Challenge 4: Proxy Pattern for Role-Based Access & Action Logging in a CMS

**Scenario** A Content Management System (CMS) must protect each Content object so that:

* **Admins** can read / edit / delete
* **Editors** can read / edit
* **Viewers** can read only

Every call (allowed or denied) must be written to a log.  
 You cannot touch the existing RealContent implementation because it already powers production sites.  
 Apply the **Proxy pattern** to enforce permissions and record activity without changing RealContent.

#### **Tasks**

1 — analysis/proxy\_cms\_need.md  
 • Explain how a direct call from UI to RealContent would lead to duplicated security checks in many places.  
 • Show how a single ContentProxy centralises both **access control** and **logging**.

2 — Domain model (src/main/java/cms/)  
 \* Role – ADMIN, EDITOR, VIEWER  
 \* User – holds String name & Role role  
 \* Content (**Subject** interface) with:  
 String read(User)  
 void edit(User,String newBody)  
 void delete(User)

\* RealContent (**RealSubject**) stores title & body; prints operation success.

3 — ContentProxy  
 \* Implements Content and keeps a reference to RealContent.  
 \* In each method:  
 1. log request (System.out.printf(...))  
 2. verify role; if denied, log refusal and return / throw  
 3. lazily instantiate RealContent the first time it’s really needed (lazy-loading)  
 4. delegate to RealContent

4 — Demo (ProxyDemo.java)  
 \* Create three users (admin, editor, viewer).  
 \* Show allowed and denied operations, demonstrating lazy construction and logging.

5 — Unit tests (src/test/java/cms/)  
 \* RolePermissionTest – ensure viewers cannot edit; editors cannot delete.  
 \* LazyInitTest – static counter in RealContent proves only one instance after first permitted call.  
 \* LoggingTest – capture System.out, verify “LOG:” lines appear for each call.

6 — Reflection (reflection.md)  
 \* How Proxy keeps **Single Responsibility** (log & guard outside real content).  
 \* Performance cost of indirection; mitigation via lightweight checks.  
 \* Road-map for adding a **CachingProxy** on top of ContentProxy.

#### **Deliverables**

analysis/proxy\_cms\_need.md

src/main/java/cms/Role.java

src/main/java/cms/User.java

src/main/java/cms/Content.java

src/main/java/cms/RealContent.java

src/main/java/cms/ContentProxy.java

src/main/java/cms/ProxyDemo.java

src/test/java/cms/RolePermissionTest.java

src/test/java/cms/LazyInitTest.java

src/test/java/cms/LoggingTest.java

reflection.md

README.md

## **Detailed Solution**

### **1 Core contracts**

/\* Role.java \*/

package cms;

public enum Role { ADMIN, EDITOR, VIEWER }

/\* User.java \*/

package cms;

public record User(String name, Role role) {}

/\* Content.java \*/

package cms;

public interface Content {

String read(User user);

void edit(User user, String newBody);

void delete(User user);

}

### **2 RealSubject**

/\* RealContent.java \*/

package cms;

public class RealContent implements Content {

private static int instances = 0; // for lazy-init test

private final String title;

private String body;

public RealContent(String title, String body){

this.title = title; this.body = body; instances++;

}

public static int instanceCount(){ return instances; }

@Override public String read(User u){

System.out.printf("%s reads %s%n", u.name(), title);

return body;

}

@Override public void edit(User u,String b){

this.body = b;

System.out.printf("%s edits %s%n", u.name(), title);

}

@Override public void delete(User u){

System.out.printf("%s deletes %s%n", u.name(), title);

}

}

### **3 Proxy**

/\* ContentProxy.java \*/

package cms;

public class ContentProxy implements Content {

private final String title;

private String initialBody;

private RealContent real; // lazy

public ContentProxy(String title,String initialBody){

this.title = title; this.initialBody = initialBody;

}

private RealContent real(){

if(real==null){

System.out.println("LOG: lazy-creating RealContent …");

real = new RealContent(title, initialBody);

}

return real;

}

private boolean can(User u, String op){

return switch(op){

case "read" -> true;

case "edit" -> u.role()==Role.ADMIN || u.role()==Role.EDITOR;

case "delete" -> u.role()==Role.ADMIN;

default -> false;

};

}

@Override public String read(User u){

System.out.printf("LOG: %s tries READ on %s%n",u.name(),title);

if(!can(u,"read")) return "Access denied";

return real().read(u);

}

@Override public void edit(User u,String b){

System.out.printf("LOG: %s tries EDIT on %s%n",u.name(),title);

if(!can(u,"edit")){ System.out.println("Access denied"); return; }

real().edit(u,b);

}

@Override public void delete(User u){

System.out.printf("LOG: %s tries DELETE on %s%n",u.name(),title);

if(!can(u,"delete")){ System.out.println("Access denied"); return; }

real().delete(u);

}

}

### **4 Demo**

/\* ProxyDemo.java \*/

package cms;

public class ProxyDemo {

public static void main(String[] args){

User admin = new User("Alice",Role.ADMIN);

User editor = new User("Eve",Role.EDITOR);

User viewer = new User("Bob",Role.VIEWER);

Content article = new ContentProxy("Design Patterns","Bridge vs Proxy …");

article.read(viewer); // allowed

article.edit(viewer,"oops"); // denied

article.edit(editor,"Updated body");// allowed

article.delete(editor); // denied

article.delete(admin); // allowed

}

}

**Sample console**

LOG: Bob tries READ on Design Patterns

LOG: lazy-creating RealContent …

Bob reads Design Patterns

LOG: Bob tries EDIT on Design Patterns

Access denied

LOG: Eve tries EDIT on Design Patterns

Eve edits Design Patterns

LOG: Eve tries DELETE on Design Patterns

Access denied

LOG: Alice tries DELETE on Design Patterns

Alice deletes Design Patterns

### **5 Tests (outline)**

/\* LazyInitTest \*/

RealContent.instances = 0;

Content p = new ContentProxy("x","y");

assertEquals(0, RealContent.instanceCount());

p.read(new User("u",Role.ADMIN));

assertEquals(1, RealContent.instanceCount());

/\* RolePermissionTest \*/

Content p = new ContentProxy("x","y");

String result = p.read(new User("v",Role.VIEWER));

assertFalse(result.contains("Access denied"));

p.edit(new User("v",Role.VIEWER),"bad"); // should print Access denied

/\* LoggingTest \*/

String out = capture(() -> {

Content p2 = new ContentProxy("x","y");

p2.read(new User("a",Role.ADMIN));

});

assertTrue(out.contains("LOG:"));

*(capture redirects System.out.)*

### **Reflection (key points)**

| **Concern** | **How ContentProxy solves it** |
| --- | --- |
| Access control | role-based checks before delegation |
| Logging/monitoring | unified “LOG:” lines in proxy |
| Lazy resource creation | instantiates RealContent on first access |

The **Proxy pattern** cleanly enforces security and observability without polluting the core RealContent logic, and it keeps the system open to future concerns—like caching or rate-limiting—via additional proxy layers.