package main

import (

    "encoding/json"

    "fmt"

)

type Animal interface {

    Speak() string

}

type Dog struct {

    Name string

}

func (d Dog) Speak() string {

    return "Woof!"

}

type Cat struct {

    Name string

}

func (c Cat) Speak() string {

    return "Meow!"

}

func main() {

    animals := []Animal{Dog{Name: "Buddy"}, Cat{Name: "Whiskers"}}

    fmt.Println(animals[0].(Dog))

    jsonData, err := json.Marshal(animals)

    if err != nil {

        fmt.Println(err)

        return

    }

    fmt.Println(string(jsonData))

}

In the code above we have a situation

* We are storing different two different structs in a slice of interface
* When we Marshal it the type of structs will be lost
* So during unmarshal how will you get it back to the two different structs of different type

**SOLUTION**

* We will create a wrapper struct around it which will store the **interface** and the **type of struct**
* Then we will create custom marshal and unmarsal for the wrapper
* Which during unmarshal will check the type struct and unmarshal it

package main

import (

    "encoding/json"

    "errors"

    "fmt"

)

// Animal interface

type Animal interface {

    Speak() string

    GetType() string

}

// Dog struct

type Dog struct {

    Name string

}

func (d Dog) Speak() string {

    return "Woof!"

}

func (d Dog) GetType() string {

    return "dog"

}

// Cat struct

type Cat struct {

    Name string

}

func (c Cat) Speak() string {

    return "Meow!"

}

func (c Cat) GetType() string {

    return "cat"

}

// AnimalWrapper for serializing and deserializing with type information

type AnimalWrapper struct {

    Type   string          `json:"type"`

    Animal json.RawMessage `json:"animal"`

}

func WrapAnimal(a Animal) (\*AnimalWrapper, error) {

    data, err := json.Marshal(a)

    if err != nil {

        return nil, err

    }

    return &AnimalWrapper{

        Type:   a.GetType(),

        Animal: data,

    }, nil

}

func UnwrapAnimal(w \*AnimalWrapper) (Animal, error) {

    var a Animal

    switch w.Type {

    case "dog":

        a = &Dog{}

    case "cat":

        a = &Cat{}

    default:

        return nil, errors.New("unknown animal type")

    }

    err := json.Unmarshal(w.Animal, a)

    if err != nil {

        return nil, err

    }

    return a, nil

}

// Main function to demonstrate serialization and deserialization

func main() {

    dog := Dog{Name: "Buddy"}

    cat := Cat{Name: "Whiskers"}

    animals := []Animal{dog, cat}

    // Serialize animals

    var wrappedAnimals []\*AnimalWrapper

    for \_, animal := range animals {

        wrappedAnimal, err := WrapAnimal(animal)

        if err != nil {

            fmt.Println(err)

            return

        }

        wrappedAnimals = append(wrappedAnimals, wrappedAnimal)

    }

    jsonData, err := json.Marshal(wrappedAnimals)

    if err != nil {

        fmt.Println(err)

        return

    }

    fmt.Println("Serialized:", string(jsonData))

    // Deserialize animals

    var unwrappedAnimals []Animal

    for \_, wrappedAnimal := range wrappedAnimals {

        animal, err := UnwrapAnimal(wrappedAnimal)

        if err != nil {

            fmt.Println(err)

            return

        }

        unwrappedAnimals = append(unwrappedAnimals, animal)

    }

    for \_, animal := range unwrappedAnimals {

        fmt.Println(animal.Speak())

    }

}