

A photograph of a server room with rows of server racks illuminated by blue light. The racks are filled with server units, and the perspective leads the eye down a central aisle. The lighting is predominantly blue, with some yellow light from the server units themselves.

Introduction to Programming in Go

10. Interfaces

Module Topics

1. Interfaces in Go
2. Interface Variables
3. Type Testing and Switch Statements
4. The Empty Interface

Interfaces in Go

Interfaces

1. Interfaces are defined in terms of bundles of functionality that define a "type."
2. Interfaces define a set of method signatures

```
type Swimmer interface {  
    swim()  
}
```

3. Any type which has a swim() method "implements" the interface and is of type "Swimmer."
4. There is no need for any declaration, just the implementation of the methods (ie. things that swim are of type Swimmer).

Interface Example

```
// Example 10-01 Swimmer Interface
...
type Swimmer interface {
    swim()
}
type fish struct{ species string }
type human struct{ name string }

func (f *fish) swim() {
    fmt.Println("Underwater")
}
func (f *human) swim() {
    fmt.Println("Dog Paddle")
}
func (f *human) walk() {
    fmt.Println("Strolling around")
}
```

Interface Example

```
// Example 10-01 Swimmer Interface
```

```
...
```

```
func main() {  
    tuna := new(fish)  
    tuna.swim()  
    knuth := new(human)  
    knuth.new()  
    knuth.walk()  
}
```

```
[Module10]$ go run ex10-01.go  
Underwater  
Dog Paddle  
Strolling around
```

Interface Variables

Interface Variables

1. Variables can be of an interface type.
2. Any type that implements an interface can be assigned to that variable of that interface type.
3. The variable has a pointer for each method defined in the interface definition.
4. When a concrete type is assigned, the interface variable updates its pointers to reference the concrete implementations.

Interface Variables

```
// Example 10-02  Interface Variables

...
type Swimmer interface {
    swim()
}

func main() {
    var s Swimmer // s is of type "Swimmer"
    s = new(fish)
    s.swim()
    s = new(human)
    s.swim()
}
```

```
[Module10]$ ./ex10-02
Underwater
Dog Paddle
```

Interface Variable Parameters

```
// Example 10-03  Interface variables as parameters
```

```
...
```

```
type Swimmer interface {  
    swim()  
}
```

```
func submerge(s Swimmer) {  
    s.swim()  
}
```

```
func main() {  
    submerge(new(fish))  
    submerge(new(human))  
}
```

```
[Module10]$ ./ex10-03  
Underwater  
Dog Paddle
```

Non-interface Methods

```
// Example 10-04  Interface Variables

...
type Swimmer interface {
    swim()
}

func main() {
    var s Swimmer // s is of type "Swimmer"
    s = new(human)
    s.swim()
    s.walk() // in human but not Swimmer
}
```

```
[Module10]$ ./ex10-04
./ex10-04.go:8: s.walk undefined (type Swimmer has no field or method walk)
```

Combining Interfaces

Combining Interfaces

1. Interfaces in Go are usually small - only several methods each.
2. Interfaces can be combined like structs.
3. All of the methods are just merged into one interface.

Combining Interfaces

```
// Example 10-05 Amphib Type
```

```
...
```

```
type Swimmer interface {  
    swim()  
}
```

```
type Walker interface {  
    walk()  
}
```

```
type Amphib interface {  
    Walker  
    Swimmer  
}
```

```
func main() {  
    var a Amphib  
    a = new(human)  
    a.walk()  
    a.swim()  
}
```

```
[Module10]$ ./ex10-05  
Strolling around  
Dog Paddle
```

Type Testing and Switching

Type Testing

1. Example 10-04 showed that we need sometimes to know what the concrete type is of the contents of an interface variable.
2. This is done by type checking using the syntax:
`v, ok := ivar.(T)`
3. If the object in the interface variable "ivar" is of type "T" the ok is true and the object is assigned to variable v which is of type T.
4. If "ivar" is not of type "T" then ok is false and v is nil.
5. In the following example we added a few more Swimmer types and use a switch statement to do type specific processing.

Type Testing

```
// Example 10-06 Switching on types

type Swimmer interface {
    swim()
}

func main() {
    var x = []Swimmer{&human{"bobby"}, &fish{}}
    for _, swimmer := range x {
        if h, ok := swimmer.(*human); ok {
            h.walk()
        }
    }
}
```

```
[Module10]$ ./ex10-06
Strolling around
```

Type Testing

```
// Example 10-07 Switching on types
```

```
var x = []Swimmer{&human{"bobby"}, &fish{}, &cat{},  
                &squid{}}
```

```
func main() {  
    for index, swimmer := range x {  
        switch t := swimmer.(type) {  
            case *human:  
                fmt.Printf("Item %d is human and is ", index)  
                t.walk()  
            case *squid:  
                fmt.Println("Item ", index, "is a squid")  
            case *fish:  
                fmt.Println("Item ", index, "is a fish")  
            default:  
                fmt.Printf("Item %d is type %T\n", index, t)  
        }  
    }  
}
```

```
[Module10]$ ./ex10-07  
Item 0 is a human and is Strolling around  
Item 1 is a fish  
Item 2 is of type *main.cat  
Item 3 is a squid
```


The Empty Interface

Empty Interface

1. The empty interface has no methods specified.
2. Everything in Go vacuously implements every empty interface type.
3. Every type, built-in or user defined in the same package as these definitions below are of type "whatever" and "stuff"

```
type whatever interface {}  
type stuff interface {}
```

4. Empty interfaces are useful for creating collections of arbitrary objects of different types.

Empty Interface

```
// Example 10-08 The empty Interface - data
...

type Swimmer interface {
    swim()
}
type myint int32
type point struct{ x, y int }

var i myint = 0

type whatever interface{}
```

Empty Interface Collection

// Example 10-08 The empty Interface continued -

```
func main() {
    mylist := []whatever{i, float64(45), &point{2, 3},
                                                                    true, &fish{"tuna"}}

    for index, object := range mylist {
        switch v := object.(type) {
            case bool:
                fmt.Printf("%d is bool = %v\n", index, v)
            case myint:
                fmt.Printf("item %d is myint = %v\n", index, v)
            case float64:
                fmt.Printf("item %d is a float64 = %v\n", index, v)
            case *point:
                fmt.Printf("item %d is a point = %v\n", index, *v)
            default:
                fmt.Printf("item %d is a %T = %v\n", index, v, v)
        }
    }
}
```

Empty Interface Collection - Output

```
[Module10]$ ./ex10-08  
item 0 is a myint = 0  
item 1 is a float64 = 45  
item 2 is a point = {2 3}  
item 3 is a bool = true  
item 4 is a *main.fish = &{tuna}
```


Lab 10: Interfaces