Prof. Dr. Dominikus Herzberg, FB06 (MNI)


# “Programmierung interaktiver Systeme (PiS)“ 80%-Klausur, Sommersemester 2017

| **Vor- und Nachname** | **Matrikelnummer** | Note |
|---|---|---|
| | | |

**Prüfungsdauer:** 90 Minuten

**Wertung:** Insgesamt können 80 Punkte erreicht werden. Die pro Aufgabe erzielbare Punktzahl ist angegeben.

**Hilfsmittel:** keine

**Hinweise:**

- Bitte verwenden Sie keinen Rotstift oder Bleistift.
- Tragen Sie in das obige Feld Ihren Namen und Ihre Matrikelnummer ein. Unterschreiben Sie das Deckblatt unten.
- Schreiben Sie Ihre Antworten bitte in die dafür vorgesehenen Abschnitte. Bei Platzmangel können Sie das beigefügte Extrablatt nutzen. Nur ausdrücklich als Lösungen ausgewiesene Bereiche auf dem Extrablatt können gewertet werden.
- Geben Sie dieses Deckblatt zusammen mit allen Aufgabenblättern ab.
- Überprüfen Sie die Klausur auf Vollständigkeit. Bitte beanstanden Sie Mängelexemplare umgehend.
- Es werden nur leserliche Klausuren bewertet!
- Programmcode hat sich in Art und Form an den in der Veranstaltung verwendeten Konventionen zu orientieren. Schreiben Sie syntaktisch korrekten Code.

| **Datum** | **Unterschrift** |
|---|---|
| 29. Sep. 2017 | |

# Multiple-Choice-Fragen (24 Punkte)

Für jede der folgenden Fragen sind 2 Punkte erreichbar. Bei den Auswahlfragen (*multiple choice*) können mehrere Antworten richtig sein. Nur vollständig korrekte Antworten werden mit 2 Punkten bewertet. Lassen Sie sich bitte von ungünstig gesetzten Seitenumbrüchen nicht irritieren.

## 1. Frage

Was stimmt?

- ☐ Ausdruck und Anweisung sind ein- und dasselbe
- ☐ Ein Ausdruck ist eine Berechnung, die einen Wert liefert
- ☐ Eine Anweisung ist ein Ausdruck, der im Prinzip `void` ist
- ☐ Ein Lambda-Ausdruck ist streng genommen eine Anweisung

## 2. Frage

Mit `Boolean[] board = new Boolean[9];` deklariert ein Student das Spielfeld für Tic-Tac-Toe. Was meinen Sie dazu?

- ☐ Clever, er will Bitboards nutzen!
- ☐ Moment, es muss `boolean[] board = new boolean[9]` heißen!
- ☐ Clever, tolle Idee, denn `null` kann man für leere Felder nutzen
- ☐ Keine gute Idee, `null` als dritter Wert verletzt Zweck von `Boolean`

## 3. Frage

Mit `boolean move(int pos)` deklariert eine Studentin die Methode im Zuggenerator für Tic-Tac-Toe. Der Rückgabewert soll anzeigen, ob der mit der Position übergebene Zug gültig ist oder nicht. Was meinen Sie dazu?

- ☐ Besser wäre es, eine `IllegalArgumentException` zu werfen!
- ☐ Überflüssig, der Zuggenerator wird nur mit gültigen Zügen aufgerufen
- ☐ Da ein Zug das Spielbrett nicht verändert, muss es `void` heißen
- ☐ Die Rückgabe muss `int` sein, weil die Methode einen Zug zurückgibt

## 4. Frage

Der Code `if (isWin()) return 1; else return 0;` ist kritikwürdig. Oder?

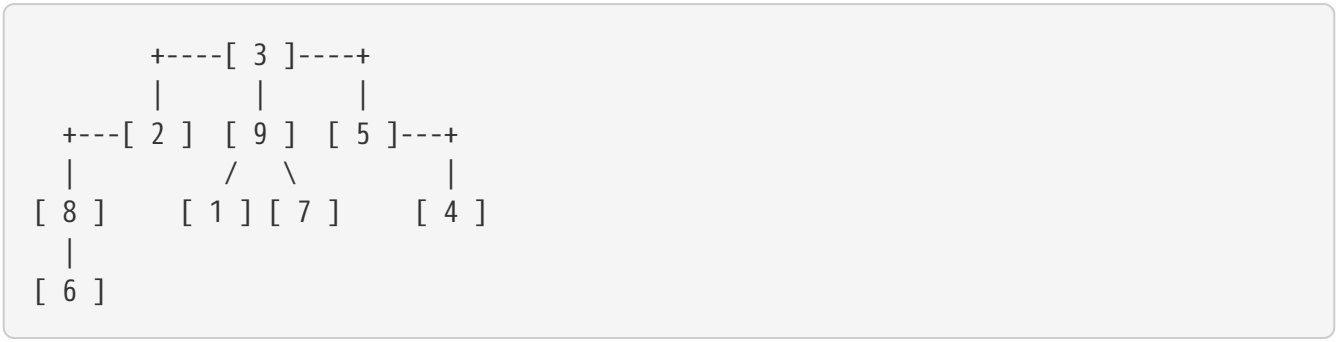
- ☐ Es fehlen die geschweiften Klammern, sonst läuft er nicht!
- ☐ Richtig ist `if (isWin()) return 0; else return 1;`
- ☐ Kürzer: `if (isWin()) return 1; return 0;`

- ☐ Kürzer: `return isWin() ? 0 : 1;`

# 5. Frage

Das Bild beschreibe einen Spielbaum, dessen Knoten willkürlich mit den Zahlen von 1 bis 9 gekennzeichnet sind. Welche Folge des Besuchs von Knoten beschreibt eine gültige Tiefen- oder Breitensuche?

*Spielbaum mit den Knoten von 1 bis 9*




- ☐ Breitensuche: 6-8-1-7-4-2-9-5-3
- ☐ Tiefensuche: 3-2-8-6-9-1-7-5-4
- ☐ Breitensuche: 8-6-2-3-9-1-7-5-4
- ☐ Tiefensuche: 6-8-2-3-1-9-7-4-5

# 6. Frage

Bei Streams unterscheidet man *intermediäre* und *terminale* Operationen. Welche der genannten Operationen sind intermediär?

- ☐ limit
- ☐ filter
- ☐ count
- ☐ forEach

# 7. Frage

Kreuze an, was korrekt ist.

- ☐ Ein Lambda-Ausdruck evaluiert zu einer funktionalen Methode
- ☐ Ein Lambda-Ausdruck evaluiert zu einer Instanz eines funktionalen Interfaces
- ☐ Ein funktionales Interface hat nur genau eine Methode
- ☐ Ein funktionales Interface kann mehrere abstrakte Methoden haben

# 8. Frage

Kreuze an, was ein syntaktisch korrektes Konstrukt für einen Lambda-Ausdruck ist.

- ☐ `n -> n + 1`
- ☐ `(n) -> n + 1`
- ☐ `Integer n -> n + 1`
- ☐ `(Integer n) -> n + 1`

# 9. Frage

Kreuze an, was ein syntaktisch korrektes Konstrukt für einen Lambda-Ausdruck ist.

- ☐ `x, y -> x + y`
- ☐ `() -> n + 1`
- ☐ `-> return n + 1`
- ☐ `(x, y) -> x + y`

# 10. Frage

Kreuze an, was korrekt ist.

- ☐ Eine Methodenreferenz evaluiert zu einer Methode
- ☐ Eine Methodenreferenz evaluiert zu einem Funktionsobjekt
- ☐ Eine Methodenreferenz referenziert eine Methoden-Implementierung
- ☐ Eine Methodenreferenz referenziert eine Interface-Methode

# 11. Frage

Falls Typargumente in einer Methodenreferenz angegeben werden müssen bzw. sollen, dann

- ☐ vor dem Identifier bzw. vor `new`
- ☐ nach dem Identifier bzw. nach `new`
- ☐ vor den Doppelpunkten `::`
- ☐ nach den Doppelpunkten `::`

# 12. Frage

Nach der folgenden Eingabe ist welche Fortsetzung in der JShell korrekt?

```
jshell> (Predicate<String>) String::isEmpty
$45 ==> $Lambda$66/508198356@4f51b3e0
```

- ☐ `$45.apply("Hi")`

- [ ] `$45.predicate("Hi")`
- [ ] `$45.test("Hi")`
- [ ] `$45.query("Hi")`

# Kleinere Programmieraufgaben (32 Punkte)

Generell gilt für diese Aufgaben, dass die von den Sprachkonstrukten her kompaktesten Java-Lösungen gesucht sind.

## Beverage (4 Punkte)

Implementieren Sie eine Klasse namens `Beverage` (*Getränk*), die die Eigenschaften `name`, `price` und `size` hat und sich auf die folgende Art instanziieren und konfigurieren lässt:

```
Beverage cola = new Beverage().setName("Cola").setPrice(1.30).setSize(0.33);
```

*Lösung*

## `for` in einen Stream wandeln (4 Punkte)

Ersetzen Sie die `for`-Anweisung durch einen Stream, der auf die gleiche Weise die Liste der `moves` erstellt. Sie wissen von `board` nur, dass es vom Typ `int[]` ist.

```
ArrayList<Integer> moves = new ArrayList<>();
for(int i=0; i<board.length; i++) {
    if (board[i] == 0) moves.add(i);
}
```

*Lösung*

# `makeMove` mit Varargs (4 Punkte)

Gegeben sei die Klasse `Board`, die u.a. die überladene `makeMove`-Methode enthält. Implementieren Sie die gegebene Methode, die nichts anderes macht, als für alle `positions` nacheinander die Methode `void makeMove(int pos)` aufzurufen. Machen Sie das mit Hilfe eines Streams.

*Implementieren Sie den Rumpf als Einzeiler mit einem Stream*

```
void makeMove(int... positions) {


}
```

# Minimum ermitteln (4 Punkte)

Vervollständigen Sie den gegebenen Ausdruck in der JShell so, dass der kleinste Wert in der Liste zurückgegeben wird; im Beispiel wäre das der Wert `2`.

*Vervollständigen Sie den Ausdruck*

```
jshell> List.of(5,3,9,4,2,3,8).
$1 ==> 2
```

# Wert ermitteln (4 Punkte)

Gegeben sei der folgende Stream. Wie lautet der 10. Wert, den der Stream produzieren würde?

```
Stream.iterate(new int[]{1,2,3}, a -> new int[]{a[0]+a[1],a[2],a[1]}).mapToInt(a ->
a[0])
```

*Lösung*

Der 10. Wert lautet:

# Funktionales Interface erstellen (4 Punkte)

Erstellen Sie ein geeignetes funktionales Interface namens `S2I`, so dass folgender Dialog in der JShell möglich ist:

```
jshell> ((S2I) String::length).apply("Hello")
$13 ==> 5
```

*Lösung*

## Primzahlen (4 Punkte)

1. Geben Sie ein `IntPredicate isPrime` an, das prüft, ob eine positive Ganzzahl eine Primzahl ist; im Code darf nicht mehr als ein Semikolon vorkommen
2. Erstellen Sie unter Rückgriff auf `isPrime` eine Methode `IntStream primes()`, die einen (von der Idee her) unendlich langen Strom an Primzahlen erzeugt; wieder ist nur ein Semikolon erlaubt

*Lösung*

```
IntPredicate isPrime =

IntStream primes()
```

## Optionales (4 Punkte)

Die Variable `nOpt` sei vom Typ `Optional<Integer>`. Merkwürdigerweise führt der folgende Aufruf zu einem Problem. Was für ein Fehler wurde gemacht?

```
jshell> nOpt.isPresent()
|  java.lang.NullPointerException thrown:
|        at (#45:1)
```

*Lösung*

# Programmierung des Nim-Spiels (24 Punkte)

Das Spiel namens "Nim" unterscheidet vier Reihen in denen zu Beginn 1, 3, 5 und 7 Streichhözer liegen. Die Spieler ziehen abwechselnd. Pro Zug muss ein Spieler ein oder mehr Streichhölzer aus einer der Reihen entfernen. Es ist nicht erlaubt, in einem Zug aus mehr als einer Reihe Streichhölzer zu entfernen. Verloren hat der Spieler, der das letzte Streichholz nehmen muss — das ist die sogenannte Misère-Variante des Nim-Spiels.

*Ausgangslage beim Nim-Spiel*

```
1. Reihe        |
2. Reihe      | | |
3. Reihe    | | | | |
4. Reihe  | | | | | | |
```

Sie sollen das Spiel gemäß des gegebenen Interfaces implementieren.

```
public interface MutableBoard<Move> {
    void makeMove(Move move);
    void undoMove();
    List<Move> moves();
    List<Move> getHistory(); // last move in list = recent move
    boolean isLost();
    default boolean isBeginnersTurn() {
        return getHistory().size() % 2 == 0;
    }
    String toString();
}
```

Ihr Programm muss die beispielhafte Interaktion exakt reproduzieren können. Da Nim auch mit mehr Reihen und anderen Zahlen pro Reihe gespielt wird, sollte der Code so geschrieben sein, dass die Dimensionierung des Nim-Spielfeldes nicht fix ist.

```
jshell> Board b = new Board(1,3,5,7)
b ==> 1 3 5 7

jshell> b.makeMove(new NimMove(2,3))

jshell> b
b ==> 1 3 2 7

jshell> b.moves()
$7 ==> [0:1, 1:1, 1:2, 1:3, 2:1, 2:2, 3:1, 3:2, 3:3, 3:4, 3:5, 3:6, 3:7]

jshell> b.undoMove()

jshell> b
b ==> 1 3 5 7

jshell> b.moves()
$10 ==> [0:1, 1:1, 1:2, 1:3, 2:1, 2:2, 2:3, 2:4, 2:5, 3:1, 3:2, 3:3, 3:4, 3:5, 3:6,
3:7]
```

Die Implementierung muss kompakt programmiert sein. Die Anzahl der in der Musterlösung verwendeten Semikolons ist jeweils angegeben und ist einzuhalten.

# Implementiere Klasse `NimMove` (5 Punkte)

Implementieren Sie die Klasse `NimMove`, die nur einen Konstruktor und eine `toString`-Methode hat. Ein Zug ist definiert über die Reihe (*row*) und die Anzahl (*number*) der zu entfernenden Streichhölzer. Die Musterlösung kommt mit vier Semikolons (`;`) im Code aus.

- Zählen Sie die Reihen mit `0` beginnend, so wie typisch für Informatiker/innen
- Machen Sie die Felder der Klasse lesbar aber nicht veränderbar
- Es gibt keine Setter- oder Getter-Methoden

*Lösung (4 Semikolons)*

# Deklariere Klassenkopf, Felder, Konstruktor (5 Punkte)

Bevor wir uns den einzelnen Methoden widmen, deklarieren Sie den Kopf der Klasse `Board`, die benötigten Felder und den Konstruktor. Die interne Darstellung des Spielbretts soll durch ein "normales" Array aus `int`-Werten erfolgen, jeder `int`-Wert gibt die Anzahl der verbliebenen Streichhölzer für die Reihe an. Die Musterlösung kommt mit drei Semikolons aus.

*Lösung (3 Semikolons)*

# Methode `makeMove` (3 Punkte)

*Lösung (Implementieren Sie die Methode, 2 Semikolons)*

## Methode `undoMove` (3 Punkte)

*Lösung (Implementieren Sie die Methode, 2 Semikolons)*

## Methode `moves` (6 Punkte)

*Lösung (Implementieren Sie die Methode, 3 abschließende Semikolons)*

## Methode `isLost` (2 Punkte)

*Lösung (Implementieren Sie die Methode, 1 Semikolon)*

(Zusatzblatt)