

Programmieren II

Lukas Abelt

lukas.abelt@airbus.com

DHBW Ravensburg
Wirtschaftsinformatik

Ravensburg
17. März 2019

Inhalt

1 Über mich

2 Vorlesung

3 Datentypen

Inhalt

1 Über mich

2 Vorlesung

3 Datentypen

Allgemeines

Wer bin ich?

- Lukas Abelt

Allgemeines

Wer bin ich?

- Lukas Abelt
- 21 Jahre (Jahrgang '97)

Allgemeines

Wer bin ich?

- Lukas Abelt
- 21 Jahre (Jahrgang '97)
- Ursprünglich aus Werder (Havel)

Allgemeines

Wer bin ich?

- Lukas Abelt
- 21 Jahre (Jahrgang '97)
- Ursprünglich aus Werder (Havel)
 - ...in Brandenburg

Allgemeines

Wer bin ich?

- Lukas Abelt
- 21 Jahre (Jahrgang '97)
- Ursprünglich aus Werder (Havel)
 - ...in Brandenburg
 - ...bei Potsdam

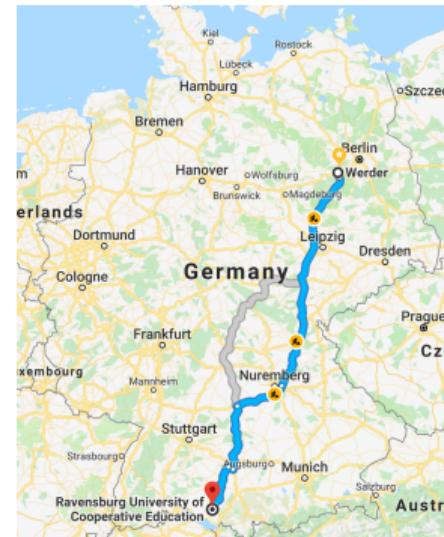
Allgemeines

Wer bin ich?

- Lukas Abelt
- 21 Jahre (Jahrgang '97)
- Ursprünglich aus Werder (Havel)
 - ▣ ...in Brandenburg
 - ▣ ...bei Potsdam
 - ...bei Berlin

Allgemeines

Wo komme ich her?



Bildquelle: https://upload.wikimedia.org/wikipedia/commons/4/47/Werder_an_der_Havel.jpg (Abgerufen: 26.02.2019)

Beruflicher&Akademischer Werdegang

- Juli 2015 - Abitur
- Ab September 2015 - Duales Studium
 - Hochschule: DHBW Ravensburg **Campus Friedrichshafen**
 - Studiengang: Informationstechnik (Mobile Informatik)
 - Firma: Airbus Defence and Space (Immenstaad)
- September 2018 - Bachelorarbeit und -abschluss
- Seit Oktober 2018 - Software Architect bei Airbus

Was habe ich bisher gemacht?

Praxisphasen

- Arbeit im Bereich SIGINT(Signal Intelligence)
- Implementierung des TCP Stacks zur Übertragung von Signaldaten(C++, Matlab, Simulink)
- Später Abteilungswechsel zu Simulationssoftware
- Implementierung eines neuen Schadenmodells in das bestehende System (C++)
- **Analyse und Bewertung neuer Methoden zur Durchführung simulationsgestützter Parameterstudien mit cloud-basierten Systemen (Bachelorarbeitsthema)**

Was habe ich bisher gemacht?

Theoriephasen

- Mathetutorium für untere Semester

Was habe ich bisher gemacht?

Theoriephasen

- Mathetutorium für untere Semester
- Studienarbeit: Entwickeln eines selbstlernenden Chatbots (Tensorflow, Python)

Was habe ich bisher gemacht?

Theoriephasen

- Mathetutorium für untere Semester
- Studienarbeit: Entwickeln eines selbstlernenden Chatbots (Tensorflow, Python)
 - Mit fraglichen Erfolgen

Was habe ich bisher gemacht?

Theoriephasen

- Mathetutorium für untere Semester
- Studienarbeit: Entwickeln eines selbstlernenden Chatbots (Tensorflow, Python)
 - Mit fraglichen Erfolgen
- Dreimaliger Teilnehmer des Bierathlons Friedrichshafen

Was habe ich bisher gemacht?

Theoriephasen

- Mathetutorium für untere Semester
- Studienarbeit: Entwickeln eines selbstlernenden Chatbots (Tensorflow, Python)
 - Mit fraglichen Erfolgen
- Dreimaliger Teilnehmer des Bierathlons Friedrichshafen
 - 3. Platz

Was habe ich bisher gemacht?

Theoriephasen

- Mathetutorium für untere Semester
- Studienarbeit: Entwickeln eines selbstlernenden Chatbots (Tensorflow, Python)
 - Mit fraglichen Erfolgen
- Dreimaliger Teilnehmer des Bierathlons Friedrichshafen
 - 3. Platz
- Organisator des alljährlichen Glühweingrillens Friedrichshafen (Nächster Termin: 27. April 2019)

Was habe ich bisher gemacht?

Studienarbeit

Olaf's erste Worte

„they're just fucking fucking
fucking not just fucking with“

Inhalt

1 Über mich

2 Vorlesung

3 Datentypen

Allgemeines

Skript

- Mit \LaTeX erstellt
- Im Druck verfügbar...
 - ...wenn man bezahlt hat
- Digitale Version als PDF verfügbar
- Source Code zum selbst compilieren auf GitHub verfügbar:
 - https://github.com/LuAbelt/WI18_ProgrammierenII

Git repository clonen

```
git clone https://github.com/LuAbelt/WI18_ProgrammierenII
```

Allgemeines

Skript

Fork me on GitHub



Allgemeines

Organisation

- Insgesamt 60 UE über 15 Termine
- Also 4 UE pro Termin
- Aufteilung (Vorschlag)
 - 2 UE Theorie
 - Kaffeepause
 - 2 UE praktische Anwendung

Allgemeines

Organisation

- Zu (fast) jedem Termin wird es eine praktische Aufgabe zur Implementierung geben (Für den zweiten Vorlesungsteil)
- Aufgaben & meine Beispielimplementierung: Nach der Vorlesung im Git Repo zu finden
- Beispielimplementierung wird immer in Java sein
- Hinweise:
 - Ich empfehle IntelliJ als IDE (Für Studenten kostenlose Pro Version)
 - DHBW Rechner haben (leider) nur Eclipse
 - Im Git wird es Projektdateien für bei IDE's geben

Allgemeines

Feedback

- Auch ich bin nicht unfehlbar
 - Hauptsächlich in C++ unterwegs
 - Dadurch ggf. Ungenauigkeiten und Fehler bei Java spezifischen Aspekten
- Fragen gerne immer und sofort
- ...Gleiches gilt für (themenbezogene) Diskussionen
- Feedback gerne über alle Kanäle wie zum Beispiel:
 - Persönlich
 - Per Mail
 - Über GitHub/GitLab
 - usw...
- Kontaktdaten am Ende jedes Foliensatzes

Ziele der Vorlesung I

Laut Modulbeschreibung

Fachkompetenz

Die Studierenden kennen fortgeschrittene Konzepte objektorientierter Programmiersprachen. Sie besitzen Kenntnisse über wichtige Algorithmen und Datenstrukturen sowie Methoden zur Beurteilung der Effizienz und Qualität von Algorithmen.

Ziele der Vorlesung II

Laut Modulbeschreibung

Methodenkompetenz

Die Studierenden können fortgeschrittene Konzepte der Objektorientierung anwenden und autonom mittlere bis größere lauffähige Programme implementieren und testen. Sie sind in der Lage, Algorithmen verschiedener Darstellungsarten zu verstehen und ihre Effizienz bzw. Qualität zu beurteilen, aber auch selbstständig Algorithmen und dazu erforderliche Datenstrukturen zu entwickeln und zu implementieren.

Ziele der Vorlesung III

Laut Modulbeschreibung

Personale und Soziale Kompetenz

Die Studierenden können eigenständig Algorithmen und Lösungsverfahren erarbeiten. Sie können stichhaltig und sachangemessen über Konzepte und eigene Algorithmen und deren Implementierung und die damit verbundenen Probleme argumentieren, eigene Umsetzungen plausibel darstellen und eventuelle Fehler nachvollziehbar gegenüber anderen begründen.

Ziele der Vorlesung IV

Laut Modulbeschreibung

Übergreifende Handlungskompetenz

Die Studierenden können unter Einsatz der Programmiersprache komplexe praktische Probleme modellieren, algorithmisch behandeln und in anwenderfreundliche und effiziente Lösungen umsetzen. Sie können praktische Problemstellungen analysieren und bekannte Algorithmen und Datenstrukturen effizienzorientiert darauf anwenden und falls notwendig an die Problemstellung anpassen.

Ziele der Vorlesung V

In meinen Worten

Kurzgesagt

Am Ende der Vorlesung sollt ihr mit den vorgestellten Konzepten der fortgeschrittenen Objektorientierung vertraut sein. Dies beinhaltet unter anderem das theoretische Verständnis der zugrundeliegenden Konzepte, sowie die **sprachunabhängige** Anwendung des gelernten. Ziel ist **nicht** das lernen von Java, sondern das übergreifende Verständnis, sodass das gelernte (theoretisch) in jeder Sprache angewandt werden kann!

Termine & Themen

- 01.04.2019, 14:00-17:15:
 - Vorstellung
 - Allgemeine Informationen zur Vorlesung
 - Datentypen
- 03.04.2019, 14:00-17:15:
 - Generische Interfaces & Klassen
 - Nutzung der Klassenbibliothek
- 08.04.2019, 14:00-17:15:
 - Algorithmen Teil 1
 - Beschreibung
 - Analyse

Termine & Themen

□ 10.04.2019, 14:00-17:15:

- Listenstrukturen Teil 1
 - Grundoperationen
 - Arrays
 - Verkettete Listen

□ 24.04.2019, 14:00-17:15:

- Listenstrukturen Teil 2
 - Stacks
 - Queues
 - Bäume

Termine & Themen

□ 25.04.2019, 14:00-17:15:

- Abstrakte Datentypen

- Collections
- Iteratoren

□ 29.04.2019, 14:00-17:15:

- Algorithmen Teil 2

- Suchverfahren

□ 02.05.2019, 14:00-17:15:

- Algorithmen Teil 3

- Sortierverfahren

Termine & Themen

- 06.05.2019, 14:00-17:15:
 - Algorithmen Teil 4
 - Divide&Conquer
 - Backtracking
- 08.05.2019, 14:00-17:15:
 - Fortgeschrittene Konzepte Teil 1
 - Parallelisierung
 - Synchronisationskonzepte
- 13.05.2019, 14:00-17:15:
 - Fortgeschrittene Konzepte Teil 2
 - Synchronisationskonzepte
 - Ein- und Ausgabe über Streams

Termine & Themen

- 15.05.2019, 14:00-17:15:
 - Aufbau grafischer Oberflächen Teil 1
 - Layout
 - Typische Komponenten
- 20.05.2019, 14:00-17:15:
 - Aufbau grafischer Oberflächen Teil 2
 - Typische Komponenten
 - Ereignisbehandlung
- 22.05.2019 & 27.05.2019:
 - Puffertermin

Puffertermine

- Geplante Themen sind nach vorraussichtlich 13 Terminen abgearbeitet
- Sofern es nicht zu Verzögerungen kommt, könnte man diese nutzen für:
 - Wiederholungen
 - Exkurse zu anderen Themen der SW-Entwicklung
 - Freies arbeiten für die Prüfungsleistung
- Gestaltung der Puffertermine frei nach euren Interessen

Prüfungsleistung

- Keine Klausur, denn:
 - Schon 6 Klausuren dieses Semester
 - Rein theoretische Überprüfungen für Programmieren sowieso eher fraglich
- Stattdessen: **Portfolioprüfung**
 - Besteht aus mehreren Teilleistungen (Hier: 3)
 - Teil 1: Kurztest
 - Teil 2&3: Programmentwurf inklusive Dokumentation

Prüfungsleistung

Kurztest

Teil 1: Kurztest

- Zu Themenblöcken Algorithmen und Datentypen
- Bearbeitungszeit: 30 Minuten
- Noch kein fester Termin gesetzt
- Gewichtung an der Gesamtnote: **20%**

Prüfungsleistung

Programmentwurf

Teil 2: Programmentwurf

- Wird Teilespekte aus (fast) allen Themenblöcken enthalten
 - Vermutlich Schwerpunkt auf Datenstrukturen
 - Vermutlich keine (oder wenig) Teile aus grafischen Overflächen
- Zu bearbeiten in Gruppen von 3-4 Personen
- Abgabetermin: **15.07.2019**
- Umsetzung empfohlen in Java
 - Jedoch nicht darauf beschränkt!
 - Bei Interesse Details bei mir erfragen
- Gewichtung an der Gesamtnote: **50%**

Prüfungsleistung

Dokumentation

Teil 3: Dokumentation

- Dokumentation der im Programmentwurf implementierten Lösung
- Bearbeitung parallel zum Programmentwurf mit gleichem Termin
- Gleiche Gruppen wie im Programmentwurf
- Gewichtung an der Gesamtnote: **30%**

Inhalt

1 Über mich

2 Vorlesung

3 Datentypen

Datentypen

Unterscheidung

- Primitive Datentypen
- Objektorientierte Datentypen
- Strukturierte Datentypen

Primitive Datentypen

Beispiele

- Beispiele:
 - int
 - boolean
 - char
 - float
 - double
 - long
 - byte

Primitive Datentypen

Eigenschaften

- Rein zum Speichern von Daten
- Feste Speichergröße
- Feste Präzision \Rightarrow Diskret
- Fest definierte Ober- und Untergrenze
- Implementieren selbst keine Algorithmen, sondern nur simple Operationen
- Menge an Operationen beschränkt ($+, -, *, /$ und ggf. Bitwise Operations)
 - Und nicht erweiterbar (Vgl. Objektorientierte Datentypen)

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte			
char			
short			
int			
long			
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8		
char			
short			
int			
long			
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	
char			
short			
int			
long			
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char			
short			
int			
long			
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16		
short			
int			
long			
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	
short			
int			
long			
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short			
int			
long			
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16		
int			
long			
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	
int			
long			
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int			
long			
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int	32		
long			
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int	32	-2^{31}	
long			
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int	32	-2^{31}	$2^{31} - 1$
long			
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int	32	-2^{31}	$2^{31} - 1$
long	64		
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int	32	-2^{31}	$2^{31} - 1$
long	64	-2^{63}	
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int	32	-2^{31}	$2^{31} - 1$
long	64	-2^{63}	$2^{63} - 1$
float			
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int	32	-2^{31}	$2^{31} - 1$
long	64	-2^{63}	$2^{63} - 1$
float	32		
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int	32	-2^{31}	$2^{31} - 1$
long	64	-2^{63}	$2^{63} - 1$
float	32	$\pm 1.4E-45$	
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int	32	-2^{31}	$2^{31} - 1$
long	64	-2^{63}	$2^{63} - 1$
float	32	$\pm 1.4E-45$	$\pm 3.4E+38$
double			
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int	32	-2^{31}	$2^{31} - 1$
long	64	-2^{63}	$2^{63} - 1$
float	32	$\pm 1.4E-45$	$\pm 3.4E+38$
double	64		
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int	32	-2^{31}	$2^{31} - 1$
long	64	-2^{63}	$2^{63} - 1$
float	32	$\pm 1.4E-45$	$\pm 3.4E+38$
double	64	$\pm 4.9E-324$	
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int	32	-2^{31}	$2^{31} - 1$
long	64	-2^{63}	$2^{63} - 1$
float	32	$\pm 1.4E-45$	$\pm 3.4E+38$
double	64	$\pm 4.9E-324$	$\pm 1.7E+324$
boolean			

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int	32	-2^{31}	$2^{31} - 1$
long	64	-2^{63}	$2^{63} - 1$
float	32	$\pm 1.4E-45$	$\pm 3.4E+38$
double	64	$\pm 4.9E-324$	$\pm 1.7E+324$
boolean	Undefiniert		

Primitive Datentypen

Eigenschaften (In Java)

Typ	Größe (Bits)	Minimum	Maximum
byte	8	-128	127
char	16	0	$2^{16} - 1$
short	16	-2^{15}	$2^{15} - 1$
int	32	-2^{31}	$2^{31} - 1$
long	64	-2^{63}	$2^{63} - 1$
float	32	$\pm 1.4E-45$	$\pm 3.4E+38$
double	64	$\pm 4.9E-324$	$\pm 1.7E+324$
boolean	Undefiniert	Nur true und false	

Primitive Datentypen

Umwandlung

Implizite Umwandlung

Für die meisten Fälle gilt: Primitive Datentypen lassen sich grundsätzlich implizit umwandeln, wenn die Präzision des Zieldatentyps höher ist als die des Ursprungsdatentyps. Also:

`byte ⇒ short ⇒ int ⇒ long ⇒ float ⇒ double`

Primitive Datentypen

Umwandlung

Explizite Umwandlung

Alle anderen Umwandlungen müssen explizit durchgeführt werden zum Beispiel durch nutzen von Casting.

Achtung: Dies kann zu Datenverlust führen. Beispiel: Umwandlung von float in int ⇒ Dezimalstellen gehen verloren.

Objektorientierte Datentypen

Grundlegendes

- Zusammenfassen von Daten und Algorithmen in eine gemeinsame Instanz ⇒ **Klasse**
- Von einer Klasse können mehrere unabhängige Instanzen existieren ⇒ **Objekt**
- Entwickler entscheidet wie mit der Klasse interagiert werden muss ⇒ **Datenkapselung**
- Klassen lassen sich wiederverwenden ⇒ Verringelter Arbeitsaufwand durch gut designete Klassen
 - Durch Verwendung weiterführender Konzepte (wie z.B. Design Patterns) noch gestärkt
- Bilden die Grundlage von Java

Objektorientierte Datentypen

Vererbung

- Grundpfeiler der Objektorientierung
- Unterklassen **müssen** die Schnittstellen der Superklasse übernehmen
 - Können neue Methoden und Variablen hinzufügen
 - Können vorhandene Methoden überschreiben ⇒ Implementieren spezifischer Logik
 - Variablen bzw. Methoden können durch Vererbung **NICHT** entfernt werden
- Objekte der Unterklassen können anstelle der Oberklasse treten
 - Frage: Warum nicht umgekehrt?

Objektorientierte Datentypen

Vererbung

- Formales OOP erlaubt Mehrfachvererbung, nutzbar in z.B.:
 - C++
 - Python
 - Perl
- In Java: Nur Einfachvererbung erlaubt
 - Weitere Sprachen mit Einfachvererbung:
 - C#
 - Ruby
- Einige Funktionalitäten lassen sich über Interfaces abbilden

Objektorientierte Datentypen

Interfaces



„Java hat das Konzept der Mehrfachvererbung über die Nutzung von Interfaces durch die Hintertür eingeführt.“

Quelle:

Andreas Judt

https://www.researchgate.net/profile/Andreas_Judt

Objektorientierte Datentypen

Interfaces

- Definieren einen Satz an Methoden
- Implementierung der Funktionen erfolgt auf Klassenebene
 - Seit Java 8: Definition von Standardimplementierung im Interface möglich
- Klassen implementieren beliebig viele Interfaces
- Java bietet viele Standardinterfaces wie z.B.:
 - Comparable
 - Iterable
 - Serializable

Strukturierte Datentypen

Allgemeines

- Sind irgendwo zwischen Primitiven und Objektorientierten Datentypen einzuordnen
- In Java jedoch als Klassen umgesetzt (Sofern vorhanden)
- Wir betrachten folgende:
 - Enumerations
 - Structs
 - Union

Strukturierte Datentypen

Enumerations

- Ist eine Menge von geordneten, benannten Werten
- Werte der Enumeration können über Namen geprüft werden
- Anwendung:
 - Beschränken einer Variable auf bestimmte definierte Werte
 - Vermeiden von „Magic Numbers“ ⇒ Code-Lesbarkeit
- In den meisten Sprachen sind Enum Werte im Hintergrund nur Integer Konstanten
- In Java: Automatisch Klasse
 - Dadurch höhere Flexibilität

Enumerations

Beispiel I

Enumeration zur Repräsentation der Wochentage sieht zum Beispiel so aus:

```
1 enum Weekday {  
2     MONDAY,  
3     TUESDAY,  
4     WEDNESDAY,  
5     THURSDAY,  
6     FRIDAY,  
7     SATURDAY,  
8     SUNDAY  
9 }  
10 //Spaetere Nutzung im Code:  
11 Weekday myDay = Weekday.TUESDAY;
```

Enumerations

Beispiel II

- Durch die Klassenstruktur können Enumerations in Java komplexer sein
- Definition von mehreren Variablen der Enum-Werte
- Definition von Konstruktoren
- Definition von Methoden

Enumerations

Komplexes Beispiel I

```
1  public enum Planet {  
2      MERCURY (3.303e+23, 2.4397e6),  
3      VENUS   (4.869e+24, 6.0518e6),  
4      EARTH   (5.976e+24, 6.37814e6),  
5      MARS    (6.421e+23, 3.3972e6),  
6      JUPITER (1.9e+27, 7.1492e7),  
7      SATURN  (5.688e+26, 6.0268e7),  
8      URANUS  (8.686e+25, 2.5559e7),  
9      NEPTUNE (1.024e+26, 2.4746e7);  
10  
11      private final double mass;    // in kilograms  
12      private final double radius; // in meters
```

Enumerations

Komplexes Beispiel II

```
13     Planet(double mass, double radius) {
14         this.mass = mass;
15         this.radius = radius;
16     }
17     private double mass() { return mass; }
18     private double radius() { return radius; }
19
20     // universal gravitational constant (m3 kg-1 s-2)
21     public static final double G = 6.67300E-11;
22
23     double surfaceGravity() {
24         return G * mass / (radius * radius);
25     }
```

Enumerations

Komplexes Beispiel II

```
26     double surfaceWeight(double otherMass) {  
27         return otherMass * surfaceGravity();  
28     }  
29 } // public enum Planet  
30 //Verwendung:  
31 Planet jup = Planet.JUPITER;  
32 double someMass = 10;  
33 double weightOnJupiter = jup.surfaceWeight(someMass);
```

Strukturierte Datentypen

Structs

- Fassen mehrere Einzelvariablen zu einer Struktur zusammen
- Reine Datenstruktur zum erfassen von Daten
 - Keine Methoden
 - Keine Zugriffsspezifizierer
- Kein separates Konzept in Java ⇒ Sonderfall einer Klasse

Structs

Beispiel in C++

```
1 struct Date{  
2     int day;  
3     int month;  
4     int year;  
5  
6     Date():day(1)  
7             ,month(1)  
8             ,year(1970){}  
9 };  
10 //Verwendung:  
11 struct Date oneDate = {17,3,2019}  
12 oneDate.month=7;
```

Structs

Beispiel in Java

```
1 public class DateStruct{  
2     int day = 1;  
3     int month = 1;  
4     int year = 1970;  
5 }  
6 //Verwendung:  
7 DateStruct aDate = new DateStruct();  
8 aDate.day=1;  
9 aDate.month=4;  
10 aDate.year=2019;
```

Structs

Verwendung in Java

Auszug der Java Code Conventions

10 - Programming Practices

10.1 Providing Access to Instance and Class Variables

Don't make any instance or class variable public without good reason. Often, instance variables don't need to be explicitly set or gotten-often that happens as a side effect of method calls.

Structs

Verwendung in Java

Auszug der Java Code Conventions

10 - Programming Practices

10.1 Providing Access to Instance and Class Variables

Don't make any instance or class variable public without good reason. Often, instance variables don't need to be explicitly set or gotten—often that happens as a side effect of method calls.

One example of appropriate public instance variables is the case where the class is essentially a data structure, with no behavior. In other words, if you would have used a struct instead of a class (if Java supported struct), then it's appropriate to make the class's instance variables public.

Strukturierte Datentypen

Unions

- Enthält mehrere Variablen
- Diese teilen sich den gleichen Speicher
- Variablen können demnach nicht unabhängig voneinander verändert werden
- Streng genommen immer nur ein Member „aktiv“
 - „Aktiv“ ist immer das member, auf das zuletzt geschrieben wurde
 - Lesen von den anderen membern ist undefiniertes Verhalten
 - Von den meisten Compilern jedoch definiert
- Kein (mir bekanntes) äquivalent in Java

Unions

Codebeispiel in C++

```
1 Union Example{  
2     std::int32_t n;          //Belegt 4 Byte  
3     std::uint16_t s[2]       //Belegt 4 Byte  
4     std::uint8_t c;         //Belegt 1 Byte  
5 };                           //Gesamte Union belegt 4 Byte  
6 //Verwendung:  
7 Example ex = {0x12345678};   //n wird initialisiert. Lesen  
    ↪ von s und c ist undefiniertes Verhalten  
8 ex.s[0] = 0x0011;           //s wird zum aktiven member  
9 std::cout << ex.c << std::endl; //0x11 oder 0x00,  
    ↪ Plattformspezifisch  
10 std::cout << ex.n << std::endl; //0x12340011 oder 0  
    ↪ x00115678
```

Kontakt

- E-Mail: lukas.abelt@airbus.com
- GitHub: <https://www.github.com/LuAbelt>
- GitLab: <https://www.gitlab.com/LuAbelt>
- Telefon(Firma): 07545 - 8 8895
- Telegram: LuAbelt