

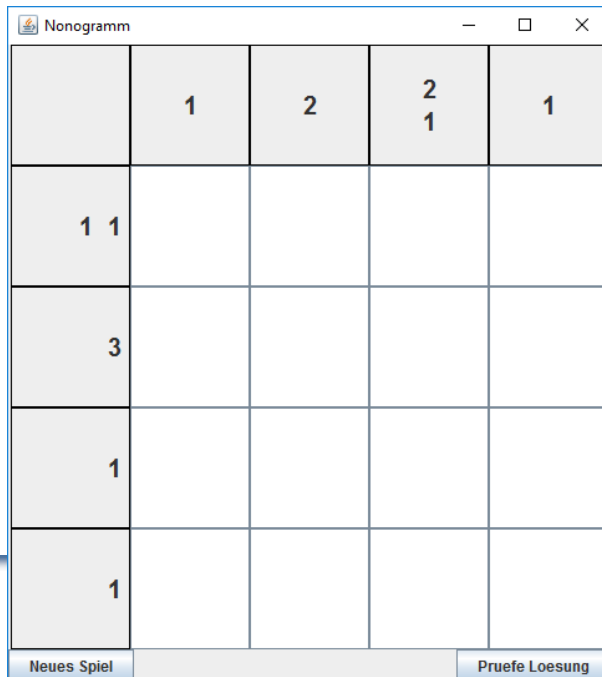
Aufgaben zur Abgabe als Prüfungsleistung für Prog II

- Regeln wie bisher
- Falls Anforderungen nicht erfüllt sind, dann bitte im **Kopf des Hauptprogramms (Beginning Comment)** dokumentieren
 - Beispiel: Resize des Fensters funktioniert nicht, Prüfung geht im Spezialfall xyz nicht usw.

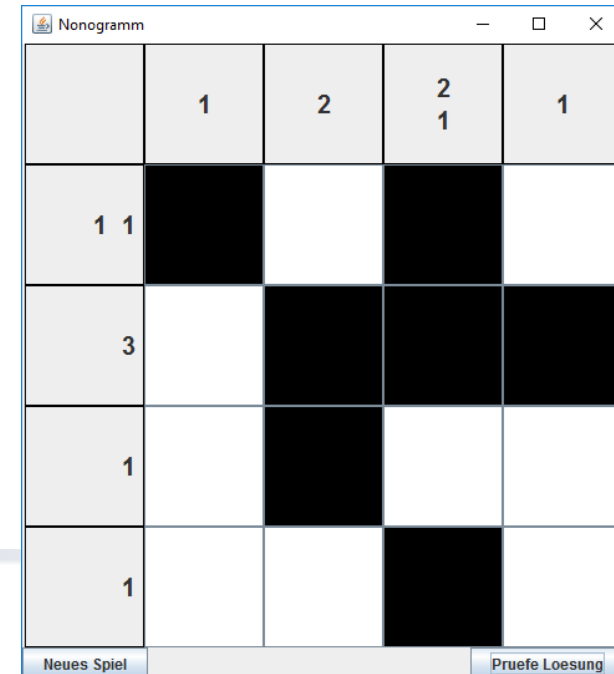
Aufgabe 2

- Schreiben Sie ein Programm, mit dem Sie ein „Nonogramm“ spielen können
 - Das Spiel ist ein Zahlenpuzzle: Ein leeres, weißes Spielfeld (ein quadratisches Gitter z.B. der Größe 4 x 4) wird vom Benutzer mit schwarzen Kästchen befüllt
 - Ziel ist es, dass die Zeilensummen und Spaltensumme der schwarzen Kästchen mit den Angaben je Zeile bzw. je Spalte übereinstimmen
 - Mehr als eine Ziffer in einer Zeile/Spalte (z.B. 2 und 1) bedeuten, dass erst zwei schwarze Kästchen kommen, dann mind. ein leeres und dann ein schwarzes

Start:



Lösung:



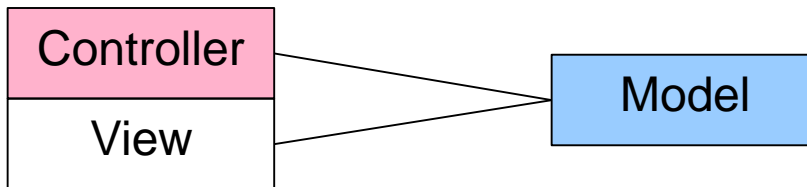
Weitere Funktionen

- Bedienung
 - Mit der linken Maustaste wird ein schwarzes Kästchen erzeugt, ein erneuter Klick mit der linken Maustaste nimmt das schwarze Kästchen wieder weg und erzeugt ein weißes
 - Mit der rechten Maustaste wird ein Kreuz erzeugt. Dieses Kreuz setzt der Benutzer, wenn er weiß, dass manche Zellen nicht befüllt werden können
 - Die linke Maustaste überschreibt auch ein eventuell vorhandenes Kreuz, die rechte Maustaste überschreibt auch ein eventuell vorhandenes schwarzes Kästchen
 - Es gibt einen Button „Prüfe Lösung“, der untersucht, ob die Zeilen- und Spaltensummen alle erfüllt sind; falls nicht, wird ein Hinweis ausgegeben, in welcher Zeile und Spalte ein Fehler zu finden ist
 - Sind mehr als eine fehlerhafte Zeile/Spalte vorhanden, kann ein beliebiger Fehler ausgegeben werden, z.B. der erste oder der letzte



Anforderungen

- Das Design soll einem vereinfachten Model-View-Controller (MVC) Entwurfsmuster folgen:
 - Model = Modell, enthält die darzustellenden Daten und Geschäftslogik, repräsentiert den internen Zustand eines Systems und speichert alle interessanten Geschäftsdaten
 - View = Oberfläche, Präsentation, d.h. Komponenten wie Fenster, Buttons, etc. Die View stellt die Daten des Models zur Ansicht dar. Die View nutzt das Model, um die Informationen auszulesen.
 - Controller = Steuerung, die die Folgeaktionen einleitet. Nach einer Interaktion mit der grafischen Oberfläche werden die Daten im Modell aktualisiert und anschließend vom View neu angezeigt
- Hier Vereinfachung: View und Controller sind in Java eng verbunden, deshalb werden V und C zu **einer** Komponente verschmolzen



Anforderungen

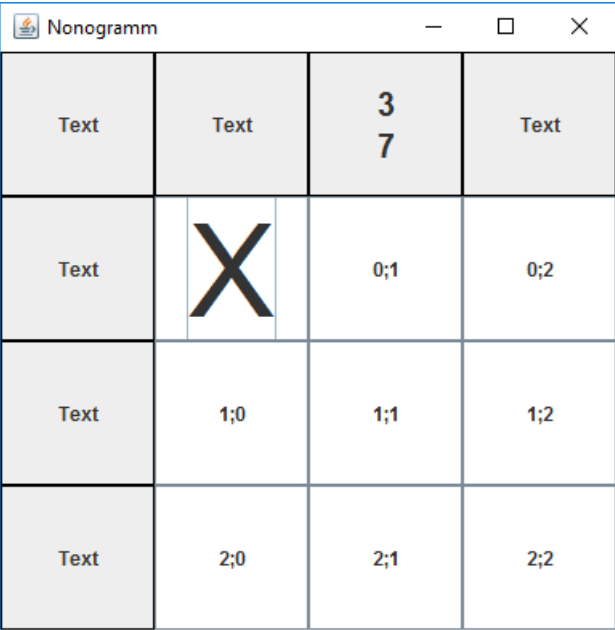
- Funktionale Anforderungen der Anwendung
 - Erfüllen der Spielregeln und der Bedienung wie eben beschrieben
 - Hartcodierte Initialisierung des 4 x 4-Beispiels entsprechend der Vorbelegung hier in den Folien beim Start der Anwendung
 - Schaltfläche mit Funktionalität zum automatischen Erzeugen und Anzeigen eines neuen Spiels
 - Dabei wird eine neue Größe des Spielfelds abgefragt
 - Die Vorbelegung soll zufällig sein, aber zum Spielen brauchbar
 - Schwierige Anforderung, 3 Punkte weniger beim Weglassen
 - Hinweis: `this.dispose()` gibt Fenster `JFrame` frei, dann neues GUI und neues Model erzeugen (vielleicht gibt es auch bessere Lösungen)

Anforderungen

- Anforderungen an das Layout/Design
 - Resize (Vergrößern/Verkleinern der Oberfläche) insgesamt möglich
 - Layout/Oberfläche entsprechend der Vorlage hier in den Folien
 - Zum Vergleich des Verhaltens siehe Beispielanwendung
- Anforderungen an die Implementierung
 - Änderbarkeit der Größe `MAX` des Spielfeldes an einer zentralen Stelle
 - Verwendung eines Aufzählungstyps `enum State` für den Zustand eines Feldes
 - Trennung von Modell und View+Controller
 - Die Hauptklasse soll `Nonogramm` heißen, dort steht auch die `main`-Methode
 - Model-Klasse heißt `Model`, ein Attribut darin: `State[][] feldStatus;`
 - In der Model-Klasse gibt es weitere Attribute, z.B. `int[][] zeileVorgabe` und `int[][] spalteVorgabe`, die die Zeilensummen und Spaltensumme verwalten. Die Größe dieser Felder darf `MAX` mal `MAX` sein, auch wenn tatsächlich nur rund die Hälfte davon benötigt wird
 - Weitere Hilfsklassen sind natürlich möglich

Hinweise

- Siehe Hilfsklasse `Vorlage.java`: Enthält viele Hinweise für die Oberfläche
 - Es kann hilfreich sein, die Buttons (= Zellen auf dem Spielfeld) intern mit einer eindeutigen x/y-Koordinate zu beschriften:
`myButton[x][y].setName(x + ";" + y);`
 - Im Listener lassen sich diese Namen wieder auslesen:
`component.getName();`
- Mögliches Vorgehen für eigene Implementierung
 - Oberfläche erweitern, dann Model hinzunehmen, dann Button Eingaben prüfen, dann Button neues Spiel
 - Tipp: Ein Model (= Attribut von `Nonogramm`) kann man mittels Konstruktor oder Setter/Getter auch an andere Klassen übergeben (falls man das möchte)
 - Tipp: Button `Pruefe Loesung` könnte eine Kopie des `feldStatus` verwenden, um geprüfte Zellen zu löschen



Text	Text	3 7	Text
Text	X	0;1	0;2
Text	1;0	1;1	1;2
Text	2;0	2;1	2;2