Al Jaaf, Ileyan; Alsleben, Yannick; Greguric, Luka; Karpuzoglu, Yasemin, Rodovnitschenko, Anastasia; Rudenko, Yuliya

Agenda

- Einführung
- Schlüsselwörter
- Datentypen
- Schleifen
- Bedingungen
- Groovy – Truth
- Bedingte Operatoren
- Klassen & Methoden
- Exception Handling
- Datei I/O
- Datenstrukturen
- Testen

Java Virtual Machine (JVM)

- Teil der Laufzeitumgebung für Java Programme
- Java Programm wird in einer eigenen virtuellen Maschine ausgeführt

  → Ist für die Ausführung von Java-Bytecodes verantwortlich
- Beispiele für Sprachen auf der JVM: Java, Scala, Kotlin, Groovy

Geschichte zu Groovy

- Wurde 2003 von Bob McWhirter & James Strachan entwickelt
- Version 1.0: 2007,
  - → 2.0: 2012
  - → 3.0: 2014
- 2015: Projekt bei Apache Software Foundation

Allgemeines

- Unterstützt die dynamische und statische Typisierung
- Wird auf der JVM ausgeführt und sorgt für eine Verfügbarkeit für viele Plattformen wie Linux, MacOS, Windows
- Ist objektorientiert
- Groovy-Quellcode wird in Java-Bytecode kompiliert

→ Kann auf jeder Plattform ausgeführt werden, auf der JRE installiert ist

Gründe für Groovy

- Agil und dynamisch
- Nahtlose Integration mit allen vorhandenen Java-Objekten und Bibliotheken

  → Einfach für Java Entwickler

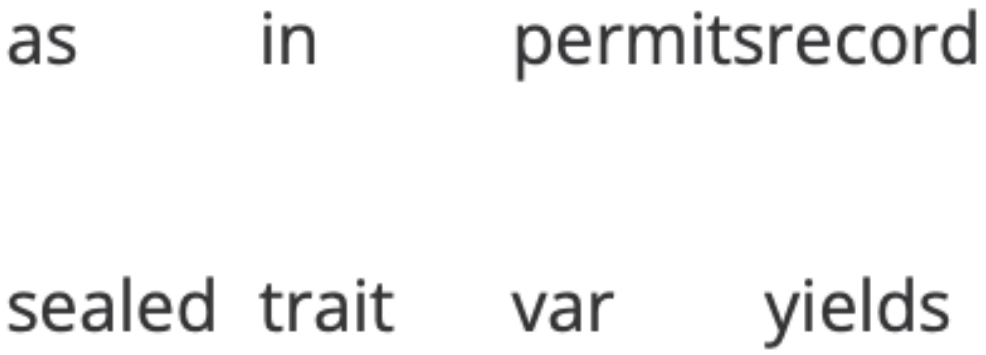
Schlüsselwörter

- Im Prinzip dieselben Schlüsselwörter wie in Java
- Nicht für Variablen und Methodennamen benutzen
- const, goto, strictfp und threadsafe sind Platzhalter
  →haben bisher keine Verwendung in Groovy

as	assert	break	case
catch	class	const	continue
def	default	do	else
enum	extends	false	finally
for	goto	if	implements
import	in	instanceof	interface
new	null	package	return
super	switch	this	throw
throws	trait	true	try
var	while		

Kontextuelle Schlüsselwörter

- Werden nur in bestimmten Fällen benutzt
- Dürfen für Variablennamen benutzt werden



Schlüsselwörter als Variablennamen

```
class A {
    def "this"() {
        print "Diese Methode trägt den Namen \"this\"."
    }

    def "while"(){
        this.this()
    }
}
//Beispielaufruf
A a = new A()
a.while()
```

Datentypen

Build–In Datentypen

- Groovy unterstützt alle eingebauten Datentypen von Java und hat zusätzlich eigene Datentypen und erweiterte Funktionen
- primitive Datentypen
  - Logisch - boolean
  - Ganzzahl – byte, short, int, long
  - Fliesskommazahlen – float, double
  - Zeichen – char
- Komplexe Datentypen
  - BigInteger
  - BigDecimal

Datentyp	Bezeichnung	Größe (Bits)	Wertebereich/Details
Logisch	boolean	n/a	true und false
Ganzzahl	byte	8	-128 bis 127
Ganzzahl	short	16	-32.768 bis 32.767
Ganzzahl	int	32	-2.147.483.648 bis 2.147.483.647
Ganzzahl	long	64	-9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807
Ganzzahl	BigInteger	unbegrenzt	Unbegrenzt, nur durch verfügbaren Speicher eingeschränkt
Fließkommazahlen	float	32	ca. 1.4E-45 bis 3.4E+38
Fließkommazahlen	double	64	ca. 4.9E-324 bis 1.7E+308
Dezimalzahlen	BigDecimal	unbegrenzt	Hohe Präzision, nur durch verfügbaren Speicher eingeschränkt
Zeichen	char	16	16-Bit Unicode Zeichen

Numerische Datentypen

```java
class Beispiel {
    static void main(String[] args) {
        // Beispiel für einen long Datentyp
        long y = 2_036_854_775_807L; // oder 50000L

        // Beispiel für einen Gleitkommazahlen-Datentyp
        float a = 10.56f;

        // Beispiel für einen double Datentyp
        double b = 10.54;

        // Beispiel für einen BigInteger Datentyp
        BigInteger bi = 35g;

        // Beispiel für einen BigDecimal Datentyp
        BigDecimal bd = 3.5g;


    }
}
```

Dynamische Typdeklaration

```
class Example {
    static void main(String[] args) {

def a = 1
boolean isInteger = a instanceof Integer
println(isInteger) // -> true

println(a)
    }
}
```

Komplexe Datentypen

Zeichenketten

Strings

```
class Beispiel {
    static void main(String[] args) {
      def einfach = 'einfache Anführungszeichen'
      def doppelt = "doppelte Anführungszeichen\n"
      def slashy = /ein "Slashy-String" ohne 'Escape'/
      def dollar = $/andere Möglichkeit und Einfügen von "/"/$
      def dreifach = '''
Ich bin
ein String der über
mehr Zeilen geht\n'''

      def tripple = """
erste Zeile
zweite Zeile
dritte Zeile"""

    println(einfach)
    println(doppelt)
    println(slashy)
    println(dollar)
    println(dreifach)
    println(tripple)
   }
}
```

```
Ausgabe:
einfache Anführungszeichen
doppelte Anführungszeichen

ein "Slashy-String" ohne 'Escape'
andere Möglichkeit und Einfügen von "/"

Ich bin
ein String der über
mehr Zeilen geht


erste Zeile
zweite Zeile
dritte Zeile
```

Statische vs. Dynamische Typen

Statisch typisiert (Java)

```java
public class MyClass {
    public static void main(String args[]) {
      String s = "Hallo Welt";
      System.out.println(s); // -> Hallo Welt

      s = 123;
      System.out.println(s); //-> Compiler-Fehler
      }
}
```

Dynamisch typisiert (Groovy)

```groovy
class Beispiel {
    static void main(String[] args) {
    def s = "Hallo Welt"
    println s.getClass() // -> class java.lang.String
    s = 123
    println s.getClass() // -> class java.lang.Integer

  }
}
```

GString

```
class Beispiel {
    static void main(String[] args) {
     def person = [name: 'Thomas Smits', lehrt: 'PR3']
     def hochschule = "Hochschule Mannheim"
     def ausdruck = "Hallo, mein Name ist ${person.name}. Ich unterrichte ${person.lehrt}. An der ${hochschule}."

     println(ausdruck)
   }
}
```

Ausgabe:
Hallo, mein Name ist Thomas Smits. Ich unterrichte PR3. An der Hochschule Mannheim.

String Konkatenation

- *Jeder* String kann mit „+“ konkateniert werden

```
class Beispiel {
    static void main(String[] args) {
     def eins = "Ein String"
     def zwei = ' wird konkateniert'
     println(eins + zwei) // -> Ein String wird konkateniert
    }
}
```

String Index

- Mit positiven & negativen Indizes auf Zeichen eines Strings zugreifen

```
class Beispiel {
    static void main(String[] args) {
    def greeter = "Hallo Welt"

    println(greeter[1]) //-> a
    println(greeter[-4]) // -> W
   }
}
```

Operatoren

Grundlegende Operatoren

- Arithmetische Operatoren (-, +, *, **,...)
- Relationale Operatoren (==, !=, <, >,...)
- Logische Operatoren (&&, ||, !)
- Bitweise Operatoren (&, |, ^, -)
- Bitweise Verschiebung (<<, >>,>>>)
- Konditionale Operatoren (!, ?, :, ?:)

Sicherheitsnavigations-Operator

- Verwendung: <Objekt>?.<Eigenschaft>
- Sicherer Zugriff auf verschachtelte Eigenschaften oder Methoden

→ Vermeiden von NullPointerExceptions

```
class Person {
    String name
    Address address
}

class Address {
    String city
}
def person = new Person(name: "Tom", address: new Address(city: "Mannheim"))
def nullPerson = new Person(name: "Max")

println person?.address?.city
println nullPerson?.adress?.city
```

```
Ausgabe: person = Mannheim
         nullPerson = null
```

Methodenreferenz-Operator (.&)

- Verwendung: <Methode> { <Objekt>.&<Methode>()}
- Referenziert eine Methode als Closure

```
class Person {
    String name

    Person(String name) {
        this.name = name
    }

    String getName() {
        this.name
    }
}

def personen = [new Person("Tom"), new Person("Max"), new Person("Tim")]
def names = personen.collect { it.&getName() }
println names
```

```
Ausgabe: [Tom, Max, Tim]
```

Spread-Operator (*.)

- Verwendung: <Collection>*.<Methode>
- Wendet eine Methode auf alle Elemente einer Collection an

```groovy
class Person {
    String name

    Person(String name) {
        this.name = name
    }

    String getName() {
        this.name
    }
}

def personen = [new Person("Tom"), new Person("Max"), new Person("Tim")]
def names = personen*.getName()
println names // Ausgabe: [Tom, Max, Tim]
```

Spread-Map-Operator

- Verwendung: def map2 = [<Element1>, <Element2>,... *:<map1>]
- Ermöglicht es die Inhalte einer Map in eine andere einzufügen

```
def map1 = [a: 1, b: 3]
def map2 = [c: 4, b: 2, *:map1]

println map2 // Ausgabe: [c:4, b:3, a:1]
```

Raumschiff-Operator (<=>)

- Verwendung: <Wert1> <=> <Wert2>
- Wird beim Vergleich verwendet

  → Gibt entweder -1, 0 oder 1 zurück

```
println 3 <=> 5
println 5 <=> 5
println 7 <=> 5
```

```
Ausgabe: (3, 5) = -1
         (5, 5) =  0
         (7, 5) =  1
```

Range-Operator (..)

- Verwendung: <Wert1>..<Wert2>
- Erstellt eine Sequenz von Werten innerhalb eines spezifischen Bereiches
- Der rechte Wert ist inklusiv, außer man benutzt den Exclusive-Range-Operator (..<)

```
def range = 1..5
println range.collect() // Ausgabe: [1, 2, 3, 4, 5]

def range2 = 1..<5
println range2.collect() // Ausgabe: [1, 2, 3, 4]

def range3 = (1..10).step 2
println range3.collect() // Ausgabe: [1, 3, 5, 7, 9]
```

Regex-Operator (=~) / (==~)

- Verwendung:

  <String> ~= <Muster>
- Werden für reguläre Ausdrücke verwendet
- =~ erstellt ein Matcher Objekt
- ==~ prüft ob eine Zeichenkette vollständig mit dem Muster übereinstimmt

```groovy
def text = "Groovy ist toll!"
def pattern = ~/Groovy/

if (text =~ pattern) {
    println("Der Text enthält das Wort 'Groovy'.")
} else {
    println("Der Text enthält nicht das Wort 'Groovy'.")
}
```

Schleifen

while - Schleife

Kopfgesteuert

```
class Beispiel {
    static void main(String[] args) {
  def y = 0
  while ( y < 5){
      print(y + " ") // -> 0 1 2 3 4
      y++
  }
   }
}
```

Fußgesteuert

```
def count = 1

do {
    count++
} while(count < 5)
```

for - Schleife

- Wie in Java gilt für while- und for-Schleifen:
  - break: aus der Schleife springen
  - continue: weiter mit der nächsten Iteration

```
for (int i = 0; i < 5; i++) {
    if (i == 2) {
        continue // 2 wird übersprungen
    }
    if (i == 4){
        break // Hier wird die Schleife bei i == 4 verlassen
    }
    print (i + " ") // Ausgabe: 0 1 3
}
```

times-Schleife

In Groovy besitzen Zahlen die Methode times → bekommt ein Closure als Parameter

Mit benanntem Parameter

```
5.times {def e -> println e + " "}
/* Ausgabe:
0
1
2
3
4
*/
```

Ohne benannten Parameter → Automatisch it als Name

```
5.times {println it + " "}
/* Ausgabe:
0
1
2
3
4
*/
```

each-Schleife

- Über die Elemente von Collections kann man noch mit Hilfe der **each**-Schleife iterieren
- Sie bekommt ein Closure als Parameter und wendet den Ausdruck auf alle Elemente an

```
def map = [name: "Thomas", nachname: "Smits"]

map.each {key, value -> println "$key: $value"}

/* Ausgabe:
name: Thomas
nachname: Smits
*/
```

for – in Schleife

```
for (variable in iterable) { body }
```

- Durchläuft das Objekt „Iterable“
- Häufig verwendete Iterable sind:
  - Ranges
  - Lists
  - Arrays
  - Maps
  - Zeichenfolgen

Iteration über Ranges

```
class Beispiel {
    static void main(String[] args) {

    for (i in 0..5){
        print(i + " ") // -> 0 1 2 3 4 5
    }
    for (i in 'a'..<'d')
        print(i + " ") // -> a b c letzter Buchstabe exklusiv
    }
}
```

Iteration über Listen

```
class Beispiel {
    static void main(String[] args) {

    def list = [1, 2, 3, 4, 5]
    for (element in list) {
        print element + " "  // -> 1 2 3 4 5
    }

  }
}
```

```
class Beispiel {
    static void main(String[] args) {

    def list = [1, 2, 3, 4, 5]
    for (element in list) {
        print element + " "  // -> 1 2 3 4 5
    }

  }
}
```

Iteration über Arrays

Iteration über Maps

```
class Beispiel {
    static void main(String[] args) {

    def map = [name: 'Alice', age: 18]
    for (entry in map) {
        print ("${entry.key}: ${entry.value} ") // -> name: Alice age: 18

    }
  }
}
```

Iteration über Zeichenfolgen

```
class Beispiel {
    static void main(String[] args) {

    def text = "Groovy"
    for (ch in text) {
        print (ch + " ") // -> G r o o v y
    }
  }
}
```

Bedingungen

Groovy-Truth und Operatoren

Bedingungen: If/else

```
def zahl = 8

if(zahl > 10){
    println "Zahl größer 10"
}else if(zahl < 10){
    println "Zahl ist kleiner 10"
}else{
    println "Zahl ist gleich 10"
}
```

```groovy
def x = 3
switch (x) {
    case 1:
        println "x is one"
        break
    case 2:
        println "x is two"
        break
    case 3..5: // Bereich
        println "x is between 3 and 5"
        break
    case Integer:
        println "x is an integer"
        break
    case ~/^\\d+$/: // Regex als Case erlaubt
        println "x is a number"
        break
    default:
        println "x is something else"
}
```

Switch - Case

```
def x = 3
switch (x) {
    case 1:
        println "x is one"
        break
    case 2:
        println "x is two"
        break
    case 3..5: // Bereich
        println "x is between 3 and 5"
        break
    case Integer:
        println "x is an integer"
        break
    case ~/^\\d+$/: // Regex als Case erlaubt
        println "x is a number"
        break
    default:
        println "x is something else"
}
```

Switch - Case

- Switch-Case unterstützt folgende Arten des Vergleichs
- • Case-Values stimmen überein, wenn Switch-Value eine Instanz derselben Klasse ist
- • „Regulärer Ausdruck-Case“-Value, wenn toString() Repräsentation von Switch-Value dem Regex gleicht
- • „Collection-Case“-Value stimmen überein, wenn S-Value in Collection vorkommt
- • „Closure-Case“-Value match, wenn Call auf Clojure ein „truthy“-Return liefert

Alternative Schreibweise Switch - Case

```
def partner = switch(person){
    case 'Romeo' -> 'Juliet'
    case 'Adam' -> 'Eve'
    case 'Eins' -> 'Zwei'
}
```

Groovy-Truth

Groovy beachtet folgende Regeln:

Boolean

- True wenn entsprechender Boolean-Wert true ist

```
assert true
assert !false
```

Collections und Arrays

- Non-Empty Collections und Arrays sind true

```
assert [1,2,3]
assert ![]
```

Matchers

- Matcher geben True zurück, wenn mind. ein „Match“

```
assert 'a' =~ /a/
assert !('a' =~ /b/)
```

Iteratoren und Enumerationen

- Iteratoren und Enumerationen mit weiteren Elementen sind true

```
assert [0].iterator()
assert ![].iterator.hasNext()
Vector v = [0] as Vector
Enumeration enumeration = v.elements()
assert Enumeration
enumeration.nextElement()
assert !enumeration.hasMoreElements()
```

Maps

- Non – Empty Maps sind true

```
assert ['one' : 1]
assert ![:]
```

Strings

Non – Empty Strings, GStrings, CharSequences sind true

```
assert 'a'
assert !''
def nonEmpty = a
assert "$nonEmpty"
def empty = ''
assert !"$empty"
```

Numbers

- Non – Zero Numbers sind true

```
assert 1
assert 3.5
assert !0
```

Object - References

- Non-Null Object-References sind true

```
assert new Object()
assert !null
```

asBoolean() - Methode

- Verhalten von Groovy kann durch asBoolean()-Methode verändert werden
- Dafür muss man asBoolean()-Methode implementieren

```groovy
class Color{

    String name

    boolean asBoolean(){
        name == 'green' ? true : false
    }
}

// ...

assert new Color(name: 'green')
assert !new Color(name: 'red')
```

Bedingte Operatoren

Not-Operator

- Der "Nicht"-Operator wird mit einem "!" dargestellt.

```
def wasbinIch = true
println !wasBinIch // Ausgabe "false"
```

Ternärer Operator

Statt:

```
if(string!=null && string.length()>0){
    result = 'Found'
} else {
    result = 'Not Found'
}
```

Kann man:

```
result = (string!=null && string.length()>0) ? 'Found' : 'Not Found'
```

Elvis - Operator

- Kurzform des Sternären Operators
- Wird verwendet, um einen Standardwert anzugeben, wenn Variable „null“ oder „falsly“ ist

```
def result = ausdruck1 ?: ausdruck2
```

→ausdruck1: Der Ausdruck, der ausgewertet wird, wenn dieser „truthy“ ist

→ausdruck2: Der Ausdruck, der ausgewertet wird, wenn ausdruck1 „null“ oder „falsly“ ist

Klassen

Definition einer Klasse

Klassen in Groovy sind ähnlich wie in Java, aber syntaktisch einfacher

```
class Person {
    String name
    int age
}
```

```groovy
class Person {
  String name
  int age

  //Default-Konstruktor
  def person = new Person()

  //benutzerdefinierter Konstruktor
  Person(String name, int age) {
      this.name = name
      this.age = age
    }
}
```

Konstruktoren

- Groovy fügt automatisch einen Standardkonstruktor hinzu.
- Benutzerdefinierte Konstruktoren

Standardkonstruktor mit Map

```groovy
class Person {
    String name
    int age

    // Map-Konstruktor
    Person(Map properties) {
        properties.each { key, value -> this."$key" = value }
    }
}

Person person = new Person(name: 'John', age: 30)
println(person.name)  // Ausgabe: John
println(person.age)   // Ausgabe: 30
```

Konstruktorüberladung

Standardkonstruktor +
benutzerdefinierter Konstruktoren

```groovy
class Person {
    String name
    int age

    Person() {
        // Standardkonstruktor
    }

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

def person1 = new Person()
println(person1.name)  // Ausgabe: null
println(person1.age)   // Ausgabe: 0

def person2 = new Person("Alice", 30)
println(person2.name)  // Ausgabe: Alice
println(person2.age)   // Ausgabe: 30
```

Eigenschaften

- Direkte Felddefinition.
- Automatische Getter- und Setter-Generierung.

```
Person person = new Person()
person.name = "John"
person.age = 30
println person.name  // John
```

Manuelle Getter und Setter

```groovy
class Fruits {
  private String fruitName
  private String fruitColor

  def setFruitName(String name) {
    fruitName = name
  }

  def getFruitName() {
    return "The fruitname is $fruitName"
  }

  def setFruitColor(String color) {
    fruitColor = color
  }

  def getFruitColor() {
    return "The color is $fruitColor"
  }

  static void main(args) {
    Fruits apple = new Fruits()
    apple.setFruitName("apple")
    apple.setFruitColor("red")
  }
}
```

Methoden

Definition einer Methode

- Methoden können optional einen Rückgabetyp haben

```
class Calculator {
  int add(int a, int b) {
    return a + b
  }
}
```

Methoden ohne Klasse

- Methoden können direkt definiert und aufgerufen werden.

```
def printHello() {
  println "Hello..."
}

printHello()  // Hello...
```

Instanzmethoden

Methoden können auf Instanzvariablen zugreifen.

```
class Person {
    String name
    int age

    // Instanzmethode, um die Person vorzustellen
    def introduce() {
        println("Hello, my name is ${name} and I am ${age} years old.")
    }
}

Person person = new Person(name: 'Alice', age: 30)
person.introduce()
// Hello, my name is Alice and I am 30 years old.
```

Statische Methoden

```
class MathUtils {
  static int add(int a, int b) {
    return a + b
  }

  static void main(String[] args) {
    int result = MathUtils.add(5, 10)
    println("Sum is $result")  // Sum is 15
  }
}
```

Dynamische Methoden

Methoden können zur Laufzeit hinzugefügt werden

```
class DynamicExample {}

DynamicExample.metaClass.sayHello = {-> println "Hey"}
def example = new DynamicExample()
example.sayHello()  // Hey
```

Expando

Dynamisches Hinzufügen von Methoden und Eigenschaften

```groovy
def expando = new Expando()
expando.name = "Groovy"
expando.sayHello = { -> println "Hello from $name" }
expando.sayHello()  // Hello from Groovy
```

```groovy
class Greeter {
  void greet(String name = "World") {
    println "Hello, $name!"
  }
}

def sum(int a = 10, int b = 3) {
  println "Sum is "+(a+b)
}

sum()  // Sum is 13
```

Defaultparameter

Methodenparameter können Standardwerte haben

Closures

Closures können auf Variablen im umgebenden Kontext zugreifen

```
def greet = { String name -> println "Hello, $name!" }
greet.call("John")
```

Closure – Referenzierung und Rückgabewerte

Referenzierung von Variablen und Rückgabewerten

```
def createCounter() {
    def count = 0
    return { ->
        count += 1
        return count
    }
}
```

Closure als Parameter

```groovy
def performOperation(int x, Closure operation) {
  return operation(x)
}

def closure = { y -> y * 2 }
def result = performOperation(5, closure)
println(result)  // 10
```

Closure aufrufen

```
def greet = { name -> return "Hello, ${name}!" }
println(greet("Alice"))  // Hello, Alice!
println(greet.call("Bob"))  // Hello, Bob!
```

Closures auf Maps und Listen

```
def myMap = [ 'subject': 'groovy', 'topic': 'closures']
println myMap.each { it }

def myList = [1, 2, 3, 4, 5]
println myList.find { item -> item == 3 }  // 3
println myList.findAll { item -> item > 3 }  // [4, 5]
println myList.any { item -> item > 5 }  // false
println myList.every { item -> item > 3 }  // false
println myList.collect { item -> item * 2 }  // [2, 4, 6, 8, 10]
```

Methodenverkettung

Durch das Rückgeben von this kann man Methodenaufrufe verketten

```groovy
class FluentPerson {
    String name
    int age

    FluentPerson setName(String name) {
        this.name = name
        return this
    }

    FluentPerson setAge(int age) {
        this.age = age
        return this
    }
}

// Erstellen einer neuen FluentPerson-Instanz und Methodenverkettung
def person = new FluentPerson()
    .setName("Alice")
    .setAge(30)

println "Name: ${person.name}, Age: ${person.age}"
// Ausgabe: Name: Alice, Age: 30
```

Mixin

```
class ExtraMethods {
  String shout(String str) {
    return str.toUpperCase()
  }
}

@Mixin(ExtraMethods)
class MyClass {}

def myObject = new MyClass()
println myObject.shout("hello")  // HELLO
```

Exception Handling

Checked Exception

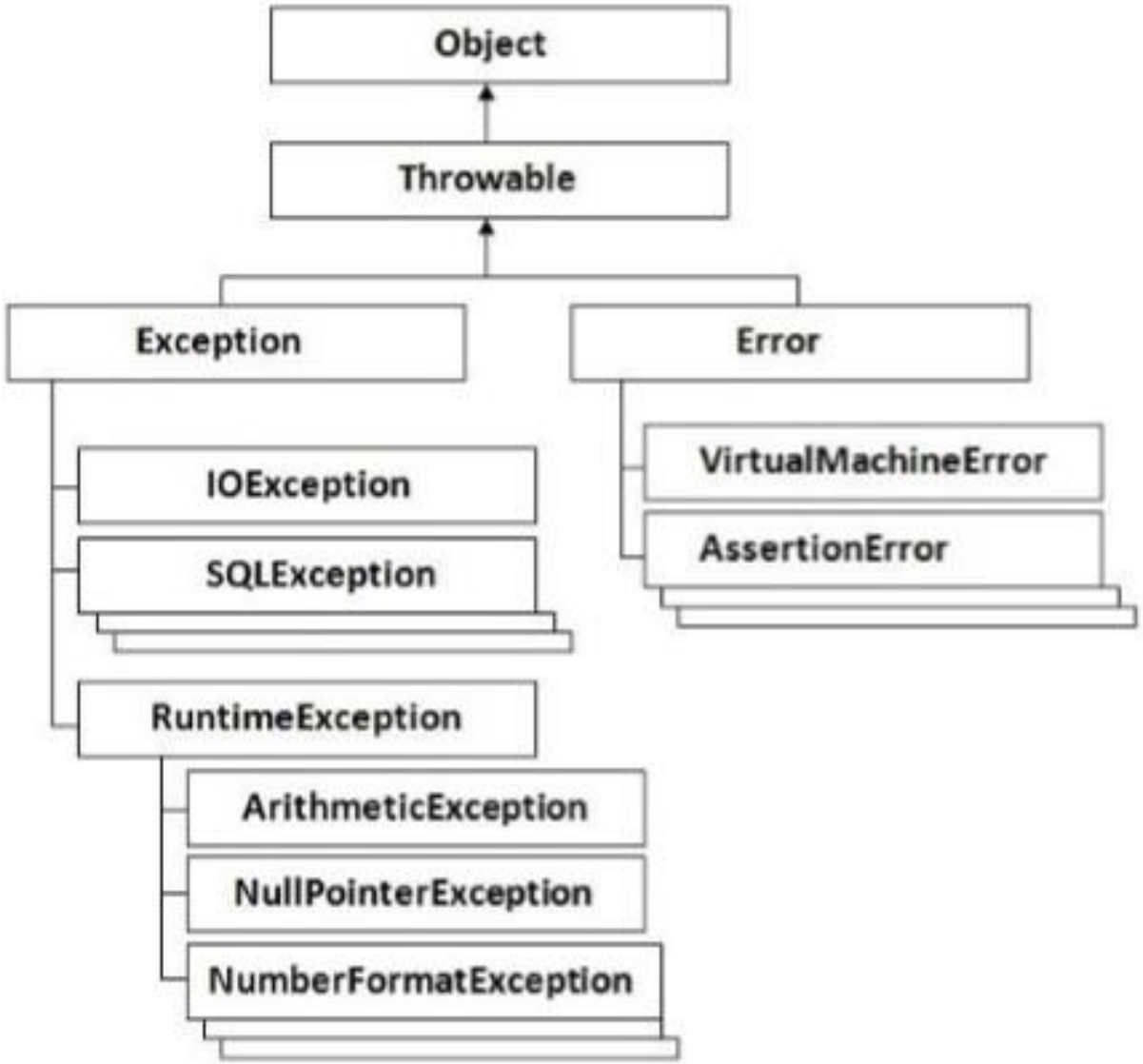
```
class Example {
   static void main(String[] args) {
      File file = new File("E://file.txt");
      FileReader fr = new FileReader(file);
   }
}
```

```
Caught: java.io.FileNotFoundException: E:/file.txt (No such file or directory)
java.io.FileNotFoundException: E:/file.txt (No such file or directory)
    at Example.main(jdoodle.groovy:4)
```

Unchecked Exception

```
class Example {
    static void main(String[] args) {
        def arr = new int[3];
        arr[5] = 5;
    }
}
```

```
Caught: java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3
    at Example.main(jdoodle.groovy:4)
```


Object
Throwable
Exception
Error
IOException
SQLException
RuntimeException
ArithmeticException
NullPointerException
NumberFormatException
VirtualMachineError
AssertionError

Caching Exceptions

```
class Example {
   static void main(String[] args) {
      try {
         def arr = new int[3];
         arr[5] = 5;
      } catch(Exception ex) {
         println("Catching the exception");
      }

      println("Let's move on after the exception");
   }
}
```

Ausgabe:

```
Catching the exception
Let's move on after the exception
```

Multiple Catch Blocks

```
class Example {
   static void main(String[] args) {
      try {
         def arr = new int[3];
         arr[5] = 5;
      }catch(ArrayIndexOutOfBoundsException ex) {
         println("Catching the Array out of Bounds exception");
      }catch(Exception ex) {
         println("Catching the exception");
      }

      println("Let's move on after the exception");
   }
}
```

Ausgabe:

```
Catching the Array out of Bounds exception
Let's move on after the exception
```

Finally Block

```
class Example {
   static void main(String[] args) {
      try {
         def arr = new int[3];
         arr[5] = 5;
      } catch(ArrayIndexOutOfBoundsException ex) {
         println("Catching the Array out of Bounds exception");
      }catch(Exception ex) {
         println("Catching the exception");
      } finally {
         println("The final block");
      }

      println("Let's move on after the exception");
   }
}
```

Ausgabe:

```
Catching the Array out of Bounds exception
The final block
Let's move on after the exception
```

Datei I/O

Dateien lesen

```
import java.io.File
class Example {
   static void main(String[] args) {
      new File("E:/Example.txt").eachLine {
         line -> println "line : $line";
      }
   }
}
```

Ausgabe:

```
line : Example1
line : Example2
```

Lesen des Inhalts einer Datei als Ganzes

```
class Example {
    static void main(String[] args) {
        File file = new File("E:/Example.txt")
        println file.text
    }
}
```

Ausgabe:

```
line : Example1
line : Example2
```

Schreiben zu Dateien

```
import java.io.File
class Example {
    static void main(String[] args) {
        new File('E:/','Example.txt').withWriter('utf-8') {
            writer -> writer.writeLine 'Hello World'
        }
    }
}
```

Die Größe einer Datei

```
class Example {
    static void main(String[] args) {
        File file = new File("E:/Example.txt")
        println "The file ${file.absolutePath} has ${file.length()} bytes"
    }
}
```

Testen, wenn eine Datei ein Verzeichnis ist

```
class Example {
    static void main(String[] args) {
        def file = new File('E:/')
        println "File? ${file.isFile()}"
        println "Directory? ${file.isDirectory()}"
    }
}
```

Ausgabe:

```
File? false
Directory? True
```

Erstellen eines Verzeichnisses

```
class Example {
    static void main(String[] args) {
        def file = new File('E:/Directory')
        file.mkdir()
    }
}
```

Datenstrukturen

Ranges

```
1..10 // Inklusive 10
1..<10 // Exklusive 10
'a'..'x' // Kann auch aus Character bestehen
10..1 // Kann auch absteigende Reihenfolge haben
'x'..'a' // Character können auch absteigende Reihenfolge haben
```

Arrays

- Von allen Typen können Arrays erstellt werden
- Größe muss angegeben werden, sind nicht dynamisch

```
int[] intArray = new int[4];
intArray[2] = 2
println intArray // [0,0,2,0]
```

Lists

Geordnete Sammlung von Elementen

```
def list = [0,1,2,3]
println list // [0,1,2,3]
println list[0] // [0]

list << 4 // Hinzufügen eins Elements am Ende der Liste
println list // [0,1,2,3,4]

list.add(0,20) // [Index, Wert]
println list // [20,0,1,2,3,4]
```

Maps

- Ungeordnete Collection von Object-Referenzen

```
def map = [name: "Peter", alter: 81]
```

- Zugriff auf Maps:

```
println map.name // "Peter"
println map["alter"] // 81
```

Maps

- Hinzufügen von Schlüsselwerten:

```
map["Auto"] = "Bugatti"
println map["Auto"] // "Bugatti"
```

Maps

- Entfernen von Elementen:

```
def map [hello: "World", erde: "Rund", eins: "Zwei"]
println map // [hello: "World", erde: "Rund", eins: "Zwei"]
def newMap = map.minus([hello: "World"])
println newMap // [erde: "Rund", eins: "Zwei"]
```

- (minus()-Methode nimmt eine Map und gibt neue Map mit entfernten Schlüssel-Wert Paaren zurück)

Maps

Map ohne Elemente initialisieren

```
def emptyMap = [:]
```

Sets

Ungeordnete Sammlung von eindeutigen Elementen

```
def mySet = [1,1,1,2,3,3,4,5,5,5,5] as Set
println mySet //[1,2,3,4,5]
```

Sets

- Hinzufügen von Elementen:

```
mySet << 6
println mySet // [1,2,3,4,5,6]
// oder
mySet.add(7)
println mySet // [1,2,3,4,5,6,7]
```

- Entfernen eines Elements

```
mySet.remove(1)
println mySet // [2,3,4,5,6,7]
```

Testen in Groovy

Testen in Groovy

- Support für JUnit 5 (und älter)
- Liefert eigenen Satz von Testmethoden, um die testgetriebene Programmierung zu erleichtern

  → Power Assertions

  → Spock

Power Assertions

- Sind im Gegensatz zu den Java Assertions automatisch aktiviert
- Erleichtern die Fehlersuche und das Debugging
- Wenn eine Assertion fehlschlägt:

→Zeigt die Power Assertion den Ausdruck mitsamt Werten in einer übersichtlichen, mehrzeiligen Darstellung an.

→Entwickler sieht genau, welcher Teil des Ausdrucks die Probleme verursacht

Beispiel: Power Assertions

```
def a = 1
def b = 2
def c = 4

//Ausdruck liefert 9 statt 10
assert a + b * c == 10
```

```
Assertion failed:

assert a + b * c == 10
       | | | | | |
       1 9 2 8 4 false

    at test.run(test.groovy:5)

[Done] exited with code=1 in 0.538 seconds
```

Achtung

- Es kann während der Auswertung von Power Assertion zu inkonsistenten Fehlermeldungen kommen

→ Das liegt daran, dass bei den Power Assertions nur Referenzen auf die Werte gespeichert werden

→ Werden die Werte also durch eine Methode verändert, wird der vorherige Stand in der Fehlernachricht nicht mehr angezeigt

Beispiel:

```
def getLastAndRemove(list) {
    return list.remove(list.size() - 1)
}
def list = [1, 2, 3]
assert getLastAndRemove(list) == 4
```

Ausgabe

- Es wird nicht der Stand von list angezeigt, wie er vor der Auswertung der Power Assertion war, sondern wie er danach war.

```
Assertion failed:

assert getLastAndRemove(list) == 4
       |                |     |
       3                |     false
                        [1, 2]

    at test.run(test.groovy:7)
```

```
assert [[1,2,3,3,3,3,4]].first().unique() == [1,2,3]
```

```
Assertion failed:

assert [[1,2,3,3,3,3,4]].first().unique() == [1,2,3]
       |                |       |        |
       |                |       |        false
       |                |       [1, 2, 3, 4]
       |                [1, 2, 3, 4]
       |
       at test.run(test.groovy:1)
```

Komplexes Beispiel

Spock

- Spezifikationsgetriebene Syntax
- Lesbare und verständlichere Tests
- Sowohl in Java als auch in Groovy verfügbar, wurde jedoch in Groovy geschrieben
- Benutzung durch den Import von spock.lang.Specification

  →Testklassen müssen von dieser Klasse erben

```
class Calculator {
    int add(int a, int b) {
        return a + b
    }
}
```

Beispiel Calculator

Ausgabe

- Name der Methode ist ein String, der die Erwartungen an den Test beschreibt
- Das Schlüsselwort “setup“ beschreibt die Voraussetzungen / Ausgangssituation für den Test (vgl.: @BeforeEach in JUnit)
- “when“: Die zu testende Methode
- “then“: Das erwartete Verhalten

```
import spock.lang.Specification

class CalculatorSpec extends Specification {
    //1
    def "addition of two numbers"() {
        //2
        setup: "a calculator"
            def calculator = new Calculator()

        //3
        when: "the numbers 2 and 3 are added"
            def result = calculator.add(2, 3)

        //4
        then: "the result is 5"
            result == 5
    }
}
```