Non-Exported Fields

This lesson covers the important concept of how to use fields of a struct if a struct is present in another file or in a separate custom package.

WE'LL COVER THE FOLLOWING

Methods and non-exported fields

How can we change, or even read the name of the object of a *user-defined* type in another program? This is accomplished using a well-known technique from OO-languages: provide getter and setter methods. For the setter-method, we use the prefix **Set**, and for the getter-method, we only use the field name.

Look at the example below:

Environment Variables

Key:	Value:
GOROOT	/usr/local/go
GOPATH	//root/usr/local/go/src
PATH	//root/usr/local/go/src/bin:/usr/local/go...

```
package main
import (
"fmt"
"pers"
)

func main() {
    p := new(pers.Person)
    // error: p.firstName undefined
    // (cannot refer to unexported field or method firstName)
    //p.firstName = "Eric"
    p.SetFirstName("Eric")
    fmt.Println(p.FirstName()) // Output: Eric
}
```

See the **pers.go** file. We have a struct of type `Person` with two string fields `firstName` and `lastName` at **line 3**.

Look at the header of the `FirstName()` method at **line 8**: `func (p *Person) FirstName() string`. From the name, it's obvious that it is a *getter-method*. Only the pointer to the object of the `Person` type can call it. This method returns the `firstName` (internal field) of `p`.

Now, look at the header of the `SetFirstName()` method at **line 12**: `func (p *Person) SetFirstName(newName string)`. From the name, it's obvious that it is a *setter-method*. Only the pointer to the object of the `Person` type can call it. This method sets the value of `firstName` (internal field) of `p` equal to `newName` (parameter). Keep in mind that the type `Person` can be exported anywhere, but its field can't be imported.

Let's look at the **main.go** file to grasp this concept. At **line 4**, we import the package `pers`. In `main`, at **line 8**, we make a `Person` type object `p` using `new()` by accessing `Person` as: `pers.Person`. We can easily import the type `Person` with a selector statement, but we can't export its field. See the commented **line 11**, where `firstName` (internal field of `p`) is directly accessed. It will give an error. That's why we made setter and getter methods.

At **line 12**, we are using the setter to set the name **Eric** to the `firstName` of `p`. In the next line, we are using the getter method to get the `firstName` of `p` and then printing the returned value.

That's it about using a struct from another package. In the next lesson, we'll study the print mechanism of a struct.