tipePointer();
      Q.namaKota  = "Klaten";

      tipePointer R;
      R = new tipePointer();
      R.namaKota  = "Solo";

      tipePointer S;
      S = new tipePointer();
      S.namaKota  = "Sragen";

      tipePointer T;
      T = new tipePointer();
      T.namaKota  = "Ngawi";

      System.out.println("Elemen namaKota untuk pointer P,Q,R,S,T adalah :");
      System.out.println("-------------------------------------------------");
      System.out.println("Nilai P.namaKota adalah = " + P.namaKota);
      System.out.println("Nilai Q.namaKota adalah = " + Q.namaKota);
      System.out.println("Nilai R.namaKota adalah = " + R.namaKota);
```

```
      System.out.println("Nilai S.namaKota adalah = " + S.namaKota);
      System.out.println("Nilai T.namaKota adalah = " + T.namaKota);

      P.kanan = Q;
      Q.kanan = R;
      R.kanan = S;
      S.kanan = T;
      T.kanan = null;

      P.kiri = null;
      Q.kiri = P;
      R.kiri = R;
      S.kiri = R;
      T.kiri = S;

      System.out.println("Elemen namaKota untuk pointer P adalah :");
      System.out.println("----------------------------------------");
      System.out.println(P.namaKota);
      System.out.println(P.kanan.kiri.namaKota);
      System.out.println(P.kanan.kanan.kiri.kiri.namaKota);

      tipePointer BANTU;
      BANTU = T;
      while (BANTU!=null)
      {  System.out.println("Nilai BANTU.namaKota adalah = " + BANTU.namaKota);
         BANTU = BANTU.kiri;
      }
   }
}
```

Sekarang running lah program di atas dan lihat apa yang terjadi dengan BANTU.kanan ketika ditampilkan? Berapa kalikah instruksi `System.out.println("Nilai BANTU.namaKota adalah = " + BANTU.namaKota);` dieksekusi oleh compiler? Selalu samakah hasilnya? Bagaimana ini bisa terjadi? Jelaskan dalam laporan anda.

Latihan

-

Tugas

-