**SE311 Term Project**

**A Pluggable Authentication Mechanism**

**Gözde Güngör - 20160601023**

**Yiğit Ege Miman - 20160602022**

**Cem Özcan - 20170601024**

**DESIGN PATTERNS**

In our app we used different design patterns to make the system more solid. We paid attention to choose the proper patterns for required parts of the project. Here are the design patterns that we used and the explanation of why we needed them.

**1- Singleton Pattern:** We have such classes that need to have only one reference. For example; there should not exist more than one reference for OS (Operating System) class. The other singleton classes are Authentication Mechanism classes (such as LOCAL, LDAP and KERBEROS) and the Database class.

**2- Adapter Pattern:** The OS class authenticates the user by checking the user in different authentication mechanisms. All authentication mechanisms have the similar operations in different names. We have authentication method for LOCAL as local\_authenticate , for LDAP as ldap\_authenticate and for KERBEROS as kerberos\_authenticate. We designed middlewares (adapters) for this purpose. In this way, OS class can use the adapters LOCAL Adapter, LDAP Adapter, KERBEROS Adapter. All of these adapter classes has one generic method “authentication” to use.

**3- Iterator Pattern:** We need to store some users and authenticate on these users. In the Database class, while searching for some particular User we need to iterate over user list and search for a user that matches with given credentials. For this purpose we implemented the User Iterator class.

**4- Command Pattern:** The credential packages, that comes to authentication mechanisms, need to be used by Database while searching for a user. We decided to transfer this package with a command. The command receiver (Database), and the command creator (Authentication Mechanisms) do not need to know each other. We called this command as the CheckCommand.

**5- Template Pattern:** Our CheckCommands carry credential packages without a parameter for authentication mechanism identifier. While they are invoking the Database check method, they put the parameters of user credential. We implemented LocalCheckCommand, LDAPCheckCommand and KERBEROSCheckCommand classes to get authentication mechanism identifiers from different subcommands

**6- Observer Pattern:**  We implemented an observer pattern to get the results of user checking operation done in the Database. The OS class becomes Observer attached to a subject which is AuthSubject class. When the user checking operation done in Database class, the results are given to that subject. AuthSubject notifies the observers. OS class reads the result and determines the next operation. If the user checking operation get success, the User object can be accessible from the subject.

**7- Factory Pattern:** Session is created if the user authentication get success. We assumed two different Sessions, which are TerminalSession and GUISession. We create one of these sessions with respect to user’s desire. We designed a Factory pattern for the session creation process. Our SessionFactory class takes a type parameter according to the user’s choice then creates a Session. OS assigns an uid from authenticated User object to the session.

**CLASSES AND METHODS**

**-Main-**

Gets OS instance and AuthSubject instance. Attaches OS to AuthSubject. Gets username and password from user and starts authentication process on OS.

**-OS-**

Manages the authentication process with 3 different authentication mechanisms. Implements Observer interface. Listens the AuthSubject for a state message as if authentication is success or fail for the relevant authentication mechanism. Depending on the state message, it determines either to continue with another authentication mechanisms or create a session for user.

**void authenticate(String username, String password, AuthAdapter adapter):** Calls the authenticate method of given adapter with parameters of username and password. If the given adapter is null, it assigns to LOCALAdapter.

**void Update(AuthSubject authSubject):** Ifa change happens in the state of subject, this method reads the state message and determines the next operation.

**void createSession() :** Creates a GUI or Terminal session according to the user choice by calling the sessionCreate() method of sessionFactory class. Sets uid for the created session.

**-Observer-**

This is an interface for the classes that need to be attached to AuthSubject.

(In our case OS class is Observer)

**-AuthSubject-**

Stores 2 states about authentication process. Message and User states are reachable from attached Observers, when this class notifies them.

**void Attach():** Adds observer object to the list.

**void Notify():** Sends a class reference to each observer by calling Update() method on each of them.

**void setStates(String stateMessage, User stateUser)** Assigns given parameters to states. Then calls the Notify() method.

**-AuthAdapter-**

It is an interface that includes a method called authenticate() to be implemented in each of the authmechanism adapters.

**void authenticate(String name, String pwd)**

**-KERBEROSAdapter- / -LDAPAdapter- / -LOCALAdapter-**

Each class implements AuthAdapter and its authenticate method.

**-AuthMechanism-**

It is an abstract class that contains only the doCheck() method implemented for all subclasses.

**void doCheck():** Performs Execute() method of CheckCommand class on the relevant command of each authentication mechanism in the doCheck method()

**-KERBEROS- / -LDAP- / -LOCAL-**

Each extends the AuthMechanism class and has an additional authenticate method with different names.

**void krb\_authenticate (String username, String password):**

**void ldap\_authenticate (String username, String password):**

**void local\_authenticate (String username, String password):**

Each method creates a relevant command and assigns to a variable. All of them invokes doCheck() method to execute that command.

**-CheckCommand-**

It is an abstract class including Execute() method and abstract getMechanism() method to be implemented in sub check commands.

**void Execute():** This method invokes the checkUser() method with the username, password and mechanism identifier parameters . The mechanism identifier comes from an abstract method which is implemented by subclasses.

**abstract String getMechanism()**

**-KerberosCheckCommand- / -LocalCheckCommand- / -LDAPCheckCommand-**

Each class extends the CheckCommand class and implements the getMechanism().

**String getMechanism():** Returns a mechanism identifier.

**-Database-**

This class stores the User objects. Checks for the user existance and sets the states of AuthSubject.

**User checkUser(String username, String pwd, String mech):** This iterates over the User list with a UserIterator and checks for the given parameters if they are matches any User in that list. After checking operation it sets the states of AuthSubject with the message and found (or null) User object.

**-User-**

Includes User attributes and get() methods.

**-AbstractIterator-**

Includes required iteration methods to be implemented.

**-UserIterator-**

Accepts a User list and iterates over it.

**void First():** Sets the cursor position to 0.

**void Next():** Increments the value of cursor position.

**Boolean IsDone():** Returns if the cursor position reached the end of User list.

**User CurrentItem():** Returns User object from User list at the cursor position.

**-Session-**

It is an abstract class including one abstract method getSessionType() to be implemented.

**void setUid(int Uid):** Assigns given parameter to uid variable.

**int getUid():** Returns the value of uid variable.

**-GUISession- / -TerminalSession-**

Each implement getSessionType() method.

**String getSessionType():** Returns the session type.

**-SessionFactory-**

Provides a method whenever a session is needed to be created.

**Session sessionCreate(String userSessionChoice):** According to the user choice creates a session





