Getting a handle on resources

**Goal:** In this lab, we will create domain classes to store resource information. By the end of this lab, you should know how to create a boilerplate grails application, how to add domain classes, and how to express relationships between them.  
  
**Instructions**  
  
The first thing we will do is creating a grails application. Open up your command prompt and cd into the directory where you would like to create the grails application for our blog. Then, run the following command

> grails create-app zynxblog

If you run the ls(Unix) or dir(Windows) command, you should see a directory layout like below:

* conf
* controllers
* domain
* i18n
* services
* taglib
* utils
* views

We will now create domain classes, which are the classes we will be using to represent our resources. For our purposes, we want to store the following information with each resource:

* Posts
  + title
  + teaser
  + content
  + lastUpdated (Date)
  + published (Boolean)
* Comments
  + post (Post)
  + comment
  + commentator (Comentator)
  + dateCreated (Date)
* Commentator
  + name
  + url
  + email
  + comment (Comment)
* Tag
  + tag

All fields without types should be Strings.  
  
**Creating our first resource**  
  
First, we will create our Post domain class. cd into your project directory and launch the grails console using the grails command. You should be greeted by a prompt that reads

grails>

Type in the command “create-domain-class zynxblog.Post.” You should see output like the following:

grails> create-domain-class zynxblog.Post

| Created file grails-app/domain/zynxblog/Post.groovy

| Created file test/unit/zynxblog/PostTests.groovy

The first file contains the class where you will declare the resource properties. The second file is for tests.  
  
Open up the grails-app/domain/zynxblog/Post.groovy class and add in the fields for the resources we defined above (Hint: you can look at our example project if you get stuck).  
  
**Adding validation**  
  
We can’t just accept any data that users might hand to us. For example, the text string “asdf” should not be valid as a value of the Commentator.email field. We need a way of expressing this concept to grails. In each of your domain classes, you should see a variable declared as follows:

static constraints = {

}

This is where you declare restrictions on fields in grails. So, for example, if, in the Commentator domain class, you wanted to restrict the email field to only accept valid e-mail addresses, you would add the following line into the constraints section:

email(nullable: false, blank: false, email: true)

This tells grails that the email field cannot be null, cannot be left blank, and must be an e-mail. Grails has a number of built-in constraints, and each one of them has the syntax constraint-type:  constraint-parameter. A full list of constraints can be found at  <http://www.grails.org/doc/latest/ref/Constraints/Usage.html>.  
  
Using this guide, add the following constraints

* Post
  + title (not null, not blank, between 1 and 50 characters)
  + teaser (between 0 and 100 characters)
  + content (not null)
  + lastUpdated (not null, not blank)
  + published (not null)
* Comment
  + post (not null, not blank)
  + comment (not null, not blank)
  + dateCreated(not null, not blank)
* Commentator
  + name (not null, not blank)
  + url (null ok, blank ok, is valid url)
  + email (null ok, blank ok, is valid e-mail address)
  + comment (not null)
* Tag
  + tag (not null, between 0 and 50 characters)

**Relationships between the classes**  
  
So far, we have classes to represent the object types we will be working with, but we haven’t told grails how the objects relate. For example, given a Post, it would be nice to retrieve all comments that belong to that Post. Luckily, grails has an automated mechanism to do this for us!  
  
When we created the Comment class, we declared that all Comment objects have a Post with which they are associated. To solidify the relationship, and let grails work its magic, we must also add the following line to the Comment class:

static belongsTo = Post

This will let us access the Post associated with a Comment, given the Comment object. However, in order to establish the inverse relationship (getting a collection of Comments from a Post), we must modify the Post class. Add the following line to the Post class:

static hasMany = [comments: Comment]

This tells grails that all Comments belonging to this post should be accessible under the variable comments. Notice that the value of the hasMany variable is simply a groovy dictionary.  
  
Finally, we want the comments to be sorted by dateCreated. In order to do this, we will must first tell grails that we want the collection to be sorted. Add the following declaration to Post.groovy:

SortedSet comments

Now, we must tell grails how to sort a Comment. Edit the Comment.groovy class and modify the class declaration so that Comment implements the Comparable interface. This means that

class Comment {

should become

class Comment implements Comparable {

Now add the following method into the Comment class:

int compareTo(obj)

{

   return dateCreated.compareTo(obj.dateCreated)

}

Using your experience with comments, do the same thing for Tag’s, except make them sorted by tag name.