핵심! Go 프로그래밍

- 3일차

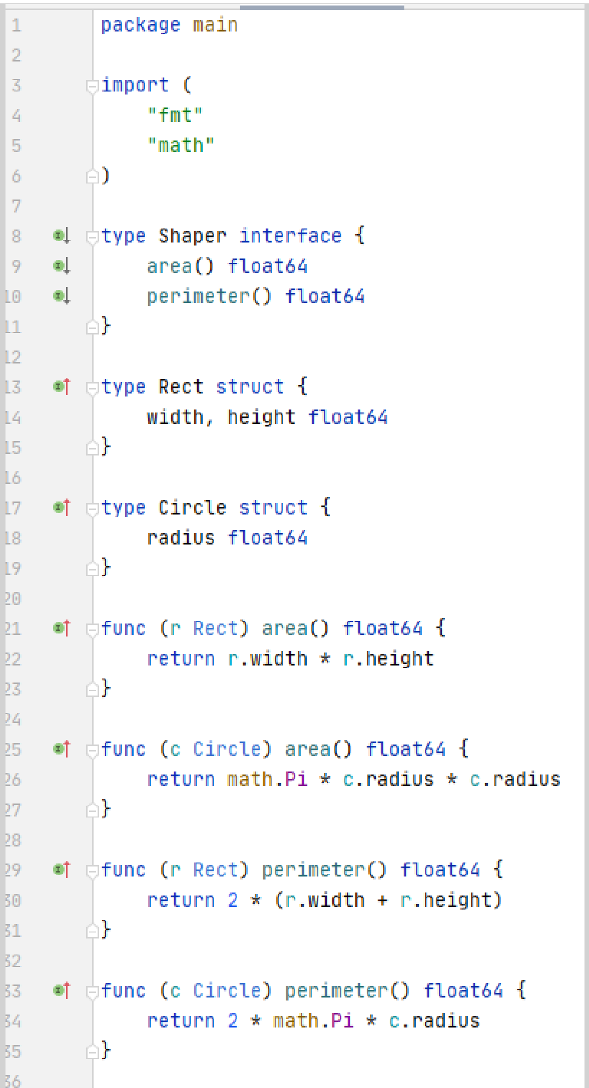
목차

- 인터페이스 (Interface)
- 뮤텍스 (Mutex)
- 풀 (Pool)
- 대기 그룹 (WaitGroup)

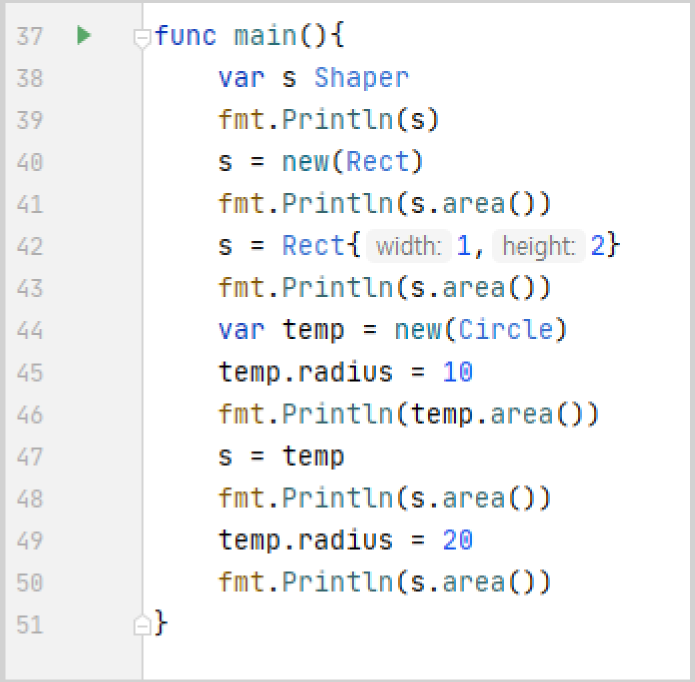
인터페이스

- 구체적인 동작을 구현할 메소드의 집합
- 인터페이스를 구현
  - 정의한 모든 메소드 구현
  - 다른 타입의 동작을 정의
- Interface를 키워드로 사용

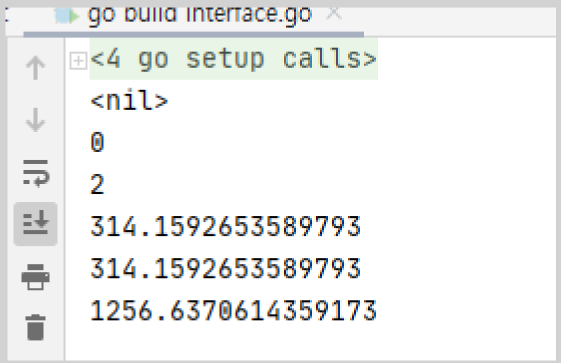
인터페이스



```go
package main

import (
	"fmt"
	"math"
)

type Shaper interface {
	area() float64
	perimeter() float64
}

type Rect struct {
	width, height float64
}

type Circle struct {
	radius float64
}

func (r Rect) area() float64 {
	return r.width * r.height
}

func (c Circle) area() float64 {
	return math.Pi * c.radius * c.radius
}

func (r Rect) perimeter() float64 {
	return 2 * (r.width + r.height)
}

func (c Circle) perimeter() float64 {
	return 2 * math.Pi * c.radius
}
```



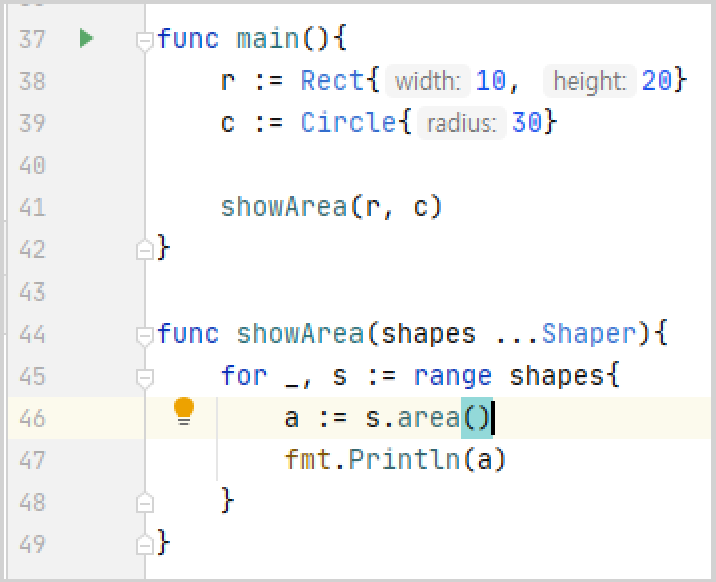
```go
func main(){
	var s Shaper
	fmt.Println(s)
	s = new(Rect)
	fmt.Println(s.area())
	s = Rect{ width: 1, height: 2}
	fmt.Println(s.area())
	var temp = new(Circle)
	temp.radius = 10
	fmt.Println(temp.area())
	s = temp
	fmt.Println(s.area())
	temp.radius = 20
	fmt.Println(s.area())
}
```



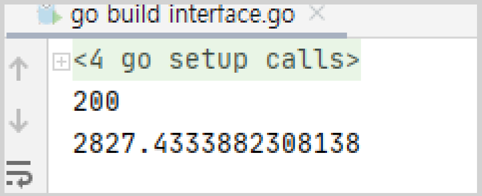
go build interface.go

```
<4 go setup calls>
<nil>
0
2
314.1592653589793
314.1592653589793
1256.6370614359173
```

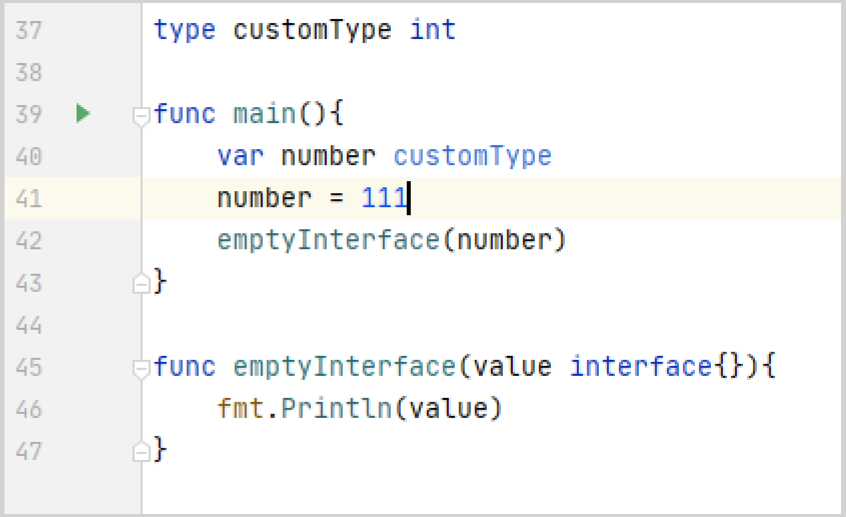
인터페이스



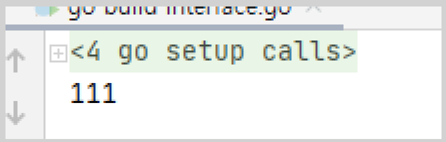
```go
func main(){
	r := Rect{ width: 10,  height: 20}
	c := Circle{ radius: 30}

	showArea(r, c)
}

func showArea(shapes ...Shaper){
	for _, s := range shapes{
		a := s.area()
		fmt.Println(a)
	}
}
```



```
go build interface.go
<4 go setup calls>
200
2827.4333882308138
```



```go
type customType int

func main(){
	var number customType
	number = 111
	emptyInterface(number)
}

func emptyInterface(value interface{}){
	fmt.Println(value)
}
```

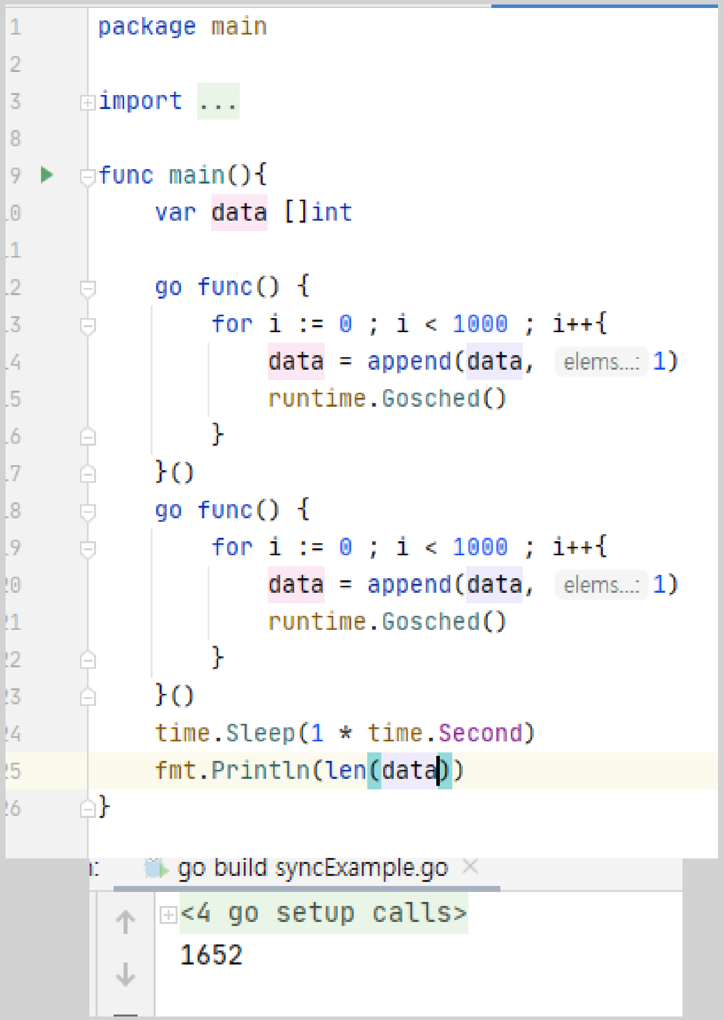


```
<4 go setup calls>
111
```

동기화 객체

- Mutex
  - 여러 스레드(고루틴)에서 공유되는 데이터를 보호할때 사용
- RWMutex
  - 읽기/쓰기 뮤텍스
- Cond
  - 조건 변수, 대기중인 고루틴 컨트롤
- Pool
  - 객체 풀, 객체 재사용 가능
- WaitGroup
  - 스레드 타이밍 컨트롤

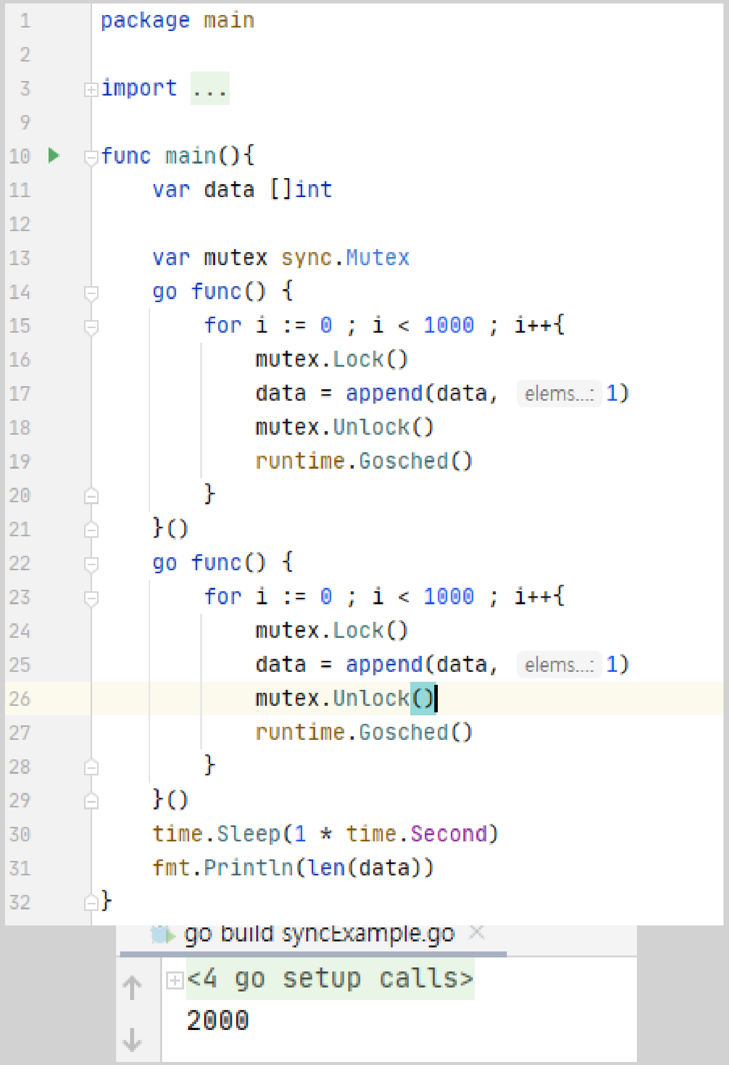
동기화 객체



```go
package main

import ...

func main(){
    var data []int

    go func() {
        for i := 0 ; i < 1000 ; i++{
            data = append(data, elems...: 1)
            runtime.Gosched()
        }
    }()
    go func() {
        for i := 0 ; i < 1000 ; i++{
            data = append(data, elems...: 1)
            runtime.Gosched()
        }
    }()
    time.Sleep(1 * time.Second)
    fmt.Println(len(data))
}
```

go build syncExample.go

```
<4 go setup calls>
1652
```

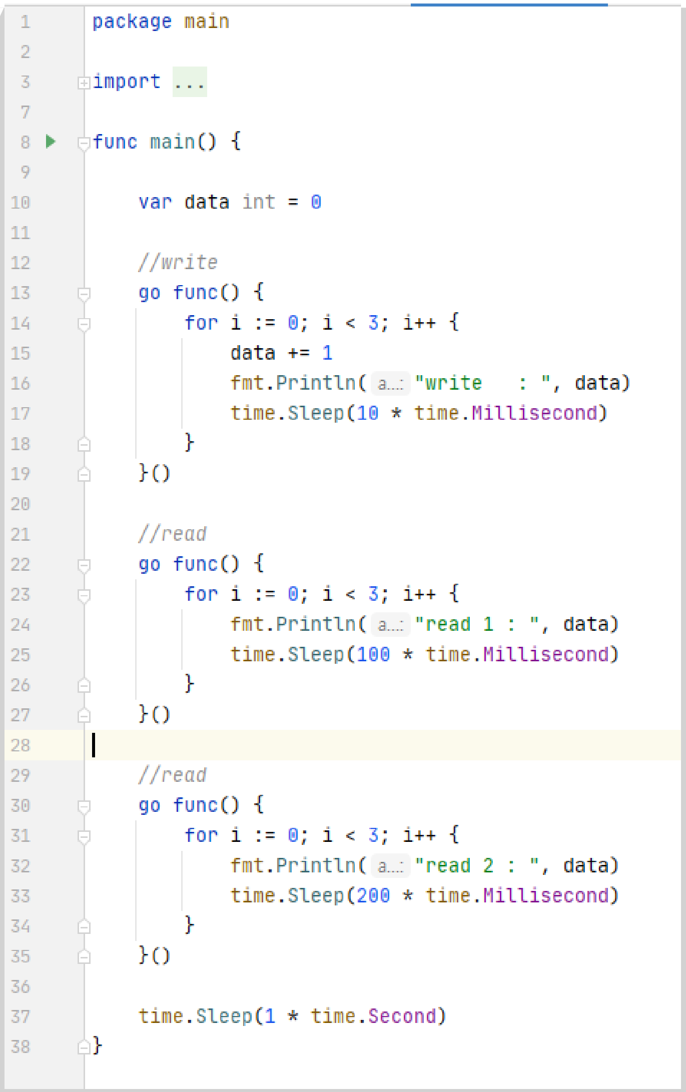


```go
package main

import ...

func main(){
    var data []int

    var mutex sync.Mutex
    go func() {
        for i := 0 ; i < 1000 ; i++{
            mutex.Lock()
            data = append(data, elems...: 1)
            mutex.Unlock()
            runtime.Gosched()
        }
    }()
    go func() {
        for i := 0 ; i < 1000 ; i++{
            mutex.Lock()
            data = append(data, elems...: 1)
            mutex.Unlock()
            runtime.Gosched()
        }
    }()
    time.Sleep(1 * time.Second)
    fmt.Println(len(data))
}
```

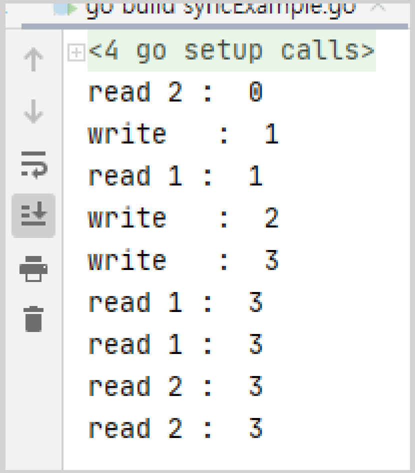
go build syncExample.go

```
<4 go setup calls>
2000
```

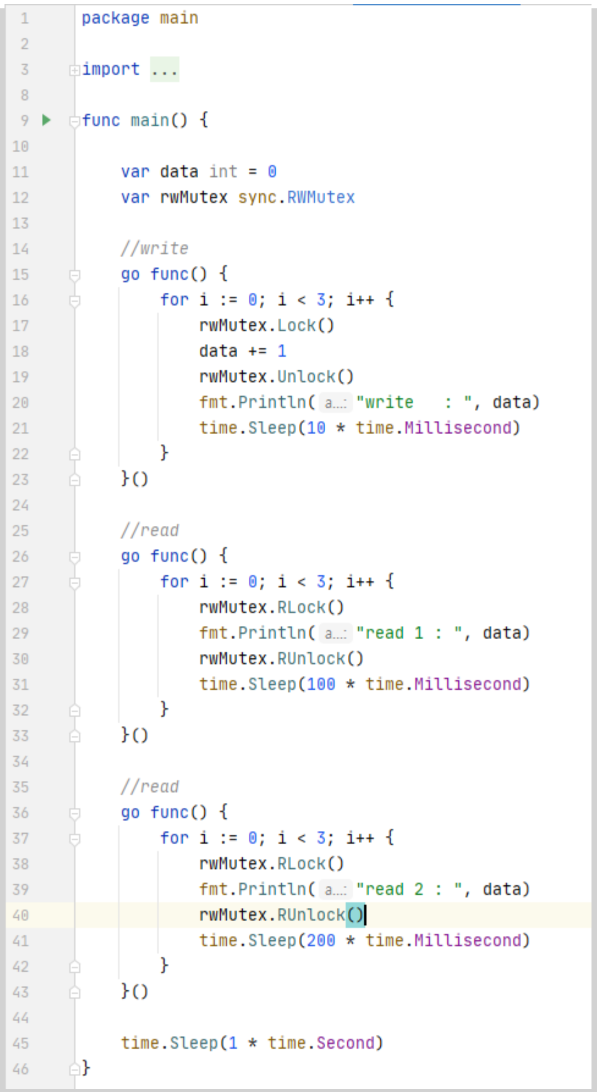
동기화 객체



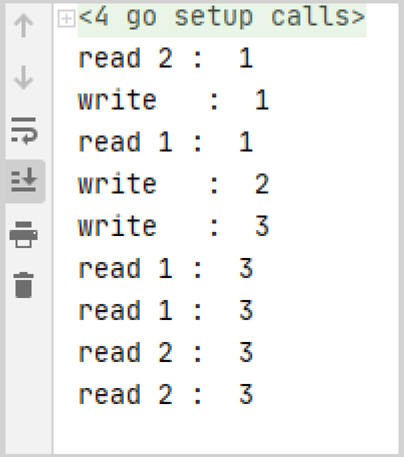
```go
package main

import ...

func main() {

    var data int = 0

    //write
    go func() {
        for i := 0; i < 3; i++ {
            data += 1
            fmt.Println("write   : ", data)
            time.Sleep(10 * time.Millisecond)
        }
    }()

    //read
    go func() {
        for i := 0; i < 3; i++ {
            fmt.Println("read 1 : ", data)
            time.Sleep(100 * time.Millisecond)
        }
    }()

    //read
    go func() {
        for i := 0; i < 3; i++ {
            fmt.Println("read 2 : ", data)
            time.Sleep(200 * time.Millisecond)
        }
    }()

    time.Sleep(1 * time.Second)
}
```



```
go build syncExample.go
<4 go setup calls>
read 2 :  0
write   :  1
read 1 :  1
write   :  2
write   :  3
read 1 :  3
read 1 :  3
read 2 :  3
read 2 :  3
```

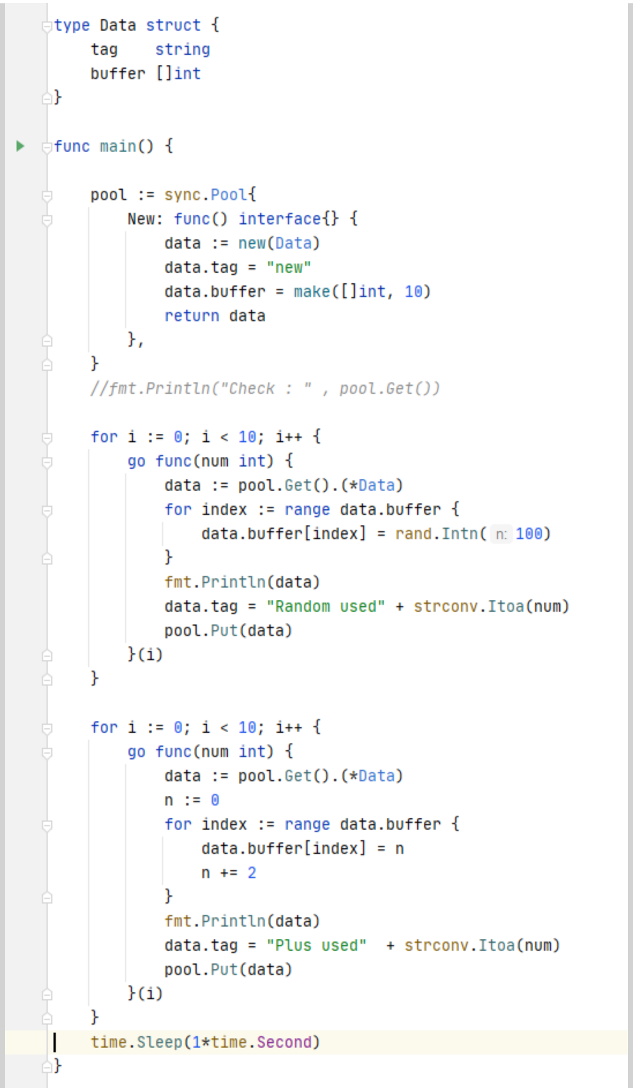


```go
package main

import ...

func main() {

    var data int = 0
    var rwMutex sync.RWMutex

    //write
    go func() {
        for i := 0; i < 3; i++ {
            rwMutex.Lock()
            data += 1
            rwMutex.Unlock()
            fmt.Println("write   : ", data)
            time.Sleep(10 * time.Millisecond)
        }
    }()

    //read
    go func() {
        for i := 0; i < 3; i++ {
            rwMutex.RLock()
            fmt.Println("read 1 : ", data)
            rwMutex.RUnlock()
            time.Sleep(100 * time.Millisecond)
        }
    }()

    //read
    go func() {
        for i := 0; i < 3; i++ {
            rwMutex.RLock()
            fmt.Println("read 2 : ", data)
            rwMutex.RUnlock()
            time.Sleep(200 * time.Millisecond)
        }
    }()

    time.Sleep(1 * time.Second)
}
```

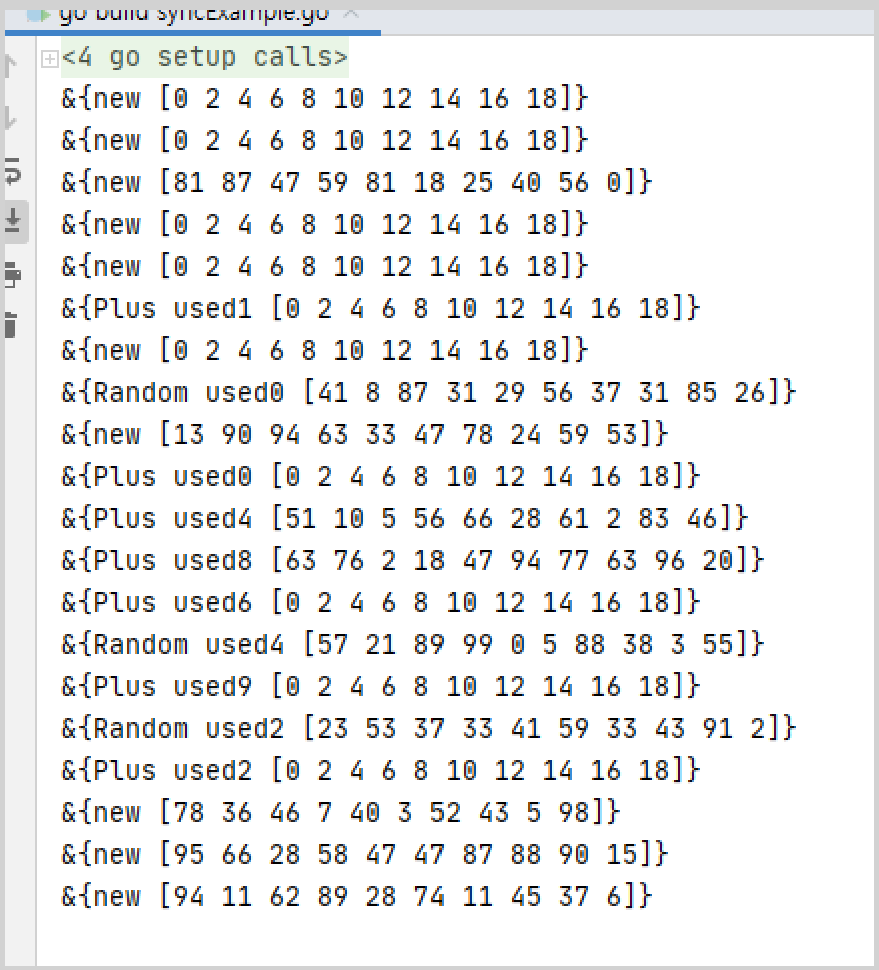


```
<4 go setup calls>
read 2 :  1
write   :  1
read 1 :  1
write   :  2
write   :  3
read 1 :  3
read 1 :  3
read 2 :  3
read 2 :  3
```

동기화 객체



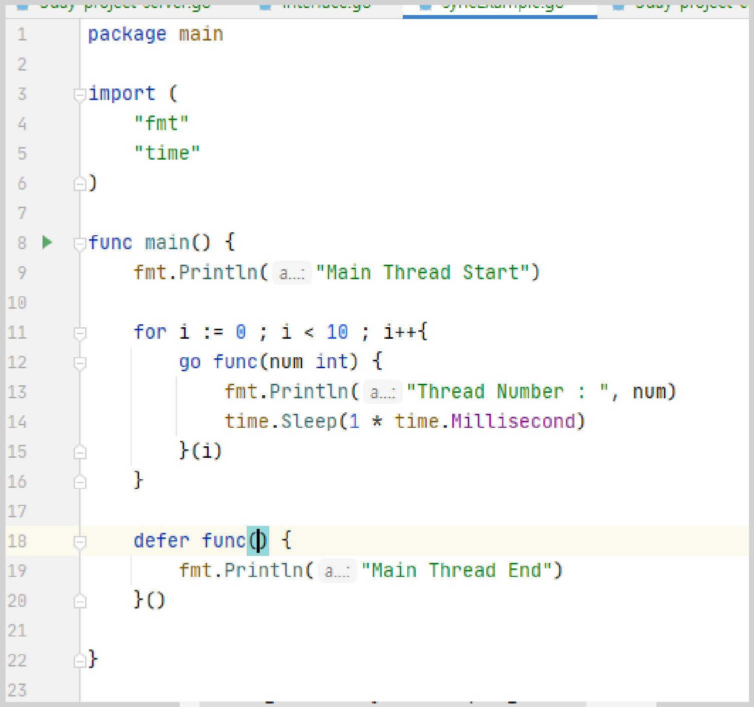
```go
type Data struct {
    tag    string
    buffer []int
}

func main() {

    pool := sync.Pool{
        New: func() interface{} {
            data := new(Data)
            data.tag = "new"
            data.buffer = make([]int, 10)
            return data
        },
    }
    //fmt.Println("Check : " , pool.Get())

    for i := 0; i < 10; i++ {
        go func(num int) {
            data := pool.Get().(*Data)
            for index := range data.buffer {
                data.buffer[index] = rand.Intn( n: 100)
            }
            fmt.Println(data)
            data.tag = "Random used" + strconv.Itoa(num)
            pool.Put(data)
        }(i)
    }

    for i := 0; i < 10; i++ {
        go func(num int) {
            data := pool.Get().(*Data)
            n := 0
            for index := range data.buffer {
                data.buffer[index] = n
                n += 2
            }
            fmt.Println(data)
            data.tag = "Plus used"  + strconv.Itoa(num)
            pool.Put(data)
        }(i)
    }
    time.Sleep(1*time.Second)
}
```



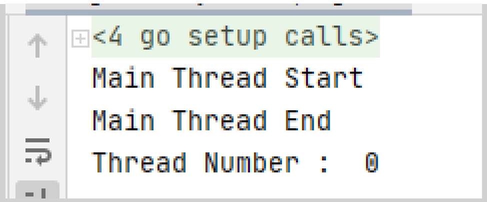
go build syncExample.go

```
<4 go setup calls>
&{new [0 2 4 6 8 10 12 14 16 18]}
&{new [0 2 4 6 8 10 12 14 16 18]}
&{new [81 87 47 59 81 18 25 40 56 0]}
&{new [0 2 4 6 8 10 12 14 16 18]}
&{new [0 2 4 6 8 10 12 14 16 18]}
&{Plus used1 [0 2 4 6 8 10 12 14 16 18]}
&{new [0 2 4 6 8 10 12 14 16 18]}
&{Random used0 [41 8 87 31 29 56 37 31 85 26]}
&{new [13 90 94 63 33 47 78 24 59 53]}
&{Plus used0 [0 2 4 6 8 10 12 14 16 18]}
&{Plus used4 [51 10 5 56 66 28 61 2 83 46]}
&{Plus used8 [63 76 2 18 47 94 77 63 96 20]}
&{Plus used6 [0 2 4 6 8 10 12 14 16 18]}
&{Random used4 [57 21 89 99 0 5 88 38 3 55]}
&{Plus used9 [0 2 4 6 8 10 12 14 16 18]}
&{Random used2 [23 53 37 33 41 59 33 43 91 2]}
&{Plus used2 [0 2 4 6 8 10 12 14 16 18]}
&{new [78 36 46 7 40 3 52 43 5 98]}
&{new [95 66 28 58 47 47 87 88 90 15]}
&{new [94 11 62 89 28 74 11 45 37 6]}
```

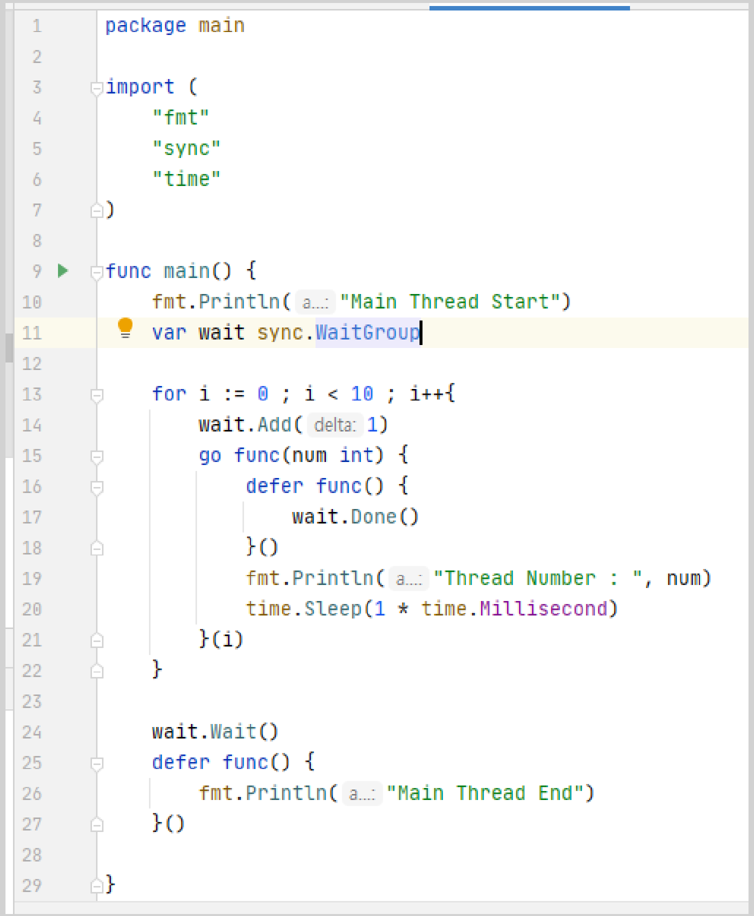
동기화 객체



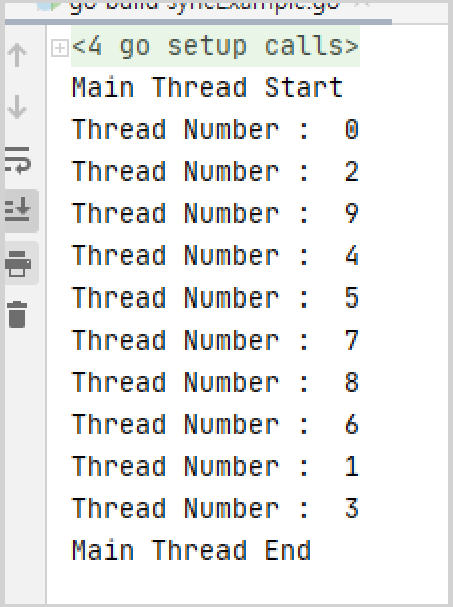
```go
package main

import (
	"fmt"
	"time"
)

func main() {
	fmt.Println( a...: "Main Thread Start")

	for i := 0 ; i < 10 ; i++{
		go func(num int) {
			fmt.Println( a...: "Thread Number : ", num)
			time.Sleep(1 * time.Millisecond)
		}(i)
	}

	defer func() {
		fmt.Println( a...: "Main Thread End")
	}()

}
```



```
<4 go setup calls>
Main Thread Start
Main Thread End
Thread Number :  0
```



```go
package main

import (
	"fmt"
	"sync"
	"time"
)

func main() {
	fmt.Println( a...: "Main Thread Start")
	var wait sync.WaitGroup

	for i := 0 ; i < 10 ; i++{
		wait.Add( delta: 1)
		go func(num int) {
			defer func() {
				wait.Done()
			}()
			fmt.Println( a...: "Thread Number : ", num)
			time.Sleep(1 * time.Millisecond)
		}(i)
	}

	wait.Wait()
	defer func() {
		fmt.Println( a...: "Main Thread End")
	}()

}
```



```
<4 go setup calls>
Main Thread Start
Thread Number :  0
Thread Number :  2
Thread Number :  9
Thread Number :  4
Thread Number :  5
Thread Number :  7
Thread Number :  8
Thread Number :  6
Thread Number :  1
Thread Number :  3
Main Thread End
```