# VU Grundlagen der Programmkonstruktion — Übungsteil

## Organisatorisches

### Wie sollen Ihre Lösungen aussehen

Beachten Sie, dass nicht die konkrete Lösung im Vordergrund steht, sondern der Weg, wie Sie zu dieser gelangen. Geben Sie daher bei Aufgaben immer auch den Rechenweg an. Bei jeder Aufgabe ist ein Beispiel angeführt, wie dieser aussehen soll. Ohne Angabe des Rechenwegs erhalten Sie für die Aufgabe keine Punkte.

Geben Sie ihre Lösungen im TUWEL als (bevorzugt) Text-Dokument (Erweiterung `.txt`) oder als PDF-Dokument (Erweiterung `.pdf`) ab.

### Gruppenarbeiten

Sie können die Aufgaben in Gruppen zu 2-6 Personen lösen. Die Zusammenstellung und Organisation der Gruppen ist dabei Ihnen überlassen.

Wenn Sie die Aufgabe in einer Gruppe gelöst haben, so soll EIN Gruppenmitglied die Aufgabe in TUWEL abgeben. Die Namen MIT Matrikelnummern aller Gruppenmitglieder müssen im abgegebenden Dokument am Anfang ihres Dokuments vermerkt sein.

### Abgabe Ihrer Lösungen

Die Abgabe Ihrer Lösungen erfolgt elektronisch via TUWEL (TU Wien E-Learning Center). Sie finden einen Link dorthin am Beginn der LVA-Seite im TISS ("Zum TUWEL Online-Kurs").

Die Abgabe muss via TUWEL bis spätestens 15. Dezember 2011, 12:00, erfolgen. Nachträgliche Abgaben werden nicht berücksichtigt.

### Fragen

Bei Fragen zu TUWEL wenden Sie sich bitte zunächst an das TUWEL-Hilfe-System und Ihre Kollegen. Sollte in der Angabe etwas unklar sein, sehen Sie bitte im TUWEL nach, ob dort bereits ergänzende Anmerkungen zum Übungsteil stehen. Wichtige Ankündigungen zum Übungsteil werden auch in der Vorlesung und online bekannt gegeben.

# 4. Übungsaufgabe

# 1 Zusicherungen, Non-Termination

Ein nicht ganz so bekannter MP3-Player einer bekannten IT-Firma hatte einen lästigen Bug, der am 31. Dezember in Schaltjahren auftrat: Der Startvorgang terminierte nicht. Verantwortlich war dafür eine fehlerhafte Schleife, die die aktuelle Jahreszahl ermitteln sollte:

```
static boolean isLeapYear(int year) { /* Liefert true, wenn year ein Schaltjahr ist */ }

static int year(int days) {
    // days = Tage, die seit 1. Jaenner 1980 vergangen sind.

    int year = 1980;

    while (days > 365) {
        if (isLeapYear(year)) {
            if (days > 366) {
                days -= 366;
                year += 1;
            }
        } else {
            days -= 365;
            year += 1;
        }
    }

    return year;
}
```

## Aufgabestellung

- Warum terminiert die Schleife nicht immer?
- Ergänzen Sie den Programmcode um Schleifen-Invarianten und erklären Sie, welche dieser Invarianten verletzt ist. Sie können dafür neue lokale Variablen anlegen, dürfen aber den Programm-Code sonst nicht ändern.
- Geben Sie eine korrekte Version der Methode `year(int days)` an (inklusive Schleifen-Invarianten).

## 1.1 Lösung

Die Schleife terminiert nicht, da im Fall `isLeapyear(year)` und `days == 366` keine Werte verändert werden und somit die Abbruchbedingung unerfüllt bleibt.

Invarianten:

- `year >= 1980`
- `days >= 0`
- `days == 365 * (year - 1980) + Anzahl der Schaltjahre[1980 .. years]`

Mit zusätzlicher lokalen Variable (z.B. Zwischenspeichern von `year`):

```
static int year(int days) {
    int lastYear = 0;
    int year = 1980;

    while (days > 365) {
        lastYear = year;

        if (isLeapYear(year)) {
            if (days > 366) {
                days -= 366;
                year += 1;
            }
        } else {
            days -= 365;
            year += 1;
        }
    }

    return year;
}
```

Invariante: `lastYear < year`; diese Invariante ist nicht erfüllt (nämlich im Fehlerfall).

Korrigierte Version:

```
static int year(int days) {
    int year = 1980;

    while (days > 366 || (days > 365 && !isLeapYear(year))) {
        if(isLeapYear(year)) {
            // In diesem Fall ist days > 366!
            days -= 366;
        } else {
            days -= 365;
        }

        year += 1;
    }

    return year;
}
```

## 2 Generischer Array-Stack

Die folgende Klasse repräsentiert einen Stack, implementiert über ein Array, für Integer-Werte. Zusätzlich ist das Interface `Iterable` (aus `java.lang`) implementiert. Dieses enthält eine Methode, die einen Iterator zurückliefert.

Der Iterator für unseren Stack ist als *innere Klasse* ausgeführt, damit wir darin auf die `private`-Objektvariablen zugreifen können, ohne weitere Methoden dem Stack hinzuzufügen.

```
import java.util.Iterator;

public class Stack implements Iterable<Integer> {
    private int[] array;
    private int pointer; // Index des ersten freien Feldes

    public Stack(int maxCapacity) {
        this.array = new int[maxCapacity];
        this.pointer = 0;
    }

    public void push(int value) { array[pointer++] = value; }

    public int pop() { return array[--pointer]; }

    public Iterator<Integer> iterator() { return new StackIterator(); }

    // Als innere Klasse, damit auf die private-Objektvariablen der
    // Eltern-Klasse zugegriffen werden kann
    private class StackIterator implements Iterator<Integer> {
        int index;
        StackIterator() { index = 0; }

        public boolean hasNext() { return index < pointer; }
        // public int next() { return array[index++]; }
        public Integer next() { return array[index++]; } // Korrigiert 6.12.
        public void remove() {
            // Optionale Methode, wird hier nicht unterstuetzt.
            throw new UnsupportedOperationException();
        }
    }
}
```

Man kann Instanzen des Interface `Iterable` in *for-each*-Schleifen verwenden:

```
Stack stack = new Stack(10);
stack.push(1);
stack.push(2);
```

```
stack.push(3);

for(int i : stack) { // Weil stack eine Instanz von Iterable<Integer> ist
     System.out.print(i); // Ausgabe: "123"
}
```

## Aufgabenstellung

Wandeln Sie den Stack (inklusive Iterator) in eine generische Klasse `Stack<T>` um.

**Anmerkung 6.12.2011:** Sie können in Java keine generischen Arrays in der Form `new T[maxCapacity]` anlegen. Es stehen ihnen mehrere Möglichkeiten zur Verfügung, diese Einschränkung von Generizität in Java zu umgehen:

- Sie können für diese Aufgabe dieses Problem ignorieren.
- Sie können ein Array vom Typ `Object` anlegen und auf `T[]` casten (`(T[]) new Object(maxCapacity)`). Diese Möglichkeit ist im Allgemeinen nicht empfehlenswert, da es zu Typ-Fehlern kommen kann.
- Sie können das Array durch eine `ArrayList` ersetzen.

```
// Stack fuer Integer mit ArrayLists
public static class Stack implements Iterable<Integer> {
    private ArrayList<Integer> array;
    private int pointer; // Index des ersten freien Feldes

    public Stack(int maxCapacity) {
        this.array = new ArrayList<Integer>(maxCapacity);
        this.pointer = 0;
    }

    public void push(int value) { array.set(pointer++, value); }

    public int pop() { return array.get(--pointer); }

    public Iterator<Integer> iterator() { return new StackIterator(); }

    // Als innere Klasse, damit auf die private-Objektvariablen der
    // Eltern-Klasse zugegriffen werden kann
    private class StackIterator implements Iterator<Integer> {
        int index;
        StackIterator() { index = 0; }

        public boolean hasNext() { return index < pointer; }
        public Integer next() { return array.get(index++); }
        public void remove() {
```

```
            // Optionale Methode, wird hier nicht unterstuetzt.
            throw new UnsupportedOperationException();
        }
    }
}
```

## 2.1 Lösung

```
// Stack fuer Integer mit ArrayLists
public static class Stack<T> implements Iterable<T> {
    private ArrayList<T> array;
    private int pointer; // Index des ersten freien Feldes

    public Stack(int maxCapacity) {
        this.array = new ArrayList<T>(maxCapacity);

        // Anlegen von maxCapacity Elementen; set und get fuehren sonst
        // zu Fehlern.

        for(int i = 0; i < maxCapacity; i++) this.array.add(null);

        this.pointer = 0;
    }

    public void push(T value) { array.set(pointer++, value); }

    public T pop() { return array.get(--pointer); }

    public Iterator<T> iterator() { return new StackIterator(); }

    // Als innere Klasse, damit auf die private-Objektvariablen der
    // Eltern-Klasse zugegriffen werden kann
    private class StackIterator implements Iterator<T> {
        int index;
        StackIterator() { index = 0; }

        public boolean hasNext() { return index < pointer; }
        public T next() { return array.get(index++); }
        public void remove() {
            // Optionale Methode, wird hier nicht unterstuetzt.
            throw new UnsupportedOperationException();
        }
    }
}
```

# 3 `Comparable` und `Comparator`

Gegeben ist die folgende Klasse `IntNumber`, die eine ganze Zahl repräsentiert:

```
public class IntNumber {
    private int value;

    public IntNumber(int initValue) {
        value = initValue;
    }

    public int get() {
        return value;
    }
}
```

Im folgenden Programm-Code fügen wir einer Liste verschiedene Instanzen von `IntNumber` hinzu:

```
Scanner sc = new Scanner(System.in);

List<IntNumber> list = new LinkedList<IntNumber>();

while(sc.hasNextInt()) {
    list.add(sc.nextInt());
}
```

Wir wollen nun die Liste sortieren. Anstatt jedoch selbst einen Sortieralgorithmus zu implementieren, wollen wir auf vorgefertigte Methoden zurückgreifen. In der Klasse `java.util.Collections` gibt es dazu zwei statische Methoden (wir haben einige irrelevante Typ-Spezifkatoren weggelassen):

1. `void sort(List<T> list)`, und
2. `void sort(List<T> list, Comparator<T> c)`.

## 3.1 `Comparable`

Um die erste Methode nutzen zu können, muss `IntNumber` das Interface `Comparable<IntNumber>` implementieren. Lesen Sie dazu die Dokumentation zu diesem Interface (z.B. unter `http://docs.oracle.com/javase/6/docs/api/java/lang/Comparable.html`) und implementieren Sie alle darin definierten Methoden in `IntNumber`.

## 3.2 `Comparator`

Die zweite Methode bietet die Möglichkeit, einen selbst definierten "Vergleicher" anzugeben. Das kann man nutzen, um die Reihenfolge beim Sortieren zu ändern.

Definieren Sie eine Klasse, die das `Comparator`-Interface für `IntNumber` implementiert (sie dürfen dabei auf die Methoden aus `Comparable` in `IntNumber` zugreifen), sodass der Aufruf von `void sort(List<IntNumber> list, Comparator<IntNumber> c)` alle Werte in der Liste *absteigend* sortiert.

Die Dokumentation von `Comparator` finden Sie z.B. hier: `http://docs.oracle.com/javase/6/docs/api/java/util/Comparator.html`

### 3.3 Lösung

```
public class IntNumber implements Comparable<IntNumber> {
    private int value;

    public IntNumber(int initValue) {
        value = initValue;
    }

    public int get() {
        return value;
    }

    public int compareTo(IntNumber that) {
        // return this.value - that.value kann Probleme
        // bei Ueberlaeufen verursachen
        return this.value < that.value ? -1 : this.value == that.value : 0 : 1;
    }
}

public class MyComparator implements Comparator<IntNumber> {
    public int compare(IntNumber a, IntNumber b) {
        // Es soll absteigend sortiert werden
        return b.compareTo(a);
    }
}
```