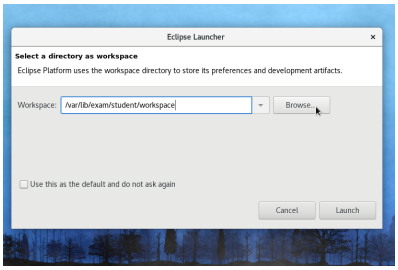
Anmelden und Eclipse starten

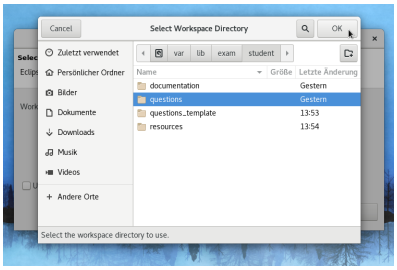
1. Sobald die Programmierprüfung startet, können Sie sich an Ihrem Computer anmelden. Geben Sie zuerst Ihren vollen Namen und im nächsten Schritt Ihren NETHZ-Namen und Ihre Legi-Nummer ein. (Sie brauchen *nicht* Ihr NETHZ-Passwort.)
2. Starten Sie Eclipse, indem Sie oben links auf "Aktivitäten" klicken und dann im Suchfeld "Eclipse" eingeben. Wählen Sie "Eclipse" (*nicht* "Eclipse C/C++"). Warten Sie, bis Eclipse gestartet ist. Dies kann einige Minuten in Anspruch nehmen.

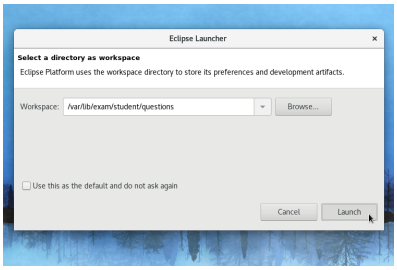
3. Wenn sich das Fenster "Eclipse Launcher" öffnet, stellen Sie sicher, dass der richtige Prüfungs-Workspace ausgewählt ist. Im Feld "Workspace" sollte folgender Pfad stehen, bevor Sie auf "Launch" klicken:

   /var/lib/exam/student/questions

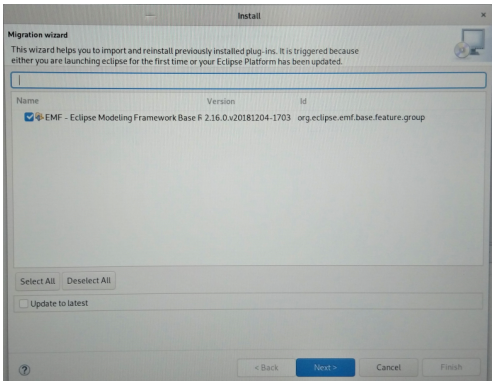
   Falls dies nicht der Fall ist, klicken Sie auf "Browse..." und wählen Sie dann im Auswahldialog den "questions"-Ordner aus. Klicken Sie oben rechts auf "OK" und dann unten auf "Launch".

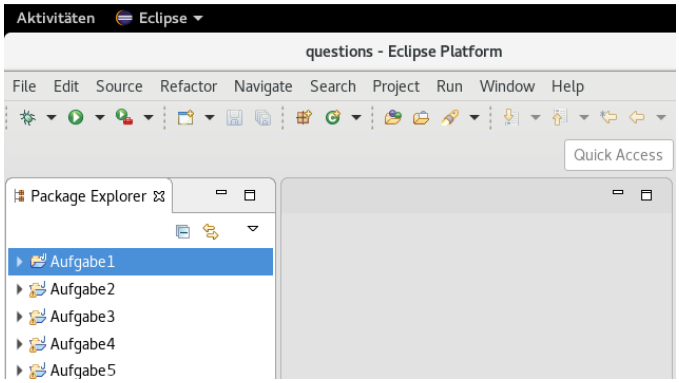






4. Nachdem der Workspace geöffnet wurde, erscheint ein Migration Wizard. Klicken Sie "Cancel". Wenn Eclipse fertig gestartet ist, sehen Sie den Willkommens-Bildschirm. Klicken Sie wenn nötig oben rechts auf "Workbench". Nun sollten Sie links die fünf Projekte "Aufgabe 1" bis "Aufgabe 5" sehen. Viel Spass!





Hinweise

Während der Programmierprüfung dürfen Sie nicht mehr an der schriftlichen Prüfung weiterarbeiten, auch wenn diese noch nicht eingezogen worden ist. **Dies gilt als Täuschungsversuch.**

1. Im Prüfungsraum bitte keine Gespräche oder Lärm.
2. Die Programmierprüfung dauert 3 Stunden (180 Minuten). Falls Sie sich durch irgendjemanden oder irgendetwas gestört fühlen, oder technische Probleme an Ihrem Computer auftreten, so melden Sie dies sofort der Aufsichts. (Falls es unerwartete Fehlermeldungen gibt: Lassen Sie solche Fehlermeldungen oder PopUp Nachrichten auf dem Bildschirm und informieren Sie die Aufsicht.) Sollten Sie durch die Behandlung eines technischen Problems Zeit verlieren, so werden Sie die verlorene Zeit nachholen können.
3. Die Prüfung hat 10 Seiten. Vergewissern Sie sich, dass Ihr Exemplar vollständig ist. Die letzten zwei Seiten können Sie für Skizzen o.ä. benutzen, aber diese werden nicht für die Benotung hinzugezogen.
4. Lesen Sie die Aufgabenstellungen genau durch. Es ist wichtig, dass Ihre Antworten den Anforderungen der Aufgaben *genau* entsprechen.
5. Wenn Sie zur Toilette müssen, melden Sie dies einer Aufsichtsperson durch Handzeichen und legen Sie Ihre Maske an. Es darf zur gleichen Zeit immer nur eine Studentin oder ein Student zur Toilette.
6. Wir beantworten keine inhaltlichen Fragen während der Prüfung.
7. Eine gut gelöste Aufgabe gibt mehr Punkte als zwei halb gelöste Aufgaben.
8. Jede Aufgabe ist mit einer Anzahl von Sternen **(★)** versehen, welche ungefähr den Aufwand und die erreichbare Punktzahl der Aufgabe widerspiegeln. Je mehr Sterne, desto aufwändiger.
9. Für jede Aufgabe gibt es ein separates Java-Projekt in Ihrem Eclipse-Workspace.
10. Die Programmieraufgaben werden vorwiegend automatisch getestet und bewertet. Programme, welche nicht mindestens teilweise ein korrektes Resultat zurückgeben (oder gar nicht erst kompilieren), erhalten keine Punkte.
11. Stellen Sie regelmässig sicher, dass Ihre Dateien *im Workspace* gespeichert sind. Nur diese Dateien werden von einem Backup-Prozess während der Prüfung gespeichert. Was nicht gespeichert ist, kann nicht bewertet werden.
12. Sollten Sie eine Ihrer Lösungsdateien überschreiben, so kann die Aufsicht Ihnen helfen! Melden Sie sich sofort.
13. Ändern Sie unter keinen Umständen die Signaturen der im Aufgabentext erwähnten Methoden (Name, Typ und Reihenfolge der Parameter), ihren Rückgabetyp, Modifizierer wie `static`, `public` oder gegebenfalls die Liste der geworfenen Exceptions. Das gleiche gilt für Signaturen von Konstruktoren. Auch die Namen der erwähnten Klassen dürfen Sie nicht ändern. Solche Änderungen können dazu führen, dass Sie keine Punkte für die Aufgabe erhalten. Wenn nicht anders vermerkt, dürfen Sie Methoden, Attribute oder Klassen zu den vorhandenen hinzufügen. Ebenso dürfen Sie, sofern keine Einschränkungen aufgeführt sind, Klassen importieren.
14. In jedem Projekt gibt es neben dem "src"-Ordner einen "test"-Ordner mit einigen JUnit-Tests. Wir empfehlen, diese mit ihren eigenen Tests zu erweitern. **Tests werden nicht bewertet.**
15. Falls gewisse Tests beim Ausführen scheinbar keine Resultate liefern, könnte es daran liegen, dass Ihre Lösung eine Endlosschleife enthält. Stoppen Sie in diesem Fall die Tests von Hand, und zwar mit dem "Terminate"-Knopf in der "Console View"; ansonsten kann Ihr System einfrieren, was Zeit kostet.
16. Die Prüfungscomputer haben keinen Internet-Zugang. Dadurch kann es in Eclipse zu Fehlermeldungen kommen. Meldungen im Zusammenhang mit fehlendem Internet-Zugang können Sie ignorieren.
17. Die Dokumentation der Java-Klassen ist offline vorhanden. Sie können die "Javadoc"-Ansicht im unteren Teil des Eclipse-Fensters verwenden, um Informationen zu Klassen und Methoden zu erhalten.
18. Als zusätzliche Sicherheitsmassnahme wird Ihr Bildschirm während der Prüfung aufgezeichnet.
19. Wenn Sie in der IDE Zeichen ersetzen statt einfügen, dann drücken Sie die Insert Taste (über der Delete Taste).
20. Auf einer Schweizer Tastatur schreiben Sie eckige und geschweifte Klammern durch `Alt` + `Ctrl` + die entsprechende Taste links neber der Enter Taste.
21. Wenn Sie früher abgeben wollen, melden Sie sich bitte lautlos, und wir holen die Prüfung ab. Vorzeitige Abgaben sind nur bis 20 Minuten vor Prüfungsende möglich.
22. Verlassen Sie bitte den Prüfungsraum *leise* nach der Prüfung. Es kann sein, dass andere noch weiterarbeiten da sie eine Zeitgutschrift bekommen haben. Auch diese Kandidaten sollen in Ruhe arbeiten können. Nehmen Sie bitte keine Aufgabenstellung mit – wir sammeln diese später ein.

Aufgabe 1 (★)

In dieser Aufgabe sollen Sie Scrabble-Steine legen, mittels ASCII-Art auf der Konsole. Vervollständigen Sie die Methode `drawNameSquare` in der Klasse `Scrabble`. Diese Methode nimmt einen Namen als String-Parameter und soll den Namen als in einem Quadrat angeordnete Scrabble-Steine auf der Konsole (`System.out`) ausgeben. Wenn z.B. der String `Alfred` übergeben wird, sollte folgendes Bild ausgegeben werden:

```
+---+---+---+---+---+---+
| A | L | F | R | E | D |
+---+---+---+---+---+---+
| L |               | E |
+---+               +---+
| F |               | R |
+---+               +---+
| R |               | F |
+---+               +---+
| E |               | L |
+---+---+---+---+---+---+
| D | E | R | F | L | A |
+---+---+---+---+---+---+
```

Ihr Code braucht nur Namen der Länge 3 oder länger unterstützen. Für einen Namen mit Länge 3, z.B. `Jim`, sollte die Ausgabe so aussehen:

```
+---+---+---+
| J | I | M |
+---+---+---+
| I |   | I |
+---+---+---+
| M | I | J |
+---+---+---+
```

Diese beiden Beispiele finden Sie als Unit Tests im “tests”-Ordner in der Vorlage. Beachten Sie, dass Ihr Programm keinerlei andere Ausgabe als das Scrabble-Quadrat machen darf.

Aufgabe 2 (★)

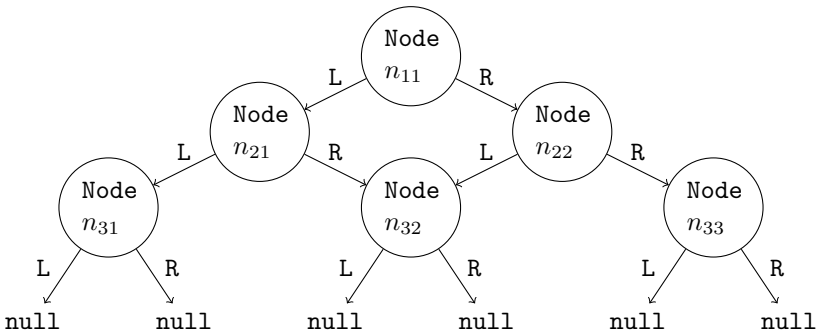
Gegeben sei eine $n \times n$ Matrix M deren Elemente positive ganze Zahlen sind und für die gilt $0 < m_{i,j} \leq n^2$ und $m_{x,y} = m_{p,q} \Rightarrow (x = p) \wedge (y = q)$. Wir sagen, dass die Matrix M *perfekt* ist, wenn zusätzlich alle Zeilensummen und Spaltensummen gleich sind (also $\sum_{k=0}^{k=n-1} m_{i,k} = \sum_{k=0}^{k=n-1} m_{j,k}$ für alle i, j und $\sum_{k=0}^{k=n-1} m_{k,i} = \sum_{k=0}^{k=n-1} m_{k,j}$ für alle i, j mit $0 \leq i, j < n$).

Vervollständigen Sie die Methode `boolean checkMatrix(int [][] m)`, so dass diese Methode `true` zurückgibt wenn die Input Matrix *perfekt* ist, und `false` sonst. Sie können davon ausgehen, dass der Parameter `m` nicht `null` ist. Alle anderen Eigenschaften, müssen Sie selber testen. Eine Matrix ist nur perfekt, wenn alle genannten Eigentschaften gelten.

Aufgabe 3 (★★)

Die Klasse `Node` repräsentiert einen Knoten in einem gerichteten Graphen, wobei es für jeden Knoten n_1 höchstens zwei gerichtete Kanten von n_1 zu anderen Knoten n_2, n_3 geben kann (n_2 und n_3 können gleich sein). Wir unterscheiden dabei zwischen dem linken und dem rechten Knoten. Die Methode `Node.getLeft()` gibt den linken Knoten und `Node.getRight()` den rechten Knoten zurück (als `Node`-Objekt). Wenn der linke Knoten von n_1 nicht existiert, dann gibt `Node.getLeft()` `null` zurück (analog für den rechten Knoten).

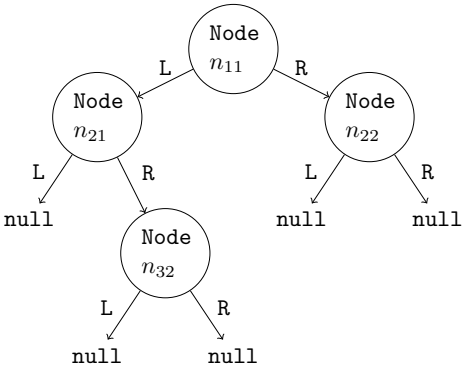
Das Ziel dieser Aufgabe ist, für ein `Node`-Objekt zu entscheiden, ob der durch das `Node`-Objekt definierte Graph einer Pyramide entspricht. Zum Beispiel entspricht der folgende Graph einer Pyramide:

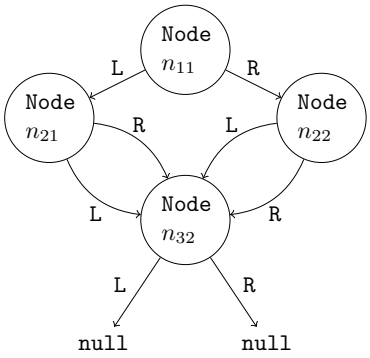


Beachten Sie, dass der rechte Knoten von n_{21} gleich ist wie der linke Knoten von n_{22} (das heisst die Node-Objekte sind gleich!). Ein Graph (wie oben repräsentiert) definiert eine Pyramide genau dann, wenn folgende Bedingungen gelten:

- Der Graph kann in $k \geq 1$ Stufen (Stufe 1, Stufe 2,..., Stufe k) aufgeteilt werden, wobei Stufe i aus i unterschiedlichen Knoten n_{i1}, n_{i2}, ..., n_{ii} besteht. Falls der Graph k Stufen hat, dann hat dieser genau $\frac{k(k+1)}{2}$ unterschiedliche Knoten (Knoten aus verschiedenen Stufen sind unterschiedlich).
- Für Stufe i ($1 \leq i < k$) gilt: der linke Knoten von n_{ij} ($1 \leq j \leq i$) ist durch $n_{(i+1)j}$ gegeben und der rechte Knoten von n_{ij} ist durch $n_{(i+1)(j+1)}$ gegeben.
- Für Stufe k gilt: es gibt keinen linken und keinen rechten Knoten für n_{kj} ($1 \leq j \leq k$).

Die folgenden Graphen entsprechen zum Beispiel keinen Pyramiden:





Implementieren Sie die `Pyramid.isPyramid(Node node)`-Methode, welche, für den Graph G durch `node` definiert, entscheidet, ob G eine Pyramide definiert. Sie dürfen annehmen, dass G keine Zyklen hat.

Tipp: Prüfen Sie die Bedingungen Stufe für Stufe, beginnend bei Stufe 1.

Aufgabe 4 (★★★)

In dieser Aufgabe implementieren Sie das Punktesystem von Hogwarts, bei welchem Studenten eines Hauses Punkte verliehen oder abgezogen bekommen können und dadurch der kumulative Punktestand ihres Hauses sich verändert. Wir verwenden drei Klassen, `School`, `House`, und `Student`, für die Schule, Häuser, und Studenten. Die Klassen können folgendermassen verwendet werden:

```
School hogwarts = new School();

// Haeuser werden erstellt (Anzahl Haeuser und Namen sind nicht eingeschraenkt).
House hufflepuff = hogwarts.createHouse("Hufflepuff");
House ravenclaw  = hogwarts.createHouse("Ravenclaw");

// Studenten haben Vor- und Nachnamen.
Student hannah = new Student("Hannah", "Abbott");
Student newton = new Student("Newton", "Scamander");
Student luna   = new Student("Luna",   "Lovegood");
Student filius = new Student("Filius", "Flitwick");

// Studenten werden den Haeusern zugeordnet.
hufflepuff.assign(hannah);
hufflepuff.assign(newton);
ravenclaw.assign(luna);
ravenclaw.assign(filius);

// Punkte werden an Studenten vergeben. Punkte koennen auch negativ sein.
hannah.givePoints(10);
newton.givePoints(-5);
luna.givePoints(8);

// Informationen zu der Summe an Punkten und dem aktuellen Siegerhaus
// koennen immer abgefragt werden.
System.out.println("Siegerhaus: " + hogwarts.winner().name());
System.out.println("Siegerpunkte: " + hogwarts.winner().points());
System.out.println("Hogwarts Punkte Insgesamt: " + hogwarts.points());
```

In der Vorlage finden Sie das Skelett für die drei Klassen sowie eine Test-Klasse `SchoolTest`, welche Sie als Starthilfe für das Testen Ihrer Lösung verwenden können.

(a) **(⯪)** Implementieren Sie den `School`-Konstruktor und die Methode `createHouse(String name)`, welche als Parameter den gewünschten Namen des Hauses nimmt und ein `House` Objekt zurück zurückgibt. Der Name eines Hauses darf nicht `null` sein oder bereits für die Schule vorhanden sein. Die Methode soll in diesen Fällen eine `IllegalArgumentException` werfen. Alle anderen Namen sind erlaubt. Implementieren Sie zusätzlich die Methode `name()` der Klasse `House`, welche den Namen des Hauses als String zurückgibt.

(b) **(★)** Implementieren Sie den Konstruktor von `Student`, welcher zwei Strings, den Vor- und Nachnamen (in dieser Reihenfolge) nimmt. Sie dürfen annehmen, dass es jeden Namen (Vor- und Nachname zusammen) nur einmal gibt. Vor- und Nachnamen sollen über die Methode `firstName()` beziehungsweise `lastName()` erhalten werden können. Implementieren Sie zusätzlich die Methode `assign(Student student)` der Klasse `House`, welche einen Studenten als Argument nimmt und ihn in dieses Haus einschreibt. Bei einem `null` Argument oder falls der Student bereits bei einem Haus der gleichen Schule eingeschrieben ist, dann soll die Methode eine `IllegalArgumentException` werfen.

(c) **(★⯪)** Als letztes implementieren Sie das Punktesystem. Implementieren Sie dafür vier Methoden: Die Method `points()` von `House` gibt die Punkte eines Hauses zurück. Jedes Haus beginnt mit einem Punktestand von 0, wenn es erstellt wird. Dieser Punktestand kann sich dann durch die Leistungen der Studenten verändern. Die Methode `givePoints(int points)` von `Student` nimmt eine positive oder negative Anzahl Punkte, welche dem Studenten verliehen werden. Erhaltene Punkte zählen nur, wenn der Student einem Haus bereits zugewiesen wurde. Die erhaltenen Punkte werden dann den Häusern zugeschrieben, welchen der Student zugewiesen ist. Dabei können die Punkte eines Hauses nicht kleiner als 0 werden. Auch wenn einem Studenten mehr Punkte abgezogen werden, geht der Punktestand eines Hauses nur auf 0. Zum Beispiel, wenn Hufflepuff in der Summe 5 Punkte hat und Hannah -10 Punkte verliehen werden, dann werden nur -5 Punkte tatsächlich für Hufflepuff verrechnet, der Rest wird ignoriert. Zusätzlich implementieren Sie die Methode `winner()` von `School`, welche das Haus mit den meisten Punkten zurückgibt. Falls mehrere Häuser die gleiche Punktzahl haben, dann kann ein beliebiges dieser Häuser zurückgegeben werden. Falls es kein Haus mit Punkten gibt, dann soll die Methode eine `IllegalArgumentException` werfen. Und implementieren Sie die Methode `points()` von `School`, welche die Summe der Punktestände der Häuser zurückgibt.

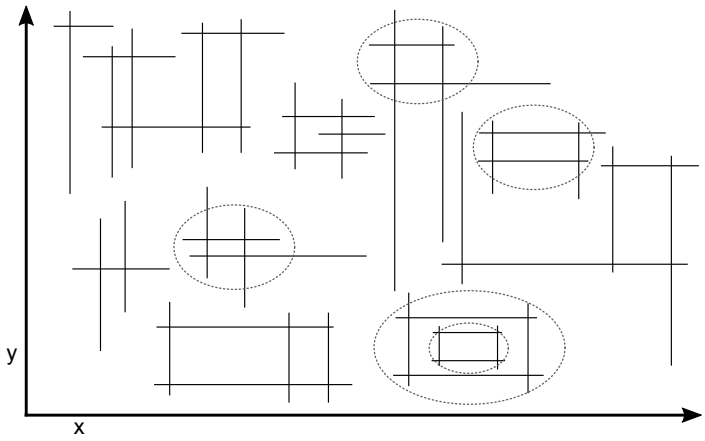
Aufgabe 5 (★★★★)

In dieser Aufgabe entwickeln Sie einen Algorithmus, welcher die Anzahl aller freistehenden Rechtecke eines Arrays von Linien-Segmenen berechnet und implementieren ihn in der Methode `findeRechtecke(Segmente[] linien)`. Ein Linien-Segment besteht aus zwei Koordinaten (A_x, A_y) und (E_x, E_y) in der zweidimensionalen (X,Y) Ebene, wobei $A_x <= E_x$ und $A_y <= E_y$ gilt.

Untenstehend ein Beispiel mit 5 Rechtecken (eingekreist). Die 4 Segmente, welche ein freistehendes Rechteck ausmachen, überschneiden sich nicht mit anderen Segmenten (jedes Segment eines Rechteckes schneidet exakt 2 andere Segmente).

Weiter dürfen Sie folgende Annahmen machen:

- Alle Segmente sind entweder horizontal oder vertikal, d.h., $A_x = E_x$ oder $A_y = E_y$.
- Alle Segmente haben mindestens Länge 3.
- Zwei horizontale oder zwei vertikale Segmente schneiden sich nie.
- Kein Segment kann durch ein anderes Segment verlängert werden, d.h. wenn es ein Segment S mit $A = (2, 2)$ und $E = (2, 5)$ gibt, dann gibt es kein Segment T mit $A = (2, 5)$ und $E = (2, z)$.



(a) **(★★)** Implementieren Sie die Methode `schneidetSich(Segment linie)`, welche `true` zurück gibt falls sich das Segment-Objekt mit dem Argument der Methode schneidet.

(b) **(★★)** Implementieren Sie die Methode `findeRechtecke(Segment[] linien)`, welche ein Array von Segmenten akzeptiert und die **Anzahl** der Rechtecke (wie oben definiert) berechnet.

```
public class Segment {
  public final int start_x;
  public final int start_y;
  public final int end_x;
  public final int end_y;

  Segment(int sx, int sy, int ex, int ey) {
    start_x = sx;
    start_y = sy;
    end_x = ex;
    end_y = ey;
  }

  public boolean schneidetSich(Segment linie) {
    // Vervollstaendigen Sie diese Methode

  }

}
```

```
public class Rechteck {
  public static void main(String[] args) {
    Segment[] linien = {
      new Segment(2,2,2,7),
      new Segment(1,3,9,3),
      new Segment(0,6,8,6),
      new Segment(5,0,5,8)
    };
    int anzahl = findeRechtecke(linien);
    System.out.println(anzahl);
  }

  public static int findeRechtecke(Segment[] linien) {
    // TODO: Vervollstaendigen Sie diese Methode

  }

}
```

Notizen

Notizen