Go

An introduction for Java Developers

Luca Schimweg, May 12th 2020

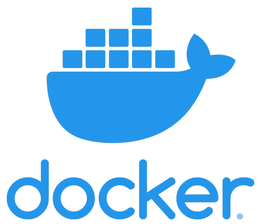
About Go

General

- Statically typed & compiled
- Designed at Google
- Memory safe & garbage collected
- Focus on improving productivity for multicore and network processing

About Go

Notable Projects



Variable declarations

Java

```
int i = 5;
```

Go

```
var i int = 5
// or
i := 5
```

Array initialization

Java

```java
int[] ary = new int[5];
```

Go

```go
var ary []int = make([]int, 5)
```

Array Iteration

Java

```
for (x in array) {
    // do something
}
```

Go

```
for x := range array {
    // do something
}
```

Functions

Java

```java
// inside a class

int add(int a, int b) {
    return a + b;
}
```

Go

```go
func Add(a int, b int) int {
    return a + b
}
```

Pass-by-value vs Pass-by-reference

General

- Pass-by-value: Give a copy of the actual data to a function, etc…
- Pass-by-reference: Give a reference (or pointer) of the actual data to a function, etc…

- **Important**: Functions called with pass-by-value **cannot** change the data, functions called with pass-by-reference **can** change the data

Pass-by-value vs Pass-by-reference

In Go

- Parameters are always passed by value in Go
- To use pass-by-reference logic, pointers can be passed
- Pointer expression look like they do in C (with ***** and **&**)

```
func Add(a *int, b *int) *int {
    i := *a + *b
    return &i
}
```

Classes vs Structs

- There are no classes in Go
- C-like structs
- Structs can have “methods” defined
- Inheritance by “including” other structs in a struct

Type declaration

Java

```java
// we do not have type aliases in Java

class Test {
    // class body
    int a;
}
```

Go

```go
type mySpecialInt int

type Test struct {
    // struct body
    a int
}
```

Access modifiers

Java

```java
class Test {
    private int a;  // private
    protected int b; // package-local
    public int c;   // public
}
```

Go

```go
type Test struct {
    // no private fields in Go
    b int   // package-local
    C int   // public
}
```

Class methods

Java

```java
class A {
    int num;

    public int getNumSquared() {
        return num * num;
    }
}
```

Go

```go
type A struct {
    num int
}

function (a *A) GetNumSquared() int {
    return a.num * a.num
}
```

Inheritance

Java

```
class B {
    public int i;
}

class A extends B {

}
```

Go

```
type B struct {
    I int
}

type A struct {
    B
}
```

Interfaces

Java

```java
interface B {
    public void doSomething();
}

class A implements B {
    // implement doSomething()
}
```

Go

```go
type B interface {
    DoSomething()
}

type A struct {
    // implement DoSomething()
}
```

Error Handling

Java

```
void doSomething() {
    try {
        int result = doSomethingRisky()
    } catch(Exception e) {
        // error handling
    }
}
```

Go

```
func DoSomething() {
    result, err := doSomethingRisky()

    if err != nil {
        // error handling
    }
}
```

Parallel Processing

Goroutines

- “Goroutines” are lightweight “threads”
- Started by adding “go” before function call:

```
func MyFunction() {
    numbers := []int {1, 2, 3, 4, 5, 6, 7, 8, 9}

    for i := range numbers {
        go HeavyProcessing(i)
    }
}
```

Parallel Processing

Channels

- Channels are similar to thread-safe queues
- Buffer size can be specified (defaults to 0)
- Operations on queue are blocking for that goroutine

```
func MyFunction() {
    channel := make(chan string)
    go func() { channel <- “test" }

    msg := <-channel
}
```

Parallel Processing

Channel directions

- Channels can specify their direction when used as function parameters

```
func MyFunction(receivingChannel <-chan string) {
    // we can read but not write to receivingChannel
}

func MyFunction2(sendingChannel chan<- string) {
    // we can only write to sendingChannel
}
```

Parallel Processing

Select

- More powerful version of C's switch-case instruction, supporting channels

```
func MyFunction() {
    channel1 := make(chan string)
    channel2 := make(chan int)
    SomeFunctionCall(channel1, channel2)

    select {
    case msg1 := <-channel1:
        // channel1 was sent to first
    case msg2 := <-channel2:
        // channel2 was sent to first
    }
}
```

Packages

- Everything in Go is in a package
- A package is represented by one folder on the file system
  - The package's name does not have to match the folder's name
- For executables there has to be a package *main* containing the main function

```
package main

import "fmt"

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

Live Demo