CS 4743
Applied Software Engineering
Spring 2015

Assignment 5: Stateful Session Management and Product Creation
Assigned: 3/16/2015
Due: 3/30/2015 11:59pm

Background

Now that the *Cabinetron* facility management system is starting to have some functional range, they want to begin testing it with users from various departments, including inventory personnel, production personnel, and administrators. As you would expect, users from different departments are limited in what they can access in the software and *Cabinetron* is ready to see access control as part of that testing.

With Product Templates and their respective Part quantities in place, the client also wants to implement Product creation. Product creation is a fairly meaty piece of business logic that involves checking if sufficient quantities of Parts are available in Inventory according to the Product's Template, and if so, deducting appropriate quantities from Inventory and adding a new instance of that Product to Inventory (including specifying the Location of the newly created Product). This also means that an Inventory Item can now be either a Part or a Product.

Specifications

Model: User
Represents a user of the software. In this assignment, the User will be very simple:
    Full Name: String
    Email Address: String (will serve as the login)

Model: Session
Represents the currently authenticated User (current User) of that software client and the user's access privileges. When the software needs to determine if the current User can access a particular function, it will ask the Session object. User members are:
    User: the current user associated with that Session instance
    Appropriate member variables to provide the below behavior (e.g., a bunch of booleans)

Session Behavior
    canViewProductTemplates: can access the Product Templates list view or the detail views
    canAddProductTemplates: can add a new Product Template or edit an existing one

canDeleteProductTemplates: can delete a Product Template or any of its Parts
canCreateProducts: can create a Product from a Product Templates (new function)
canViewInventory: can access the Inventory list view or the Inventory Item detail view
canAddInventory: can add new Inventory Items to Inventory or edit existing ones
canViewParts: can access the Parts list view or its detail view
canAddParts: can add a new Part or edit an existing one
canDeleteParts: can delete a Part
canDeleteInventory: can delete an InventoryItem

Cabinetron **Role-based Access Control Policy v1**
The 3 roles in this assignment are Inventory Manager, Production Manager, and Admin. Below is a table of the software functions and the roles that are allowed to access the respective function. A role not in the set of allowed roles is assumed to be denied access to that function.

Function	**Allowed Roles**
View Product Templates	Prod. Mgr., Admin
Add/Edit Product Templates	Prod. Mgr., Admin
Delete Product Templates	Prod. Mgr., Admin
Create Products	Prod. Mgr., Admin
View Inventory	All
Add/Edit Inventory Items	Inv. Mgr., Admin
View Parts	All
Add/Edit Parts	Inv. Mgr., Admin
Delete Parts	Admin
Delete Inventory	Admin

Cabinetron **Users and Roles**
Tom Jones: Production Manager
Email: you decide
Password: you decide

Sue Smith: Inventory Manager
Email: you decide
Password: you decide

Ragnar Nelson: Admin
Email: you decide
Password: you decide

Model: Authenticator
This class is responsible for:
1. Receiving a set of login credentials (i.e., login and password) and
2. Returning either a Session instance for the User/Role matching the credentials,

or an error value or exception if the credentials are invalid.
We are not going to be hard-nosed security conscious about this less-than-awesome authentication technique. We belabored that topic in CS 3773 and will return to it again in CS 4953.

Authenticator really only needs a single static method that receives credentials as parameters and returns a Session object based on those credentials (if correct) or throws an Exception. You should of course test that all users can authenticate and no other credentials can.

Model Change: Inventory Item
Items in Inventory can now be one of 2 types of stock: Parts and Products. Products are a physical instantiation of a Product Template. The Inventory Item model and database table(s) must be extended to accommodate either a Part or a Product at a particular location and in specified quantity. One simple way to accomplish this is to add another foreign key, product_template_id, which if non-zero means that the particular Inventory Item is a Product based off of the associated Product Template.

Constraint: this may sound obvious, but a single instance of an Inventory Item record/object can only ever be associated with a Part OR a Product Template. Not both.

Example:
Product Template:
    Id: 1
    Description: Chair

Product Template Parts:
    Part Id 5: Chair back, Qty: 1
    Part Id 6: Chair seat, Qty: 1
    Part Id 7: Chair legs, Qty: 4

My Inventory Items in my Inventory view look like this:
    Inventory Item Id: 10, Part Id: 5, Part Name: Chair back, Qty: 10, Location: Facility 2
    Inventory Item Id: 14, Part Id: 6, Part Name: Chair seat, Qty: 22, Location: Facility 1 WH 1
    Inventory Item Id: 17, Part Id: 7, Part Name: Chair leg, Qty: 90, Location: Facility 1 WH 1

After executing the Create Product behavior below to create a new Chair in Facility 2, my Inventory Items in my Inventory view might look like this:
    Inventory Item Id: 10, Part Id: 5, Part Name: Chair back, Qty: 9, Location: Facility 2
    Inventory Item Id: 14, Part Id: 6, Part Name: Chair seat, Qty: 21, Location: Facility 1 WH 1
    Inventory Item Id: 17, Part Id: 7, Part Name: Chair leg, Qty: 86, Location: Facility

1 WH 1
	Inventory Item Id: 18, Product Template Id: 1, Product Name: Chair, Qty: 1, Location: Facility 2

Behavior: Create Product
This function/method takes a given Product Template and, if there are sufficient quantities in Inventory of the Product Template Parts, creates a new Inventory Item based on that Product Template id at the provided Location (or increments the quantity if the Product already exists at that Location). If the quantities of Inventory Item Parts are insufficient then some type of error should be returned/thrown. You can make a new class to provide the Create Product behavior OR extend an existing class, like Inventory.

Create Product **MUST** use a Gateway to perform the database work necessary for changing Inventory and the database statements **MUST** be within a single transaction that is only committed if the product is successfully created (i.e., Inventory has sufficient quantities of parts). The Create Product transaction should rollback in all other cases.

Business Rule for Inventory Allocation during Product Creation
You will implement a simple, naive business rule for allocating inventory during product creation (which also means that this rule is subject to change):

1. a product template part quantity may only come from a single location (i.e., you cannot split the a inventory deduction over multiple locations)
2. the sequence of locations in which to check for quantity is:
   2.a. Facility 1 Warehouse 1
   2.b Facility 1 Warehouse 2
   2.c Facility 2

Super Awesome Optional Step
If you really want to be awesome, you can optionally lock your Inventory table to make sure no other client process changes part quantities once Create product is underway (i.e., a pessimistic lock on the entire Inventory table). You can temporarily add a Thread.sleep() in the Create Product behavior to make sure the table lock is working. Note that this step is not required in this assignment but is a good test of your grasp of database transactions and is great experience.

User Interfaces:
This assignment doesn't really require any new views. However, you will need to modify one or more existing views to accommodate Products as Inventory Items, including when a Product is created in order to specify a Location for where the Product will be stored.

I would also like some type of visual feedback regarding the currently logged in user, such as the user's name in the MDI Parent frame title bar.

Datasource
Nothing new required. User and access policy data can be hard-coded at this point, but feel free to provide database support if you wish. Should you do so, you will need to implement a Gateway object(s) between the Model(s) and database.

Deliverables:
1. Export your entire project directory in Eclipse, including your local repo (the .git folder), to a zip file
2. Include a simple Quality Assurance test plan that is essentially a checklist based on the above access control policy indicating that each role has been tested for its respective allow/deny privilege for each function. I will post an example on Blackboard.
3. A README.txt file that explains your approach to this assignment, including your team's members. It should contain useful details for the grader that communicate your design choices in satisfying the assignment's requirements (e.g., 2 views embedded in 1 JFrame). Try to make it easy for the grader to understand how you fulfilled this assignment's requirements, especially if your choices are particularly clever or "special".

Note: both team member's should submit the same project separately

Rubric:

10 pts	User model implemented
10 pts	Session model implemented
10 pts	Authenticator implemented
20 pts	QA test plan for access control
40 pts	Create Product implemented (including appropriate component changes to support)
10 pts	Use of Git branches for development