Suchbäume und GUI

Plenum Programmierpraktikum

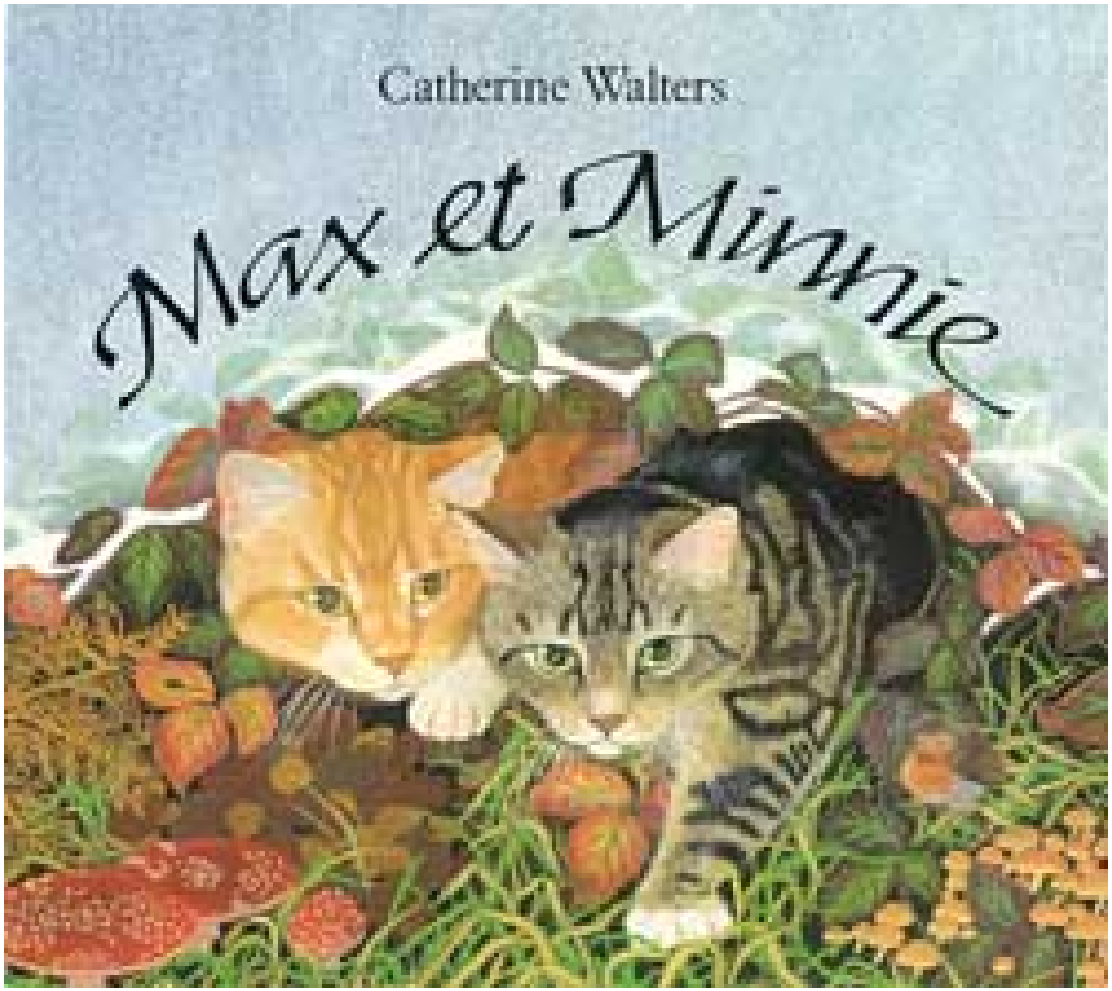
2006-11-30

Warum zuerst einen Textmodus?

. . . wenn man später sowieso eine GUI will?

- Sorgt für klareres Design, da man Fachobjekte (engl. *domain objects*) und Benutzeroberfläche klar trennen muss.
- Genauso denkbar: Web-Oberfläche.
- Ähnliche Änderungen für Unit-Tests (die ja auch nur eine weitere Benutzeroberfläche für die Fachobjekte sind).

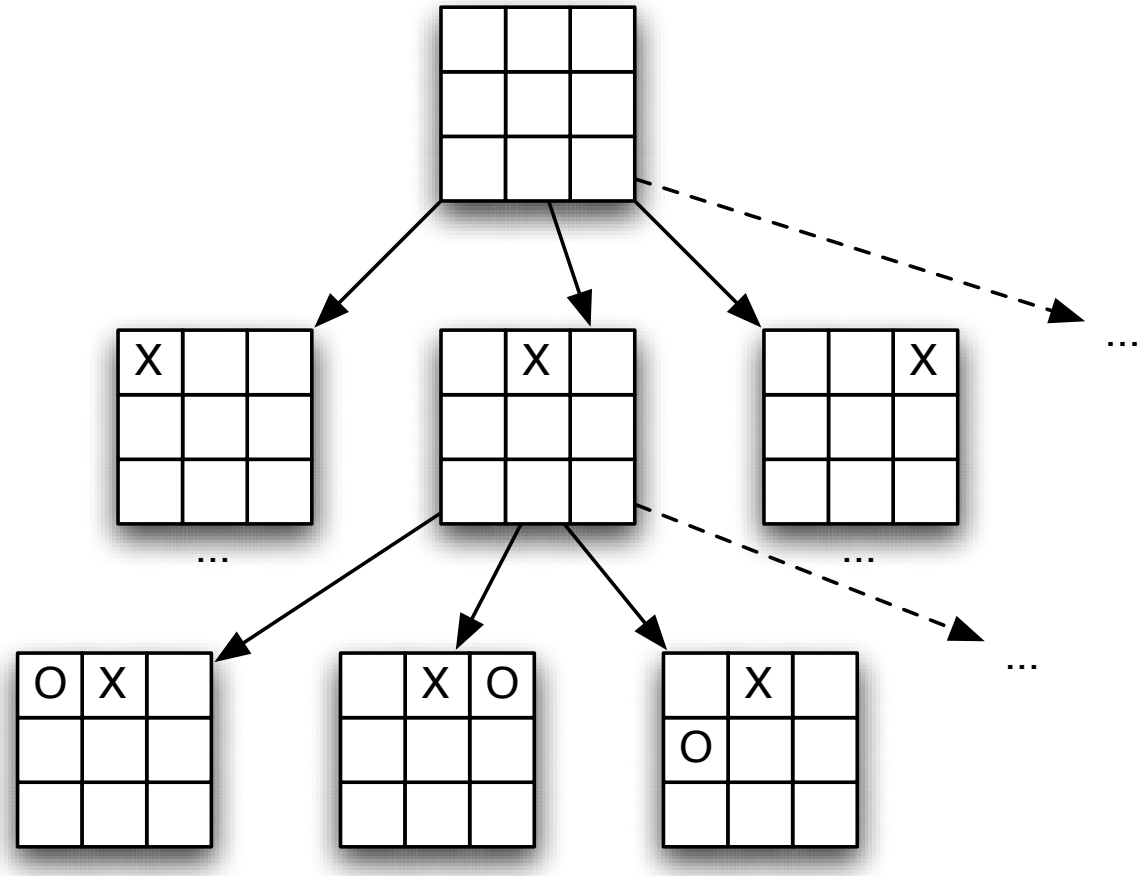
Suchbäume


Catherine Walters
Max et Minnie

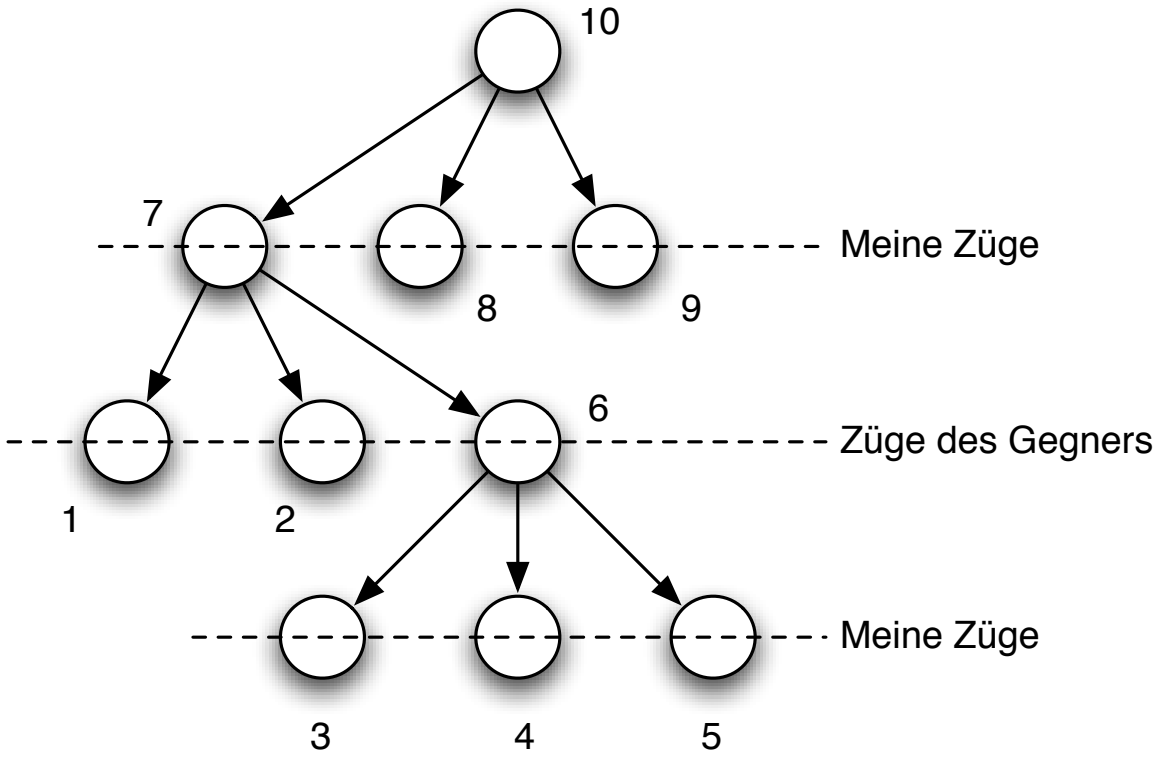
Minimax-Suche

- Motivation: Wie kann ich unter den nächsten Zügen im TacTacToe den besten auswählen?
- Tiefensuche (Kinder vor den Eltern) durch alle möglichen Züge.
- Bewerte die Blatt-Felder. Je höher die Punkte, desto besser für mich.
- Propagiere die Blattwerte richtung Wurzel: Ich maximiere bei meinen Zügen, der Gegner minimiert.

Züge vorausberechnen

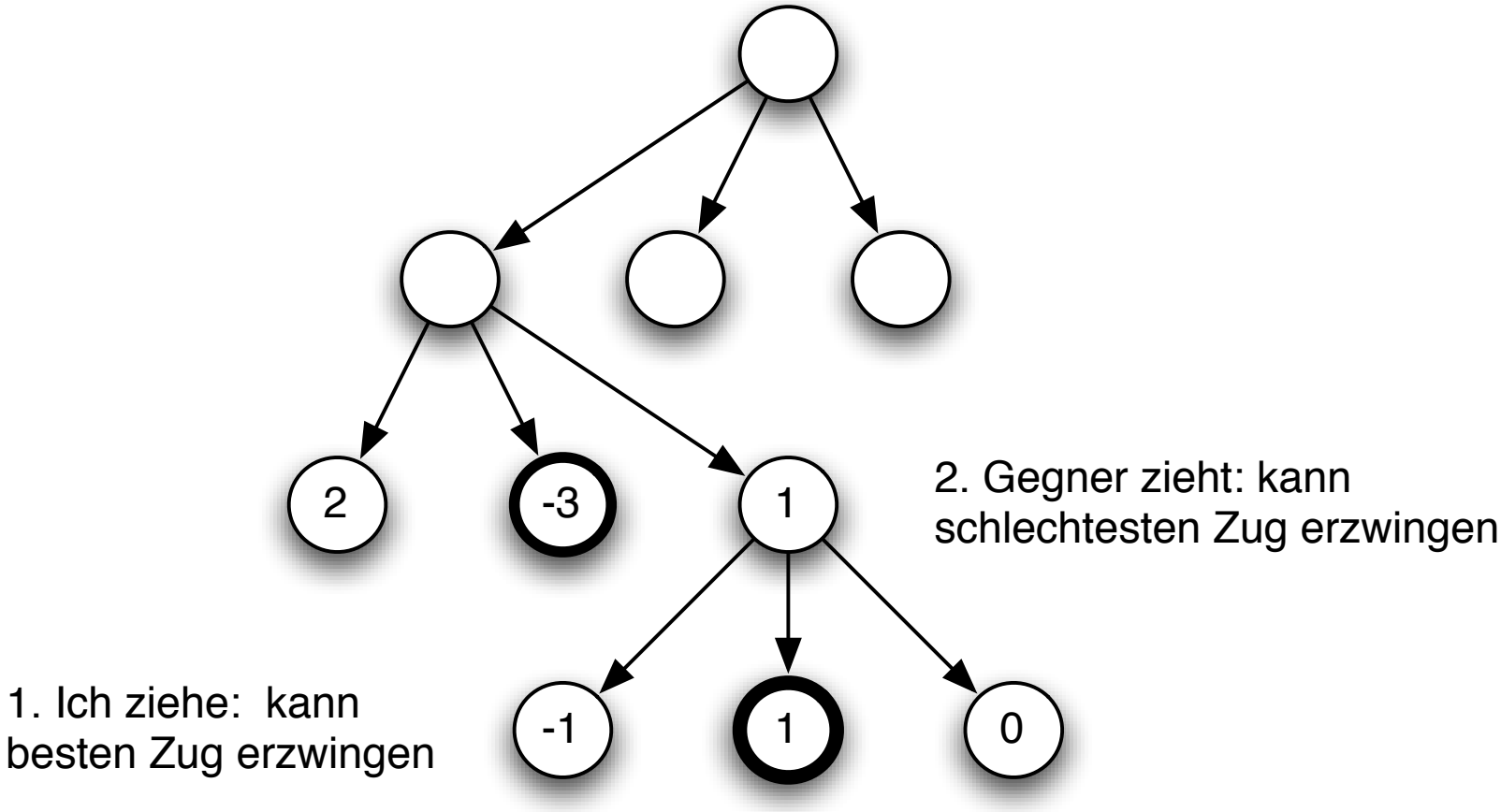


Durchlauf-Reihenfolge

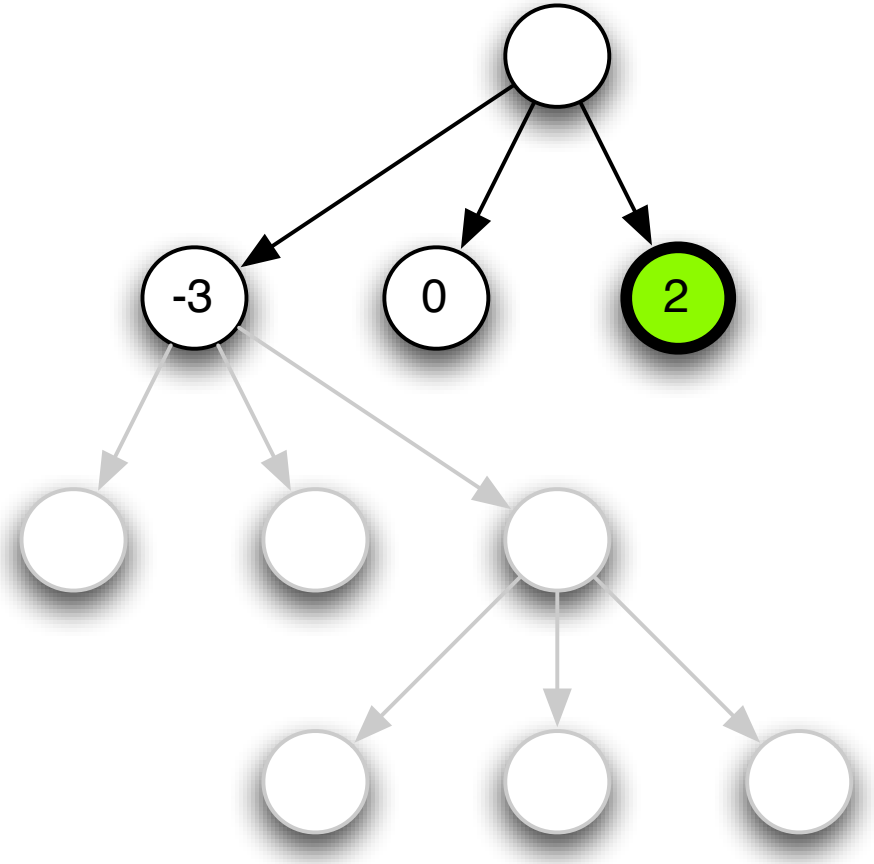


Tiefensuche (Kinder kommen vor den Eltern)

Blätter bewerten, Werte propagieren



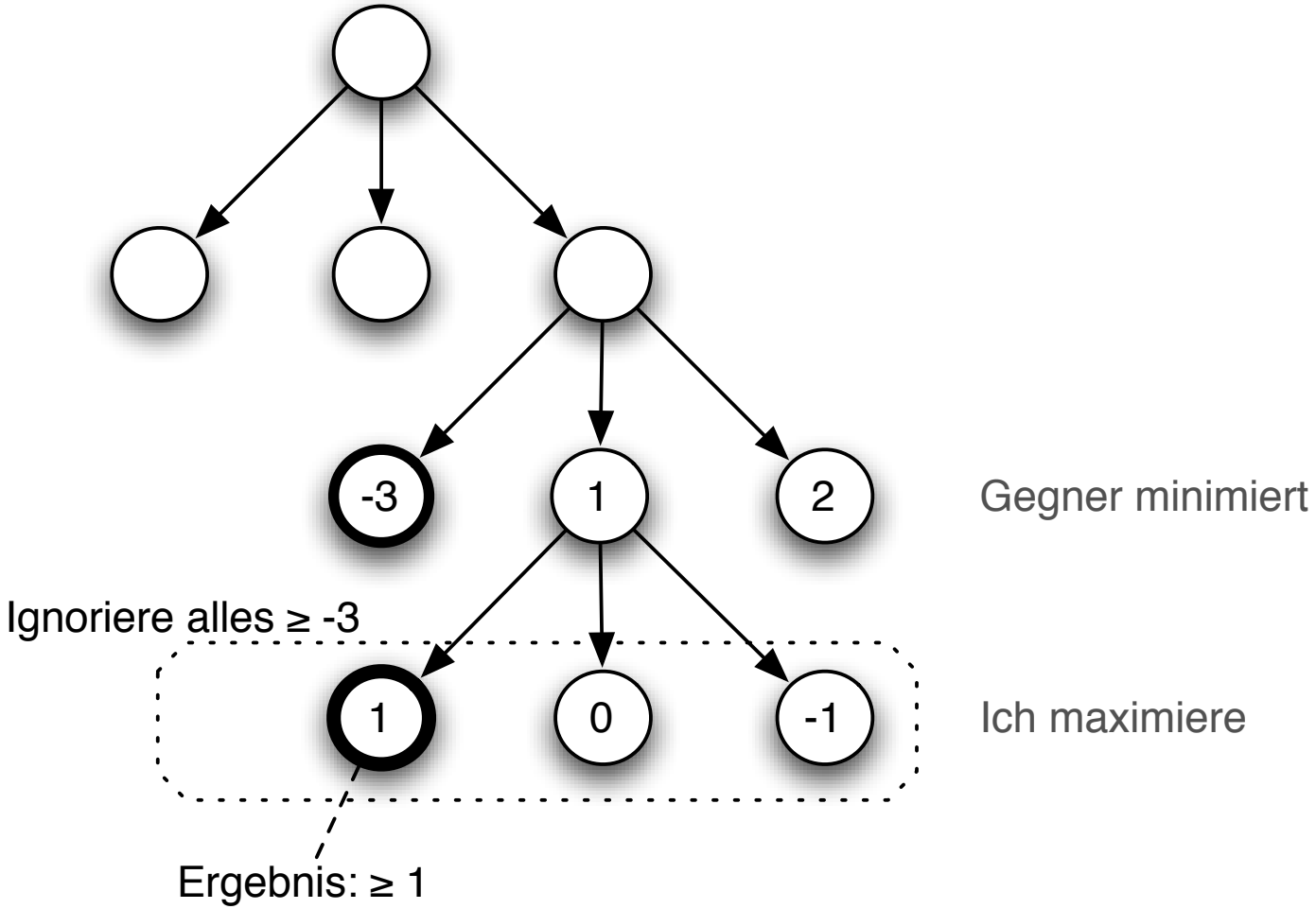
Ergebnis: bewertete nächste Züge



Minimax: Pseudocode

```
int minimax(boolean max, Board board) {
    if (isLeaf(board)) {
        return rate(board);
    }
    int best = (max ? Integer.MIN_VALUE : Integer.MAX_VALUE);
    for(Move move : board.computeMoves()) {
        board.makeMove(move);
        int child = minimax(! max, board);
        if (isBetter(max, child, best)) {
            best = child;
        }
        board.undoMove(move);
    }
    return best;
}
```

Alpha-Beta Pruning



Alpha-Beta Pruning

- Motivation: Manchmal bringt es nichts, noch weiter in den Baum hinabzusteigen.
- Vorbereitung: Züge absteigend ordnen (viel versprechendster Zug zuerst).
- Vater gibt einen Schwellenwert mit. Die Suche wird (im aktuellen Teilbaum) abgebrochen, wenn dieser Wert erreicht wird.
  - Grund: Je besser das Kind wird, desto uninteressanter ist das Ergebnis für den Vater, da er in der entgegengesetzen Richtung optimiert und schon ein bisheriges Optimum hat.

Kritisch: Kann man Züge sinnvoll ordnen?

GUI

Themen

- Komponenten schachteln, Layout, Events: siehe Zentralübung von Info II.
- Custom Painting
- MVC-Pattern
- Multi-threaded Swing

Custom Painting

Aufbau des TicTacToe-Bretts:

- BoardView:
  - zeichnet das Gitter als Hintergrund.
  - enthält CellViews, die mit einem GridLayout positioniert werden.
- CellView: zeigt (so vorhanden) einen Stein in der richtigen Farbe an.

Weitere Information: http://java.sun.com/docs/books/tutorial/uiswing/painting/

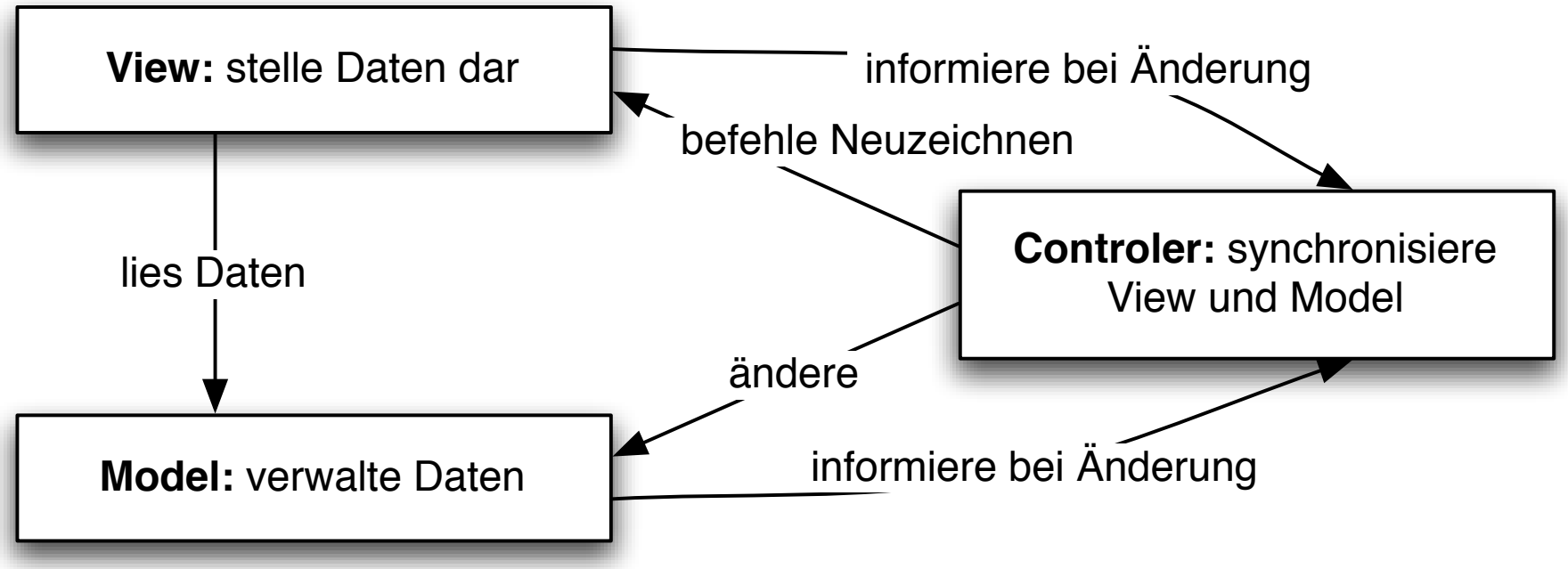
Custom Painting: CustomComponent

```
protected void paintComponent(Graphics g) {
    if (isOpaque()) { // paint background
        g.setColor(getBackground());
        g.fillRect(0, 0, getWidth(), getHeight());
    }
    Graphics2D g2d = (Graphics2D)g.create();

    Ellipse2D ellipse = new Ellipse2D.Double(
            getLocation().x+INSET, getLocation().y+INSET,
            getWidth() - (2*INSET), getHeight() - (2*INSET));
    g2d.draw(ellipse);

    g2d.dispose(); // clean up
}
```

MVC-Pattern


View: stelle Daten dar
informiere bei Änderung
befehle Neuzeichnen
Controler: synchronisiere View und Model
lies Daten
ändere
Model: verwalte Daten
informiere bei Änderung

MVC-Pattern

- Motivation: Wie strukturiert man Daten, Widgets und Verhalten am besten in einer grafischen Benutzeroberfläche.
- Variation: Nur Model und Controler sind selbst programmiert, Controler kopiert zwischen Model und View.
- Variation: View und Controler in einer Klasse vereinigt. Wird in Swing so gemacht.
- Verallgemeinerte Modelle: JavaBeans
- Verallgemeinerte Controler: Databinding.

MVC-Pattern: NameModel

```
public String getFirst() {
    return first;
}
public void setFirst(String newFirst) {
    if (newFirst == null) {
        throw new NullPointerException();
    }
    if (! (this.first.equals(newFirst))) {
        this.first = newFirst;
        fireHadChange();
    }
}
```

MVC-Pattern: NameModel

```
private Set<ChangeListener> changeListeners = new HashSet<ChangeListener>()
public void addChangeListener(ChangeListener listener) {
    this.changeListeners.add(listener);
}
public void removeChangeListener(ChangeListener listener) {
    this.changeListeners.remove(listener);
}
private void fireHadChange() {
    for(ChangeListener listener : this.changeListeners) {
        listener.hadChange();
    }
}
```

MVC-Pattern: NameView

```
public NameView() {
    this.mainPanel = new JPanel();
    this.mainPanel.setLayout(new GridLayout(3, 2));

    this.firstField = createField(mainPanel, "First: ");
    this.lastField = createField(mainPanel, "Last: ");

    this.button = new JButton("Print Model");
    this.mainPanel.add(this.button);
}
private JTextField createField(JPanel parentPanel, String labelString) {
    parentPanel.add(new JLabel(labelString));
    JTextField textField = new JTextField();
    parentPanel.add(textField);
    return textField;
}
```

MVC-Pattern: NameControler

```
public NameControler(NameModel myNameModel, NameView myNameView) {
    // [...]
    nameModel.addChangeListener(this);
    nameView.firstField.getDocument().addDocumentListener(this);
    nameView.lastField.getDocument().addDocumentListener(this);
    nameView.button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            System.out.println("Current model: "+nameModel);
        }
    });
}
```

MVC-Pattern: NameControler

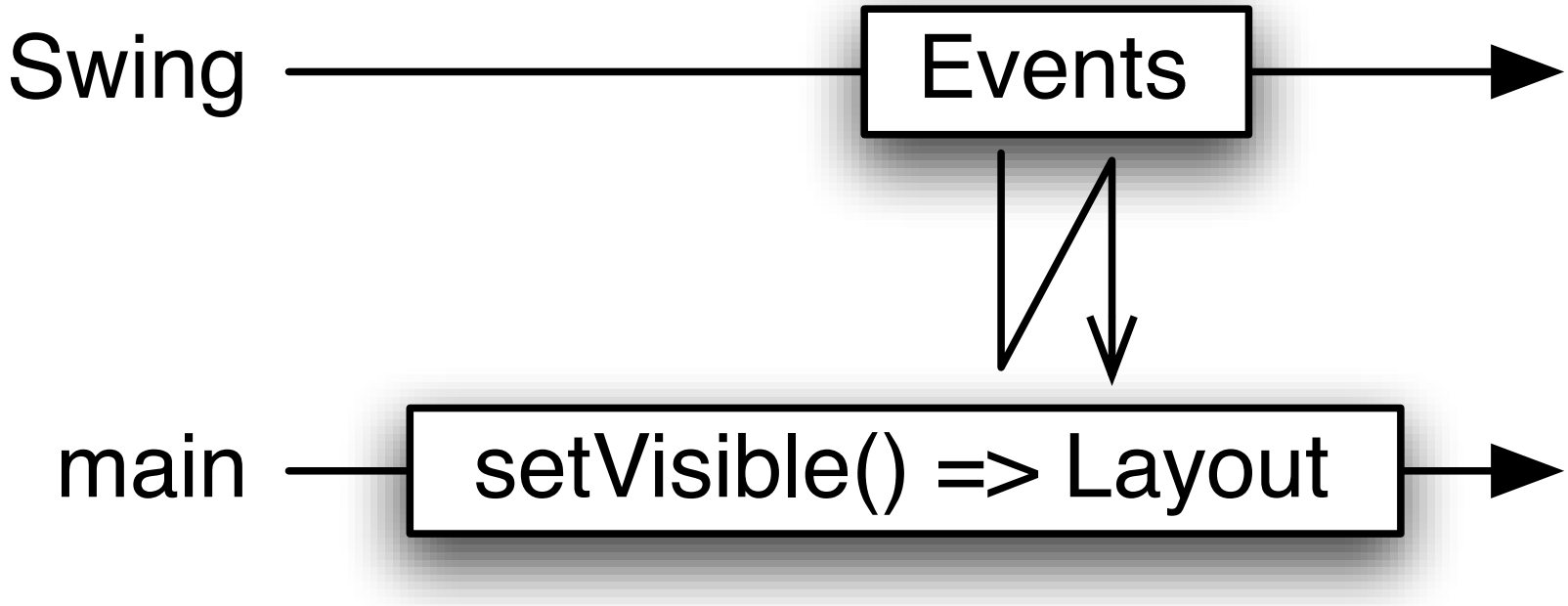
```
public void insertUpdate(DocumentEvent e) {
    viewToModel();
}
protected void viewToModel() {
    // Pause listening: prevent recursion
    nameModel.removeChangeListener(this);
    nameModel.setFirst(nameView.firstField.getText());
    nameModel.setLast(nameView.lastField.getText());
    nameModel.addChangeListener(this);
}
public void hadChange() { // model to view
    nameView.firstField.setText(nameModel.getFirst());
    nameView.lastField.setText(nameModel.getLast());
}
```

Multi-Threaded Swing

```
public static void main0(String[] args) {
    JFrame frame = new JFrame("Title");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JButton button = new JButton("Hi!");
    frame.add(button, BorderLayout.CENTER);
    frame.setSize(200, 100);
    frame.setVisible(true);
}
```

Warum ist das so nicht OK?

Illegaler gleichzeitiger Zugriff



Multi-Threaded Swing: invokeLater()

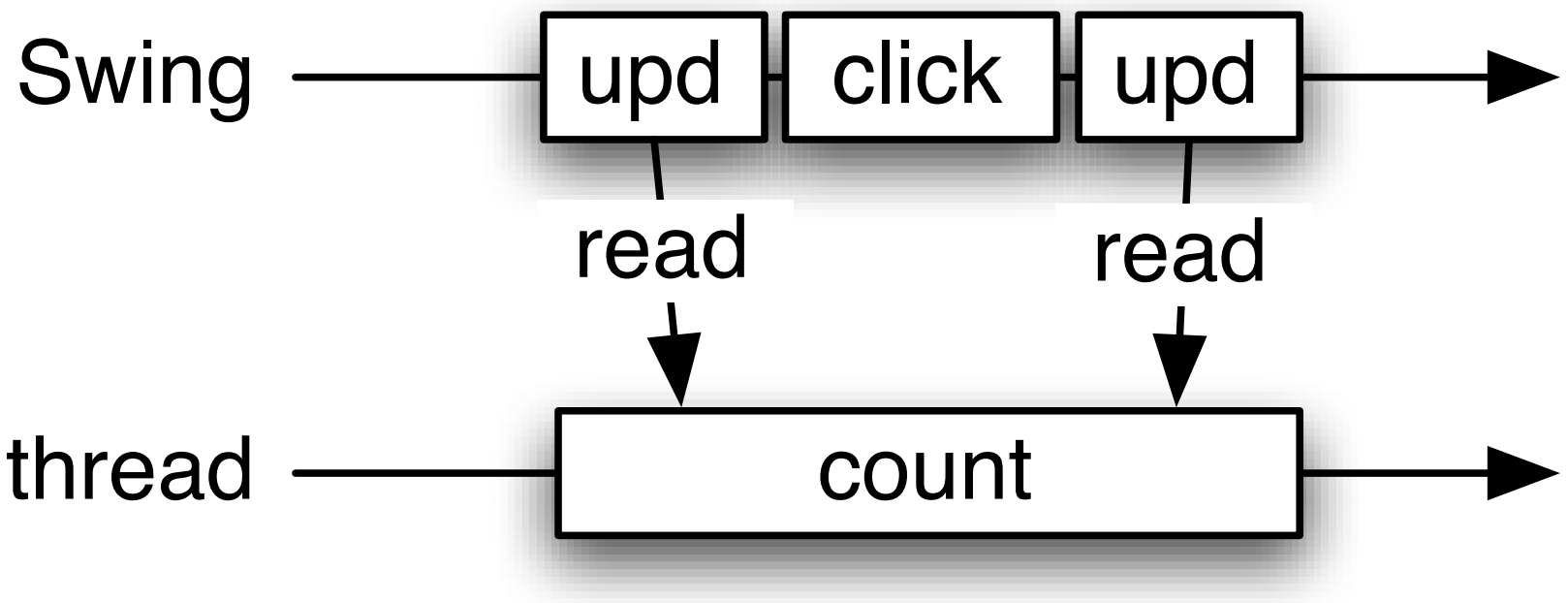
```
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            final JFrame frame = new JFrame("Title");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOS
            JButton button = new JButton("Hi!");
            frame.add(button, BorderLayout.CENTER);
            frame.setSize(200, 100);
            frame.setVisible(true);
        }
    });
}
```

Swing und lang dauernde Operationen

- `TimerExample1`

Swing: Blockieren der Oberfläche

Swing: Einsatz eines Timers



Swing: Einsatz eines Timers

- `TimerExample2`

Aufgabe

User Stories: Automatischer Spieler

- Es gibt einen automatischen Spieler, der mit dem bisherigen Client und Server zusammenarbeitet. [Technisch: Minimax-Suche mit Alpha-Beta Pruning.]

- Der Timeout wird (vorerst) noch nicht vom Server erzwungen. [Man kann also die Spielbäume zu Ende rechnen.]

- Es soll eine Offline-Version des automatischen Spielers geben, mit der man *grafisch* gegen den automatischen Spieler antreten kann.

- Wer anfängt (Mensch oder Rechner) soll man entweder grafisch oder über die Kommandozeile bestimmen können.

User Stories: Manueller Spieler

- Der manuelle Spieler (Client) soll eine grafische Benutzeroberfläche bekommen.
- Im manuellen Spieler keine illegalen Züge zulassen.

Abgabe ist in 2 Wochen.

Tipps

- Klasse SwingWorker: Übernimmt verschiedene Standardaufgaben im Bereich des Swing-Multithreading. Bei Java 6.0 standardmäßig dabei.
  http://java.sun.com/docs/books/tutorial/uiswing/concurrency/worker.html

- Glazed Lists: Vereinfachtes Erstellen von Tabellen und Listen aller Art.
  http://publicobject.com/glazedlists/

- Code Reuse: Was kann man in gemeinsame Oberklassen auslagern?
  - Beispiel: Eine abstrakte Oberklasse für den PlayerInterface-Skeleton.