Handling Classes in Go

In Go, classes are handled using structs and methods. Go does not have traditional classes like other object-oriented languages, but you can achieve similar functionality with structs and methods. Here's a quick guide:

1. Define a Struct:

   A struct is a collection of fields. It is used to define the structure of an object.

   ```go
   type Person struct {
       Name string
       Age  int
   }
   ```

2. Create Methods for the Struct:

   Methods are functions with a special receiver argument. The receiver appears in its own argument list between the `func` keyword and the method name.

   ```go
   // Method with a pointer receiver
   func (p *Person) Greet() string {
       return "Hello, my name is " + p.Name
   }
   ```

3. Instantiate and Use the Struct:

You can create an instance of the struct and call its methods.

```go
func main() {
    p := Person{Name: "John", Age: 30}
    fmt.Println(p.Greet())
}
```

4. Encapsulation:

You can control access to the fields and methods by using uppercase and lowercase letters. In Go, names starting with an uppercase letter are exported (public), while those starting with a lowercase letter are unexported (private).

```go
type person struct {
    name string
    age  int
}

func (p *person) greet() string {
    return "Hello, my name is " + p.name
}
```

5. Composition:

Instead of inheritance, Go uses composition to reuse code. You can embed a struct within another struct to achieve this.

```go
type Employee struct {
    Person
    EmployeeID string
}

func main() {
    e := Employee{
        Person:     Person{Name: "Alice", Age: 28},
        EmployeeID: "E12345",
    }
    fmt.Println(e.Greet()) // This will call the Greet method of the embedded Person struct
}
```

By using structs and methods, you can effectively manage and organize your code in Go, similar to how classes are used in other languages.