Module 4: Interfaces for Abstraction
Topic 1.3: Interface vs. Concrete Types

Concrete vs Interface Types

Concrete Types

- Specify the exact representation of the data and methods
- Complete method implementation is included

Interface Types

- Specifies some method signatures
- Implementations are abstracted

Interface Values

- Can be treated like other values
  - Assigned to variables
  - Passed, returned
- Interface values have two components

1. **Dynamic Type**: Concrete type which it is assigned to
2. **Dynamic Value**: Value of the dynamic type

- Interface value is actually a pair
  - **(dynamic type, dynamic value)**

Defining an Interface Type

```
type Speaker interface {Speak ()}

type Dog struct {name string}
func (d Dog) Speak() {
      fmt.Println(d.name)
}
func main() {
      var s1 Speaker
      var d1 Dog{“Brian”}
      s1 = d1
      s1.Speak()
}
```

- Dynamic type is Dog, Dynamic value is d1

Interface with Nil Dynamic Value

- An interface can have a nil dynamic value

```
var s1 Speaker
var d1 *Dog
s1 = d1
```

- d1 has no concrete value yet
- s1 has a dynamic type but no dynamic value

Nil Dynamic Value

- Can still call the `Speak()` method of s1
- Doesn't need a dynamic value to call
- Need to check inside the method

```
func (d *Dog) Speak() {
      if d == nil {
            fmt.Println("<noise>")
      } else {
            fmt.Println(d.name)
      }
}
var s1 Speaker
var d1 *Dog
s1 = d1
s1.Speak()
```

Nil Interface Value

- Interface with **nil dynamic type**
- Very different from an interface with a **nil dynamic value**

Nil dynamic value and **valid dynamic type**

- Can call a method since type is known

```
var s1 Speaker
var d1 *Dog
s1 = d1
```

Nil dynamic type

- Cannot call a method, runtime error

```
var s1 Speaker
```