

# KNOW GO GENERIC

*“Beautifully explained”*  
—Pavel Anni



Go 1.22

POLYMORPHIC PROGRAMMING IN GO

JOHN ARUNDEL

# Contents

<b>Praise for ‘Know Go: Generics’</b>	<b>8</b>
<b>Introduction</b>	<b>9</b>
About the book . . . . .	9
About generics . . . . .	9
Who is the book for? . . . . .	10
What should I read first? . . . . .	10
What will I learn from this book? . . . . .	10
Completing the challenges . . . . .	11
How do I install the Go tools? . . . . .	11
Where are the code examples? . . . . .	11
<b>1. Generics</b>	<b>12</b>
Programming with types . . . . .	12
Specific programming . . . . .	12
Generic programming . . . . .	13
Interface types . . . . .	13
Interface parameters . . . . .	14
Polymorphism . . . . .	15
Interfaces make code flexible . . . . .	15
Constraining parameters with interfaces . . . . .	15
Limitations of method sets . . . . .	16
The empty interface: any . . . . .	17
Type assertions and switches . . . . .	18
Go, meet generics . . . . .	18
How it started . . . . .	18
How it’s going . . . . .	19
<b>2. Type parameters</b>	<b>20</b>
Generic functions . . . . .	20
Introducing T, the arbitrary type . . . . .	20
Type parameters . . . . .	21
Instantiation . . . . .	21
Stencilling . . . . .	22
Getting started . . . . .	22
An Identity function . . . . .	23
Instantiating Identity . . . . .	23
Exercise: Hello, generics . . . . .	24
Running the test . . . . .	24

Composite types . . . . .	25
Slices of some arbitrary type . . . . .	26
Other generic composite types . . . . .	26
Generic types . . . . .	26
Defining a generic slice type . . . . .	27
The elements all have the same type . . . . .	27
Generic types need to be instantiated . . . . .	27
Exercise: Group therapy . . . . .	28
Generic function types . . . . .	29
Generic functions as values . . . . .	29
The type is always instantiated . . . . .	29
There are no generic functions . . . . .	30
Generic types as function parameters . . . . .	30
Exercise: Lengthy proceedings . . . . .	31
Constraining type parameters . . . . .	32
We can't add any to any . . . . .	32
Not every type is "addable" . . . . .	32
<b>3. Constraints</b>	<b>34</b>
Method set constraints . . . . .	34
Limitations of the any constraint . . . . .	34
Basic interfaces . . . . .	35
Exercise: Stringy beans . . . . .	36
Type set constraints . . . . .	37
Type elements . . . . .	37
Using a type set constraint . . . . .	37
Unions . . . . .	38
The set of all allowed types . . . . .	39
Intersections . . . . .	39
Empty type sets . . . . .	39
Composite type literals . . . . .	40
A struct type literal . . . . .	40
Access to struct fields . . . . .	40
Some limitations of type sets . . . . .	41
Constraints versus basic interfaces . . . . .	41
Constraints are not classes . . . . .	41
Approximations . . . . .	42
Limitations of named types . . . . .	42
Type approximations . . . . .	43
Derived types . . . . .	43
Exercise: A first approximation . . . . .	44
Interface literals . . . . .	45
Syntax of an interface literal . . . . .	46
Omitting the <code>interface</code> keyword . . . . .	46
Referring to type parameters . . . . .	47
Exercise: Greater love . . . . .	48

<b>4. Operations</b>	<b>50</b>
Arithmetic . . . . .	50
An AddAnything function . . . . .	51
The set of all numeric types . . . . .	51
Exercise: Product placement . . . . .	52
Ordered types . . . . .	54
The > operator . . . . .	54
Strings . . . . .	55
An Ordered interface . . . . .	55
The cmp.Ordered constraint . . . . .	56
Multiple type parameters . . . . .	56
Function on two (or more) types . . . . .	57
Each type parameter is a distinct type . . . . .	57
Functions on slice types . . . . .	57
The problem with derived slice types . . . . .	58
Parameterizing by slice and element type . . . . .	59
And I need to know this because...? . . . . .	60
Comparable types . . . . .	60
Not every type is comparable . . . . .	60
Not every comparable type is ordered . . . . .	61
There are infinitely many comparable types . . . . .	61
The comparable constraint . . . . .	62
Why is comparable predeclared? . . . . .	62
Exercise: Duplicate keys . . . . .	62
Abstract types . . . . .	64
A Greatest function . . . . .	64
The scope of type parameters . . . . .	65
The zero value . . . . .	65
Switching on abstract types . . . . .	66
<b>5. Types</b>	<b>68</b>
Named types . . . . .	68
Named basic types . . . . .	69
Generic basic types . . . . .	69
Generic slice types . . . . .	69
There are no generic types . . . . .	70
Slices of interface types . . . . .	70
Generic map types . . . . .	71
Maps of abstract element types . . . . .	71
Multiple type parameters . . . . .	72
Instantiating multiple type parameters . . . . .	72
Generic struct types . . . . .	73
Self-reference . . . . .	73
Methods . . . . .	74
Adding methods to generic types . . . . .	74
Exercise: Empty promises . . . . .	74

Parameterised methods . . . . .	75
More generic composite types . . . . .	76
Interfaces . . . . .	76
Channels . . . . .	76
<b>6. Functions</b>	<b>78</b>
Functions on container types . . . . .	78
Contains . . . . .	79
Reverse . . . . .	79
Sort . . . . .	80
First-class functions . . . . .	81
Map . . . . .	81
Type inference . . . . .	82
Exercise: Func to funky . . . . .	83
Filtering and reduction . . . . .	85
Filter . . . . .	85
Filter functions . . . . .	86
Generic filter functions . . . . .	86
Reduce . . . . .	87
Implementing Reduce . . . . .	87
Other reduction operations . . . . .	88
Other considerations . . . . .	89
Constraints . . . . .	89
Concurrency . . . . .	89
Exercise: Compose yourself . . . . .	90
<b>7. Containers</b>	<b>93</b>
Sets . . . . .	93
Maps as sets . . . . .	94
Operations on sets . . . . .	94
Designing the Set type . . . . .	95
Building out the machinery . . . . .	95
The Add method . . . . .	95
The Contains method . . . . .	95
Initialising with multiple values . . . . .	96
Getting a set's members . . . . .	97
A String method . . . . .	97
Logic on sets . . . . .	98
Union . . . . .	98
Intersection . . . . .	99
A real-life example . . . . .	99
Exercise: Stack overflow . . . . .	100
<b>8. Concurrency</b>	<b>103</b>
Data races . . . . .	103
Mutexes . . . . .	104

Deadlocks . . . . .	104
A concurrency-safe set type . . . . .	104
Locking . . . . .	105
The constructor . . . . .	105
A locking Add method . . . . .	105
The read-locking methods . . . . .	106
Testing concurrency safety . . . . .	107
Smoke tests . . . . .	107
A fatal error . . . . .	107
The race detector . . . . .	108
And the rest . . . . .	109
Exercise: Channelling frustration . . . . .	110
Extending the Set type . . . . .	115
More set operations . . . . .	115
Key-value stores and caches . . . . .	116
Other container types . . . . .	116
<b>9. Packages</b>	<b>117</b>
The cmp package . . . . .	118
The slices package . . . . .	118
Comparing slices . . . . .	118
Finding elements . . . . .	120
Maxima and minima . . . . .	121
Inserting and deleting . . . . .	121
Cloning and compacting . . . . .	122
Growing and shrinking . . . . .	123
Sorting . . . . .	123
Reversing and replacing . . . . .	125
Searching . . . . .	125
The maps package . . . . .	126
Comparing maps . . . . .	126
Deleting map entries . . . . .	127
Cloning and copying . . . . .	127
New idioms . . . . .	128
Copying slices . . . . .	128
Deleting slice elements . . . . .	128
Checking for slice elements . . . . .	128
Exercise: Merging in turn . . . . .	129
<b>10. Questions</b>	<b>133</b>
Change . . . . .	133
How much do I need to know about generics? . . . . .	133
Will generics drastically change the way I write Go? . . . . .	134
Do I need to change my existing code? . . . . .	135
Performance . . . . .	135
What impact does generics have on performance? . . . . .	135

Does generic code compile more slowly? . . . . .	136
Downsides . . . . .	136
Why didn't they use angle brackets? . . . . .	137
Still no option types . . . . .	138
Still no enums . . . . .	138
No proper union types . . . . .	139
No macros . . . . .	139
No parameterised methods . . . . .	139
No generic packages . . . . .	139
No "insert cool tech here" . . . . .	140
<i>More change</i> . . . . .	140
Will there be a "Go 2"? . . . . .	141
There will be changes to Go . . . . .	141
It won't be "Go 2" . . . . .	142
But there will be a successor to Go (someday) . . . . .	142
<b>11. Iterators</b> . . . . .	<b>143</b>
Introducing iterators . . . . .	143
What is an iterator? . . . . .	144
Why are iterators useful? . . . . .	144
The signature of an iterator function . . . . .	145
Iterators in practice . . . . .	145
Returning an iterator . . . . .	146
Two-result iterators . . . . .	146
Iterators that return errors . . . . .	147
When <code>yield</code> returns <code>false</code> . . . . .	148
Composing iterators . . . . .	148
Questions . . . . .	150
Does this affect the standard library? . . . . .	150
Why not channels? . . . . .	150
Enabling the experimental support for iterators . . . . .	151
<b>About this book</b> . . . . .	<b>152</b>
Who wrote this? . . . . .	152
Feedback . . . . .	152
Free updates to future editions . . . . .	153
Join my Go Club . . . . .	153
For the Love of Go . . . . .	153
The Power of Go: Tools . . . . .	154
Further reading . . . . .	154
Credits . . . . .	154
<b>Acknowledgements</b> . . . . .	<b>155</b>