File - C:\Users\victo\IdeaProjects\test3übung2\src\List.java

```
import java.util.LinkedList;

class ListNode {
    int elem;
    ListNode next=null;

    public ListNode(int elem, ListNode next){
        this.elem = elem;
        this.next = next;
    }
    @Override
    public String toString(){
        String erg = "";

        if(this.next != null){
            erg+=this.elem+this.next.toString();

        }
        else{
            erg+=this.elem; //letztes element das durch != null nicht erkannt wird
        }

        return erg;
    }
    void addRight(int elem){

        if(this.next != null){
            this.next.addRight(elem);
        }
        else{
            this.next = new ListNode(elem, null);
        }

    }
    int sumEvenElements(){
        int erg = 0;
        if(this.next != null){
            if(this.elem%2 == 0){
                erg+=this.elem+this.next.sumEvenElements();
            }
            else{
                erg+=this.next.sumEvenElements();
            }
        }
        else{
            if(this.elem%2 == 0)erg+=this.elem;         //letztes Element auch dazu addieren
        }
        return erg;
    }
    void removeLast(){
        if(this.next.next == null) this.next = null;
        else this.next.removeLast();
    }
}

class Lists {
    private ListNode head = null;
    public Lists(){
        this.head = null;
    }
    void addLeft(int elem){
        if(this.head == null) this.head = new ListNode(elem,null);
        else this.head = new ListNode(elem,this.head);
    }
    void addRight(int elem){
        if(this.head == null) this.head = new ListNode(elem,null);
        else this.head.addRight(elem);
    }
    @Override
    public String toString(){
        return "[" + this.head.toString() + "]";
    }
    public int sumEvenElements(){
        return this.head.sumEvenElements();
    }
    public void removeFirst(){
        this.head = this.head.next;
    }
    public void removeLast(){
        this.head.removeLast();
    }
    public int pop(){
        int pop = this.head.elem;
        this.removeFirst();
        return pop;
    }
}



public class List{
    public static void main(String[] args)
    {
        Lists list = new Lists();
```

File - C:\Users\victo\IdeaProjects\test3übung2\src\List.java

```
        list.addLeft(2);
        list.addLeft(4);
        list.addLeft(3);
        list.addLeft(9);
        list.addRight(1);
        list.addRight(3);
        list.addRight(5);
        list.addRight(6);
        System.out.println(list.toString());
        System.out.println(list.sumEvenElements());

        list.removeFirst();
        list.removeFirst();
        list.removeFirst();
        System.out.println(list.toString());
        list.removeLast();
        list.removeLast();
        System.out.println(list.toString());
        System.out.println(list.pop());
        System.out.println(list.pop());
        System.out.println(list.toString());

    }
}
```

```java
import com.sun.org.apache.xml.internal.serialize.LineSeparator;

import java.util.LinkedList;

/*
Schreiben bzw. vervollständigen Sie gekennzeichnete Programmteile wie in den Kommentaren beschrieben.
Verändern Sie aber nicht vorgegebene Methodenköpfe oder andere vorgegebene Programmteile!
Verwenden Sie die gegebenen Testklassen zum Überprüfen Ihrer Lösung. Die Korrektheit dieser Testfälle
ist notwendig, aber nicht hinreichend für eine korrekte Lösung. Es werden damit nur bestimmte
Spezialfälle geprüft. Liefern die Testklassen keinen Fehler, garantiert dies noch nicht die
Korrektheit Ihrer Lösung gemäß der Angabe.
*/
public class Minenfeld {

    /*
    Aufgabe: Umgang mit Klassen und Objekten, sowie vorgefertigten Datenstrukturen.

        Ergänzen Sie fehlende Teile der Klassen 'Position' und 'Minefield' entsprechend den Kommentaren
        an den mit TODO gekennzeichneten Stellen.

    Punkte (maximal 40):
        12 Punkte in Summe für Klasse 'Position':
            3 Punkte für Konstruktoren und Objektvariablen
            2 Punkte für 'moveVertical'
            4 Punkte für 'isNeighbour'
            3 Punkte für 'equals' und 'hashCode',
        28 Punkte für Klasse 'Minefield':
            3 Punkte für den Konstruktor
            5 Punkte für 'isBomb'
            5 Punkte für 'setBomb'
            5 Punkte für 'trim'
            5 Punkte für 'toString'
            5 Punkte für 'copy'

    Auch für teilweise korrekte Lösungen werden Punkte vergeben.

    */



    // Diese Methode können Sie zum Testen nutzen. Sie geht nicht in die Beurteilung ein.
    public static void main(String[] args) {

        //Testen von Position:
        Position p = new Position(1,2);

        for (int x = 0; x < 5; x++) {
            for (int y = 0; y < 5; y++) {
                if ((x == 1 && y == 2) != new Position(x,y).equals(p)) {
                    System.out.println("Fehler in Klasse 'Position'!");
                }
            }
        }

        //Testen von Minefield:
        Minefield f = new Minefield(10, 13);
        System.out.println(f.setBomb(3, 2));
        System.out.println(f.setBomb(5, 3));
        System.out.println(f.setBomb(4, 5));
        System.out.println(f.setBomb(6, 5));
        System.out.println(f.setBomb(4, 9));
        System.out.println(f.setBomb(3, 2));
        System.out.println(f.setBomb(9, 9));
        System.out.println();
        System.out.println(f);
        Minefield g = f.copy();
        f.trim();
        System.out.println();
        System.out.println(f);
        System.out.println();
        System.out.println(g);
        /* Erwartete Ausgabe:

        true
        true
        true
        true
        true
        false
        true

        0000000000
        0011100000
        001*211000
        00112*1000
        0001232100
        0001*2*100
        0001121100
        0000000000
        0001110011
        0001*1001*
        0001110011
        0000000000
        0000000000
        10 13
```

```
        001*211000
        00112*1000
        0001232100
        0001*2*100
        0001121100
        0000000000
        0001110011
        0001*1001*
        10 8

        0000000000
        0011100000
        001*211000
        00112*1000
        0001232100
        0001*2*100
        0001121100
        0000000000
        0001110011
        0001*1001*
        0001110011
        0000000000
        0000000000
        10 13

        */


    }
}

class Position {
    public int x,y;

    public Position(int x, int y) {
        if(x<0||y<0){
            x=0;
            y=0;
        }
        this.x=x;
        this.y=y;
        //TODO: Implementieren Sie diesen Konstruktor

    }
    public Position(Position p) {
        x=p.x;
        y=p.y;
    }
    public void moveVertical(int deltaY) {
        y+=deltaY;

        if(y<0){
            y=0;
        }
    }
    @Override
    public boolean equals(Object o) {
        if (o == null) {
            return false;
        }
        if (o.getClass() != Position.class) {
            return false;
        }
        Position p = (Position) o;

        if(p.x==x&&p.y==y){
            return true;
        }
        return false;
    }
    public int hashCode() {
        return x;
    }
    public boolean isNeighbour(Position p) {
        return (x+1>=p.x&&x-1<=p.x) && (y+1>=p.y&&y-1<=p.y);
    }
}

class Minefield {

    private LinkedList<Position> bombs;
    private int width;
    private int height;

    public Minefield(int width, int height) {
        bombs = new LinkedList<>();
        this.width=width;
        this.height=height;
    }
    public boolean isBomb(int x, int y) {

        for (Position p: bombs) {
            if((new Position(x,y).equals(p))){
                return true;
```

```
            }
        }


        //TODO: Implementieren Sie diese Methode
        return false; //TODO: Diese Zeile entfernen oder entsprechend ändern

    }
    public boolean setBomb(int x, int y) {

        if(x>=0&&y>=00&&x<width&&y<height){

            if(!isBomb(x,y)){

                bombs.add(new Position(x,y));
                return true;
            }
        }

        //TODO: Implementieren Sie diese Methode
        return false; //TODO: Diese Zeile entfernen oder entsprechend ändern

    }
    public int bombNeighbours(int x, int y) {
        int count=0;

        for (Position p: bombs) {
            if((new Position(x,y)).isNeighbour(p)){
                count++;
            }
        }
        return count;

    }
    @Override
    public String toString() {

        String s = "";

        for (int iy=0;iy<height;iy++){

            for(int ix=0; ix<width;ix++){
                if(isBomb(ix,iy)){
                    s+="*";
                } else{
                    s+=bombNeighbours(ix,iy);
                }
            }
            s+= System.lineSeparator();
        }
        s+=width+" "+height;

        //TODO: Implementieren Sie diese Methode
        return s; //TODO: Diese Zeile entfernen oder entsprechend ändern

    }
    public void trim() {
        boolean bombinRow=false;
        int delta =0;

        for (int iy=height-1;iy>=0;iy--){


            for (Position p: bombs) {
                if(p.y==iy){
                    bombinRow=true;
                }
            }

            if(!bombinRow){
                delta++;
            } else {
                break;
            }

            bombinRow=false;

        }

        height-=delta;

        delta=0;
        bombinRow=false;

        int moveDelta= 0;

        for (int iy=0;iy<height;iy++){


            for (Position p: bombs) {
                if(p.y==iy){
                    bombinRow=true;
                }
            }
```

```
            if(!bombinRow){
                delta++;
            } else {
                break;
            }

            bombinRow=false;
        }

        for (Position p: bombs) {
            if(p.y>delta){
                p.moveVertical(-delta);
            }
        }

        height-=delta;


    }
    public Minefield copy() {

        Minefield m = new Minefield(width,height);

        for (Position p: bombs) {
            m.setBomb(p.x,p.y);
        }

        return m;

    }
}
```

File - C:\Users\victo\IdeaProjects\test3übung2\src\PointPlane.java

```java
import java.util.LinkedList;

/*
Schreiben bzw. vervollständigen Sie gekennzeichnete Programmteile wie in den Kommentaren beschrieben.
Verändern Sie aber nicht vorgegebene Methodenköpfe oder andere vorgegebene Programmteile!
Verwenden Sie die gegebenen Testklassen zum Überprüfen Ihrer Lösung. Die Korrektheit dieser Testfälle
ist notwendig, aber nicht hinreichend für eine korrekte Lösung. Es werden damit nur bestimmte
Spezialfälle geprüft. Liefern die Testklassen keinen Fehler, garantiert dies noch nicht die
Korrektheit Ihrer Lösung gemäß der Angabe.
*/


class Point
{
    private int x, y;
    public Point(int x, int y){
        this.x = x;
        this.y = y;
    }
    public Point(Point p){
        this.x = p.x;
        this.y = p.y;
    }
    public void moveHorizontal(int deltaX){
        this.x += deltaX;
    }
    public int distanceTo(Point p) {
        int diffx, diffy;
        diffx = Math.abs(x - p.x);
        diffy = Math.abs(this.y - p.y);

        return diffx + diffy;
    }
    @Override
    public String toString(){

        return "[" + this.x + "," + this.y + "]";
    }
    @Override
    public boolean equals(Object o){
        if(o == null) return false;
        if(o.getClass() != Point.class) return false;
        Point p = (Point) o;

        return p.x == this.x && p.y == this.y;
    }
    @Override
    public int hashCode()
    {
        return this.y;
    }









}

class Plane
{
    // Eine Liste zum Speichern der Punkte.
    private LinkedList<Point> list;
    // Eine Referenz auf den ausgewählten Punkt (siehe 'selectNearestTo'). Ist zunächst null.
    private Point selection = null;

    // Konstruktor: initialisiert diese Ebene als leere Ebene ohne Punkte.
    public Plane()
    {
        this.list = new LinkedList<>();
    }
    public Plane(Plane plane){
        this.list = new LinkedList<>();
        for (Point p : plane.list)
        {
            this.list.add(new Point(p));
        }
    }
    public boolean add(Point point){
        if(this.list.contains(point)) return false;
        else{
            this.list.add(point);
            return true;
        }
    }
    public boolean add(int x, int y){
        return add(new Point(x,y));
    }
    public void selectNearestTo(Point point){
        for (Point p : this.list) {
            if(this.selection == null) this.selection = p;
```

File - C:\Users\victo\IdeaProjects\test3übung2\src\PointPlane.java

```
            if(p.distanceTo(point)<=this.selection.distanceTo(point)) this.selection = p;
        }
    }
    public Point selection() {
        return this.selection;
    }
    public void moveSelection(int deltaX)
    {
        this.selection.moveHorizontal(deltaX);
    }
    public String toString(){
        String ret = "";

        for (Point p : this.list)
        {
            if(p.equals(selection))
                ret += ("["+p.toString()+"]") + "\n";
            else
                ret += (p.toString()) + "\n";
        }

        return ret;
    }
}



public class PointPlane {
    public static void main(String[] args) {
         Plane p = new Plane();
         p.add(2,3);
        p.add(3,2);
        p.add(7,6);
        System.out.println(p.toString());

        Plane q = new Plane(p);

        System.out.println(q.toString());

        q.selectNearestTo(new Point(1,1));
        System.out.println(q.selection());

    }
}
```

File - C:\Users\victo\IdeaProjects\test3übung2\src\PotionCabinet.java

```java
import java.util.*;

/*
Lesen Sie die Aufgaben genau durch.
Schreiben bzw. vervollständigen Sie gekennzeichnete Programmteile wie in den Kommentaren beschrieben.
Verändern Sie aber nicht vorgegebene Methodenköpfe oder andere vorgegebene Programmteile!
*/
public class PotionCabinet {

    /*
    Aufgabe: Umgang mit Klassen und Objekten, sowie vorgefertigten Datenstrukturen.

        Ergänzen Sie fehlende Teile der Klassen Potion und Ingredient entsprechend den Kommentaren
        an den mit TODO gekennzeichneten Stellen.
        Sonstige Anforderungen: Es sollen keine zusätzlichen Methoden oder Objektvariablen
        implementiert werden.
        Hinweis: 'Cabinet' kann auch implementiert werden, ohne dass 'Potion' implementiert
        wird. In diesem Fall weichen die Ausgaben von den Sollausgaben ab.

    Punkte (maximal 34):

        20 Punkte in Summe für Klasse 'Potion':
            5 Punkte für den zweiten Konstruktor
            2 Punkt für 'stir'
            3 Punkt für 'ready'
            4 Punkte für 'power'
            6 Punkte für 'mixWith',
        14 Punkte für Klasse 'Cabinet':
            4 Punkte für den Konstruktor und Variablen
            6 Punkte für 'put'
            4 Punkte für 'toString'

    */

    // Diese Methode können Sie zum Testen der Klassen 'Potion' und 'Cabinet'
    // verwenden. Diese Methode wird nicht bewertet.
    public static void main(String[] args) {

        // Überprüft die korrekte Basisimplementierung der Klasse 'Potion':
        // Konstruktoren und 'toString'
        // Erwarteter Output:
        // [Bergblume, Säbelzahntigerauge]:27
        // [Bergblume, Weizen]:15
        // [Mondfalterflügel, Vampirstaub]:27
        Potion healing1 = new Potion(new String[]{"Bergblume","Säbelzahntigerauge"}, 3);
        Queue<String> ingr = new LinkedList<String>();
        ingr.offer("Bergblume");
        ingr.offer("Weizen");
        Potion healing2 = new Potion(ingr, 2);
        Potion invisibility = new Potion(new String[]{"Mondfalterflügel","Vampirstaub"}, 2);
        System.out.println(healing1);
        System.out.println(healing2);
        System.out.println(invisibility);

        // Überprüft die korrekte Implementierung der von 'stir':
        // Erwarteter Output:
        // [Bergblume, Säbelzahntigerauge]:54
        healing1.stir();
        healing1.stir();
        healing1.stir();
        System.out.println(healing1);

        // Überprüft die korrekte Basisimplementierung der Klasse 'Cabinet':
        // Konstruktoren und 'toString'
        // Erwarteter Output (Reihenfolge kann abweichen):
        // {Heilung=[Bergblume, Säbelzahntigerauge, Bergblume, Weizen]:84, Unsichtbarkeit=[Mondfalterflügel, Vampirstaub]:54}
        //Cabinet potionArchive = new Cabinet();
        //potionArchive.put("Heilung", healing1);
        //potionArchive.put("Heilung", healing2);
        //potionArchive.put("Unsichtbarkeit", invisibility);
        //System.out.println(potionArchive);

    }

}


// Potion ist ein Zaubertrank mit einer beliebigen Anzahl an Zutaten. Er hat eine beliebige Anzahl an
// Zutaten und eine bestimmte Stärke, die sich aus der
// Summe der Stärken der Zutaten ergibt.
// Wird er fertig umgerührt wirkt er
// doppelt so stark.
class Potion {

    // Objektvariablen für den Zaubertrank. Es sind keine weiteren Objektvariablen zulässig.
    private String[] ingredients;
    private int howOftenToBeStired;
    private int howOftenStired = 0;

    public Potion(String[] ingredients, int howOftenToBeStired) {
        this.howOftenToBeStired = howOftenToBeStired;
        this.ingredients = ingredients;
    }
    public void add(String ingredient) {
```

```
        String[] ingredients = new String[this.ingredients.length+1];
        int i = 0;
        for (String ing: this.ingredients) {
            ingredients[i++] = ing;
        }
        ingredients[i] = ingredient;
        this.ingredients = ingredients;

    }
    public Potion(Queue<String> ingredients, int howOftenToBeStired) {
        this.howOftenToBeStired = howOftenToBeStired;

        String [] zutaten= new String[ingredients.size()];

        for (int i = 0; i < zutaten.length; i++) {
            zutaten[i] = ingredients.poll();
        }
        this.ingredients = zutaten;
    }
    public void stir() {
        this.howOftenStired ++;

        //TODO: implementieren Sie diese Methode

    }
    public int power(){
        int power = 0;
        for (int i = 0; i < this.ingredients.length; i++) {
            power += ingredients[i].length();
        }
        return (this.ready()?2*power : power);
    }
    public boolean ready() {

        if (howOftenToBeStired == howOftenStired){
            return true;
        }

        //TODO: implementieren Sie diese Methode
        return false; //TODO: diese Zeile entfernen oder entsprechend ändern

    }
    public void mixWith(Potion potionToAdd){
        this.howOftenToBeStired += potionToAdd.howOftenToBeStired;
        this.howOftenStired += potionToAdd.howOftenStired;
        String[] mixedIngredients = new String[this.ingredients.length + potionToAdd.ingredients.length];

        for (int i = 0; i < this.ingredients.length; i++) {
            mixedIngredients[i] = this.ingredients[i];
        }
        for (int i = this.ingredients.length; i < this.ingredients.length+potionToAdd.ingredients.length; i++) {
            mixedIngredients[i] = potionToAdd.ingredients[i];
        }
        this.ingredients = mixedIngredients;
    }
    public String toString() {
        return Arrays.toString(ingredients) + ":" + power();
    }
}

// Repäsentiert einen Schrank mit Zaubertränken. Es können beliebig viele Tränke in den Schrank gestellt werden.
// Jeder Trank im Schrank bekommt eine Beschreibung seines Effektes.
class Cabinet {

    Map<String,Potion> schrankinhalt =new HashMap<>();



    // Erzeugt einen leeren Schrank.
    public Cabinet(Map<String,Potion> schrankinhalt) {
        this.schrankinhalt = schrankinhalt;


        //TODO: implementieren Sie den Konstruktor.

    }

    // Stellt einen Zaubertrank in den Schrank. Dabei wird die Beschreibung 'effect'
    // als Schlüssel verwendet, um später den Zaubertrank wieder zu finden.
    // Gibt es im Schrank bereits einen Zaubertrank mit der angegebenen Beschreibung,
    // wird 'potionToPut' dem bereits vorhandenen Zaubertrank hinzugefügt.
    // In jedem Fall wird sichergestellt, dass ein Zaubertrank der im Schrank steht,
    // fertig umgerührt ist.
    public void put(String effect, Potion potionToPut) {

        if (potionToPut.ready()){

            schrankinhalt.put(effect,potionToPut);


            if (schrankinhalt.containsKey(effect)){
                potionToPut.mixWith(potionToPut);
            }
        }
        //TODO: implementieren Sie diese Methode.
```

File - C:\Users\victo\IdeaProjects\test3übung2\src\PotionCabinet.java

```
    }

    // Liefert eine lesbare Repräsentation (String) des Schranks in
    // folgendem Format:
    // {effect=Zaubertrank, effect=Zaubertrank, ... }
    // also z.B.
    // {Unsichtbarkeit=[Mondfalterflügel, Vampirstaub]:54, Heilung=[Bergblume, Säbelzahntigerauge, Bergblume, Weizen]:84}
    // Hinweis: Nutzen Sie ggfs. die toString-Methode der verwendeten Map Klasse (TreeMap, HashMap).
    public String toString() {

        //TODO: implementieren Sie diese Methode.
        return  "= "; //TODO: diese Zeile entfernen oder entsprechend ändern.

    }

}
```

File - C:\Users\victo\IdeaProjects\test3übung2\src\SongBookApp.java

```
import java.util.*;

public class SongBookApp {

    /*
    Aufgabe: Umgang mit Klassen und Objekten, sowie vorgefertigten Datenstrukturen.

        Ergänzen Sie fehlende Teile der Klassen Melody und SongBook entsprechend den Kommentaren
        an den mit TODO gekennzeichneten Stellen.
        Sonstige Anforderungen: Es sollen keine zusätzlichen Methoden oder Objektvariablen
        implementiert werden.
        Hinweis: 'SongBook' kann auch implementiert werden, ohne dass 'Melody' implementiert
        wird. In diesem Fall kann kann die Ausgabe von den angegebenen Sollausgaben abweichen.

    Punkte (maximal 34):
        21 Punkte in Summe für Klasse 'Melody':
            7 Punkte für 'copy'
            2 Punkte für 'setBPM'
            7 Punkte für 'transpose'
            5 Punkte für 'toString',
        13 Punkte für Klasse 'SongBook':
            4 Punkte für den Konstruktor und Variablen
            6 Punkte für 'addMelody'
            3 Punkte für 'getMelody'

    */

    // Diese Methode steht Ihnen zum Testen zur Verfügung. Ihr Inhalt wird
    // nicht beurteilt.
    public static void main(String[] args) {

        // Überprüft die korrekte Basisimplementierung der Klasse Melody:
        // 'setBPM' und 'toString'
        // Erwarteter Output:
        // do 2 fa 4 si 6
        // 6.0 seconds
        // do 2 fa 4 si 6
        // 4.5 seconds
        Melody m1 = new Melody(120);
        m1.addNote(new Note(0, 2));
        m1.addNote(new Note(3, 4));
        m1.addNote(new Note(6, 6));
        System.out.println(m1);
        m1.setBPM(160);
        System.out.println(m1);

        // Überprüft die korrekte Umsetzung der Teilaufgabe 'transponieren'.
        // Erwarteter Output:
        // mi 2 la 4 re 6
        // 6.0 seconds
        m1 = new Melody(120);
        m1.addNote(new Note(0, 2));
        m1.addNote(new Note(3, 4));
        m1.addNote(new Note(6, 6));
        m1.transpose(2);
        System.out.println(m1);

        // Überprüft die korrekte Umsetzung der Teilaufgabe 'kopieren'.
        // Erwarteter Output:
        // re 4 mi 1 la 1
        // 6.0 seconds
        // re 4 mi 1 la 1
        // 6.0 seconds
        m1 = new Melody(60);
        m1.addNote(new Note(0, 2));
        m1.addNote(new Note(3, 4));
        m1.addNote(new Note(1, 4));
        m1.addNote(new Note(2, 1));
        m1.addNote(new Note(5, 1));
        Melody m2 = m1.copy(2,5);
        System.out.println(m2);
        m1.transpose(1);
        System.out.println(m2);


        // Überprüft die korrekte Implementierung des Songbooks.
        // Erwarteter Output:
        // true
        // true
        // true
        // false
        // true
        m1 = new Melody(60);
        m1.addNote(new Note(0, 2));
        m1.addNote(new Note(3, 4));
        SongBook sb1 = new SongBook();
        System.out.println(sb1.addMelody("Testtitel", m1));
        System.out.println(sb1.addMelody("Another Song", new Melody(1)));
        m2 = sb1.getMelody("Testtitel");
        System.out.println(m1 == m2);
        System.out.println(sb1.addMelody("Testtitel", new Melody(2)));
        Melody m3 = sb1.getMelody("Testtitel");
        System.out.println(m1 == m3);

    }
```

```
}

// Die Klasse Note ist fertig vorgegeben und repräsentiert eine Note.
// Eine Note ist der kleinste Bestandteil einer Melodie und ist
// charakterisiert durch Höhe und Länge. Die Notenlänge
// wird in Schlägen angegeben. Es gibt insgesamt 7 Noten. In aufsteigender
// Reihenfolge: do - re - mi - fa - sol - la - si. Die Notennamen haben
// numerische Entsprechungen.
//
// do re mi fa sol la si
//  0  1  2  3   4  5  6
class Note {

    private int noteIndex;
    private int beats;

    public static final String[] steps = {"do","re","mi","fa","sol","la","si"};
    public Note(int noteIndex, int beats) {

        this.noteIndex = noteIndex;
        this.beats = beats;

    }
    public Note(Note note) {

        this(note.noteIndex, note.beats);

    }
    public int getBeats() {

        return this.beats;

    }
    public void transpose(int steps) {

        this.noteIndex += steps;

        while(this.noteIndex < 0) {
            this.noteIndex += Note.steps.length;
        }

        this.noteIndex %= Note.steps.length;

    }
    public String toString() {

        return Note.steps[this.noteIndex] + " " + beats;

    }
}

class Melody {
    private Note[] notes;
    private int bpm;

    public Melody(int beatsPerMinute) {

        this.bpm=beatsPerMinute;
        this.notes = new Note[0]; //zu Beginn eine leere Notenfolge

    }
    public void addNote(Note note){
        Note[] newnotes = new Note[this.notes.length+1];
        for (int i = 0; i < this.notes.length; i++) {
            newnotes[i] = notes[i];
        }
        newnotes[this.notes.length-1] = note;
        this.notes = notes;
    }
    public Melody copy(int beginIndex, int endIndex) {
        Melody copy = new Melody(this.bpm);
        for (int i = beginIndex; i <= endIndex; i++) {
            copy.addNote(this.notes[i]);
        }
        return copy;
    }
    public void setBPM(int beatsPerMinute) {
        this.bpm = beatsPerMinute;
    }
    public void transpose(int steps){
        for (int i = 0; i < this.notes.length-1; i++) {
            notes[i].transpose(steps);
        }
    }
    public String toString() {
        String res = "";
        double beats = 0;
        for(Note n : notes){
            res+= n + " ";
            beats += n.getBeats();
        }
        res+= "\n";
        double length = 60/(double)this.bpm * beats;
        res += length + " seconds";
        return res;
```

```
    }
}

class SongBook {

    Map<String,Melody> saved;

    // Initialisiert die Instanz von SongBook.
    public SongBook() {
        this.saved = new TreeMap<>();
    }

    // Speichert im SongBook unter dem angegebenen Titel (title) eine Melodie
    // (melody) und gibt true zurück. Gibt es jedoch im SongBook unter diesem
    // Namen bereits eine Melodie, so wird die übergebene Melodie nicht
    // hinzugefügt, die Methode gibt false zurück.
    public boolean addMelody(String title, Melody melody) {
        if(this.saved == null) {
            return false;
        }
        if(this.saved.containsKey(title)){
            return false;
        }
        this.saved.put(title, melody);
        return true;


    }


    // Gibt die unter dem angegeben Titel gespeicherte Melodie zurück.
    // Existiert keine Melodie unter diesem Titel, so wird null zurückgegeben.
    public Melody getMelody(String title) {
        if(this.saved == null){
            return null;
        }
        if(this.saved.containsKey(title)){
            return this.saved.get(title);
        }
        return null;

    }

}
```

File - C:\Users\victo\IdeaProjects\test3übung2\src\Tree.java

```java
/*
Lesen Sie die Aufgaben genau durch.
Schreiben bzw. vervollständigen Sie gekennzeichnete Programmteile wie in den Kommentaren beschrieben.
Verändern Sie aber nicht vorgegebene Methodenköpfe oder andere vorgegebene Programmteile!
*/
public class Tree {

    /*
    Aufgabe:
        Objekte der Klasse Tree stellen binäre Suchbäume über ganzen Zahlen mit Knoten vom Typ Node dar.
        Ergänzen Sie fehlende Teile der Klasse entsprechend den Kommentaren an den mit TODO gekennzeichneten Stellen.
        Hinweis: Die Methode 'add' ist bereits vollständig vorgegeben.

    Punkte (maximal 16):
        8 Punkte für removeLeaves,
        8 Punkte für toString.
        Auch für teilweise korrekte Lösungen werden Punkte vergeben.
    */

    class Node {

        // Keine zusätzlichen Variablen definieren.
        private int elem;
        private Node left;
        private Node right;

        public Node() {
            this.elem = 0;
            this.left = null;
            this.right = null;
        }
        public Node(int elem) {
            this.elem = elem;
            this.left = null;
            this.right = null;
        }
        public Node(Node node) {
            this.elem = node.elem;
            this.left = node.left;
            this.right = node.right;
        }

        void add(int elem) {
            if (elem < this.elem) {
                if (this.left == null) this.left = new Node(elem);
                else this.left.add(elem);
            }
            if (elem > this.elem) {
                if (this.right == null) this.right = new Node(elem);
                else this.right.add(elem);
            }
        }
        @Override
        public String toString(){

            String erg = "";
            if(this.left!=null) erg+=this.left.toString() + ",";
            erg+=this.elem;
            if(this.right!=null) erg+="," + this.right.toString();


            return erg;
        }
        int maxDiff(){
            int min;
            int max;

            Node current = this;
            while(current.left!=null) current = current.left;
            min = current.elem;

            current = this;
            while(current.right!=null) current = current.right;
            max = current.elem;

            return max-min;
        }
        @Override
        public int hashCode() {
            int erg = 0;

            if(this.left!=null) erg+=this.left.hashCode();
            erg+=this.elem;
            if(this.right!=null) erg+=this.right.hashCode();

            return erg;
        }
        int sumlevel(int level){
            int erg = 0;

            if(level == 0){
                erg+=this.elem;
            }
            else{
                if(this.left!=null) erg += this.left.sumlevel(level-1);
```

```
                if(this.right!=null) erg += this.right.sumlevel(level-1);
            }

            return erg;
        }
        void removeLeaves(){
            if(this.left.checkLeave())this.left = null;
            if(this.right.checkLeave())this.right = null;

            if(this.left!=null) this.left.removeLeaves();
            if(this.right!=null) this.right.removeLeaves();
        }
        boolean checkLeave(){
           return this.left == null&&this.right == null;
        }

    }

    private Node root;

    public void add(int elem){
        if (this.root == null) this.root = new Node(elem);
        else this.root.add(elem);
    }
    @Override
    public String toString(){
        if (this.root == null) return "[]";
        else return "[" + this.root.toString() + "]";
    }
    public int maxDiff(){
        return this.root.maxDiff();
    }
    @Override
    public int hashCode(){
        return this.root.hashCode();
    }
    public int sumlevel(int level){
        if(level == 0) return this.root.elem;
        else return this.root.sumlevel(level);
    }
    public void removeLeaves(){


       this.root.removeLeaves();
    }




    // Nur zum Testen (geht nicht in die Beurteilung ein).
    public static void main(String[] args) {

        Tree tree = new Tree();
        tree.add(5);
        tree.add(2);
        tree.add(7);
        tree.add(6);
        tree.add(8);
        tree.add(1);
        tree.add(1);
        tree.add(3);
        tree.add(10);
        tree.add(9);
        tree.add(4);

        System.out.println(tree);
        System.out.println();
        System.out.println("maxDiff: " + tree.maxDiff());
        System.out.println("hashcode: " + tree.hashCode());
        System.out.println("sumlevel: " + tree.sumlevel(4));
        System.out.println(tree);
        tree.removeLeaves();
        System.out.println(tree);
    }

}
```