Martin Häcker

Die drei ???

120

Die Idee zum Programm

4!

Freitagsrunde

```
private void solveAnyProblem() {
    while (isStillUnsolved()) {
        analyzeProblem();
        chooseSuitablePart();
        implementPart();
    }
}
```

„As simple as possible, but no simpler“

-- Albert Einstein

Zuerst: Information Hiding

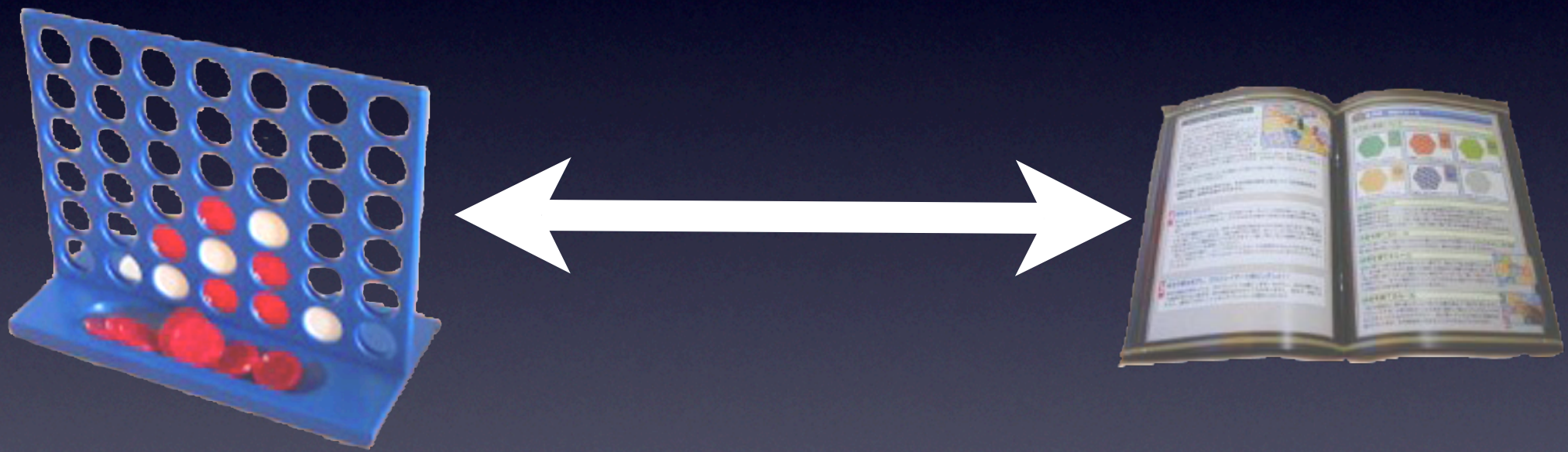
- Lösungen (meist) Module
- Was, nicht wie
- Wie „privat“

`private`

`public`

Wie: Teile?

Zwei Arten Objekte



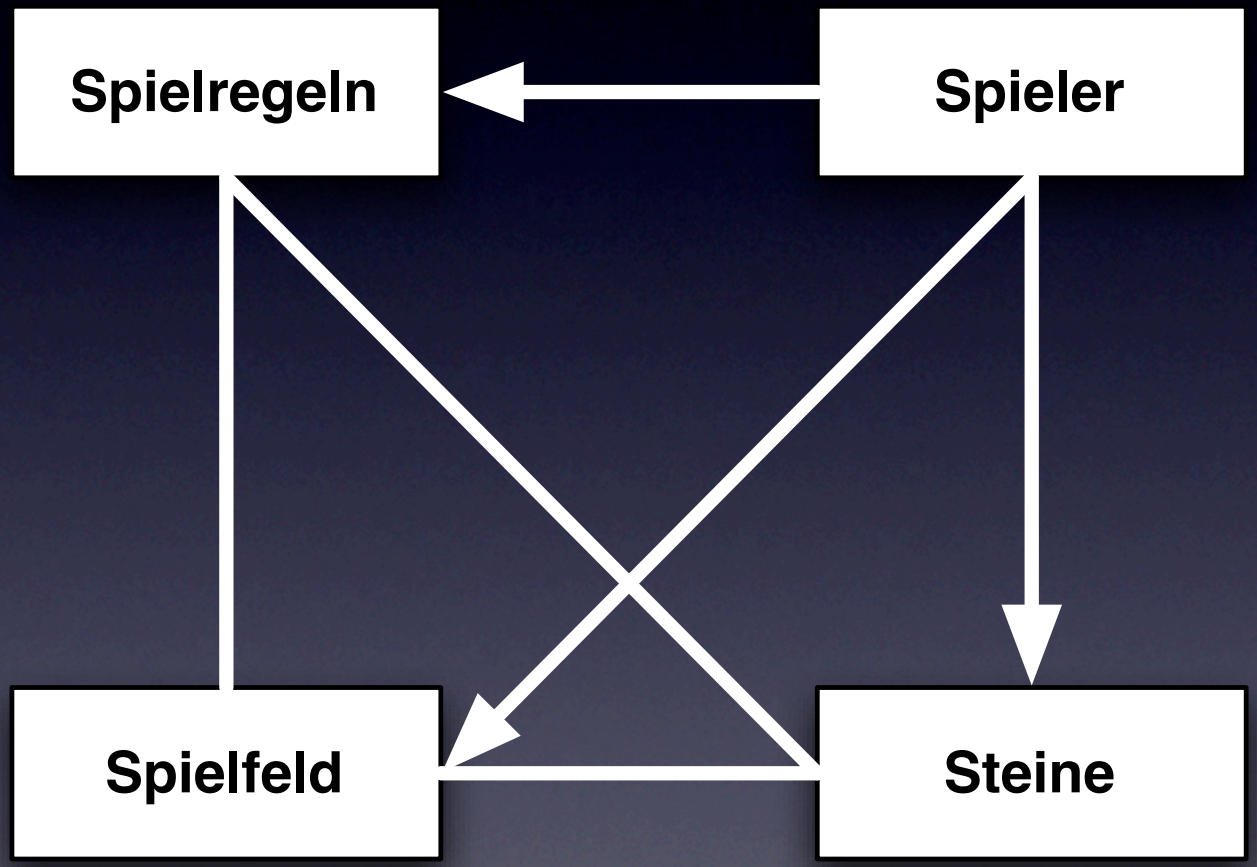
Jucheeh!

Bullshit!

„Debugging is twice as hard
as writing the code in the first place.
Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.“

-- Brian W. Kernighan

Beispiel



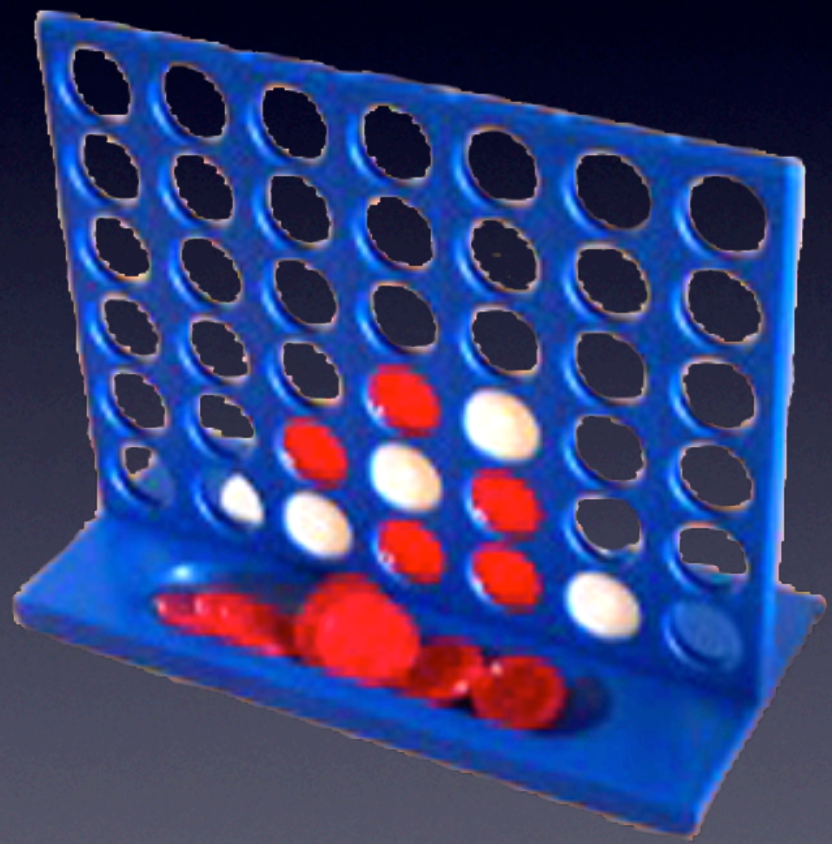
-regeln

- Benutzer-schnittstelle
- Rest

Unabhängig, Zentral?

Umsetzung: Spielfeld

- Was muss ein Spielfeld können?
- Was ist das einfachste davon?



Code

- Spielfeld leer, wenn nichts passiert ist

- Spielfeld leer, wenn nichts passiert ist

```
// File BoardTest.java
class BoardTest extends TestCase {
    public void testEmptyBoard() {

        assertTrue(board.isEmpty());
    }
}
```

- Spielfeld leer, wenn nichts passiert ist

```
// File BoardTest.java
class BoardTest extends TestCase {
    public void testEmptyBoard() {
        Board board = new Board();
        assertTrue(board.isEmpty());
    }
}
```

- Spielfeld leer, wenn nichts passiert ist

```
// File BoardTest.java
class BoardTest extends TestCase {
    public void testEmptyBoard() {
        Board board = new Board();
        assertTrue(board.isEmpty());
    }
}
// File Board.java
class Board {
    public boolean isEmpty() {
        return true;
    }
}
```

- ~~Spielfeld leer, wenn nichts passiert ist~~

```
// File BoardTest.java
class BoardTest extends TestCase {
    public void testEmptyBoard() {
        Board board = new Board();
        assertTrue(board.isEmpty());
    }
}
// File Board.java
class Board {
    public boolean isEmpty() {
        return true;
    }
}
```

- isEmpty() == false, wenn schon Steine gelegt

- isEmpty() == false, wenn schon Steine gelegt

```
public void testBoardWithOneStone()
{

    assertFalse(board.isEmpty());
}
```

- isEmpty() == false, wenn schon Steine gelegt

```
public void testBoardWithOneStone()
{
    board.placeWhiteStoneInColumn(5);
    assertFalse(board.isEmpty());
}
```

- isEmpty() == false, wenn schon Steine gelegt
- Möglichkeit Steine zu legen

```
public void testBoardWithOneStone()
{
    board.placeWhiteStoneInColumn(5);
    assertFalse(board.isEmpty());
}
```

- isEmpty() == false, wenn schon Steine gelegt
- Möglichkeit Steine zu legen

```
public void testBoardWithOneStone()
{
    Board board = new Board();
    board.placeWhiteStoneInColumn(5);
    assertFalse(board.isEmpty());
}
```

- Möglichkeit Steine zu legen
- isEmpty() == false, wenn schon Steine gelegt

- Möglichkeit Steine zu legen
- isEmpty() == false, wenn schon Steine gelegt

```
public void testWhiteStonePlacement() {

    assertTrue(board.hasWhiteStoneAtPosition(5, 0));

}
```

- Möglichkeit Steine zu legen
- isEmpty() == false, wenn schon Steine gelegt

```
public void testWhiteStonePlacement() {
    Board board = new Board();
    board.placeWhiteStoneInColumn(5);
    assertTrue(board.hasWhiteStoneAtPosition(5, 0));

}
```

- Möglichkeit Steine zu legen
- Stein legen, lässt Rest frei
- isEmpty() == false, wenn schon Steine gelegt

```
public void testWhiteStonePlacement() {
    Board board = new Board();
    board.placeWhiteStoneInColumn(5);
    assertTrue(board.hasWhiteStoneAtPosition(5, 0));

}
```

- Möglichkeit Steine zu legen
- Stein legen, lässt Rest frei
- isEmpty() == false, wenn schon Steine gelegt

```
public void testWhiteStonePlacement() {
    Board board = new Board();
    board.placeWhiteStoneInColumn(5);
    assertTrue(board.hasWhiteStoneAtPosition(5, 0));
    assertFalse(board.hasWhiteStoneAtPosition(5, 1));
}
```

- ~~Möglichkeit Steine zu legen~~
- ~~Stein legen, lässt Rest frei~~
- isEmpty() == false, wenn schon Steine gelegt

```
public void testWhiteStonePlacement() {
    Board board = new Board();
    board.placeWhiteStoneInColumn(5);
    assertTrue(board.hasWhiteStoneAtPosition(5, 0));
    assertFalse(board.hasWhiteStoneAtPosition(5, 1));
}
```

```
class Board {
    private int [][] board;
    final static int BOARD_ROWS = 6;
    final static int BOARD_COLUMNS = 7;

    final static int NO_STONE = 0;
    final static int WHITE_STONE = 1;
    final static int RED_STONE = 2;

    public Board() {
        board = new int [BOARD_ROWS][BOARD_COLUMNS];
        // new sets all array-members to 0,
        // since NO_STONE has a value of 0
        // the array is already correctly initialized
    }
}
```

- ~~Spielfeld leer, wenn nichts passiert ist~~
- ~~Möglichkeit Steine zu legen~~
- ~~Stein legen, lässt Rest frei~~
- ~~isEmpty() == false, wenn schon Steine gelegt~~
- Rote Steine

Fertig

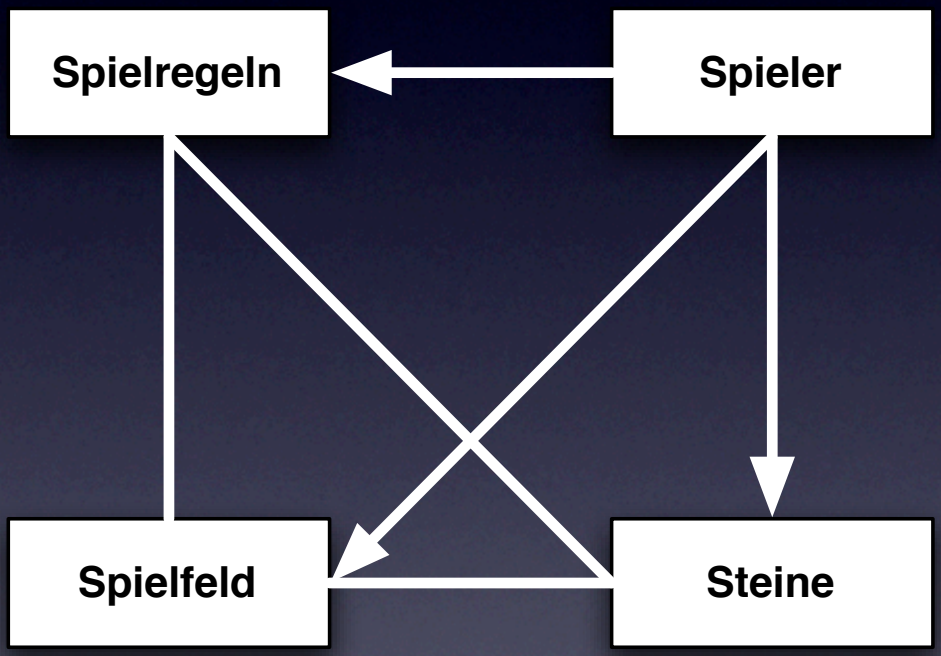
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0

- Spielfeld anzeigen
- Abwechselnd Spieler nach Reihe fragen

Nun ja...

- Stein nur einwerfen wenn Platz
- Ist das Spiel vorbei (vier in einer Reihe)
  - vertikal
  - horizontal
  - diagonal

Erstes vs. Endgültiges Design



Ausblick KI

- Aus möglichen Zügen auswählen
  - Lässt Zug mich gewinnen?
  - Lässt Zug Gegner gewinnen?

```
private void solveAnyProblem() {
    while (isStillUnsolved()) {
        analyzeProblem();
        chooseSuitablePart();
        implementPart();
    }
}
```

Nachmittagsaufgabe