



Informatik II - Übung 1 (Spoilerfree)

Pascal Schärli

info2@pascscha.ch

25.09.2020

Administratives

- Die Übungen sind auf [CodeExpert](#)
- Die Files welche Ihr für die Übungen braucht sind auf der [Vorlesungswebseite](#).
- Meine Slides findet ihr auf meiner Website pascscha.ch/info2
- Bei Fragen: info2@pascscha.ch

Code Expert

- Code Expert bleibt gleich wie in Info I
- Wenn Ihr die Aufgaben gut genug löst, bekommt ihr XP.
- Mit XP kann man Bonusübungen freischalten, welche bis zu $\frac{1}{4}$ Note Bonus auf die Prüfung geben.
- Textaufgaben könnt ihr mit Markdown schön formatieren.
 - Schaut euch mal [Julias Cheatsheet](#) an

Code Expert - Files hochladen

Gerichtete Graphen - Student Attempt

Project Files

- main.md
- test_1.pdf
- test.pdf

Click or drag file to this area to upload files into this folder.
Support for a single or bulk upload. Maximum file size 2 MB each.

1. Ordner auswählen

2. Datei hochladen

3. Submit

Already submitted
Your last submission counts

Create new Submission

Aufgabenstellung

Ein gerichteter Graph besteht aus einer Menge von Kanten und einer Menge von Knoten, wobei jede Kante als Pfeil dargestellt wird, welcher einen Ausgangsknoten mit einem Zielknoten verbindet. Knoten stellt man im Allgemeinen in Form von Kreisen dar. Betrachten Sie die folgende Problemstellung:

Sie besitzen drei Kannen; die erste fasst 8 Liter und ist bis zum Rand mit Wasser gefüllt, die zweite fasst 5 Liter und ist leer, die dritte fasst 2 Liter und ist ebenfalls leer. Dies ist der Startzustand. Eine Umschüttung von Wasser aus einer Kanne in eine andere führt von einem Zustand in einen anderen, wobei entweder eine Kanne vollständig geleert oder vollständig gefüllt werden muss. Ein Zustand wird durch das Tripel $(a\ b\ c)$ kodiert, wobei a der Inhalt der 8-Liter-Kanne, b der Inhalt der 5-Liter-Kanne und c der Inhalt der 2-Liter-Kanne ist.

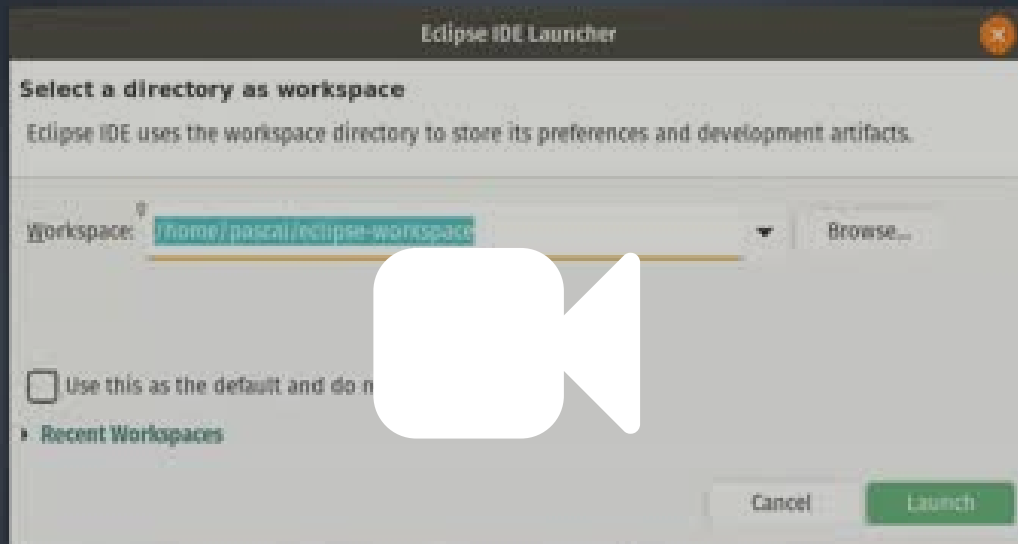
- Zeichnen Sie den Graphen, der aus allen möglichen Zuständen (= Knoten) und möglichen Umschüttungen (= gerichteten Kanten) besteht. Markieren Sie die Knoten des Graphen mit den Zuständen, die durch die Folge der Umschüttungen auftreten. Achten Sie darauf, dass Knoten mit gleichem Tripel nicht mehrfach im Graphen auftreten.
- Versehen Sie die Knoten des Graphen mit der minimalen Anzahl von benötigten Umschüttungen, um den entsprechenden Zustand vom Startzustand ausgehend zu erreichen. Wie gehen Sie dabei vor?

Eclipse

- Da der Text Editor in CodeExpert noch nicht gleich mächtig ist wie eine **IDE**, benutzen wir Eclipse fürs Programmieren.
- Eclipse könnt ihr hier gratis herunterladen:
<https://www.eclipse.org/downloads/>
- Workflow:
 1. Ladet euch das .zip File der Übung von der **Vorlesungsseite** herunter
 2. Löst die Übung in Eclipse
 3. Kopiert euer Code zu CodeExpert

Eclipse

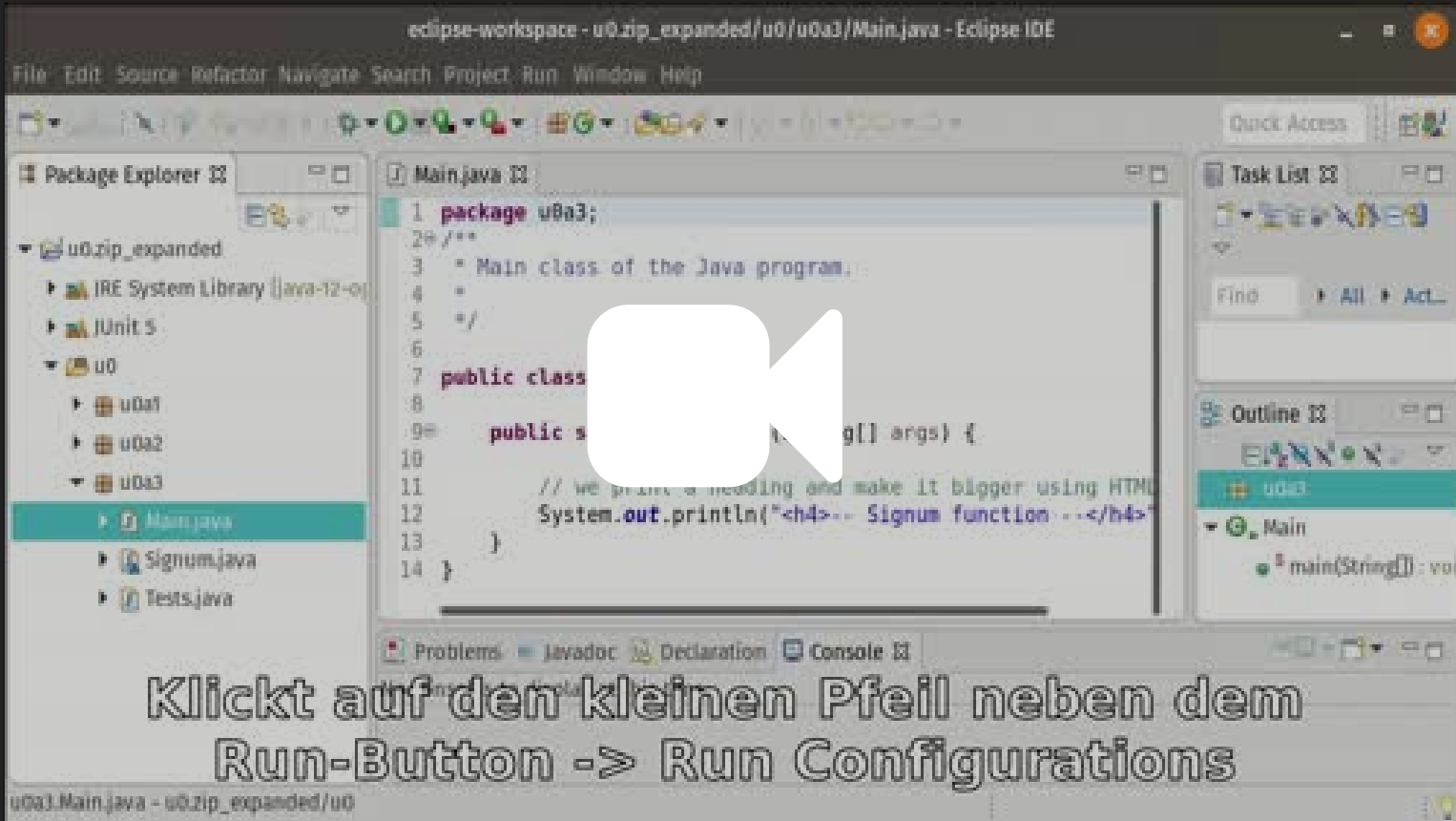
Wie man das .zip File in Eclipse öffnet



Den Workspace müsst ihr nicht ändern

JUnit

Wie man automatische Tests mit JUnit macht



Eclipse Shortcuts

- Ctrl+Shift+F Code Formattieren
- Ctrl+Space Autocomplete
- Ctrl+D Zeile Löschen
- Ctrl+Shift+7 markierte Zeilen als Kommentar
- Ctrl+F11 Programm ausführen
- Ctrl+Shift+L Liste aller Shortcuts anzeigen

Eclipse Shortcuts

- **Ctrl+Shift+F** **Code Formattieren**

- Ctrl+Space
- Ctrl+D
- Ctrl+Shift+7
- Ctrl+F11
- Ctrl+Shift+L

Autocomplete
Zeile Löschen
markierte Zeilen als Kommentar
Programm ausführen
Liste aller Shortcuts anzeigen

Nachbesprechung



Serie 0 - Hello World

1. Java Development Kit (JDK) Version 13 herunterladen:
 - **OpenJDK - Open Source, GPL**
 - (OracleJDK - Proprietär & Login erforderlich)
2. HelloWorld.java in Konsole ausführen
3. Programm in **Eclipse** ausführen
4. Programm in **CodeExpert** rüberkopieren und einreichen

Serie 0 - Signum

1. Programm kann mit dem grünen Knopf ausgeführt werden



2. Fügt in Main.java eine weitere Zeile ein, z.B.:

```
System.out.println("signum(3) = " + Signum.signum(3));
```

3. Programm in **CodeExpert** rüberkopieren und einreichen

Serie 0 - JUnit

1. Im Video vorher habe ich gezeigt wie ihr JUnit einbinden könnt.

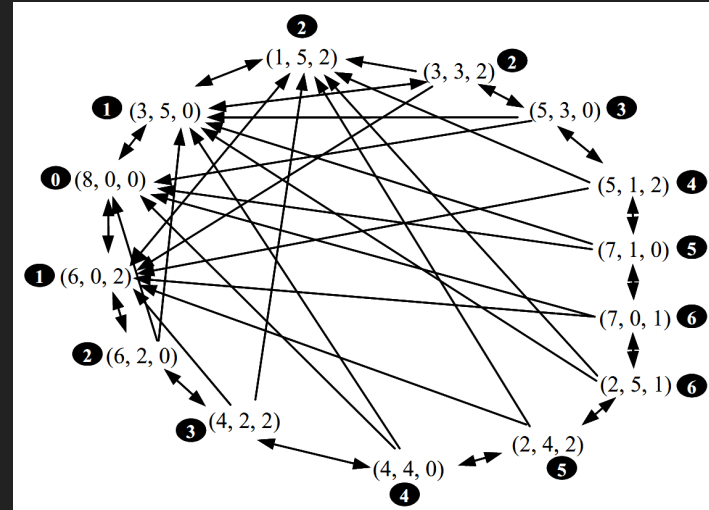
2.

```
@Test public void positive2() {  
    Assert.assertEquals(1, Signum.signum(10));  
}
```

3. Programm in [CodeExpert](#) rüberkopieren und einreichen

Serie 0 - Graphen

Zeichnen Sie den Graphen,
der aus allen möglichen
Zuständen und
Umschüttungen besteht.



$$\frac{\sum f(x) * x}{\sum f(x)} = \frac{1*2 + 2*3 + 3*2 + 4*2 + 5*2 + 6*2}{2 + 3 + 2 + 2 + 2 + 2}$$
$$= \frac{44}{13} \approx 3.38$$

Best of

Vorlesung

Altägyptische Multiplikation

(Russische Bauernmultiplikation / Russian Peasant Multiplication)

$$f(a, b) = \begin{cases} a & , \text{ falls } b = 1 \\ f(2a, b/2) & , \text{ falls } b \text{ gerade} \\ f(2a, \frac{b-1}{2}) + a & , \text{ sonst} \end{cases}$$

$$f(a, b) = a \cdot b$$

Altägyptische Multiplikation

(Russische Bauernmultiplikation / Russian Peasant Multiplication)

$$f(a, b) = \begin{cases} a & , \text{ falls } b = 1 \\ f(2a, b/2) & , \text{ falls } b \text{ gerade} \\ f(2a, \frac{b-1}{2}) + a & , \text{ sonst} \end{cases}$$


a	b	f(a,b)
5	98	$f(5, 98) = f(10, 49)$
10	49	$f(10, 49) = f(20, 24) + 10$
20	24	$f(20, 24) = f(40, 12)$
40	12	$f(40, 12) = f(80, 6)$
80	6	$f(80, 6) = f(160, 3)$
160	3	$f(160, 3) = f(320, 1) + 160$
320	1	$f(320, 1) = 320$

$$f(5, 98) = 320 + 160 + 10 = 490$$

Altägyptische Multiplikation

Induktiver Beweis über b (Slides 207)

Induktionsverankerung: $b = 1$

$$f(a, 1) = a = a \cdot 1$$


$$f(a, b) = \begin{cases} a & , \text{ falls } b = 1 \\ f(2a, b/2) & , \text{ falls } b \text{ gerade} \\ f(2a, \frac{b-1}{2}) + a & , \text{ sonst} \end{cases}$$

Induktionsschritt:

Gegeben, dass $f(a, b) = a \cdot b$ für $b \in [1 \dots n - 1]$.

Zeige daraus, dass $f(a, n) = a \cdot n$.

n gerade:

$$f(a, n) = f(2a, \frac{n}{2})$$


$$= 2a \cdot \frac{n}{2}$$

$$= a \cdot n$$


n ungerade:

$$f(a, n) = f(2a, \frac{n-1}{2}) + a$$

$$= 2a \cdot \frac{n-1}{2} + a$$

$$= a \cdot (n - 1) + a$$


$$= a \cdot n$$

Laufzeitkomplexität

Beispiel

```
1 static int summe(int x) {  
2     if (x == 0) return 0;  
3     return summe(x-1) + x;  
4 }
```

$$\text{summe}(5) = \text{summe}(4)^1 + 5 = 10 + 5 = 15$$

$$\text{summe}(4) = \text{summe}(3)^2 + 4 = 6 + 4 = 10$$

$$\text{summe}(3) = \text{summe}(2)^3 + 3 = 3 + 3 = 6$$

$$\text{summe}(2) = \text{summe}(1)^4 + 2 = 1 + 2 = 3$$

$$\text{summe}(1) = \text{summe}(0)^5 + 1 = 0 + 1 = 1$$


$$\text{summe}(0) = 0 = 0$$

-> $\text{summe}(x)$ ruft sich selbst x mal rekursiv auf

Exceptions

```
1 public static void pleaseDontPassZero(int i) throws IllegalArgumentException {
2     if(i == 0){
3         throw new IllegalArgumentException("I told you so!");
4     }
5 }
```

 "Werfe" einen neuen Fehler

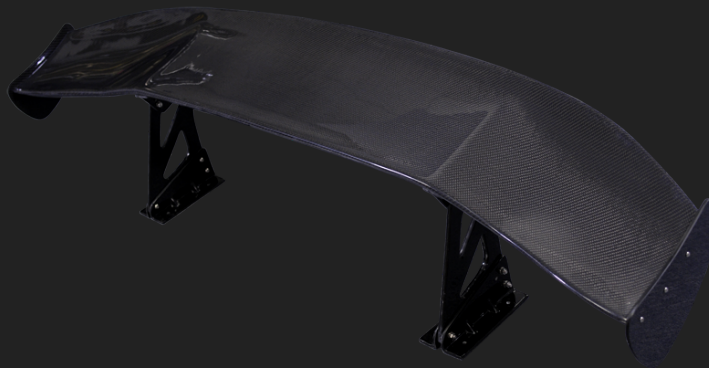
 Achtung, diese Funktion könnte einen Fehler werfen

Fehler müssen weitergeleitet oder gefangen werden:

```
1 public static void foo1() throws IllegalArgumentException {
2     pleaseDontPassZero(1);
3 }
4
5 public static void foo2() {
6     try{
7         foo1();
8     }
9     catch(IllegalArgumentException e){
10        System.out.println("Something went wrong.");
11    }
12 }
```



Vorbesprechung



Altägyptische Multiplikation

(Russische Bauernmultiplikation / Russian Peasant Multiplication)

1. Wäre der Beweis (siehe BestOf) auch mit Induktion über a möglich?
2. Beweist, dass der Algorithmus für alle erlaubten Eingabewerte terminiert

3.

```
1 static int f(int a, int b){
2     System.out.println(a + " " + b);
3     if (b == 1)
4         return a;
5     else if (b%2 == 0)
6         return f(a+a, b/2);
7     else
8         return a + f(a+a, (b-1)/2);
9 }
```



```
1 static int f(int a, int b){
2     System.out.println(a + " " + b);
3     if (b == 0)
4         return 0;
5     else if (b%2 == 0)
6         return f(a+a, b/2);
7     else
8         return a + f(a+a, (b-1)/2);
9 }
```

Laufzeitkomplexität

1. Wie oft rufen sich die Funktionen auf bevor sie terminieren?
2. Wie viele Funktionen werden bei $f(a,b)$ vor der return-Zeile aufgerufen?
3. Wie viele Funktionen werden bei $f(a,b)$ inklusive return-Zeile aufgerufen?

```
1 static boolean gerade(int x) {
2     if (x == 0) return true;
3     return !gerade(x-1);
4 }
5
6 static int verdopple(int x) {
7     if ( x == 0 ) return 0;
8     return 2 + verdopple(x-1);
9 }
10
11 static int halbiere(int x) {
12     if (x == 0) return 0;
13     if (x == 1) return 0;
14     return 1 + halbiere(x-2);
15 }
16
17 static int f(int a, int b) {
18     if (b == 0) return 0;
19     if (gerade(b))
20         return f(verdopple(a), halbiere(b));
21     else
22         return a + f(verdopple(a), halbiere(b));
23 }
```


Überprüfung Benutzereingaben

- Teilaufgabe a)
 - Implementation der Altägyptische Multiplikation gegeben, nur positive Zahlen erlaubt!
 - Nicht erlaubte Eingaben müssen einen Fehler (Exception) auslösen
- Teilaufgabe b)
 - Findet den Fehler in der Funktion f

```
1 private static int f(int a, int b) {  
2     if (b==0) return a;  
3     if (b%2 == 0) return f(2*a, b/2);  
4     else return a + f(2*a, b/2);  
5 }
```

Überprüfung Benutzereingaben

c) - Java Doc

- Java Doc: Standard für Java Dokumentationen (Kommentarte)
- Vor jeder Funktion:

```
/**
 * Beschreibung
 *
 * @param a      Beschreibung Variable a
 * @param b      Beschreibung Variable b
 * @return      Beschreibung Rückgabewert
 * @throws      Beschreibung Exceptions
 */
```

- In Eclipse: /** + Enter

Viel Spass!

When you turn 32

