****

**SE 311 SOFTWARE ARCITECHTURE 2019-2020 SPRING GROUP PROJECT**

**A PLUGGABLE AUTHENTICATION MECHANISM**

**PROJECT MEMBERS**

SERCAN KAVDIR

EMRE SERBEST

HATİCE NUR RUŞEN

İREM ULUSOY

**LECTURER**

ASST. PROF UFUK ÇELİKKAN

**PATTERNS MOTIVATION**

**FACADE PATTERN**

Facade pattern is used to hide the complexity from the user. In our case, user can reach a single method (authanticate()) which activates the whole system. Although the user could login the system without facade pattern, in order to decrease the number of the process, to serve an easy to use system and to provide a logical flow, facade pattern is implemented.

**PARTICIPANTS**

**Facade:**

OperatingSystem

**Subsystem Classes:**

LocalAPIPluggableAuthentication

LdapAPIPluggableAuthentication

KerberosAPIPluggableAuthentication

**ITERATOR PATTERN**

Iterator pattern is used to access the elements of a collection. In our case, we have a general collection which contains users defined in OS. In order to find a user from the collection, users are iterated one by one in findUser() and getUid() methods.

**PARTICIPANTS**

**Iterator:**

AbstractIterator

**Concrete Iterator:**

Iterator

**Aggregate:**

AbstractCollection

**Concrete Aggregate:**

Collection

**ADAPTER PATTERN**

Adapter pattern is used to connect incompatible interfaces. In our case, we have three API classes which are Local, Ldap, and Kerberos and also one interface. In order to reach these three classes over a single class, adapter pattern is implemented. Each class has an adapter and these adapters are created as singleton. Adapters extend classes and implements the interface. Methods in each classes orient the flow to the Authenticator Mechanism which templete method is used in. In this case, adapter pattern and template pattern are used intertwined. An access is provided to functions in different classes over a method (authenticate()) by courtesy of the adapter pattern.

**PARTICIPANTS**

**Target:**

PluggableAuthentication

**Adapter:**

LocalAPIPluggableAuthentication,

LocalAPIPluggableAuthentication,

LocalAPIPluggableAuthentication

**Client:**

OperatingSystem

**Adaptee:**

LocalAPI,

LdapAPI,

KerberosAPI

**TEMPLATE PATTERN**

Template pattern is used to define a structure and the sequence of the algorithm. In our case, since the template pattern and the adapter pattern are used intertwined, methods in extended classes by adapter are proceed step by step in template method. When a distinction between classes (Local, Ldap, and Kerberos) is encountered, that different method is defined as abstract, template pattern is extended in class, and the function can be written in different ways. Although we have three different authentication APIs in our system, the code flow of each API is the same in our Project. Therefore, the flow is planned as the following. First, the user is found in database and his/her password is matched. Then, to be implemented methods are defined by corcerned API. Since userfind and verify password are the same in all APIs, they are identified as final. The special method of APIs (isloggedin()) is defined as abstract.

**PARTICIPANTS**

**AbstractClass:**

AuthenticationMechanism

**ConcreteClass:**

LocalAuthenticator,

LdapAuthenticator,

KerberosAuthenticator

**SINGLETON PATTERN**

Singleton pattern is used to ensure that only single object is created in a class. In our case, all adapters are created as singleton. In each adapter, there is only one object. Since classes’ constructors are private, new() method cannot be used. Therefore, firstly an object which is assigned as “null” is created and then a getInstance() method is defined for each acces role code.

**PARTICIPANTS**

LocalAPIPluggableAuthentication,

LdapAPIPluggableAuthentication,

KerberosAPIPluggableAuthentication

**THE CLASS BASED FLOW OF THE CODE**

**Class User {**

In this class we keep users. It contains three properties which are username, password, and uid and also their setters and getters. In addition a constructor is implemented in User class. We overrided hash() and equals() methods in order to add a unique user to the database according to given username and Uid.

**}**

**Class UserDatabase {**

In this class we created a collection and wrote a method (addUserToDatabase()) to add new users to the collection. We print the all users in database by printDb(). By using getUid() method, we get current user’s Uid. The only difference between GetUid() and findUser() is, findUser() gives us the all information about the current user. Iterator pattern is used in UserDatabase class.

**}**

**Class Collection {**

Users, defined in the database of the OS, are gathered up in a collection which is created as an ArrayList in the class. This class implements an interface which is called as AbstractAggregate with its all defined methods in it.

**}**

**Class CollectionIterator {**

In this class the interface of the AbstractIterator is implemented. This class includes two properties which are \_collection and \_current. In order to find the user from the collection, we use \_collection property, and while searching a user, \_current property is used to understand our current location among the users. By using getCount() method, the number of users are checked. According to this number, isDone() method works to understand if there is a next element or not.

**}**

**Class OperatingSystem {**

In this class a single method (authenticate()) is assigned to the user to prevent the complexity of the system, and without the knowledge of the user, the system performs all processes step by step. Facade pattern is used in OperatingSystem class.

**}**

**Class LocalAPI {**

In this class we have authenticate method for local. (local\_authenticate()). We create an instance from AuthenticationMecanism which is our template pattern. According to AccessRoleCode, the concerned mechanism will be performed.

**}**

**Class LdapAPI {**

In this class we have authenticate method for ldap. (ldap\_authenticate()). We create an instance from AuthenticationMecanism which is our template pattern. According to AccessRoleCode, the concerned mechanism will be performed.

**}**

**Class KerberosAPI {**

In this class we have authenticate method for kerberos. (kerberos\_authenticate()). We create an instance from AuthenticationMecanism which is our template pattern. According to AccessRoleCode, the concerned mechanism will be performed.

**}**

We created a PluggableAuthenticate interface to connect LocalAPI, LdapAPI, and KerberosAPI classes.

**Class LocalAPIPluggableAuthentication {**

This class is an adapter and it is singleton. There is has-a relation (LocalAPIPluggableAuthentication has LocalAPI). It extends LocalAPI and implements PluggableAuthentication.

**}**

**Class LdapAPIPluggableAuthentication {**

This class is an adapter and it is singleton. There is has-a relation (LdapAPIPluggableAuthentication has LdapAPI). It extends LdapAPI and implements PluggableAuthentication.

**}**

**Class KerberosAPIPluggableAuthentication {**

This class is an adapter and it is singleton. There is has-a relation (KerberosAPIPluggableAuthentication has KerberosAPI). It extends KerberosAPI and implements KerberosAuthentication.

**}**

**Class AuthenticatorMechanism {**

This abstract class is used to find username from the database and match the user password and the entered password. User is informed according to the result of the matching. The main goal of this class is to divide all authenticate processes into the steps and perform these steps sequentially. The sequence of the steps are defined in auhenticatebyGivenAccess() method which is called by concerned adapter to start the flow of the authentication steps. Some of these steps are the same for all APIs such as verifyPassword() and findUser(), thats why these methods are defined as final. However, since isLoggedin() method is different for each API, it is defined as abstract.

**}**

**Class LocalAuthenticator {**

This class extends AuthanticatorMechanism class. Therefore, isLoggedin() method for local API must be implemented in this class since the method is abstract. This class ensures that the user is authenticated to the system on local API successfully and it returns 0 as success code which is rc value.

**}**

**Class LdapAuthenticator {**

This class extends AuthanticatorMechanism class. Therefore, isLoggedin() method for ldap API must be implemented in this class since the method is abstract. This class ensures that the user is authenticated to the system on ldap API successfully and it returns 0 as success code which is rc value.

**}**

**Class KerberosAuthenticator {**

This class extends AuthanticatorMechanism class. Therefore, isLoggedin() method for kerberos API must be implemented in this class since the method is abstract. This class ensures that the user is authenticated to the system on kerberos API successfully and it returns 0 as success code which is rc value.

**}**

**KEY METHODS**

**authenticate()**

It is the only method that user can reach. It gets username and password as parameter. We get users’ Uid by getUid() method and in order to be able to use startswith() method, we convert Uid to string. Now we can monitor the AccsessRoleCode according to the first three letters of the Uid. 100,200, and 300 conform local, ldap, and kerberos respectively. Considering the AccessRoleCode result, an adapter will be created in switch-case.

**getAdapterInstance()**

A method which returns a single object is created for each adapter (getLocalAdapterInstance, getLdapAdapterInstance, getKerberosAdapterInstance). Thanks to these methods, we create a single instance from adapter and over this instance, we perform the processes.

**authenticateByGivenAccess()**

Firstly, findUser() method is called to find the user from the database and found user is assigned to User object. Then, we call verifyPassword() method and we send the password which is entered by the user and the password in database of the same user in the method. By matching these passwords, the method returns as a boolean value (True or False). This returned value is assigned to isPasswordMatch variable. According to the value of this variable, the access is permitted or not.In short, this method is used to verify the password of the user.

**UML DIAGRAM**

metin, harita içeren bir resim

Açıklama otomatik olarak oluşturuldu