**DEPARTMENT OF INFORMATION TECHNOLOGY**

**COURSE CODE:** DJ19ITC802 **DATE: 08-3-2025**

**COURSE NAME:** Design Patterns Laboratory **CLASS:** BE - IT

**EXPERIMENT NO. 5**

**CO/LO:** Identify and apply the most suitable design pattern to address a given application design problem.

**AIM:** Implement the CoR Pattern using any language of your choice for the given problem statement. (A system in which several managers and executives can respond to a purchase request or hand it off to a superior. You are free to have your own set of rules to approve the orders.)

**DESCRIPTION:**

Chain of responsibility pattern is used to achieve loose coupling in software design where a request from the client is passed to a chain of objects to process them. Later, the object in the chain will decide themselves who will be processing the request and whether the request is required to be sent to the next object in the chain or not. This pattern is recommended when multiple objects can handle a request and the handler doesn’t have to be a specific object. Also, the handler is determined at runtime. Please note that a request not handled at all by any handler is a valid use case.

**SOURCE CODE:**

**Roles : TeamLead, SeniorManager, CFO**

**Approval logic : (budget constraints, high-priority approvals).**

package main

import "fmt"

// PurchaseRequest struct

type PurchaseRequest struct {

    amount   int

    priority string // Can be "low", "medium", or "high"

}

// Approver interface

type Approver interface {

    SetNext(Approver)

    HandleRequest(\*PurchaseRequest)

}

// BaseHandler struct (implements common logic for chaining)

type BaseHandler struct {

    next Approver

}

func (b \*BaseHandler) SetNext(next Approver) {

    b.next = next

}

func (b \*BaseHandler) PassToNext(request \*PurchaseRequest) {

    if b.next != nil {

        b.next.HandleRequest(request)

    } else {

        fmt.Println("No one can approve this request. Manual approval required.")

    }

}

// TeamLead struct

type TeamLead struct {

    BaseHandler

}

func (tl \*TeamLead) HandleRequest(request \*PurchaseRequest) {

    if request.amount <= 500 && request.priority == "low" {

        fmt.Printf("Team Lead approved the purchase of $%d (Priority: %s)\n", request.amount, request.priority)

    } else {

        tl.PassToNext(request)

    }

}

// Manager struct

type Manager struct {

    BaseHandler

}

func (m \*Manager) HandleRequest(request \*PurchaseRequest) {

    if request.amount <= 2000 {

        fmt.Printf("Manager approved the purchase of $%d (Priority: %s)\n", request.amount, request.priority)

    } else {

        m.PassToNext(request)

    }

}

// SeniorManager struct

type SeniorManager struct {

    BaseHandler

}

func (sm \*SeniorManager) HandleRequest(request \*PurchaseRequest) {

    if request.amount <= 5000 {

        fmt.Printf("Senior Manager approved the purchase of $%d (Priority: %s)\n", request.amount, request.priority)

    } else {

        sm.PassToNext(request)

    }

}

// Director struct

type Director struct {

    BaseHandler

}

func (d \*Director) HandleRequest(request \*PurchaseRequest) {

    if request.amount <= 10000 {

        fmt.Printf("Director approved the purchase of $%d (Priority: %s)\n", request.amount, request.priority)

    } else {

        d.PassToNext(request)

    }

}

// VicePresident struct

type VicePresident struct {

    BaseHandler

}

func (vp \*VicePresident) HandleRequest(request \*PurchaseRequest) {

    if request.amount <= 20000 || request.priority == "high" {

        fmt.Printf("Vice President approved the purchase of $%d (Priority: %s)\n", request.amount, request.priority)

    } else {

        vp.PassToNext(request)

    }

}

// CFO struct

type CFO struct {

    BaseHandler

}

func (c \*CFO) HandleRequest(request \*PurchaseRequest) {

    if request.amount <= 50000 {

        fmt.Printf("CFO approved the purchase of $%d (Priority: %s)\n", request.amount, request.priority)

    } else {

        c.PassToNext(request)

    }

}

// CEO struct

type CEO struct {

    BaseHandler

}

func (c \*CEO) HandleRequest(request \*PurchaseRequest) {

    if request.amount > 50000 {

        fmt.Printf("CEO approved the purchase of $%d (Priority: %s) as a special case\n", request.amount, request.priority)

    } else {

        c.PassToNext(request)

    }

}

func main() {

    // Creating the chain of responsibility

    teamLead := &TeamLead{}

    manager := &Manager{}

    seniorManager := &SeniorManager{}

    director := &Director{}

    vp := &VicePresident{}

    cfo := &CFO{}

    ceo := &CEO{}

    // Setting up the chain order

    teamLead.SetNext(manager)

    manager.SetNext(seniorManager)

    seniorManager.SetNext(director)

    director.SetNext(vp)

    vp.SetNext(cfo)

    cfo.SetNext(ceo)

    // Example purchase requests with different priorities

    requests := []\*PurchaseRequest{

        {300, "low"},

        {1500, "medium"},

        {4000, "high"},

        {12000, "medium"},

        {25000, "high"},

        {60000, "medium"},

    }

    // Processing requests

    for \_, req := range requests {

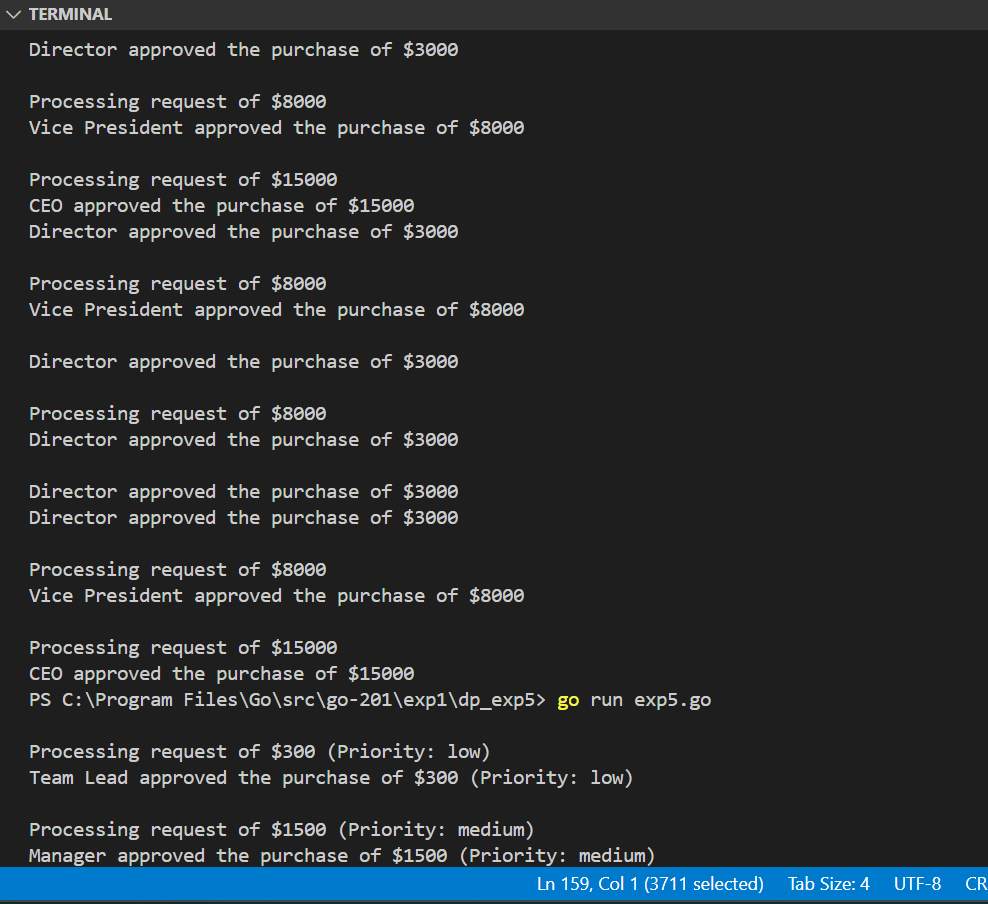
        fmt.Printf("\nProcessing request of $%d (Priority: %s)\n", req.amount, req.priority)

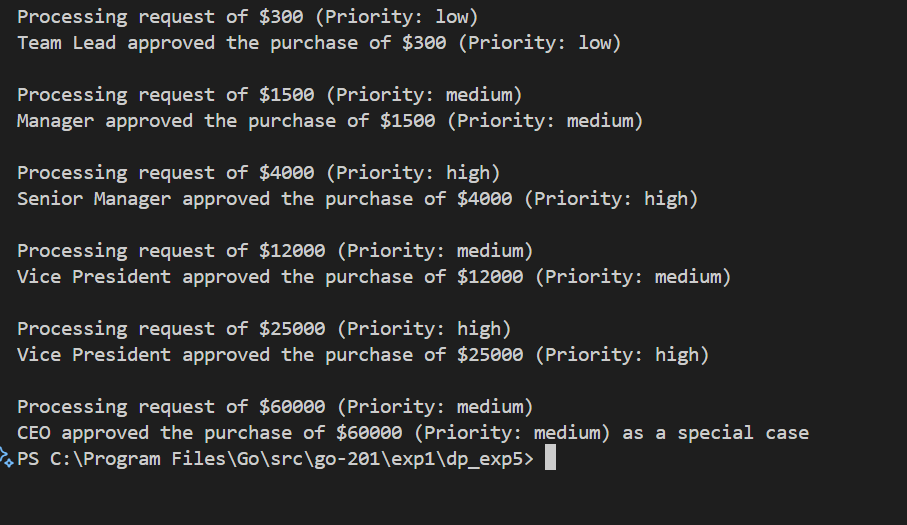
        teamLead.HandleRequest(req)

    }

}

**OUTPUT**

****

****

**CONCLUSION:**

Thus we implemented the Chain of responsibility patter in golang with multiple roles assigned and approval logic defined.