CHAIN OF RESPONSIBILITY DESIGN PATTERN

Carrie Ann Crot
Behavioral pattern
- Concerned with algorithms and assignments of responsibility between objects
- Describe the pattern of communication between objects and classes
- Characterize complex control flow that’s difficult to follow at runtime

Avoid coupling the sender of a request to its receiver
- By giving more than one object a chance to handle the request

Chain the receiving objects and pass the request along the chain until an object handles it
IN GENERAL TERMS

- Describes how to handle a single request by a chain of multiple handler objects.
- The request has to be processed by only one handler object from this chain.
- The determination of processing the request is decided by the current handler.
- If the current handler object is able to process the request, then the request will be processed in the current handler.
- Otherwise the current handler object needs to shirk responsibility and push the request to the next chain handler object.
- Pattern continues on until the request is processed.
Applicability
- You want to decouple a request’s sender and receiver
- Multiple objects, determined at runtime, are candidates to handle the request
- You don’t want to specify handlers explicitly in your code

Consequences
- Sender and receiver have not explicit knowledge of each other
- Receipt is not guaranteed - some request might not get handled
- The chain of handlers can be modified dynamically
At a University, to purchase new equipment requires prior approval, the level of approval depends on how much money you intend to spend.

For example the chain is:
- Manager → Lab Director → Department Business Manager → Vice Chancellor of Research

Chain of responsibility is utilized to check who is responsible to approve your expenditure.
import java.io.*;
abstract class PurchasePower {
    protected final double base = 500;
    protected PurchasePower successor;
    public void setSuccessor(PurchasePower successor)
    {
        this.successor = successor;
    }
    abstract public void processRequest(PurchaseRequest request);
}
class ManagerPPower extends PurchasePower{
    private final double ALLOWABLE = 10*base;
    public void processRequest(PurchaseRequest request){
        if(request.getAmount()<ALLOWABLE)
        {
            System.out.println("Manager will approve \\
$"+request.getAmount());
        }
        else
        {
            if (successor !=null)
            {
                sucessor.processRequest(request);
            }
        }
    }
}

http://www.javacamp.org/designPattern/chains.html
class LabDirectorPPower extends PurchasePower {
    private final double ALLOWABLE = 20 * base;
    public void processRequest(PurchaseRequest request) {
        if (request.getAmount() < ALLOWABLE )
            System.out.println("Lab Director will approve $"+request.getAmount());
        else
            if (successor != null)
                successor.processRequest(request);
    }
}

//Above class method is copied for
    //Department Business Manager
    //Vice Chancellor of Research
//class PurchaseRequest
    //is a helper class that hold the request information
class CheckAuthority {
    public static void main(String[] args) {
        //create an object each for Manager, Lab Director, Dept Business Manager and Vice Chancellor of Research
        ManagerPPower manager = new ManagerPPower();
        LabDirectorPPower labDirector = new LabDirectorPPower();
        DeptBusinessManagerPPower deptBusManager = new DeptBusinessManagerPPower();
        ViceChancellorOfResearchPPower viceChancellor = new ViceChancellorOfResearchPPower();

        //Build the responsibility chain to handle the different requests from the client
        manager.setSuccessor(labDirector);
        labDirector.setSuccessor(deptBusManager);
        deptBusManager.setSuccessor(viceChancellor);

        //read input value and send to manager for screening, to see who is able to approve request
        try {
            while (true) {
                System.out.println("Enter the amount to check who should approve your expenditure.");
                System.out.print("> ");
                double d = Double.parseDouble(new BufferedReader(new InputStreamReader(System.in)).readLine());
                manager.processRequest(new PurchaseRequest(0, d, "General"));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
ccrot@sheridan:~/cspp51023/presentation$ javac CheckAuthority.java
ccrot@sheridan:~/cspp51023/presentation$ java CheckAuthority
Enter the amount to check who should approve your expenditure.
>500
Manager will approve $500.0
Enter the amount to check who should approve your expenditure.
>5500
Lab Director will approve $5500.0
Enter the amount to check who should approve your expenditure.
>15000
Department Business Manager will approve $15000.0
Enter the amount to check who should approve your expenditure.
>25000
Vice Chancellor of Research will approve $25000.0
Enter the amount to check who should approve your expenditure.
>55000
Your request for $55000.0 needs a board meeting!
Enter the amount to check who should approve your expenditure.
>