MODUL 7
POINTER DALAM JAVA

CAPAIANPEMBELAJARAN

Mahasiswa dapat mengekploitasi variabel pointer di dalam java

KEBUTUHAN ALAT/BAHAN/SOFTWARE

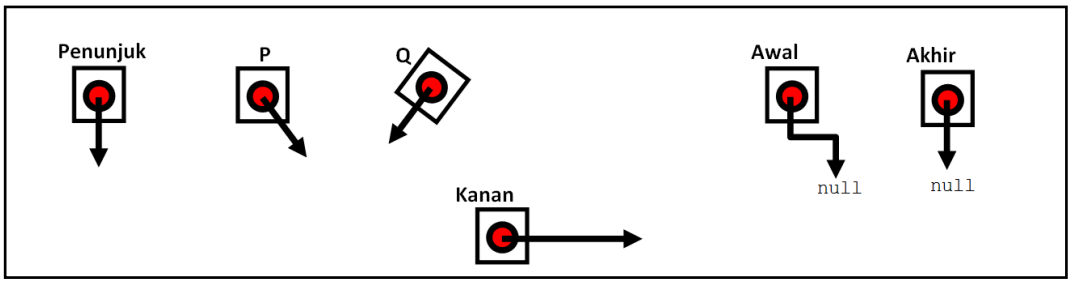
1. TextPad

DASAR TEORI

A. PENGENALAN POINTER

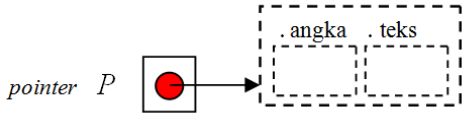
Pointer (Penunjuk) adalah sebuah variabel bertipe khusus yang dapat menunjuk (menyimpan alamat) sebuah variabel/ heap/ obyek. Tipe data dari variabel pointer (selanjutnya disebut pointer) tidak bertipe primitif sebagaimana int, char, boolean melainkan bertipe data buatan sehingga harus dideklarasikan terlebih dahulu menggunakan sebuah ***class*** *(pelajari kembali modul 2 bagian "Struktur Penyimpan Terstruktur berbasis Record/Rekaman")*.

Pointer dapat diberi nama apapun yang menggambarkan untuk apa pointer tersebut diciptakan. Contoh :



Lalu bagaimana sebuah pointer diciptakan? Untuk lebih jelasnya perhatikanlah program di bawah ini.

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

tipePointer  P;
```



Pada script di atas, pointer P adalah sebuah pointer yang dapat menyimpan sebuah alamat dari sebuah variabel obyek (HEAP) yang bertipe 'tipePointer'. Di dalam heap tersebut nantinya terdapat dua buah anggota yaitu *angka* dan *teks* (tetapi heap itu sediri sat ini belum diciptakan). Dalam pengertian yang lebih sederhana, kata "menyimpan" alamat sebuah heap dapat diartikan bahwa pointer P mampu menunjuk ke sebuah heap yang berisi variabel angka dan teks.

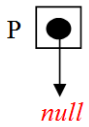
Ada tiga karakteristik dari sebuah pointer yaitu :

a. Pointer dapat menunjuk ke NULL
b. Pointer dapat menunjuk ke sebuah HEAP
c. Pointer dapat saling mengcopy (alamat) heap yang sedang ditunjuk

a. Pointer dapat menunjuk ke NULL

Null adalah sebuah tempat kosong di memori yang tidak memiliki nilai. Null dapat diumpamakan seperti ground pada sebuah rangkaian elektronika. Pointer dapat diarahkan untuk menunjuk ke null tersebut seperti gambar di bawah ini.

```
tipePointer P;

P = null
```

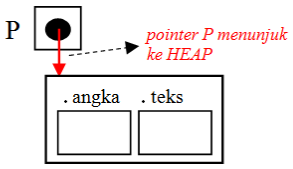


b. Pointer dapat menunjuk ke sebuah HEAP baru.

Pointer dapat menunjuk ke sebuah heap baru. Heap adalah sebuah variabel tidak bernama yang dapat menyimpan sebuah nilai sebagaimana sebuah variabel pada umumnya. Dalam konsep OOP heap dikenal dengan nama **obyek**. Heap juga sering disebut **Node /Simpul/ List/ Senarai/ Record**.

Agar pointer dapat menunjuk ke sebuah heap, heap harus diciptakan terlebih dahulu menggunakan perintah ***new()*** baru kemudian sebuah pointer dapat diarahkan untuk menunjuk ke heap tersebut. Perintah yang digunakan agar pointer P menunjuk ke heap yang baru tersebut adalah "**P =**".

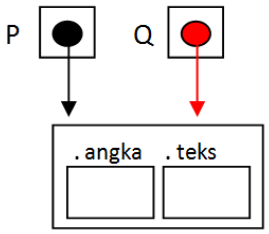
```
tipePointer P;

P = new tipePointer();
```



c. Pointer dapat saling mengcopykan alamat heap yang sedang ditunjuk

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

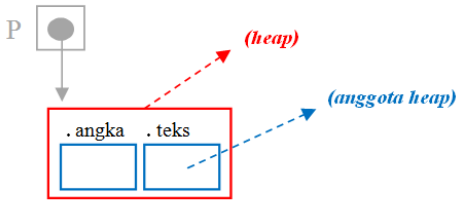
```
tipePointer P;
P = new tipePointer;

tipePointer Q;

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



B. PENGENALAN HEAP

Heap, disebut juga obyek/ simpul/ node, adalah sebuah variabel tak bernama yang dialokasikan secara khusus dalam memori yang dapat menyimpan banyak variabel. Heap diciptakan dengan perintah ***new***. Perhatikan program berikut.

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

tipePointer  P;

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



Pada program di atas, perintah '`new tipePointer()`' digunakan untuk menciptakan sebuah heap, sedangkan perintah '`P =`' digunakan untuk mengarahkan pointer P untuk menunjuk ke heap yang baru saja diciptakan.

Tipe data dari sebuah heap tergantung dari tipe kelas di mana heap tersebut dibentuk. Pada program di atas, heap diciptakan bertipe "tipePointer" yang memiliki 2 variabel anggota yaitu `angka` (bertipe int) dan `teks` (bertipe String). Oleh karena heap diciptakan dalam tipe data "tipePointer" maka heap tersebut secara otomatis juga memiliki 2 variabel, angka dan teks.

Sebagaimana variabel biasa bertipe primitif yang digunakan untuk menyimpan nilai, heap juga dapat menyimpan sebuah nilai.

Anggota heap yang bernama *angka* dan *teks* dapat diakses, baik untuk ditampilkan ke layar menggunakan `System.out.print()`, maupun untuk dimanipulasi nilainya penggunakan penugasan variabel (assigment). Format yang digunakan untuk mengakses maupun memanipulasi anggota sebuah heap adalah :

[nama pointer] ● *[nama variabel anggota]*

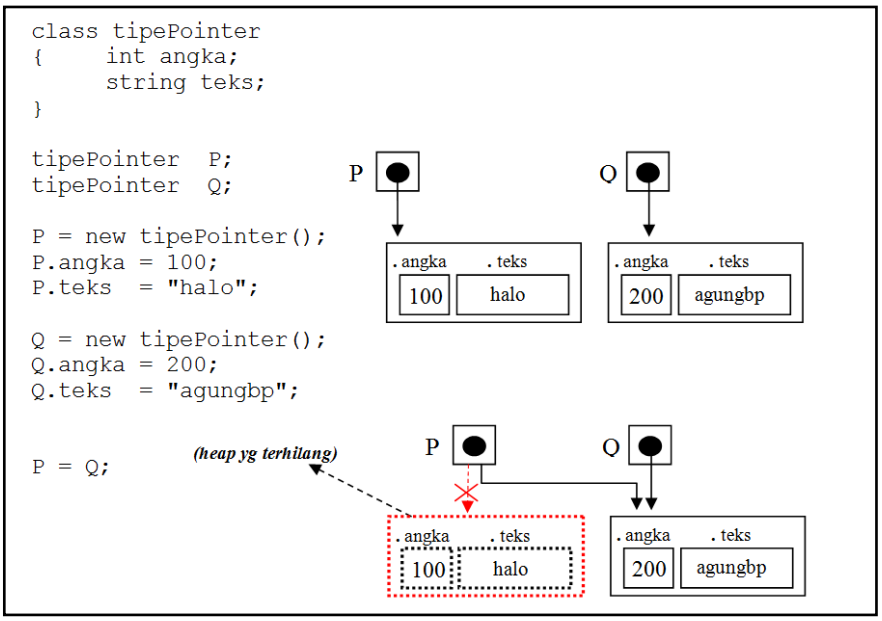
Sebagai contoh perhatikan program di bawah ini.

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

Berbeda dengan bahasa C++, heap dalam bahasa java dapat diciptakan namun tidak disediakan perintah untuk menghapusnya. Hal ini karena compiler java secara omatis akan memusnahkan heap saat program ataupun subprogram selesai dieksekusi.

Dalam prakteknya, sebuah heap dapat juga "terhilang" di dalam memori 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 berikut ini.

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

tipePointer  P;
tipePointer  Q;

P = new tipePointer();
P.angka = 100;
P.teks  = "halo";

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

P = Q;
```

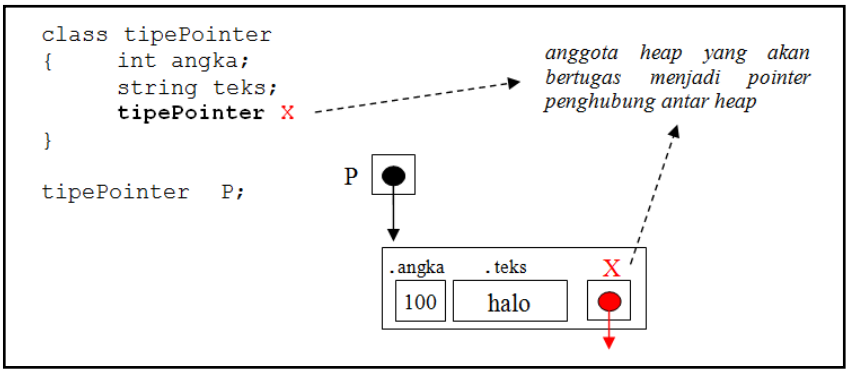


Pada program di atas pointer P menunjuk ke heap yang berisi angka=*100* dan teks=*"halo"*, sedangkan pointer Q menunjuk ke heap yang berisi angka=*200* dan teks=*"agungbp"*. Pada kondisi ini apabila kemudian dilakukan perintah `P = Q;` maka `P` yang tadinya menunjuk ke heap yang berisi *100* dan "*halo*" 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 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.

C. HEAP YANG SALING TERHUBUNG (LINKEDLIST / SENARAI BERANTAI)

Heap dapat juga dibuat untuk saling terhubung satu sama lain. Untuk dapat melakukan hal ini kita harus menambahkan sebuah variabel baru sebagai anggota heap yang harus bertipe pointer. Perhatikan contoh berikut.

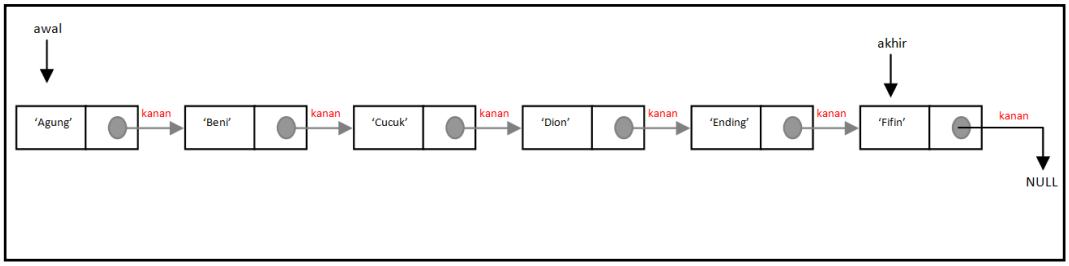


Dari struktur heap di atas tampak bahwa ada sebuah pointer tambahan X yang merupakan anggota kelas "`tipePointer`" yang dideklarasikan juga menggunakan tipe data "`tipePointer`" yang adalah dirinya sendiri. Karena dideklarasikan menggunakan tipe data "`tipePointer`" sebagaimana pointer `P`, maka pointer `X` juga memiliki kemampuan dan karakteristik yang persis sama dengan pointer `P`. Pointer `X` inilah yang nantinya dimanfaatkan untuk menjadi pointer penghubung (pointer pengait) antara heap tempat dia berada dengan heap yang lain. Kosep heap terhubung inilah yang nantinya menjadi cikal bakal adanya **Linkedlist (senarai berantai).**

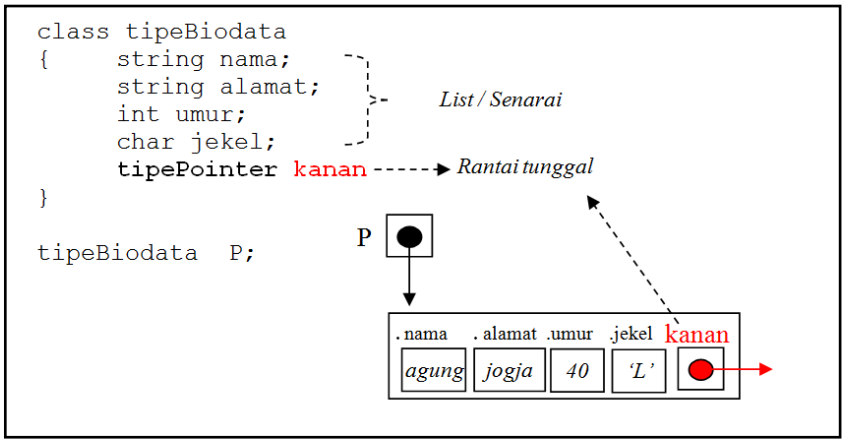
Linked list (daftar terhubung) atau disebut juga *Senarai Berantai,* adalah pengalokasian memori secara dinamis untuk dapat digunakan untuk menyimpan data (list). Mengapa disebut dinamis karena alokasi memori ini dilakukan dengan membuat simpul (*heap)* di memori. Jika pada larik/ array kita hanya dapat menyimpan data dalam jumlah tertentu dan tidak dapat diubah (statis), pada *linked list* kita dapat menyimpan data secara lebih dinamis karena pengalokasian simpul di memori baru akan dilakukan pada saat datanya datang.

D. PENGENALAN SINGLE LINKED LIST (SENARAI BERANTAI TUNGGAL)

Single Linked List adalah kumpulan heap/ obyek/ simpul/ node yang saling terhubung satu sama lain (*linked*) yang dimanfaatkan untuk menyimpan sederet data (*list*) dimana pada setiap heap yang ada terdapat **satu buah pointer anggota (*single*)** yang bertugas sebagai pointer pengait yang digunakan untuk mengkaitkan diri dengan simpul sejenis yang ada di sebelah kanannya. Kebanyakan orang menyingkat Single Linked List (Senarai Berantai Tunggal) hanya dengan sebutan Linked List (Senarai Berantai).



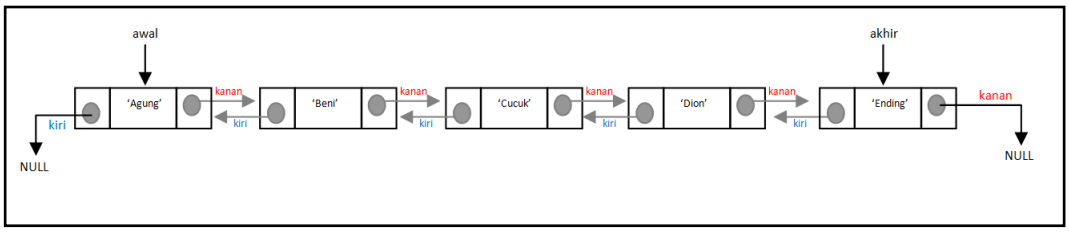
Lalu bagaimana Single Linkedlist diciptakan? Untuk lebih jelasnya perhatikanlah program di bawah ini.



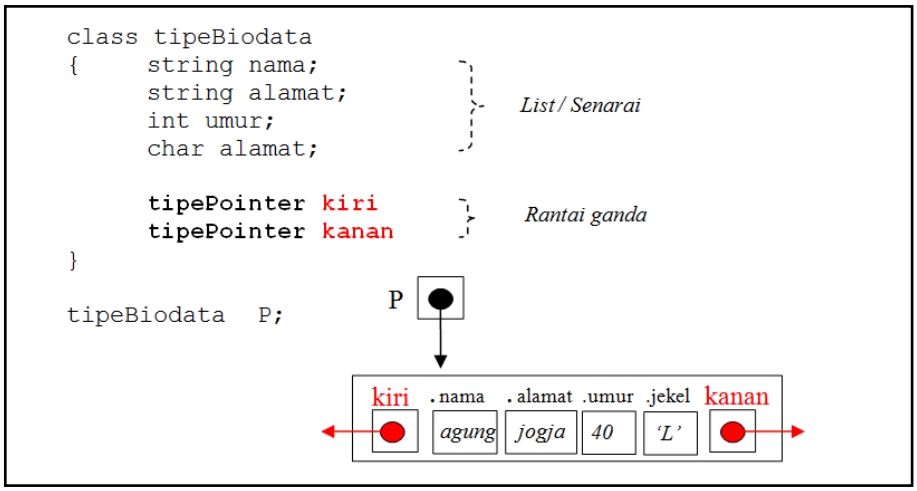
Dari struktur heap di atas terlihat ada 2 kelompok variabel. Pertama adalah List/Senarai yang berisi kumpulan biodata (`nama, alamat, umur` dan `jenis kelamin`), dan yang kedua adalah sebuah rantai penghubung `kanan` yang nantinya akan bertugas untuk merangkaikan setiap heap dengan heap yang lain. Pada single linkedlist rantai ini berjumlah tunggal yaitu `kanan`.

E. PENGENALAN DOUBLE LINKED LIST (SENARAI BERANTAI GANDA)

Double Linked List adalah kumpulan heap/ obyek/ simpul/ node yang saling terhubung satu sama lain (*linked*) yang dimanfaatkan untuk menyimpan sederet data (*list*) dimana pada setiap heap yang ada terdapat **dua buah pointer anggota (*double*)** yang bertugas sebagai pointer pengait yang digunakan untuk mengkaitkan diri dengan simpul sejenis yang ada di sebelah kiri dan kanannya.



Lalu bagaimana Double Linkedlist diciptakan? Untuk lebih jelasnya perhatikanlah program di bawah ini.



Dari struktur heap di atas terlihat ada 2 kelompok variabel. Pertama adalah List/Senarai yang berisi kumpulan biodata (`nama`, `alamat`, `umur` dan `jenis kelamin`), dan yang kedua adalah dua buah rantai penghubung yaitu `kiri` dan `kanan` yang nantinya akan bertugas untuk merangkaikan setiap heap dengan heap yang lain. Pointer `kiri` nantinya akan bertugas menghubungkan heap dengan heap tetangga sebelah kiri sementara pointer `kanan` akan bertugas menghubungkan heap dengan tetangga heap sebelah kanan.

Pembahasan lebih lanjut tentang Double Linked List dapat dilihat pada bab 10

F. MEMANIPULASI ANGGOTA HEAP MILIK POINTER TETANGGA

Pada heap-heap yang telah terhubung dalam sebuah linkedlist, sebuah elemen heap dapat diakses dari pointer lain yang bahkan tidak sedang menunjuk heap tersebut. Sebagai contoh perhatikan program di bawah ini.

```
import java.util.Scanner;
class tipePointer
{  String namaKota;
   tipePointer kanan;
}
class belajarPointer
{  public static void main(String[] args)
   {
      tipePointer P;
      P = new tipePointer();
```

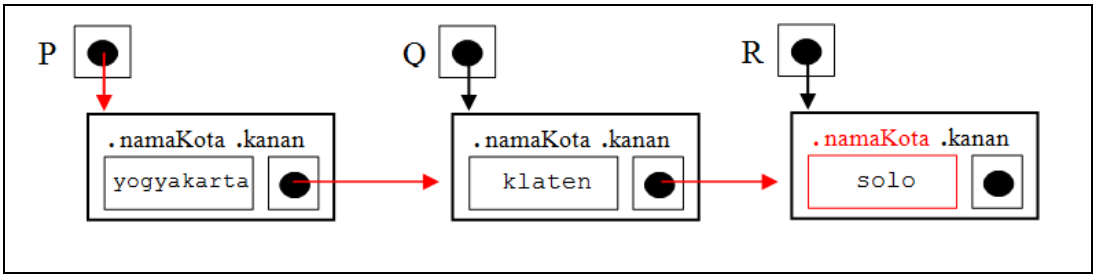
```
      P.namaKota  = "Yogyakarta";

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

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

      P.kanan = Q;
      Q.kanan = R;
      R.kanan = null;
      System.out.println("Nilai elemen namaKota pointer P,Q dan R 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);

      P.kanan.kanan.namaKota = "Surakarta"; <--mengakses R.namaKota dari pointer P

      System.out.println("Nilai elemen namaKota pointer P,Q dan R 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);
  }
}
```

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 terlihat ada upaya untuk mengganti elemen "*Solo*" yang sedang ditunjuk oleh pointer `R` menjadi "*Surakarta*" namun proses penggantiannya dilakukan melalui pointer `P` (perhatikan instruksi `P.kanan.kanan.namaKota = "Surakarta";`) Hal ini mengakibatkan nilai elemen `R.namaKota` akan berubah dari yang semula menampilkan "*Solo*" berubah menjadi "*Surakarta*".

Perhatikan ilustrasi berikut ini.



Pada gambar di atas, selain melalui pointer `R`, elemen `R.namaKota` yang berisi teks "*Solo*" dapat juga diakses melalui pointer "`P.kanan.kanan.namaKota`".

PRAKTIK

1. Praktek 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.

2. Praktek 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.

3. Praktek 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

1. …..
2. .....

TUGAS

1. …..
2. …..

REFERENSI

Disadur dari Buku Ajar Struktur Data Menggunakan Java, Agung Budi Prasetyo, 2017, hal: 66-80, http://agungbudiprasetyo.atspace.com/buku/index.html, diakses pada 12:07 PM 8/06/2019.