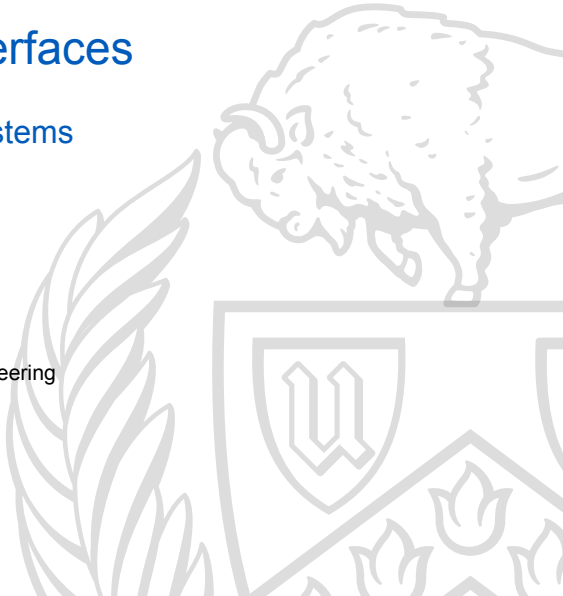
Go Methods and Interfaces

CSE 486/586: Distributed Systems

Ethan Blanton

Department of Computer Science and Engineering

University at Buffalo



Structural Typing

Go uses structural typing for polymorphism.

Any type in go may have methods.

A collection of methods can form an interface.

Any type providing those methods implements the interface.

Note that it need not declare this implementation!

Many Go functions accept interface implementations.

Methods

Go methods are functions with an implicit receiver.

```
func (r Receiver) DoSomething(a Argument) {
  r.a = a
}
```

The receiver can be any type in the current package.

Once a method is defined, any instance of the type provides it.

```
var r Receiver
r.DoSomething(a)
```

Interfaces

Go interfaces are a collection of methods.

They do not define the receiver type.

They do not restrict other methods defined on the receiver.

They are an abstract type.

```
type SomeInterface interface {
    Method(a Argument) ReturnType
}
```

Any type defining all methods of an interface implements it.

Method Polymorphism

Go methods are polymorphic over receiver type.

More than one receiver can implement the same method name.

A method may accept different arguments on different receivers.

Only one method of a given name can be defined for a given receiver type.

Argument Polymorphism

Interfaces provide polymorphism for arguments.

If an argument requires an interface type, any implementation of that interface satisfies the argument.

Functions and methods are not polymorphic by signature.

This is used heavily in the Go standard libraries.
`Reader`, `Writer`, *etc.* are common.

(Lack of) Generics

Go has no generics (yet).

The type `interface{}` is as close as it gets.

Any object satisfies `interface{}`. (Because it requires no methods.)

You must type assert to use `interface{}`:

```
func F(arg interface{}) {
        v, ok = arg.(SomeType)
        // v is SomeType, ok indicates success/failure
}
```

Examples

This is best illustrated by example.

Summary

- Go uses structural (“duck”) typing
- Go provides polymorphism through
  - Methods
  - Interfaces

References I

Required Readings

[1] *Effective Go*. URL: https://golang.org/doc/effective_go.html.

To retrieve a copy of this material, or related materials, see `https://www.cse.buffalo.edu/~eblanton/`.