NimView

In dieser Dokumentation gehe ich wesentlich nur auf meinen Code ein, den ich selber geschrieben habe. Diese wären in der Nim-Klasse die Regel, dass man nur *maximal fünf Reihen und maximal sieben "Hölzchen"* haben darf, die *equals- und hashCode-Methode* und die gesamte *NimView-Klasse*.

Aufbau des Nim-Spiels

Hier werden die Eingabedaten eines Arrays *rows* im Konstruktor geprüft, um sicherzustellen, dass sie bestimmten Anforderungen entsprechen.

1. Maximale Reihenanzahl: Falls das Array *rows* mehr als 5 Elemente enthält, wird eine *IllegalArgumentException* mit der Nachricht "Es sind maximal 5 Reihen erlaubt." ausgelöst. Dies beschränkt die Anzahl der erlaubten Reihen auf 5.

2. Wertebereich jeder Reihe: Der Code iteriert durch jedes Element von *rows*. Wenn ein Element kleiner als 0 oder größer als 7 ist, wird eine weitere *IllegalArgumentException* ausgelöst, diesmal mit der Nachricht "Jede Reihe darf maximal 7 Hölzchen enthalten." Damit wird sichergestellt, dass jede Reihe nur zwischen 0 und 7 Hölzchen enthält.

```
if (rows.length > 5) throw new IllegalArgumentException("E
for(int i = 0; i < rows.length; i++) {
    if (rows[i] < 0 || rows[i] > 7) throw new IllegalArgum
}
```

equals-Methode: Sind beide Nim-Spiele gleich?

Die equals-Methode in der Klasse Nim prüft, wann zwei Nim-Objekte als gleich betrachtet werden. Die Methode überprüft mehrere Bedingungen, um die Gleichheit sicherzustellen:

Null-Check: Falls das übergebene Objekt *other null* ist, wird *false* zurückgegeben.

Selbstvergleich: Wenn *other* dasselbe Objekt wie *this* ist, wird *true* zurückgegeben, da ein Objekt immer gleich zu sich selbst ist.

Typprüfung: Wenn *other* eine andere Klasse hat als *this*, wird *false* zurückgegeben, da sie nicht vergleichbar sind.

Casting: *other* wird als Objekt der Klasse *Nim* gecastet, um auf dessen Eigenschaften zugreifen zu können.

Reihenanzahl: Es wird überprüft, ob beide Objekte dieselbe Anzahl an Reihen (*rows.length*) haben. Wenn nicht, wird *false* zurückgegeben.

Gleichheit der Werte: Es wird iterativ geprüft, ob jede Zahl in *this.rows* auch in *that.rows* vorkommt. Die Reihenfolge der Zahlen spielt dabei keine Rolle. Falls ein Wert in *this.rows* nicht in *that.rows* vorhanden ist, wird *false* zurückgegeben.

```
@Override
public boolean equals(Object other) {
    if (other == null) return false; // Null abwehren
    if (other == this) return true; // Bin ich's selbst?
    if (other.getClass() != getClass()) return false; // Ander
    Nim that = (Nim)other; // Casting
    // Was definiert die Gleichheit zweier Nim-Objekte?
    // 1. Reihenanzahl muss gleich sein:
    if (this.rows.length != that.rows.length) return false;
    // 2. Werte müssen gleich sein, aber nicht unbedingt in de
    for (int i = 0; i < this.rows.length; i++) {
        boolean found = false;
        for (int j = 0; j < that.rows.length; j++) {
            if (this.rows[i] == that.rows[j]) {
                found = true;
                break;
            }
        }
        if (!found) return false;
```

```
        }
        return true;
    }
```

hashCode-Methode: Gleicher Hashcode, wenn zwei oder mehrere Nim-Spiele “equal” sind

Die hashCode-Methode liefert einen eindeutigen Hash-Wert für ein *Nim-Objekt*. Der Hash-Wert hilft, *Nim*-Objekte effizient in Hash-Tabellen wie *HashMap* oder *HashSet* zu speichern.

Kopieren und Sortieren: Eine Kopie des Arrays rows wird erstellt und in sortedRows gespeichert, die dann sortiert wird. Dies stellt sicher, dass zwei Nim-Objekte mit denselben Werten, aber in unterschiedlicher Reihenfolge, denselben Hash-Wert haben.

Prime-Faktor und Initialisierung: Der Wert *prime = 31* wird als Multiplikator verwendet, da Primzahlen eine gute Streuung für Hash-Werte fördern. Die Variable *hash* wird initial auf 0 gesetzt.

Berechnung des Hash-Werts: Für jedes Element in *sortedRows* wird hash mit dem Wert von *prime* multipliziert und der aktuelle Wert des Arrays hinzugefügt. So entsteht ein eindeutiger Hash, der von den Werten in *rows* abhängt.

Rückgabe des Hash-Werts: Der berechnete hash-Wert wird zurückgegeben.

```
    @Override
    public int hashCode() {
        int[] sortedRows = Arrays.copyOf(rows, rows.length);
        Arrays.sort(sortedRows);
        int prime = 31;
        int hash = 0;
        for (int i = 0; i < sortedRows.length; i++) {
            hash = hash * prime + sortedRows[i];
        }
        return hash;
    }
```

NimView-Klasse: LiveViewProgramming

Die NimView-Klasse dient zur visuellen Darstellung und Verwaltung eines Nim-Spiels. Sie kombiniert die Spiellogik der *Nim*-Instanz mit der grafischen Ausgabe mittels einer Turtle-Grafik. Jede *NimView*-Instanz zeigt den aktuellen Spielzustand an und ermöglicht die Durchführung von Zügen sowie das Abrufen möglicher Züge.

Attribute

static Turtle turtle: Eine statische *Turtle*-Instanz, die verwendet wird, um das Spielfeld anzuzeigen. Durch die statische Deklaration wird verhindert, dass mehrere Turtle-Grafiken gleichzeitig gezeichnet werden.

Nim nimGame: Eine Instanz der *Nim*-Klasse, die den aktuellen Zustand des Spiels speichert und die Spiellogik bereitstellt.

Konstruktor

NimView(Nim nimGame): Initialisiert eine neue *NimView*-Instanz mit einer *Nim*-Instanz, die den Anfangszustand des Spiels enthält.

Methoden

void show(): Zeichnet das aktuelle Spielfeld mithilfe der Turtle-Grafik. Die Turtle wird an die Startposition versetzt und zeichnet vertikale Linien (“Hölzchen”) entsprechend dem Zustand der *nimGame*-Instanz. Nach jedem *turtle.reset()* wird das Spielfeld komplett neu gezeichnet, um den aktuellen Zustand darzustellen.

NimView play(Move move): Führt einen Spielzug aus und gibt ein neues *NimView-Objekt* zurück, das den aktualisierten Zustand des Spiels enthält. Der Move wird über die *Nim*-Instanz ausgeführt, wodurch eine neue *Nim*-Instanz mit dem modifizierten Zustand erzeugt wird. Anschließend wird das neue Spielfeld über die *show()*-Methode dargestellt.

Move bestMove(): Gibt den besten Spielzug für den aktuellen Zustand des Spiels zurück. Diese Methode delegiert den Aufruf an die *bestMove()*-Methode der *nimGame*-Instanz.

Move randomMove(): Gibt einen zufälligen, gültigen Spielzug für den aktuellen Zustand des Spiels zurück. Der Aufruf wird an die *randomMove()*-Methode der *nimGame*-Instanz weitergeleitet.

boolean isGameOver(): Prüft, ob das Spiel beendet ist. Wenn alle Reihen leer sind, gibt die Methode *true* zurück. Ansonsten gibt sie *false* zurück. Auch hier wird der Aufruf an die *nimGame*-Instanz delegiert.

String toString(): Gibt eine textuelle Darstellung des aktuellen Spiels zurück, indem die *toString()*-Methode der *nimGame*-Instanz aufgerufen wird. Zusätzlich ruft *toString()* die *show()*-Methode auf, um das Spielfeld in der aktuellen Grafik darzustellen.

```
class NimView {
    static Turtle turtle = new Turtle(500, 500);
    Nim nimGame;
    NimView(Nim nimGame) {
        this.nimGame = nimGame;
    }

    void show() {
        turtle.reset();
        int[] rows = nimGame.rows;
        // Startposition
        int start = 200;
        turtle.penUp().backward(start).left(90).forward(start).rig

        for (int r = 0; r < rows.length; r++) {
            for(int i = 0; i < rows[r]; i++) {
                // Hölzchen zeichnen
                turtle.forward(40).penUp().left(90).forward(30).le
            }
            // in eine neue Reihe gehen
            turtle.penUp().left(90).backward(30 * rows[r]).right(9
        }
    }

    // Objekt aktualisieren
    public NimView play(Move move) {
        Nim updatedNim = nimGame.play(move);
        NimView newView = new NimView(updatedNim);
        return newView; // Neues NimView-Objekt zurückgeben
    }

    // Liefert den besten Spielzug aus der aktuellen Nim-Instanz
    public Move bestMove() {
```

```
        return nimGame.bestMove();
    }

    // Liefert einen zufälligen Spielzug aus der aktuellen Nim-Ins
    public Move randomMove() {
        return nimGame.randomMove();
    }

    // Prüft, ob das Spiel beendet ist
    public boolean isGameOver() {
        return nimGame.isGameOver();
    }

    @Override
    public String toString() {
        show();
        return nimGame.toString();
    }
}
```