Object-Oriented Programming

Object Oriented Programming is a paradigm in programming based on the concept of "objects". Objects contain data in the form of properties and methods. It is the most popular style used in programming due to its modulatiry, code reusability and organisability. In general naming convention, basic concepts of OOPS include:

1. **Class:** A class is a blueprint of an object to be created.
2. **Object:** Instance of class containing both data and methods.
3. **Encapsulation:** Limiting access to properties, methods or variables in a class for code external to that class.
4. **Inheritance:** It allows a class (commonly referred to as subclass) to inherit properties and methods from another class (commonly referred to as parent class).
5. **Polymorphism:** Allowing objects of different classes to be treated as objects of a common superclass.
6. **Abstraction:** Abstracting a basic concept of multiple classes in a single abstract class. This simplifies logic and makes code more readable.

We don't dive into the details and understanding of OOPS in this article. This article focuses on the implementation of these concepts in GO progamming language, since unlike normal programming languages like Java, C++, etc, Go doesn't have the concepts of OOPS directly with common naming conventions. For example, it does not have have a ***class*** keyword.

OOP in GO

Although GO does not have classes, it allow us to use OOP concepts using structs and interfaces. Let's see how we can use OOP concepts in GO with the examples:

Classes, properties and methods

We can use structs in GO to achieve the same functionality as class in other programming languages. Structs can have methods and properties. Let's say we are building a billing application in which we want to define a company:

```
type Company struct {
        Id string
        Name string
        Country string
}

func newCompany(name string, country string) Company {
    return Company{
        Id: uuid.New().String(),
        Name: name,
        Country: country,
    }
}
```

Go does not have the concept of constructors and classes, so we defined a custom function to return the Compay. This would work as a constructor for the Company. We initialize Id in this function. Now to create an object of type Company, we will call *newCompany* method like this: $\star$