Kotlin

Das bessere Java?

Kotlin

- Open Source
- Statisch typisiert
- Compile targets:
  - JVM (Java 1.6 Bytecode), Android!
  - JavaScript
- Version 1.0 Anfang 2016
  - Aktuell: 1.0.4
- JetBrains

Basics

```
int count;
```

```
var count : Int
```

Basics

```
int count = 10;
```

```
var count : Int = 10
```

Basics

```
int count = 10;
```

```
var count = 10
```

Basics

```
int    count = 10;
Button okay  = new Button();
```

```
var count = 10
var okay  = Button()
```

Basics

```
final int count = 10;
```

```
val count = 10
```

Basics - Properties

```
public final class Item {
    private final int    id;
    private       String name;

    public Item(int id, String name) {
        this.id   = id;
        this.name = name;
    }

    public int getId() { return id; }

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

Basics - Properties

```
public final class Item {
    private final int    id;
    private       String name;

    public Item(int id, String name) {
        this.id   = id;
        this.name = name;
    }

    public int getId() { return id; }

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

Basics - Properties

```
public final class Item {
    private final int    id;
    private       String name;

    public Item(int id, String name) {
        this.id   = id;
        this.name = name;
    }

    public int getId() { return id; }

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

```
class Item(id:Int, name:String) {
    val id   = id
    var name = name
}
```

Basics - Properties

```
public final class Item {
   private final int    id;
   private       String name;

   public Item(int id, String name) {
       this.id   = id;
       this.name = name;
   }

   public int getId() { return id; }

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

```
class Item(val id    : Int,
           var name  : String)
```

Basics - Properties

```
class Item(val id   : Int,
           var name : String)

val item = Item(10, "")
item.name = "Item 10"
println(item.name)
item.id = 20
```

Basics - Properties

```
class Item(id:Int, name:String) {
    val id = id
        get() { return field }
    var name = name
        set(name:String) { field = name }
}
```

Basics - Properties

```
class Item(id:Int, name:String) {
    val id = id
        get() = field
    var name = name
        set(name:String) { field = name }
}
```

Classes

- Einfachvererbung
- `class`, `interface`, `enum class`
- Default Modifier: `public final`
- Überschreiben erlauben: `open`
- `data class`
- `sealed class`

null


SIMPLY EXPLAINED
ARE YOU REALLY SURE THAT THIS VARIABLE CAN NEVER EVER BE NULL?
OF COURSE!!!
NullPointerException

null

```
public interface ItemRepository {
    Item create(String name);
    Item find(int id);
}
```

null

```
public interface ItemRepository {
    Item create(String name);
    Optional<Item> find(int id);
}
```

null

```
public interface ItemRepository {
    Item create(String name);
    Optional<Item> find(int id);
}
```

```
Optional<Item> item = repo.find(10);

item.ifPresent(i -> System.out.println(i.getName()));
System.out.println(item.orElse(Item.NULL).getName());
```

null

```
public interface ItemRepository {
    @NotNull Item create(String name);
    Item find(int id);
}
```

null

```
interface ItemRepository {
    fun create(name:String) : Item
    fun find(id:Int) : Item?
}
```

null

```
interface ItemRepository {
    fun create(name:String) : Item
    fun find(id:Int) : Item?
}
```

```
println(repo.find(10).name)
println(repo.find(10)?.name)
println(repo.find(10)!!.name)
println(repo.find(10)?.name ?: "no name")
```

Operatoren

```
class Vector(val a:Double, val b:Double) {
   operator fun plus(v:Vector)   : Vector { return Vector(a+v.a, b+v.b) }
   operator fun minus(v:Vector)  : Vector { return Vector(a-v.a, b-v.b) }
   operator fun times(v:Double)  : Vector { return Vector(a*v,   b*v  ) }
   operator fun div(v:Double)    : Vector { return Vector(a/v,   b/v  ) }
   operator fun times(v:Vector)  : Double { return a*v.a + b*v.b       }
   operator fun mod(v:Vector)    : Double { return a*v.b - v.a*b       }
   operator fun get(i:Int)       : Double { return when (i){ 0->a 1->b }}
}
```

Operatoren

```
class Vector(val a:Double, val b:Double) {
    operator fun plus(v:Vector)   : Vector { return Vector(a+v.a, b+v.b) }
    operator fun minus(v:Vector)  : Vector { return Vector(a-v.a, b-v.b) }
    operator fun times(v:Double)  : Vector { return Vector(a*v,   b*v  ) }
    operator fun div(v:Double)    : Vector { return Vector(a/v,   b/v  ) }
    operator fun times(v:Vector)  : Double { return a*v.a + b*v.b       }
    operator fun mod(v:Vector)    : Double { return a*v.b - v.a*b       }
    operator fun get(i:Int)       : Double { return when (i){ 0->a 1->b }}
}
```

```
val t = 0.1
val m = 100.0
val F = Vector(10.0, 10.0)
val a = F / m
val s = a / 2 * t*t + v0 * t + s0
```

Operatoren

```
class Vector(val a:Double, val b:Double) {
    operator fun plus(v:Vector)   : Vector { return Vector(a+v.a, b+v.b) }
    operator fun minus(v:Vector)  : Vector { return Vector(a-v.a, b-v.b) }
    operator fun times(v:Double)  : Vector { return Vector(a*v,   b*v  ) }
    operator fun div(v:Double)    : Vector { return Vector(a/v,   b/v  ) }
    operator fun times(v:Vector)  : Double { return a*v.a + b*v.b       }
    operator fun mod(v:Vector)    : Double { return a*v.b - v.a*b       }
    operator fun get(i:Int)       : Double { return when (i){ 0->a 1->b }}
}
```

```
val t = 0.1
val m = 100.0
val F = Vector(10.0, 10.0)
val a = F / m
val s = a / 2 * t*t + v0 * t + s0
println(s+"m ")
```

String Interpolation

```
class Vector(val a:Double, val b:Double) {
    operator fun plus(v:Vector)   : Vector { return Vector(a+v.a, b+v.b) }
    operator fun minus(v:Vector)  : Vector { return Vector(a-v.a, b-v.b) }
    operator fun times(v:Double)  : Vector { return Vector(a*v,   b*v  ) }
    operator fun div(v:Double)    : Vector { return Vector(a/v,   b/v  ) }
    operator fun times(v:Vector)  : Double { return a*v.a + b*v.b       }
    operator fun mod(v:Vector)    : Double { return a*v.b - v.a*b       }
    operator fun get(i:Int)       : Double { return when (i){ 0->a 1->b }}
}
```

```
val t = 0.1
val m = 100.0
val F = Vector(10.0, 10.0)
val a = F / m
val s = a / 2 * t*t + v0 * t + s0
println("$s m")
```

String Interpolation

```
val t = 0.1
val m = 100.0
val F = Vector(10.0, 10.0)
val a = F / m

println("${a / 2 * t*t + v0 * t + s0} m")
```

Classes - extension methods

```
operator fun BigDecimal.plus(v:BigDecimal) : BigDecimal {
    return this.add(v)
}
```

```
val one = BigDecimal.ONE
val ten = BigDecimal.TEN
val x   = one + ten
```

Classes - extension methods

```
fun Any?.toString() : String {
    if( this==null ) {
        return "null"
    }
    return this.toString()
}

null.toString()
```

Collection API

```
int[]          array = {1,2,3};
List<Integer> list  = Arrays.asList(1,2,3);
```

```
val array1 : Array<Int> = arrayOf(1,2,3)
val array2 = arrayOf(1,2,3)
val list1  = listOf(1,2,3)
val list2  = mutableListOf(1,2,3)
```

Collection API

- Array, List, Set, Map
- Mutable vs Immutable
- Default: normale Java Collection Klassen
  - ArrayList, HashSet, HashMap
- Zusätzliche extension methods am Collection Interface

```
list
   .filter { i -> i%2==0 }
   .map    { it.toString() }
   .first()
```

Generics - Java

```
interface Collection<E> {
   void addAll(Collection<E> source);
}

Collection<Object> receiver = ...;
Collection<Item>   sender   = ...;

receiver.addAll(sender);
```

Generics - Java

```
interface Collection<E> {
   void addAll(Collection<? extends E> source);
}

Collection<Object> receiver = ...;
Collection<Item>   sender   = ...;

receiver.addAll(sender);
```

Generics - Java

```
interface Collection<E> {
   void copyTo(Collection<? super E> target);
   void addAll(Collection<? extends E> source);
}

Collection<Object> receiver = ...;
Collection<Item>   sender   = ...;

sender.copyTo(receiver);
```

Generics - Java

- PECS
  - Producer - extends
  - Consumer - super

Generics - Java

```
interface Source<T> {
    T next();
}

void foo(Source<String> strings) {
    Source<Object> objs = strings;
}
```

Generics - Java

```
interface Source<T> {
    T next();
}

void foo(Source<String> strings) {
    Source<? extends Object> objs = strings;
}
```

Generics - Kotlin

```
interface Source<out T> {
    T next()
}

fun foo(strings:Source<String>) {
    val objs : Source<Any> = strings
}
```

Generics - Kotlin

-

```
interface Comparable<in T> {
    fun compareTo(v:T) : Int
}

fun foo(x:Comparable<Number>) {
    val y : Comparable<Double> = x
}
```

Type-safe builder

```
html {
    head { }
    body { }
}
```

Type-safe builder

```
html {
    head { }
    body { }
}
```

```
class HTML {}

fun html(init:HTML.()->Unit) : HTML {
    val html = HTML()
    html.init()
    return html
}
```

Type-safe builder

```
class Head
class Body
class HTML {
    fun head(init:Head.()->Unit) : Head {
        val head = Head()
        head.init()
        return head
    }
}
fun html(init:HTML.()->Unit) : HTML {
    val html = HTML()
    html.init()
    return html
}
```

Type-safe builder

- Beispiele:
  - reakt (Kotlin Bindings für react Framework)
  - Gradle 3.0
  - JavaFX/Swing-Builder

JavaScript / native

- Noch in Entwicklung
- Vergleichbar mit GWT Transpiler
- JavaScript Libraries brauchen typisierte Kotlin API
- Resultierendes JavaScript recht verständlich
- Buildsystem & Abhängigkeiten noch schwierig
  - siehe reakt für funktionierendes Beispiel

Fazit

- Sehr viele syntaktische Verbesserungen
- Kotlin ändert einige Dinge grundlegend
  - Das ist mit Java kaum möglich
- Ergebnis ist normaler Java Bytecode
  - Integration in bestehende Systeme
- Android!! ;-)
- JavaScript als Target