Korrekte Software: Grundlagen und Methoden
Vorlesung 1 vom 07.04.15: Einführung

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2016

Organisatorisches

- Veranstalter:

Christoph Lüth	Serge Autexier
`christoph.lueth@dfki.de`	`serge.autexier@dfki.de`
MZH 4185, Tel. 59830	Cartesium 2.11, Tel. 59834

- Termine:
  - Vorlesung: Montag, 16 – 18, MZH 1460
  - Übung: Donnerstag, 14 – 16, MZH 1460
- Webseite:

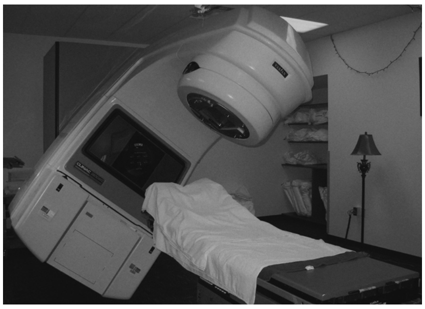
`http://www.informatik.uni-bremen.de/~cxl/lehre/ksgm.ss16`

Prüfungsformen

- 10 Übungsblätter (geplant)
- Prüfungsform 1:
  - Bearbeitung der Übungsblätter,
  - Fachgespräch,
  - Note aus den Übungsblättern.
- Prüfungsform 2:
  - Mind. ausreichende Bearbeitung der Übungsblätter (50%),
  - mündliche Prüfung,
  - Note aus der Prüfung.

Warum Korrekte Software?

Software-Disaster I: Therac-25



Bekannte Software-Disaster II: Ariane-5

Bekannte Software-Disaster III: Airbus A400M

Inhalt der Vorlesung

Themen

Korrekte Software im Lehrbuch:

- Spielzeugsprache
- Wenig Konstrukte
- Kleine Beispiele

Korrekte Software im Einsatz:

- Richtige Programmiersprache
- Mehr als nur ganze Zahlen
- Skalierbarkeit — wie können große Programme verifiziert werden?

Inhalt

- Grundlagen:
  - Der Hoare-Kalkül — Beweis der Korrektheit von Programmen
  - Bedeutung von Programmen: Semantik
- Erweiterung der Programmkonstrukte und des Hoare-Kalküls:
  1. Reiche Datenstrukturen (Felder, `struct`)
  2. Funktion und Prozeduren (Modularität)
  3. Referenzen (Zeiger)
- Übungsbetrieb:
  - Betrachtete Programmiersprache: “C0” (erweiterte Untermenge von C)
  - Entwicklung eines Verifikationswerkzeugs in Scala
  - Beweise mit Isabelle (mächtiger Theorembeweiser)

Nächste Woche

- Aussagenlogik
- Erstes Übungsblatt

Introduction to Scala

Based on the “Scala Training Course” by
Fredrik Vraalsen (`fredrik@vraalsen.no`) and
Alf Kristian Støyle (`alf.kristian@gmail.com`)
of scalaBin released under
Creative Commons Attribution 3.0 Unported license

Conciseness

```
public class Person {
  private int age;
  private String name;

  public Person(int age, String
      name) {
    this.age = age;
    this.name = name;
  }

  public int getAge() {
    return this.age;
  }

  public void setAge(int age) {
    this.age = age;
  }

  public String getName() {
    return this.name;
  }

  public void setName(String
      name) {
    this.name = name;
  }
}
```

```
class Person(var age: Int, var name:
    String)
```

Conciseness

```
List<Person> persons = ...
List<Person> adults = new LinkedList<Person>();
List<Person> kids = new LinkedList<Person>();
for (Person person : persons) {
  if (person.getAge() < 18) {
    kids.add(person);
  } else {
    adults.add(person);
  }
}
```

```
val (kids, adults) = persons.partition(_.age < 18)
```

Conciseness

```
String s = "!em esreveR";
System.out.println(s.reverse());
```

```
val s: java.lang.String = "!em esreveR"
println(s.reverse)

=> Reverse me!
```

Higher-Order

```
List<Person> persons = ...
List<Person> adults = new LinkedList<Person>();
List<Person> kids = new LinkedList<Person>();
for (Person person : persons) {
  if (person.getAge() < 18) {
    kids.add(person);
  } else {
    adults.add(person);
  }
}
```

```
val (kids, adults) = persons.partition(_.age < 18)
```

Java Interaction, Higher-Order

```
BufferedReader reader = null;
try {
  reader = new BufferedReader(new FileReader("f.txt"));
  System.out.println(reader.readLine());
} finally {
  if (reader != null) {
    try {
      reader.close();
    } catch (IOException e) {
      // Exception on close, ignore
    }
  }
}
```

```
using(new BufferedReader(new FileReader("f.txt"))) {
    reader => println(reader.readLine())
}
def using[A, B <: {def close(): Unit}] (closeable: B) (f: B =>
     A): A =
  try { f(closeable) } finally { closeable.close() }
```

```
val myList = List(1, 2, 3)
val res = (10 /: myList) (_+_)
```

```
=> ??
```

Scala

- Object oriented and functional
- Statically typed
- Java compatible
  - Compiles to Java bytecode (and CLR)
  - Existing libraries/frameworks
- Better Java

Topics

- Basic syntax
- REPL
- First class functions
- Pattern matching
- OO and traits
- Functional programming
- Higher-Order Functions
- Implicits
- (XML)

Basic Syntax

;

- Is optional (inferred)
- Except if multiple statements in a line

```
val s = "hello"
println(s)
```

```
val s = "hello"; println(s)
```

Variables

Scala	Java
s:String i:Int	String s int i / Integer i
val s = "Hello World"	public final String s = "Hello World";

var i = 1
private var k = 3
	public int i = 1;
	private int j = 3;

Methods

Scala	Java

`def add(x: Int, y: Int): Int = {`
`    x + y`
`}`
	`public int add(int x, int y) {`
	`    return x + y;`
	`}`

`def add(x: Int, y: Int) = x + y`	

`def doSomething(text: String) {`
`}`
	`public void doSometing(String`
	`    text) {`
	`}`

Methods

Scala	Java
myObject.myMethod(1) myObject myMethod(1) myObject myMethod 1	myObject.myMethod(1);
myObject.myOtherMethod(1, 2) myObject myOtherMethod(1, 2)	myObject.myOtherMethod(1, 2);
myObject.myMutatingMethod() myObject.myMutatingMethod // myObject myMutatingMethod	myObject.myMutatingMethod()

Methods

Scala

```
override def toString = ...
```

Java

```
Override
public String toString() {...}
```

Classes And Constructors

Scala

```
class Person(val name: String)
```

Java

```
public class Person {
 private final String name;
 public Person(String name) {
    this.name = name;
 }
 public String getName() {
 return name;
 }
}
```

Traits (= Interface + Mixin)

Scala

```
trait Shape {
   def area: Double
}

class Circle extends Object
          with Shape
```

Java

```
interface Shape {
public double area();
}

public class Circle extends
       Object
          implements Shape
```

No “Static” in Scala

Scala

```
object PersonUtil {
  val AgeLimit = 18

  def countPersons(persons:
        List[Person]) = ...
}
```

Java

```
public class PersonUtil {
 public static final int
        AGE_LIMIT = 18;

 public static int
        countPersons(List<Person>
           persons) {
     ...
 }
}
```

if-then-else

Scala

```
if (foo) {
...
} else if (bar) {
...
} else {
...
}
```

Java

```
if (foo) {
...
} else if (bar) {
...
} else {
...
}
```

For-Loops

Scala

```
for (i <- 0 to 3) {
     ...
}

for (s <- args) println(s)
```

Java

```
for (int i = 0; i < 4; i++) {
     ...
}

for (String s : args) {
     System.out.println(s);
}
```

While-Loops

Scala

```
while (true) {
 ...
}
```

Java

```
while (true) {
 ...
}
```

Exceptions

Scala

```
throw new Exception("...")

try {
} catch {
   case e: IOException => ...
} finally {
}
```

Java

```
throw new Exception("...")

try {
} catch (IOException e) {
 ...
} finally {
}
```

Varargs

Scala

```
def foo(values: String*){ }


foo("bar", "baz")


val arr = Array("bar", "baz")
foo(arr: _*)
```

Java

```
public void foo(String ...
    values){ }


foo("bar", "baz");


String [] arr =
    new String[]{"bar", "baz"}
foo(arr);
```

(Almost) everything is an expression

```
val res = if (foo) x else y

val res = for (i <- 1 to 10) yield i      // List(1, ..., 10)

val res = try { x } catch { ...; y } finally { }   // x or y
```

Collections – List

Scala

```
val numbers = List(1, 2, 3)

val numbers = 1 :: 2 :: 3 :: Nil



numbers(0)
=> 1
```

Java

```
List<Integer> numbers =
    new ArrayList<Integer>();
numbers.add(1);
numbers.add(2);
numbers.add(3);

numbers.get(0);
=> 1
```

Collections – Map

Scala

```
var m = Map(1 -> "apple")
m += 2 -> "orange"



m(1)
=> "apple"
```

Java

```
Map<Int, String> m =
    new HashMap<Int, String>();
m.put(1, "apple");
m.put(2, "orange");

m.get(1);
=> apple
```

Generics

Scala

```
List[String]
```

Java

```
List<String>
```

Tuples

Scala

```
val tuple: Tuple2[Int, String] =
(1, "apple")

val quadruple =
(2, "orange", 0.5d, false)
```

Java

```
Pair<Integer, String> tuple =
    new Pair<Integer, String>(1,
    "apple")


 ... ;-)
```

Packages

Scala

```
package mypackage
 ...
```

Java

```
package mypackage;
 ...
```

Imports

Scala

```
import java.util.{List,
    ArrayList}


import java.io._

import java.sql.{Date => SDate}
```

Java

```
import java.util.List
import java.util.ArrayList

import java.io.*

???
```

Nice to Know

Scala	Java
`println ("Hello")`	`System.out. println ("Hello");`

`val line = readLine()`	`BufferedReader r = new BufferedReader(new`
	`InputStreamRead(System.in));`
	`String line = r.readLine();`

`sys. error ("Bad")`	`throw new RuntimeException("Bad")`

`1 + 1`
`1 .+(1)`
	`new Integer(1). toInt () + new`
	`Integer (1). toInt ();`

`1 == new Object`
`1 eq new Object`
	`new Integer(1).equals(new Object());`
	`new Integer(1) == new Object();`

`"""A\sregex""".r`	`java. util .regex.Pattern.compile("A\\sregex");`
`s"3 + 4 = ${3 + 4}" // "3 + 4 = 7"`	`"3 + 4 = " + (3 + 4)`

Topics

- Basic syntax
- REPL
- First class functions
- Pattern matching
- OO and traits
- Functional programming
- Higher-Order Functions
- Implicits
- (XML)

REPL - Read eval print loop

- Command line shell for on-the-fly execution of Scala statements
- bin/scala

IDE and Build Tools

- Scala IDE for Eclipse is the officially supported Platform by the creators of Scala.`http://scala-ide.org/`
- Scala Plugin for IDEA is very good too. (And IDEA is avaliable in a free edition)
- There used to be support for Netbeans, but that seems to be dead right now.

Build Tool

- SBT
  (Scala Build Tool) is an Mawen compatible build tool for Scala and Java
  `http://www.scala-sbt.org/`

First Class Functions

```
val even = Function[Int, Boolean] {
    def apply(i: Int) = i % 2 == 0
}

val even: (Int => Boolean) = (i: Int) => i % 2 == 0
val even = (i: Int) => i % 2 == 0

even.apply(42)        // true
even(13)              // false
```

First Class Functions

```
val numbers = List(1, 2, 3, 4, 5)

numbers. filter (even)                 // List(2, 4)

numbers. filter ((i: Int) => i > 2)    // List(3, 4, 5)
numbers. filter (i => i > 2)           // List(3, 4, 5)
numbers. filter (_ > 2)                // List(3, 4, 5)
```

Collections

```
numbers. filter (i => i > 2)                // List(3, 4, 5)
numbers.find(i => i > 2)                    // Some(3)
numbers.exists(i => i > 2)                  // true
numbers. forall (i => i > 2)                // false

numbers.map(i => i *2)                      // List(2, 4, 6, 8, 10)

numbers.foldLeft(0) { (a, b) => a + b }     // 15
```

Deferred execution - constructed example

```
helloButton.addActionListener(e =>
    println ("Hello World!")
)
```

Closure

```
val people = List(Person("Alf"), Person("Fredrik"))

val name = "Fredrik"
val nameFilter = (p: Person) => p.name == name

people. filter (nameFilter) // Person("Fredrik")
```

Closures

```
val people = List(Person("Alf"), Person("Fredrik"))

var name = "Fredrik"
val nameFilter = (p: Person) => p.name == name

people. filter (nameFilter)    // Person("Fredrik")
name = "Alf"
people. filter (nameFilter)    // Person(Älf")
```

Pattern Matching

```
myObject match {
  case 1 => println("First was hit")
  case 2 => println("Second was Hit")
  case _ => println("Unknown")
}
```

Pattern Matching

```
myObject match {
  case i: Int  => println("Found an int")
  case s: String  => println("Found a String")
  case _ => println("Unknown")
}
```

Pattern Matching

```
myObject match {
  case i: Int  => println("Found an int")
  case s: String  => println("Found an String")
  case other => println("Unknown " + other)
}
```

Pattern Matching

```
myObject match {
  case i: Int if i == 1 => println("Found an int")
  case s: String  => println("Found a String")
  case other => println("Unknown " + other)
}
```

Pattern Matching

```
val res = myObject match {
    case i: Int if i == 1 => "Found an int"
    case s: String => "Found a String"
    case other => "Unknown " + other
}
```

Pattern Matching

```
val res = myObject match {
    case ( first , second) => second
    case ( first , second, third ) => third
}
```

Pattern Matching

```
val mathedElement = list match {
  case List( firstElement , lastElement) => firstElement
  case List( firstElement , _ *) => firstElement
  case _ => "failed"
}
```

Pattern Matching

```
def length( list : List [_]): Int =
   list match {
     case Nil => 0
     case head :: tail => 1 + length(tail)
   }
```

Pattern Matching

```
public static Integer getSecondOr0(List<Integer> list) {
      if ( list != null && list. size () >= 2) {
            return list .get(1);
      } else {
            return 0;
      }
}
```

⇓

```
def second_or_0(list : List [ Int ]) = list match {
  case List (_, x, _*) => x
  case _ => 0
}
```

Case classes

- Class types that can be used in pattern matching
- Generated into your class:
  - `equals`
  - `hashCode`
  - `toString`

Case classes

```
abstract class Person(name: String)
case class Man(name: String) extends Person(name)
case class Woman(name: String, children: List [Person])
  extends Person(name)
```

Case Classes

```
p match {
  case Man(name) => println("Man with name " + name)
  case Woman(name, children) => println("Woman with name" +
     name + " and with " + children. size + " children")
}
```

Regular Expressions

```
val regex = """(\d+)(\w+)""".r

val myString = ...

val res : String = myString match {
  case regex( digits , word) => digits
  case _ => "None"
}
```

Regular Expressions

```
val regex = """(\d+)(\w+)""".r

val myString = ...

val res : Option[String ] = myString match {
  case regex( digit , word) => Some(digit)
  case _ => None
}
```

Options

- Never NullPointerException again!
- Option has two possible values:
  - `Some(value)`
  - `None`

```
val someOption: Option[String] = Some("value")
val noOption:   Option[String] = None
```

Options

```
def getValue(s: Any): Option[String]

getValue(object) match {
  case Some(value) => println(value)
  case None => println("Nothing")
}

val result = getValue(object).getOrElse("Nothing")
```