ONLINE MARKETPLACE

Assignment 3

Samira Khorshidi

TABLE OF CONTENTS

ASSIGNMENT
DISCUSSION

The purpose of this practice is designing Domain Model of online Marketplace using Java RMI and MVC design pattern and by leveraging useful and proper design patterns.

RBAC

role-based access control (RBAC) is an approach to restricting system access to authorized users**. This** method is a tool of regulating access to resources based on the **roles** of individual users within an enterprise. In this context, access is the ability of an individual user to perform a specific task, such as view, create, or modify a file.
In this particular experiment, we have two specific roles, Administrator and Customer.

JAVA PROXY

A dynamic proxy class (simply referred to as a **proxy**) is a class that implements a list of interfaces specified at runtime when the class is created. A proxy interface is such an interface that is implemented by a proxy class. A proxy instance is an instance of a proxy class. Each proxy instance has an associated invocation handler object, which implements the interface `InvocationHandler`. A method invocation on a proxy instance through one of its proxy interfaces will be dispatched to

the invoke method of the instance's invocation handler, passing the proxy instance, a `java.lang.reflect.Method` object identifying the method that was invoked, and an array of type Object containing the arguments. The invocation handler processes the encoded method invocation as appropriate and the result that it returns will be returned as the result of the method invocation on the proxy instance.

JAVA ANNOTATION

Annotations, a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate. Annotations have a number of uses, among them:

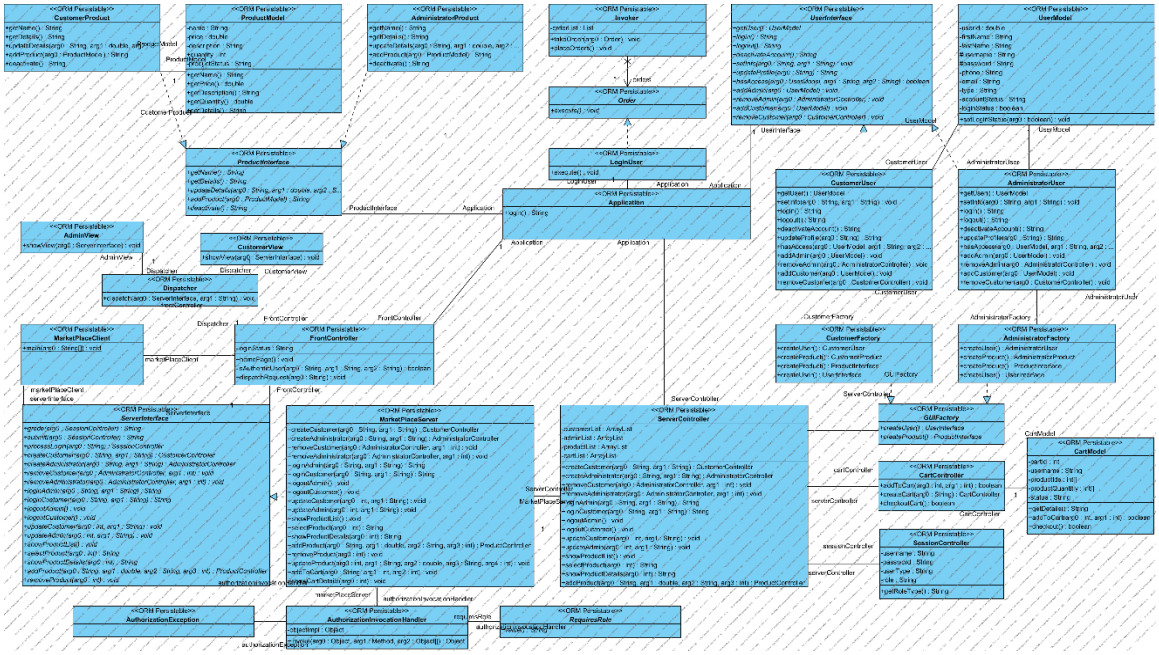
- *Information for the compiler — Annotations can be used by the compiler to detect errors or suppress warnings.*
- *Compile-time and deployment-time processing — Software tools can process annotation information to generate code, XML files, and so forth.*
- *Runtime processing — Some annotations are available to be examined at runtime. On the other hand, RMI let us to call remote methods, so we come up with an architecture that prohibit View component in MVC to have access to the Model directly and keep our data safe.*

Annotations are meta-meta-objects which can be used to describe other meta-objects. Meta-objects are classes, fields and methods. Asking an object for its meta-object (e.g. anObj.getClass()) is called introspection. The introspection can go further and we can ask a meta-object what are its annotations (e.g. aClass.getAnnotations). Introspection and annotations belong to what is called reflection and meta-programming.

FIGURE,
SAMPLE CODES

In following we provide you a Class Diagram of our system along with some sample codes.

UML DIAGRAM



The original UML diagram has been provided in Documentation directory. As you see, in our proposed architecture, we introduced a

new class named SessionController and also a group of several classes including AuthorizationInvocationHandler and exception handler to implementing the RBAC subsystem of our online market place.

SAMPLE CODES

In following we provide a sample code of the usage of Java Proxy, Java Annotation and custom exception handler in our system in order to address role based access control: Java proxy and Java Reflection:

```
import java.io.Serializable;

import java.lang.reflect.InvocationHandler;

import java.lang.reflect.Method;

public class AuthorizationInvocationHandler implements InvocationHandler,
Serializable {

       private Object objectImpl;

       public AuthorizationInvocationHandler(Object impl) {

          this.objectImpl = impl;

       }

       @Override

       public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable {
```

- *Java Annotation:*

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface RequiresRole {
	String value();
}
```

- *Exception handler:*

```
public class AuthorizationException extends RuntimeException {

	public AuthorizationException(String methodName) {

		super("Invalid Authorization - Access Denied to " + methodName +
"()		function!");

	}
```

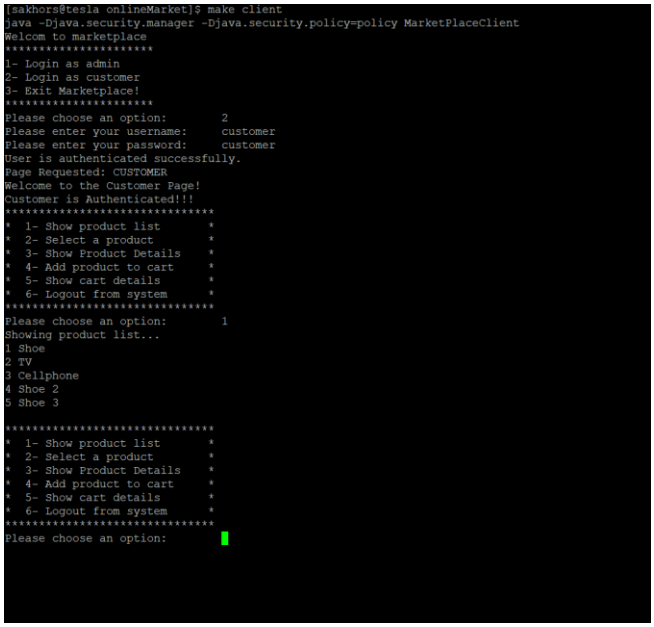
SAMPLE RUNS

In this section you'll see screenshots of sample runs. The system is minimal, so there is no GUI, instead every interaction happen using command prompt.

SCREENSHOTS

```
[sakhors@tesla onlineMarket]$ make client
java -Djava.security.manager -Djava.security.policy=policy MarketPlaceClient
Welcom to marketplace
**********************
1- Login as admin
2- Login as customer
3- Exit Marketplace!
**********************
Please choose an option:
```

```
[sakhors@tesla onlineMarket]$ make client
java -Djava.security.manager -Djava.security.policy=policy MarketPlaceClient
Welcom to marketplace
**********************
1- Login as admin
2- Login as customer
3- Exit Marketplace!
**********************
Please choose an option:        1
Please enter your username:     admin
Please enter your password:     admin
User is authenticated successfully.
Page Requested: ADMIN
Welcome to the Admin Page!
Administrator is Authenticated!!!
*******************************
*  1- Create a Customer        *
*  2- Update a Customer        *
*  3- Remove a Customer        *
*          ----                *
*  4- Create an Administrator  *
*  5- Update an Administrator  *
*  6- Remove an Administrator  *
*          ----                *
*  7- Add new product          *
*  8- Update a product         *
*  9- Remove a product         *
*  10- Logout from system      *
*******************************
Please choose an option:
```



DISCUSSION

During this assignment, different new patterns and contexts have been used to overcome with a role based access control subsystem as a part of our market place. However, there is a quite coupling between RMI and this subsystem, we are restricting users through our RMI methods, and also we have to keep in mind that proxy has some overhead. For most use cases the overhead won't be significant, though. The real problem is that the (over)use of dynamic proxies makes an application harder to understand and debug. For example, a dynamic proxy will show up with multiple lines in a stack trace.

CONCLUTION
REFERENCES

CONCLUTION

In conclusions, we found that role based access control can be implemented using Java proxy, Java annotation and java reflection within java RMI as a part of infrastructure. This practice can be in some way difficult to understand in the term of keeping low coupling as well as high cohesion properties of the system.

REFERENCES

- *https://patricklam.ca/papers/10.sacmat.rbac.pdf*
- *https://docs.oracle.com/javase/7/docs/api/java/lang/reflect/Proxy.html*
- *https://docs.oracle.com/javase/tutorial/java/annotations/*
- *Lecture slides*