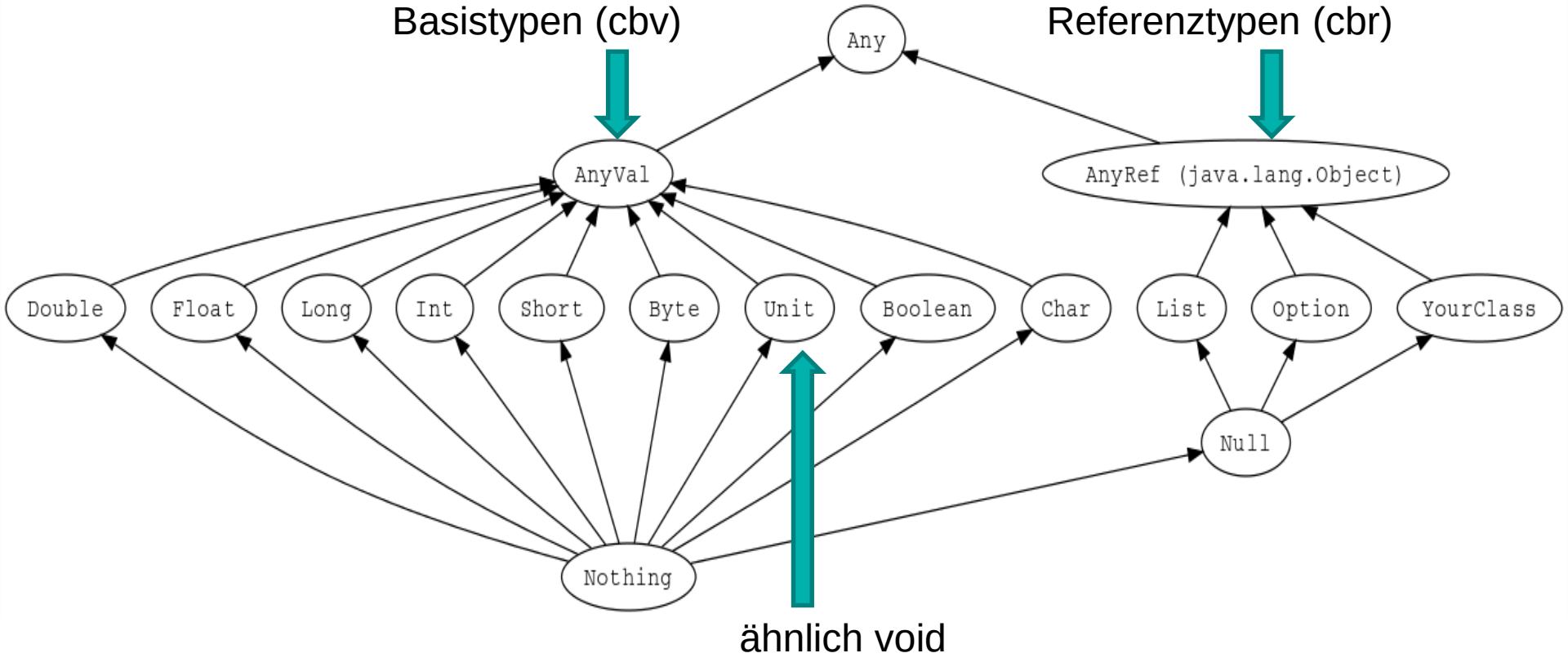
FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Typen, Ausdrücke & Funktionen

Basis Typen in Scala

- alles Objekte, auch Basistypen



Komplexere Typen in Scala

- Arrays, siehe Einleitung
- Listen: ähnliche Notation wie in Prolog, jedoch ohne Klammern

  Liste kann leer sein: `Nil` Prolog: `[]`

  Liste nicht leer: `head::tail` `[head|tails]`

Bsp.: `1::2::3::Nil`

- weitere Typen, wie Binärbäume später

Typinferenz:
Typen können weggelassen werden, wenn Sie von Scala selbst bestimmt werden

Ausdrücke in Scala

```
scala> 87 + 145
unnamed0: Int = 232

scala> 5 + 2 * 3
unnamed1: Int = 11

scala> "hello" + " world!"
unnamed2: java.lang.String = hello world!
```

werden später

- Funktionsaufrufe
- If_Else
- Konstanten
- Variablen

enthalten

Konstanten und Variablen in Scala
Variablen funktional nicht verwenden

- Konstanten

  EBNF: `val x : (T)? ( = e )?`

// definiert eine Konstante `x` des Typs `T` mit optionaler Zuweisung eines Ausdruckwertes (immutable = kann nur einmal gesetzt werden)

Bsp.: `val x: Int = 3+5`

`-> x: Int = 8`

- Variablen

  EBNF: `var x : (T)? ( = e )?`

// definiert eine Variable `x` des Typs `T` mit optionaler Zuweisung eines Ausdruckwertes (Wert kann im Laufe des Programms verändert werden)

Bsp.: `var x: Int = 3+5`

```
-> x: Int = 8
x = 3 * x -> 24
```

Funktionen

- in Mathematik: Abbildungen von einer Menge in andere
- Definition durch:
  - Angabe des Typs (Signatur): `add: int x int -> int`
  - Abbildungsvorschrift: `(x,y) -> x+y`

    oder `add(x,y) = x+y`
- Unterschiede zur Relation:
  - Ausgabe bei Eingabe eindeutig: `vater: x -> y oder z nicht`
  - können verschachtelt auftreten: `add(x,y,z) = add(x,add(y,z))`
- in imperativen Sprachen zur
  - Steuerung des Kontrollflusses
  - Verhinderung von Redundanzen
  - können bei einer Eingabe verschiedene Ausgaben liefern (Seiteneffekte)

Funktionen in FuPS

- Methode, die nicht an Klasse gebunden ist
- können in Programmen aufgerufen werden
- und in Ausdrücken ausgewertet werden
- z.B.: `fib(n-1) + fib(n-2)`

  -> flexibler als in OOS

Funktionen in Scala

- Funktionsdefinitionen:
  - Typdeklaration und Implementierung zusammengefasst
  - EBNF:
    ```
    def name (parameters?)(: return type)?  = expression
    ```
  - definiert:
    - Funktionsname
    - mit (möglicherweise 0) Parametern
    - einem optionalem Rückgabetyp
    - Implementierung = Ausdruck über
      - Parametern
      - Funktionsaufrufen
      - Operationen
      - if_else
      - (Pattern Matching)

Funktionen in Scala
Beispiele

- Quadrat einer Int-Zahl
  - Definition: `def square(x: Int): Int = x*x`
  - Aufruf: `square(5)`
  - liefert `Int = 25`

- Summe zweier Int-Zahlen
  - Definition: `def add(x: Int, y: Int): Int = return x+y`
  - Aufruf: `add(5,7)`
  - liefert `Int = 12`

- kann Rückgabetyp und return weglassen
- Compiler -> Typ und Wert aus letztem Statement

```
def sum(l: Int, r: Int) = l + r
```

- **Konvention: Typ immer angeben**

FunScala-Programme

Definition:

Ein (rein funktionales) Scala-Programm ist eine endliche Aufzählung von

- Konstanten
- Funktionsdefinitionen