JavaAssginment

Theory QA

Q1:  What’s the difference between a goroutine and an operating system thread?  
Ans: i- Every concurrent executing activity in Go language is known as goroutine.  
Whereas thread is process in operating system which is responsible for executing some tasks.  
ii- Goroutines are managed by Go runtime, whereas thread is managed by operating system kernel.  
iii- Goroutines do not have id because Go does not have thread local storage, on the other hand threads have their unique id.  
iv- Goroutines have easy communication medium called channel which has low latency, then that of inter communication between threads.            
v- Goroutines has lower startup time than threads  
vi- Goroutines have growable segmented stacks, but threads do not have it.

Q2: Can constant be computed in Go?  
Ans: Constant cannot be computed in Go they provide complete safety in regard to the value they hold.

Q3: What does the Go ecosystem use for package and dependency management?  
Ans: Go modules helps in managing dependency in Go ecosystem with the  
i- go.mod file being the root of dependencies management in Go (similar to package.json)  
ii- go.sum is generated file which developer need not to modify  
go mode tidy - bind the current imports in the project and packages listed in go.mod  
go mode vendor - generates a vendor directory with the versions available. It copies all third-party dependencies to a vendor folder in your project root  
go clean -modcache - is used to clear the mod cache which is stored at $GOPATH/pkg/mod

Q4. How would you succinctly swap the values of two variables in Go?  
Ans:package main  
import (  
"fmt"  
)  
func main() {  
x := 10  
y := 20  
fmt.Printf("Before swapping, x: %d, y: %d\n", x, y)  
x, y = y, x  
fmt.Printf("After swapping, x: %d, y: %d\n", x, y)  
}

Q5. Do you have any preferences for error handling methodologies in Go?  
Ans: Creating custom errors:The interface contains a single method Error() that returns an error message as a string.  
String-based errors:String-based errors can be created using two out-of-the-box options in Golang and are used for simple errors that just need to return an error message.  
Custom error with data:The PathError implements the Error() function and therefore satisfies the error interface. The implementation of the Error() function now returns a string with the path of the PathError struct. You can now use PathError whenever you want to throw an error.  
Error handling in functions Most of the time errors are not directly handled in functions but are returned as a return value instead. Here we can take advantage of the fact that Golang supports  
multiple return values for a function.

Defer, panic and recover are the methods used to handle errors in GoLang.

Q6. What is a pointer and when would you use it?  
Ans: Pointers are used to store and manage the addresses of dynamically allocated blocks of memory. Such blocks are used to store data objects or arrays of objects.

Q7. Describe the difference between sync.Mutex and sync.RWMutex?  
Ans:mutex:  
1. There are two operations for mutex: acquire lock and release lock  
2. When a goroutine acquires the mutex, no goroutine can acquire the mutex. You can only wait for the goroutine to release the mutex  
3. Mutex is applicable to the situation that the number of read and write operations is almost the same  
4. Both reading and writing can be put into the mutex  
Read write lock:  
1. There are four operations of read lock, read lock, read unlock, write lock and write unlock  
2. There can be at most one write lock and multiple read locks (the maximum data is related to the number of CPU s)  
3. The priority of the write lock is higher than that of the read lock. This is because the write lock has been blocked to prevent too many read locks  
4. When one goroutine obtains a write lock, other goroutines cannot obtain a read lock or a write lock until the write lock is released

Q9. Are channels and maps safe for concurrent access?  
Ans:Maps are not safe for concurrent use: it's not defined what happens when you read and write to them simultaneously. If you need to read from and write to a map  
from concurrently executing goroutines, the accesses must be mediated by some kind of synchronization mechanism.

Q10. How would you sort a slice of custom structs?

Ans**:** GO has a **sort** package that provides utility primitives for the sorting of slices and user-defined types. Any collection can be sorted by the **Sort** function of **sort** package of GO it if implements the **sort.Interface**