Informatik II Übung, Woche 11

Giuseppe Accaputo

17. März, 2016

Plan für heute

1. Tipps zum Judge
2. Aufbau einer Klasse
   - Instanzmethoden und Instanzvariablen
   - Klassenmethoden und Klassenvariablen (`static`)
3. Vorbesprechung Übung 4
4. Nachbesprechung Übung 3 (inkl. Live-Programmierung)

Tipps zum Judge I: Definition der `Main` Klasse

- Definiert eure `Main` Klasse nur mittels `class` (ohne `public` oder anderen Zugriffsmodifikatoren)

Falsch:

```
public class Main{
    ...
}
```

Richtig:

```
class Main{
    ...
}
```

Tipps zum Judge II: Eigene Package-Definitionen entfernen

- Entfernt eure eigenen Package-Definitionen aus dem Code beim Submitten der Lösung an den Judge

Falsch:

```
package aufgabe_03_gaussian;

class Main{...
```

Richtig:

```
class Main{...
```

Aufbau einer Klasse

Klasse `MyClass`
Klassenmethoden: Zugriff ohne Instanz möglich
Klassenvariablen: Zugriff ohne Instanz möglich
Instanzmethoden: Zugriff nur mittels Instanz möglich
Instanzvariablen: Zugriff nur mittels Instanz möglich

Aufbau einer Klasse: Beispiel `MyClass`

```
class MyClass {
    // Klassenvariable
    static int s = 0;

    // Klassenmethode
    static void setS(int new_s){ s = new_s;
        ↪ }

    // Instanzvariable (ohne static)
    int i = 0;

    // Instanzmethode (ohne static)
    void setI(int new_i){ i = new_i; }
}
```

Instanzen einer Klasse

```
public static void main(String[] args) {
    MyClass a = new MyClass();
    MyClass b = new MyClass();
    MyClass c = new MyClass();
}
```

- `a`,`b`, und `c` sind *Instanzen der Klasse* `MyClass` und werden mittels `new`-Operator *instanziert*

`MyClass` Klassenmethoden		
`MyClass` Klassenvariablen		
a	b	c
Instanzmeth.	Instanzmeth.	Instanzmeth.
Instanzvar.	Instanzvar.	Instanzvar.

Beispiel: Instanzmethoden und Instanzvariablen

```
MyClass a = new MyClass();
a.i = 100;
MyClass b = new MyClass();
b.i = 300;
```

- Ändert eine Instanz den Wert der eigenen Instanzvariable, so ist die Änderung nur für die Instanz geltend, d.h. `a.i = 100` hat keinen Einfluss auf `b.i`

`setS(int new_s)`	
`s = 0`	
`a`	`b`
`setI(int new_i)`	`setI(int new_i)`
`i = 100`	`i = 300`

Beispiel: Klassenmethoden und Klassenvariablen

```
// Fortsetzung Beispiel von vorherigem Slide
MyClass.s = 100;
MyClass.setS(200);
b.s = 300;
a.setS(190)
```

- Wird der Wert einer Klassenvariable verändert, so ist die Änderung für alle anderen Instanzen sichtbar und geltend

setS(int new_s)	
s = 190	
a	b
setI(int new_i)	setI(int new_i)
i = 100	i = 300

Zugriffsregeln zwischen Variablen und Methoden einer Klasse

1. Instanzmethoden können auf Instanzmethoden und Instanzvariablen direkt zugreiffen
2. Instanzmethoden können auf Klassenmethoden und Klassenvariablen direkt zugreiffen
3. Klassenmethoden können auf Klassenmethoden und Klassenvariablen direkt zugreiffen
4. Klassenmethoden *können nicht* direkt auf Instanzmethoden und Instanzvariablen zugreiffen

Beispiel: Programmausführung

```
MyClass c = new MyClass();
MyClass d = new MyClass();
MyClass e = new MyClass();

c.s = 100;
d.s += 200;
d.i = 100;
c.i += 200;
e.setS(1);

System.out.println(d.s);
System.out.println(d.i);
```

- Frage: Was wird auf der Konsole ausgegeben?

Beispiel: Programmausführung

```
MyClass c = new MyClass();
MyClass d = new MyClass();
MyClass e = new MyClass();

c.s = 100;
d.s += 200;
d.i = 100;
c.i += 200;
e.setS(1);

System.out.println(d.s);
System.out.println(d.i);
```

- Antwort: `d.s = 1, d.i = 100`

Übung 4, Aufgabe 1: Selfgrowing Array

- Implementation eines Arrays, welches sich automatisch vergrössert
- Wenn der Benutzer ein Element ausserhalb des aktuellen Arrays hinzufügen möchte, dann wird Array auf die nächste Zweierpotenz vergrössert
  - Beispiel: User verlangt Element $931 \implies$ Array wird auf Grösse $2^{10} = 1024$ vergrössert
- Array kopieren: kann mittels Referenz gemacht werden (ohne explizite `for`-Schleife), i.e., `data = tmp` (Java Garbagecollector sei Dank!)

Übung 4, Aufgabe 2: Sorted NameList

- Implementation der Klasse `NameList`, welche eine sortierte Liste von Namen enthält
- Arbeitet zuerst mit Papier und Stift bevor ihr an der konkreten Implementation arbeitet

Übung 3, Aufgabe 2: Hashfunktionen

Live-Programmierung

Übung 3, Aufgabe 2: Hashfunktionen

Die Funktion `ComputeHash(String s)` berechnet den Hash eines Strings `s` basierend auf der folgenden Formel,

$$\texttt{sum} = \sum_{\texttt{i=0}}^{\texttt{s.length - 1}} \left(\texttt{s[i]} \cdot b^{(i+1)} \right) \quad , \tag{1}$$

und gibt dabei den berechneten Hash als `int` zurück. Wir machen dabei folgende Beobachtungen (für $b = 31$):

1. `ComputeHash('TestG')` $= 2\ 143\ 323\ 787 \implies$ ✓
2. `ComputeHash('TestH')` $= -2\ 123\ 014\ 358 \implies$ ✗

Übung 3, Aufgabe 2: Überläufe I

Problem: `ComputeHash('TestH')` liefert einen grösseren Wert als der grösst-möglich darstellbare `int` Wert $\Longrightarrow$ Überlauf! ϟ

- $\max(\texttt{int}) = 2^{31} - 1 = 2\ 147\ 483\ 647$
- `Compu...('TestG')`$= 2\ 143\ 323\ 787 < \max(\texttt{int}) \Longrightarrow$ ✓
- `Compu...('TestH')`$= 2\ 171\ 952\ 938 > \max(\texttt{int}) \Longrightarrow$ ✗

Übung 3, Aufgabe 2: Überläufe II

Die Bit-Darstellung zeigt das Problem genauer auf:

- $-2\ 123\ 014\ 358 = 10000001011101010101111100101010$
- Die gleiche Bit-Darstellung in `long` (64 Bits) liefert $10000001011101010101111100101010 = 2\ 171\ 952\ 938$
- Da ein `int` 32-Bit lang ist, agiert das vorderste Bit als Vorzeichenbit, daher erhalten wir

$$
\begin{aligned}
&10000001011101010101111100101010 = \\
&-2^{31} + 2^{24} + 2^{22} + 2^{21} + 2^{20} + 2^{18} + 2^{16} \\
&+ 2^{14} + 2^{12} + 2^{11} + 2^{10} + 2^{9} + 2^{8} + 2^{5} + 2^{3} + 2^{1} \\
&= \underline{\underline{-2\ 123\ 014\ 358}}
\end{aligned}
$$

Übung 3, Aufgabe 2: Lösung mittels Modulo I

Verwende den Modulo-Operator um Überläufe bei der Berechnung der Summe

$$\texttt{s[0]} \cdot b^1 + \texttt{s[1]} \cdot b^2 + \texttt{s[2]} \cdot b^3 \ldots \tag{2}$$

zu vermeiden. Dabei sind folgende Rechenregeln zu beachten:

$$(a + b) \mod c = ((a \mod c) + (b \mod c)) \mod c \tag{3}$$
$$(a \cdot b) \mod c = ((a \mod c) \cdot (b \mod c)) \mod c \tag{4}$$

Übung 3, Aufgabe 2: Lösung mittels Modulo II

Mit Hilfe der Rechenregeln in (3) und (4) lässt sich nun die folgende Summe

$$(\texttt{s[0]} \cdot b^1 + \texttt{s[1]} \cdot b^2 + \texttt{s[2]} \cdot b^3 \dots) \mod m \tag{5}$$

mittels folgendem Algorithmus ohne Überläufe berechnen:

```
int B = b;
int S= 0;
for (int i = 0; i <s.length(); ++i){
    S = (S + B * s.charAt(i)) % m;
    B = (B * b) % m;
}
return S;
```