LAB 8

BEHAVIORAL DESIGN PATTERNS IN SMART WHEEL CHAIR SYSTEM

Prepared by :

Akeel Ahmed - 190028C

Thilina Ilesinghe - 190238U

Hafeel - 190211G

Contents

Chain Responsibility Pattern 3

Scenario 3

Problem 3

Solution 3

Class Diagram 4

Sequence Diagram 4

Code 5

Chain Responsibility Pattern

Scenario

Problem

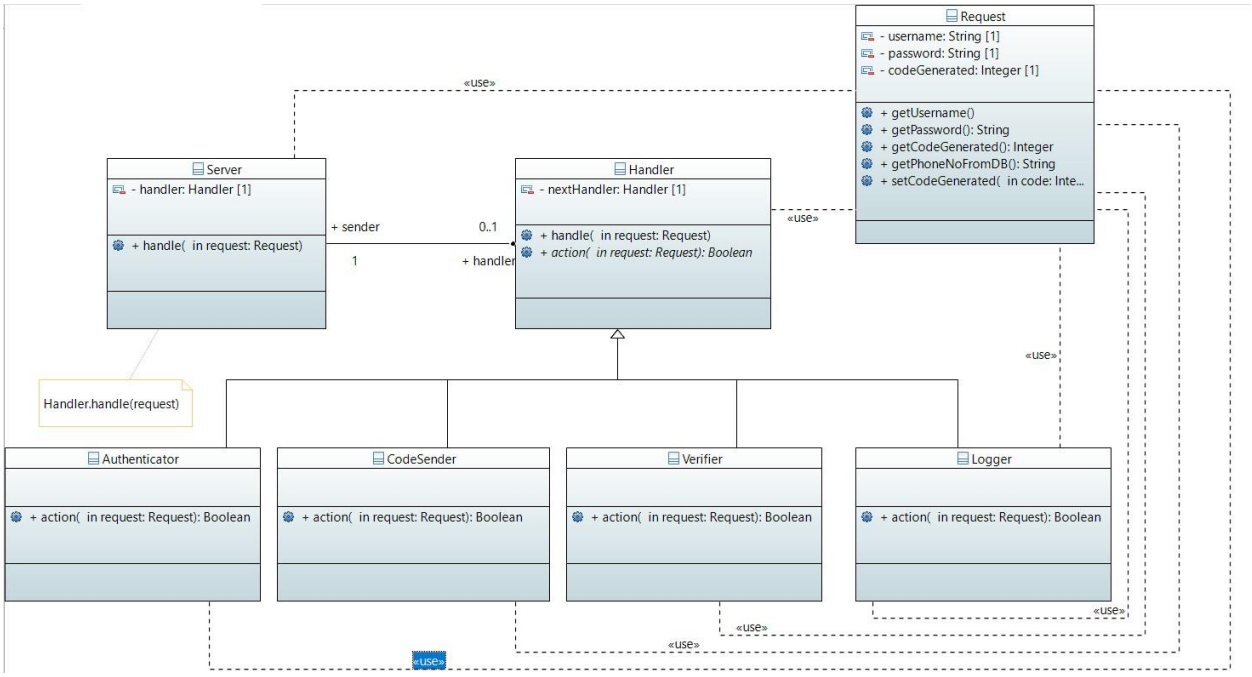
The request for verification is made upon attempt to login from a device. The authentication using phone number is needed to be an added protection. This added protection is only needed under certain instances like login attempt from a new device while logged in on another device, while it could be skipped under instances like a recently logged in device. In such an instance, we would need to handle the request step by step before the logging in happens. The order of the steps would differ, thus we need to have control of the order of the request handlers as well. There could be needs to add new protection features as well in the future , so the code must be extensible for new protection features.

Solution

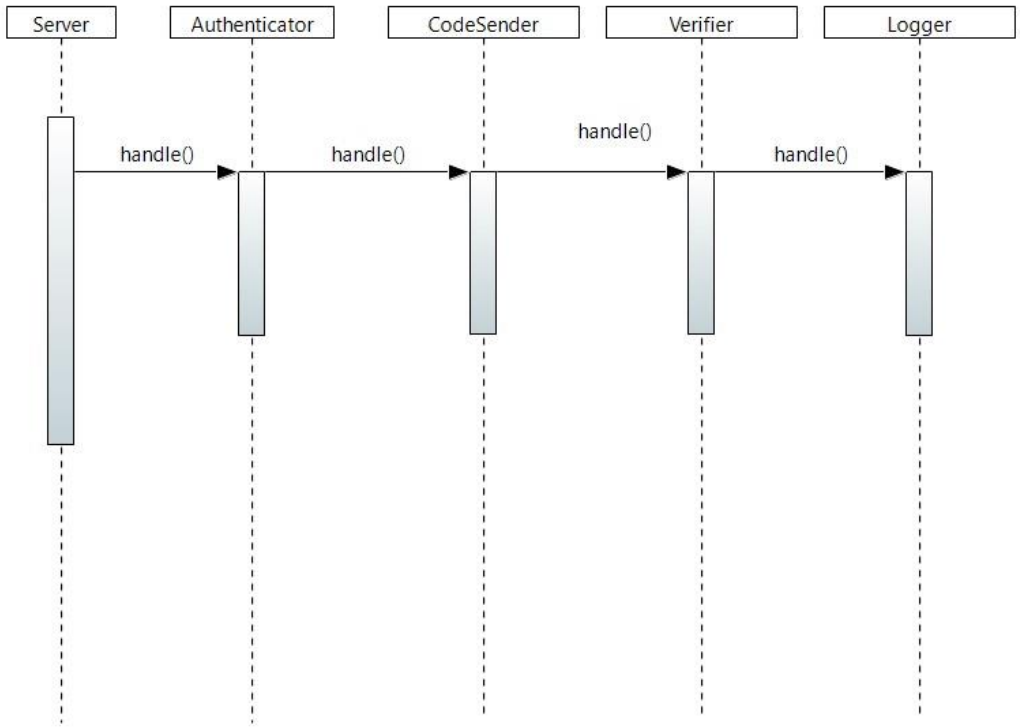
The chain responsibility pattern is used to achieve loose coupling in software design. This pattern provides the possibility of easily changing the order of request handling steps without altering the codes of the concrete classes. That is, we could omit the additional code to phone number protection feature when required without altering the Authenticator class in the code below. Also, a new feature like captcha verification or secret question verification could also be added without altering any other part of the code. We would only need to add a class inheriting the Handler abstract class and coding the action to be performed within the override action method in it.

Here, the server acts as the sender while the authenticator, code sender, verifier and Logger are the receivers.

Class Diagram



Sequence Diagram



Code

```
class Request{
    // A database query is a request for data from a database.
    private String username,password;
    private int codeGenerated;
    Request(String username,String password){
        this.username = username;
        this.password = password;
    }
    public String getUsername() {
        return this.username;
    }
    public String getPassword() {
        return this.password;
    }
    public String getPhoneNoFromDB() {
        //  get from database
        return "1234567890";
    }
    public int getCodeGenerated() {
        return this.codeGenerated;
    }
    public void setCodeGenerated(int codeGenerated) {
        this.codeGenerated = codeGenerated;
    }

}
abstract class Handler{
    private Handler nextHandler;
    Handler(Handler nextHandler){
        this.nextHandler = nextHandler;
    }
    void handle(Request request){
        if (action(request) && nextHandler!=null){
            nextHandler.handle(request);
        }
        System.out.println("Done");
        System.exit(0);

    }
    abstract boolean action(Request request);
}
class Server {
```

```
    private Handler handler;
    Server(Handler handler){
        this.handler = handler;
    }
    void handle(Request request){
        handler.handle(request);
    }
}
class Authenticator extends Handler{
    Authenticator(Handler nextHandler){
       super(nextHandler);
    }
    @Override
    boolean action(Request request) {
        boolean isValid = false;
        if (request.getUsername().equals("admin") &&
request.getPassword().equals("password")){
            System.out.println("Authenticated");
            isValid = true;
        }
        System.out.println("VALID : "+isValid);
        return isValid; // if valid go to next step
    }

}

class CodeSender extends Handler{

    CodeSender(Handler nextHandler) {
        super(nextHandler);
    }

    @Override
    boolean action(Request request) {
        int codeGenerated = 111222; //can use random to generate
        System.out.println("Code Generated = 111 222");

        //  Assume always successful
        boolean success = true;
        request.setCodeGenerated(codeGenerated);
        System.out.println("Sending code to "+request.getPhoneNoFromDB()+" :
"+success);
        return success;
    }
```

```java
}


class Verifier extends Handler{

    Verifier(Handler nextHandler) {
        super(nextHandler);
    }

    @Override
    boolean action(Request request) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter pin code to verify: ");
        int code = sc.nextInt();
        System.out.println("Verified Code");
        sc.close();
        return (code == request.getCodeGenerated());
    }


}

class Logger extends Handler{

    Logger(Handler nextHandler) {
        super(nextHandler);
    }
    @Override
    boolean action(Request request) {
        System.out.println("Logging In");
        return true;
    }

}
class Scenario1{
    void test(){
        Logger log = new Logger(null);
        Verifier verify = new Verifier(log);
        CodeSender codeSender = new CodeSender(verify);
        Authenticator auth = new Authenticator(codeSender);
        Request request =  new Request("admin", "password");
        Server server = new Server(auth);
        server.handle(request);
    }
```

```
}

public class chainofResponsibilty {
    public static void main(String[] args) {
        Scenario1 scene1 = new Scenario1();
        scene1.test();
    }
}
```