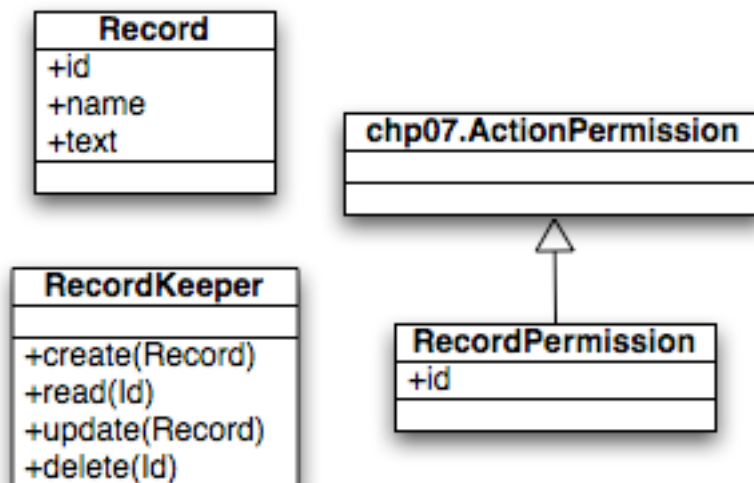


## 8 JAAS for Data Access Control

This chapter is an overview of using JAAS to protect access to specific instances or pieces of data. For example, in a system where you have employee records, you might want to restrict access to those files based on who is trying to view or modify those records. This is called “data access control.” JAAS is very good at, and commonly used to providing permissions for broad, class-level actions in systems, for example, controlling which users may change the system-wide `java.security.Policy` in effect. However, it’s less common to see JAAS used to protect access to specific instances of classes. This chapter goes over using JAAS in this respect, using a custom `java.security.Permission` class to protect access to class instances.

### 8.2 The Record Domain



To demonstrate data access control, we’ll use a simple domain of generic records. Each `Record` has a unique ID, a name, and text content. The records are immutable, and are maintained by the `RecordKeeper`. The `RecordKeeper` is a simplistic service that manages the persistence and data access control for records. Before creating, reading, updating, or deleting any `Record`, the `RecordKeeperService` does a security check to verify that the current security context has been granted the appropriate `RecordPermission`, a custom `java.security.Permission` class used to protect `Records`.

The `RecordPermission` extends the `ActionsPermission` from chapter 7, providing an implementation of `implies()` that uses `RecordPermission`’s actions. `RecordPermission`’s actions are used to specify which CRUD operation (create, read, update, or delete) the permission is granting. For example, to create an instance of `RecordPermission` that granted the right to read and update a `Record`, you would use the following code:

```
RecordPermission perm = new RecordPermission(id, "read, update");
```



This work is licensed under a Creative Commons Attribution-NonCommercial 2.5

License: <http://creativecommons.org/licenses/by-nc/2.5/>

### 8.1.2 Record

The record class is a simple, immutable data object:

```
package chp08;

import util.id.Id;

public class Record
{
    public Id id;
    public String name;
    public String text;

    public Record(Id id, String name, String text)
    {
        this.id = id;
        this.name = name;
        this.text = text;
    }

    public Id getId()
    {
        return id;
    }

    public String getName()
    {
        return name;
    }

    public String getText()
    {
        return text;
    }
}
```

### 8.1.3 RecordKeeper

RecordKeeper is a simplistic service that manages the life of Record instances. To keep this example simple, RecordKeeper stores Records in memory instead of in a more durable persistence store, like a database. RecordKeeper stores Record instances in a `java.util.HashMap`, where the key is the Record's ID and the value is the Record instance itself. Before performing any of the four data modification actions, RecordKeeper creates a `RecordPermission` to verify that the current security context has been granted the appropriate `RecordPermission`.

The code for RecordKeeper is below:



This work is licensed under a Creative Commons Attribution-NonCommercial 2.5  
License: <http://creativecommons.org/licenses/by-nc/2.5/>

```
package chp08;

import java.security.AccessController;
import java.util.HashMap;
import java.util.Map;

import util.id.Id;

public final class RecordKeeper {

    private Map records = new HashMap();

    public void create(Record record) {
        RecordPermission perm = new RecordPermission(record.getId(),
            "create");
        AccessController.checkPermission(perm);
        records.put(record.getId(), record);
    }

    public Record read(Id recordId) {
        RecordPermission perm = new RecordPermission(recordId, "read");
        AccessController.checkPermission(perm);
        return (Record) records.get(recordId);
    }

    public void update(Record record) {
        RecordPermission perm = new RecordPermission(record.getId(),
            "update");
        AccessController.checkPermission(perm);
        records.put(record.getId(), record);
    }

    public void delete(Id recordId) {
        RecordPermission perm = new RecordPermission(recordId, "delete");
        AccessController.checkPermission(perm);
        records.remove(recordId);
    }
}
```

### 8.1.4 RecordPermission

The custom permission used to protected Records uses the permission's name to specify the Record's unique Id, and the actions attribute to specify if permission is granted to create, read, update, or delete the Record in question. Because RecordPermission extends ActionsPermission, all we need to implement in the code is a constructor that takes the Id of the Record being protected and the actions granted for that Record:

```
package chp08;

import util.id.Id;
```



This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License: <http://creativecommons.org/licenses/by-nc/2.5/>

```
import chp07.ActionsPermission;

public class RecordPermission
    extends ActionsPermission {

    public RecordPermission(String recordId, String actions) {
        super(recordId, actions);
    }

    public RecordPermission(Id recordId, String actions) {
        this(recordId.getId(), actions);
    }
}
```

## 8.2 JAAS Code

We'll use the authentication code developed in chapter 4 along with the database-backed authorization code developed in chapter 6. Recall that the authorization code stores Permission grants in a table with columns for the Permission type, a unique ID for the Permission, the Permission's name, and the Permission's actions.

## 8.3 The Example Application

Once again, we'll use a script-class with a main method to demonstrate this chapter's code in action. The "application" first configures JAAS to use our custom authentication and authorization code, then attempts to use the RecordKeeper without any RecordPermission being granted, grants the needed RecordPermission, and then uses RecordKeeper.

The code is below:

```
package chp08;

import java.security.Policy;
import java.security.PrivilegedAction;

import javax.security.auth.Subject;

import util.id.Id;
import chp04.UserGroupPrincipal;
import chp06.AuthHelper;
import chp06.DbPolicy;
import chp06.PermissionService;

public class Main {
```



This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License: <http://creativecommons.org/licenses/by-nc/2.5/>

```
public static void main(String[] args) throws Exception {
    AuthHelper authHelper = new AuthHelper();

    try {
        authHelper.createTestUser("testuser", "password");
        UserGroupPrincipal p = authHelper.getUserGrp();
        authHelper.loginTestUser();
        Subject subject = authHelper.getSubject();

        Policy.setPolicy(new DbPolicy());
        boolean granted = true;
        try {
            Subject.doAsPrivileged(subject, new PrivilegedAction() {

                public Object run() {
                    Record r = new Record(Id.create("id1"), "record1",
                        "this is some content.");
                    RecordKeeper rk = new RecordKeeper();
                    rk.create(r);
                    return null;
                }
            }, null);
        } catch (SecurityException e) {
            granted = false;
        }

        System.out.println("Permission granted? " + granted);

        RecordPermission perm = new RecordPermission(
            Id.create("id1"), "create,read");
        PermissionService.addPermission(p.getId(), Id
            .create("permId1"), perm);
        System.out.println("Added grant for RecordPermission.");
        granted = true;
        try {
            Subject.doAsPrivileged(subject, new PrivilegedAction() {

                public Object run() {
                    Record r = new Record(Id.create("id1"), "record1",
                        "this is some content.");
                    RecordKeeper rk = new RecordKeeper();
                    rk.create(r);
                    return null;
                }
            }, null);
        } catch (SecurityException e) {
            granted = false;
        }

        System.out.println("Permission granted? " + granted);
    }
}
```



```
        }, null);
    } catch (SecurityException e) {
        granted = false;
    }
    System.out.println("Permission granted? " + granted);
} finally {
    authHelper.cleanup();
    PermissionService.removePermission(Id.create("permId1"));
}
}
```

To run the above, change to the root directory of this book's project, and type `ant run-chp08`. The output of will include the following:

```
Permission granted? false
Added grant for RecordPermission.
Permission granted? true
```

## Summary

This chapter provided a concise example of using JAAS for data access control. When you're restricting access to specific instance of object or data, not just general, system-wide actions, you're doing data access control. For our example, we created a small data object that represented a `Record`, and a service layer, `RecordKeeper`, that performed persistence and lookup of and for that data object. Also, we created a custom `Permission`, `RecordPermission`, that was used by `RecordKeeper` to restrict the actions of creating, reading, updating, and deleting specific `Records`. As the demonstration at the end of this chapter showed, this powerful, yet simple model allowed us to easily control access to each individual `Record`.

