# Object-oriented programming with the GO language

1. The GO language does not support the object-oriented programming style. Virtual functions and polymorphism in the GO language are implemented with interfaces, but this is not a complete implementation of object-oriented style.The example below demonstrates this:
2. package main
   * 1. import "fmt"
     2. type ianimal interface {
     3. speak()
     4. sayHello()
     5. }
     6. type animal struct {
     7. }
     8. func (animal) speak() {
     9. fmt.Println("???")
     10. }
     11. func (nml animal) sayHello() {
     12. nml.speak()
     13. }
     14. type dog struct {
     15. animal
     16. }
     17. func (dog) speak() {
     18. fmt.Println("gav gav")
     19. }
     20. func main() {
     21. var pet ianimal = dog{}
     22. pet.speak()
     23. pet.sayHello()
     24. }
3. The application prints:
   1. gav gav
   2. ???
   3. The speak() function is not trully virtual.
   4. There are many ways to fix this error. The program patch below suggests ways to increase the competitiveness of the GO language:
   5. package main
   6. import "fmt"
   7. type ianimal interface {
   8. speak()
   9. }
   10. type animal struct {
   11. }
   12. func (animal) speak() {
   13. fmt.Println("???")
   14. }
   15. type dog struct {
   16. animal
   17. }
   18. func (dog) speak() {
   19. fmt.Println("gav gav")
   20. }
   21. func sayHello(nml ianimal) {
   22. nml.speak()
   23. }
   24. func main() {
   25. var pet ianimal = dog{}
   26. pet.speak()
   27. **sayHello(pet)**
   28. }
   29. The program will print the correct answer after these minor corrections:
   30. gav gav
   31. gav gav
   32. The initial version of the application would print correct values if an interface could be written in the receiver expression:
   33. func (**nml ianimal**) sayHello() {
   34. nml.speak()
   35. }
   36. Unfortunately, the current version of the compiler catches an error in such an expression, and it's not entirely clear why interfaces were banned in the receiver’s expression.
   37. I suggest to extend GO language adding the concept of "soft members". These soft members will have interface in the receiver. This small fix would allow GO language programmers to work with virtual functions and polymorphism as comfortably as in C++, C# or Java languages.