IF184982
Pengantar Logika dan Pemrograman
Pertemuan ke-10

Misbakhul Munir IRFAN SUBAKTI
司馬伊凡
Мисбакхул Мунир Ирфан Субакти

Pohon Biner = Binary Tree

- Kelas/*class* rekursif
- Tidak ada pemberian nilai langsung (*assignment*) → semuanya menggunakan fungsi/*function*/*method* rekursif
- Variabel lokal dapat digunakan → deklarasi dan inisialisasi
  - Sekali diinisialisasi, variabel tersebut akan memegang nilai-nilainya sampai akhir dari jangkauan (*scope*) nya. Mereka tidak berubah
- Perulangan (*loop*) juga tidak boleh dilakukan
- Akar dari Pohon memiliki nilai integer
- Memiliki 2 cabang:
  - Kiri → merupakan Pohon
  - Kanan → merupakan Pohon

Pohon.java

```java
package pohon;
/**
 * Kelas Pohon merupakan kelas rekursif. Beberapa constructor dan fungsi/method untuk
 * mengakses dijelaskan dalam kelas ini
 */
public class Pohon {
    protected final boolean kosong;
    protected final int nilai;
    protected final Pohon kiri, kanan;

    /**
     * Buat Pohon baru, di mana nilai akarnya adalah n,
     * cabang pohon ke kiri adalah kiri, cabang pohon ke kanan adalah kanan
     */
    public Pohon (int n, Pohon kiri, Pohon kanan) {
        this.kosong = false;
        this.nilai = n;
        this.kiri = kiri;
        this.kanan = kanan;
    }

    /**
     * Buat Pohon kosong
     */
    public Pohon() {
        this.kosong = true;
        this.nilai = 0;
        this.kiri = null;
        this.kanan = null;
    }

    public Pohon (int n) {
        this.kosong = false;
        this.nilai = n;
        this.kiri = new Pohon();
        this.kanan = new Pohon();
    }

    /**
     * Mengembalikan nilai true jika pohon ini kosong (nilainya nil)
     */
    public boolean adalahKosong() {
        return kosong;
    }

    /**
     * Dapatkan nilai akar dari pohon ini
     */
    public int nilai() {
        if (adalahKosong()) {
            throw new IllegalStateException(
                    "Mencoba untuk mengakses akar dari pohon kosong");
        }
        return nilai;
    }
```

Pohon.java (lanjutan)

```
56
57      /**
58       * Dapatkan cabang ke kiri dari simpul (node) ini
59       */
61      public Pohon kiri() {
62          if (adalahKosong()) {
63              throw new IllegalStateException(
64                      "Mencoba untuk mengakses cabang dari pohon kosong");
65          }
66          return kiri;
67      }
68
69      /**
70       * Dapatkan cabang ke kanan dari simpul (node) ini
72       */
73      public Pohon kanan() {
74          if (adalahKosong()) {
75              throw new IllegalStateException(
76                      "Mencoba untuk mengakses cabang dari pohon kosong");
77          }
78          return kanan;
      }
```

```
79      /**
80       * Buat String dalam baris jamak untuk menampilkan pohon ini. Formatnya adalah sbb;
81   <code>
82   10
83      |
84      |- 14
85      |   |
86      |   |- 17
87      |   |
88      |   |- 13
89      |       |
90      |       | - [kosong]
91      |       |
92      |       |- 12
93      |
94      |- 6
95   </code>
96       * Di mana anak pohon paling bawah adalah cabang pohon kiri dan anak pohon paling atas
97       * adalah cabang pohon kanan Jika kedua anak pohon adalah pohon kosong (nil) maka
98       * mereka tidak akan dicetak. Jika hanya satu anak pohon yang kosong, maka mereka akan
99       * dicetak sehingga dapat diketahui mana yang anak pohon kanan, mana yang anak
100      * pohon kiri.
         * @param pohon Obyek pohon yang boleh jadi tidak boleh null
102      * @return String yang berisi pohon yang terformat
103      */
105     @Override public String toString() {
106         return PohonAksi.toString(this);
107     }
```

```
109     @Override public boolean equals(Object o) {
110         Pohon p = (Pohon) o;
111         if (kosong) {
112             return p.kosong;
113         } else {
114             return !p.kosong && nilai == p.nilai &&
115                     kiri.equals(p.kiri) && kanan.equals(p.kanan);
116         }
117     }
118 }
```

PohonAksi.java

- Fungsi/*function*/*method* dan contoh bagaimana mengelola kelas Pohon
- Mendayagunakan ListKu.java dan ListKuFungsi.java

```java
package pohon;

import java.util.Arrays;
import java.util.Collections;
import java.util.Iterator;
import java.util.LinkedList;
import uts.ListKu;
import uts.ListKuFungsi;

/**
 * Definisi beberapa fungsi/method dan contoh bagaimana mengelola pohon, termasuk juga
 * fungsi/method cetak (print)
 */
public class PohonAksi extends Pohon {

    /**
     * Menghitung tinggi dari pohon
     *@return tinggi dari pohon
     */
    public static int tinggi(Pohon p) {
        if (p.adalahKosong())
            return 0;
        else {
            int kiriTinggi = tinggi(p.kiri());
            int kananTinggi = tinggi(p.kanan());

            return Math.max(kiriTinggi, kananTinggi) + 1;
        }
    }
```

PohonAksi.java (lanjutan)

```
30
31      /**
32       * Mendapatkan nilai pohon dengan masing-masing elemen dikalikan 3 (tiga)
33       * dari input Pohon
34       */
35      public static Pohon tigaKali(Pohon p) {
36          if (p.adalahKosong()) {
37              return new Pohon();
38          } else {
39              return new Pohon(3 * p.nilai(), tigaKali(p.kiri()), tigaKali(p.kanan()));
40          }
41      }
42
43      /**
44       * Mendapatkan nilai ListKu yang diperoleh dari penelusuran Pohon secara
45       * INORDER, yaitu: cabang kiri dulu, lalu akar/simpul saat ini, lalu cabang kanan
46       */
47      public static ListKu inorder(Pohon p) {
48          if (p.adalahKosong()) {
49              return ListKu.kosong();
50          } else {
51              ListKu kiriElemen = inorder(p.kiri());
52              ListKu kananElemen = inorder(p.kanan());
53              return ListKuFungsi.sambung(kiriElemen, ListKu.buat(p.nilai(), kananElemen));
54          }
55      }
```

```
56
57      /**
58       * Sumber (dengan perubahan seperlunya) http://www.connorgarvey.com/blog/?p=82
59       * Cetak representasi terformat dari pohon yang ingin dicetak. Formatnya adalah sbb.
60  <code>
61  10
62      |
63      |- 14
64      |   |
65      |   |- 17
66      |   |
67      |   |- 13
68      |       |
69      |       | - [kosong]
70      |       |
71      |       |- 12
72      |
73      |- 6
74  </code>
75
76       * Di mana anak pohon paling bawah adalah cabang pohon kiri dan anak pohon paling atas
77       * adalah cabang pohon kanan Jika kedua anak pohon adalah pohon kosong (nil) maka
78       * mereka tidak akan dicetak. Jika hanya satu anak pohon yang kosong, maka mereka akan
79       * dicetak sehingga dapat diketahui mana yang anak pohon kanan, mana yang anak
80       * pohon kiri.
81       * @param pohon Obyek pohon yang boleh jadi tidak boleh null
82       */
83      public static void cetak(Pohon pohon) {
84          System.out.print(PohonAksi.toString(pohon));
85      }
```

PohonAksi.java (lanjutan)

```
86
87      public static String toString(Pohon pohon) {
88          final StringBuilder buffer = new StringBuilder();
89          return toStringPohonBantu(pohon, buffer,
                                      new LinkedList<Iterator<Pohon>>()).toString();
91      }
92
93      private static String toStringPohonBuatBaris(
                              java.util.List<Iterator<Pohon>> indukIterators,
95                            boolean terakhir) {
96          StringBuilder hasil = new StringBuilder();
97          Iterator<Iterator<Pohon>> itr = indukIterators.iterator();
98          while (itr.hasNext()) {
99              Iterator<Pohon> it = itr.next();
100             if (it.hasNext() || (!itr.hasNext() && terakhir)) {
101                 hasil.append("   |");
102             } else {
103                 hasil.append("    ");
104             }
105         }
106         return hasil.toString();
107     }
```

```
108
109     private static StringBuilder toStringPohonBantu(
                              Pohon p, StringBuilder buffer,
111                           java.util.List<Iterator<Pohon>> indukIterators) {
112
113         if (!indukIterators.isEmpty()) {
114             boolean terakhir =
115                 !indukIterators.get(indukIterators.size() - 1).hasNext();
116             String baris = toStringPohonBuatBaris(indukIterators, terakhir);
117             buffer.append("\n").append(baris).append("\n").
118                 append(baris).append("- ");
119         }
120
121         if (p.adalahKosong()) {
122             buffer.append("[kosong]");
123             return buffer;
124         }
125         else
126             buffer.append(p.nilai());
127
128         if (!(p.kiri().adalahKosong() && p.kanan().adalahKosong())) {
129             Iterator<Pohon> it = getAnakIterator(p);
130             indukIterators.add(it);
131             while (it.hasNext()) {
132                 Pohon anak = it.next();
133                 toStringPohonBantu(anak, buffer, indukIterators);
134             }
135             indukIterators.remove(it);
136         }
137         return buffer;
138     }
```

```
139
140     private static Iterator<Pohon> getAnakIterator(Pohon p) {
141         if (p.adalahKosong()) {
142             return Collections.<Pohon>emptyList().iterator();
143         } else {
144             return Arrays.asList(new Pohon[] {
145                     p.kanan(),
146                     p.kiri()}).iterator();
147         }
148     }
149
150 }
```

PohonAksiTest.java

- Cek fungsi/*function*/*method* yang ada di Pohon.java dan PohonAksi.java

```
1   package pohon;
2   class PohonAksiTest extends UAS1 {
3       public static void main (String args[]) {
4           // ==========
5           // Pohon.java
6           // ==========
7           // pohon1 -> pohon kosong
8           System.out.println("Buat pohon1 -> pohon baru = kosong");
9           Pohon pohon1 = new Pohon();
10          // nilai()
11          System.out.println("Nilai dari pohon1 adalah " + pohon1.nilai);
12          System.out.println("pohon1: " + pohon1);
13          // pohon1 -> adalahKosong()
14          System.out.println("Cek jika pohon1 adalah pohon kosong");
15          if (pohon1.adalahKosong()) {
16              System.out.println("pohon1 adalah pohon kosong");
17          } else {
18              System.out.println("pohon1 bukan pohon kosong");
19          }
20          // pohon2 -> 9
21          System.out.println("Buat pohon2 -> pohon baru dengan akar bernilai 9");
22          Pohon pohon2 = new Pohon(9);
23          System.out.println("Nilai dari pohon2 adalah " + pohon2.nilai);
24          System.out.println("pohon2: " + pohon2);
25          // pohon2 -> adalahKosong()
26          System.out.println("Cek jika pohon2 adalah pohon kosong");
27          if (pohon2.adalahKosong()) {
28              System.out.println("pohon2 adalah pohon kosong");
29          } else {
30              System.out.println("pohon2 bukan pohon kosong");
31          }
```

PohonAksiTest.java (lanjutan)

```
32              // pohon3 -> 12
33              System.out.println("Buat pohon3 -> pohon baru dengan akar bernilai 12");
34              Pohon pohon3 = new Pohon(12);
35              System.out.println("Nilai dari pohon3 adalah " + pohon3.nilai);
36              System.out.println("pohon3: " + pohon3);
37              // pohon4 -> 15
38              System.out.println("Buat pohon4 -> pohon baru dengan akar bernilai 15");
39              Pohon pohon4 = new Pohon(15);
40              System.out.println("pohon4: " + pohon4);
41              // pohon5 -> akar bernilai 13 dengan cabang kiri pohon3 (12) dan cabang
42              // kanan pohon4 (15)
43              System.out.println("Buat pohon5 -> pohon baru dengan akar bernilai 13");
44              System.out.println("dengan cabang kiri pohon3 (12) & cabang kanan pohon4 (15)");
45              Pohon pohon5 = new Pohon(13, pohon3, pohon4);
46              System.out.println("Nilai dari pohon5 adalah " + pohon5.nilai);
47              System.out.println("pohon5:");
48              System.out.println(pohon5);
49              // kiri
50              System.out.println("Cabang kiri dari pohon5 adalah:" + pohon5.kiri);
51              // kanan
52              System.out.println("Cabang kanan dari pohon5 adalah:" + pohon5.kanan);
53              // pohon6 -> 15
54              System.out.println("Buat pohon6 -> pohon baru dengan akar bernilai 15");
55              Pohon pohon6 = new Pohon(15);
56              System.out.println("pohon6: " + pohon6);
57              // Bandingkan antara pohon4 yang bernilai 15 dengan pohon6 yang juga bernilai 15
58              if (pohon4.equals(pohon6)) {
59                  System.out.println("pohon4 sama nilainya dengan pohon6");
60              } else {
61                  System.out.println("pohon4 tidak sama nilainya dengan pohon6");
62              }
```

```
63              // ==============
64              // PohonAksi.java
65              // ==============
66              // tinggi()
67              System.out.println("Tinggi dari pohon5 adalah " + PohonAksi.tinggi(pohon5));
68              // tigaKali()
69              System.out.println("tigaKali() dari pohon5 adalah:");
70              System.out.println(PohonAksi.tigaKali(pohon5));
71              // inorder
72              System.out.println("inorder() dari pohon5 adalah:" + PohonAksi.inorder(pohon5));
73              // cetak()
74              System.out.println("cetak() dari pohon5 adalah:");
75              PohonAksi.cetak(pohon5);
76              System.out.println();
77          }
78      }
```

PohonAksiTest: Hasil

```
run:
Buat pohon1 -> pohon baru = kosong
Nilai dari pohon1 adalah 0
pohon1: [kosong]
Cek jika pohon1 adalah pohon kosong
pohon1 adalah pohon kosong
Buat pohon2 -> pohon baru dengan akar bernilai 9
Nilai dari pohon2 adalah 9
pohon2: 9
Cek jika pohon2 adalah pohon kosong
pohon2 bukan pohon kosong
Buat pohon3 -> pohon baru dengan akar bernilai 12
Nilai dari pohon3 adalah 12
pohon3: 12
Buat pohon4 -> pohon baru dengan akar bernilai 15
pohon4: 15
Buat pohon5 -> pohon baru dengan akar bernilai 13
dengan cabang kiri pohon3 (12) & cabang kanan pohon4 (15)
Nilai dari pohon5 adalah 13
pohon5:
13
    |
    |- 15
    |
    |- 12
Cabang kiri dari pohon5 adalah:12
Cabang kanan dari pohon5 adalah:15
Buat pohon6 -> pohon baru dengan akar bernilai 15
pohon6: 15
pohon4 sama nilainya dengan pohon6
Tinggi dari pohon5 adalah 2
tigaKali() dari pohon5 adalah:
39
    |
    |- 45
    |
    |- 36
inorder() dari pohon5 adalah:[12, 13, 15]
cetak() dari pohon5 adalah:
13
    |
    |- 15
    |
    |- 12
BUILD SUCCESSFUL (total time: 0 seconds)
```

Soal 1

- Negasikan semua elemen dalam pohon

```
public static Pohon negasiSemua (Pohon p)
```

- Dengan masukan pohon integer `p`, tuliskan fungsi yang mengembalikan pohon baru dari `p` di mana semua tanda elemen di dalamnya dinegasikan, integer positif menjadi integer negatif dan sebaliknya integer negatif menjadi integer positif

Soal 1 (lanjutan)

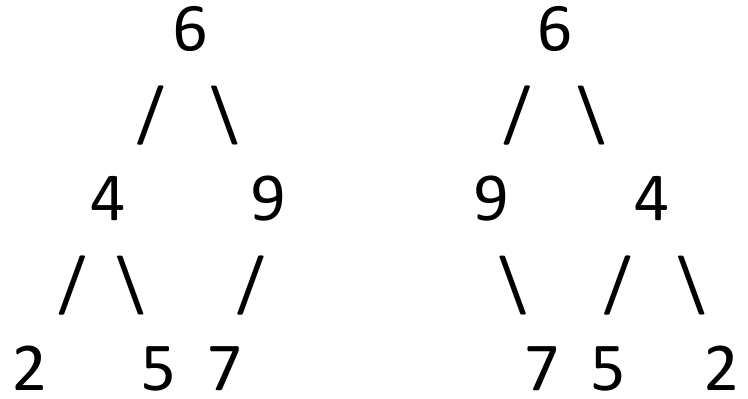
```
1   package pohon;
2
3   import uts.ListKu;
4   import uts.ListKuFungsi;
5
6   public class UAS1 extends Pohon {
7
8       // Soal 1
9
10      public static Pohon negasiSemua (Pohon p) {
11          if (p.adalahKosong()) {
12              return p;
13          } else {
14              return new Pohon (
15                      - p.nilai(),
16                        negasiSemua(p.kiri()),
17                        negasiSemua(p.kanan() )
18              );
19          }
20      }
```

Soal 2

- Cerminkan semua elemen dalam pohon

```
public static Pohon cermin (Pohon p)
```

- Dengan masukan pohon integer `p`, tuliskan fungsi yang mengembalikan cerminan sepanjang sumbu kiri-kanan dari semua elemen di pohon `p`. Contoh seperti di bawah. Pohon di kiri adalah cerminan pohon di kanan dan sebaliknya.



Soal 2 (lanjutan)

```
22      // Soal 2
23
24      public static Pohon cermin (Pohon p) {
25          if (p.adalahKosong()) {
26              return p;
27          } else {
28              return new Pohon (p.nilai(),
29                      p.kanan(), // kanan pada posisi sebelah kiri
30                      p.kiri()   // sedangkan kiri pada posisi sebelah kanan
31              );
32          }
33      }
```

Soal 3

- Penelusuran `postorder`

```
public static ListKu postorder (Pohon p)
```

- Dengan masukan pohon integer `p`, tuliskan fungsi yang mengembalikan list yang memiliki nilai-nilai dari `p` dengan penelusuran simpul-simpul dengan cara `postorder`. `Postorder` yaitu dari penelusuran dari setiap simpul/node, maka semua nilai dari cabang kiri pohon akan ditelusuri dulu, lalu semua nilai dari cabang kanan pohon ditelusuri, baru terakhir telusuri nilai di simpul itu sendiri.
- Contoh kode program penelusuran dengan cari `inorder` dari kelas PohonAksi.java di atas dapat dijadikan acuan

Soal 3 (lanjutan)

```
35      // Soal 3
36
37      /**
38       * POSTORDER, penelusuran ke:
39       *  - kiri
40       *  - kanan
41       *  - simpul/node ini
42       *
43       * Jika kita lebih dulu menelusuri cabang kiri, maka kita perlu menyambung/menambahkan
44       * elemen baru ke list di saat kita menelusuri cabang kanan.
45       * Menyambung (fungsi sambung) membutuhkan pencarian sepanjang list untuk
46       * sampai di titik akhir. Maka hasil dari kode POSTORDER PELAN
47       * memerlukan waktu O(n^2)
48       *
49       * Untuk menanggulangi hal ini, dibuat fungsi dengan parameter akumulasi
50       * yang mencatat list yang didapat dari sisa pohon. Hal ini membutuhkan
51       * waktu O(n)
52       */
53
54      public static ListKu postorderPelan (Pohon p) {
55          if (p.adalahKosong()) {
56              return ListKu.kosong();
57          } else {
58              return ListKuFungsi.sambung(postorder(p.kiri()),
59                                          ListKuFungsi.sambung(postorder(p.kanan()),
60                                                               ListKu.buat(p.nilai(), ListKu.kosong())));
61          }
62      }
```

Soal 3 (lanjutan)

```
64      public static ListKu postorder (Pohon p) {
65          return postorder(p, ListKu.kosong());
66      }
67
68      // Fungsi bantuan untuk postorder (Pohon p).
69      // Tambahkan list penelusuran postorder dari p pada posidi di depan lst
70
71      private static ListKu postorder (Pohon p, ListKu lst) {
72          if ( p.adalahKosong()) {
73              return lst;
74          } else {
75              return postorder(p.kiri(),
76                      postorder(p.kanan(),
77                              ListKu.buat(p.nilai(), lst)
78                      )
79              );
80          }
81      }
```

Soal 4

`public boolean semuaPositif (Pohon p)`

- Dengan masukan pohon integer `p`, tuliskan fungsi yang mengembalikan nilai `boolean` (`true` atau `false`) yang mengindikasikan apakah semua nilai di dalam simpul-simpul pada `p` semuanya positif, yaitu ≥ 0

Soal 4 (lanjutan)

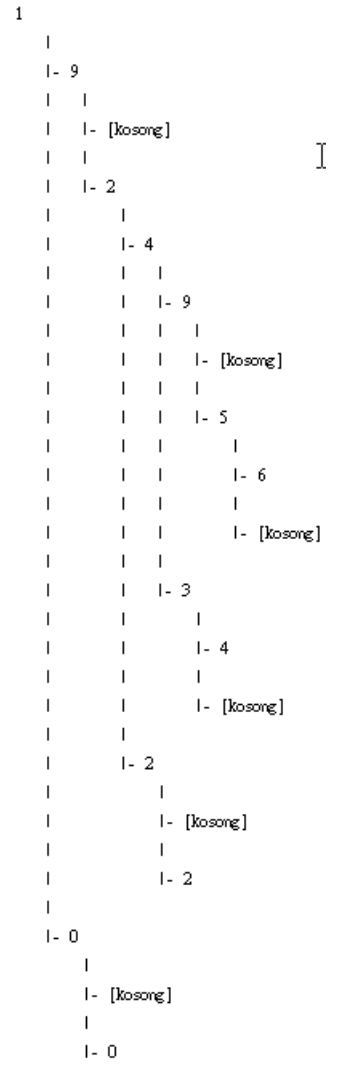
```
83      // Soal 4
84
85      /**
86       *  Jika simpul ini bukan positif, segera keluar dari iterasi rekursif.
87       */
        public static boolean semuaPositif (Pohon p) {
89          if (p.adalahKosong() ) {
90              return true;
91          } else {
92              /*
93               * Perlu diingat bahwa "nilai" haruslah dicek pertama kali
94               * sebab hal ini dipandang sebagai langkah pendek yang menguntungkan
95               *
96               * Alternatifnya if dapat digunakan
97               */
98              return p.nilai() >= 0 &&
99                      semuaPositif(p.kiri()) &&
100                     semuaPositif(p.kanan());
101         }
102     }
```

Fungsi Konversi Array ke Pohon

- Konversikan `Array` ke `Pohon`

```
private static Pohon arrayKePohon
(int a[])
```

- Memudahkan dalam pengelolaan masukan (`input`) dari bentukan `Array` ke bentukan `Pohon`
- Contoh:
  - Array → [1, 9, 2, 4, 0, 2, 9, 3, 0, 2, 4, 5, 6] → Pohon →



Fungsi Konversi Array ke Pohon

```
63  private static Pohon arrayKePohon (int a[]) {
64      Pohon p = new Pohon();
65      for (int i = 0; i < a.length; i++) {
66          p = sisipkan(a[i], p);
67      }
68      return p;
69  }
70
71  private static Pohon sisipkan(int n, Pohon p) {
72      if (p.adalahKosong()) {
73          return new Pohon(n, new Pohon(), new Pohon());
74      } else if (n <= p.nilai()) {
75          return new Pohon(p.nilai(), sisipkan(n, p.kiri()), p.kanan());
76      } else {
77          return new Pohon(p.nilai(), p.kiri(), sisipkan(n, p.kanan()));
78      }
79  }
```

Soal 1-4: Test

```java
package pohon;

import uts.ListKu;

class UAS1Test extends UAS1 {
    public static void main (String args[]) {
        Pohon pohon = arrayKePohon(new
                int[] {1, 9 ,2, 4, 0, 2, 9, 3, 0, 2, 4, 5, 6});

        cetak("pohon asli: Array ke Pohon", pohon);

        System.out.println("Soal 1: negasiSemua");
        cetak("negasiSemua", negasiSemua(pohon));

        System.out.println("Soal 2: cermin");
        cetak("cermin", cermin(pohon));

        System.out.println("Soal 3: postorder");
        ListKu list = postorder(pohon);
        for (ListKu l = list; !l.adalahKosong(); l = l.ekor()) {
            System.out.println("- " + l.kepala());
        }

        System.out.println("Soal 4: semuaPositif");
        if (semuaPositif(pohon)) {
            System.out.println("Semua elemen dalam pohon bernilai positif");
        } else {
            System.out.println("Tidak elemen dalam pohon bernilai positif");
        }
    }

    static void cetak (String judul) {
        System.out.println( "----------------------------" );
        System.out.println( "# " + judul + " #");
        System.out.println();
    }

    static void cetak (String judul, Pohon p) {
        cetak(judul);
        cetak(p);

        System.out.println();
    }

    static void cetak (String judul, ListKu list) {
        cetak(judul);
        cetak(list);

        System.out.println();
    }

    static void cetak (Pohon p) {
        PohonAksi.cetak(p);
        System.out.println();
    }

    static void cetak (ListKu list ) {
        for (ListKu l = list; ! l.adalahKosong(); l = l.ekor() ) {
            System.out.println( "- " + l.kepala() );
        }
    }
```

Soal 1-4: Test (lanjutan)

```
62
63      private static Pohon arrayKePohon (int a[]) {
64          Pohon p = new Pohon();
65          for (int i = 0; i < a.length; i++) {
66              p = sisipkan(a[i], p);
67          }
68          return p;
69      }
70
71      private static Pohon sisipkan(int n, Pohon p) {
72          if (p.adalahKosong()) {
73              return new Pohon(n, new Pohon(), new Pohon());
74          } else if (n <= p.nilai()) {
75              return new Pohon(p.nilai(), sisipkan(n, p.kiri()), p.kanan());
76          } else {
77              return new Pohon(p.nilai(), p.kiri(), sisipkan(n, p.kanan()));
78          }
79      }
80
81  }
```

Soal 1-4: Hasil Test