

Arrays, Strings, Referenzen, Pass-By-Value

Der Zufallssurfer / Pagerank

3. JAVA II

Arrays - Motivation

Programm: (Eingaben unterstrichen)

```
Wie viele Tage? 7  
Regenmenge Tag 0? 8  
Regenmenge Tag 1? 12  
Regenmenge Tag 2? 13  
Regenmenge Tag 3? 9  
Regenmenge Tag 4? 80  
Regenmenge Tag 5? 7  
Regenmenge Tag 6? 100
```

Mittelwert: 32.714285714285715

Ausnahmeereignisse: 2

Wo liegt das Problem?

Jede Eingabe zweifach benötigt

- Berechnung des Durchschnittes
- Zählen, wie viele Tage weit über dem Mittelwert lagen

Wir könnten jeden Wert in einer Variablen speichern

- Anzahl benötigter Variablen vorher aber unbekannt.

→ Wir benötigen ein Array.

Arrays

Array: Objekt, welches mehrere Wertes desselben Typs speichert

Element: Ein Wert an einem Index des Arrays.

Index: Position eines Elements im Array. Startet bei 0.

<i>Index</i>	0	1	2	3	4	5	6	7	8	9
<i>Wert</i>	12	49	-2	26	5	17	-6	84	72	3

Element 0				Element 4					Element 9
-----------	--	--	--	-----------	--	--	--	--	-----------

Array Deklaration

Typ[] ***Name*** = new ***Typ***[***Länge***];

Beispiel:

int[] werte = new **int**[10];

<i>Index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>
<i>Wert</i>	0	0	0	0	0	0	0	0	0	0

Elementzugriff

***name*[*index*]** **// Zugriff**
***name*[*index*] = value;** **// Modifikation**

Beispiel

```
werte[0] = 27;
```

```
werte[3] = -2;
```

```
System.out.println(werte[0]);
```

```
if (werte[3] < 0) {
```

```
    System.out.println("Element 3 is negative.");
```

```
}
```

<i>index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>
<i>value</i>	27	0	0	-2	0	0	0	0	0	0

Arrays

Arrays sind grundsätzlich *dynamisch* erzeugt

```
int [] b; // array von ints
```

```
b = new int[10]; // Grösse 10: Indizes von 0..9
```

```
...
```

```
b = new int[20]; // kann neu zugewiesen werden
```

Die Grösse eines Arrays kann also zur Laufzeit festgelegt werden.

- Ein Array wächst jedoch nicht automatisch

Arrays sind nicht primitiv

Arrays tragen *Metadaten* mit sich herum:

```
int[] sq = new int[7];  
for (int i=0; i < sq.length; i++) {  
    sq[i]=i*i;  
}  
sq[8] = 64;
```

`java.lang.ArrayIndexOutOfBoundsException`

Es funktioniert sogar `print(sq);` mit

```
static void print (int a[]) {  
    for (int i=0; i < a.length; ++i)  
        System.out.println("a[" + i + "]= " + a[i]);  
}
```


Regenmengen: Eingabe

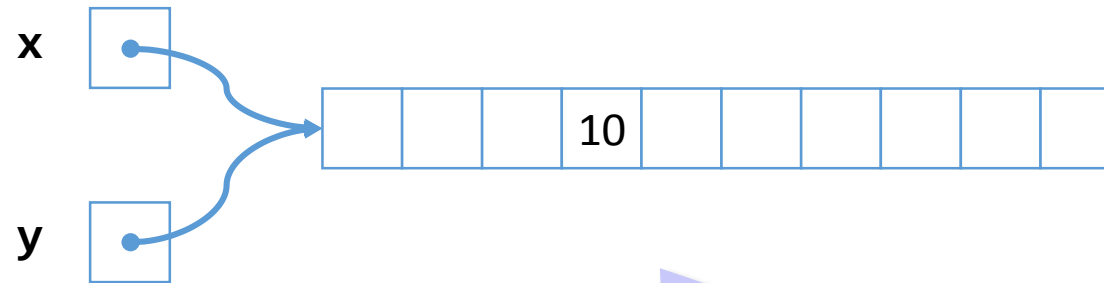
```
Scanner scanner = new Scanner(System.in);
System.out.print("Wie viele Tage? ");
int tage = scanner.nextInt();
int werte[] = new int[tage];
int sum = 0;
for (int i = 0; i < tage; ++i) {
    System.out.print("Regenmenge Tag "+i+ "? ");
    werte[i] = scanner.nextInt();
    sum += werte[i];
}
scanner.close();
```

Regenmengen: Berechnung und Ausgabe

```
int ausnahmen = 0;
double mean = (double)sum / tage;
for (int i= 0; i< tage; ++i) {
    if (werte[i] > 2 * mean)
        ausnahmen++;
}
System.out.println("Mittelwert: " + mean);
System.out.println("Ausnahmeereignisse:" + ausnahmen);
```

Arrays sind Zeiger

```
int [] x = new int[10];  
int [] y;  
y = x;           // was passiert hier?  
y[3] = 10;       // und hier?
```



Wann gilt also `y == x` ?

Arrays sind Zeiger auf Speicherobjekte:
Vorsicht bzgl. Kopiersemantik ("Aliaseffekt")
und beim Vergleich zweier Array-Variablen.

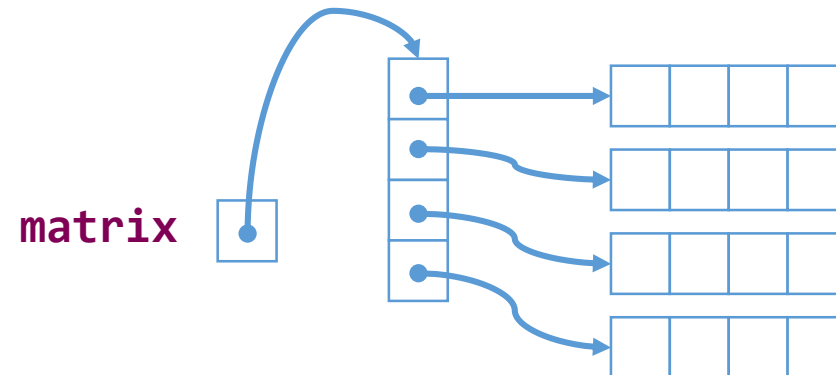


Mehrdimensionale Arrays

```
float [][] matrix = new float [4][4];  
for (int r=0; r < matrix.length; ++r)  
    for (int c=0; c < matrix[r].length; ++c)  
    {  
        if (r==c) matrix[r][c] = 1;  
        else matrix[r][c] = 0;  
    }
```

matrix[x] ist ein float[] somit geht auch
matrix[x] = new float[y];

Eine mehrdimensionales Array
ist also ein Array von Zeigern!



Strings

String: Ein Objekt, welches eine Zeichenkette speichert.

- String kann auch ohne new erzeugt werden:

```
String name = "text";  
String name = expression;
```

- Beispiel:

```
String name = "Informatik II";  
int x = 3;  
int y = 5;  
String point = "(" + x + ", " + y + ")";
```

Indizes

Zeichen in einem String sind auch mit 0-basierten Indizes erreichbar

```
String name = "Informatik";
```

index	0	1	2	3	4	5	6	7	8	9
char	I	n	f	o	r	m	a	t	i	k

- Index des ersten Zeichens : 0
- Index des letzten Zeichens: Länge der Zeichenkette minus 1
- Jedes Zeichen hat Typ `char`

Zeichenketten vergleichen

Relationale Operatoren wie `==` und `!=` funktionieren auf Strings nicht wie erwartet

```
Scanner scanner = new Scanner(System.in);  
String n1 = scanner.next();  
String n2 = scanner.next();  
if (n1 == n2)  
    System.out.println(n1 + "==" + n2);  
else if (n1 != n2)  
    System.out.println(n1 + "!=" + n2);  
}
```

Eingabe:

Info
Info

Ausgabe:

Info != Info

String Methoden

Stringmethoden werden mit der Punkt-Notation aufgerufen

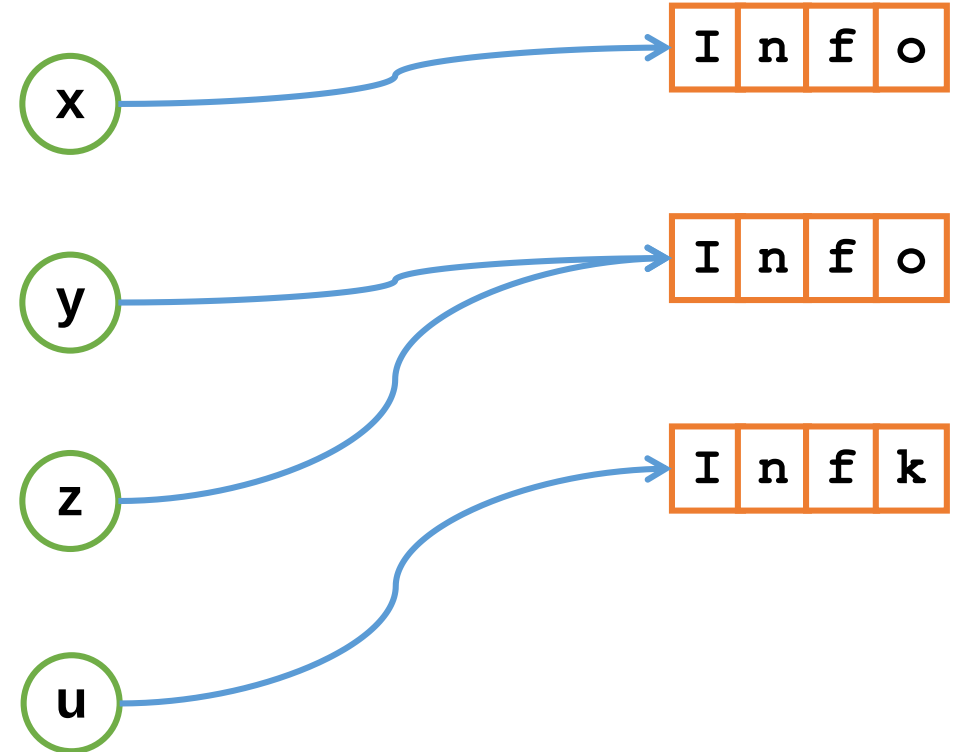
```
String super = "Info II";  
System.out.println(super.length());    // 7
```


Zeichenkettenvergleiche

`x == x` → true
`x == y` ⚠ → false
`y == z` → true
`x == u` → false

`x.equals(y)` → true
`y.equals(z)` → true
`x.equals(u)` → false

Variablen sind Referenzen auf Werte im Speicher



Auszug aus der Java API

compareTo

public int compareTo(String anotherString)
Compares two strings lexicographically.

Parameters:

anotherString - the String to be compared.

Returns:

The value 0 if the argument string is equal to this string; a value less than 0 if this string is lexicographically less than the string argument; and a value greater than 0 if this string is lexicographically greater than the string argument.

Pass-By-Value ()

In Java gibt es **nur pass-by-value***. Das bedeutet, jeder Parameter der übergeben wird, wird grundsätzlich kopiert!

Man findet z.B. beim Suchen im Internet beliebig viele Leute, die das Gegenteil behaupten. Glauben Sie denen **nicht**!

Die Tatsache, dass man den Inhalt eines Objektes oder eines Arrays in einer aufgerufenen Methode ändern kann, hat nichts mit einer pass-by-reference Semantik zu tun! Das liegt lediglich daran, dass Objekte und Arrays als Referenzen repräsentiert sind und somit die Kopie einer Referenz übergeben wird.

Pass by Value

```
static void tryswap(int x, int y) {  
    System.out.println("2: x=" + x + ", y=" + y);  
    int temp = x;  
    x = y;  
    y = temp;  
    System.out.println("3: x=" + x + ", y=" + y);  
}
```

```
public static void main(String[] args) {  
    int i = 1;  
    int j = 2;  
    System.out.println("1: i=" + i + ", j=" + j);  
    tryswap(i,j);  
    System.out.println("4: i=" + i + ", j=" + j);  
}
```

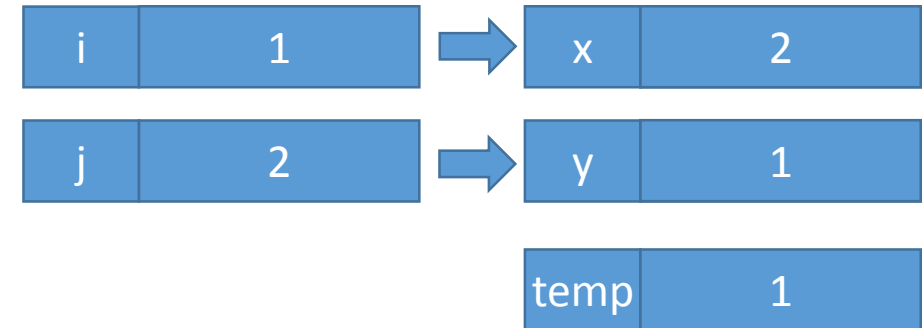
Ausgabe:

1: i=1, j=2

2: x=1, y=2

3: x=2, y=1

4: i=1, j=2



Pass by Value

```
static void tryRename(String x) {  
    System.out.println("2: x= " + x);  
    x = "Niklas";  
    System.out.println("3: x= " + x);  
}  
  
public static void main(String[] args){  
    String name = "Hannes";  
    System.out.println("1: name= " + name);  
    tryRename(name);  
    System.out.println("4: name= " + name);  
}
```

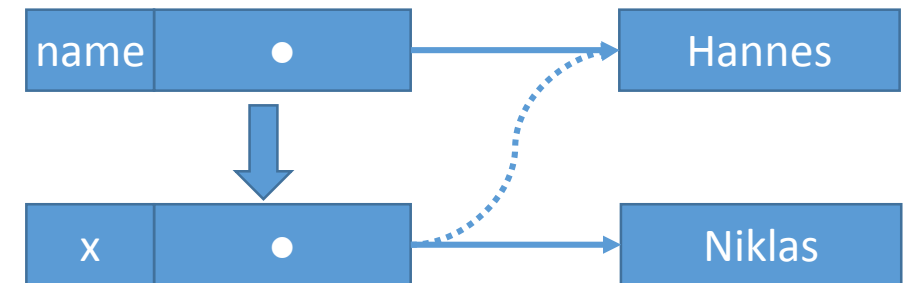
Ausgabe:

1: name= Hannes

2: x= Hannes

3: x= Niklas

4: name= Hannes



From the Java API

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
```

```
String str = new String(data);
```

Pass by Value

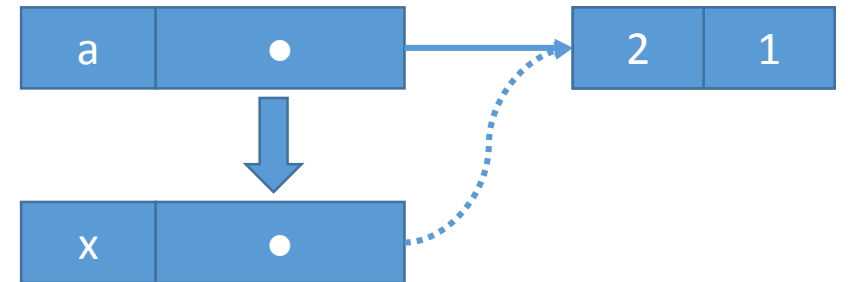
```
static void modify(int[] x) {  
    if (x.length != 2) throw new RuntimeException();  
    int temp = x[0];  
    x[0] = x[1];  
    x[1] = temp;  
}
```

```
public static void main(String[] args) {  
    int[] a = new int[2];  
    a[0] = 1;  
    a[1] = 2;  
    System.out.println("1: a[0] = " + a[0]  
                        + ", a[1] = " + a[1]);  
    modify(a);  
    System.out.println("2: a[0] = " + a[0]  
                        + ", a[1] = " + a[1]);  
}
```

Ausgabe:

1: a[0] = 1, a[1] = 2

2: a[0] = 2, a[1] = 1



Wozu `String[] args` in `main`?

Aufruf eines Programmes mit Parametern

```
java Programmname [Parameter]
```

Whitespace getrennte Parameter der `main` Methode werden als Array von Strings in der Variable `args` übergeben.

```
public class Parameter{  
    public static void main(String[] args) {  
        for (int i = 0; i<args.length; ++i)  
            System.out.println("arg[" + i + "]= " + args[i]);  
    }  
}
```


Wozu **String[] args** in **main**?

Eingabe `java Parameter a13 b c=1`

Ausgabe: `arg[0]=a13`
 `arg[1]=b`
 `arg[2]=c=1`

Vor allem praktisch an der Kommandozeile.

Übergabe von Parametern in Eclipse umständlich → **wir verwenden das nicht weiter in diesem Kurs.**

System.in, System.out, System.error

Beim Aufruf eines Programmes* werden ein Ausgabe-, ein Fehlerausgabe- und ein Eingabestrom instanziiert.

System.out, System.err

Ausgabe und Fehlerausgabe unterscheiden sich oft nur marginal

- Typische Verwendung `System.out.print()`

Eingabestrom

System.in

liest typischerweise Zeichen von der Tastatur und liefert Buchstaben.

- Typische Verwendung über `java.util.Scanner`

Wiederholung Zufallszahlen: LCG

Methode zum Ziehen einer Zufallszahl $L \in \{0, \dots, m - 1\}$

mit $\mathbb{P}(L = x) = \frac{1}{m}$ für jedes $x \in \{0, \dots, m - 1\}$

```
static final long m = 2147483647;  
static final long b = 12345;  
static final long a = 1103515245;  
static long u = System.currentTimeMillis();  
u = (u * a + b) % m;
```

Wiederholung Zufallszahlen: Uniform

Uniform: Methode zum Ziehen einer Zahl $U \in [0,1)$ mit $\mathbb{P}(U \in [l, r)) = r - l$ for $0 \leq l < r < 1$

Von LCG zu Uniform $\{0, \dots, m - 1\} \rightarrow [0,1): L \mapsto L/m$

```
public static double Uniform(){  
    u = (u * a + b) % m;  
    return(double)u/m;  
}
```

Wiederholung Zufallszahlen: fairer Würfel

Uniform(): gibt $U \in [0,1)$ zurück mit

$$\mathbb{P}(U \in [l, r)) = r - l$$

Dice(): soll $Y \in \{1, \dots, 6\}$ zurückgeben, so dass

$$\mathbb{P}(Y = k) = \frac{1}{6}, \text{ für jedes } k \in \{1, \dots, 6\}$$

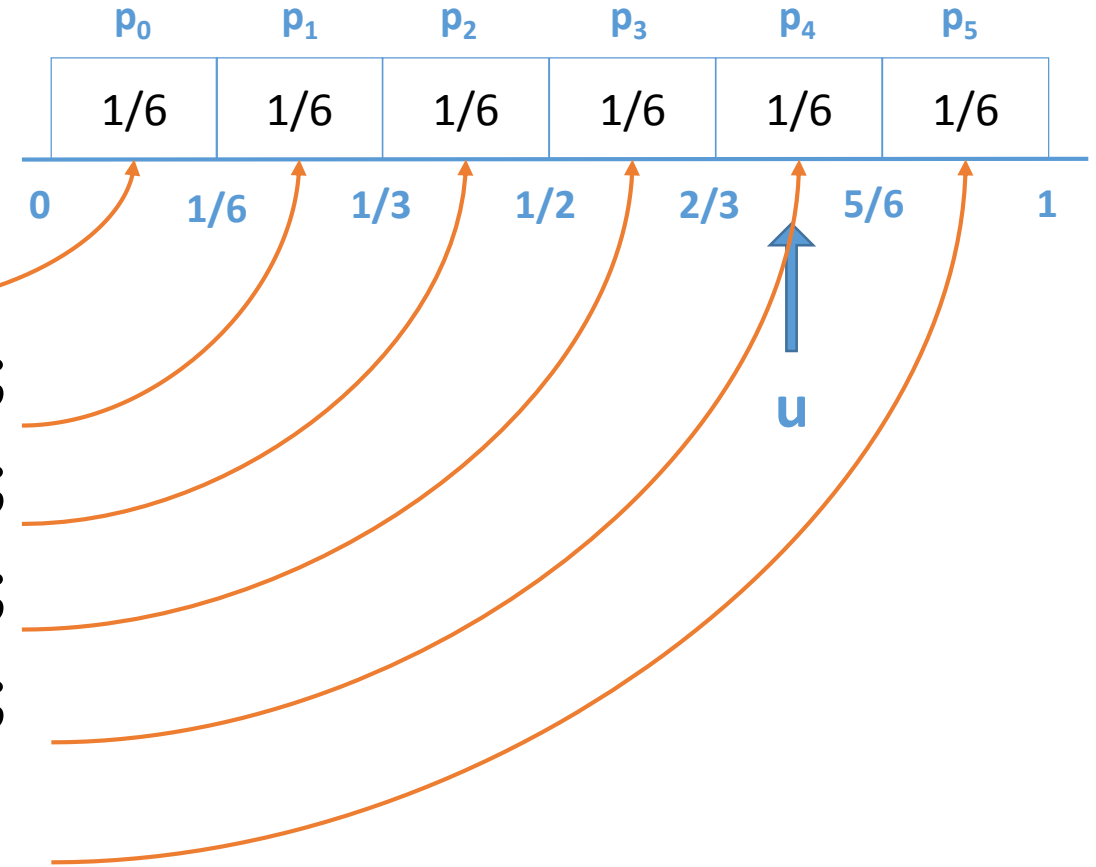


Wie machen wir das?

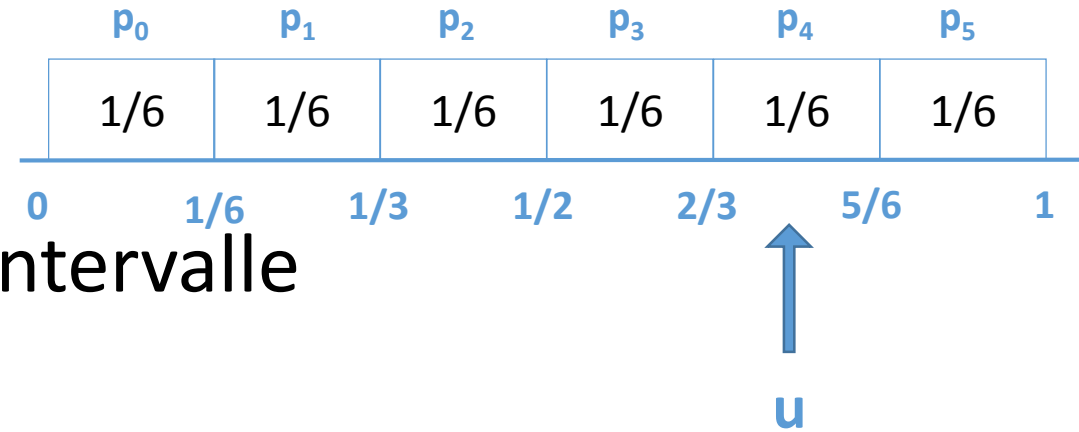
Wir wissen, dass $\mathbb{P}\left(X \in [0, \frac{1}{6})\right) = 1/6$, $\mathbb{P}\left(X \in [\frac{1}{6}, \frac{2}{6})\right) = \frac{1}{6}$, *etc.*

Uniform $\rightarrow$ Dice: erste Idee

```
static int Dice(){  
    double u = Uniform();  
    if (u < 1.0/6) return 1;  
    else if (u < 1.0/3) return 2;  
    else if (u < 1.0/2) return 3;  
    else if (u < 2.0/3) return 4;  
    else if (u < 5.0/6) return 5;  
    else return 6;  
}
```



Uniform $\rightarrow$ Dice kürzer



Beobachtung: Unterteilung in gleiche Intervalle

Diskretisierung wie folgt: Transformiere $[0,1)$ nach $[1,7)$, dann Runden nach unten:

```
static int Dice(){
    double u = Uniform();
    return (int)(u*6+1);
}
```

Werfen eines **unfairen** Würfels

Die lange Lösung von oben war nicht so übel: Angenommen es soll ein unfairen 6-seitiger Würfel simuliert werden mit

$$\mathbb{P}(X = 0) = 0.1$$

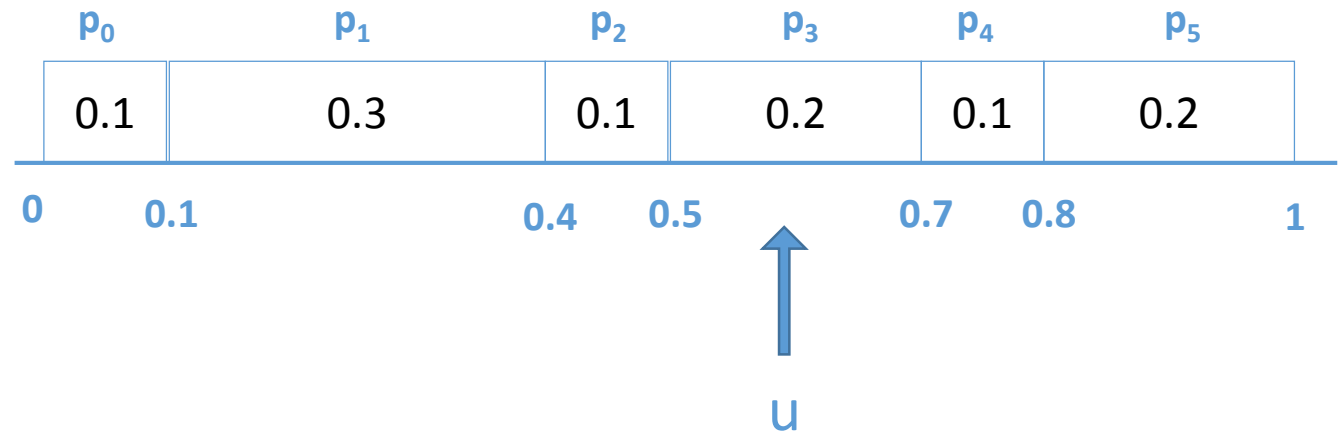
$$\mathbb{P}(X = 1) = 0.3$$

$$\mathbb{P}(X = 2) = 0.1$$

$$\mathbb{P}(X = 3) = 0.2$$

$$\mathbb{P}(X = 4) = 0.1$$

$$\mathbb{P}(X = 5) = 0.2$$



Uniform $\rightarrow$ UnfairDice

```
static int UnfairDice(){  
    double u = Uniform();
```

```
    if (u < 0.1) return 0;
```

```
    else if (u < 0.4) return 1; // 0.4 = 0.1 + 0.3
```

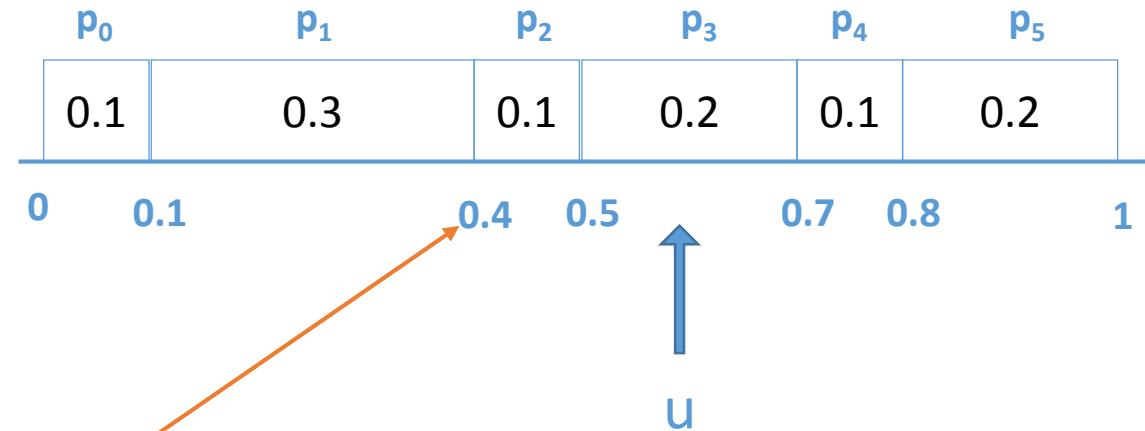
```
    else if (u < 0.5) return 2; // 0.5 = 0.4 + 0.1
```

```
    else if (u < 0.7) return 3; // 0.7 = 0.5 + 0.2
```

```
    else if (u < 0.8) return 4; // 0.8 = 0.7 + 0.1
```

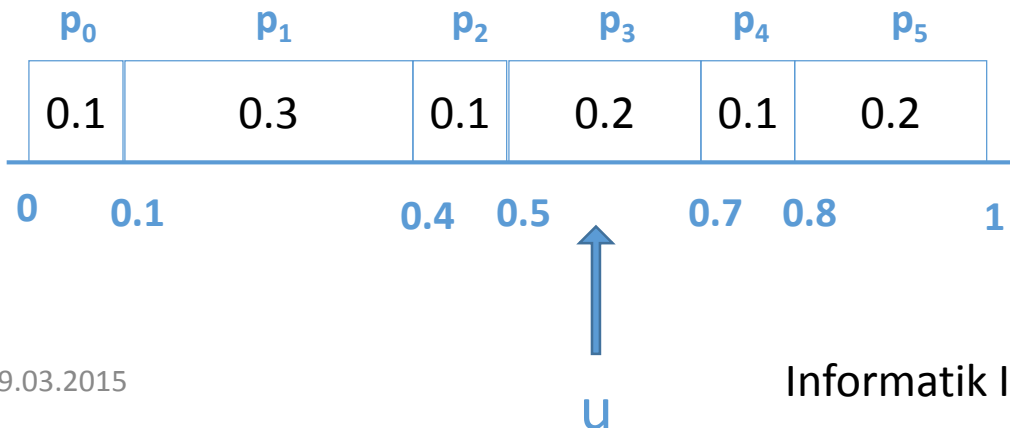
```
    else return 5;           // 1 = 0.8 + 0.2
```

```
}
```



Und nun das ganze generisch:

```
static int UnfairDice(){
    double u = Uniform();
    if (u < 0.1) return 0;
    else if (u < 0.4) return 1;
    else if (u < 0.5) return 2;
    else if (u < 0.7) return 3;
    else if (u < 0.8) return 4
}
```

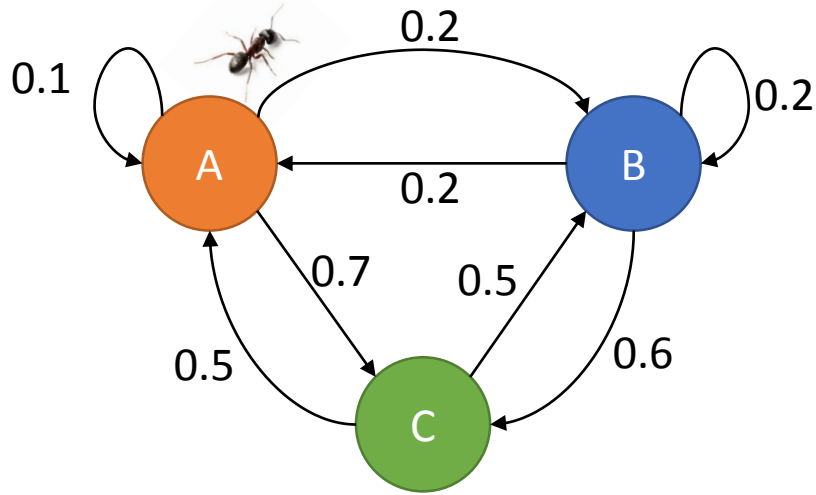


```
double[] p = {0.1,0.3,0.1,0.2,0.1,0.2};
int dice = UnfairDice(p);
```



```
static int UnfairDice(double[] p){
    double u = LCG.Uniform();
    double sum = p[0];
    int res= 0;
    while (res < p.length-1 && sum<u) {
        res++;
        sum += p[res];
    }
    return res;
}
```

Anwendung: Die Reise der Ameise.



Diese Matrix nennt man Übergangsmatrix oder Wahrscheinlichkeitsmatrix.

Ameise startet in Nest A

Mit den angegebenen W'ten geht die Ameise in jedem Schritt zu einem anderen Nest oder bleibt.

In welchem Nest ist sie am häufigsten?

Ziel \ Start	A	B	C
	A	B	C
A	0.1	0.2	0.7
B	0.2	0.2	0.6
C	0.5	0.5	0

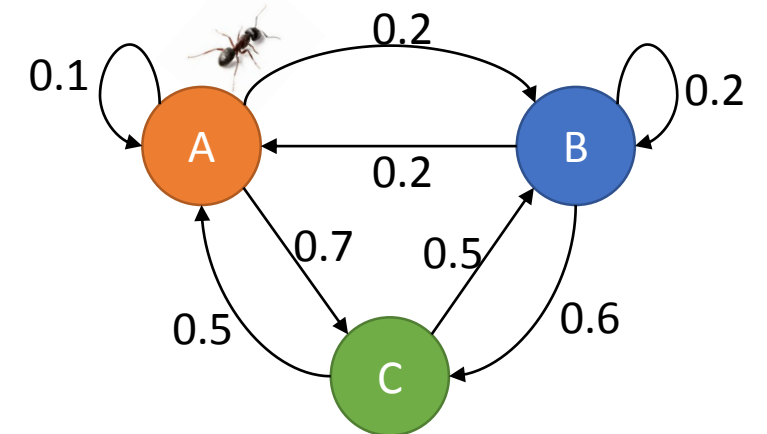
Was sagt Ihre Intuition?

Wie simuliert man das?

```
int[] count = new int[3];  
double[][] P={ {0.1, 0.2, 0.7},  
               {0.2, 0.2, 0.6},  
               {0.5, 0.5, 0} };  
  
int loc = 0; // 0=A, 1=B, 2=C
```

```
for (int i = 0; i<steps; ++i){  
    ++count[loc];  
    loc = UnfairDice(P[loc]);  
}
```

P			
	A	B	C
A	0.1	0.2	0.7
B	0.2	0.2	0.6
C	0.5	0.5	0



Direkte Berechnung statt Simulation

Starte mit $\mu_0 = (1, 0, 0)$; d.h. Ameise ist zu Beginn mit W't 1 bei A.

Ein Schritt

$$\mu_1 = \mu_0 \cdot P = (0.1, 0.2, 0.7);$$

Nächster Schritt

$$\mu_2 = \mu_1 \cdot P = (0.4, 0.41, 0.19);$$

u.s.w.

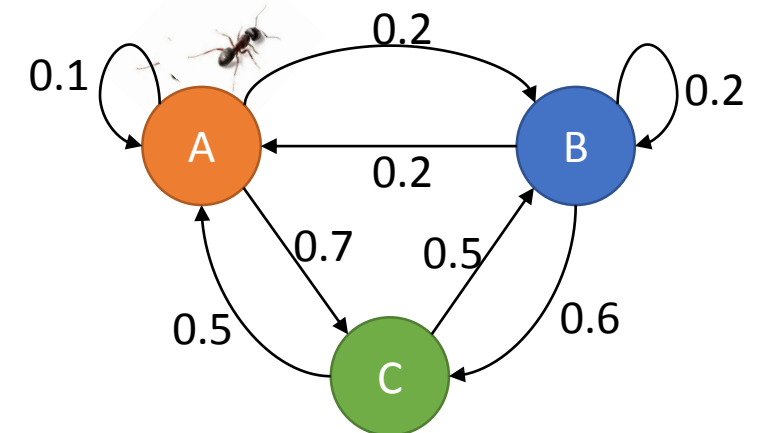
$$\mu_\infty = \mu_0 \cdot P \cdot \dots \cdot P = (0.289 \ 0.318 \ 0.393)$$

W'ten, nach einem Schritt
bei A, B or C zu sein

W'ten, nach zwei Schritten
bei A, B or C zu sein

W'ten **nach sehr vielen
Schritten** bei A, B oder C zu
sein.

P			
	A	B	C
A	0.1	0.2	0.7
B	0.2	0.2	0.6
C	0.5	0.5	0



Etwas Mathematik

Unter gewissen Bedingungen weiss man, dass $\mu_0 P^n$ für $n \rightarrow \infty$ gegen einen W'tsvektor μ_∞ **konvergiert**.

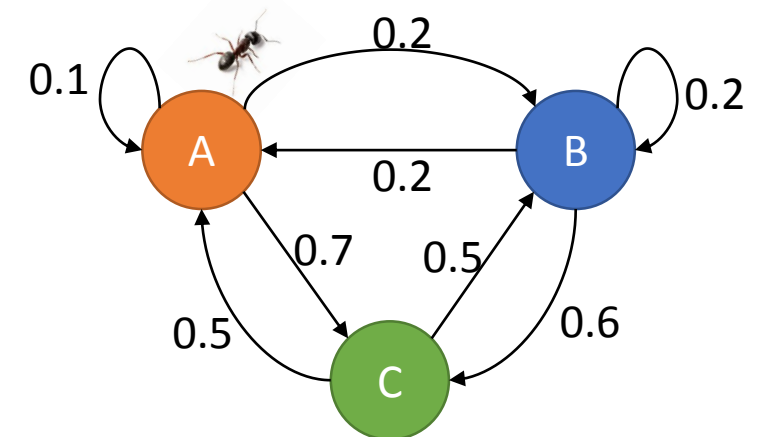
Für μ_∞ gilt: $\mu_\infty = \mu_\infty \cdot P$

Das ist ein **Eigenwertproblem** und μ_∞ ist der Eigenvektor zum Eigenwert 1.

Wir können $\mu_\infty \approx \mu_\infty \cdot P$ benutzen, um **auf Konvergenz zu prüfen**.

Also: Anwendung von $\mu \leftarrow \mu \cdot P$ bis $\mu \approx \mu \cdot P$

P	A	B	C
A	0.1	0.2	0.7
B	0.2	0.2	0.6
C	0.5	0.5	0



Credits

Teile dieser Folien inspiriert durch Vorlesungsfolien zu Parallel Programming 2015 von Otmar Hilliges.