

Hochschule -
Fakultät IV – Technische Informatik
Modul: Informatik
Professor: -

Portfolio

von
Sebastian Schramm Matrikel-Nr. -

6. Februar 2021

Inhaltsverzeichnis

1	Deckblatt	3
2	Persönlichkeit der Informatik	3
3	Dezimalzahlen in Hexadezimalsystem	3
4	Äquivalenz mit XOR	4
5	UML	4
6	zweiKonzentrischeQuadrate	5
7	IntStack	6
7.1	Main.java	6
7.2	IntStack.java	7
8	Aritmetischer Ausdruck	8
9	Automaten	8
10	Reflexion	9

1 Deckblatt

Deckblatt mit Ihrem vollständigen Namen, Matrikelnummer, Semester, Studiengang und Ihre unterschriebene Erklärung, dass Sie das Portfolio selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt haben. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen.

2 Persönlichkeit der Informatik

Schreiben Sie einen Aufsatz über eine Persönlichkeit der Informatik. Dieser Aufsatz sollte maximal 1500 Zeichen lang sein (ca. eine DIN A4 Seite).

Konrad Zuse und seine Errungenschaften Konrad Ernst Otto Zuse war ein deutscher Bauingenieur, Erfinder und Unternehmer der Zuse KG. Konrad Zuse wurde am 22. Juni 1910 in Deutsch-Wilmersdorf geboren und starb am 18. Dezember 1995 in Hünfeld. Zuse baute 1941 den ersten funktionstüchtigen, vollautomatischen, programmgesteuerten und frei programmierbaren, in binärer Gleitkommarechnung arbeitenden Rechner und somit den ersten funktionsfähigen Computer der Welt (Wiki, 2021). Aufgrund der monotonen und mühseligen Berechnungen im Bauingenieurwesen, wollte Zuse diese Arbeit automatisieren. Er entschloss sich 1935, seinen Beruf als Statiker zu kündigen und widmete sich der Entwicklung eines mechanischen Gehirns. 1937 wurde der erste mechanische Rechner Z1 fertiggestellt und basierte auf dem Binärsystem. Zusätzlich besaß der Z1 ein Ein-/Ausgabewerk, ein Rechenwerk, ein Speicherwerk und ein Programmwerk, das die Programme von gelochten Kinofilmstreifen ablas (Wiki, 2021). Zuse entwickelte Methoden auf der Grundlage von Mantisse und Exponenten, dies ermöglichte dem Z1 mit Gleitkommazahlen zu arbeiten (vgl. Wiki, 2021). Aufgrund der Unzuverlässigkeit des Z1, musste Zuse eine Alternative für die Schaltelemente finden. Dadurch entstand der Z2 welcher nun Relais verwendet. 1941 wurde der Z3 vollendet welcher der erste elektrisch programmierbare Computer der Welt war, welcher aber 1943 bei einem Bombenangriff zerstört wurde. Während des Zweiten Weltkrieges wurde von Zuse eine Erweiterung des Z3 gebaut. Welcher in den letzten Kriegsmonaten im Algäu versteckt wurde. Am Ende wurde der Z4 dann an die technische Hochschule Zürich verkauft. (vgl. Mark und Eniac, 2900)

- 1 https://de.wikipedia.org/w/index.php?title=Konrad_Zuse&oldid=204389120
- 2 <https://homecomputermuseum.de/historie/konrad-zuses-z1-z4/>

3 Wandeln Sie die folgenden Dezimalzahlen in das Hexadezimalsystem um: 2989, 57005, 48879.

- $2989 : 16 = 186 \text{ | Rest } 2989 - 16 * 186 = 13$
- $186 : 16 = 11 \text{ | Rest } 186 - 16 * 11 = 10$
- $11 : 16 = 0 \text{ | Rest } 11$

2989 = BAD

- $57005 : 16 = 3562 \text{ | Rest } 57005 - 16 * 3562 = 13$
- $3562 : 16 = 222 \text{ | Rest } 3562 - 16 * 222 = 10$
- $222 : 16 = 13 \text{ | Rest } 222 - 16 * 13 = 14$
- $13 : 16 = 0 \text{ | Rest } 13$

57005 = DEAD

- $48879 : 16 = 3054 \text{ | Rest } 48879 - 16 * 3054 = 15$
- $3054 : 16 = 190 \text{ | Rest } 3054 - 16 * 190 = 14$
- $190 : 16 = 11 \text{ | Rest } 190 - 16 * 11 = 14$
- $11 : 16 = 0 \text{ | Rest } 11$

48879 = BEEF

4 Wie zeigen Sie, die Äquivalenz der folgenden Ausdrücke mit XOR

$(x \vee y) \wedge \neg(x \wedge y)$				
x	y	$(x \vee y)$	$\neg(x \wedge y)$	$a \wedge b$
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

$\neg((x \wedge y) \vee (\neg x \wedge \neg y))$				
x	y	$(x \wedge y)$	$(\neg x \wedge \neg y)$	$\neg(a \vee b)$
0	0	0	1	0
0	1	0	0	1
1	0	0	0	1
1	1	1	0	0

$\neg(x \wedge y) \wedge (x \vee y)$				
x	y	$\neg(x \wedge y)$	$(x \vee y)$	$a \wedge b$
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

5 UML

Schreiben Sie einen Aufsatz zum Thema UML. Dieser Aufsatz sollte maximal 3000 Zeichen lang sein (ca. zwei DIN A4 Seiten).

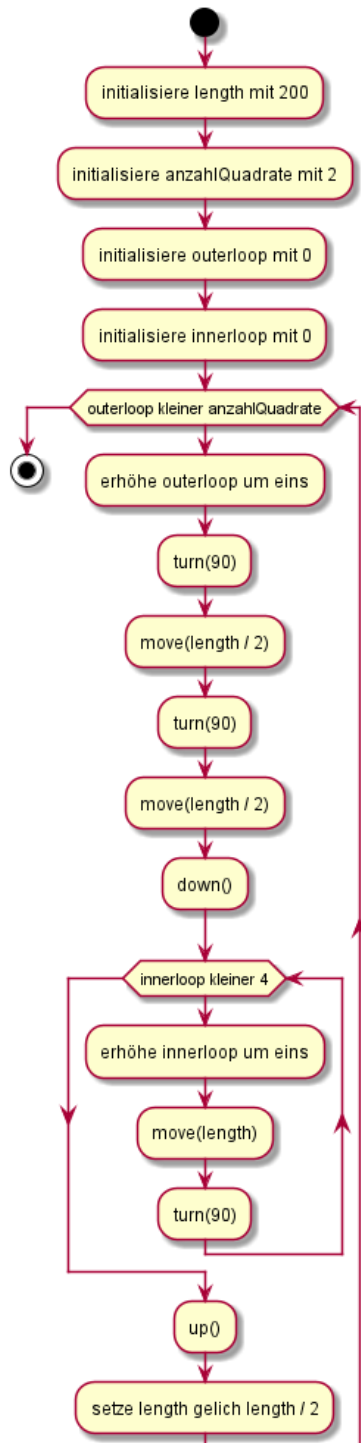
UML Während der Blüte der Objekt orientierten Programmierung in den 1990er Jahren, kamen vermehrt Vorschläge für eine Modellierungssprache. Dazu haben sich die drei Grady Booch, Ivar Jacobson und James Rumbaugh. Sie hatten schon ihre eigenen Modellierungssprachen entworfen, nun haben sich zusammengesetzten und daraus ein Konzept der UML entwickelten. Die UML oder auch Unified Modeling Language ist eine Objektorientierte Sprache und besitzt Notationen zur Beschreibung von Softwaresystemen. Der Grundgedanke hierbei, ist eine einheitliche Notation für alle Softwaresysteme. Sie dient zur Spezifikation, Konstruktion, Dokumentation und Visualisierung von Software-Teilen und anderen Systemen (UML, 2020). In der UML gibt es Notationselemente, aus denen sich verschiedene Diagramme erzeugen lassen. Im Jahre 1999 stieß OMG die Entwicklung von UML 2.0 an. Dabei verdeutlicht jedes Diagramm eine bestimmte Perspektive auf das zu modellierende System und kann dem Entwickler, oder Nutzer einen guten Überblick auf die Software verschaffen. Es gibt verschiedene Modellierungssprachen, wie das Aktivitätsdiagramm welches die Ablaufmöglichkeiten eines Systems beziehungsweise Anwendungsfalls mit Aktionen darstellt. Wobei eine Aktion einen einzelnen schritt darstellt und mit Kanten wird der Kontrollfluss angegeben. Bei einem Anwendungsfalldiagramm oder auch Use Case Diagram, wird das Verhalten eines Systems aus der Benutzersicht dargestellt. Diese beschreiben aber auch die geplante Funktionalität eines Systems. Anschließend wird noch ein Akteur benötigt, dieser kann zum Beispiel eine Person oder ein System sein und gibt an, was dieses System tun soll. (vgl. UML, 2021) Bei einem Klassendiagramm werden die Beziehungen zwischen den einzelnen Klassen verdeutlicht sowie, welche Attribute und Methoden es in der jeweiligen Klasse vorhanden sind. Zusätzlich ist noch ersichtlich um welche Datentypen es sich handelt und welche Attribute/Methoden public, private oder protected sind. Mit der Generalisierung wird angegeben in welcher Beziehung die jeweiligen Klassen zueinander stehen, dies wird mit einer durchgezogenen Linie und einem an einen der beiden enden dargestellt. Der Pfeil gibt an von welcher Klasse geerbt wird. (vgl. Klassendiagramm, 2021) Ein Zustandsdiagramm beschreibt den Lebenszyklus der Objekte einer Klasse und ist eine graphische Darstellung eines Zustandsautomaten welche auf dem Konzept der endlichen Automaten basieren. Mithilfe eines Zustandsdiagrammes kann dargestellt werden, in welchem Zustand sich das betrachtete Objekt befindet und welches Verhalten diese Objekt in einer aktiven Klasse modelliert. (vgl. Endlicher_Automat, 2021)

- 1 https://de.wikipedia.org/wiki/Unified_Modeling_Language
- 2 <https://de.wikipedia.org/wiki/Klassendiagramm>
- 3 https://de.wikipedia.org/wiki/Endlicher_Automat

6 zweiKonzentrischeQuadrate

Damit dieses Programm funktioniert gehen dir davon aus, dass wir oben Links starten. Also bei 0,0 und unser Startrichtung nach rechts(y) verläuft. Zusätzlich kann sich das Programm merken, in welche richtung sich der Kopf zuletzt bewegt hat.

```
1 int length = 200;
2 int anzahlQuadrate = 2;
3
4 wiederhole (anzahlQuadrate) {
5     turn(90);
6     move(length / 2);
7     turn(90);
8     move(length / 2);
9     down();
10    wiederhole (4) {
11        move(length);
12        turn(90);
13    }
14    up();
15    length = length / 2;
16 }
```



7 IntStack

Schreiben Sie eine Java-Klasse IntStack, die einen ADT Stack implementiert. Die Größe (Kapazität) des Stacks soll fix sein und bei der Erzeugung eines IntStack-Objekts festgelegt werden. Intern sollen die Daten in einem int-Array abgelegt werden. Geben Sie den Source-Code ab.

7.1 Main.java

```

1 package aufgabe_08;
2

```

```

3 public class Main {
4
5     public static void main(String[] args) {
6         IntStack stack = new IntStack(32);
7         stack.print();
8         System.out.println(stack.empty());
9         stack.push(12);
10        stack.push(2);
11        for (int i = 0; i < 12; ++i)
12            stack.push((int) (Math.random()*100));
13        stack.print();
14        System.out.println(stack.empty());
15        System.out.println(stack.full());
16        stack.pop();
17        stack.pop();
18        stack.pop();
19        stack.pop();
20        stack.print();
21        for (int i = 0; i < 12; ++i)
22            stack.push((int) (Math.random()*100));
23        System.out.println(stack.empty());
24        System.out.println(stack.full());
25        stack.print();
26        for (int i = 0; i < 12; ++i)
27            stack.push((int) (Math.random()*100));
28        System.out.println(stack.empty());
29        System.out.println(stack.full());
30        stack.print();
31
32        for (int i = 0; i < 40; ++i)
33            stack.pop();
34
35        stack.push(3);
36        stack.print();
37    }
38 }

```

7.2 IntStack.java

```

1 package aufgabe_08;
2
3 public class IntStack {
4     private int[] stack;
5     private int stackPos;
6
7     public IntStack(int size) {
8         init(size);
9         stackPos = -1;
10    }
11
12    /**
13     * Legt ein neues stack array mit der gröSse size an
14     * @param size
15     */
16    private void init(int size) {
17        this.stack = new int[size];
18    }
19
20    /**
21     * Prüft ob die stackPos gleich -1 ist und gibt true zurück,
22     * andernfalls false
23     * @return

```

```

24     */
25     public boolean empty() {
26         return (stackPos == -1);
27     }
28
29     /**
30      * Prüft ob die stackPos gröSSer gleich stack.length ist und
31      * gibt true zurück, andernfalls false
32      * @return
33      */
34     public boolean full() {
35         return (stackPos >= stack.length-1);
36     }
37
38     /**
39      * Fügt eine Zahl zum stack hinzu
40      * @param number
41      */
42     public void push(int number) {
43         //Prüft ob der stack voll ist
44         if (!full())
45             stack[++stackPos] = number;
46     }
47
48     /**
49      * Entfernt eine Zahl vom stack
50      */
51     public void pop() {
52         //Prüft ob der stack leer ist
53         if (!empty())
54             stack[stackPos--] = 0;
55     }
56
57     /**
58      * Gibt die letzte Zahl aus dem stack zurück
59      * @return
60      */
61     public int top() {
62         return stack[stackPos];
63     }
64
65     /**
66      * Gibt alle Zahlen die sich im stack befinden aus
67      */
68     public void print() {
69         for (int tmp : stack)
70             System.out.print(tmp + ", ");
71         System.out.println();
72     }
73 }

```

8 Aritmetischer Ausdruck

Preoder: $*+a/bc-d*ef$

Inorder: $a+b/c*d-e*f$

Postorder: $abc/+def*-*$

9 Automaten

Schreiben Sie einen Aufsatz zum Thema Automaten. Dieser Aufsatz sollte maximal 3000 Zeichen lang sein (ca. zwei DIN A4 Seiten).

Automaten Ein Automat ist ein Bestandteil der Informatik. Es ist egal, ob ein System sinnvoll oder gar möglich ist zu bauen. Mithilfe eines Automaten kann das Verhalten leichter verstanden werden. Im Prinzip ist das Verhalten eines Automaten immer gleich. Der Automat befindet sich in einem bestimmten Zustand. Jedes Mal, wenn ein Eingabezeichen eintrifft, kann sich abhängig vom Eingabezeichen und dem gegenwärtigen Zustand ein neuer Zustand, der Folgezustand, einstellen. (Automat, 2021) Es wird unterschieden zwischen einem Deterministischen und nichtdeterministischen Automaten. Bei einem Deterministischen Automaten treten nur definierte und reproduzierbare Zustände auf. Das heißt, wenn ich ein Wort einlese, werden die selben Zustände durchlaufen und ich bekomme so am Ende die selbe Ausgabe. Der Begriff Determinismus ist vom Begriff Determiniertheit zu unterscheiden: Ein deterministischer Algorithmus ist immer determiniert, d. h., er liefert bei gleicher Eingabe immer die gleiche Ausgabe. (Determinismus, 2021) Andersherum gilt dies nicht. Zum Beispiel ist der Quicksort nicht-deterministisch, weil dessen Zwischenergebnisse sich stets unterscheiden, jedoch ist das Ergebnis nach der Terminierung immer identisch. (vgl., Determinismus, 2021) Während hingegen bei einem nichtdeterministischen Automaten bei gleicher Eingabe mehrere mögliche Durchlaufmöglichkeiten existieren. Im Allgemeinen sind diese Modelle nur theoretisch und praktisch nicht realisierbar. In der theoretischen Informatik dienen sie dazu, um herauszufinden, ob ein bestimmtes Problem mit einem nichtdeterministischen Algorithmus angegeben werden kann. Da sich ein Problem damit leichter lösen lässt. (vgl., Nichtdeterminismus, 2021) Unter anderem gibt es auch Automaten für die Ein- und Ausgabe. Hierbei wird unterschieden zwischen einem Akzeptor, welcher ein spezieller endlicher Automat ist. Dieser erzeugt keine Ausgaben, kann aber Wörter einlesen. Diese werden Zeichen für Zeichen eingelesen, wobei diese im selben Zustand bleiben können, oder in einen neuen übergehen. Zusätzlich besitzen sie einen Startzustand und einen Endzustand und nur dann, wenn der Finalzustand im Endzustand terminiert, gilt die Eingabe als akzeptiert. Ein Transduktor zeichnet sich dadurch aus, dass er keine Ausgabe erzeugt. Im Gegenteil, der Transduktor entwickelt aus einer vordefinierten Quellspalte ein Wort. Bei einer Ein- oder Ausgabe hat man einen Akzeptor oder einen Transduktor. Ein Akzeptor besitzt einen Startzustand und einen Endzustand. Hier wird ein Wort eingelesen und Zeichen für Zeichen verarbeitet. Wenn der Automat am Ende im Endzustand aufhört, wird die Eingabe akzeptiert. Im Gegensatz zum Akzeptor, erzeugt der Transduktor eine Ausgabe. Hier werden entweder jedem oder paarweise der Zustand vom Eingabezeichen zu einem Ausgabezeichen zugeordnet. Aber welche Automaten werden in der Praxis bei der Programmierung verwendet? Dafür kommen Endliche Automaten und Kellerautomaten in Frage, da diese komplexe Probleme übersichtlich lösen lassen können. (vgl., Automat, 2021)

- 1 [https://de.wikipedia.org/wiki/Automat_\(Informatik\)](https://de.wikipedia.org/wiki/Automat_(Informatik))
- 2 [https://de.wikipedia.org/wiki/Determinismus_\(Algorithmus\)](https://de.wikipedia.org/wiki/Determinismus_(Algorithmus))
- 3 <https://de.wikipedia.org/wiki/Nichtdeterminismus>

10 Reflexion

- Was waren Ihre zwei größten Herausforderungen in diesem Modul?

Einmal die Aufsätze zu schreiben, da es mir bei einigen Themen schwer fiel, aufgrund der Menge an wichtigen Informationen. Diese aber nur in ein Paar setzen wiederzugeben.

Und andererseits haben wir eine Aufgabe zur Turing Maschine bekommen. Den egal was ich da eingestellt hatte, sie wollte einfach nicht so wie ich es gerne hätte.

- Was zeigt das Portfolio von Ihnen und Ihrer Arbeit?

Was ich in diesem Modul gelernt habe.

- Wo sehen Sie noch Fehlstellen und Lernmöglichkeiten ihrerseits?

Mich mehr mit Automaten beschäftigen.

- Auf welche Bereiche ließe sich das von Ihnen Gelernte übertragen?

Ein großer Punkt werden vermutlich die UML Diagramme sein. Da diese essentiell für die Softwareentwicklung sind.

1. Zu Beginn des Moduls haben Sie Erwartungen gehabt, was Sie wahrscheinlich innerhalb des Moduls lernen werden und wo Sie diese Erfahrungen später anwenden können. Erläutern Sie, inwiefern diese Aussagen zutreffen und inwiefern nicht. Wie kommen ggf. Unterschiede zustande? 2. Wie sehen Sie aus

heutiger Sicht, wo Sie die Lernerfahrungen dieses Moduls im Arbeitsleben anwenden können? 4. Wie sind Sie mit diesen Herausforderungen umgegangen?