

Klassen und Methoden

1. Klassen
2. Methoden

Definition einer Klasse

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

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

Eigenschaften

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

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

Möglichkeit, Getter und Setter zu überschreiben.

Manuelle Getter und Setter

```
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")  
    }  
}
```

Konstruktoren

- Groovy fügt automatisch einen Standardkonstruktor hinzu.
- Benutzerdefinierte Konstruktoren

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

Standardkonstruktor mit Map

```
Person person = new Person(name: 'John', age: 30)
```

Methoden

Definition einer Methode

- Methoden können optional einen Rückgabetyt 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 Method {
    static void main(args) {
        Method myFunc = new Method()
        myFunc.myMethod()
    }

    def myMethod() {
        println("I am inside my method")
    }
}
```

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

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

Dynamisches Hinzufügen
von Methoden und
Eigenschaften

Defaultparameter

Methodenparameter können Standardwerte haben

```
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
```

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 Rückgabewerte

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


Closure als Parameter

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

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

Wie ruft man eine Closure auf

```
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,
```

Methodenverkettung

```
class FluentPerson {
    String name
    int age

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

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

def person = new FluentPerson().setName("John").setAge
```

Mixin

```
class ExtraMethods {  
    String shout(String str) {  
        return str.toUpperCase()  
    }  
}  
  
@Mixin(ExtraMethods)  
class MyClass {}  
  
def myObject = new MyClass()  
println myObject.shout("hello") // HELLO
```