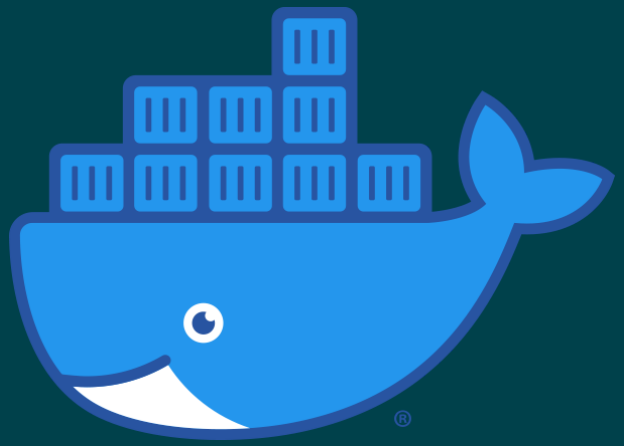
JAVA TO GO

GOOGLE GO FÜR JAVA ENTWICKLER



467.882 ZEILEN CODE

2.130 CONTRIBUTORS

3 GRÜNDE FÜR GO

1. Einfach
2. Mächtig
3. Langweilig

HELLO GOPHER

```go
package main

import "fmt"

func main() {
  fmt.Println("Hello Gopher!")
}
```

Ausführen

```
go build hellogopher.go // 1. Code kompilieren
./hellogopher           // 2. Binary ausführen
```

```
go run hellogopher.go   // Code kompilieren und ausführen
```

ENTWICKLUNG

JetBrains
GoLand

Visual
Studio Code

Vim Go

5 FAKTEN ZU GO

1. statisches Typsystem
2. Garbage Collection
3. keine Vererbung
4. Concurrency eingebaut
5. native Ausführung
   Linux, Win, z/OS, 386, amd64, ARM, wasm, ...

VARIABLEN, SLICES, SCHLEIFEN

```go
// Variable
var frank string = "Frank"
claire := "Claire"

// Array (fixe Länge)
namesArray := [3]string{frank, claire, "Zoe"}

// Slice (variable Länge)
namesSlice := make([]string, 2)
namesSlice[0] = frank

// Schleife
for i, name := range namesSlice {
  fmt.Println("Hello " + name + "!")
}
```

STRUCT STATT KLASSE

```
1 type Congressman struct {
2   Name string
3 }
4
5 func main() {
6   c := Congressman{Name: "Peter Russo"}
7   fmt.Println("Hello " + c.Name + "!")
8 }
```

FUNCTION RECEIVER STATT INSTANZMETHODE

```
 1 type Congressman struct {
 2   Name string
 3 }
 4
 5 func (c Congressman) swearOathOfOffice() {
 6   fmt.Printf("I, %v, swear to serve the USA.", c.Name)
 7 }
 8
 9 func main() {
10   c := Congressman{Name: "Peter Russo"}
11   c.swearOathOfOffice();
12 }
```

INTERFACE

```go
type Greeter interface {
  greet()
}

func passBy(c1 Greeter, c2 Greeter)
  c1.greet()
  c2.greet()
}

func main() {
  c := Congressman{Name: "Frank U.'
  e := Enemy{}
  passBy(c, e)
}
```

```go
type Congressman struct {
    Name string
}

func (c Congressman) greet() {
  fmt.Println("Hello", c.Name)
}
```

```go
type Enemy struct{}

func (e Enemy) greet() {
 fmt.Println("Go to hell!")
}
```

ZU EINFACH?

STRUCT EMBEDDING STATT VERERBUNG

```
type Congressman struct {
  Name string
}

type President struct {
  Congressman  // Embedded

  NuclearWeaponCode string
}

func main() {
  p := President{NuclearWeaponCode: "123"}
  p.Name = "Frank Underwood"
  p.swearOathOfOffice();
}
```

FEHLER

```go
// Fehler als Rückgabewert
func (c Congressman) bribe(amount float64) error {
  if c.Name != "Peter Russo" {
      return errors.New("Not corrupt!")
  }
  c.AccountBalance += amount
  return nil
}

func main() {
  c := Congressman{Name: "Jackie Sharp", AccountBalance: -10.0}

  // Fehler behandeln
  err := c.bribe(5000.0)
  if err != nil {
      fmt.Printf("%v is not bribable.", c.Name)
  }
}
```

GENERICS

```
func printSliceOfInts(numbers []int) {
    for _, num := range numbers {
        fmt.Print(num, " ")
    }
}
```

Generics kommen in Go 2

```
fun                          (strings []string) {
                  := range strings {
        fmt.Print(num, " ")
    }
}
```

MÄCHTIG

CONCURRENCY

GOROUTINE
leichtgewichtiger Thread

CHANNEL
Kanal für Nachrichten

GOROUTINE

```
1 func HelloCongressman(name string) {
2   fmt.Println("Hello Congressman", name)
3 }
4
5 func main() {
6   go HelloCongressman("Russo")
7 }
```

GOROUTINE MIT SLEEP

```
1 func HelloCongressman(name string) {
2   fmt.Println("Hello Congressman", name)
3 }
4
5 func main() {
6   go HelloCongressman("Russo")
7
8   time.Sleep(5 * time.Millisecond)
9 }
```

GOROUTINE MIT WAIT GROUP

```
 1 func HelloCongressman() {
 2     fmt.Println("Hello Congressman", name)
 3 }
 4
 5 func main() {
 6   var waitGroup sync.WaitGroup
 7   waitGroup.Add(1)
 8   go func() {
 9       HelloCongressman("Russo")
10       waitGroup.Done()
11   }()
12   waitGroup.Wait()
13 }
```

GOROUTINE MIT WAIT GROUP UND DEFER

```
 1 func HelloCongressman() {
 2     fmt.Println("Hello Congressman", name)
 3 }
 4
 5 func main() {
 6   var waitGroup sync.WaitGroup
 7   waitGroup.Add(1)
 8   go func() {
 9       defer waitGroup.Done()
10       HelloCongressman("Russo")
11   }()
12   waitGroup.Wait()
13 }
```

CHANNEL

```go
func main() {
  money := make(chan int)
  go Congressman(money)

  // Nachricht senden
  money <- 100
}

func Congressman(money chan int) {
  // Nachricht empfangen
  amount := <-money

  fmt.Println("Received", amount, "$!")
}
```

CHANNEL MIT SELECT

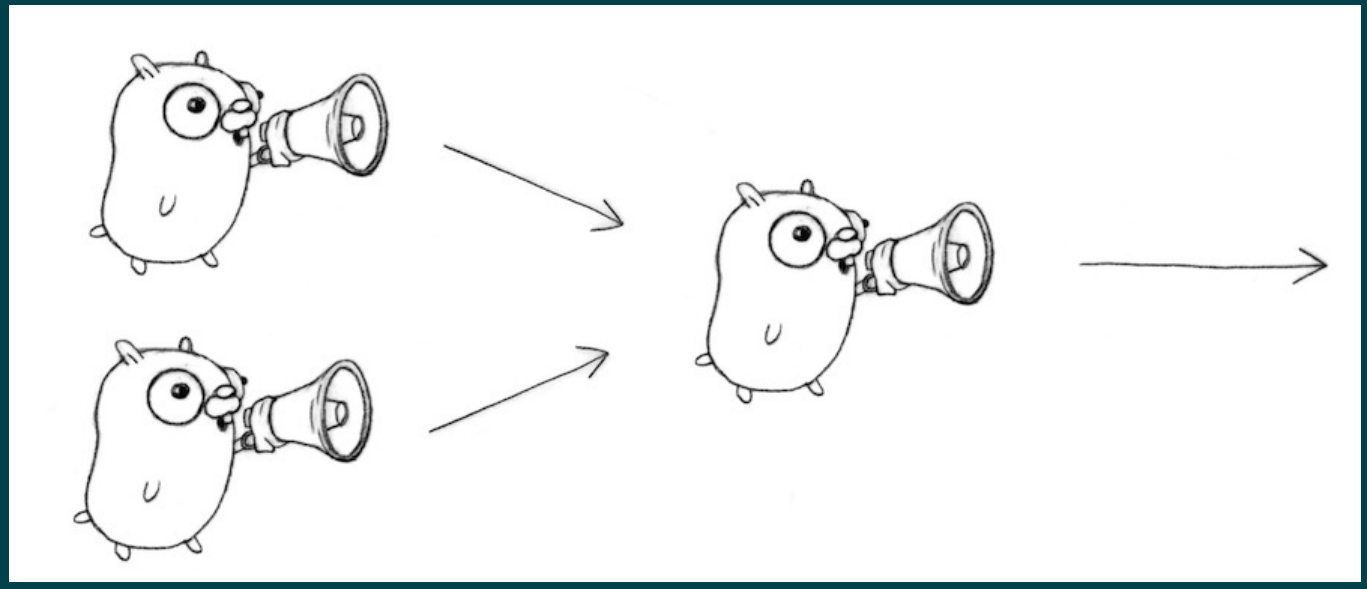
```
 1 func main() {
 2   money := make(chan int)
 3   go Congressman(money)
 4
 5   // Nachricht senden
 6   money <- 100
 7 }
 8
 9 func Congressman(money chan int) {
10   select {
11   case amount := <-money:
12     fmt.Println("Received", amount, "$!")
13   }
14 }
```

CHANNELMIT TIMEOUT

```
 1 func main() {
 2   money := make(chan int)
 3   go Congressman(money)
 4
 5   time.Sleep(2 * time.Second)
 6 }
 7
 8 func Congressman(money chan int) {
 9   select {
10   case amount := <-money:
11     fmt.Println("Received", amount, "$!")
12   case <-time.After(1 * time.Second):
13     fmt.Println("Got nothing ...!")
14   }
15 }
```

CONCURRENCY

mit Goroutinen und Channels



STANDARDBIBLIOTHEK

- Tests
- HTTP(2) Server und Router
- JSON
- Logging

EINFACH

LANGWEILIG

- Variablen, Slices, Schleifen
- Struct

ZU EINFACH?

- Struct Embedding
- Fehler
- Generics

MÄCHTIG

- Interface
- Goroutine
- Channel
- Standardbibliothe

DEMO


GO!
GO!

JAN STAMER

Solution Architect

jan.stamer@comdirect.de