Aufgabe Bäume: Gegeben sei folgender Baum (kein Suchbaum) mit Integerwerten:

a) **Definieren Sie eine Datenstruktur** zur Repräsentation der einzelnen Knoten des Baums.
b) **Erzeugen** Sie genau diesen Baum und speichern Sie die Referenz auf den Wurzelknoten.
c) Schreiben Sie Methode oder Funktion zur Berechnung der **Summe** aller Werte im Baum.

Aufgabe Bäume:
Implementieren Sie die Methoden `size()`, `printInOrder()` **und** `addSorted(int i)` **für die Klasse** `Node`. `Node` **stellt dabei den Knoten eines Baumes dar. Die Knoten können nur** `int` **Werte speichern. In dieser Implementierung werden leere Bäume durch** `null` **repräsentiert.**

Folgender Code soll z.B. ausgeführt werden können:

```
public class TestTree {
      public static void main(String[] args) {

} }

// Wurzel des Baums:
Node root = new Node (40);

// Hinzufügen von Werten:
root.addSorted( 30);
root.addSorted( 5);
root.addSorted( 8);
root.addSorted( 12);
root.addSorted( 99);
// Gibt auf der Konsole aus: 5 8 12 30 40 99
root.printInOrder();
```

Hier beginnt die Implementierung der `Node` Klasse: `public class Node {`

```
      // Linker und rechter Teilbaum
      private Node left , right;
      // Wert dieses Knotens
      private int value;

// Der leere Baum wird durch null repräsentiert public final static Node EMPTY =
null;

// Konstruktor
public Node (int value) {

this.value = value; left = EMPTY; right = EMPTY;

}
```

Platz für die zu implementierenden Methoden finden Sie auf der nächsten Seite.

```
// TODO: Diese Methode soll die Anzahl der Knoten des Baumes zurückliefern

      public int size ()  {

}
```

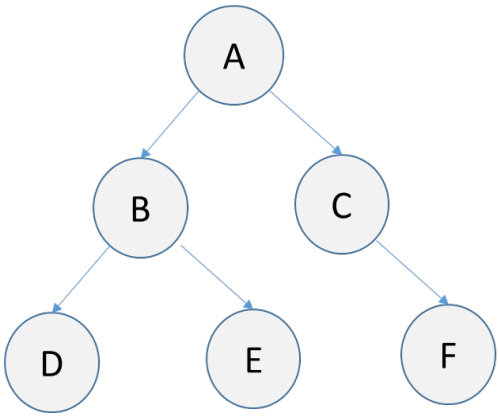
```
// TODO: Diese Methode soll alle Knoten in-order ausgeben

      public void printInOrder () {

System.out.println(value);

}

// TODO: Diese Methode soll den Wert i sortiert einfügen. // Beachten Sie die
Sonderfälle für leere Teilbäume public void addSorted (int i) {

}
```

Aufgabe Bäume: Gegeben sei folgender Baum (kein Suchbaum) mit Buchstaben:

a) Geben Sie die Elemente des Baums aus in Pre-Order, In-Order und Post-Order-Reihenfolge: Pre-Order:

In-Order:

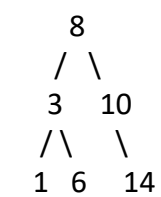
Post-Order:



b) **Definieren Sie eine Datenstruktur** zur Repräsentation der einzelnen Knoten des Baums.

c) Schreiben Sie eine Methode oder Funktion zur Berechnung der **Anzahl** der Knoten im Baum.

Aufgabe 4: Gegeben sei der folgende binäre Baum.



Die einzelnen Knoten enthalten Buchstaben. Geben Sie diese Buchstaben in Pre-Order, In- Order und Post-Order Reihenfolge an.

Pre-Order:

In-Order:

Post-Order:

Aufgabe Bäume: Gegeben sei folgendes Interface für einen Baum mit Integerwerten:

```
public interface IntegerTree {
     /**
      * Ist der Teilbaum leer oder gibt es Kindelemente?
      * @return leerer Baum -> true, sonst false
      */

/**
* Durchläuft den Baum und summiert die Werte aller Knoten * @return Summe
aller Knoten
*/

/**
* Durchläuft den Baum und ersetzt alle Vorkommen
* von oldValue durch newValue.
* Z.B. können alle Vorkommen der Zahl 5 durch die Zahl 10 ersetzt werden *
@param oldValue ist der zu ersetzende Wert
* @param newValue ist der neue Wert
*/

}
```

Die Implementierung des leeren Knotens sieht so aus:

```
 public class EmptyTree implements IntegerTree {
     @Override
     public boolean isEmpty() {
public boolean isEmpty ();
public int sum ();
public void replace (int oldValue , int newValue);

return true;

@Override
public int sum() {

return 0;

 // Der leere Baum ist leer.
```

```
// Die Summe eines leeren Baums ist 0

}

}

     @Override
     public void replace(int oldValue, int newValue) {
         // Keine Aktion, beim leeren Baum wird nichts ersetzt.
     }

}
```

Implementieren Sie (auf der nächsten Seite) den nicht-leeren Knoten.

Ihre Implementierung:

```
public class IntegerNode implements IntegerTree {

    private IntegerTree left, right;
    private int value;

public IntegerNode (int value , IntegerTree left , IntegerTree right) {
this.value = value;
this.left = left;

  this.right = right;
}

@Override
public boolean isEmpty() {

}

}

// 4a) TODO: ist dies der leere Baum?
@Override
public int sum() {

// 4b) TODO: Liefere Summe dieses Knotens plus Summe der Teilbäume

@Override
public void replace(int oldValue, int newValue) {
// 4c) TODO: Ggf. Wert ersetzen.
//     TODO: Linken + rechten Teilbaum rekursiv aufrufen

} }
```