IF184401 Design & Analysis of Algorithms (E)

Midterm Exam

Starting date:	7 March 2020
Deadline:	14 March 2020, 23:59 WIB. **Penalty: 0.15% of grade/minute of tardiness**.
Exam type:	Open
Send to:	
	MM Irfan Subakti <yifana@gmail.com> CC to Christopher Andrew <andrew.public107@gmail.com> & Karina Soraya Puspitasari <karinasoraya.ks@gmail.com> with the subject: IF184401_DAA(E)_MID_StudentID_Name
File type and format:	A full report of the source code, output and analysis; in PDF format. Put this report along with your declaration and all of .java files (or any other programming language's source code) into 1 (one) .ZIP file.
Filename format:	IF184401_DAA(E)_MID_StudentID_Name.ZIP

Instruction
Please do these steps as in the following.

1. Use these following codes/files as the building blocks for the next steps. You have to implement these codes either in Java or in any other programming language. File: MidtermExam.zip can be downloaded from subakti.com > DAA(D) > Midterm exam: the source code. It uses NetBeans as IDE for Java programming, however, please feel free to use any IDE/programming language that suits you. MidtermExam.zip has 1 folder: `MidtermExam`. Inside folder `MidtermExam`, `src` folder will be found. Inside `src` folder, there are two folders/`packages`: `pohon` and `uts`. Inside folder/`package`: `uts`, there are 3 files: `ListKu.java`, `ListKuFungsi.java` and `ListKuFungsiTest.java`. Inside folder/`package`:`pohon`, there are 8 files: `Pohon.java`, `PohonAksi.java`, `PohonAksiTest.java`, `PohonAVL.java`, `DAA1.java`, `DAA1Test.java`, `DAA2.java` and `DAA2Test.java`. Your task is to update and continue for writing codes for the file `DAA1.java` and `DAA2.java`. You can test your code by running the file `DAA1Test.java` and `DAA2Test.java`.

File: 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;
    }

```

```
32      public Pohon (int n) {
33          this.kosong = false;
34          this.nilai = n;
35          this.kiri = new Pohon();
36          this.kanan = new Pohon();
37      }
38
39      /**
40       * Mengembalikan nilai true jika pohon ini kosong (nilainya nil)
41       */
43      public boolean adalahKosong() {
44          return kosong;
45      }
46
47      /**
48       * Dapatkan nilai akar dari pohon ini
50       */
51      public int nilai() {
52          if (adalahKosong()) {
53              throw new IllegalStateException(
54                      "Mencoba untuk mengakses akar dari pohon kosong");
55          }
56          return nilai;
57      }
58
59      /**
61       * Dapatkan cabang ke kiri dari simpul (node) ini
62       */
63      public Pohon kiri() {
64          if (adalahKosong()) {
65              throw new IllegalStateException(
66                      "Mencoba untuk mengakses cabang dari pohon kosong");
67          }
68          return kiri;
69      }
70
72      /**
73       * Dapatkan cabang ke kanan dari simpul (node) ini
74       */
75      public Pohon kanan() {
76          if (adalahKosong()) {
77              throw new IllegalStateException(
78                      "Mencoba untuk mengakses cabang dari pohon kosong");
          }
          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.
101      * @param pohon Obyek pohon yang boleh jadi tidak boleh null
102      * @return String yang berisi pohon yang terformat
103      */
104     @Override public String toString() {
105         return PohonAksi.toString(this);
106     }
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
}
```

File: ListKu.java

```java
package uts;
/**
 * Kelas ListKu mendefinisikan tipe rekursif dari ListKu
 * Menyediakan constructor dan fungsi/metode pengaksesan
 */
public class ListKu {
    protected boolean kosong;
    protected int kepala;
    protected ListKu ekor;
    public ListKu(int kepala, ListKu ekor) {
        kosong = false;
        this.kepala = kepala;
        this.ekor = ekor;
    }
    public ListKu() {
        kosong = true;
    }
    /**
     * Buat list baru dari elemen kepala dan ListKu ekor
     * @param kepala adalah elemen yang ditambahkan di awal list
     * @param ekor adalah bagian akhir list yang berupa list
     * @return ListKu adalah tipe pengembalian fungsi
     */
    public static ListKu buat(int kepala, ListKu ekor) {
        return new ListKu(kepala, ekor);
    }
    /**
     * Buat ListKu kosong yang baru
     * @return ListKu adalah tipe pengembalian fungsi
     */
    public static ListKu kosong() {
        return new ListKu();
    }
```

```
34      /**
35       * Mengecek kondisi list, memberikan nilai true jika list-nya memang kosong
36       * @return kosong adalah tipe boolean: true atau false
37       */
38      public boolean adalahKosong() {
39          return kosong;
40      }
41      /**
42       * Mendapatkan nilai kepala dari list ini, atau memberikan eksepsi/protes jika list-nya kosong
43       * @return kepala bernilai bulat (integer) sebagai nilai yang didapat
44       * @throws IllegalStateException jika list-nya kosong
45       */
46      public int kepala() {
47          if (adalahKosong()) {
48              throw new IllegalStateException("Coba akses kepala dari list yang kosong");
49          }
50          return kepala;
51      }
52      /**
53       * Mendapatkan ekor dari list yang berupa list, atau memberikan eksepsi jika list-nya kosong
54       * @return ekor bernilai list
55       * @throws IllegalStateException jika list-nya kosong
56       */
57      public ListKu ekor() {
58          if (adalahKosong()) {
59              throw new IllegalStateException("Coba akses ekor dari list yang kosong");
60          }
61          return ekor;
62      }
```

```
63      /**
64       * Mendapatkan representasi/isi dari list, contoh: "[1, 2, 3]"
65       * @return toString bernilai String
66       */
67      @Override
68      public String toString() {
69          return toStringBantuan(this, new java.util.ArrayList<Integer>()); // ArrayList of Integer
70      }
71      private String toStringBantuan(ListKu list, java.util.List<Integer> listSkr) {
72          if (list == null) {
73              throw new IllegalStateException(
74                      "Elemen berikutnya dari list adalah null. " +
75                      "Seharusnya adalah list yang lain atau nil. " +
76                      "List sampai saat ini: " + listSkr);
77          } else if (list.adalahKosong()) {
78              return listSkr.toString();
79          } else {
80              listSkr.add(list.kepala());
81              return toStringBantuan(list.ekor(), listSkr);
82          }
83      }
84  }
```

File: ListKuFungsi.java

```java
package uts;
/**
 * Kelas ListKuFungsi mendefinisikan sejumlah fungsi static yang bekerja untuk list
 * menggunakan kelas ListKu.
 */
public class ListKuFungsi extends ListKu {
    /**
     * Pilih dan dapatkan nilai elemen ke-n dari list a.
     * Asumsikan bahwa elemen ke-n ada.
     * @param n indeks elemen yang akan dipilih
     * @param a adalah list yang diakses
     * @return Integer (bilangan bulat), yaitu bilangan yang terpilih
     */
    public static int pilih(int n, ListKu a) {
        if (a.adalahKosong()) {
            throw new IllegalStateException(
                    "Pilih - list tidak memiliki elemen sama sekali.");
        } else if (n == 0) {
            return a.kepala();
        } else {
            return pilih(n-1, a.ekor());
        }
    }

    /**
     * Mendapatkan elemen terakhir dari list a.
     * Asumsikan bahwa paling tidak ada satu elemen dalam list a
     * @param a adalah list yang diakses
     * @return  Integer (bilangan bulat), yaitu bilangan yang didapatkan
     */
    public static int terakhir(ListKu a) {
        if (a.adalahKosong()) {
            throw new IllegalStateException("List tidak punya elemen sama sekali.");
        } else if (a.ekor().adalahKosong()) {
            return a.kepala();
        } else {
            return terakhir(a.ekor());
        }
    }
    /**
     * Tambahkan satu elemen pada akhir dari list a.
     * Return the extended list.
     * @param a adalah list yang akan ditambahkan
     * @param e adalah elemen yang ditambahkan
     * @return ListKu adalah tipe pengembalian fungsi
     */
    public static ListKu tambahAkhir(ListKu a, int e) {
        if (a.adalahKosong()) {
            return buat(e, kosong());
        } else {
            return buat(a.kepala(), tambahAkhir(a.ekor(), e));
        }
    }
}
```

```
53      /**
54       * Buat sebuah list yang merupakan hasil dari penyambungan list b yang
55       * disambungkan ke akhir dari list a.
56       * @param a adalah list pertama yang akan disambungkan sebagai list awal
57       * @param b adalah list kedua yang akan disambungkan ke akhir dari list a
58       * @return ListKu adalah tipe pengembalian fungsi
59       */
60      public static ListKu sambung(ListKu a, ListKu b) {
61          if (a.adalahKosong()) {
62              return b;
63          } else {
64              return buat(a.kepala(), sambung(a.ekor(), b));
65          }
66      }
67      /**
68       * tambahAkhir juga dapat didefinisikan menggunakan sambung tanpa perlu rekursif.
69      public static ListKu tambahAkhir(ListKu a, int e) {
70          return sambung(a, buat(e, kosong()));
71      }
72      */
73      /**
74       * Implementasi dari pembalikan urutan elemen list secara sederhana.
75       * Membutuhkan waktu lama untuk list yang besar
76       * @param a adalah list yang ingin dibalik urutan elemennya
77       * @return ListKu adalah tipe pengembalian fungsi
78       */
79      public static ListKu pembalikanSederhana(ListKu a) {
80          if (a.adalahKosong()) {
81              return kosong();
82          } else {
83              return tambahAkhir(pembalikanSederhana(a.ekor()), a.kepala());
84          }
85      }
```

```
86      /**
87       * Implementasi yang efisien dari pembalikan urutan elemen list yang menggunakan
88       * fungsi pembantu dan pengumpul
89       * @param list adalah list yang ingin dibalik urutan elemennya
90       * @return ListKu adalah tipe pengembalian fungsi
91       */
92      public static ListKu pembalikan(ListKu list) {
93          return balikKumpul(list, kosong());
94      }
95      private static ListKu balikKumpul(ListKu asli, ListKu balikan) {
96          if (asli.adalahKosong()) {
97              return balikan;
98          }
99          else {
100             return balikKumpul(asli.ekor(), buat(asli.kepala(), balikan));
101         }
102     }
103 }
```

File: PohonAksi.java

```
1   package pohon;
2
3   import java.util.Arrays;
4   import java.util.Collections;
5   import java.util.Iterator;
6   import java.util.LinkedList;
7   import uts.ListKu;
8   import uts.ListKuFungsi;
9
10  /**
11   * Definisi beberapa fungsi/method dan contoh bagaimana mengelola pohon, termasuk juga
12   * fungsi/method cetak (print)
13   */
14  public class PohonAksi extends Pohon {
15
16      /**
17       * Menghitung tinggi dari pohon
18       *@return tinggi dari pohon
19       */
20      public static int tinggi(Pohon p) {
21          if (p.adalahKosong())
22              return 0;
23          else {
24              int kiriTinggi = tinggi(p.kiri());
25              int kananTinggi = tinggi(p.kanan());
26
27              return Math.max(kiriTinggi, kananTinggi) + 1;
28          }
29      }
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      }
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(
94                                  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(
110                     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 }
```

2. Based on `Pohon.java` and `PohonAksi.java` above, please create a function, namely `adalahBST()` which has a *recursive* function inside this function. It checks whether a tree, i.e., `Pohon p`, is BST (*Binary Search Tree*). **Hint**: it is allowed to use a supported function for `adalahBST()`. Please update the function `adalahBST()` in file `DAA1.java`. **[20 points]**

Function name: `public static boolean adalahBST (Pohon p)`
Supported function name:
`private static Boolean adalahBST(Pohon p, int batasBawah, int batasAtas)`

Supported function `adalahBST()` will be used to get a Boolean value whether `p` is BST or not where its value can be found between the range of `batasBawah` and `batasAtas`.

3. Please create a recursive function, namely `cetakMenurun()` which receives an input of a BST `p` (where `p` is `Pohon` and its values are an integer), that be able to print the values of `p` in *descending order*. This function has to be created without making a separate list of values from `p`. Please update the function `cetakMenurun()` in file `DAA1.java`. **[10 points]**

Function name: `public static void cetakMenurun (Pohon p)`

4. Please create an efficient recursive function, namely `maks()` which receives an input of a BST `p` (where `p` is `Pohon` and its values are an integer), that be able to get the maximum value of the `p`'s values. It is not allowed to traverse and to compare all of the nodes in the tree. However, you should traverse at most one path in the tree from the root. It means this function works in $O(\log n)$ time for BST. **Hint**: assume we are a node `x` in BST, then all of the values from the tree's left branches of `x` always have the less than or equal (≤) values compared to the value of node `x`. So, the maximum value won't be existing in the tree's left branches. Where is the maximum value? Please update the function `maks()` in file `DAA1.java`. **[10 points]**

Function name: `public static int maks (Pohon p)`

5. Please create a function, namely `hapus()` which receives the inputs of `Pohon p` and `int n`, whose has a recursive function to delete the value of *n* from the tree `p`, if *n* exists. If *n* does not exist, then return a new tree. Please update the function `hapus()` in file `DAA1.java`. **[10 points]**

Function name: `public static Pohon hapus (Pohon p, int n)`
Pseudo-code:

```
public static Pohon hapus (Pohon p, int n)
        Jika p adalah kosong maka
                kembalikan: p
        Yang lain
                Buat variabel, misal a <-- nilai dari p
                Buat variabel, misal Ki <-- kiri dari p
                Buat variabel, misal Ka <-- kanan dari p
                Jika n lebih besar dari a
                        Buat variabel, misal baruKa <-- hapus(Ka, n)
                        Kembalikan: buat Pohon baru dengan parameter: nilai dari p, Ki, baruKa
                Jika n lebih kecil dari a
                        Buat variabel, misal baruKi <-- hapus(Ki, n)
                        Kembalikan: buat Pohon baru dengan parameter: nilai dari p, baruKi, Ka
                Yang lain
                        Jika Ki adalah kosong
                                Kembalikan: Ka
                        Yang lain jika Ka adalah kosong
                                Kembalikan: Ki
                        // Kedua cabang tidak kosong
                        // Maka salah satu cabang akan kehilangan satu node
                        // Misal di cabang kiri -> maks dari cabang kiri
                        // Atau misal di cabang kanan -> min dari cabang kanan
                        Yang lain
                                Buat variabel, misal, sebelum <-- maks(Ki)
                                Kembalikan: buat Pohon baru dengan parameter: sebelum, hapus(Ki, sebelum), Ka
```

6. Use this following code/file as the building blocks for the next steps. You have to implement these codes either in Java or in any other programming language.

File: PohonAVL.java

```
1  package pohon;
2
3  /**
4   * Kelas PohonAVL dikembangkan kelas Pohon, dengan menambahkan variabel "tinggi"
5   * untuk mewujudkan operasi keseimbangan tinggi (height-balanced) secara efisien
6   */
7  class PohonAVL extends Pohon {
8      protected final boolean kosong;
9      protected final int nilai;
10     protected final PohonAVL kiri, kanan;
11     protected final int tinggi;
12
13     /**
14      * Buat Pohon baru, di mana nilai akarnya adalah n, cabang pohon ke kiri
15      * adalah kiri, cabang pohon ke kanan adalah kanan
16      */
17     public PohonAVL(int n, PohonAVL kiri, PohonAVL kanan) {
18         this.kosong = false;
19         this.nilai = n;
20         this.kiri = kiri;
21         this.kanan = kanan;
22         this.tinggi = 1 + Math.max(kiri.tinggi(), kanan.tinggi());
23     }
24
25     /**
26      * Buat Pohon kosong
27      */
28     public PohonAVL() {
29         this.kosong = true;
30         this.nilai = 0;
31         this.kiri = null;
32         this.kanan = null;
33         this.tinggi = 0;
34     }
```

```
35
36      public PohonAVL(int n) {
37          this.kosong = false;
38          this.nilai = n;
39          this.kiri = new PohonAVL();
40          this.kanan = new PohonAVL();
41          this.tinggi = 1;
42      }
43
44      /**
45       * Mengembalikan nilai true jika pohon ini kosong (nilainya nil)
46       */
47      public boolean adalahKosong() {
48          return kosong;
49      }
50
51      /**
52       * Dapatkan nilai akar dari pohon ini
53       */
54      public int nilai() {
55          if (adalahKosong()) {
56              throw new IllegalStateException(
57                      "Mencoba untuk mengakses akar dari pohon kosong");
58          }
59          return nilai;
60      }
61
```

```
62      /**
63       * Dapatkan cabang ke kiri dari simpul (node) ini
64       */
65      public PohonAVL kiri() {
66          if (adalahKosong()) {
67              throw new IllegalStateException(
68                      "Mencoba untuk mengakses cabang dari pohon kosong");
69          }
70          return kiri;
71      }
72
73      /**
74       * Dapatkan cabang ke kanan dari simpul (node) ini
75       */
76      public PohonAVL kanan() {
77          if (adalahKosong()) {
78              throw new IllegalStateException(
79                      "Mencoba untuk mengakses cabang dari pohon kosong");
80          }
81          return kanan;
82      }
83
84      int tinggi() {
85          return tinggi;
86      }
87
```

```
88      /**
89       * Buat String dalam baris jamak untuk menampilkan pohon ini. Formatnya adalah sbb;
90   <code>
91   10
92      |
93      |- 14
94      |   |
95      |   |- 17
96      |   |
97      |   |- 13
98      |       |
99      |       | - [kosong]
100     |       |
101     |       |- 12
102     |
103     |- 6
104  </code>
105      * Di mana anak pohon paling bawah adalah cabang pohon kiri dan anak pohon paling atas
106      * adalah cabang pohon kanan Jika kedua anak pohon adalah pohon kosong (nil) maka
107      * mereka tidak akan dicetak. Jika hanya satu anak pohon yang kosong, maka mereka akan
108      * dicetak sehingga dapat diketahui mana yang anak pohon kanan, mana yang anak
109      * pohon kiri.
110      * @param pohon Obyek pohon yang boleh jadi tidak boleh null
111      * @return String yang berisi pohon yang terformat
112      */
113     @Override public String toString() {
114         return PohonAksi.toString(this);
115     }
116
117     @Override public boolean equals(Object o) {
118         Pohon p = (Pohon) o;
119         if (kosong)
120             return p.kosong;
121         else
122             return !p.kosong && nilai == p.nilai &&
123                     kiri.equals(p.kiri) && kanan.equals(p.kanan);
124     }
125
126 }
```

7. Please create a recursive function, namely `adalahTinggiSeimbang()` which receives an input `PohonAVL p`, that be able to check whether `p` has a balanced height (AVL tree condition). However, first of all, please define a function, namely `tinggi()`, which returns the height of `PohonAVL p`, as below. Please update the function `adalahTinggiSeimbang()` in file `DAA2.java`. **[10 points]**

```
public static int tinggi (PohonAVL p) {
        return p.tinggi();
}
```

Function name: `public static Boolean adalahTinggiSeimbang (PohonAVL p)`
Pseudo-code:

```
public static boolean adalahTinggiSeimbang (PohonAVL p)
        Jika p adalah kosong
                Kembalikan: benar
        Yang lain
                Buat variabel, kiriT <-- tinggi(kiri dari p)
                Buat variabel, kananT <-- tinggi(kanan dari p)
                Jika Math.abs(kiriT - kananT) kurang dari atau sama dengan 1
                        Kembalikan: adalahTinggiSeimbang(kiri dari p) DAN
                                    adalahTinggiSeimbang(kanan dari p)
                Yang lain
                        Kembalikan: salah
```

8. Please create a recursive function, namely `sisipkanTinggiSeimbang()` which receives the inputs of `int n` and `PohonAVL p`, that be able to insert `n` in `p` while it keeps preserving the AVL condition. Please update the function `sisipkanTinggiSeimbang()` in file `DAA2.java`. **[20 points]**

Function name: `public static PohonAVL sisipkanTinggiSeimbang (int n, PohonAVL p)`
Pseudo-code:

```
public static PohonAVL sisipkanTinggiSeimbang (int n, PohonAVL p)
        Jika p adalah kosong
                Kembalikan: buat PohonAVL baru dengan parameter: n, PohonAVL baru kosong, PohonAVL baru kosong
        Yang lain jika n kurang dari atau sama dengan nilai dari p
                Buat variabel, misal: hasil <-- seimbangkanKembaliKiri dengan
                parameter: Buat pohonAVL baru dengan parameter: nilai dari p,
                sisipkanTinggiSeimbang(n, kiri dari p), kanan dari p)
                // assert adalahBST(hasil) && adalahTinggiSeimbang(hasil);
                Kembalikan: hasil
        Yang lain
                Buat variabel, misal: hasil <-- seimbangkanKembaliKanan dengan
                parameter: Buat pohonAVL baru dengan parameter: nilai dari p,
                kiri dari p, sisipkanTinggiSeimbang(n, kanan dari p)
                // assert adalahBST(hasil) && adalahTinggiSeimbang(hasil);
                Kembalikan: hasil


private static PohonAVL seimbangkanKembaliKiri (PohonAVL p)
        Jika tinggi(kiri dari p) kurang dari atau sama dengan (tinggi(kanan dari p) + 1)
                Kembalikan: p
        Yang lain
                Buat variabel, misal: Ki <-- kiri dari p
                Buat variabel, misal: Ka <-- kanan dari p
                Buat variabel, misal: KiKi <-- kiri dari Ki
                Buat variabel, misal: KiKa <-- kanan dari Ki
                Jika tinggi(KiKi) lebih dari tinggi(Ka)
                        // Rotasi KiKi - rotasi tunggal
                        Kembalikan: Buat pohonAVL baru dengan parameter: nilai dari
                                Ki, KiKi, buat pohonAVL baru dg parameter: nilai p, KiKa, Ka
                Yang lain
                        // assert tinggi(KiKa > tinggi(Ka)
                        // Rotasi KiKa - ganda
                        Kembalikan: Buat pohonAVL baru dg parameter: nilai dari KiKa,
                                // Kiri
                                (Buat pohonAVL baru dg parameter: nilai dari Ki, kiri dari Ki, kiri dari KiKa),
                                // Kanan
                                (Buat pohonAVL baru dg parameter: nilai dari p, kanan dari KiKa, kanan dari p)
```

```
private static PohonAVL seimbangkanKembaliKanan (PohonAVL p)
        Jika tinggi(kanan dari p) kurang dari atau sama dengan (tinggi(kiri dari p) + 1)
                Kembalikan: p
        Yang lain
                Buat variabel, misal: Ki <-- kiri dari p
                Buat variabel, misal: Ka <-- kanan dari p
                Buat variabel, misal: KaKi <-- kiri dari Ka
                Buat variabel, misal: KaKa <-- kanan dari Ka
                Jika tinggi(KaKa) lebih dari tinggi(Ki)
                        // Rotasi KaKa - rotasi tunggal
                        Kembalikan: Buat pohonAVL baru dengan parameter: nilai dari
                                Ka, (buat pohonAVL baru dg parameter: nilai p, Ki, KaKi), KaKa
                Yang lain
                        // assert tinggi(KaKi > tinggi(Ki)
                        // Rotasi KaKi - ganda
                        Kembalikan: Buat pohonAVL baru dg parameter: nilai dari KaKi,
                                // Kiri
                                (Buat pohonAVL baru dg parameter: nilai dari p, kiri dari p, kiri dari KaKi),
                                // Kanan
                                (Buat pohonAVL baru dg parameter: nilai dari Ka, kanan dari KaKi, kanan dari Ka)
```

9. Please create a recursive function, namely `hapusTinggiSeimbang()` which receives the inputs of `PohonAVL p` and `int n`, that be able to delete `n` from `p` while it keeps preserving the AVL condition. Please update the function `hapusTinggiSeimbang()` in file `DAA2.java`. **[20 points]**

Nama fungsi: `public static PohonAVL hapusTinggiSeimbang (PohonAVL p, int n)`
Pseudo-code:

```
public static PohonAVL hapusTinggiSeimbang (PohonAVL p, int n)
        Jika p adalah kosong
                Kembalikan: p
        Yang lain
                Buat variabel, misal: a <-- nilai dari p
                Buat variabel, misal: Ki <-- kiri dari p
                Buat variabel, misal: Ka <-- kanan dari p

                Jika n lebih dari a
                        Buat variabel, misal: baruKa <-- hapusTinggiSeimbang(Ka, n);
                        Kembalikan: seimbangkanKembaliKiri() dg parameter: buat pohonAVL
                                baru dg parameter: nilai dari p, Ki, baruKa
                Yang lain jika n kurang dari a
                        Buat variabel, misal: baruKi <-- hapusTinggiSeimbang(Ki, n);
                        Kembalikan: seimbangkanKembaliKanan() dg parameter: buat pohonAVL
                                baru dg parameter: nilai dari p, baruKi, Ka

                Yang lain
                        Jika Ki adalah kosong
                                Kembalikan: Ka
                        Yang lain jika Ka adalah kosong
                                Kembalikan: Ki
                        Yang lain
                                Buat variabel, misal: sebelum <-- maks(Ki)
                                Kembalikan: seimbangkanKembaliKanan() dg parameter: buat pohonAVL
                                        baru dg parameter: sebelum,
                                                hapusTinggiSeimbang(Ki, sebelum), Ka
```

10. To avoid plagiarism/cheating, every student needs to pledge and declare, then she/he must submit her/his **signed pledge and declaration** as in the following. Failed to do so will be resulted in getting 0 (zero) grade. Attach the **scanned**/**photo** of your *declaration* in your report.

    "By the name of Allah (God) Almighty, herewith I pledge and truly declare that I have solved midterm exam by myself, didn't do any cheating by any means, didn't do any plagiarism, and didn't accept anybody's help by any means. I am going to accept all of the consequences by any means if it has proven that I have been done any cheating and/or plagiarism."

    [Place, e.g., Surabaya], [date, e.g., 13 March 2020]

    <Signed>

    [Full name, e.g., Pariyem Pandanwangi]
    [StudentID, e.g., 05111940000xxx]

11. Please create a full report, in PDF format, of the source codes (.java or any other programming language's source codes), the outputs of your programs and its analysis. Then put this PDF format report along with your declaration (see step #10 above) and all of .java files (or any other programming language's source code) into 1 (one) .ZIP file. Please name this file: IF184401_DAA(E)_MID_StudentID_Name.ZIP.

12. Have an amazing day, guys! Good luck! 😊