

Programmierpraktikum

Ralf Hinze

Fachbereich Informatik
Technische Universität Kaiserslautern

SS 2021
12. April 2021

Teil I

Überblick

Gliederung

- 1 Organisation
- 2 Java in a Nutshell

Lernziele

- Programmieren, Programmieren, Programmieren
- Programmieren in Java
- Programmieren in Teams
- verteilte Versionsverwaltung mit Git
- (keine Anforderungsanalyse)

Organisation

- Corona-bedingt findet das Programmierpraktikum dieses Semester vollständig online statt!
- Beachten Sie bitte immer die aktuellen Informationen auf unserer Website:
<https://pl.cs.uni-kl.de/homepage/de/teaching/ss21/pp/>

- Wir stellen Ihnen wie üblich Foliensätze als pdf Datei zur Verfügung.
- Zusätzlich erstellen wir Videos, bei denen die Folien mit Audio versehen sind. Dieses Format ersetzt die klassischen Vorlesungen im Hörsaal.
- Anstelle von Live-Demos im Hörsaal erstellen wir Video-Screencasts.
- Die Materialien werden nach und nach bereitgestellt, so wie sie für den Ablauf des Praktikums benötigt werden.
- Alle Materialien und Videos werden von unserer Website ausgehend verlinkt.

Zeitplan Vorlesungen

- **1. Vorlesungswoche** (ab 19.04.2021):
Organisation und Überblick
- **2. Vorlesungswoche** (ab 26.04.2021):
Grundlagen der Programmierung in Java
- **3. Vorlesungswoche** (ab 03.05.2021):
Objektorientierte Programmierung in Java
- **4. Vorlesungswoche** (ab 10.05.2021):
Git
- **6. Vorlesungswoche** (ab 24.05.2021):
 - Build Tools
 - Testen
- **14. Vorlesungswoche** (ab 19.07.2021):
Abschlussveranstaltung

Übungsbetrieb

Woche	Aufgabe	Dauer	Teamgröße	Abgabe
2	Übungsblatt 1	1 Woche	2-er Gruppen	ExClaim
3	Übungsblatt 2	1 Woche	2-er Gruppen	ExClaim
4-5	Miniprojekt 1	2 Wochen	4-er Gruppen	Git
6-7	Miniprojekt 2	2 Wochen	4-er Gruppen	Git
8-10	Projekt 1	3 Wochen	4-er Gruppen	Git
11-13	Projekt 2	3 Wochen	4-er Gruppen	Git

- Das Praktikum besteht aus zwei Übungsblättern, zwei Miniprojekten und zwei Projekten.
- Das erste Übungsblatt erscheint am 27.04. und ist am 04.05. um 15 Uhr abzugeben.
- Die Zuteilung der Abgabepartner erfolgt via ExClaim.

Sprechstunden

- Bei Problemen mit den Übungsaufgaben können Sie Fragen über Mattermost stellen:
Team-URL: <https://teamwork.rarp-kl.de/pp-ss21/>
Loggen Sie sich via RHRK-GitLab mit Ihrem RHRK-Account ein.
Um dem Team beizutreten, müssen Sie einmalig den Einladungslink öffnen:
https://teamwork.rarp-kl.de/signup_user_complete/?id=xycw3pi1apgjp8psdrxxz9q3ra
Zu den auf der Website angekündigten Sprechstundenzeiten sind Tutoren im Kanal Sprechstunde verfügbar. Von dort aus können Sie dann bei Bedarf Meetings per Jitsi (oder eine ähnliche Technologie) vereinbaren.
- Beachten Sie, dass die Mattermost-Sprechstunden, wie herkömmliche Präsenz-Sprechstunden auch, zeitlich begrenzt sind.

Erfolgreicher Abschluss

- Jedes Übungsblatt, jedes Miniprojekt und jedes Projekt wird zufriedenstellend gelöst.
- Projekte werden von den Tutor*innen abgenommen.
- Jedes Teammitglied muss zum Erfolg beitragen.

Benötigte Software

- Java (JDK), Version 11 oder neuer.
- Eine beliebige IDE (Integrierte Entwicklungsumgebung) für Java.
Support von uns gibt es nur für IntelliJ IDEA.
- Details finden Sie in der Installationsanleitung auf unserer Website.

- Dokumentation: <https://docs.oracle.com/en/java/>
- *Java Precisely*, Peter Sestoft
- *Effective Java*, Joshua Bloch
- *Java Generics and Collections: Speed Up the Java Development Process*, Maurice Naftalin und Philip Wadler
- [*Einführung in die Programmierung mit Java*, Robert Sedgewick, Kevin Wayne]
- [*Java in a Nutshell: A Desktop Quick Reference*, Ben Evans, David Flanagan]

Java

One of the most important influences on the design of Java was a much earlier language called Simula.
— James Gosling

Your development cycle is much faster because Java is interpreted. The compile-link-load-test-crash-debug cycle is obsolete.
— James Gosling

Like the creators of sitcoms or junk food or package tours, Java's designers were consciously designing a product for people not as smart as them.
— Paul Graham

Geschichte [Sun]

- 1990–1992: Green Projekt; *7: portabler Minicomputer zur Steuerung von Haushaltsgeräten; Oak Interpreter (James Gosling)
- 1995: Veröffentlichung von Java; Internet-Browser (Applets)
- JDK 1.0: 23. Januar 1996
- JDK 1.1: 19. Februar 1997
 - innere Klassen
- Java 2 (J2SE 1.2): 8. Dezember 1998
 - Just-In-Time-Compiler (JIT)
- Java 3 (J2SE 1.3): 8. Mai 2000
- Java 4 (J2SE 1.4): 6. Februar 2002
 - Zusicherungen (engl. assertions)
- Java 5 (J2SE 5.0): 30. September 2004
 - parametrischer Typpolymorphismus (“generics”)
 - „for each“ Schleifen
- Java 6: 11. Dezember 2006

Geschichte [Oracle]

- [2010: Oracle kauft Sun]
- Java 7: 28. Juli 2011
- Java 8: 18. März 2014
 - Lambda-Ausdrücke (Funktionsausdrücke, „closures“)
 - **default** Methoden
 - „try with resources“
- Java 9: 21. September 2017
 - Module
 - interaktiver Interpreter (REPL: `jshe11`)
- Java 10: 20. März 2018
 - lokale Typinferenz (*var*)
- Java 11: 25. September 2018
- Java 12: 19. März 2019
- Java 13: 17. September 2019

Das Java System

Das Java System besteht aus

- der eigentlichen Programmiersprache,
- dem Übersetzer und dem Laufzeitsystem (Java Virtual Machine, JVM) und
- einer umfangreichen Bibliothek.

Java: Entwurfsziele

Ziele beim Entwurf von Java und der Laufzeitumgebung:

- *Portabilität:*
einfache Übertragung von Programmen über das Internet; Ausführung in fremden Umgebungen;
- *Verlässlichkeit:*
Vermeidung von Laufzeitfehlermeldungen und 'core dumped';
 - zur Übersetzungszeit: Typprüfung von Java-Quellcode,
 - zur Ladezeit: Typprüfung von Java-Bytecode,
 - zur Laufzeit: z.B. Bereichsprüfung bei Arraysubskription;
- *Sicherheit:*
Schutz der Umgebung vor fehlerhaften oder bösartigen Programmen;
- *Einfachheit:*
die Sprache sollte C und C++ Programmierer ansprechen;
- *Effizienz:*
sekundäres Entwurfsziel.

Java: Features

- Java ist eine klassenbasierte, objektorientierte Sprache
- (ein Java Programm ist eine Sammlung von Modulen ...)
- (ein Modul ist eine Sammlung von Paketen ...)
- ein Java Paket besteht aus Schnittstellen und Klassen
- Java kennt im Wesentlichen nur Klassendefinitionen
 - Records sind spezielle Klassen
 - Varianten werden nicht unterstützt (Simulation: „Composite“ und „Visitor“ Entwurfsmuster)
 - Module sind spezielle Klassen
 - Arrays sind spezielle Objekte
 - Ausnahmen sind spezielle Objekte
- Java kennt im Wesentlichen nur Methoden
 - Funktionen sind spezielle Methoden
- Methoden sind standardmäßig virtuell
- Java unterscheidet zwischen Ausdrücken und Anweisungen
- Bezeichner sind standardmäßig veränderlich
- (keine Mehrfachvererbung)
- 🖱️ *Annahme*: Sie können F#

F# versus Java: Klassen — Bankinstitut

```
type TrustMe (seed : int) =  
  let mutable funds = seed  
  
  member self.Deposit amount =  
    funds ← funds + amount  
  
  member self.Withdraw amount =  
    if amount > funds then  
      raise (Insufficient funds)  
    else  
      funds ← funds - amount  
  
  member self.Balance =  
    funds
```

```
class TrustMe {  
  private long funds = 0;  
  TrustMe (long seed) {  
    this.funds = seed;  
  }  
  void deposit (long amount) {  
    funds += amount;  
  }  
  void withdraw (long amount)  
    throws Insufficient {  
    if (amount > funds)  
      throw new Insufficient (funds);  
    else  
      funds -= amount;  
  }  
  long balance () { return funds; }  
}
```

F# versus Java: Algorithmik — ternäre Suche

```
let ternarySearch key (a : 'a []) =  
  let mutable l = 0  
  let mutable u = a.Length - 1  
  let mutable found = None  
  while l ≤ u && found = None do  
    let m = (l + u) / 2  
    if key < a.[m] then  
      u ← m - 1  
    elif key = a.[m] then  
      found ← Some m  
    (*key > a.[m]*) else  
      l ← m + 1  
  found
```

```
static final <T extends Comparable<T>>  
Optional<Integer>  
ternarySearch (T key, T [] a) {  
  int l = 0;  
  int u = a.length - 1;  
  while (l ≤ u) {  
    int m = (l + u) / 2;  
    int cmp = key.compareTo (a [m]);  
    if (cmp < 0)  
      u = m - 1;  
    else if (cmp == 0)  
      return Optional.of (m);  
    /* cmp > 0 */ else  
      l = m + 1;  
  }  
  return Optional.empty ();  
}
```

- ein F# Programm ist eine Sammlung von Modulen
- ein Modul enthält Wertedefinitionen, Typdefinitionen und lokale Module
- *Alternative*: Namensräume
- ein Namensraum enthält Typdefinitionen und lokale Module

```
namespace TUK.FBI.PP
type Person = Person of string
module Tutors =
    let tutors = [Person "Lisa"]
```

- Namensräume sind offen: mehrere Quelldateien können zum gleichen Namensraum beitragen

F# versus Java Programme

- ein Java Programm ist eine Sammlung von Paketen
- ein Paket enthält Schnittstellen- und Klassendefinitionen

```
package tuk.fbi.pp;
public class Person {
    public String name;
    public Person (String name) {
        this.name = name; }
    static public Person [] tutors =
        { new Person ("Lisa") };
}
```

- Pakete sind offen: mehrere Quelldateien können zum gleichen Paket beitragen

F# versus Java Programme

- Verwendung des Namensraums

```
open TUK.FBI.PP
```

- gezielt

```
type Person = TUK.FBI.PP.Person
```

- Verwendung des Pakets

```
import tuk.fbi.pp.*;
```

- gezielt

```
import tuk.fbi.pp.Person;
```

Module versus Klassen

- zwei Konzepte:
- F# kennt Module
- F# kennt Klassen

```
module Mathematics
```

```
...
```

- ein Konzept:
- Java kennt nur Klassen
- Module sind „statische“, nicht vererbte Klassen: **final class**

```
public final class Mathematics {
```

```
...
```

```
}
```

Funktionen versus Methoden

- zwei Konzepte:
- F# kennt Methoden
- F# kennt Funktionen

```
let rec factorial (n : int) =  
    if n = 0 then 1  
    else n * factorial (n - 1)
```

- ein Konzept:
- Java kennt nur Methoden
- Funktionen sind Klassenmethoden

```
static final int factorial (int n) {  
    if (n == 0)  
        return 1;  
    else  
        return n * factorial (n - 1);  
}
```


Ausdrücke versus Anweisungen

- F# kennt nur Ausdrücke
- Ausdrücke können Effekte haben
- Anweisung: effektvoller Ausdruck vom Typ *unit*
- Prozedur: effektvolle Funktion vom Typ $t \rightarrow unit$

- Java unterscheidet syntaktisch zwischen Ausdrücken und Anweisungen
- Alternative (Ausdruck):

$e_1 ? e_2 : e_3$

- Alternative (Anweisung):

if (*e*)
 s1
else
 s2

- sowohl Ausdrücke als auch Anweisungen können Effekte haben

Konstanten versus Variablen

- Konstante (default)

```
let pi = 3.141592654
```

- Variable

```
let mutable counter = 0
```

- Konstante

```
static final double pi = 3.141592654;
```

- Variable (default)

```
int counter = 0
```

Java: Basistypen versus Referenztypen

- Java unterscheidet zwischen
 - Basistypen (primitive types): *boolean*, *int*, ...und
 - Referenztypen: Schnittstellen und Klassen
- (aus Effizienzgründen)
- nur Referenztypen können selbst definiert werden
- jeder Referenztyp enthält ein zusätzliches Element:

null

- sendet man dem „Objekt“ *null* eine Nachricht, erhält man eine

java.lang.NullPointerException

- (siehe Sir Tony Hoare: „my billion-dollar mistake“)

- die Syntax von Java ist an die Syntax von C angelehnt
- Auszug (der Methodenrumpf ist ein Block)

```
MethodBody = Block | ;  
    Block = { {Declaration | Statement} }  
Declaration = Modifier Type Id [= Expression] ;  
Statement = ;  
            | Expression ;  
            | if ( Expression ) Statement [else Statement]  
            | while ( Expression ) Statement  
            | do Statement while ( Expression ) ;  
            | break ;  
            | return [Expression] ;  
            | throw Expression ;  
            | ...  
            | Block
```

Syntax: Fallstricke — dangling *else*

- „dangling *else*“ Problem (engl. für baumeln)

```
if (x == 0)
  if (y > 3) y = 1;
else
  y = 0;
```

- *else* Zweig gehört zum vorangegangenen *if*
- Klammern helfen

```
if (x == 0) {
  if (y > 3) y = 1;
} else {
  y = 0;
}
```

Syntax: Fallstricke — Fallunterscheidung

- Fallunterscheidung (nur für Ordinaltypen und Strings)

```
switch (month) {  
  case 9 : case 4 : case 6 : case 11 :  
    days = 30;  
    break;  
  case 2 :  
    days = 28;  
    break;  
  default :  
    days = 31;  
}
```

- **break** nicht vergessen

Ausführbare Programme

- der Klassiker

```
printfn "hello, world"
```

- der Klassiker

```
public class HelloWorld  
{  
  public static  
    void main (String [] args)  
  {  
    println ("hello, world");  
  }  
}
```

- ein Programm wird durch den Aufruf der *main* Methode gestartet

Ausführbare Programme

- echo

```
[< EntryPoint >]
let main (args : string []) : int =
    for s in args do
        System.Console.WriteLine s
    0
```

- Einstiegspunkt muss den Typ `string []` → `int` haben

- echo

```
public class Echo
{
    public static
        void main (String [] args)
    {
        for (String s : args)
            System.out.println (s);
        System.exit (0);
    }
}
```


Java: Pragmatik

- eine **public** Klasse pro Datei (gleichen Namens)
- die Verzeichnisstruktur spiegelt die Paketstruktur wider

HelloWorld.java

```
public class HelloWorld  
{  
    public static  
        void main (String [] args) {  
            println ("hello, world"); }  
}
```

Echo.java

```
public class Echo  
{  
    ...  
}
```

tuk/fbi/pp/Person.java

```
package tuk.fbi.pp;  
public class Person {  
    ...  
}
```

tuk/fbi/pp/UsePerson.java

```
import tuk.fbi.pp.Person;  
class UsePerson {  
    ...  
}
```

Java: Übersetzer und Entwicklungsumgebungen

- Kommandozeile:
javac und java
- Java-Shell (REPL):
jshell
- BlueJ:
integrierte Entwicklungsumgebung für Java, speziell für Ausbildungszwecke konzipiert
- Eclipse:
quelloffenes Programmierwerkzeug zur Entwicklung von Software, insbesondere Java
- IntelliJ IDEA:
integrierte Entwicklungsumgebung des Softwareunternehmens JetBrains