



Eine Einführung in go

Hauke Stieler

16. Januar 2018

Fachbereich Informatik der Universität Hamburg

Agenda

Thank to Fred, on [whose slides](#) I was able to create these :)

Agenda

- some history
- basic features
- cool web stuff
- concurrency
- interfaces

Why go?

In 2007, three guys at Google were frustrated with the existing languages for writing server software:

- Compiling C++ was too slow
- Writing Java felt too verbose
- Aversion against inheritance and design patterns
- Getting concurrency right was hard

C++

```
1 // Within large projects, popular header files
2 // get included thousands of times and hence
3 // have to be recompiled over and over again
4 #include <iostream>
5 #include <string>
6 #include <vector>
```

C++

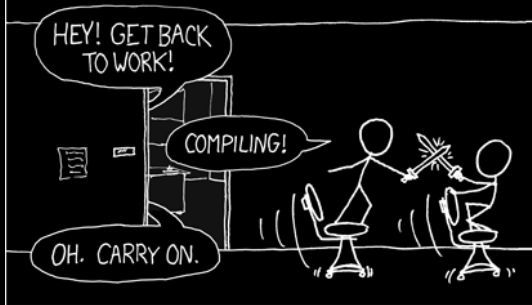
```
1 // Within large projects, popular header files
2 // get included thousands of times and hence
3 // have to be recompiled over and over again
4 #include <iostream>
5 #include <string>
6 #include <vector>
```

gcc copies specified file by `#include` recursively into source file. The same header file gets recompiled over and over again.

→ Rob Pike: [Public Static Void](#) at OSCON 2010

THE #1 PROGRAMMER EXCUSE
FOR LEGITIMATELY SLACKING OFF:

"MY CODE'S COMPILING."



Java

Let's do some Java.

Write a `public class Person` that does the following:

- store a string name
- store an int age

Simple, right?

Java

Let's do some Java.

Write a `public class Person` that does the following:

- store a string name
- store an int age

Simple, right?

NO :(

Java I

```
1 public class Person {
2     private String name;
3     private int age;
4
5     public Person(String name, int age) {
6         this.name = name;
7         this.age = age;
8     }
9
10    public String getName() {
11        return name;
12    }
13
14    public void setName(String name) {
15        this.name = name;
16    }
17 }
```

Java II

```
18     public int getAge() {
19         return age;
20     }
21
22     public void setAge(int age) {
23         this.age = age;
24     }
25
26     @Override
27     public String toString() {
28         return "Person [" + "name=" + name + ", " + "age=" +
29             ↪ age + "]";
30     }
31
32     @Override
33     public int hashCode() {
34         final int prime = 31;
```

Java III

```
34     int result = 1;
35     result = prime * result + age;
36     result = prime * result + ((name == null) ? 0 :
    ↪     name.hashCode());
37     return result;
38 }
39
40 @Override
41 public boolean equals(Object obj) {
42     if (this == obj)
43         return true;
44     if (obj == null)
45         return false;
46     if (getClass() != obj.getClass())
47         return false;
48     Person other = (Person) obj;
49     if (age != other.age)
```

Java IV

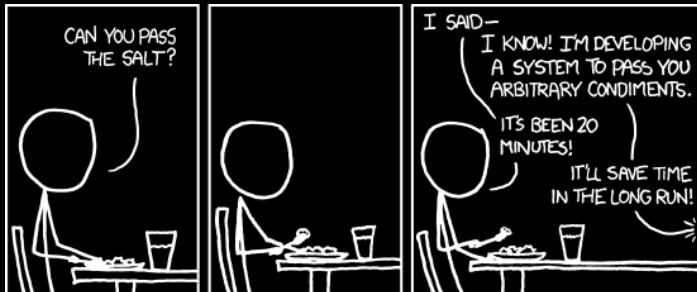
```
50         return false;
51     if (name == null) {
52         if (other.name != null)
53             return false;
54     } else if (!name.equals(other.name))
55         return false;
56     return true;
57 }
58 }
```

Initial design by 3 people with different backgrounds:

- Rob Pike (Concurrency)
- Robert Griesemer (Modules)
- Ken Thompson (Operating Systems)

All design decisions had to be agreed upon unanimously. Design team later joined by more people at Google.

- **simplicity**
- **simplicity**
- **simplicity**
- clean package model for fast compilation
- built-in concurrency based on CSP
- interfaces instead of inheritance
- no radical changes after Go 1.0



Hello world!

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("hello world")
7 }
```

- The import declaration imports entire packages
- All imported names must be qualified
- Uppercase names are visible to other packages
- Unused imports are compile-time errors!

Hello world!

Get the go compiler:

```
$ sudo apt-get install golang-go
```

```
$ sudo pacman -S go
```

...or download from <https://golang.org/dl>

Run the code¹:

```
$ go run hello.go
```

¹The `go run` command works for single files, not always for projects

Basics

Keywords:

<code>break</code>	<code>default</code>	<code>func</code>	<code>interface</code>	<code>select</code>
<code>case</code>	<code>defer</code>	<code>go</code>	<code>map</code>	<code>struct</code>
<code>chan</code>	<code>else</code>	<code>goto</code>	<code>package</code>	<code>switch</code>
<code>const</code>	<code>fallthrough</code>	<code>if</code>	<code>range</code>	<code>type</code>
<code>continue</code>	<code>for</code>	<code>import</code>	<code>return</code>	<code>var</code>

Constants:

<code>true</code>	<code>false</code>	<code>nil</code>	<code>iota</code>
-------------------	--------------------	------------------	-------------------

Functions:

<code>new</code>	<code>len</code>	<code>complex</code>	<code>panic</code>
<code>make</code>	<code>cap</code>	<code>real</code>	<code>recover</code>
<code>close</code>	<code>append</code>	<code>imag</code>	
	<code>copy</code>		
	<code>delete</code>		

Basics

Basic types:

```
int      int8    int16   int32    int64
uint     uint8    uint16   uint32    uint64    uintptr

float32   float64
complex64 complex128

bool      byte    rune     string   error
```

- `int` and `uint` are platform-dependent
- `byte` is the same as `uint8`
- `rune` is the same as `uint32`
- `uintptr` is large enough to hold pointers
- `error` is a special type for error handling

Operators:

```
*    /    %    &    &^    <<    >>
+    -    ^    |
==   !=   <    <=   >    >=
&&
||
```

- only 5 precedence levels!
- `^` is both bitwise-xor (infix) and bitwise-not (prefix)
- `&^` is bitwise-andn

Declarations

```
// three semantically identical alternatives
```

```
var x int = 0
```

```
var x int
```

```
var x = 0
```

```
// fourth alternative for local variables only
```

```
x := 0
```

Strings

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var s = "Käsebrötchen"
7     fmt.Println(s)
8     fmt.Println(len(s)) // 14
9
10    // loop over the bytes (UTF-8 code units)
11    for i := 0; i < len(s); i++ {
12        fmt.Printf("%02d: %c\n", i, s[i])
13    }
14
15    // loop over the runes (Unicode code points)
16    for i, r := range s {
17        fmt.Printf("%02d: %c\n", i, r)
18    }
```


Arrays

```
1 func initSha1(a [20]byte) {  
2     // a is an array of 20 bytes  
3     a[0] = 0x01  
4     a[1] = 0x23  
5     a[2] = 0x45  
6     a[3] = 0x67  
7     // ...  
8     fmt.Println(a)  
9 }  
10  
11 func main() {  
12     var x [20]byte // arrays have a fixed size  
13     initSha1(x)    // arrays are passed by value!  
14     fmt.Println(x) // x remains unchanged  
15 }
```

Pointers

```
1 func initSha1(a *[20]byte) {
2     a[0] = 0x01
3     a[1] = 0x23
4     a[2] = 0x45
5     // ...
6     fmt.Println(*a)
7 }
8
9 func main() {
10     var x [20]byte
11     initSha1(&x) // &x is a pointer to x
12     fmt.Println(x)
13 }
```

Slices

```
1 func initSha1(a []byte) {  
2     a[0] = 0x01  
3     a[1] = 0x23  
4     a[2] = 0x45  
5     // ...  
6     fmt.Println(a)  
7 }  
8  
9 func main() {  
10     var x [20]byte  
11     initSha1(x[:]) // x[:] is a slice (view) into x  
12     fmt.Println(x) // x[:] is short-hand for x[0:20]  
13 }
```

Slices can grow

```
1 func main() {
2     var compound [100]bool
3     for i := 2; i*i < len(compound); i++ {
4         if !compound[i] {
5             for j := i * i; j < len(compound); j += i {
6                 compound[j] = true
7             }
8         }
9     }
10    var primes []int
11    for i := 2; i < len(compound); i++ {
12        if !compound[i] {
13            primes = append(primes, i)
14        }
15    }
16    fmt.Println(primes)
17 }
```

Exercise

- I. Extract two functions from the last main function:
 - `markCompounds`
 - `gatherPrimes`
- II. Determine the growth strategy of `append` by printing after each call:
 - either a pointer to the first element
or
 - the result of calling the special `cap` function

Maps

```
1 func main() {
2     birth := map[string]int{
3         "C":    1972,
4         "Java": 1994,
5     }
6     birth["Go"] = 2007
7     examine(birth, "Go")
8     delete(birth, "Go")
9     examine(birth, "Go")
10 }
11
12 func examine(birth map[string]int, language string) {
13     if year, present := birth[language]; present {
14         fmt.Printf("%s was released in %d\n", language, year)
15     } else {
16         fmt.Printf("Never heard of %s...\n", language)
17     }
18 }
```

Maps

```
1 func main() {  
2     birth := map[string]int{  
3         "C":    1972,  
4         "Java": 1994,  
5         "go":   2007,  
6     }  
7     for language, year := range birth {  
8         fmt.Printf("%s was released in %d\n", language, year)  
9     }  
10 }
```

Exercise

Write a function that counts the occurrences of all characters in a given string.

Which value is returned by map lookup for missing keys?

Structs

```
1  type Person struct {
2      Name string
3      Age  int
4  }
5
6  func main() {
7      alice := Person{"Alice", 21}
8
9      bob := alice
10     bob.Name = "Bob"
11     bob.Age++
12
13     fmt.Printf("%v\n", alice) // {Alice 21}
14     fmt.Printf("%#v\n", bob)  // main.Person{Name:"Bob",
15                               ↪   Age:22}
16 }
```

Structs

```
1  type Person struct {  
2      Name string  
3      Age  int  
4  }  
5  
6  func main() {  
7      myBoss := &Person{"Guido", 60}  
8  
9      yourBoss := myBoss  
10     yourBoss.Age++  
11  
12     fmt.Println(myBoss, yourBoss) // {Guido 61} {Guido 61}  
13 }
```

Web client I

```
1  type Xkcd struct { // full spec at https://xkcd.com/json.html
2      Title string
3      Hover string `json:"alt"`
4  }
5
6  func FetchCurrentXkcdComic() (*Xkcd, error) {
7      response, err := http.Get("https://xkcd.com/info.0.json")
8      if err != nil {
9          return nil, err
10     }
11     defer response.Body.Close()
12     if response.StatusCode != http.StatusOK {
13         return nil, errors.New(response.Status)
14     }
15     var xkcd Xkcd
16     decoder := json.NewDecoder(response.Body)
17     if err := decoder.Decode(&xkcd); err != nil {
18         return nil, err
19     }
20     return &xkcd, nil
21 }
```

Web client II

```
1 func main() {  
2     xkcd, err := FetchCurrentXkcdComic()  
3     if err != nil {  
4         fmt.Println(err)  
5     } else {  
6         fmt.Printf("%s\n\n%s\n", xkcd.Title, xkcd.Hover)  
7     }  
8 }
```

Exercise

Fetch the 3 most recent XKCD comics and print additional information of your choice.

Web server I

```
1 func main() {
2     http.HandleFunc("/", root)
3     fmt.Println("waiting for requests...")
4     http.ListenAndServe(":8080", nil)
5 }
6 var counter = 0
7 func root(writer http.ResponseWriter, request *http.Request) {
8     fmt.Printf("%s\n\n", request)
9     counter++
10    fmt.Fprintf(writer, "<html><body><pre>")
11    fmt.Fprintln(writer, request.URL.Path)
12    fmt.Fprintf(writer, "%d visits\n", counter)
13    fmt.Fprintf(writer, "</pre></body></html>")
14 }
```

Web server I

```
1 func main() {
2     http.HandleFunc("/", root)
3     fmt.Println("waiting for requests...")
4     http.ListenAndServe(":8080", nil)
5 }
6 var counter = 0
7 func root(writer http.ResponseWriter, request *http.Request) {
8     fmt.Printf("%s\n\n", request)
9     counter++
10    fmt.Fprintf(writer, "<html><body><pre>")
11    fmt.Fprintln(writer, request.URL.Path)
12    fmt.Fprintf(writer, "%d visits\n", counter)
13    fmt.Fprintf(writer, "</pre></body></html>")
14 }
```

Compiles and runs. Where's the bug?

Web server II

```
1  var counter = 0
2  var mutex sync.Mutex
3
4  func root(writer http.ResponseWriter, request *http.Request) {
5      fmt.Printf("%s\n\n", request)
6
7      mutex.Lock()
8      counter++
9      count := counter
10     mutex.Unlock()
11
12     fmt.Fprintf(writer, "<html><body><pre>")
13     fmt.Fprintln(writer, request.URL.Path)
14     fmt.Fprintf(writer, "%d visits\n", count)
15     fmt.Fprintf(writer, "</pre></body></html>")
16 }
```