**1. Classes e Objetos Básicos**

package main

import "fmt"

type Car struct {

Brand string

Model string

Year int

}

func main() {

car1 := Car{"Toyota", "Corolla", 2020}

car2 := Car{"Honda", "Civic", 2018}

car3 := Car{"Ford", "Mustang", 2021}

fmt.Println(car1)

fmt.Println(car2)

fmt.Println(car3)

}

**2. Métodos**

package main

import "fmt"

type Car struct {

Brand string

Model string

Year int

Speed int

}

func (c \*Car) Accelerate() {

c.Speed += 10

}

func (c \*Car) Brake() {

if c.Speed >= 10 {

c.Speed -= 10

}

}

func (c Car) ShowSpeed() {

fmt.Printf("Current speed of %s %s: %d km/h\n", c.Brand, c.Model, c.Speed)

}

func main() {

car := Car{"Toyota", "Corolla", 2020, 0}

car.Accelerate()

car.ShowSpeed()

car.Accelerate()

car.ShowSpeed()

car.Brake()

car.ShowSpeed()

}

**4. Herança**

package main

import "fmt"

type Animal interface {

MakeSound()

}

type Dog struct{}

func (d Dog) MakeSound() {

fmt.Println("Woof!")

}

type Cat struct{}

func (c Cat) MakeSound() {

fmt.Println("Meow!")

}

func main() {

var animals []Animal

animals = append(animals, Dog{}, Cat{})

for \_, animal := range animals {

animal.MakeSound()

}

}

**5. Polimorfismo**

package main

import "fmt"

type Animal interface {

MakeSound()

}

type Dog struct{}

func (d Dog) MakeSound() {

fmt.Println("Woof!")

}

type Cat struct{}

func (c Cat) MakeSound() {

fmt.Println("Meow!")

}

func AnimalSounds(animals []Animal) {

for \_, animal := range animals {

animal.MakeSound()

}

}

func main() {

animals := []Animal{Dog{}, Cat{}}

AnimalSounds(animals)

}

**6. Composição**

package main

import "fmt"

type Engine struct {

HorsePower int

}

type Car struct {

Brand string

Model string

Year int

Engine Engine

}

func main() {

engine := Engine{HorsePower: 150}

car := Car{"Toyota", "Corolla", 2020, engine}

fmt.Printf("Car: %s %s (%d) with %d HP engine\n", car.Brand, car.Model, car.Year, car.Engine.HorsePower)

}

**7. Associação**

package main

import "fmt"

type Professor struct {

Name string

}

type School struct {

Name string

Professors []Professor

}

func main() {

prof1 := Professor{"Dr. Smith"}

prof2 := Professor{"Ms. Johnson"}

school := School{"Green Valley High", []Professor{prof1, prof2}}

fmt.Printf("School: %s\n", school.Name)

fmt.Println("Professors:")

for \_, professor := range school.Professors {

fmt.Println(professor.Name)

}

}

**8. Agregação**

package main

import "fmt"

type Employee struct {

Name string

Position string

Salary float64

}

type Company struct {

Name string

Employees []Employee

}

func main() {

emp1 := Employee{"John Doe", "Manager", 60000}

emp2 := Employee{"Jane Smith", "Developer", 50000}

company := Company{"TechCorp", []Employee{emp1, emp2}}

fmt.Printf("Company: %s\n", company.Name)

fmt.Println("Employees:")

for \_, emp := range company.Employees {

fmt.Printf("%s - %s ($%.2f)\n", emp.Name, emp.Position, emp.Salary)

}

}

**9. Interfaces/Protocolos**

package main

import "fmt"

type Printable interface {

Print()

}

type Report struct {

Title string

}

func (r Report) Print() {

fmt.Println("Printing Report:", r.Title)

}

type Contract struct {

ID string

}

func (c Contract) Print() {

fmt.Println("Printing Contract:", c.ID)

}

func main() {

var doc Printable

doc = Report{"Annual Report"}

doc.Print()

doc = Contract{"1234"}

doc.Print()

}

**10. Sobrecarga de Métodos**

package main

import "fmt"

type Calculator struct{}

func (c Calculator) AddTwoNumbers(a int, b int) int {

return a + b

}

func (c Calculator) AddThreeNumbers(a int, b int, c1 int) int {

return a + b + c1

}

func main() {

calculator := Calculator{}

sumTwo := calculator.AddTwoNumbers(5, 3)

sumThree := calculator.AddThreeNumbers(1, 2, 3)

fmt.Println("Sum of two numbers:", sumTwo)

fmt.Println("Sum of three numbers:", sumThree)

}

**12. Sobrecarga de Operadores**

package main

import "fmt"

type Product struct {

Price float64

}

func (p1 Product) Add(p2 Product) Product {

return Product{Price: p1.Price + p2.Price}

}

func main() {

product1 := Product{Price: 10.50}

product2 := Product{Price: 5.25}

total := product1.Add(product2)

fmt.Printf("Total Price: $%.2f\n", total.Price)

}

**13. Métodos Estáticos**

package main

import "fmt"

type Math struct{}

func (Math) Factorial(n int) int {

if n == 0 {

return 1

}

result := 1

for i := 1; i <= n; i++ {

result \*= i

}

return result

}

func main() {

fmt.Println("Factorial of 5:", Math{}.Factorial(5))

}

**14. Singleton**

package main

import (

"fmt"

"sync"

)

type Configuration struct{}

var instance \*Configuration

var once sync.Once

func GetInstance() \*Configuration {

once.Do(func() {

instance = &Configuration{}

})

return instance

}

func main() {

config1 := GetInstance()

config2 := GetInstance()

fmt.Println(config1 == config2) // Output: true

}

**15. Exceções/Erros Personalizados**

package main

import (

"errors"

"fmt"

)

type InsufficientBalanceError struct {

Message string

}

func (e InsufficientBalanceError) Error() string {

return e.Message

}

type BankAccount struct {

Balance float64

}

func (b \*BankAccount) Withdraw(amount float64) error {

if amount > b.Balance {

return InsufficientBalanceError{"Insufficient balance for withdrawal"}

}

b.Balance -= amount

return nil

}

func main() {

account := BankAccount{Balance: 100}

err := account.Withdraw(150)

if err != nil {

fmt.Println(err)

} else {

fmt.Println("Withdrawal successful")

}

}